




# ESP32

## ESP-IDF 编程指南



Release v4.0.3  
乐鑫信息科技  
2021年06月24日



# Table of contents

<b>Table of contents</b>	<b>i</b>
<b>1 快速入门</b>	<b>3</b>
1.1 概述	3
1.2 准备工作	3
1.3 开发板简介	4
1.3.1 ESP32-DevKitC V4 入门指南	4
1.3.2 ESP-WROVER-KIT V4.1 入门指南	9
1.3.3 ESP32-PICO-KIT V4/V4.1 入门指南	34
1.3.4 ESP32-Ethernet-Kit V1.0 入门指南	44
1.4 详细安装步骤	51
1.4.1 设置开发环境	51
1.4.2 创建您的第一个工程	52
1.5 第一步：安装准备	52
1.5.1 Windows 平台工具链的标准设置	52
1.5.2 Linux 平台工具链的标准设置	53
1.5.3 MacOS 平台工具链的标准设置	54
1.6 第二步：获取 ESP-IDF	56
1.6.1 Linux 和 macOS 操作系统	56
1.6.2 Windows 操作系统	56
1.7 第三步：设置工具	56
1.7.1 Windows 操作系统	56
1.7.2 Linux 和 macOS 操作系统	57
1.7.3 自定义工具安装路径	57
1.8 第四步：设置环境变量	57
1.8.1 Windows 操作系统	57
1.8.2 Linux 和 macOS 操作系统	57
1.9 第五步：开始创建工程	58
1.9.1 Linux 和 macOS 操作系统	58
1.9.2 Windows 操作系统	58
1.10 第六步：连接设备	58
1.11 第七步：配置	58
1.11.1 Linux 和 macOS 操作系统	58
1.11.2 Windows 操作系统	59
1.12 第八步：编译工程	59
1.13 第九步：烧录到设备	60
1.14 第十步：监视器	61
1.15 更新 ESP-IDF	62
1.16 相关文档	62
1.16.1 与 ESP32 建立串口连接	62
1.16.2 Eclipse IDE 创建和烧录指南	68
1.16.3 IDF 监视器	68
1.16.4 工具链的自定义设置	72
1.16.5 快速入门 (传统 GNU Make)	77
<b>2 API 参考</b>	<b>101</b>
2.1 蓝牙 API	101

2.1.1	Controller && VHCI	101
2.1.2	BT COMMON	108
2.1.3	BT LE	113
2.1.4	CLASSIC BT	188
2.1.5	NimBLE-based host APIs	248
2.1.6	ESP-BLE-MESH	251
2.2	连网 API	462
2.2.1	Wi-Fi	462
2.2.2	以太网	532
2.2.3	IP 网络层协议	544
2.2.4	应用层协议	556
2.3	外设 API	556
2.3.1	Analog to Digital Converter	556
2.3.2	Controller Area Network (CAN)	571
2.3.3	Digital To Analog Converter	585
2.3.4	GPIO & RTC GPIO	587
2.3.5	I2C	598
2.3.6	I2S	610
2.3.7	LED Control	619
2.3.8	MCPWM	631
2.3.9	Pulse Counter	649
2.3.10	RMT	657
2.3.11	SDMMC Host Driver	672
2.3.12	SD SPI Host Driver	676
2.3.13	SDIO Card Slave Driver	679
2.3.14	Sigma-delta Modulation	691
2.3.15	SPI Master driver	693
2.3.16	SPI Slave driver	710
2.3.17	Timer	715
2.3.18	Touch Sensor	722
2.3.19	UART	735
2.4	应用层协议	754
2.4.1	mDNS Service	754
2.4.2	ESP-TLS	763
2.4.3	ESP HTTP Client	770
2.4.4	ESP WebSocket Client	782
2.4.5	HTTP 服务器	787
2.4.6	HTTPS server	807
2.4.7	ASIO port	809
2.4.8	ESP-MQTT	809
2.4.9	ESP-Modbus	818
2.4.10	ESP Local Control	823
2.4.11	Creating a property	824
2.4.12	Client Side Implementation	826
2.4.13	IP 网络层协议	831
2.5	配网 API	831
2.5.1	Unified Provisioning	831
2.5.2	Protocol Communication	834
2.5.3	Wi-Fi Provisioning	845
2.6	Storage API	860
2.6.1	SPI Flash API	860
2.6.2	SD/SDIO/MMC Driver	879
2.6.3	Non-volatile storage library	888
2.6.4	NVS Partition Generator Utility	905
2.6.5	Virtual filesystem component	912
2.6.6	FAT Filesystem Support	921
2.6.7	Wear Levelling API	925
2.6.8	SPIFFS Filesystem	929

2.6.9	Manufacturing Utility	932
2.7	System API	937
2.7.1	FreeRTOS	937
2.7.2	FreeRTOS Additions	1016
2.7.3	Heap Memory Allocation	1033
2.7.4	Heap Memory Debugging	1043
2.7.5	The himem allocation API	1052
2.7.6	Interrupt allocation	1055
2.7.7	Watchdogs	1060
2.7.8	eFuse Manager	1063
2.7.9	Inter-Processor Call	1073
2.7.10	High Resolution Timer	1074
2.7.11	Logging library	1078
2.7.12	Event Loop Library	1082
2.7.13	Application Level Tracing	1095
2.7.14	电源管理	1099
2.7.15	Sleep Modes	1104
2.7.16	Over The Air Updates (OTA)	1112
2.7.17	ESP HTTPS OTA	1122
2.7.18	ESP-pthread	1124
2.7.19	Error Codes and Helper Functions	1126
2.7.20	App Image Format	1128
2.7.21	Miscellaneous System APIs	1133
2.8	Project Configuration	1140
2.8.1	Introduction	1140
2.8.2	Project Configuration Menu	1140
2.8.3	Using sdkconfig.defaults	1140
2.8.4	Kconfig Formatting Rules	1140
2.8.5	Backward Compatibility of Kconfig Options	1141
2.8.6	Configuration Options Reference	1141
2.8.7	Customisations	1295
2.9	Error Codes Reference	1295
<b>3</b>	<b>ESP32 H/W 硬件参考</b>	<b>1301</b>
3.1	ESP32 Modules and Boards	1301
3.1.1	WROOM, SOLO, WROVER, and PICO Modules	1301
3.1.2	Development Boards	1304
3.1.3	Related Documents	1306
3.2	Previous Versions of ESP32 Modules and Boards	1306
3.2.1	ESP32-PICO-KIT V4	1308
3.2.2	ESP32-PICO-KIT V3	1308
3.2.3	ESP32 Core Board V2 / ESP32 DevKitC	1309
3.2.4	ESP-WROVER-KIT V3	1309
3.2.5	ESP-WROVER-KIT V2	1309
3.2.6	ESP-WROVER-KIT V1 / ESP32 DevKitJ V1	1312
3.2.7	ESP32 Demo Board V2	1313
3.2.8	Related Documents	1313
<b>4</b>	<b>API 指南</b>	<b>1315</b>
4.1	ESP-IDF 编程注意事项	1315
4.1.1	应用程序的启动流程	1315
4.1.2	应用程序的内存布局	1316
4.1.3	DMA 能力要求	1318
4.2	构建系统 (CMake 版)	1319
4.2.1	概述	1319
4.2.2	使用构建系统	1320
4.2.3	示例项目	1322
4.2.4	项目 CMakeLists 文件	1323

4.2.5	组件 CMakeLists 文件	1324
4.2.6	组件配置	1326
4.2.7	预处理器定义	1326
4.2.8	组件依赖	1326
4.2.9	构建的内部过程	1329
4.2.10	组件 CMakeLists 示例	1331
4.2.11	自定义 sdkconfig 的默认值	1335
4.2.12	Flash 参数	1335
4.2.13	构建 Bootloader	1335
4.2.14	选择硬件目标	1335
4.2.15	编写纯 CMake 组件	1336
4.2.16	组件中使用第三方 CMake 项目	1336
4.2.17	组件中使用预建库	1336
4.2.18	在自定义 CMake 项目中使用 ESP-IDF	1337
4.2.19	ESP-IDF CMake 构建系统 API	1337
4.2.20	文件通配 & 增量构建	1341
4.2.21	构建系统的元数据	1341
4.2.22	从 ESP-IDF GNU Make 构建系统迁移到 CMake 构建系统	1342
4.3	构建系统 (传统 GNU Make)	1344
4.3.1	使用构建系统	1344
4.3.2	概述	1344
4.3.3	构建 Bootloader	1353
4.4	错误处理	1353
4.4.1	概述	1353
4.4.2	错误码	1354
4.4.3	错误码到错误消息	1354
4.4.4	ESP_ERROR_CHECK 宏	1354
4.4.5	错误处理模式	1355
4.4.6	C++ 异常	1355
4.5	严重错误	1356
4.5.1	概述	1356
4.5.2	紧急处理程序	1356
4.5.3	寄存器转储与回溯	1358
4.5.4	GDB Stub	1359
4.5.5	Guru Meditation 错误	1359
4.5.6	其它严重错误	1361
4.6	Event Handling	1361
4.6.1	Wi-Fi, Ethernet, and IP Events	1361
4.6.2	Mesh Events	1363
4.6.3	Bluetooth Events	1363
4.7	Deep Sleep Wake Stubs	1364
4.7.1	Rules for Wake Stubs	1364
4.7.2	Implementing A Stub	1364
4.7.3	Loading Code Into RTC Memory	1365
4.7.4	Loading Data Into RTC Memory	1365
4.8	ESP32 Core Dump	1366
4.8.1	Overview	1366
4.8.2	Configuration	1366
4.8.3	Save core dump to flash	1366
4.8.4	Print core dump to UART	1367
4.8.5	ROM Functions in Backtraces	1367
4.8.6	Running ‘espcoredump.py’	1367
4.9	Flash Encryption	1367
4.9.1	Introduction	1368
4.9.2	eFuse Used During Flash Encryption Process	1368
4.9.3	Flash Encryption Process	1369
4.9.4	Steps to Setup Flash Encryption	1370
4.9.5	Key Points About Flash Encryption	1376

4.9.6	Using Encrypted Flash	1376
4.9.7	Updating Encrypted Flash	1377
4.9.8	Disabling Flash Encryption	1377
4.9.9	Limitations of Flash Encryption	1378
4.9.10	Flash Encryption and Secure Boot	1378
4.9.11	Advanced Features	1378
4.9.12	Technical Details	1380
4.10	ESP-IDF FreeRTOS SMP Changes	1380
4.10.1	Overview	1380
4.10.2	Backported Features	1381
4.10.3	Tasks and Task Creation	1382
4.10.4	Scheduling	1383
4.10.5	Critical Sections & Disabling Interrupts	1385
4.10.6	Floating Point Arithmetic	1386
4.10.7	Task Deletion	1386
4.10.8	Thread Local Storage Pointers & Deletion Callbacks	1386
4.10.9	Configuring ESP-IDF FreeRTOS	1387
4.11	Thread Local Storage	1387
4.11.1	Overview	1387
4.11.2	FreeRTOS Native API	1387
4.11.3	Pthread API	1388
4.11.4	C11 Standard	1388
4.12	High-Level Interrupts	1388
4.12.1	Interrupt Levels	1388
4.12.2	Notes	1389
4.13	JTAG 调试	1389
4.13.1	引言	1389
4.13.2	工作原理	1390
4.13.3	选择 JTAG 适配器	1390
4.13.4	安装 OpenOCD	1391
4.13.5	配置 ESP32 目标板	1391
4.13.6	启动调试器	1396
4.13.7	调试范例	1396
4.13.8	从源码构建 OpenOCD	1397
4.13.9	注意事项和补充内容	1400
4.13.10	相关文档	1400
4.14	Bootloader	1436
4.14.1	Bootloader compatibility	1436
4.14.2	FACTORY reset	1437
4.14.3	Boot from TEST firmware	1438
4.14.4	Customer bootloader	1438
4.15	分区表	1438
4.15.1	概述	1438
4.15.2	内置分区表	1438
4.15.3	创建自定义分区表	1439
4.15.4	生成二进制分区表	1440
4.15.5	烧写分区表	1441
4.16	Secure Boot	1441
4.16.1	Background	1441
4.16.2	Secure Boot Process Overview	1442
4.16.3	Keys	1442
4.16.4	Bootloader Size	1443
4.16.5	How To Enable Secure Boot	1443
4.16.6	Re-Flashable Software Bootloader	1444
4.16.7	Generating Secure Boot Signing Key	1444
4.16.8	Remote Signing of Images	1445
4.16.9	Secure Boot Best Practices	1445
4.16.10	Technical Details	1445

4.16.11	Secure Boot & Flash Encryption	1447
4.16.12	Signed App Verification Without Hardware Secure Boot	1447
4.16.13	Advanced Features	1447
4.17	ULP 协处理器编程	1447
4.17.1	ULP coprocessor instruction set	1448
4.17.2	Programming ULP coprocessor using C macros	1460
4.17.3	安装工具链	1465
4.17.4	编译 ULP 代码	1465
4.17.5	访问 ULP 程序变量	1466
4.17.6	启动 ULP 程序	1467
4.17.7	ULP 程序流	1468
4.18	ULP Coprocessor Programming	1469
4.18.1	Installing the Toolchain	1469
4.18.2	Compiling the ULP Code	1469
4.18.3	Accessing the ULP Program Variables	1470
4.18.4	Starting the ULP Program	1471
4.18.5	ULP program flow	1472
4.19	ESP32 中的单元测试	1472
4.19.1	添加常规测试用例	1473
4.19.2	添加多设备测试用例	1473
4.19.3	添加多阶段测试用例	1474
4.19.4	编译单元测试程序	1474
4.19.5	运行单元测试	1475
4.20	ESP32 中的单元测试	1476
4.20.1	添加常规测试用例	1476
4.20.2	添加多设备测试用例	1477
4.20.3	添加多阶段测试用例	1478
4.20.4	编译单元测试程序	1478
4.20.5	运行单元测试	1478
4.21	控制台终端	1480
4.21.1	行编辑	1480
4.21.2	将命令行拆分成参数列表	1481
4.21.3	参数解析	1482
4.21.4	命令的注册与调度	1482
4.21.5	示例	1482
4.22	ESP32 ROM console	1482
4.22.1	Full list of supported statements and functions	1483
4.22.2	Example programs	1484
4.22.3	Credits	1484
4.23	RF calibration	1484
4.23.1	Partial calibration	1484
4.23.2	Full calibration	1485
4.23.3	No calibration	1485
4.23.4	PHY initialization data	1485
4.24	Wi-Fi Driver	1485
4.24.1	ESP32 Wi-Fi Feature List	1485
4.24.2	How To Write a Wi-Fi Application	1486
4.24.3	ESP32 Wi-Fi API Error Code	1486
4.24.4	ESP32 Wi-Fi API Parameter Initialization	1487
4.24.5	ESP32 Wi-Fi Programming Model	1487
4.24.6	ESP32 Wi-Fi Event Description	1487
4.24.7	ESP32 Wi-Fi Station General Scenario	1490
4.24.8	ESP32 Wi-Fi AP General Scenario	1493
4.24.9	ESP32 Wi-Fi Scan	1493
4.24.10	ESP32 Wi-Fi Station Connecting Scenario	1501
4.24.11	ESP32 Wi-Fi Station Connecting When Multiple APs Are Found	1505
4.24.12	Wi-Fi Reconnect	1505
4.24.13	Wi-Fi Beacon Timeout	1505


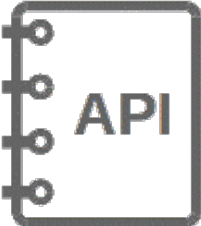


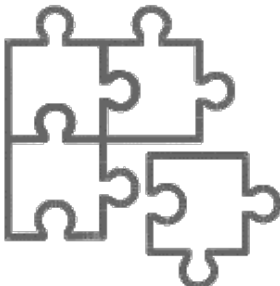

4.24.14	ESP32 Wi-Fi Configuration	1505
4.24.15	Wi-Fi Security	1511
4.24.16	ESP32 Wi-Fi Power-saving Mode	1512
4.24.17	ESP32 Wi-Fi Connect Crypto	1512
4.24.18	ESP32 Wi-Fi Throughput	1513
4.24.19	Wi-Fi 80211 Packet Send	1513
4.24.20	Wi-Fi Sniffer Mode	1515
4.24.21	Wi-Fi Multiple Antennas	1515
4.24.22	Wi-Fi Channel State Information	1516
4.24.23	Wi-Fi Channel State Information Configure	1517
4.24.24	Wi-Fi HT20/40	1518
4.24.25	Wi-Fi QoS	1518
4.24.26	Wi-Fi AMSDU	1519
4.24.27	Wi-Fi Fragment	1519
4.24.28	WPS Enrolle	1519
4.24.29	Wi-Fi Buffer Usage	1519
4.24.30	Wi-Fi Menuconfig	1520
4.24.31	Troubleshooting	1523
4.25	ESP-BLE-MESH	1528
4.25.1	Getting Started with ESP-BLE-MESH	1529
4.25.2	ESP-BLE-MESH Examples	1536
4.25.3	ESP-BLE-MESH Demo Videos	1536
4.25.4	ESP-BLE-MESH FAQ	1536
4.25.5	Related Documents	1536
4.26	ESP-MESH	1563
4.26.1	Overview	1563
4.26.2	Introduction	1563
4.26.3	ESP-MESH Concepts	1564
4.26.4	Building a Network	1570
4.26.5	Managing a Network	1575
4.26.6	Data Transmission	1578
4.26.7	Channel Switching	1580
4.26.8	Performance	1583
4.26.9	Further Notes	1584
4.27	BluFi	1584
4.27.1	概览	1584
4.27.2	BluFi 流程	1584
4.27.3	ESP32 配网流程	1584
4.27.4	配网流程图	1585
4.27.5	BluFi 传输格式	1585
4.27.6	ESP32 端的安全实现	1589
4.27.7	GATT 相关说明	1590
4.28	片外 RAM	1590
4.28.1	简介	1590
4.28.2	硬件	1590
4.28.3	配置片外 RAM	1591
4.28.4	片外 RAM 使用限制	1592
4.28.5	芯片版本	1592
4.29	链接脚本生成机制	1593
4.29.1	概述	1593
4.29.2	快速上手	1593
4.29.3	组件	1596
4.29.4	与构建系统的集成	1600
4.30	工具	1601
4.30.1	IDF Docker Image	1601

<b>5</b>	<b>Libraries and Frameworks</b>	<b>1603</b>
5.1	Cloud Frameworks	1603



5.1.1	AWS IoT	1603
5.1.2	Azure IoT	1603
5.1.3	Google IoT Core	1603
<b>6</b>	<b>Contributions Guide</b>	<b>1605</b>
6.1	How to Contribute	1605
6.2	Before Contributing	1605
6.3	Pull Request Process	1605
6.4	Legal Part	1605
6.5	Related Documents	1606
6.5.1	Espressif IoT Development Framework Style Guide	1606
6.5.2	Documenting Code	1609
6.5.3	Documentation Add-ons and Extensions Reference	1617
6.5.4	创建示例项目	1618
6.5.5	API Documentation Template	1619
6.5.6	Contributor Agreement	1621
<b>7</b>	<b>ESP-IDF 版本简介</b>	<b>1625</b>
7.1	发布版本	1625
7.2	我该选择哪个版本?	1626
7.3	版本管理	1626
7.4	查看当前版本	1627
7.5	Git 工作流	1627
7.6	更新 ESP-IDF	1628
7.6.1	更新至一个稳定发布版本	1628
7.6.2	更新至一个预发布版本	1628
7.6.3	更新至 master 分支	1628
7.6.4	更新至一个发布分支	1629
<b>8</b>	<b>资源</b>	<b>1631</b>
<b>9</b>	<b>Copyrights and Licenses</b>	<b>1633</b>
9.1	Software Copyrights	1633
9.1.1	Firmware Components	1633
9.1.2	Build Tools	1634
9.2	ROM Source Code Copyrights	1634
9.3	Xtensa libhal MIT License	1634
9.4	TinyBasic Plus MIT License	1635
9.5	TJpgDec License	1635
<b>10</b>	<b>关于本指南</b>	<b>1637</b>
<b>11</b>	<b>Switch Between Languages/切换语言</b>	<b>1639</b>
	索引	1641
	索引	1641

这里是乐鑫 IoT 开发框架 ([esp-idf](#)) 的文档中心。ESP-IDF 是 [ESP32](#) 芯片的官方开发框架。

		
<a href="#">快速入门</a>	<a href="#">API 参考</a>	<a href="#">H/W 参考</a>
		
<a href="#">API 指南</a>	<a href="#">贡献代码</a>	<a href="#">相关资源</a>



# 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。

---

**注解：** 这是 ESP-IDF 稳定版本 v4.0.3 的文档，还有其他版本的文档[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 4.2
- 高性能双核
- 超低功耗协处理器
- 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

硬件：

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

软件：

- 设置 **工具链**，用于编译 ESP32 代码；
- **编译工具**——CMake 和 Ninja 编译工具，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
- 安装 C 语言编程 (工程) 的 **文本编辑器**，例如 [Eclipse](#)

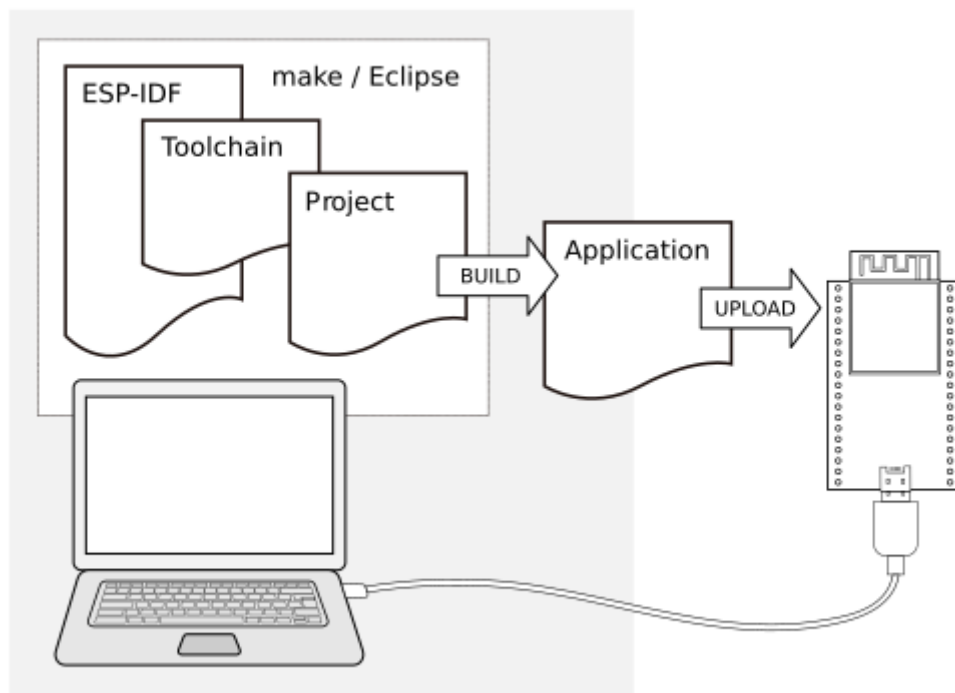


图 1: ESP32 应用程序开发

## 1.3 开发板简介

请点击下方连接，了解有关具体开发板的详细信息。

### 1.3.1 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。有关 ESP32-DevKitC 其他版本的介绍，请见：[ESP32 H/W 硬件参考](#)。

#### 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：

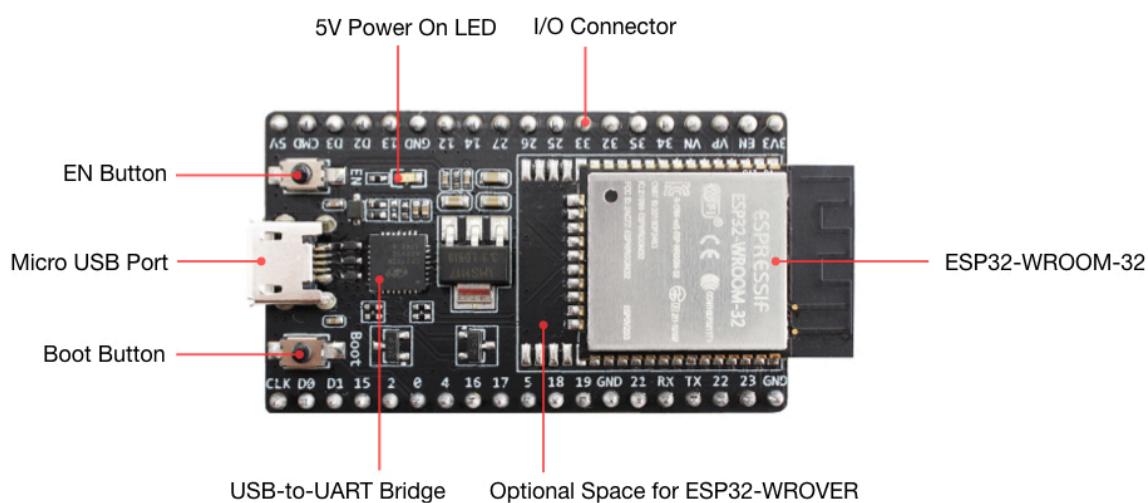
- 可选多款 ESP32 模组
  - [ESP32-WROOM-32](#)
  - [ESP32-WROOM-32D](#)
  - [ESP32-WROOM-32U](#)
  - [ESP32-SOLO-1](#)

- *ESP32-WROVER*
- *ESP32-WROVER-B*
- *ESP32-WROVER-I*
- *ESP32-WROVER-I (IPEX)*
- 可选排针或排母

详情请见《乐鑫产品订购信息》。

## 功能说明

ESP32-DevKitC V4 开发板的主要组件、接口及控制方式见下。



ESP32-DevKitC V4 (板载 ESP32-WROOM-32)

主要组件	基本介绍
ESP32-WROOM-32	基于 ESP32 的模组。更多详情，请见《 <a href="#">ESP32-WROOM-32 技术规格书</a> 》。
EN	复位按键。
Boot	下载按键。按下 <b>Boot</b> 键并保持，同时按一下 <b>EN</b> 键（此时不要松开 <b>Boot</b> 键）进入“固件下载”模式，通过串口下载固件。
USB-to-UART 桥接器	单芯片 USB-UART 桥接器，可提供高达 3 Mbps 的传输速率。
Micro USB 端口	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 <a href="#">相关文档</a> 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。

**注解：**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。

**注解：**管脚 GPIO16 和 GPIO17 仅适用于板载 ESP32-WROOM 系列和 ESP32-SOLO-1 的开发板，保留内部使用。

## 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

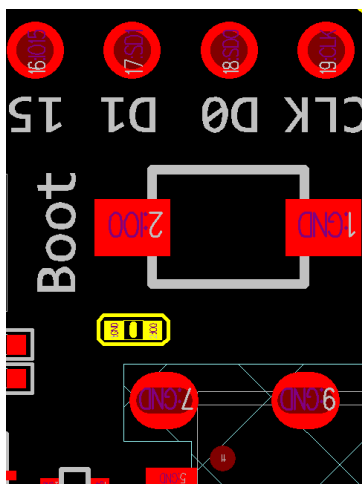
**警告：**上述供电模式 **不可同时连接**，否则可能会损坏开发板和/或电源。

## 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：

- 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。



C15（黄色）在 ESP32-DevKitC V4 开发板上的位置

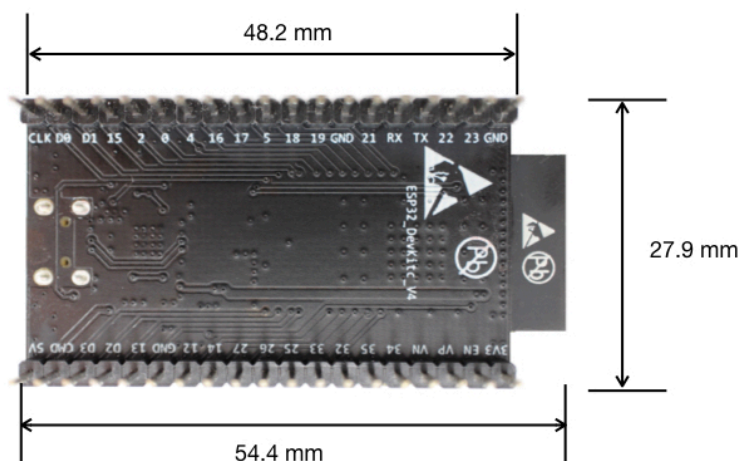
## 应用程序开发

ESP32-DevKitC V4 上电前，请首先确认开发板完好无损。

之后，请前往[快速入门](#)中的[详细安装步骤](#)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-DevKitC 开发板尺寸-仰视图



### 相关文档

- [ESP32-DevKitC V4 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-WROVER 技术规格书》 \(PDF\)](#)
- [《ESP32-WROVER-B 技术规格书》 \(PDF\)](#)
- [《乐鑫产品订购信息》 \(PDF\)](#)

**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 Mac OS)

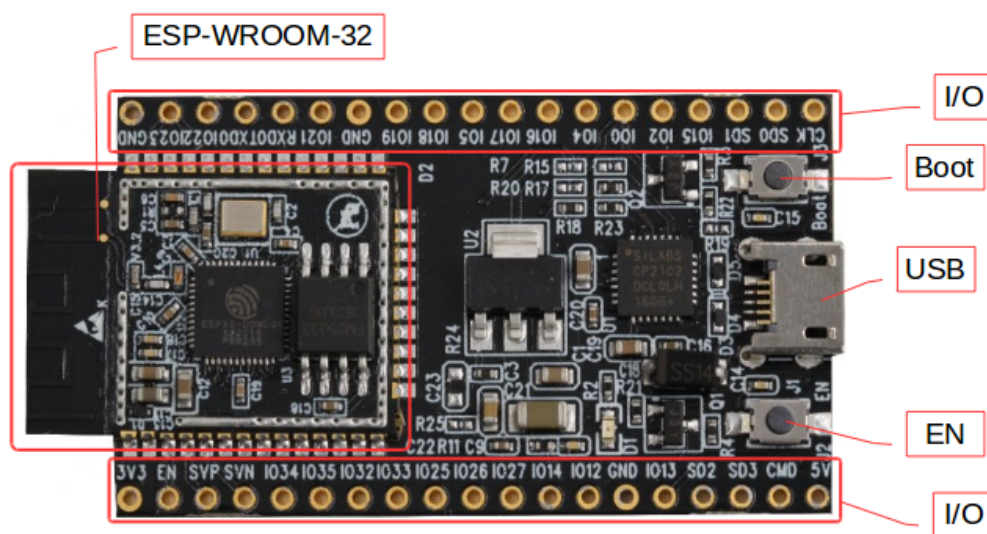
您可以跳过介绍部分，直接前往[应用程序开发](#) 章节。

**概述** ESP32-DevKitC V2 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。

**功能说明** ESP32-DevKitC V2 开发板的主要组件、接口及控制方式见下。

ESP32-DevKitC V2 开发板





主要组件	基本介绍
ESP32-WROOM-32	基于 ESP32 的模组。更多详情，请见 <a href="#">《ESP32-WROOM-32 技术规格书》</a> 。
EN	复位按键。
Boot	下载按键。按下 <b>Boot</b> 键并保持，同时按一下 <b>EN</b> 键（此时不要松开 <b>Boot</b> 键）进入“固件下载”模式，通过串口下载固件。
Micro USB 端口	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。
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。

**应用程序开发** ESP32-DevKitC V2 上电前，请首先确认开发板完好无损。

之后，请前往[快速入门](#)中的[详细安装步骤](#)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》 \(PDF\)](#)

## 1.3.2 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。有关 ESP-WROVER-KIT 其他版本的介绍，请见：[ESP32 H/W 硬件参考](#)。

### 准备工作

- [ESP-WROVER-KIT V4.1 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。

### 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-B 模组
- LCD 屏
- MicroSD 卡槽

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。

**注解：**ESP32 的 GPIO16 和 GPIO17 管脚用作 PSRAM 的片选和时钟信号。默认情况下，为了给用户提供的可靠的性能，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。

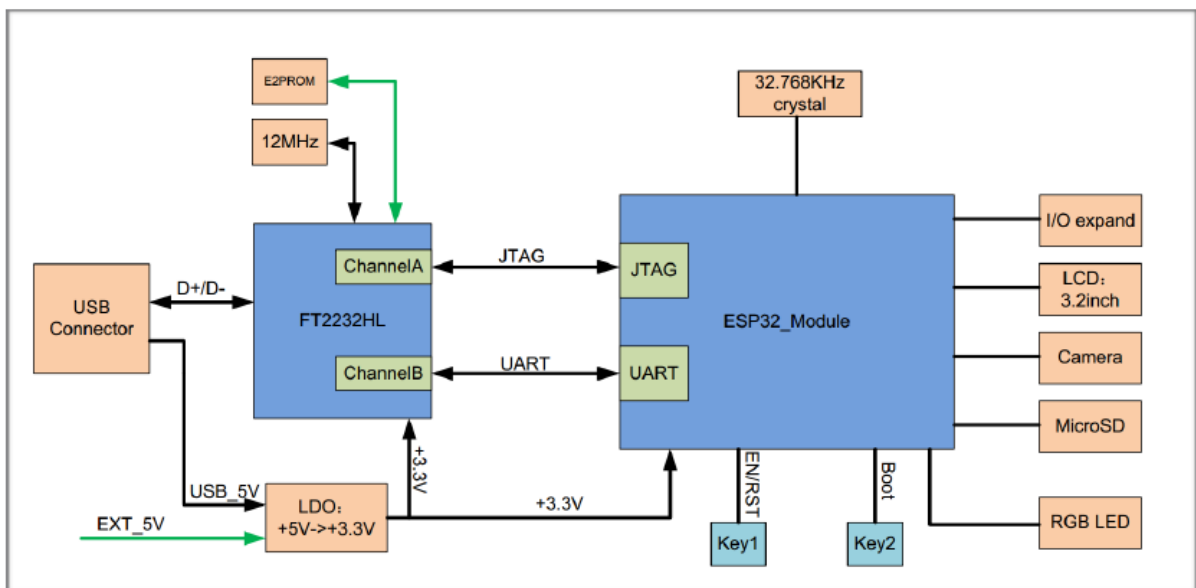


图 2: ESP-WROVER-KIT 功能框图

## 功能说明

ESP-WROVER-KIT 开发板的主要组件、接口及控制方式见下。

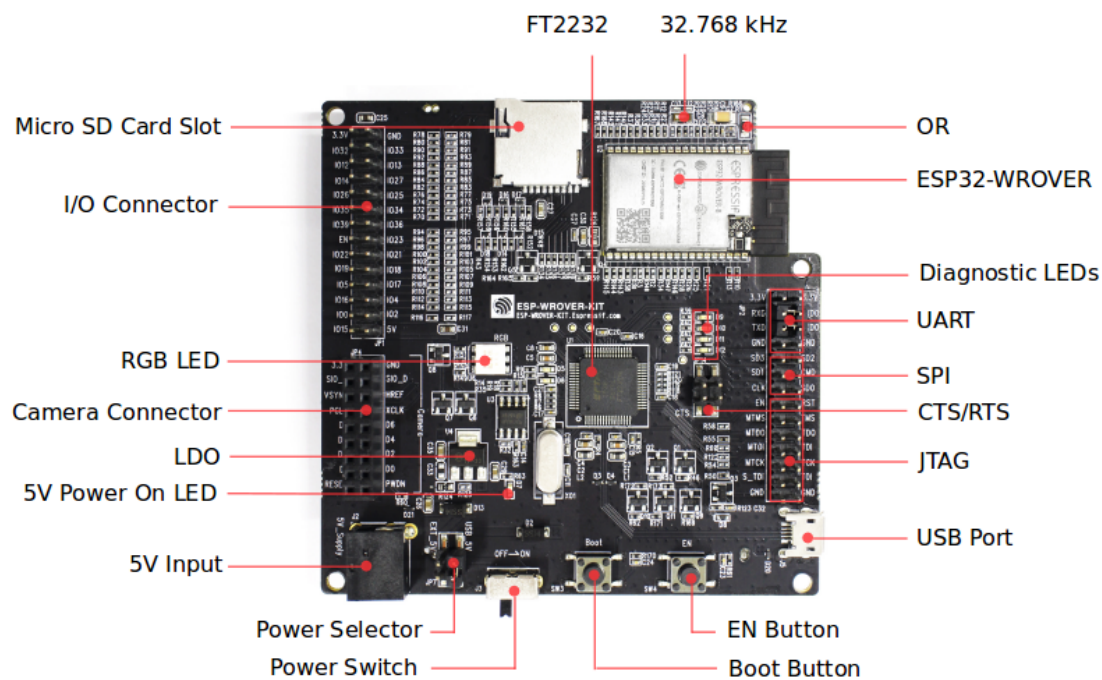


图 3: ESP-WROVER-KIT 开发板布局-俯视图

下表将从图片右上角开始，以顺时针顺序介绍图 1 中的主要组件，然后按同样顺序介绍图 2 中的主要组件。

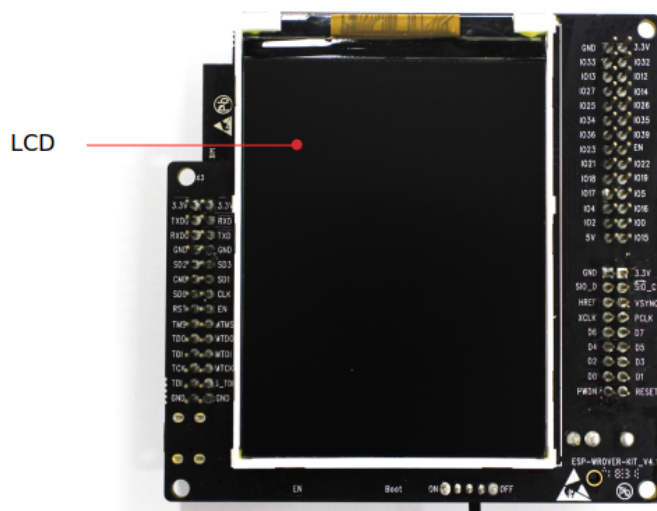
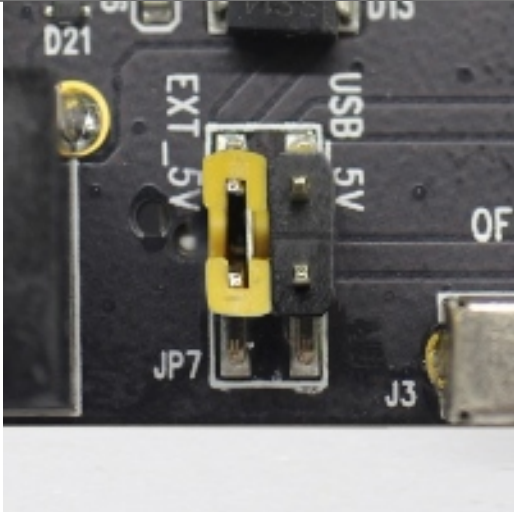
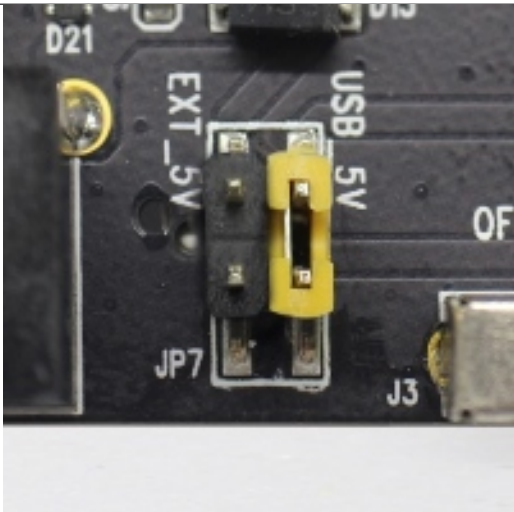
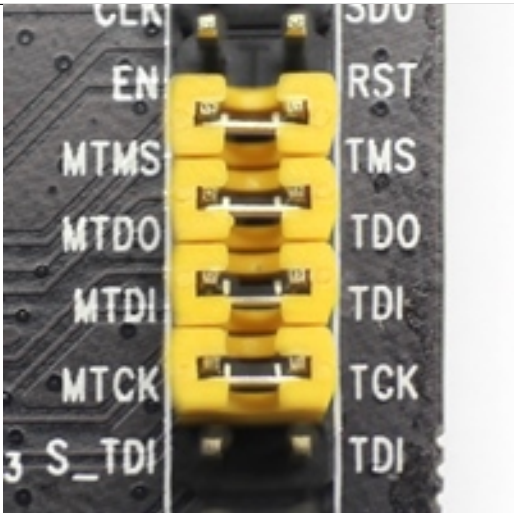
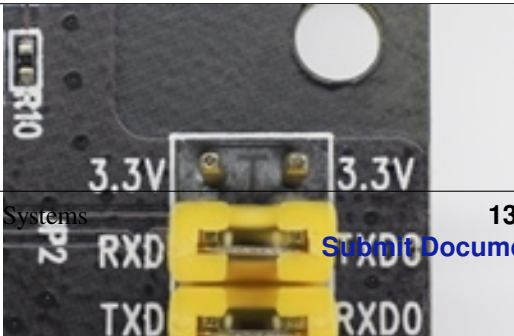


图 4: ESP-WROVER-KIT 开发板布局-仰视图

主要组件	基本介绍
FT2232	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，与 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，便利开发人员的应用开发与调试。详见 <a href="#">ESP-WROVER-KIT V4.1 原理图</a> 。
32.768 kHz	外接 32.768 kHz 晶振，可提供 Deep-sleep 下使用的低功耗时钟。
0 欧电阻	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。
ESP32-WROVER-B 模组	ESP-WROVER 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。
诊断 LED 信号灯	本开发板 FT2232 芯片的 GPIO 管脚连接了 4 个红色 LED 信号灯，以备后用。
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，本接口的工作电压为 3.3 V。
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。
JTAG	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 <a href="#">设置选项</a> 的介绍，连接跳线帽。
USB 端口	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。
EN	复位按键。
Boot 按键	下载按键。按下 <b>Boot</b> 键并保持，同时按一下 <b>EN</b> 键（此时不要松开 <b>Boot</b> 键）进入“固件下载”模式，通过串口下载固件。
电源开关	拨向 <b>Boot</b> 按键一侧，开发板上电；拨离 <b>Boot</b> 按键一侧，开发板掉电。
电源选择开关	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 <a href="#">章节设置选项</a> 中有关 JP7 连接器的描述。
5V Input	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。
5V	当开发板通电后（USB 或外部 5V 供电），该红色指示灯将亮起。

### 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：

排针	跳线设置	功能描述
JP7		使用外部电源为 ESP-WROVER-KIT 开发板供电
JP7		使用 USB 端口为 ESP-WROVER-KIT 开发板供电
JP2		使能 JTAG 功能
Espressif Systems		<p style="text-align: center;">13</p> <p style="text-align: center;"><a href="#">Submit Document Feedback</a></p>

**ESP32 管脚分配**

ESP32 模组的部分管脚/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 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。

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

说明：

- NC/XTAL - [32.768 kHz 晶振](#)
- 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 - ESP32-WROVER-B 的 PSRAM

**32.768 kHz 晶振**

	ESP32 管脚
1.	GPIO32
2.	GPIO33

**注解：**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 GP1O32 和 GPIO33 的连接从晶振移至 JP1。

### SPI Flash / JP2

.	ESP32 管脚
1.	CLK / GPIO6
2.	SD0 / GPIO7
3.	SD1 / GPIO8
4.	SD2 / GPIO9
5.	SD3 / GPIO10
6.	CMD / GPIO11

**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

.	ESP32 管脚	JTAG 信号
1.	EN	TRST_N
2.	MTMS / GPIO14	TMS
3.	MTDO / GPIO15	TDO
4.	MTDI / GPIO12	TDI
5.	MTCK / GPIO13	TCK

### 摄像头 / JP4



.	ESP32 管脚	摄像头信号
1.	n/a	3.3V
2.	n/a	地
3.	GPIO27	SIO_C / SCCB 时钟
4.	GPIO26	SIO_D / SCCB 数据
5.	GPIO25	VSYNC / 垂直同步
6.	GPIO23	HREF / 水平参考
7.	GPIO22	PCLK / 像素时钟
8.	GPIO21	XCLK / 系统时钟
9.	GPIO35	D7 / 像素数据 Bit 7
10.	GPIO34	D6 / 像素数据 Bit 6
11.	GPIO39	D5 / 像素数据 Bit 5
12.	GPIO36	D4 / 像素数据 Bit 4
13.	GPIO19	D3 / 像素数据 Bit 3
14.	GPIO18	D2 / 像素数据 Bit 2
15.	GPIO5	D1 / 像素数据 Bit 1
16.	GPIO4	D0 / 像素数据 Bit 0
17.	GPIO0	RESET / 摄像头复位
18.	n/a	PWDN / 摄像头断电

- D0 到 D7 为摄像头的数据总线

### RGB LED

.	ESP32 管脚	RGB LED
1.	GPIO0	红色
2.	GPIO2	绿色
3.	GPIO4	蓝色

**MicroSD 卡**

.	ESP32 管脚	MicroSD 信号
1.	MTDI / GPIO12	DATA2
2.	MTCK / GPIO13	CD / DATA3
3.	MTDO / GPIO15	CMD
4.	MTMS / GPIO14	CLK
5.	GPIO2	DATA0
6.	GPIO4	DATA1
7.	GPIO21	CD

**LCD / U5**

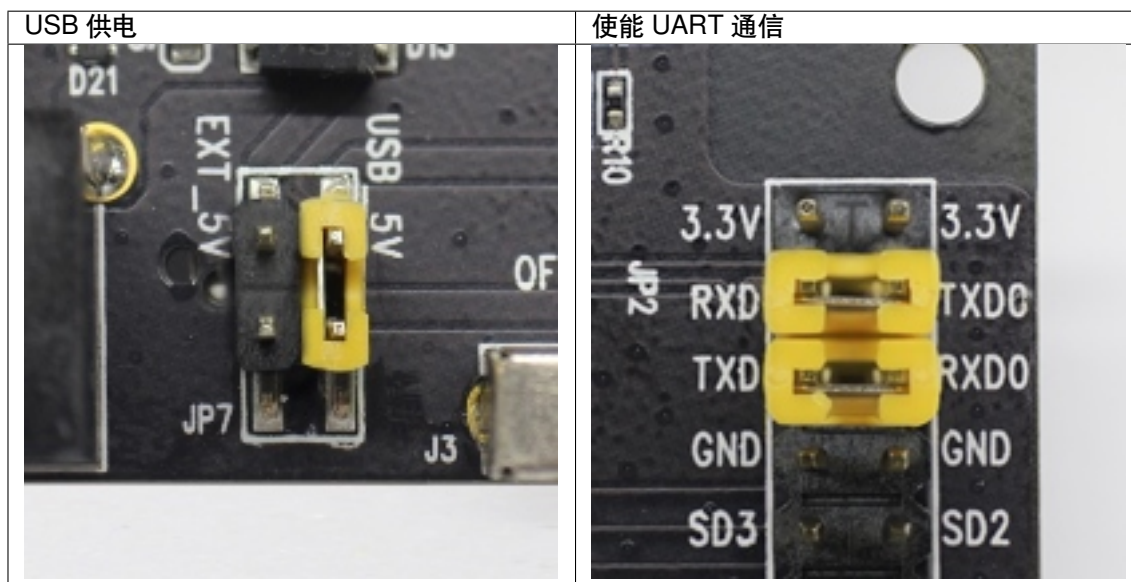
.	ESP32 管脚	LCD 信号
1.	GPIO18	复位
2.	GPIO19	SCL
3.	GPIO21	D/C
4.	GPIO22	CS
5.	GPIO23	SDA
6.	GPIO25	SDO
7.	GPIO5	背光

## 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 JP7 连接器，选择 USB 为开发板供电。
- 使用 JP2 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** 请前往[快速入门](#) 中的[详细安装步骤](#) 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[详细安装步骤](#) 章节。

## 相关文档

- [ESP-WROVER-KIT V4.1 原理图 \(PDF\)](#)
- [《ESP32 技术规格书》 \(PDF\)](#)
- [《ESP32-WROVER-B 技术规格书》 \(PDF\)](#)
- [JTAG 调试](#)
- [ESP32 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：[ESP32 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-WROOM-32](#)
- [ESP32-WROVER](#)

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。

**注解：**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，为了给用户提供的可靠的性能，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。

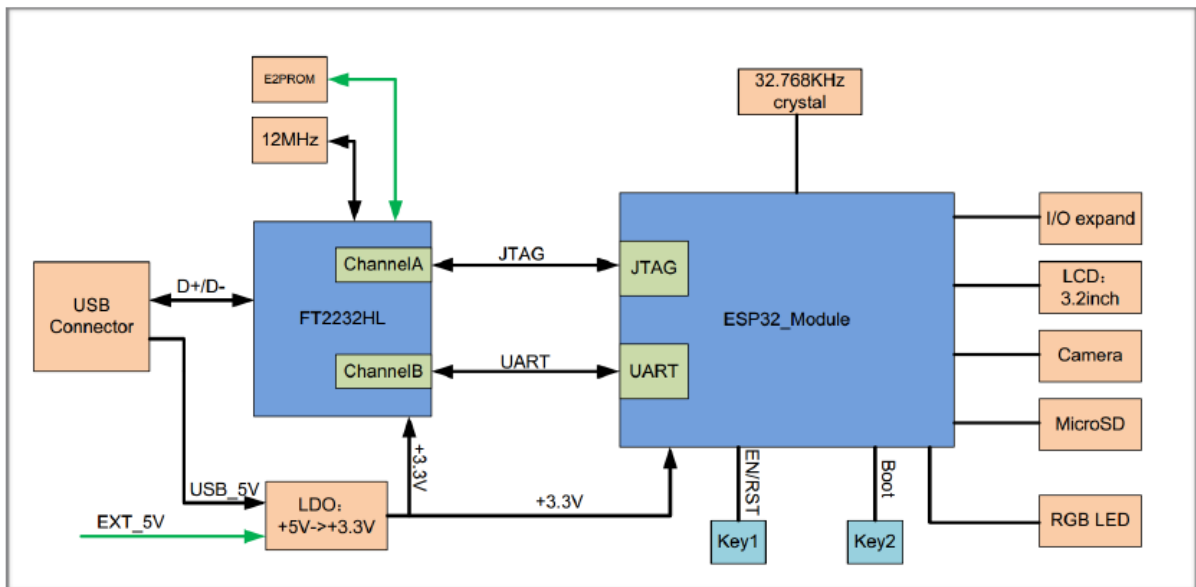


图 5: ESP-WROVER-KIT 功能框图

**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图 2 中的主要组件。

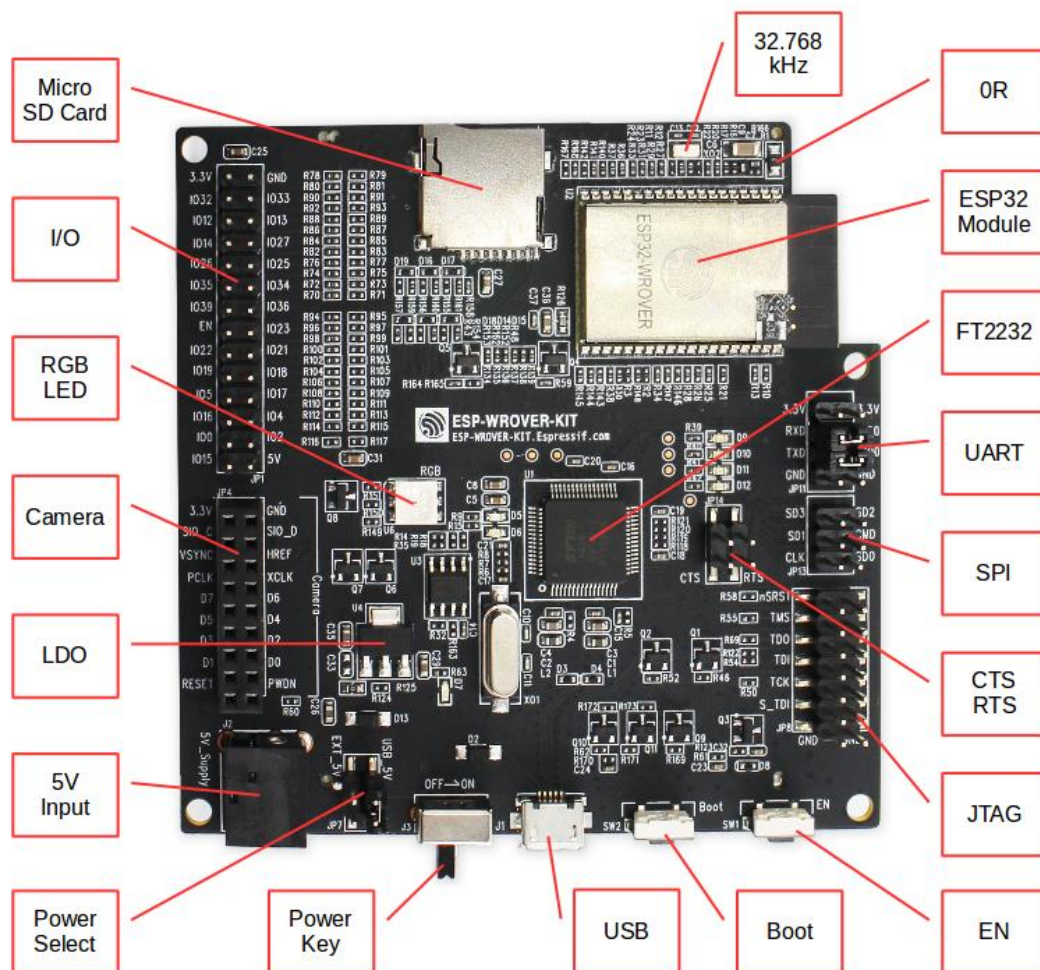


图 6: ESP-WROVER-KIT 开发板布局-俯视图

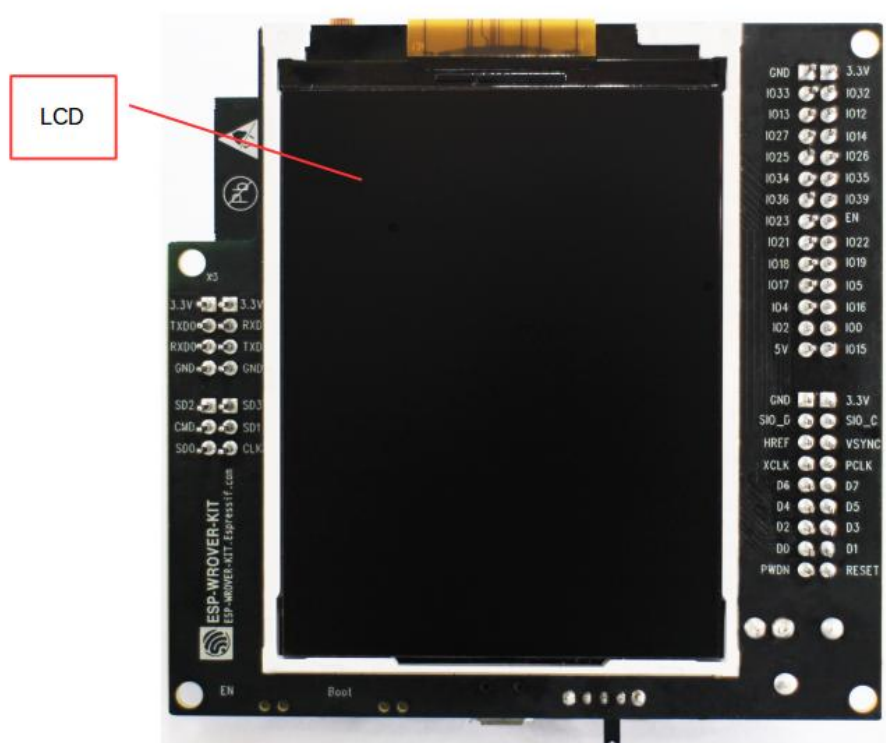
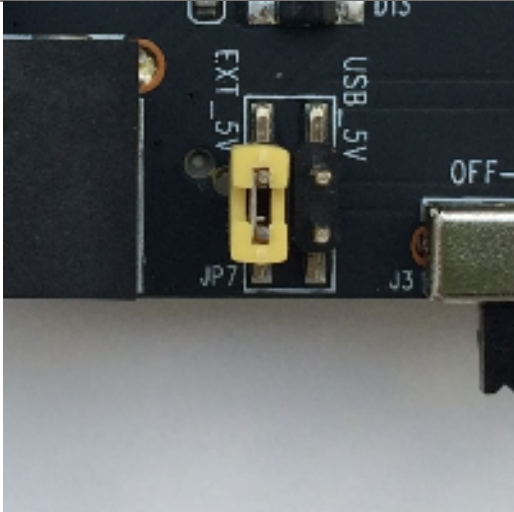
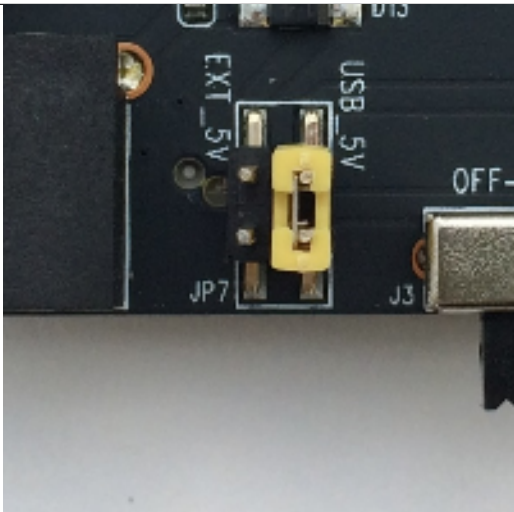
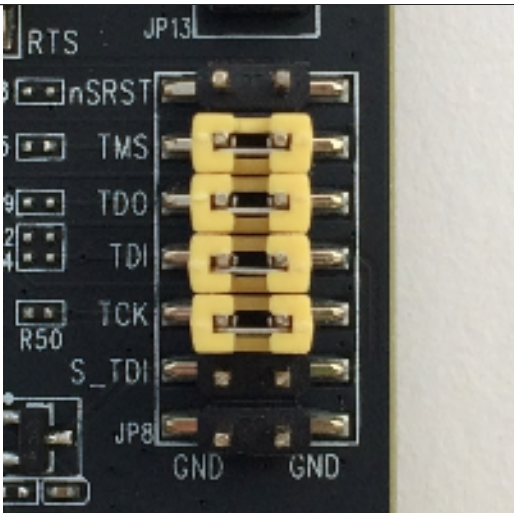
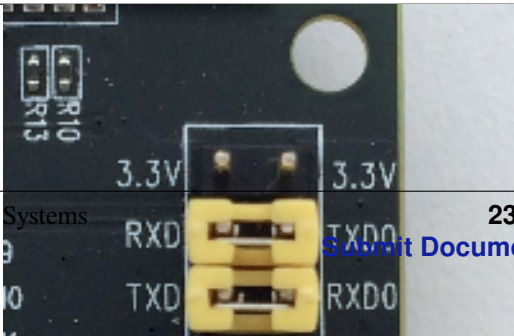


图 7: ESP-WROVER-KIT 开发板布局-仰视图

主要组件	基本介绍
32.768 kHz	外接 32.768 kHz 晶振，可提供 Deep-sleep 下使用的低功耗时钟。
0 欧电阻	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。
ESP32 模组	可选贴 ESP32-WROOM-32 或 ESP32-WROVER。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。
FT2232	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，与 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，便利开发人员的应用开发与调试。见 <a href="#">ESP-WROVER-KIT V3 原理图</a> 。
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，选贴 ESP32-WROVER 模组时，该接口的工作电压为 1.8 V；选贴 ESP32-WROOM-32 时，该接口的工作电压为 3.3V。
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。
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 <a href="#">设置选项</a> 的介绍，连接跳线帽。
EN	复位按键。
Boot	下载按键。按下 <b>Boot</b> 键并保持，同时按一下 <b>EN</b> 键（此时不要松开 <b>Boot</b> 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。
电源开关	拨向 <b>USB</b> 按键一侧，开发板上电；拨离 <b>USB</b> 按键一侧，开发板掉电。
电源选择开关	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 <a href="#">设置选项</a> 中有关 JP7 连接器的描述。
5V Input	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。
LDO	5V-to-3.3V 低压差线性稳压器 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 <a href="#">ESP-WROVER-KIT V3 原理图</a> 。
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	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。
LCD 显示屏	支持贴装一款 3.2” 的 SPI（标准四线串行外设接口）LCD 显示器，请见 <a href="#">ESP-WROVER-KIT 开发板布局-仰视图</a> 。

**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：

排针	跳线设置	功能描述
JP7		使用外部电源为 ESP-WROVER-KIT 开发板供电
JP7		使用 USB 端口为 ESP-WROVER-KIT 开发板供电
JP8		使能 JTAG 功能
Espressif Systems		<p style="text-align: center;">23</p> <p style="text-align: center;"><a href="#">Submit Document Feedback</a></p> <p style="text-align: right;">Release v4.0.3</p>



**ESP32 管脚分配** ESP32 模组的部分管脚/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 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。

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

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 - 仅适用于选贴 ESP32-WROVER 的情况。

### 32.768 kHz 晶振

	ESP32 管脚
1.	GPIO32
2.	GPIO33

**注解：**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 欧姆电阻移至 R12/R24 处，以将 GPIO32 和

GPIO33 的连接从晶振移至 JP1。

### SPI Flash / JP13

.	ESP32 管脚
1.	CLK / GPIO6
2.	SD0 / GPIO7
3.	SD1 / GPIO8
4.	SD2 / GPIO9
5.	SD3 / GPIO10
6.	CMD / GPIO11

**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

.	ESP32 管脚	JTAG 信号
1.	EN	TRST_N
2.	MTMS / GPIO14	TMS
3.	MTDO / GPIO15	TDO
4.	MTDI / GPIO12	TDI
5.	MTCK / GPIO13	TCK

### 摄像头 / JP4

.	ESP32 管脚	摄像头信号
1.	n/a	3.3V
2.	n/a	地
3.	GPIO27	SIO_C / SCCB 时钟
4.	GPIO26	SIO_D / SCCB 数据
5.	GPIO25	VSYNC / 垂直同步
6.	GPIO23	HREF / 水平参考
7.	GPIO22	PCLK / 像素时钟
8.	GPIO21	XCLK / 系统时钟
9.	GPIO35	D7 / 像素数据 Bit 7
10.	GPIO34	D6 / 像素数据 Bit 6
11.	GPIO39	D5 / 像素数据 Bit 5
12.	GPIO36	D4 / 像素数据 Bit 4
13.	GPIO19	D3 / 像素数据 Bit 3
14.	GPIO18	D2 / 像素数据 Bit 2
15.	GPIO5	D1 / 像素数据 Bit 1
16.	GPIO4	D0 / 像素数据 Bit 0
17.	GPIO0	RESET / 摄像头复位
18.	n/a	PWDN / 摄像头断电

- D0 到 D7 为摄像头的数据总线

### RGB LED

.	ESP32 管脚	RGB LED
1.	GPIO0	红色
2.	GPIO2	绿色
3.	GPIO4	蓝色

### MicroSD 卡

.	ESP32 管脚	MicroSD 信号
1.	MTDI / GPIO12	DATA2
2.	MTCK / GPIO13	CD / DATA3
3.	MTDO / GPIO15	CMD
4.	MTMS / GPIO14	CLK
5.	GPIO2	DATA0
6.	GPIO4	DATA1
7.	GPIO21	CD

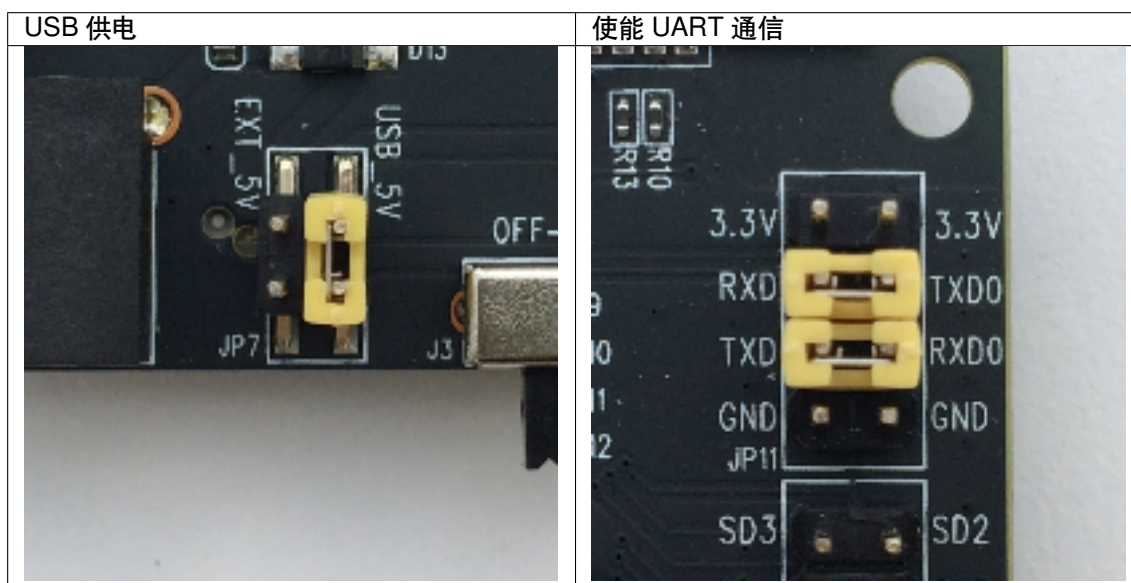
### LCD / U5

.	ESP32 管脚	LCD 信号
1.	GPIO18	复位
2.	GPIO19	SCL
3.	GPIO21	D/C
4.	GPIO22	CS
5.	GPIO23	SDA
6.	GPIO25	SDO
7.	GPIO5	背光

**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 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** 请前往[快速入门](#)中的[详细安装步骤](#)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](#)
- [ESP32 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：[ESP32 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 可选贴以下 ESP32 模组：

- [ESP32-WROOM-32](#)
- [ESP32-WROVER](#)

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。

**注解：**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。

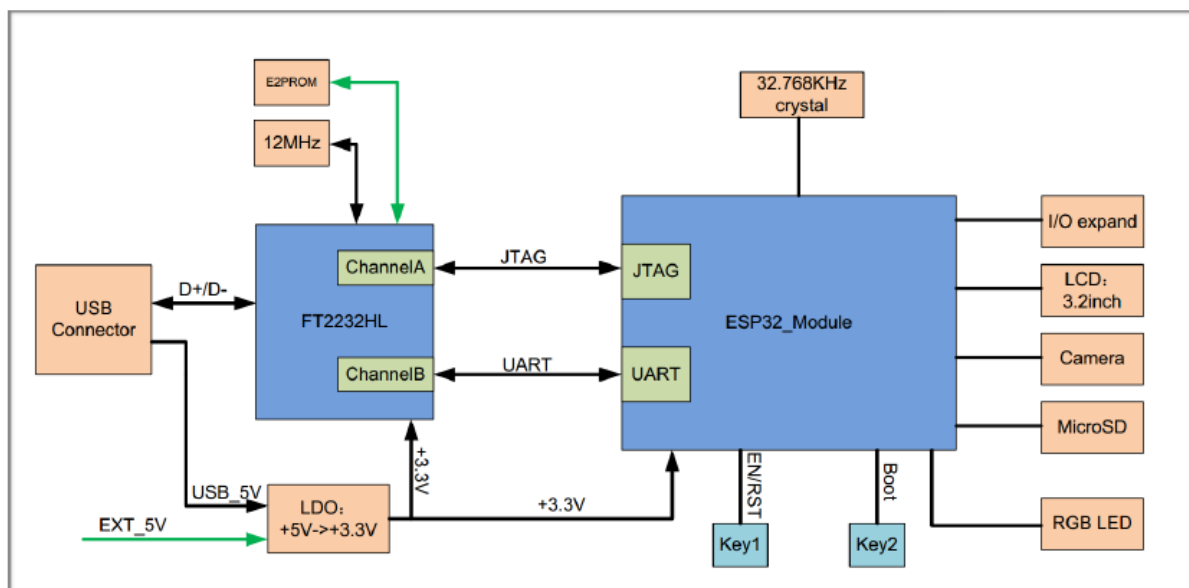


图 8: ESP-WROVER-KIT 功能框图

**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。

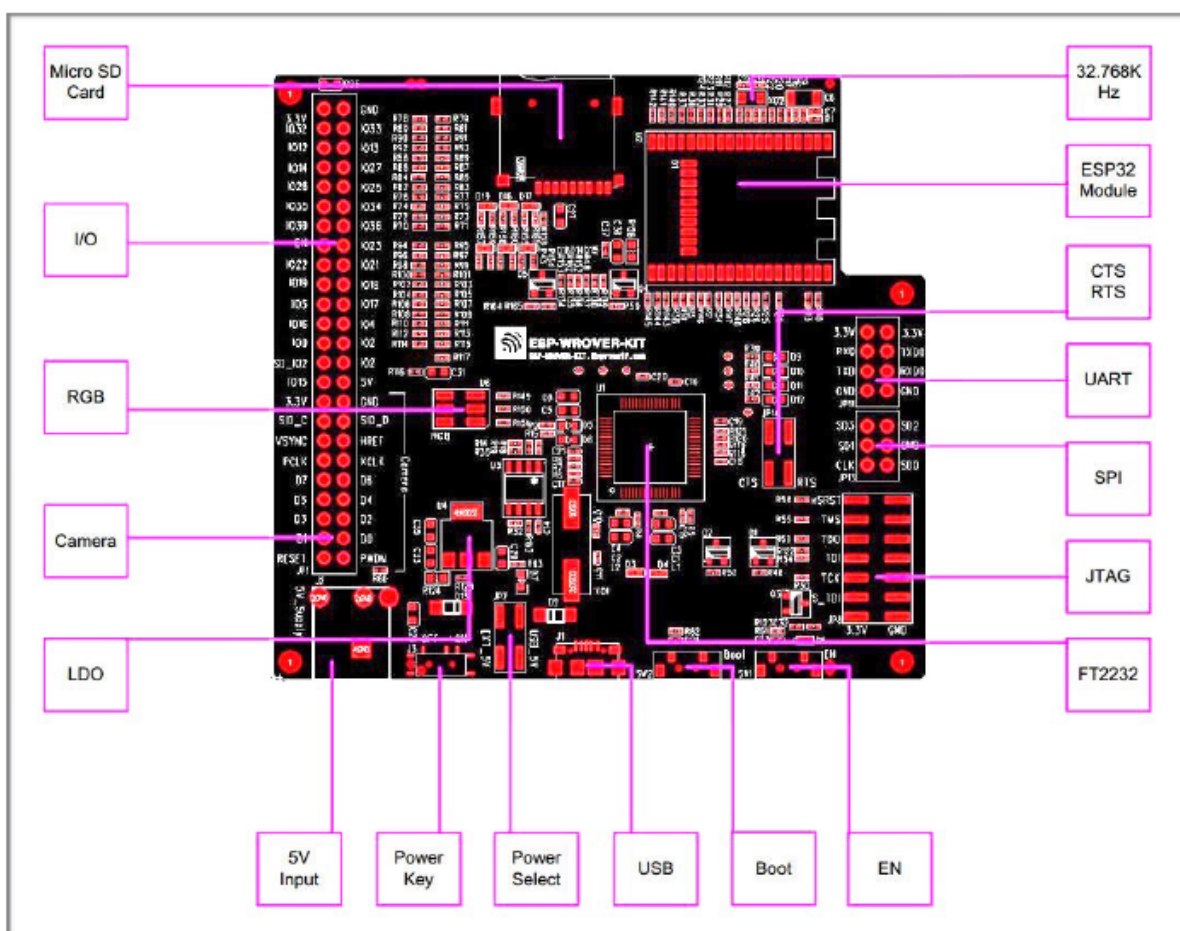


图 9: ESP-WROVER-KIT 开发板布局-俯视图

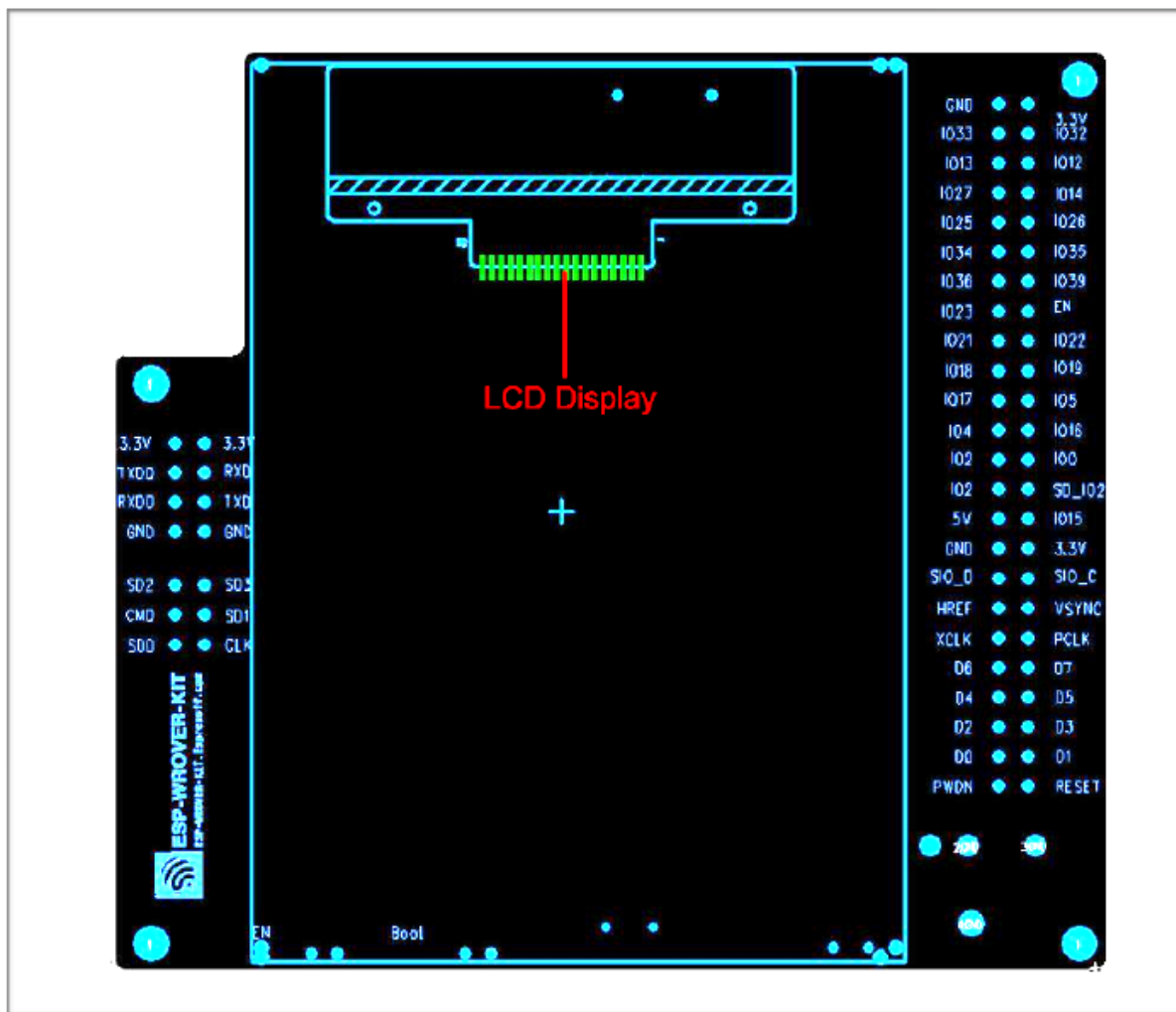
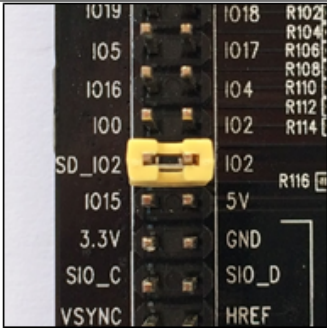
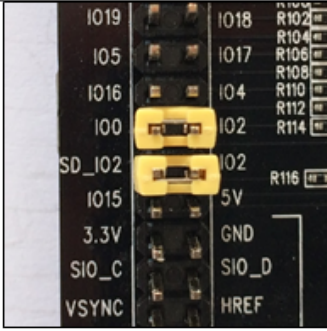
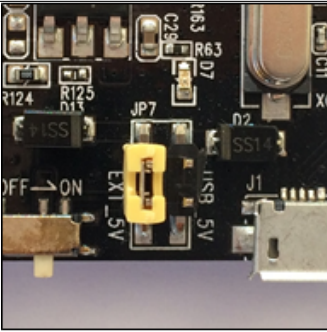
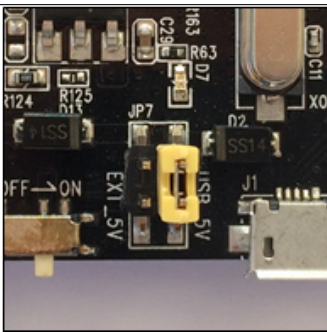
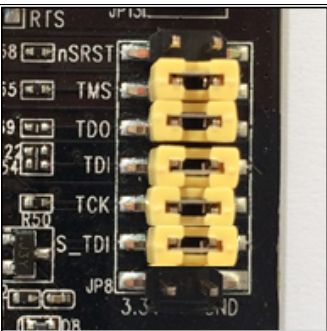
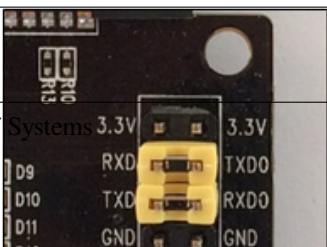


图 10: ESP-WROVER-KIT 开发板布局-仰视图



主要组件	基本介绍
32.768 kHz	外接 32.768 kHz 晶振，可提供 Deep-sleep 下使用的低功耗时钟。
ESP32 模组	可选贴 ESP32-WROOM-32 或 ESP32-WROVER。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。
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，选贴 ESP32-WROVER 模组时，该接口的工作电压为 1.8 V；选贴 ESP32-WROOM-32 时，该接口的工作电压为 3.3V。
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 <a href="#">设置选项</a> 的介绍，连接跳线帽。
FT2232	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，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 <b>Boot</b> 键并保持，同时按一下 <b>EN</b> 键（此时不要松开 <b>Boot</b> 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 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 <a href="#">设置选项</a> 中有关 JP7 连接器的描述。
电源开关	拨向 <b>USB</b> 按键一侧，开发板上电；拨离 <b>USB</b> 按键一侧，开发板掉电。
5V Input	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。
LDO	5V-to-3.3V 低压差线型稳压器 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 <a href="#">ESP-WROVER-KIT V2 原理图</a> 。
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 <a href="#">设置选项</a> 章节的要求，连接 JP1 连接器。
LCD 显示屏	支持贴装一款 3.2” 的 SPI（标准四线串行外设接口）LCD 显示器，请见 <a href="#">ESP-WROVER-KIT 开发板布局-仰视图</a> 。

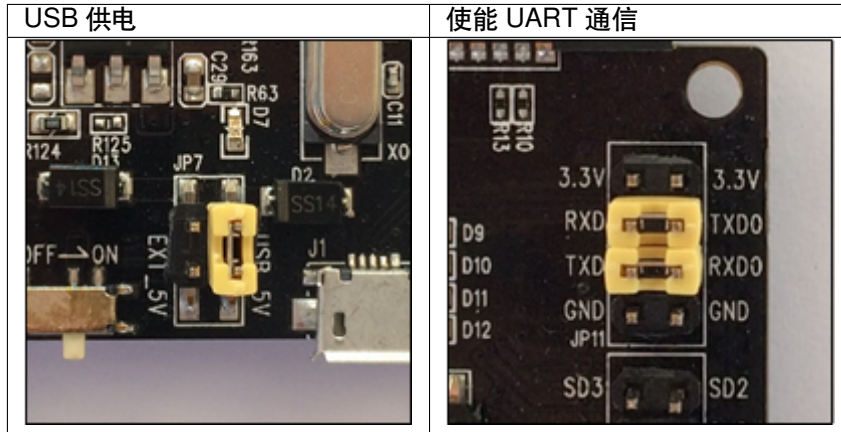
**设置选项** 用户可通过 5 组排针，设置开发板功能，其中常见功能见下表：

排针	跳线设置	功能描述
JP1		使能 MicroSD Card 功能的上拉电阻
JP1		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）
JP7		使用外部电源为 ESP-WROVER-KIT 开发板供电
JP7		使用 USB 端口为 ESP-WROVER-KIT 开发板供电
JP8		使能 JTAG 功能
Espressif Systems		

**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 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** 请前往[快速入门](#)中的[详细安装步骤](#)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](#)
- [ESP32 H/W 硬件参考](#)

### 1.3.3 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：[ESP32 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 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 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-PICO-KIT 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。与其他 ESP32 系列模组相比，ESP32-PICO-D4 模组已完整集成以下外围器件：

- 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器件的数量和成本，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。为了方便杜邦线的使用，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。

## 注解：

1. 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。
2. ESP32-PICO-D4 开发板默认采用排针。

## 功能概述

ESP32-PICO-KIT 开发板的主要组件和连接方式见下。

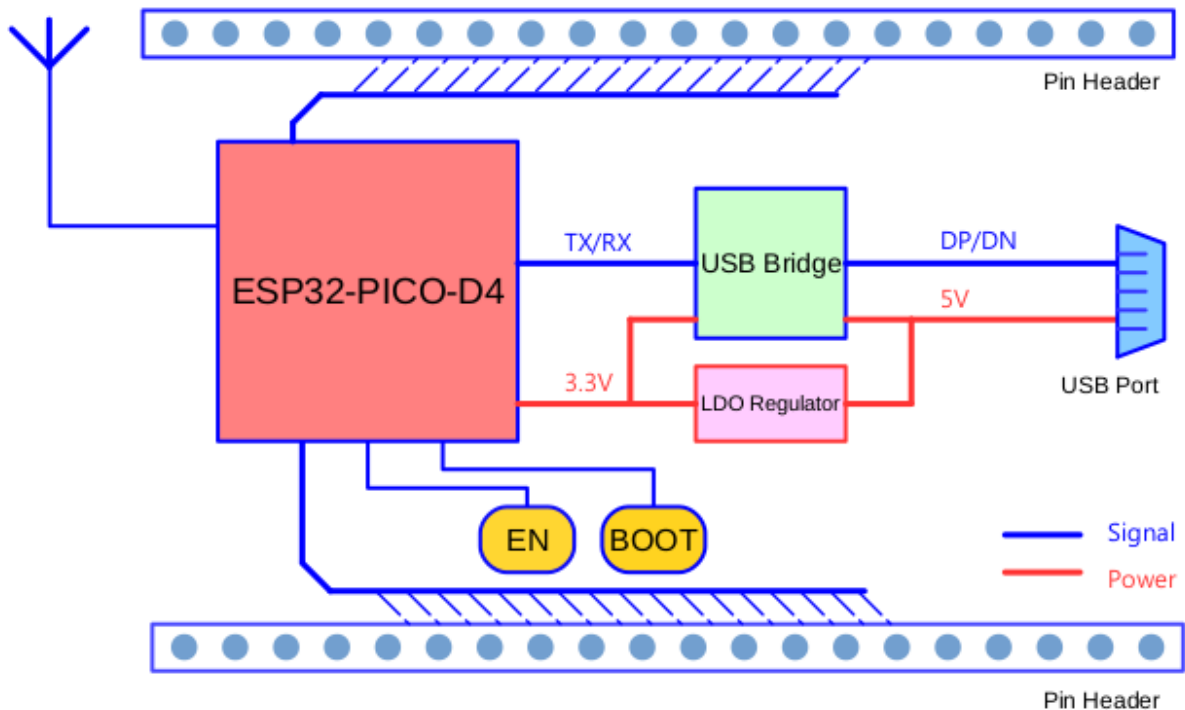


图 11: ESP32-PICO-KIT 框图

## 功能说明

ESP32-PICO-KIT 开发板的主要组件、接口及控制方式见下。

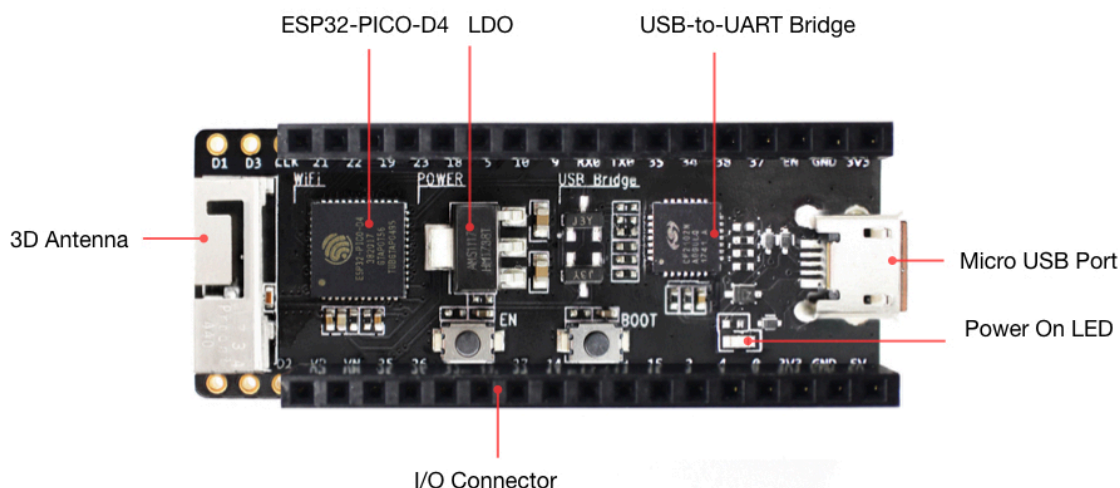


图 12: ESP32-PICO-KIT 开发板布局

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）。

主要组件	基本介绍
ESP32-PICO-D4	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。
LDO	5V-to-3.3V 低压差稳压器
USB-to-UART 桥接器	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。
Micro USB 端口	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。
5V Power On LED	开发板通电后，该红色指示灯将亮起。更多信息，请见 <a href="#">相关文档</a> 中的原理图。
I/O	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 <a href="#">管脚说明</a> 。
BOOT	下载按键。按下 <b>Boot</b> 键并保持，同时按一下 <b>EN</b> 键（此时不要松开 <b>Boot</b> 键）进入“固件下载”模式，通过串口下载固件。
EN	复位按键。

## 电源选项

开发板可任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](#)。

## Header J2

编号	名称	类型	功能
1	FLASH_SD1 (FSD1)	I/O	GPIO8, SD_DATA1, SPID, HS1_DATA1 (见说明 1), U2CTS
2	FLASH_SD3 (FSD3)	I/O	GPIO7, SD_DATA0, SPIQ, HS1_DATA0 (见说明 1), U2RTS
3	FLASH_CLK (FCLK)	I/O	GPIO6, SD_CLK, SPICLK, HS1_CLK (见说明 1), U1CTS
4	IO21	I/O	GPIO21, VSPIHD, EMAC_TX_EN
5	IO22	I/O	GPIO22, VSPIWP, U0RTS, EMAC_TXD1
6	IO19	I/O	GPIO19, VSPIQ, U0CTS, EMAC_TXD0
7	IO23	I/O	GPIO23, VSPID, HS1_STROBE
8	IO18	I/O	GPIO18, VSPICLK, HS1_DATA7
9	IO5	I/O	GPIO5, VSPICS0, HS1_DATA6, EMAC_RX_CLK
10	IO10	I/O	GPIO10, SD_DATA3, SPIWP, HS1_DATA3, U1TXD
11	IO9	I/O	GPIO9, SD_DATA2, SPIHD, HS1_DATA2, U1RXD
12	RXD0	I/O	
Espressif Systems			Release v4.0.3
			GPIO3, U0RXD (见说明 3), CLK_OUT2





## Header J3

No.	Name	Type	Function
1	FLASH_CS (FCS)	I/O	GPIO16, HS1_DATA4 (见说明 1), U2RXD, EMAC_CLK_OUT
2	FLASH_SD0 (FSD0)	I/O	GPIO17, HS1_DATA5 (见说明 1), U2TXD, EMAC_CLK_OUT_180
3	FLASH_SD2 (FSD2)	I/O	GPIO11, SD_CMD, SPICS0, HS1_CMD (见说明 1), U1RTS
4	SENSOR_VP (FSVP)	I	GPIO36, ADC1_CH0, RTC_GPIO0
5	SENSOR_VN (FSVN)	I	GPIO39, ADC1_CH3, RTC_GPIO3
6	IO25	I/O	GPIO25, DAC_1, ADC2_CH8, RTC_GPIO6, EMAC_RXD0
7	IO26	I/O	GPIO26, DAC_2, ADC2_CH9, RTC_GPIO7, EMAC_RXD1
8	IO32	I/O	32K_XP (见说明 2a) , ADC1_CH4, TOUCH9, RTC_GPIO9
9	IO33	I/O	32K_XN (见说明 2b) , ADC1_CH5, TOUCH8, RTC_GPIO8
10	IO27	I/O	GPIO27, ADC2_CH7, TOUCH7, RTC_GPIO17 EMAC_RX_DV
Espressif Systems	IO14	40 I/O	Release v4.0.3 ADC2_CH6, TOUCH6, RTC_GPIO16, MTMS, HSPICLK

有关上表的说明：

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 3.3V。因此，strapping 管脚 MTDI 在模组上电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。

### 应用程序开发

ESP32-PICO-KIT 上电前，请首先确认开发板完好无损。

之后，请前往[快速入门](#)中的[详细安装步骤](#)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" )。

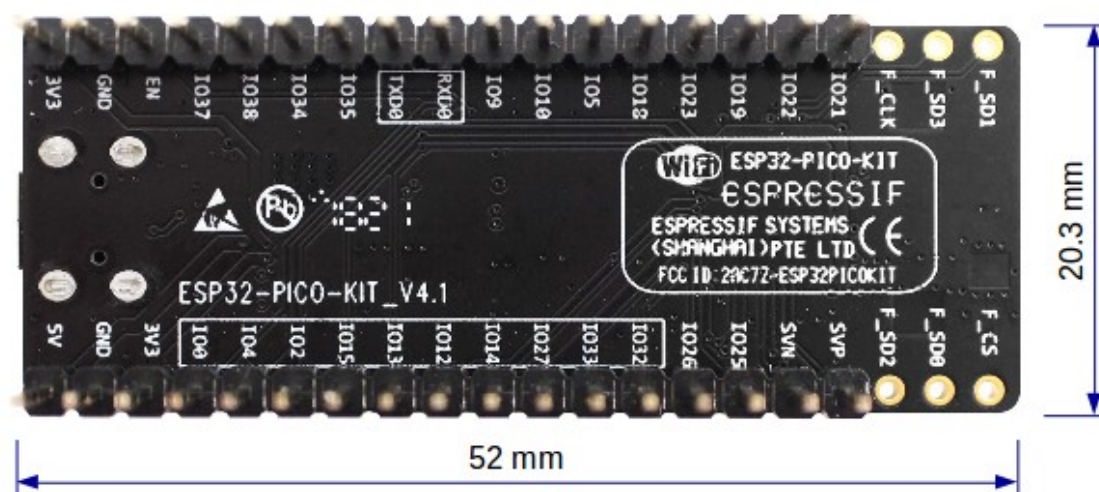


图 13: ESP32-PICO-KIT 尺寸图-背面

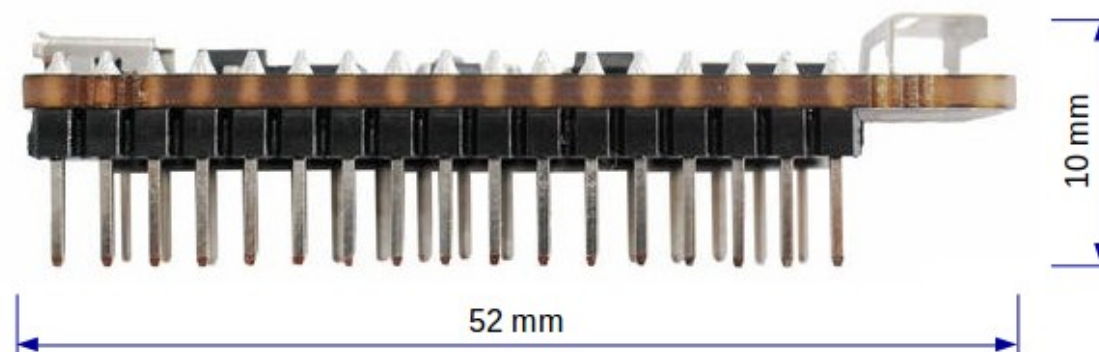


图 14: ESP32-PICO-KIT 尺寸图-侧面

有关开发板的物理结构细节，请见下方参考设计。

## 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》 \(PDF\)](#)
- [ESP32 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：[ESP32 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 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-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。

主要组件	基本介绍
ESP32-PICO-D4	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。
LDO	5V-to-3.3V 低压差稳压器。
USB-to-UART 桥接器	单芯片 USB-to-UART 桥接器，可提供高达 1 Mbps 的传输速率。
Micro USB 端口	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。
Power On LED	开发板通电后，该红色指示灯将亮起。
I/O	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。
BOOT	下载按键。按下 <b>Boot</b> 键并保持，同时按一下 <b>EN</b> 键（此时不要松开 <b>Boot</b> 键）进入“固件下载”模式，通过串口下载固件。
EN	复位按键。

**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。

之后，请前往[快速入门](#) 中的[详细安装步骤](#) 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。

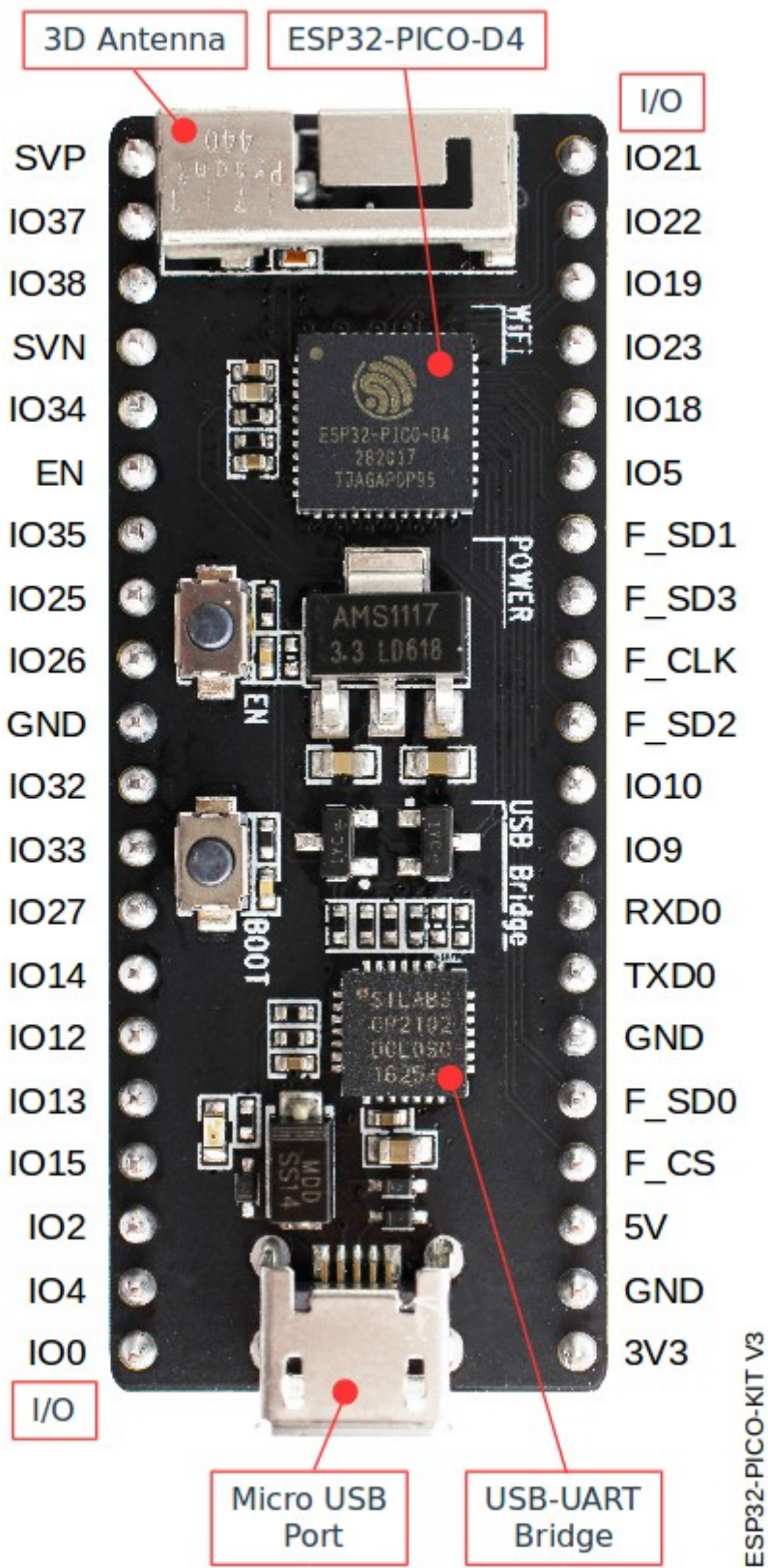


图 15: ESP32-PICO-KIT V3 开发板布局

## 相关文档

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

## 1.3.4 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 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-Ethernet-Kit 是一款来自 [乐鑫](#) 的开发板，由以太网子板 (A 板) 和 PoE 子板 (B 板) 两部分组成。其中 [以太网子板 \(A 板\)](#) 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 FT2232HL 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 调试器。

### 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。

**以太网子板 (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。

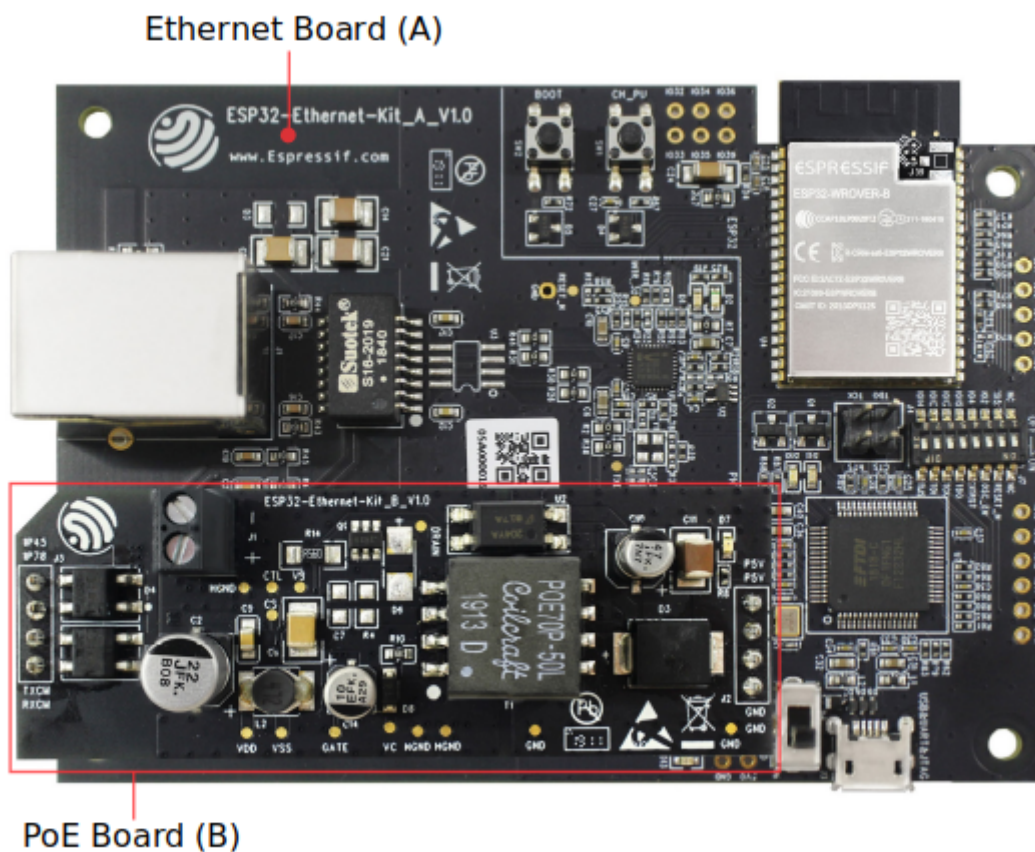


图 16: ESP32-Ethernet-Kit V1.0

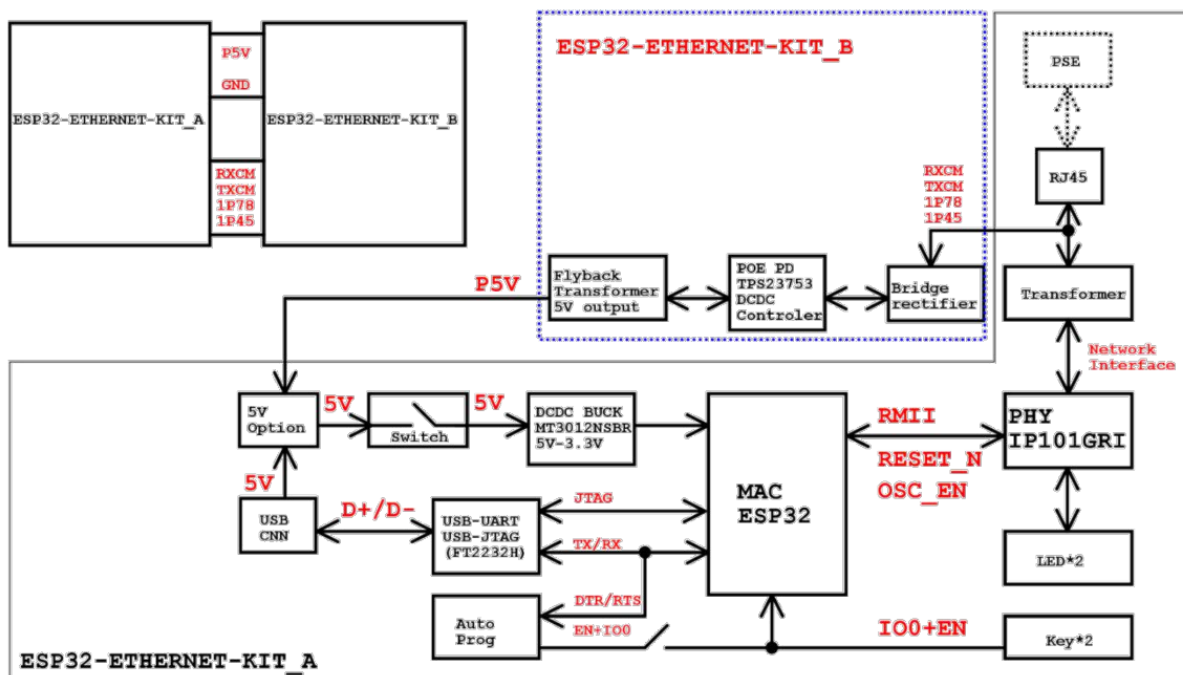


图 17: ESP32-Ethernet-Kit 功能框图 (点击放大)

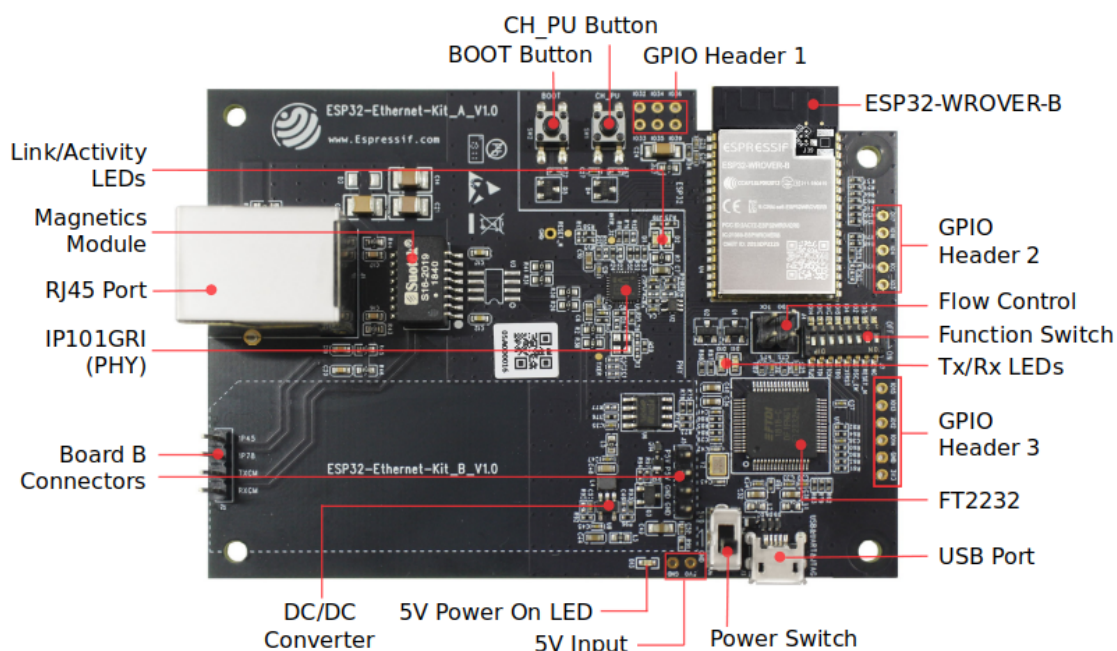


图 18: ESP32-Ethernet-Kit - 以太网子板 (A 板) 布局 (点击放大)

主要组件	基本介绍
ESP32-WROVER-B 模组	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。
GPIO Header 2	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 <a href="#">GPIO Header 2</a> 。
流控	跳线帽，可接入开发板信号。具体介绍，请见 <a href="#">流控</a> 。
功能选择开关	DIP 开关，可配置 ESP32 部分 GPIO 的功能。具体介绍，请见 <a href="#">功能选择开关</a> 。
Tx/Rx LED	2 个 LED，可显示 UART 传输的状态。
GPIO Header 3	可连接至 ESP32 的部分 GPIO，根据 <a href="#">功能选择开关</a> 的位置有不同功能。
FT2232	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，与 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，便利开发人员的应用开发与调试。见 <a href="#">ESP32-Ethernet-Kit V1.0 以太网子板 (A 板) 原理图</a> 。
USB 端口	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。
电源开关	电源开关。拨向 <b>Boot</b> 按键一侧，开发板上电；拨离 <b>Boot</b> 按键一侧，开发板掉电。
5V Input	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。
5V Power On LED	当开发板通电后（USB 或外部 5V 供电），该红色指示灯将亮起。
DC/DC 转换器	直流 5 V 转 3.3 V，输出电流高达 2 A。
IP101GRI (PHY)	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 <b>媒体独</b>

**PoE 子板 (B 板)** PoE 子板通过以太网电缆传输电能 (PoE)，为以太网子板 (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
- 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子板 (A 板) 上的 **RJ45 端口** 连接至 PoE 的交换机。以太网子板 (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 19: ESP32-Ethernet-Kit - PoE 子板 (B 板) 布局 (点击放大)

主要组件	基本介绍
A 板连接器	1 个 4 针排母，用于将 B 板连接至 <a href="#">以太网子板 (A 板)</a> 。
外部电源终端	PoE 子板 (B 板) 备用电源。

## 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。

DIP SW	GPIO 管脚	管脚功能 (DIP SW 开启状态)
1.	GPIO14	连接至 FT2232, 提供 JTAG 功能
2.	GPIO12	连接至 FT2232, 提供 JTAG 功能
3.	GPIO13	连接至 FT2232, 提供 JTAG 功能
4.	GPIO15	连接至 FT2232, 提供 JTAG 功能
5.	GPIO4	连接至 FT2232, 提供 JTAG 功能
6.	GPIO2	连接至板上 25 MHz 晶振
7.	GPIO5	连接至 IP101GRI 的 RESET_N 输入
8.	n/a	

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。

.	信号	备注
1.	MTDO	GPIO13，见功能选择开关。
2.	MTCK	GPIO15，见功能选择开关。
3.	RTS	FT2232 的 RTS 信号
4.	CTS	FT2232 的 CTS 信号

## 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 管脚 (MAC)	IP101GRI (PHY)
<i>RMII</i> 接口		
1.	GPIO21	TX_EN
2.	GPIO19	TXD[0]
3.	GPIO22	TXD[1]
4.	GPIO25	RXD[0]
5.	GPIO26	RXD[1]
6.	GPIO27	CRS_DV
7.	GPIO0	REF_CLK
串行管理接口		
8	GPIO23	MDC
9	GPIO18	MDIO
<i>PHY</i> 复位		
10	GPIO5	Reset_N

**注解：**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
1.	GPIO32
2.	GPIO33
3.	GPIO34
4.	GPIO35
5.	GPIO36
6.	GPIO39

**GPIO Header 2** 本连接器包括具有特定 RMI 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。

.	ESP32 管脚	RMI 功能	说明
1.	GPIO17	EMAC_CLK_180	见说明 1。
2.	GPIO16	EMAC_CLK_OUT	见说明 1。
3.	GPIO4	EMAC_TX_ER	
4.	GPIO2	n/a	见说明 2。
5.	GPIO5	EMAC_RX_CLK	见说明 2。

**注解:**

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 SPIRAM 的模组, 比如 ESP32-WROOM-32D 或 ESP32-SOLO-1。
2. 具体功能取决与[功能选择开关](#)的设置。

**GPIO Header 3** 本连接器中 GPIO 的功能取决与[功能选择开关](#)的设置。

.	ESP32 管脚
1.	GPIO15
2.	GPIO13
3.	GPIO12
4.	GPIO14
5.	GND
6.	3V3

### GPIO 管脚分配

ESP32-WROVER-B	IP101GRI	UART	JTAG	GPIO	说明
S_VP				IO36	
S_VN				IO39	
IO34				IO34	
IO35				IO35	
IO32				IO32	
IO33				IO33	
IO25	RXD[0]				
IO26	RXD[1]				
IO27	CRS_DV				
IO14			TMS	IO14	
IO12			TDI	IO12	
IO13		RTS	TCK	IO13	
IO15		CTS	TDO	IO15	
IO2				IO2	见下方说明 1 和说明 3。
IO0	REF_CLK				见下方说明 2 和说明 3。
IO4			nTRST	IO4	
IO16				IO16 (NC)	见下方说明 4。
IO17				IO17 (NC)	见下方说明 4。
IO5	Reset_N			IO5	
IO18	MDIO				
IO19	TXD[0]				
IO21	TX_EN				
RXD0		RXD			
TXD0		TXD			
IO22	TXD[1]				
IO23	MDC				

#### 注解:

- GPIO2 用于使能 PHY 的外部振荡器。
- 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。
- 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，请将以太网子板 (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条线帽，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 数据线将以太网子板 (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 中的详细安装步骤 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，则请参考详细安装步骤 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/ethernet 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 以太网子板 \(A 板\) 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](#)
- [ESP32 H/W 硬件参考](#)

## 1.4 详细安装步骤

请根据下方详细步骤，完成安装过程。

### 1.4.1 设置开发环境

- [Windows、Linux 和 macOS 的第一步：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](#)

## 1.4.2 创建您的第一个工程

- 第五步：开始创建工程
- 第六步：连接设备
- 第七步：配置
- 第八步：编译工程
- 第九步：烧录到设备
- 第十步：监视器

## 1.5 第一步：安装准备

在正式开始创建工程前，请先完成工具的安装，具体步骤见下：

### 1.5.1 Windows 平台工具链的标准设置

---

**注解：**目前，基于 CMake 的构建系统仅支持 64 位 Windows 版本。32 位 Windows 版本的用户可根据 [传统 GNU Make 构建系统](#) 中的介绍进行操作。

---

#### 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、menuconfig 工具、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](#) 或其他支持 CMake 的图形化工具 IDE。

---

**注解：**较早 ESP-IDF 版本使用 [传统 GNU Make 编译系统](#) 和 [MSYS2](#) Unix 兼容环境。但如今已非必需，用户可直接通过 Windows 命令提示符使用 ESP-IDF。

---

#### ESP-IDF 工具安装器

要安装 ESP-IDF 必备工具，最简易的方式是下载 ESP-IDF 工具安装器，地址如下：

<https://dl.espressif.com/dl/esp-idf-tools-setup-2.3.exe>

本安装器可为您安装所需的交叉编译器、OpenOCD、cmake 和 Ninja 编译工具，以及一款 [mconf-idf](#) 配置工具。此外，本安装器还可在有需要时下载、运行 Python 3.7 和 [Git For Windows](#) 的安装器。

本安装器还可用于下载任意 ESP-IDF 发布版本。

#### 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。

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（比如 PATH，IDF\_PATH 等）。此外，您还可以通过 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 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（比如您需要不同的 ESP-IDF 版本）需要使用快捷方式，您可以：

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“当前路径”指定为您希望使用的 ESP-IDF 路径。

2. 运行 `cmd.exe`，并更新至您希望使用的 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。

## 后续步骤

当 ESP-IDF 工具安装器安装完成后，则开发环境设置也到此结束。后续开发步骤，请前往[第五步：开始创建工程](#)查看。

## 相关文档

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-IDF 工具

**运行 `install.bat` 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
```

该命令可下载安装 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。该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`。

**运行 `export.bat` 将 ESP-IDF 工具添加至路径** 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 ESP-IDF 版本可能并未创建命令提示符快捷方式，此时您可以根据下方步骤将 ESP-IDF 工具添加至 `PATH`。

首先，请打开需要使用 ESP-IDF 的命令提示符窗口，切换至 ESP-IDF 的安装路径，然后执行 `export.bat`：

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

运行完成后，您就可以通过命令提示符使用 ESP-IDF 工具了。

## 1.5.2 Linux 平台工具链的标准设置

### 安装准备

编译 ESP-IDF 需要以下软件包：

- CentOS 7:

```
sudo yum -y update && sudo yum install git wget flex bison gperf python3-
↳python3-pip python3-setuptools cmake ninja-build ccache
```

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

- Ubuntu and Debian:

```
sudo apt-get install git wget flex bison gperf python3 python3-pip python3-
↳setuptools cmake ninja-build ccache libffi-dev libssl-dev
```

- Arch:

```
sudo pacman -S --needed gcc git make ncurses flex bison gperf python-pip cmake_
↪ninja ccache
```

**注解：**使用 ESP-IDF 需要 CMake 3.5 或以上版本。较早版本的 Linux 可能需要升级才能向后移植仓库，或安装“cmake3”软件包，而不是安装“cmake”。

## 其他提示

**权限问题 /dev/ttyUSB0** 使用某些 Linux 版本向 ESP32 烧写固件时，可能会出现 Failed to open port /dev/ttyUSB0 错误消息。此时，可以将当前用户增加至:ref:‘Linux Dialout 组 <linux-dialout-group>’。

**Arch Linux 用户** 在 Arch Linux 中运行预编译 gdb (xtensa-esp32-elf-gdb) 需要 ncurses 5，但 Arch 会使用 ncurses 6。

不过，AUR 中有针对原生和 lib32 配置的向下兼容库：

- <https://aur.archlinux.org/packages/ncurses5-compat-libs/>
- <https://aur.archlinux.org/packages/lib32-ncurses5-compat-libs/>

在安装这些软件包之前，您可能需要将作者的公钥添加到您的密钥环中，具体参考上方的“注释”部分。此外，您也可以使用 crosstool-NG 编译一个链接到 ncurses 6 的 gdb。

## 修复 Ubuntu 16.04 损坏的 pip

python3-pip 包可能已损坏无法升级。需使用脚本 [get-pip.py](#) 手动删除并安装该包：

```
apt remove python3-pip python3-virtualenv; rm -r ~/.local
rm -r ~/.espressif/python_env && python get-pip.py
```

## 停用 Python 2

Python 2 已经 **结束生命周期**，ESP-IDF 很快将不再支持 Python 2。请安装 Python 3.6 或以上版本。可参考上面列出的目前主流 Linux 发行版的安装说明。

## 后续步骤

继续设置开发环境，请前往 [第二步：获取 ESP-IDF](#) 章节。

## 1.5.3 MacOS 平台工具链的标准设置

### 安装准备

ESP-IDF 将使用 Mac OS 上默认安装的 Python 版本。

- 安装 pip:

```
sudo easy_install pip
```

- 安装 CMake 和 Ninja 编译工具：
  - 如果有安装 [HomeBrew](#)，则可通过运行以下指令完成安装：

```
brew install cmake ninja
```

- 如果有安装 [MacPorts](#)，则可通过运行以下指令完成安装:

```
sudo port install cmake ninja
```

- 若以上均不适用，请访问 [CMake](#) 和 [Ninja](#) 主页，查询有关 Mac OS 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。

---

**注解：** 如您在上述任何步骤中遇到以下错误:

```
``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，具体可运行 `xcode-select --install`。

---

**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。这时需要您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
```

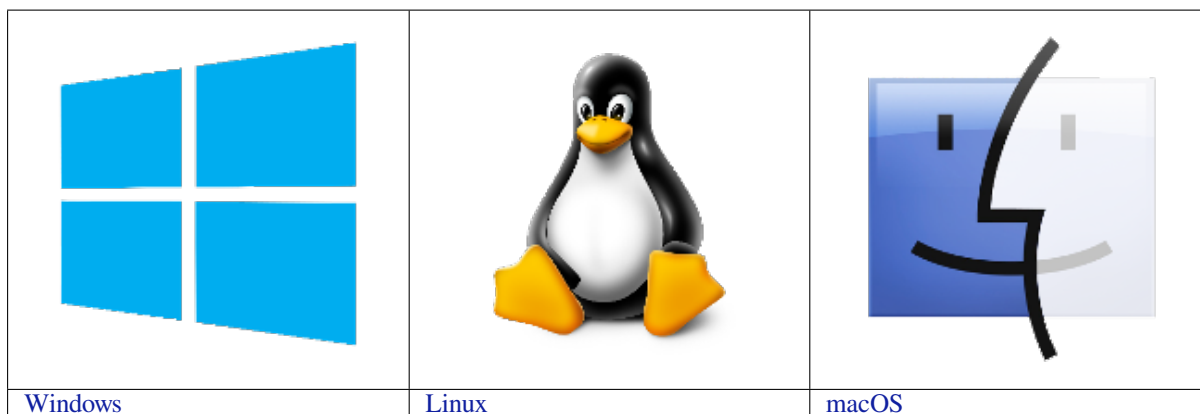
## 停用 Python 2

Python 2 已经 [结束生命周期](#)，ESP-IDF 很快将不再支持 Python 2。请安装 Python 3.6 或以上版本。可参考上面列出的 macOS 安装说明。

## 后续步骤

继续设置开发环境，请前往:[ref:get-started-get-esp-idf](#)。





## 1.6 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。

---

**注解：** 在本文档中，Linux 和 macOS 操作系统中 ESP-IDF 的默认安装路径为 `~/esp`；Windows 操作系统的默认路径为 `%userprofile%\esp`。您也可以将 ESP-IDF 安装在任何其他路径下，但请注意在使用命令行时进行相应替换。注意，ESP-IDF 不支持带有空格的路径。

---

### 1.6.1 Linux 和 macOS 操作系统

打开终端后运行以下命令：

```
cd ~/esp
git clone -b v4.0.3 --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。

### 1.6.2 Windows 操作系统

除了能安装必要工具外，第一步中介绍的 [ESP-IDF 工具安装器](#) 也能同时下载 ESP-IDF 本地副本。

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

除了使用 ESP-IDF 工具安装器，您也可以参考 [指南](#) 手动下载 ESP-IDF。

## 1.7 第三步：设置工具

除了 ESP-IDF 本身，您还需要安装 ESP-IDF 使用的各种工具，比如编译器、调试器、Python 包等。

### 1.7.1 Windows 操作系统

请根据第一步中对 Windows ([ESP-IDF 工具安装器](#)) 的介绍，安装所有必需工具。

除了使用 ESP-IDF 工具安装器，您可以通过 `** 命令提示符 **` 窗口手动安装这些工具。具体步骤见下：

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

## 1.7.2 Linux 和 macOS 操作系统

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

## 1.7.3 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根文件夹中，Linux 和 macOS 系统为 `$HOME/.espressif`，Windows 系统为 `%USERPROFILE%\espressif`。此外，您也可以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 `install.bat/install.sh` 和 `export.bat/export.sh` 脚本时均保持一致。

## 1.8 第四步：设置环境变量

此时，由于您刚刚安装的工具尚未添加至 `PATH` 环境变量，所以无法通过“命令窗口”使用这些工具。因此，必须设置一些环境变量，而这可以通过 ESP-IDF 提供的另一个脚本完成。

### 1.8.1 Windows 操作系统

Windows 安装器 (*ESP-IDF 工具安装器*) 可在“开始”菜单创建一个“ESP-IDF Command Prompt”快捷方式。该快捷方式可以打开命令提示符窗口，并设置所有环境变量。您可以点击该快捷方式，然后继续下一步。

此外，如果您希望在当下命令提示符窗口使用 ESP-IDF，请使用下方代码：

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

### 1.8.2 Linux 和 macOS 操作系统

请在您需要运行 ESP-IDF 的终端窗口运行以下命令：

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

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. 复制并粘贴以下命令到 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。

这里不建议您直接将 `export.sh` 添加到 shell 的配置文件。因为这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 的情况），从而破坏使用虚拟环境的目的，并可能影响其他软件的使用。

## 1.9 第五步：开始创建工程

现在，您可以开始准备开发 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。

将 [get-started/hello\\_world](#) 复制至您本地的 `~/esp` 目录下：

### 1.9.1 Linux 和 macOS 操作系统

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

### 1.9.2 Windows 操作系统

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

ESP-IDF 的 [examples](#) 目录下有一系列示例工程，都可以按照上面的方法进行创建。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。

---

**重要：** ESP-IDF 编译系统不支持带有空格的路径。

---

## 1.10 第六步：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：

- **Windows 操作系统：** COM1 等
- **Linux 操作系统：** 以 `/dev/tty` 开始
- **macOS 操作系统：** 以 `/dev/cu.` 开始

有关如何查看串口名称的详细信息，请见与 [ESP32 建立串口连接](#)。

---

**注解：** 请记住串口名，您会在下面的步骤中用到。

---

## 1.11 第七步：配置

请进入 [第五步：开始创建工程](#) 中提到的 `hello_world` 目录，并运行工程配置工具 `menuconfig`。

### 1.11.1 Linux 和 macOS 操作系统

```
cd ~/esp/hello_world
idf.py menuconfig
```

## 1.11.2 Windows 操作系统

```
cd %userprofile%\esp\hello_world
idf.py menuconfig
```

如果之前的步骤都正确，则会显示下面的菜单：

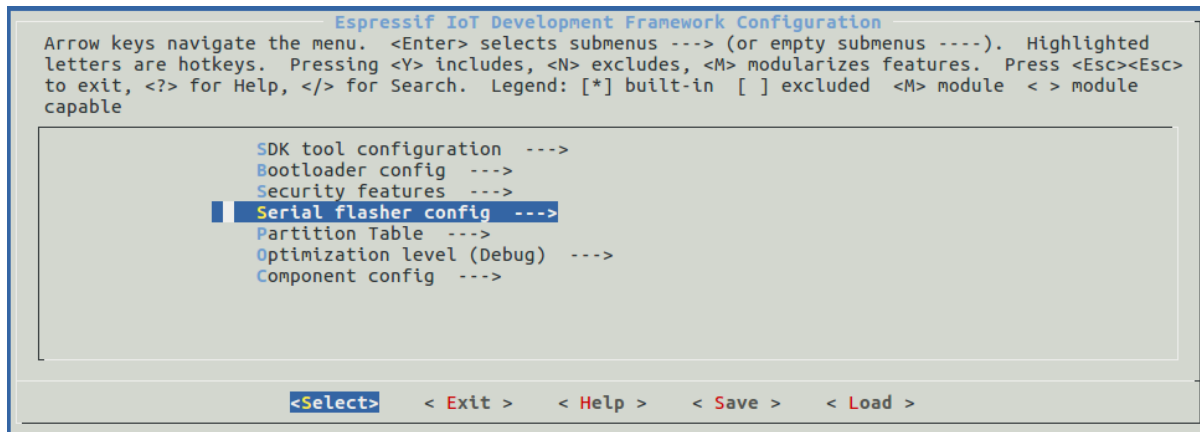


图 20: 工程配置—主窗口

menuconfig 工具的常见操作见下。

- 上下箭头：移动
- 回车：进入子菜单
- ESC 键：返回上级菜单或退出
- 英文问号：调出菜单或选项的帮助菜单，按回车键可退出帮助菜单。
- 空格或 Y 键：选择 [\*] 配置选项；N 键：禁用 [\*] 配置选项
- / 键：寻找配置工程

**注意：** 如果您使用的是 ESP32-DevKitC (板载 ESP32-SOLO-1 模组)，请在烧写示例程序前，前往 menuconfig 中使能单核模式 (`CONFIG_FREERTOS_UNICORE`)。

## 1.12 第八步：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 bootloader、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"...
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)
```

(下页继续)

```
[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.
↳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。

## 1.13 第九步：烧录到设备

请使用以下命令，将刚刚生成的二进制文件烧录至您的 ESP32 开发板：

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

请将 PORT 替换为 ESP32 开发板的串口名称，具体可见[第六步：连接设备](#)。

您还可以将 BAUD 替换为您希望的烧录波特率。默认波特率为 460800。

更多有关 idf.py 参数的详情，请见[idf.py](#)。

**注解：**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。

```
Running esptool.py in directory [...]/esp/hello_world
Executing "python [...]/esp-idf/components/esptool_py/esptool/esptool.py -b 460800.
↳write_flash @flash_project_args"...
esptool.py -b 460800 write_flash --flash_mode dio --flash_size detect --flash_freq.
↳40m 0x1000 bootloader/bootloader.bin 0x8000 partition_table/partition-table.bin.
↳0x10000 hello-world.bin
esptool.py v2.3.1
Connecting....
Detecting chip type... ESP32
Chip is ESP32D0WDQ6 (revision 1)
Features: WiFi, BT, Dual Core
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 460800
Changed.
Configuring flash size...
Auto-detected Flash size: 4MB
Flash params set to 0x0220
Compressed 22992 bytes to 13019...
Wrote 22992 bytes (13019 compressed) at 0x00001000 in 0.3 seconds (effective 558.9.
↳kbit/s)...
Hash of data verified.
Compressed 3072 bytes to 82...
Wrote 3072 bytes (82 compressed) at 0x00008000 in 0.0 seconds (effective 5789.3.
↳kbit/s)...
Hash of data verified.
Compressed 136672 bytes to 67544...
Wrote 136672 bytes (67544 compressed) at 0x00010000 in 1.9 seconds (effective 567.
↳5 kbit/s)...
Hash of data verified.
```

(下页继续)

```
Leaving...
Hard resetting via RTS pin...
```

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。

## 1.14 第十步：监视器

您可以使用 `idf.py -p PORT monitor` 命令，监视“hello\_world”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 /dev/ttyUSB0 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 /dev/ttyUSB0 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)
ets 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...
I (211) cpu_start: Starting scheduler on APP CPU.
Restarting in 9 seconds...
Restarting in 8 seconds...
Restarting in 7 seconds...
```

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发生错误，或打印信息全是乱码（见下），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。

```
e000)(Xn@0y.!00(0PW+)00Hn9a~/90!0t500P0~0k00e0ea050jA
~zY00Y(10,1 00 e000)(Xn@0y.!Dr0zY(0 jpi0|0+z5Ymvp
```

此时，请您：

1. 退出监视器。
2. 打开[menuconfig](#)，
3. 进入 Component config → ESP32-specific → Main XTAL frequency 进行配置，将 `CONFIG_ESP32_XTAL_FREQ_SEL` 设置为 26 MHz。
4. 然后，请重新编译和烧录应用程序。

**注解：**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。

## 1.15 更新 ESP-IDF

乐鑫会不定期推出更新版本的 ESP-IDF，修复 bug 或提供新的特性。因此，您在使用时，也应注意更新您本地的版本。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。

此外，您可以仅更新变更部分。具体方式取决于您使用的 ESP-IDF 版本，请前往 [更新](#) 章节查看。

注意，更新完成后，请执行 `install.sh` (Windows 系统中为 `install.bat`) 脚本，以防新版 ESP-IDF 所需的工具也有更新。具体请参考 [第三步：设置工具](#)。

一旦重新安装好工具，请使用 `export.sh` (Windows 系统中为 `export.bat`) 脚本更新环境，具体请参考 [第四步：设置环境变量](#)。

## 1.16 相关文档

### 1.16.1 与 ESP32 建立串口连接

本章节主要介绍如何创建 ESP32 和 PC 之间的串口连接。

#### 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 转串口芯片（或外部转串口适配器）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：

开发板	USB 驱动	备注
<a href="#">ESP32-DevKitC</a>	<a href="#">CP210x</a>	
<a href="#">ESP32-LyraT</a>	<a href="#">CP210x</a>	
<a href="#">ESP32-LyraTD-MSC</a>	<a href="#">CP210x</a>	
<a href="#">ESP32-PICO-KIT</a>	<a href="#">CP210x</a>	
<a href="#">ESP-WROVER-KIT</a>	<a href="#">FTDI</a>	
<a href="#">ESP32 Demo Board</a>	<a href="#">FTDI</a>	
<a href="#">ESP-Prog</a>	<a href="#">FTDI</a>	Programmer board (w/o ESP32)
<a href="#">ESP32-MeshKit-Sense</a>	n/a	Use with <a href="#">ESP-Prog</a>
<a href="#">ESP32-Sense Kit</a>	n/a	Use with <a href="#">ESP-Prog</a>

- CP210x: [CP210x USB to UART Bridge VCP Drivers](#)
- FTDI: [FTDI Virtual COM Port Drivers](#)

以上驱动仅用于参考。正常情况下，当上述任一 ESP32 开发板与 PC 连接时，打包在操作系统中的驱动程序将会开始自动安装。

#### 查看端口 (Windows 用户)

检查 Windows 设备管理器中的 COM 端口列表。断开 ESP32 与 PC 的连接，然后重连。查看从列表中消失后再次出现的是哪个端口。

以下为 ESP32 DevKitC 和 ESP32 WROVER KIT 串口：

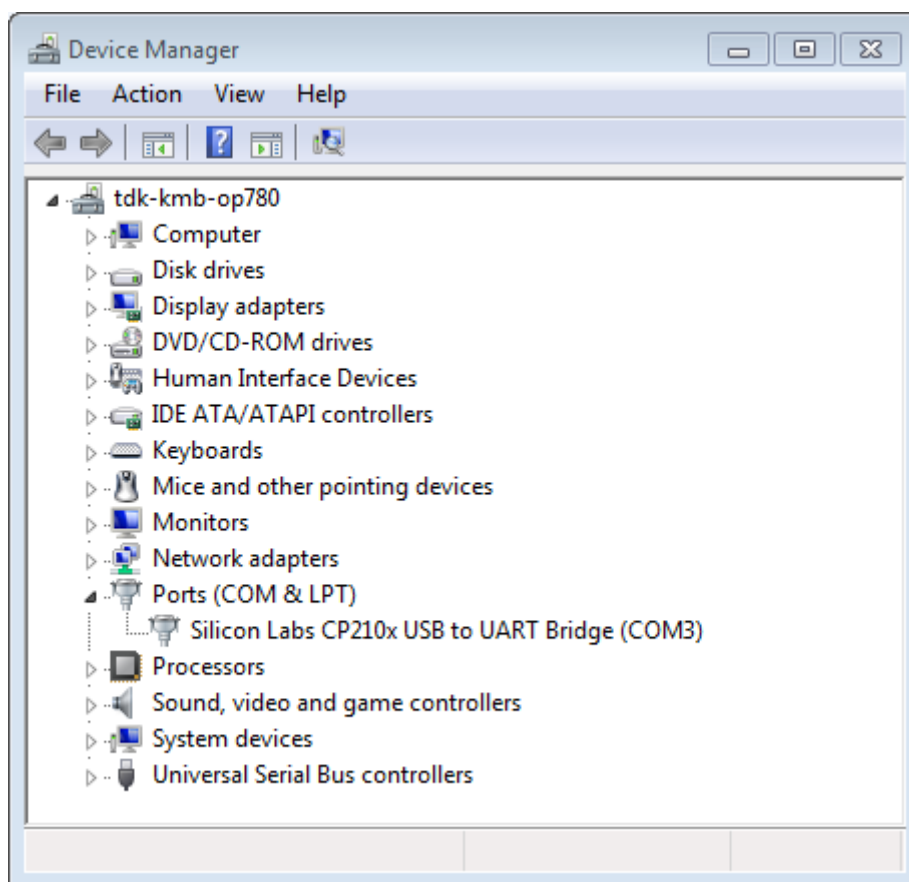


图 21: Windows 设备管理器中 ESP32-DevKitC 的 USB 至 UART 桥



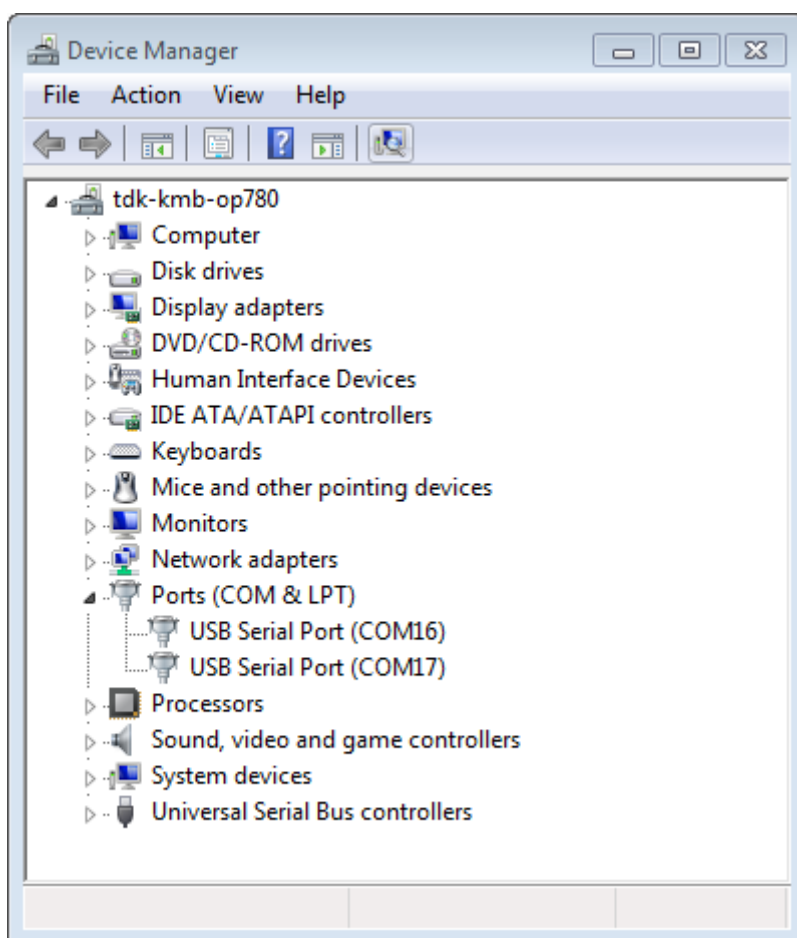


图 22: Windows 设备管理器中 ESP-WROVER-KIT 的两个 USB 串行端口

## 查看端口 (Linux 和 MacOS 用户)

查看 ESP32 开发板 (或外部转串口适配器) 的串口设备名称, 请运行两次以下命令。首先, 断开开发板或适配器, 第一次运行命令; 然后, 连接开发板或适配器, 第二次运行命令。其中, 第二次运行命令后出现的端口即是 ESP32 对应的串口:

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

```
MacOS :: ls /dev/cu.*
```

**注解:** MacOS 用户: 若你没有看到串口, 请检查你是否已按照《入门指南》安装了适用于你特定开发板的 USB/串口驱动程序。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。

## 将用户增加至 Linux 的 dialout 组

当前登录用户应当拥有通过 USB 对串口进行读写的权限。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, 来获取读写权限:

```
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:

```
sudo usermod -a -G uucp $USER
```

请重新登录, 确保串口读写权限可以生效。

## 确认串口连接

现在, 请使用串口终端程序, 验证串口连接是否可用。在本实例中, 我们将使用 [PuTTY SSH Client](#) 进行验证。该工具同时适用于 Windows 和 Linux 操作系统。您也可以使用其他串口程序, 设置通信参数如下。

运行终端, 配置串口: 波特率 = 115200, 数据位 = 8, 停止位 = 1, 奇偶校验 = N。在 Windows 和 Linux 中配置串口和通信参数 (如 115200-8-1-N) 的截图如下。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, 请在终端打开串口, 查看 ESP32 是否有任何打印, 具体打印内容取决于加载至 ESP32 的程序。ESP32 打印示例如下所示:

```
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

...
```

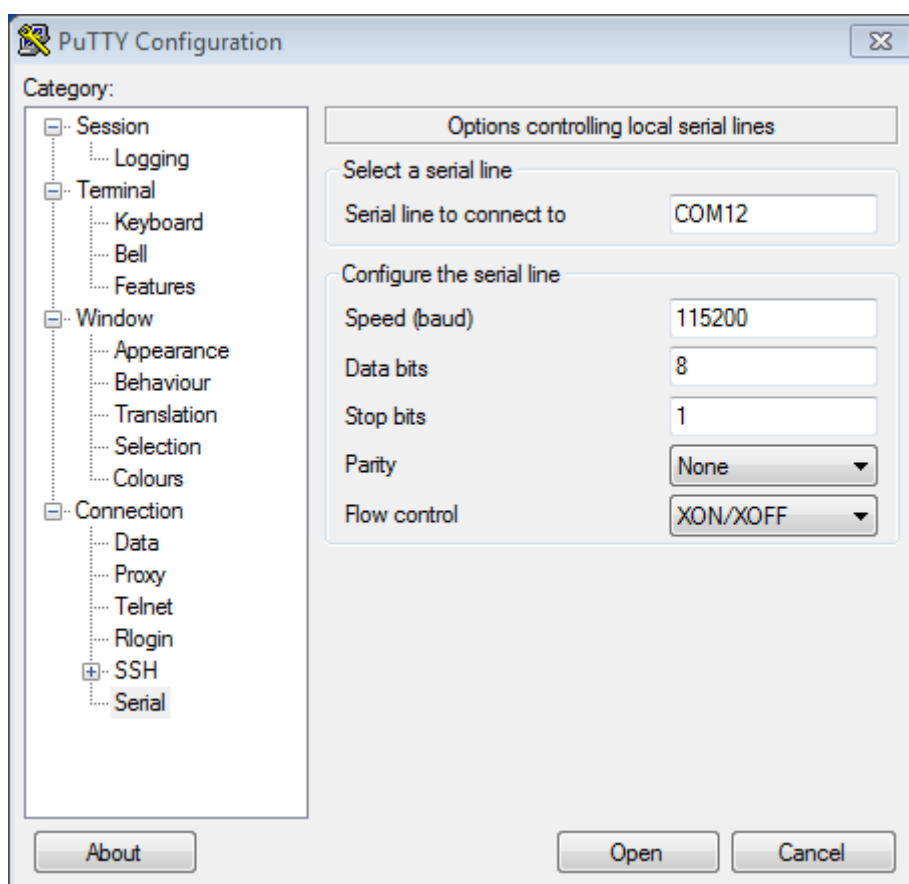


图 23: 在 Windows 操作系统中使用 PuTTY 设置串口通信参数



图 24: 在 Linux 操作系统中使用 PuTTY 设置串口通信参数

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，并最终将应用程序下载到 ESP32。

---

**注解：**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 引脚。该问题仅存在于将 RTS & DTR 引脚直接连接到 EN & GPIO0 引脚上的情况，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，参加 [esptool 文档](#)。

---

---

**注解：**验证完成后，请关闭串口终端。我们将在后续步骤中向 ESP32 下载新的固件，如果未关闭终端，则该应用程序则无法访问串口。

---

如果您是在安装 ESP32 软件的过程中从 [第六步：连接设备](#) 章节跳转至此，请返回 [第七步：配置](#) 章节。

## 1.16.2 Eclipse IDE 创建和烧录指南

ESP-IDF V4.0 将默认采用基于 CMake 的编译系统。

对此，我们还推出了针对 CMake 编译系统的新 ESP-IDF Eclipse 插件。具体操作，请见 <https://github.com/espressif/idf-eclipse-plugin/blob/master/README.md>。

如果您需要 Eclipse IDE 支持传统的 ESP\_IDF Make 构建系统，请见传统 [GNU Make 构建系统入门指南](#) 中的 [Eclipse IDE 创建和烧录指南](#)。

## 1.16.3 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 中调用以下目标函数可以启用此监视器：

- 若使用 CMake 编译系统，则请调用：`idf.py monitor`
- 若使用传统 GNU Make 编译系统，请调用：`make monitor`

### 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。

快捷键	操作	描述
Ctrl+] ]	退出监视器程序	
Ctrl+T	菜单退出键	按下如下给出的任意键之一，并按指示操作。
• Ctrl+T	将菜单字符发送至远程	
• Ctrl+] ]	将 exit 字符发送至远程	
• Ctrl+P	重置目标设备，进入 Bootloader，通过 RTS 线暂停应用程序	重置目标设备，通过 RTS 线（如已连接）进入 Bootloader，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。
• Ctrl+R	通过 RTS 线重置目标设备	重置设备，并通过 RTS 线（如已连接）重新启动应用程序。
• Ctrl+F	编译并烧录此项目	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。
• Ctrl+A (或者 A)	仅编译及烧录应用程序	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。
• Ctrl+Y	停止/恢复在屏幕上打印日志输出	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。
• Ctrl+L	停止/恢复向文件写入日志输出	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）
• Ctrl+H (或者 H)	显示所有快捷键	
• Ctrl+X (或者 X)	退出监视器程序	

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：

```
Guru Meditation Error of type StoreProhibited occurred on core 0. Exception was
↳unhandled.
Register dump:
PC      : 0x400f360d  PS      : 0x00060330  A0      : 0x800dbf56  A1      :
↳0x3ffb7e00
A2      : 0x3ffb136c  A3      : 0x00000005  A4      : 0x00000000  A5      :
↳0x00000000
A6      : 0x00000000  A7      : 0x00000080  A8      : 0x00000000  A9      :
↳0x3ffb7dd0
A10     : 0x00000003  A11     : 0x00060f23  A12     : 0x00060f20  A13     :
↳0x3ffba6d0
```

(下页继续)

(续上页)

```

A14      : 0x00000047  A15      : 0x0000000f  SAR       : 0x00000019  EXCCAUSE:↵
↵0x0000001d
EXCVADDR: 0x00000000  LBEG     : 0x4000c46c  LEND      : 0x4000c477  LCOUNT  :↵
↵0x00000000

Backtrace: 0x400f360d:0x3ffb7e00 0x400dbf56:0x3ffb7e20 0x400dbf5e:0x3ffb7e40↵
↵0x400dbf82:0x3ffb7e60 0x400d071d:0x3ffb7e90

```

IDF 监视器为寄存器转储补充如下信息:

```

Guru Meditation Error of type StoreProhibited occurred on core 0. Exception was↵
↵unhandled.
Register dump:
PC       : 0x400f360d  PS       : 0x00060330  A0       : 0x800dbf56  A1       :↵
↵0x3ffb7e00
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       : 0x3ffb136c  A3       : 0x00000005  A4       : 0x00000000  A5       :↵
↵0x00000000
A6       : 0x00000000  A7       : 0x00000080  A8       : 0x00000000  A9       :↵
↵0x3ffb7dd0
A10      : 0x00000003  A11      : 0x00060f23  A12      : 0x00060f20  A13      :↵
↵0x3ffba6d0
A14      : 0x00000047  A15      : 0x0000000f  SAR       : 0x00000019  EXCCAUSE:↵
↵0x0000001d
EXCVADDR: 0x00000000  LBEG     : 0x4000c46c  LEND      : 0x4000c477  LCOUNT  :↵
↵0x00000000

Backtrace: 0x400f360d:0x3ffb7e00 0x400dbf56:0x3ffb7e20 0x400dbf5e:0x3ffb7e40↵
↵0x400dbf82:0x3ffb7e60 0x400d071d:0x3ffb7e90
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:47
0x400dbf5e: dont_crash at /home/gus/esp/32/idf/examples/get-started/hello_world/
↵main/./hello_world_main.c:42
0x400dbf82: app_main at /home/gus/esp/32/idf/examples/get-started/hello_world/main/
↵./hello_world_main.c:33
0x400d071d: main_task at /home/gus/esp/32/idf/components/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
```

**配置 GDBStub 以启用 GDB** 默认情况下, 如果 ESP-IDF 应用程序发生 crash 事件, panic 处理器将在串口上打印相关寄存器和堆栈转储 (类似上述情况), 然后重置开发板。

或者选择配置 panic 处理器以运行 GDBStub, GDBStub 工具可以与 GDB 项目调试器进行通信, 允许读取内存、检查调用堆栈帧和变量等。GDBStub 虽然没有 JTAG 通用, 但不需要使用特殊硬件。

如需启用 GDBStub, 请运行 `idf.py menuconfig` (适用于 CMake 编译系统), 并将 `CONFIG_ESP32_PANIC` 选项设置为 Invoke GDBStub。

在这种情况下, 如果 panic 处理器被触发, 只要 IDF 监视器监控到 GDBStub 已经加载, panic 处理器就会自动暂停串行监控并使用必要的参数运行 GDB。GDB 退出后, 通过 RTS 串口线复位开发板。如果未连接 RTS 串口线, 请按复位键, 手动复位开发板。

IDF 监视器在后台运行如下命令:

```
xtensa-esp32-elf-gdb -ex "set serial baud BAUD" -ex "target remote PORT" -ex_
↳ interrupt build/PROJECT.elf
```

**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。参数默认值为空字符串, 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:W"` 只匹配并打印 `ESP_LOGW("tag1", ...)` 所写的输出, 或者写在较低日志详细度级别的输出, 即 `ESP_LOGE("tag1", ...)`。请勿指定 `<log_level>` 或使用详细级别默认值 `*`。

**注解:** 编译时, 可以使用主日志在 **日志库** 中禁用不需要的输出。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。

应用程序标签不能包含空格、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低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:"`、`"tag1:*"` 和 `"tag1"` 等同。
- 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** 详细度级别或更低级别打印 `tag1`, 在 **Warning** 详细度级别或更低级别打印 `tag2`。
- 规则 `"tag1:I tag2:W tag3:N"` 在本质上等同于上一规则, 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 (或更低级别), 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: 3ffdbf84, 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: 3ffdbf84, prio:23, stack:4096, core=0
```

PRINTF\_FILTER="light\_driver:D esp\_image:N boot:N cpu\_start:N vfs:N wifi:N \*:V" 选项的输出如下:

```
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

- 若在 Windows 环境下，出现“winpty: command not found”错误，请运行 `pacman -S winpty` 进行修复。
- 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` 或 `make` 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。

## 1.16.4 工具链的自定义设置

除了从乐鑫官网（请见 [第三步：设置工具](#)）下载二进制工具链外，您还可以自行编译工具链。

如无特殊需求，建议直接使用我们提供的预编译二进制工具链。不过，您可以在以下情况考虑自行编译工具链：

- 需要定制工具链编译配置
- 需要使用其他 GCC 版本（如 4.8.5）
- 需要破解 `gcc`、`newlib` 或 `libstdc++`
- 有相关兴趣或时间充裕
- 不信任从网站下载的 `bin` 文件

如需自行编译工具链，请查看以下文档：

### 从零开始设置 Windows 环境下的工具链

除了使用 [ESP-IDF 工具安装器](#)，用户也可以手动设置 Windows 环境下的工具链，这也是本文的主要内容。手动安装工具可以更好地控制安装流程，同时也方便高阶用户进行自定义安装。

使用 ESP-IDF 工具安装器对工具链及其他工具进行快速标准设置，请参照 [Windows 平台工具链的标准设置](#)。

---

**注解：** 基于 GNU Make 的构建系统要求 Windows 兼容 MSYS2 Unix，基于 CMake 的构建系统则无此要求。

---

### 获取 ESP-IDF

**注解：** 较早版本 ESP-IDF 使用了 MSYS2 bash 终端命令行。目前，基于 CMake 的编译系统可使用常见的 Windows 命令窗口，即本指南中使用的终端。

请注意，如果您使用基于 bash 的终端或 PowerShell 终端，一些命令语法将与下面描述有所不同。

---

打开命令提示符，后运行以下命令：

```
mkdir %userprofile%\esp
cd %userprofile%\esp
git clone -b v4.0.3 --recursive https://github.com/espressif/esp-idf.git
```

ESP-IDF 将下载至 %userprofile%\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。

**注解：** git clone 命令的 -b v4.0.3 选项告诉 git 从 ESP-IDF 仓库中克隆与此版本的文档对应的分支。

**注解：** 作为备份，还可以从 [Releases page](#) 下载此稳定版本的 zip 文件。不要下载由 GitHub 自动生成的“源代码”的 zip 文件，它们不适用于 ESP-IDF。

**注解：** 在克隆远程仓库时，不要忘记加上 --recursive 选项。否则，请接着运行以下命令，获取所有子模块：

```
cd esp-idf
git submodule update --init
```

## 工具

**CMake** 下载最新发布的 Windows 平台稳定版 [CMake](#)，并运行安装器。

当安装器询问“安装选项”时，选择“Add CMake to the system PATH for all users”（为所有用户的系统路径添加 CMake）或“Add CMake to the system PATH for the current user”（为当前用户的系统路径添加 CMake）。

### Ninja 编译工具

**注解：** 目前，Ninja 仅提供支持 64 位 Windows 版本的 bin 文件。您也可以配合其他编译工具在 32 位 Windows 版本中使用 CMake 和 idf.py，比如 mingw-make。但是目前暂无关于此工具的说明文档。

从 ([下载页面](#)) 下载最新发布的 Windows 平台稳定版 [ninja](#)。

适用于 Windows 平台的 Ninja 下载文件是一个 .zip 文件，包含一个 ninja.exe 文件。您需要将该文件解压到目录，并 [添加到您的路径](#)（或者选择您路径中的已有目录）。

**Python** 下载并运行适用于 Windows 安装器的最新版 [Python](#)。

Python 安装器的“自定义”菜单可为您提供一系列选项，最后一项为“Add python.exe to Path”（添加 python.exe 到路径中）。请将该项更改到“Will be installed”（将会安装）。

Python 安装完成后，从 Windows 开始菜单中打开“命令提示符”窗口，并运行以下命令：

```
pip install --user pyserial
```

**适用于 IDF 的 MConf** 从 [kconfig-frontends releases page](#) 下载配置工具 mconf-idf。此为 mconf 配置工具，可针对 ESP-IDF 进行少量自定义操作。

请将此工具解压到目录，并 [添加到您的路径](#)。

**工具链设置** 下载预编译的 Windows 工具链:

[https://dl.espressif.com/dl/xtensa-esp32-elf-gcc8\\_4\\_0-esp-2020r3-win32.zip](https://dl.espressif.com/dl/xtensa-esp32-elf-gcc8_4_0-esp-2020r3-win32.zip)

解压压缩包文件到 C:\Program Files (或其他位置)。压缩包文件包含一个 xtensa-esp32-elf 目录。

然后, 请将该目录下的 bin 子目录 **添加到您的路径**。例如, C:\Program Files\xtensa-esp32-elf\bin。

---

**注解:** 如果您已安装 MSYS2 环境 (适用 “GNU Make” 编译系统), 则可以跳过下载那一步, 直接添加目录 C:\msys32\opt\xtensa-esp32-elf\bin 到路径, 因为 MSYS2 环境已包含工具链。

---

**添加目录到路径** 在 Windows 环境下, 向 Path 环境变量增加任何新目录, 请:

打开系统 “控制面板”, 找到环境变量对话框 (Windows 10 用户请前往 “高级系统设置”)。

双击 Path 变量 (选择 “用户” 或 “系统路径”, 具体取决于您是否希望其他用户路径中也存在该目录)。在最后数值那里新添 ;<new value>。

**后续步骤** 继续设置开发环境, 请前往 [第三步: 设置工具](#) 章节。

## 从零开始设置 Linux 环境下的工具链

除了从乐鑫官网直接下载已编译好的二进制工具链外, 您还可以按照本文介绍, 从头开始设置自己的工具链。如需快速使用已编译好的二进制工具链, 可回到 [Linux 平台工具链的标准设置](#) 章节。

**安装准备** 编译 ESP-IDF 需要以下软件包:

- CentOS 7:

```
sudo yum -y update && sudo yum install git wget ncurses-devel flex bison gperf
↳python3 python3-pip cmake ninja-build ccache
```

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

- Ubuntu and Debian:

```
sudo apt-get install git wget libncurses-dev flex bison gperf python3 python3-
↳pip python3-setuptools python3-serial python3-cryptography python3-future
↳python3-pyparsing python3-pyelftools cmake ninja-build ccache libffi-dev
↳libssl-dev dfu-util
```

- Arch:

```
sudo pacman -Sy --needed gcc git make ncurses flex bison gperf python-pyserial
↳python-cryptography python-future python-pyparsing python-pyelftools cmake
↳ninja ccache dfu-util
```

---

**注解:** 使用 ESP-IDF 需要 CMake 3.5 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”)。

---

## 从源代码编译工具链

- 安装依赖项:
- CentOS 7:

```
sudo yum install gawk gperf grep gettext ncurses-devel python3 python3-devel_
↳automake bison flex texinfo help2man libtool make
```

- Ubuntu pre-16.04:

```
sudo apt-get install gawk gperf grep gettext libncurses-dev python python-dev_
↳automake bison flex texinfo help2man libtool make
```

- Ubuntu 16.04 或以上

```
sudo apt-get install gawk gperf grep gettext python python-dev automake bison_
↳flex texinfo help2man libtool libtool-bin make
```

- Debian 9:

```
sudo apt-get install gawk gperf grep gettext libncurses-dev python python-dev_
↳automake bison flex texinfo help2man libtool libtool-bin make
```

- Arch:

```
sudo pacman -Sy --needed python-pip
```

创建工作目录，并进入该目录:

```
mkdir -p ~/esp
cd ~/esp
```

下载并编译 crosstool-NG :

```
git clone https://github.com/espressif/crosstool-NG.git
cd crosstool-NG
git checkout esp-2020r3
git submodule update --init
./bootstrap && ./configure --enable-local && make
```

编译工具链:

```
./ct-ng xtensa-esp32-elf
./ct-ng build
chmod -R u+w builds/xtensa-esp32-elf
```

编译得到的工具链会被保存到 ~/esp/crosstool-NG/builds/xtensa-esp32-elf。请按照 [标准设置指南](#) 的介绍，将工具链添加到 PATH。

**停用 Python 2** Python 2 已经 **结束生命周期**，ESP-IDF 很快将不再支持 Python 2。请安装 Python 3.6 或以上版本。可参考上面列出的目前主流 Linux 发行版的安装说明。

**后续步骤** 继续设置开发环境，请前往 [第二步：获取 ESP-IDF](#) 章节。

请前往 [第二步：获取 ESP-IDF](#) 章节继续设置开发环境。

### 从零开始设置 macOS 环境下的工具链

**软件包管理器** 从零开始设置工具链，您需要安装 [MacPorts](#) 或 [homebrew](#) 软件包管理器。或者，您也可以直接[下载预编译的工具链](#)。

MacPorts 需要完整的 XCode 软件，而 homebrew 只需要安装 XCode 命令行工具即可。

请参考[工具链自定义设置](#) 章节，查看可能需要从头开始设置工具链的情况。

## 安装准备

- 安装 pip:

```
sudo easy_install pip
```

- 安装 pyserial:

```
pip install --user pyserial
```

- 安装 CMake 和 Ninja 编译工具:
  - 若有 HomeBrew, 您可以运行:

```
brew install cmake ninja
```

- 若有 MacPorts, 您可以运行:

```
sudo port install cmake ninja
```

## 从源代码编译工具链

- 安装依赖项:
  - 对于 MacPorts:

```
sudo port install gsed gawk binutils gperf grep gettext wget libtool ↵  
↵autoconf automake make
```

- 对于 homebrew:

```
brew install gnu-sed gawk binutils gperftools gettext wget help2man ↵  
↵libtool autoconf automake make
```

创建一个文件系统镜像 (区分大小写) :

```
hdiutil create ~/esp/crosstool.dmg -volname "ctng" -size 10g -fs "Case-sensitive ↵  
↵HFS+"
```

挂载:

```
hdiutil mount ~/esp/crosstool.dmg
```

创建指向您工作目录的符号链接:

```
mkdir -p ~/esp  
ln -s /Volumes/ctng ~/esp/ctng-volume
```

前往新创建的目录

```
cd ~/esp/ctng-volume
```

下载并编译 crosstool-NG:

```
git clone https://github.com/espressif/crosstool-NG.git  
cd crosstool-NG  
git checkout esp-2020r3  
git submodule update --init  
./bootstrap && ./configure --enable-local && make
```

编译工具链:

```
./ct-ng xtensa-esp32-elf  
./ct-ng build  
chmod -R u+w builds/xtensa-esp32-elf
```

编译得到的工具链会被保存到 `~/esp/ctng-volume/crosstool-NG/builds/xtensa-esp32-elf`。使用工具链前，请将 `~/esp/ctng-volume/crosstool-NG/builds/xtensa-esp32-elf/bin` 添加至 `PATH` 环境变量。

**停用 Python 2** Python 2 已经 **结束生命周期**，ESP-IDF 很快将不再支持 Python 2。请安装 Python 3.6 或以上版本。可参考上面列出的 [macOS 安装说明](#)。

**后续步骤** 继续设置开发环境，请前往 [第二步：获取 ESP-IDF](#)。

## 1.16.5 快速入门 (传统 GNU Make)

---

**注解：** Since ESP-IDF V4.0, the default build system is based on CMake. This documentation is for the legacy build system based on GNU Make. Support for this build system may be removed in future major releases.

---

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。

---

**注解：** 这是 ESP-IDF 稳定版本 v4.0.3 的文档，还有其他版本的文档 [ESP-IDF 版本简介](#) 供参考。

---

### 概述

ESP32 SoC 芯片支持以下功能：

- 2.4 GHz Wi-Fi
- 蓝牙 4.2 标准
- 高性能双核
- 超低功耗协处理器
- 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。

### 准备工作

硬件：

- 一款 **ESP32** 开发板
- **USB 数据线** (USB A/Micro USB B)
- PC (Windows、Linux 或 Mac OS)

软件：

- 设置 **工具链**，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；
- 安装 C 语言编程 (工程) 的 **文本编辑器**，例如 [Eclipse](#)。

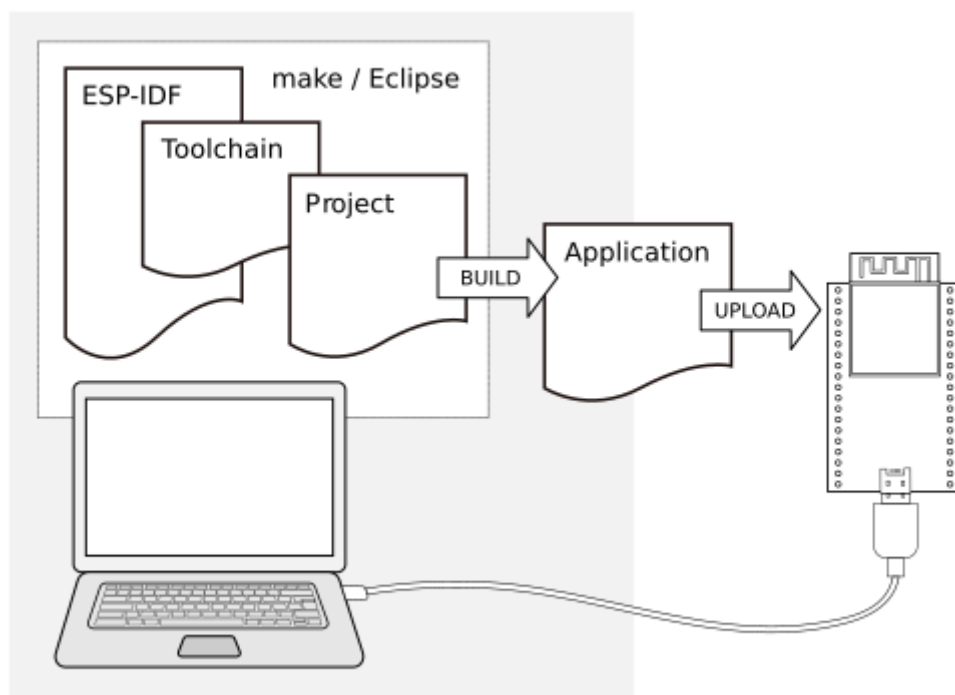


图 25: ESP32 应用程序开发

## 开发板简介

请点击下方连接，了解有关具体开发板的详细信息。

## 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

- 第一步：设置工具链
- 第二步：获取 *ESP-IDF*
- 第三步：设置环境变量
- 第四步：安装 *Python* 软件包

## 创建您的第一个工程

- 第五步：开始创建工程
- 第六步：连接设备
- 第七步：配置
- 第八步：编译和烧录
- 第九步：监视器

## 第一步：设置工具链

工具链指一套用于编译代码和应用程序的程序。

为了加快开发进度，您可以直接使用乐鑫提供的预制工具链。请根据您的操作系统，点击下方对应的链接，并按照链接中的指导进行安装。

**Windows 平台工具链的标准设置 (传统 GNU Make)**

**注解:** Since ESP-IDF V4.0, the default build system is based on CMake. This documentation is for the legacy build system based on GNU Make. Support for this build system may be removed in future major releases.

**引言** Windows 没有内置的“make”环境，因此如果要安装工具链，你需要一个 GNU 兼容环境。我们这里使用 **MSYS2** 来提供该环境。你不需要一直使用这个环境（你可以使用 *Eclipse* 或其它前端工具），但是它是在后台运行的。

**工具链的设置** 快速设置的方法是从 [dl.espressif.com](https://dl.espressif.com) 下载集成在一起的工具链和 MSYS2 压缩文件：

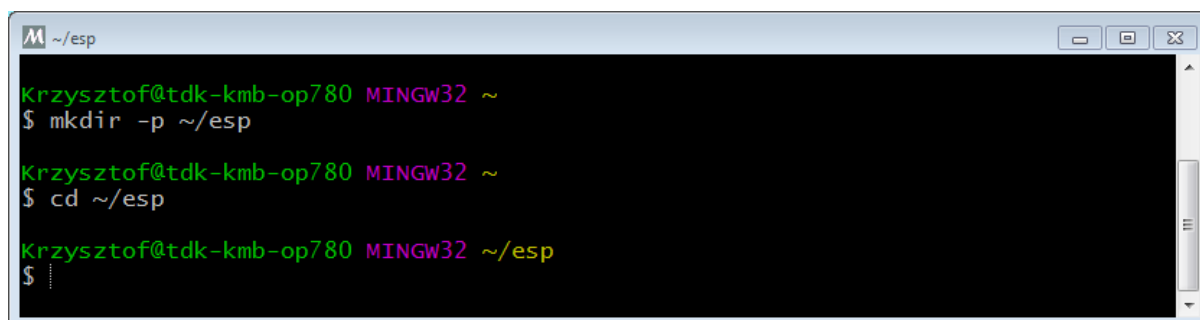
[https://dl.espressif.com/dl/esp32\\_win32\\_msys2\\_environment\\_and\\_esp2020r2\\_toolchain-20200601.zip](https://dl.espressif.com/dl/esp32_win32_msys2_environment_and_esp2020r2_toolchain-20200601.zip)

将 zip 压缩文件解压到 C:\ (或其它路径，这里假设是 C:\)，它会使用预先准备的环境创建一个 msys32 目录。

**检出** 运行 C:\msys32\mingw32.exe 打开一个 MSYS2 的终端窗口。该窗口的环境是一个 bash shell。创建一个 esp 目录作为开发 ESP32 应用的默认地址。运行指令

```
mkdir -p ~/esp
```

输入 `cd ~/esp` 就进入到新创建的目录。如果没有错误信息出现则表明此步骤已完成。



```

M ~/esp
Krzysztof@tdk-kmb-op780 MINGW32 ~
$ mkdir -p ~/esp

Krzysztof@tdk-kmb-op780 MINGW32 ~
$ cd ~/esp

Krzysztof@tdk-kmb-op780 MINGW32 ~/esp
$

```

图 26: MSYS2 终端窗口

后续步骤将会使用这个窗口来为 ESP32 设置开发环境。

**后续步骤** 要继续设置开发环境，请参考 [第二步：获取 ESP-IDF](#) 一节。

**更新环境** 当 IDF 更新时，有时需要新的工具链，或者将新的需求添加到 Windows MSYS2 环境中。要将旧版本的预编译环境中的数据移动到新版本：

- 把旧的 MSYS2 环境（即 C:\msys32）移动/重命名为不同的目录（即 C:\msys32\_old）。
- 按照前文所述步骤下载新的预编译环境。
- 将新的 MSYS2 环境解压缩到 C:\msys32（或其他位置）。
- 找到旧的 C:\msys32\_old\home 目录并把它移到 C:\msys32。
- 如果你不再需要 C:\msys32\_old 可以将它删除。

你可以在系统上拥有独立的不同的 MSYS2 环境，前提是在不同的目录中。

**相关文档**



### Setup Windows Toolchain from Scratch (Legacy GNU Make)

**注解:** Since ESP-IDF V4.0, the default build system is based on CMake. This documentation is for the legacy build system based on GNU Make. Support for this build system may be removed in future major releases.

Setting up the environment gives you some more control over the process, and also provides the information for advanced users to customize the install. The [pre-built environment](#), addressed to less experienced users, has been prepared by following these steps.

To quickly setup the toolchain in standard way, using a prebuilt environment, proceed to section [Windows 平台工具链的标准设置 \(传统 GNU Make\)](#).

**Configure Toolchain & Environment from Scratch** This process involves installing MSYS2, then installing the MSYS2 and Python packages which ESP-IDF uses, and finally downloading and installing the Xtensa toolchain.

- Navigate to the [MSYS2 installer page](#) and download the `msys2-i686-xxxxxxx.exe` installer executable (we only support a 32-bit MSYS environment, it works on both 32-bit and 64-bit Windows.) At time of writing, the latest installer is `msys2-i686-20161025.exe`.
- Run through the installer steps. **Uncheck the “Run MSYS2 32-bit now” checkbox at the end.**
- Once the installer exits, open Start Menu and find “MSYS2 MinGW 32-bit” to run the terminal. (*Why launch this different terminal? MSYS2 has the concept of different kinds of environments. The default “MSYS” environment is Cygwin-like and uses a translation layer for all Windows API calls. We need the “MinGW” environment in order to have a native Python which supports COM ports.*)
- The ESP-IDF repository on github contains a script in the tools directory titled `windows_install_prerequisites.sh`. If you haven't got a local copy of the ESP-IDF yet, that's OK - you can just download that one file in Raw format from here: [tools/windows/windows\\_install\\_prerequisites.sh](#). Save it somewhere on your computer.
- Type the path to the shell script into the MSYS2 terminal window. You can type it as a normal Windows path, but use forward-slashes instead of back-slashes. ie: `C:/Users/myuser/Downloads/windows_install_prerequisites.sh`. You can read the script beforehand to check what it does.
- The `windows_install_prerequisites.sh` script will download and install packages for ESP-IDF support, and the ESP32 toolchain.

### Troubleshooting

- While the install script runs, MSYS may update itself into a state where it can no longer operate. You may see errors like the following:

```
*** fatal error - cygheap base mismatch detected - 0x612E5408/0x612E4408. This_
↪problem is probably due to using incompatible versions of the cygwin DLL.
```

If you see errors like this, close the terminal window entirely (terminating the processes running there) and then re-open a new terminal. Re-run `windows_install_prerequisites.sh` (tip: use the up arrow key to see the last run command). The update process will resume after this step.

- MSYS2 is a “rolling” distribution so running the installer script may install newer packages than what is used in the prebuilt environments. If you see any errors that appear to be related to installing MSYS2 packages, please check the [MSYS2-packages issues list](#) for known issues. If you don't see any relevant issues, please [raise an IDF issue](#).

**MSYS2 Mirrors in China** There are some (unofficial) MSYS2 mirrors inside China, which substantially improves download speeds inside China.

To add these mirrors, edit the following two MSYS2 mirrorlist files before running the setup script. The mirrorlist files can be found in the `/etc/pacman.d` directory (i.e. `c:\msys2\etc\pacman.d`).

Add these lines at the top of `mirrorlist.mingw32`:

```
Server = https://mirrors.ustc.edu.cn/msys2/mingw/i686/
Server = http://mirror.bit.edu.cn/msys2/REPOS/MINGW/i686
```

Add these lines at the top of `mirrorlist.msys`:

```
Server = http://mirrors.ustc.edu.cn/msys2/msys/$arch
Server = http://mirror.bit.edu.cn/msys2/REPOS/MSYS2/$arch
```

**HTTP Proxy** You can enable an HTTP proxy for MSYS and PIP downloads by setting the `http_proxy` variable in the terminal before running the setup script:

```
export http_proxy='http://http.proxy.server:PORT'
```

Or with credentials:

```
export http_proxy='http://user:password@http.proxy.server:PORT'
```

Add this line to `/etc/profile` in the MSYS directory in order to permanently enable the proxy when using MSYS.

**Alternative Setup: Just download a toolchain** If you already have an MSYS2 install or want to do things differently, you can download just the toolchain here:

[https://dl.espressif.com/dl/xtensa-esp32-elf-gcc8\\_4\\_0-esp-2020r3-win32.zip](https://dl.espressif.com/dl/xtensa-esp32-elf-gcc8_4_0-esp-2020r3-win32.zip)

---

**注解:** If you followed instructions *Configure Toolchain & Environment from Scratch*, you already have the toolchain and you won't need this download.

---

---

**重要:** Just having this toolchain is *not enough* to use ESP-IDF on Windows. You will need GNU make, bash, and sed at minimum. The above environments provide all this, plus a host compiler (required for menuconfig support).

---

**Next Steps** To carry on with development environment setup, proceed to section [第二步: 获取 ESP-IDF](#).

**Updating The Environment** When IDF is updated, sometimes new toolchains are required or new system requirements are added to the Windows MSYS2 environment.

Rather than setting up a new environment, you can update an existing Windows environment & toolchain:

- Update IDF to the new version you want to use.
- Run the `tools/windows/windows_install_prerequisites.sh` script inside IDF. This will install any new software packages that weren't previously installed, and download and replace the toolchain with the latest version.

The script to update MSYS2 may also fail with the same errors mentioned under [Troubleshooting](#).

If you need to support multiple IDF versions concurrently, you can have different independent MSYS2 environments in different directories. Alternatively you can download multiple toolchains and unzip these to different directories, then use the PATH environment variable to set which one is the default.

---

### Linux 平台工具链的标准设置 (传统 GNU Make)

**注解:** Since ESP-IDF V4.0, the default build system is based on CMake. This documentation is for the legacy build system based on GNU Make. Support for this build system may be removed in future major releases.

---

**安装前提** 编译 ESP-IDF 需要以下软件包:

- CentOS 7:

```
sudo yum install gcc git wget make ncurses-devel flex bison gperf python_
↳pyserial python-pyelftools
```

- Ubuntu and Debian:

```
sudo apt-get install gcc git wget make libncurses-dev flex bison gperf python_
↳python-pip python-setuptools python-serial python-cryptography python-future_
↳python-pyparsing python-pyelftools libffi-dev libssl-dev
```

- Arch:

```
sudo pacman -S --needed gcc git make ncurses flex bison gperf python-pyserial_
↳python-cryptography python-future python-pyparsing python-pyelftools
```

**注解:** 一些旧的 (2014 年之前) Linux 发行版中使用的 pyserial 版本可能是 2.x, ESP-IDF 并不支持。在这种情况下, 请参考安装依赖的 [Python 软件包](#) 章节, 通过 pip 工具来安装支持的版本。

**工具链的设置** Linux 版的 ESP32 工具链可以从 Espressif 的网站下载:

- 64-bit Linux: [https://dl.espressif.com/dl/xtensa-esp32-elf-gcc8\\_4\\_0-esp-2020r3-linux-amd64.tar.gz](https://dl.espressif.com/dl/xtensa-esp32-elf-gcc8_4_0-esp-2020r3-linux-amd64.tar.gz)
- 32-bit Linux: [https://dl.espressif.com/dl/xtensa-esp32-elf-gcc8\\_4\\_0-esp-2020r3-linux-i686.tar.gz](https://dl.espressif.com/dl/xtensa-esp32-elf-gcc8_4_0-esp-2020r3-linux-i686.tar.gz)

1. 下载完成后, 将它解压到 ~/esp 目录: :

- 64-bit Linux:

```
mkdir -p ~/esp
cd ~/esp
tar -xzf ~/Downloads/xtensa-esp32-elf-gcc8_4_0-esp-2020r3-linux-amd64.tar.
↳gz
```

- 32-bit Linux:

```
mkdir -p ~/esp
cd ~/esp
tar -xzf ~/Downloads/xtensa-esp32-elf-gcc8_4_0-esp-2020r3-linux-i686.tar.gz
```

2. 工具链将会被解压到 ~/esp/xtensa-esp32-elf/ 目录。  
要使用工具链, 你还需要在 ~/.profile 文件中更新环境变量 PATH。要使 xtensa-esp32-elf 在所有的终端会话中都有效, 需要将下面这一行代码添加到你的 ~/.profile 文件中:

```
export PATH="$HOME/esp/xtensa-esp32-elf/bin:$PATH"
```

或者你也可以给上面的命令创建一个别名。这样做的好处是, 你只在需要使用它的时候才获取工具链。将下面这行代码添加到 ~/.profile 文件中即可:

```
alias get_esp32='export PATH="$HOME/esp/xtensa-esp32-elf/bin:$PATH"'
```

然后, 当你需要使用工具链时, 在命令行输入 get\_esp32, 然后工具链会自动添加到你的 PATH 中。

**注解:** 如果将 /bin/bash 设置为登录 shell, 且同时存在 .bash\_profile 和 .profile, 则更新 .bash\_profile。在 CentOS 环境下, alias 需要添加到 .bashrc 文件中。

3. 退出并重新登录以使 .profile 更改生效。运行以下命令来检查 PATH 设置是否正确:

```
printenv PATH
```

检查字符串的开头是否包含类似的工具链路径::

```
$ printenv PATH
/home/user-name/esp/xtensa-esp32-elf/bin:/home/user-name/bin:/home/user-
↪name/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/
↪bin:/usr/games:/usr/local/games:/snap/bin
```

除了 ``/home/user-name``，应该有具体的安装的主路径。

**权限问题 /dev/ttyUSB0** 某些 Linux 版本可能在烧写 ESP32 时会出现 Failed to open port /dev/ttyUSB0 错误消息。可以通过将当前用户添加到拨出组来解决。

**Arch Linux 用户** 在 Arch 中运行预编译的 gdb (xtensa-esp32-elf-gdb) 需要 ncurses 5，但是 Arch 使用的是 ncurses 6。在 AUR 中向下兼容的库文件，可用于本地和 lib32 的配置：

- <https://aur.archlinux.org/packages/ncurses5-compat-libs/>
- <https://aur.archlinux.org/packages/lib32-ncurses5-compat-libs/>

在安装这些软件包之前，你可能需要将作者的公钥添加到你的钥匙圈中，上面链接中的“Comments”部分有所叙述。

或者，你也可以使用 crosstool-NG 编译一个链接 ncurses 6 的 gdb。

**后续步骤** 要继续设置开发环境，请参考**第二步：获取 ESP-IDF** 一节。

### 在 Mac OS 上安装 ESP32 工具链 (传统 GNU Make)

**注解：** Since ESP-IDF V4.0, the default build system is based on CMake. This documentation is for the legacy build system based on GNU Make. Support for this build system may be removed in future major releases.

### 安装准备

- 安装 pip:

```
sudo easy_install pip
```

**注解：** pip 稍后将用于安装必要的 *Python* 软件包。

**安装工具链** Mac OS 版本的 ESP32 工具链可以从以下地址下载：

[https://dl.espressif.com/dl/xtensa-esp32-elf-gcc8\\_4\\_0-esp-2020r3-macos.tar.gz](https://dl.espressif.com/dl/xtensa-esp32-elf-gcc8_4_0-esp-2020r3-macos.tar.gz)

下载压缩文件之后，解压到 ~/esp 目录中：

```
mkdir -p ~/esp
cd ~/esp
tar -xzf ~/Downloads/xtensa-esp32-elf-gcc8_4_0-esp-2020r3-macos.tar.gz
```

工具链将被解压到 ~/esp/xtensa-esp32-elf/ 路径下。

在 ~/.profile 文件中更新 PATH 环境变量以使用工具链。为了使 xtensa-esp32-elf 在各种终端会话中都可用，在 ~/.profile 文件中加上以下指令：

```
export PATH=$HOME/esp/xtensa-esp32-elf/bin:$PATH
```

或者，您可以为上述命令创建一个别名。这样只有执行以下指令时工具链才能被使用。将下面的指令添加到您的 `~/.profile` 文件中：

```
alias get_esp32="export PATH=$HOME/esp/xtensa-esp32-elf/bin:$PATH"
```

当需要使用工具链时，在命令行里输入 `get_esp32`，就可以将工具链添加到 `PATH` 中。

**下一步** 前往 [第二步：获取 ESP-IDF](#) 继续配置开发环境。

## 相关文档

### 从零开始设置 Mac OS 环境下的工具链 (传统 GNU Make)

**注解：** Since ESP-IDF V4.0, the default build system is based on CMake. This documentation is for the legacy build system based on GNU Make. Support for this build system may be removed in future major releases.

**注解：** 安装工具链的标准流程可以通过阅读文档在 [MacOS 上安装 ESP32 工具链](#) 来获得，[工具链的自定义设置](#) 章节会介绍哪些情况下我们必须重新定义工具链。

## 安装必要的工具

- 安装 pip:

```
sudo easy_install pip
```

**注解：** pip 稍后将用于安装必要的 [Python 软件包](#)。

## 从源代码编译工具链

- 安装依赖:
  - 安装 [MacPorts](#) 或者 [homebrew](#) 包管理器。MacPorts 需要安装完整的 XCode 软件，但是 homebrew 只需要安装 XCode 命令行工具即可。
  - 对于 MacPorts:

```
sudo port install gsed gawk binutils gperf grep gettext wget libtool↵
↵autoconf automake
```

- 对于 homebrew:

```
brew install gnu-sed gawk binutils gperftools gettext wget help2man↵
↵libtool autoconf automake
```

创建大小写敏感的文件系统镜像:

```
hdiutil create ~/esp/crosstool.dmg -volname "ctng" -size 10g -fs "Case-sensitive↵
↵HFS+"
```

挂载:

```
hdiutil mount ~/esp/crosstool.dmg
```

创建指向你工作目录的符号链接:

```
mkdir -p ~/esp
ln -s /Volumes/ctng ~/esp/ctng-volume
```

进入新创建的工作目录:

```
cd ~/esp/ctng-volume
```

下载 crosstool-NG 然后编译:

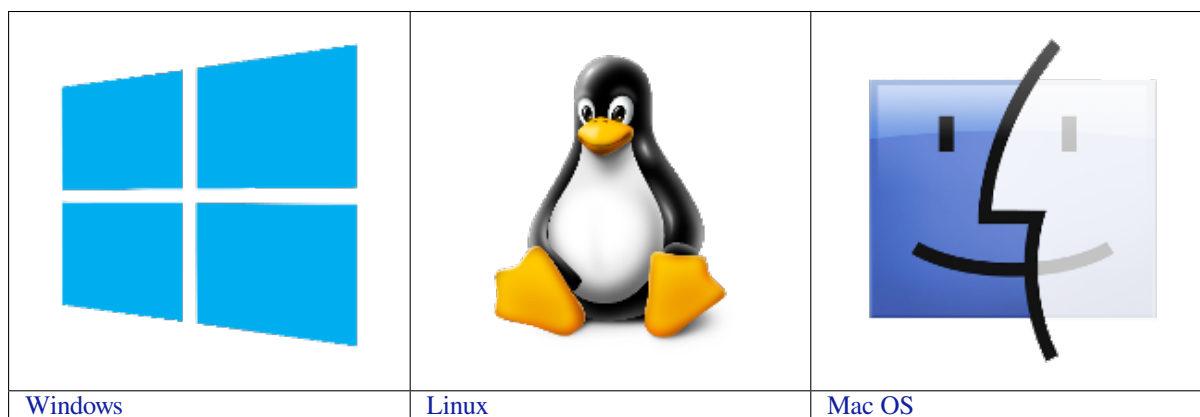
```
git clone https://github.com/espressif/crosstool-NG.git
cd crosstool-NG
git checkout esp-2020r3
git submodule update --init
./bootstrap && ./configure --enable-local && make
```

编译工具链:

```
./ct-ng xtensa-esp32-elf
./ct-ng build
chmod -R u+w builds/xtensa-esp32-elf
```

编译得到的工具链会被保存到 `~/esp/ctng-volume/crosstool-NG/builds/xtensa-esp32-elf`。根据[Mac OS 下设置环境变量的标准方法](#)中的介绍，将工具链添加到 `PATH` 中。

**下一步** 继续设置开发环境，请前往[获取 ESP-IDF](#) 章节。



---

**注解:** 在本文档中，Linux 和 MacOS 操作系统中 ESP-IDF 的默认安装路径为 `~/esp`；Windows 操作系统的默认路径为 `%userprofile%\esp`。您也可以将 ESP-IDF 安装在任何其他路径下，但请注意在使用命令行时进行相应替换。注意，ESP-IDF 不支持带有空格的路径。

---

此外，您也可以根据自身经验和实际需求，对环境进行个性化设置，而非使用预制工具链。此时，请前往[工具链的个性化设置](#) 章节获取更多信息。

## 第二步：获取 ESP-IDF

除了工具链，您还需要供 ESP32 使用的 API（软件库和源代码），具体请见 [ESP-IDF 仓库](#)。

获取本地副本：打开终端，切换到你要存放 ESP-IDF 的工作目录，使用 `git clone` 命令克隆远程仓库。打开终端，后运行以下命令：

```
cd ~/esp
git clone -b v4.0.3 --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。

---

**注解：** `git clone` 命令的 `-b v4.0.3` 选项告诉 `git` 从 ESP-IDF 仓库中克隆与此版本的文档对应的分支。

---

---

**注解：** 作为备份，还可以从 [Releases page](#) 下载此稳定版本的 zip 文件。不要下载由 GitHub 自动生成的“源代码”的 zip 文件，它们不适用于 ESP-IDF。

---

---

**注解：** 在克隆远程仓库时，不要忘记加上 `--recursive` 选项。否则，请接着运行以下命令，获取所有子模块：

```
cd esp-idf
git submodule update --init
```

---

### 第三步：设置环境变量

工具链通过环境变量 `IDF_PATH` 获得 ESP-IDF 的目录。因此，您需要在 PC 中设置该环境变量，否则无法编译工程。

您可以在每次重启会话时手动设置，也可以在用户配置中进行永久设置，具体请前往[在用户配置文件中添加 IDF\\_PATH \(传统 GNU Make\)](#) 章节，查看 [Windows](#)、[Linux](#) 及 [MacOS](#) 操作系统的具体设置方式。

### 第四步：安装 Python 软件包

ESP-IDF 所需 Python 软件包位于 `IDF_PATH/requirements.txt` 中。您可以运行以下命令进行安装：

```
python -m pip install --user -r $IDF_PATH/requirements.txt
```

---

**注解：** 请注意查询您所使用的 Python 解释器的版本（运行命令 `python --version`），并根据查询结果将上方命令中的 `python` 替换为 `python3`、`python3.7`，例如：

```
python3 -m pip install --user -r $IDF_PATH/requirements.txt
```

---

### 第五步：开始创建工程

现在，您可以开始准备开发 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。

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

### Linux 和 MacOS 操作系统

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

## Windows 操作系统

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

ESP-IDF 的 [examples](#) 目录下有一系列示例工程，都可以按照上面的方法进行创建。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。

**重要：** ESP-IDF 编译系统不支持带有空格的路径。

## 第六步：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：

- **Windows 操作系统：** COM1 等
- **Linux 操作系统：** 以 /dev/tty 开始
- **MacOS 操作系统：** 以 /dev/cu. 开始

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 \(传统 GNU Make\)](#)。

**注解：** 请记住串口名，您会在下面的步骤中用到。

## 第七步：配置

请进入 [第五步：开始创建工程](#) 中提到的 `hello_world` 目录，并运行工程配置工具 `menuconfig`。

## Linux 和 MacOS 操作系统

```
cd ~/esp/hello_world
make menuconfig
```

## Windows 操作系统

```
cd %userprofile%\esp\hello_world
make menuconfig
```

如果之前的步骤都正确，则会显示下面的菜单：

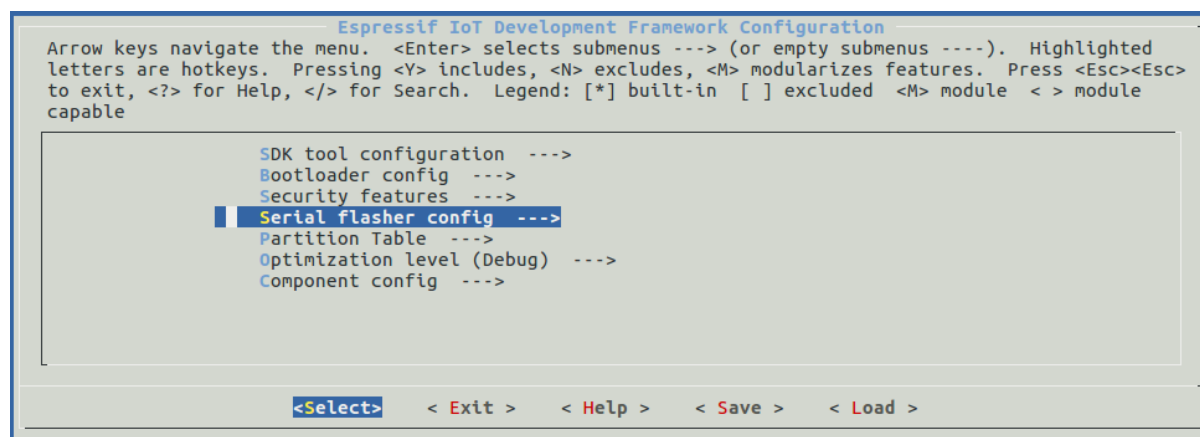


图 27: 工程配置—主窗口



进入菜单后, 选择 `Serial flasher config>Default serial port` 配置串口 (设备将通过该串口加载工程)。按回车键确认选择, 点击 `< Save >` 保存配置, 然后点击 `< Exit >` 退出 `menuconfig`。  
`menuconfig` 工具的常见操作见下。

- 上下箭头: 移动
- 回车: 进入子菜单
- ESC 键: 返回上级菜单或退出
- 英文问号: 调出帮助菜单 (退出帮助菜单, 请按回车键)。
- 空格、Y 键 `` 或 ``N 键: 使能/禁用 [\*] 配置选项
- 英文问号: 调出有关高亮选项的帮助菜单
- / 键: 寻找配置项目

**注意:** 如果您使用的是 ESP32-DevKitC (板载 ESP32-SOLO-1 模组), 请在烧写示例程序前, 前往 `menuconfig` 中使能单核模式 (`CONFIG_FREERTOS_UNICORE`)。

## 第八步: 编译和烧录

请使用以下命令, 编译烧录工程:

```
make flash
```

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 `bootloader`、分区表和应用程序二进制文件。接着, 这些二进制文件将被烧录至 ESP32 开发板。

如果一切顺利, 您可在烧录完成后看到类似下方的打印信息 (代表加载进程)。接着, 开发板将会复位, 应用程序 “`hello_world`” 开始启动。

```
esptool.py v2.0-beta2
Flashing binaries to serial port /dev/ttyUSB0 (app at offset 0x10000)...
esptool.py v2.0-beta2
Connecting.....____
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 921600
Changed.
Attaching SPI flash...
Configuring flash size...
Auto-detected Flash size:4MB
Flash params set to 0x0220
Compressed 11616 bytes to 6695...
Wrote 11616 bytes (6695 compressed) at 0x00001000 in 0.1 seconds (effective 920.5
↪kbit/s)...
Hash of data verified.
Compressed 408096 bytes to 171625...
Wrote 408096 bytes (171625 compressed) at 0x00010000 in 3.9 seconds (effective 847.
↪3 kbit/s)...
Hash of data verified.
Compressed 3072 bytes to 82...
Wrote 3072 bytes (82 compressed) at 0x00008000 in 0.0 seconds (effective 8297.4
↪kbit/s)...
Hash of data verified.

Leaving...
Hard resetting...
```

如果您希望使用 Eclipse IDE, 而非 `make` 编译系统, 请参考 [Eclipse 指南](#)。

## 第九步：监视器

您可以使用 `make monitor` 命令，监视 “hello\_world” 的运行情况。

运行该命令后，[IDF 监视器](#) 应用程序将启动：

```

$ make monitor
MONITOR
--- idf_monitor on /dev/ttyUSB0 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)
ets 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...
I (211) cpu_start:Starting scheduler on APP CPU.
Restarting in 9 seconds...
Restarting in 8 seconds...
Restarting in 7 seconds...

```

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发生错误，或打印信息全是乱码（见下），很有可能是因为您的开发板选用了 26 MHz 晶振，而 ESP-IDF 默认支持大多数开发板使用的 40 MHz 晶振。

```

e000)(Xn@0y.!00(0Pw+)00Hn9a~/90!0t500P0~0k00e0ea050jA
~zY00Y(10,1 00 e000)(Xn@0y.!Dr0zY(0 jpi0|0+z5Ymvp

```

此时，请您：

1. 退出监视器。
2. 打开 [menuconfig](#)，
3. 进入 Component config -> ESP32-specific -> Main XTAL frequency 进行配置，将 `CONFIG_ESP32_XTAL_FREQ_SEL` 设置为 26 MHz。
4. 然后，请重新编译和烧录应用程序。

**注解：**您也可以运行以下命令，一次性执行构建、烧录和监视过程：

```
make 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。

**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。

## 环境变量

用户可以在使用 `make` 命令时 **直接设置**部分环境变量，而无需进入 `make menuconfig` 进行重新配置。这些变量包括：

变量	描述与使用方式
ESPPORT	覆盖 flash 和 monitor 命令使用的串口。 例: <code>make flash ESPPORT=/dev/ttyUSB1,make monitor ESPPORT=COM1</code>
ESPBAUD	覆盖烧录 ESP32 时使用的串口速率。 例: <code>make flash ESPBAUD=9600</code>
MONITORBAUD	覆盖监控时使用的串口速率。 例: <code>make monitor MONITORBAUD=9600</code>

**注解:** 您可导出环境变量 (例: `export ESPPORT=/dev/ttyUSB1`)。在同一会话窗口中, 如果未被同步覆盖, 所有 make 命令均会使用导出的环境变量值。

## 更新 ESP-IDF

乐鑫会不时推出更新版本的 ESP-IDF, 修复 bug 或提出新的特性。因此, 在使用时, 您也应注意更新您本地的版本。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。

如果您希望将 ESP-IDF 克隆到新的路径下, 请务必重新设置 `IDF_PATH`。否则, 工具链将无法找到 ESP-IDF。

此外, 您可以仅更新变更部分。具体方式, 请前往 [更新](#) 章节查看。

## 相关文档

### 在用户配置文件中添加 `IDF_PATH` (传统 GNU Make)

**注解:** Since ESP-IDF V4.0, the default build system is based on CMake. This documentation is for the legacy build system based on GNU Make. Support for this build system may be removed in future major releases.

为了在系统多次重新启动时保留 “`IDF_PATH`” 环境变量的设置, 请按照以下说明将其添加到用户配置文件中。

**Windows** 用户配置文件脚本存放在 `C:/msys32/etc/profile.d/` 目录中。每次打开 `MSYS2` 窗口时, 系统都执行这些脚本。

1. 在 `C:/msys32/etc/profile.d/` 目录下创建一个新的脚本文件。将其命名为 `export_idf_path.sh`。
2. 确定 ESP-IDF 目录的路径。路径与系统配置有关, 例如 `C:\msys32\home\user-name\esp\esp-idf`。
3. 在脚本中加入 `export` 命令, e.g.:

```
export IDF_PATH="C:/msys32/home/user-name/esp/esp-idf"
```

请将原始 Windows 路径中将反斜杠替换为正斜杠。

1. 保存脚本。
2. 关闭 `MSYS2` 窗口并再次打开。输入以下命令检查是否设置了 `IDF_PATH`:

```
printenv IDF_PATH
```

将此前在脚本文件中输入的路径打印出来。

如果您不想在用户配置文件中永久设置 `IDF_PATH`, 则应在打开 `MSYS2` 窗口时手动输入:

```
export IDF_PATH="C:/msys32/home/user-name/esp/esp-idf"
```

如您在安装用于 ESP32 开发的软件时，从 [第三步：设置环境变量](#) 小节跳转到了这里，请返回到 [第五步：开始创建工程](#) 小节。

**Linux and MacOS** 在 `~/.profile` 文件中加入以下指令，创建 `IDF_PATH`：

```
export IDF_PATH=~/esp/esp-idf
```

注销并重新登录以使此更改生效。

---

**注解：** 如果将 `/bin/bash` 已设为登录 shell，并且 `.bash_profile` 和 `.profile` 同时存在，则更新 `.bash_profile`。

---

运行以下命令以确保 `IDF_PATH` 已经设置好：

```
printenv IDF_PATH
```

此前在 `~/.profile` 文件中输入（或者手动设置）的路径应该被打印出来。

如果不想永久设置 `IDF_PATH`，每次重启或者注销时在终端窗口中手动输入：

```
export IDF_PATH=~/esp/esp-idf
```

如果您从 [第三步：设置环境变量](#) 小节跳转到了这里，在安装用于 ESP32 开发的软件时，返回到 [第五步：开始创建工程](#) 小节。

---

### 与 ESP32 创建串口连接 (传统 GNU Make)

**注解：** Since ESP-IDF V4.0, the default build system is based on CMake. This documentation is for the legacy build system based on GNU Make. Support for this build system may be removed in future major releases.

---

本章节介绍如何在 ESP32 和 PC 之间建立串口连接。

**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，确认 ESP32 开发板上的 USB 转串口芯片（或外部串口适配器）型号，在网上搜索驱动程序并进行安装。

以下是乐鑫 ESP32 开发板驱动程序的链接：

- ESP32-PICO-KIT 和 ESP32-DevKitC - [CP210x USB to UART Bridge VCP Drivers](#)
- ESP32-WROVER-KIT 和 ESP32 Demo Board - [FTDI Virtual COM Port Drivers](#)

以上驱动仅用于参考。当您上述 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自动安装。

**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，然后再次显示。

以下为 ESP32 DevKitC 和 ESP32 WROVER KIT 串口：

**在 Linux 和 MacOS 上检查串口** 要查看 ESP32 开发板（或外部串口适配器）的串口设备名称，运行以下命令两次，第一次先拔出开发板或适配器，第二次插入开发板或适配器之后再运行命令，第二次运行指令后出现的端口即是 ESP32 对应的串口：

Linux

```
ls /dev/tty*
```

MacOS

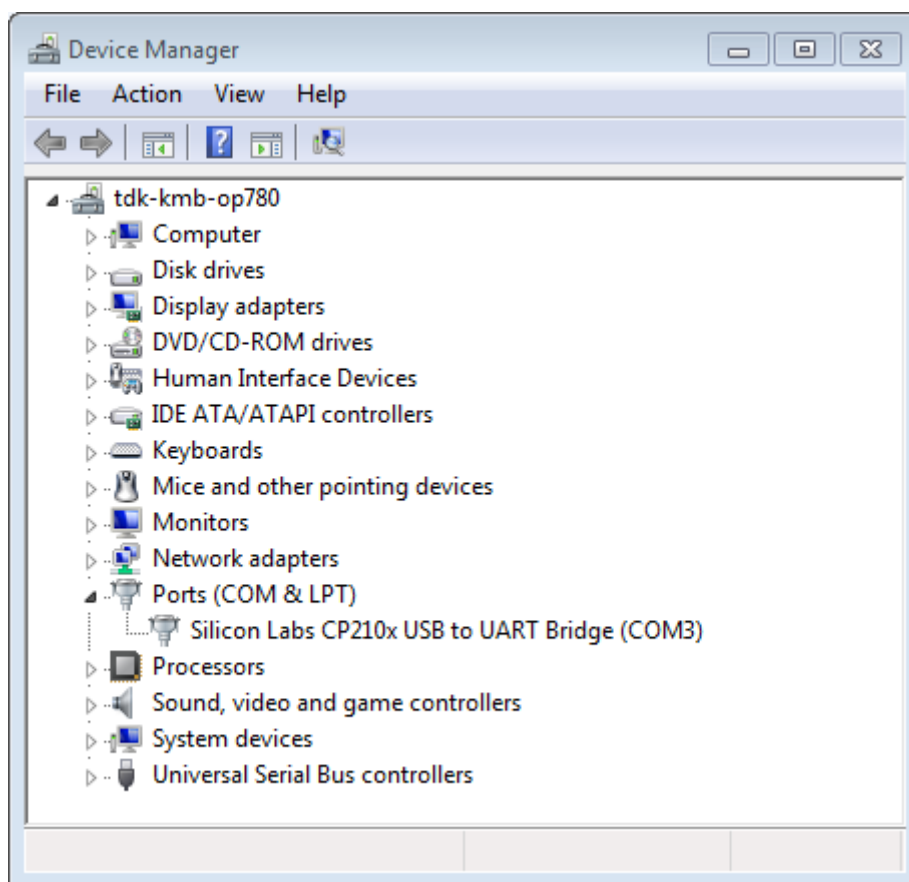


图 28: 设备管理器中 ESP32-DevKitC 的 USB 串口转换器

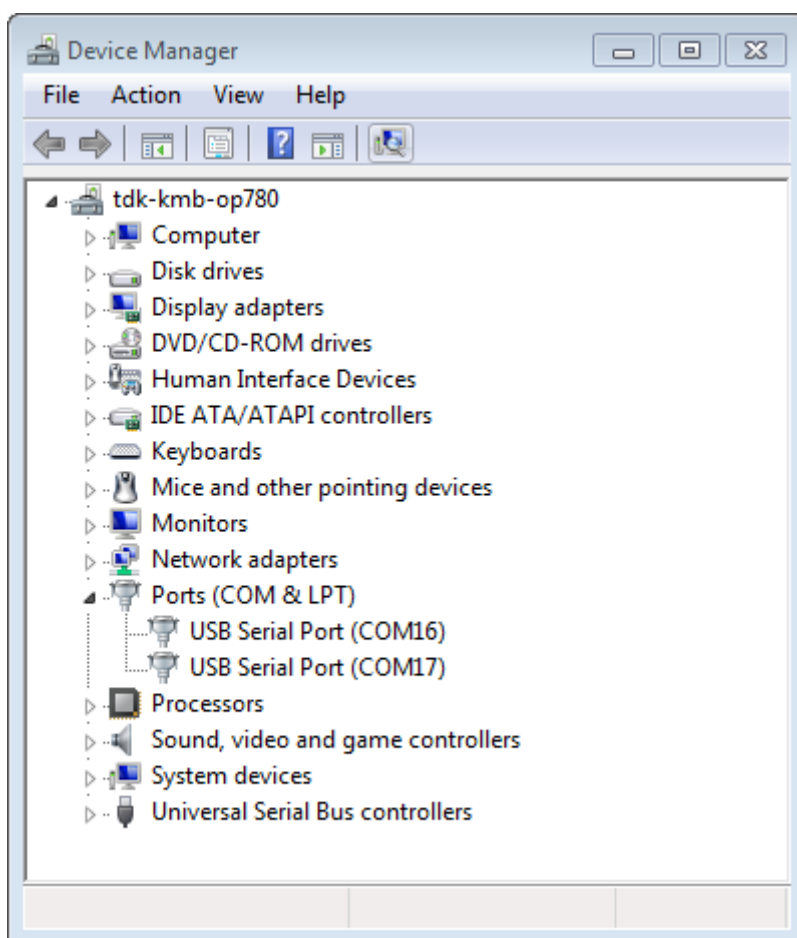


图 29: Windows 设备管理器中的两个 USB-WROVER-KIT 串行端口

```
ls /dev/cu.*
```

**在 Linux 添加用户到 dialout** 当前登录用户可以通过 USB 读写串口。在大多数 Linux 发行版中，这是通过以下命令将用户添加到 dialout 组来完成的：

```
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：

```
sudo usermod -a -G uucp $USER
```

重新登录以确保串行端口的读写权限被启用。

**确认串口连接** 现在验证串口连接是可用的。您可以使用串口终端程序来执行此操作。在这个例子中，我们将使用 **PuTTY SSH Client**，它有 Windows 和 Linux 等平台的版本。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，设置串口：波特率 = 115200，数据位 = 8，停止位 = 1，奇偶校验 = N。以下是设置串口和在 Windows 和 Linux 上传输参数（如 115200-8-1-N）的一些截屏示例。注意选择上述步骤中确认的串口进行设置。

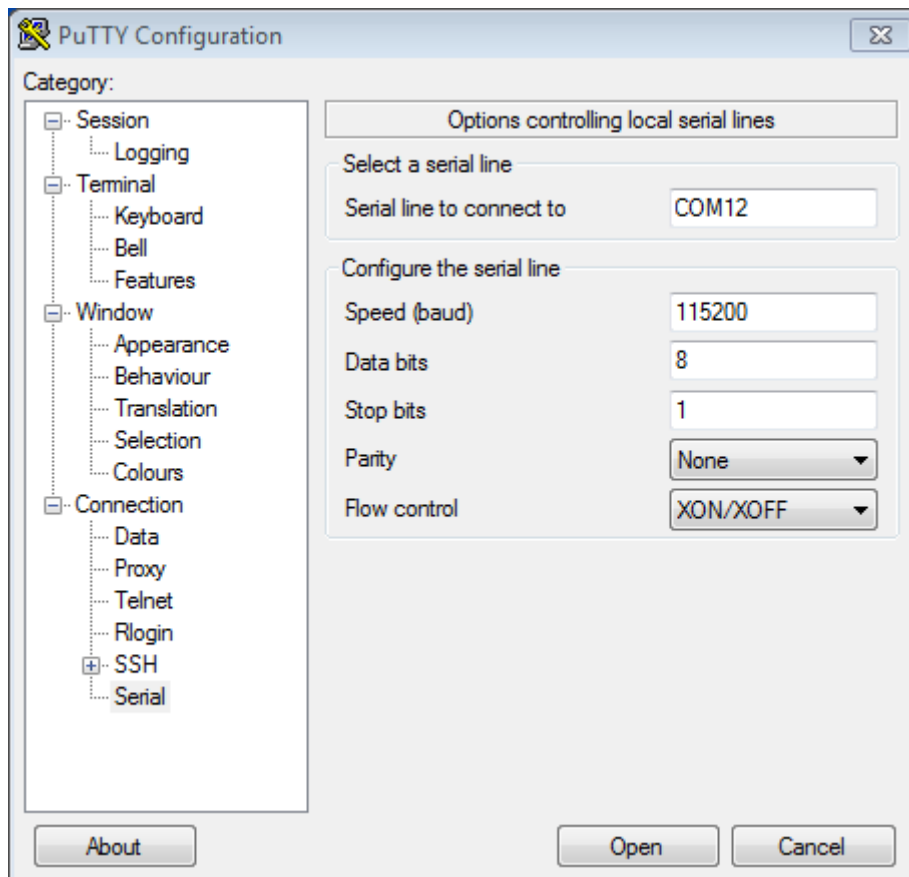


图 30: 在 Windows 上的 PuTTY 设置串口传输。

在终端打开串口，检查是否有任何打印出来的日志。日志内容取决于加载到 ESP32 的应用程序。下图为 ESP32 的一个示例日志。

```
ets Jun 8 2016 00:22:57
```

(下页继续)



图 31: 在 Linux 上的 PuTTY 设置串口传输。



(续上页)

```

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

...

```

如果您看到一些清晰的日志，则表示串行连接正常，您可以继续安装，最后将应用程序上传到 ESP32。

**注解：** 对于某些串口接线配置，在 ESP32 启动并产生串行输出之前，需要在终端程序中禁用串行 RTS & DTR 引脚。这取决于串口适配器硬件本身，大多数开发板（包括所有乐鑫开发板）没有这个问题。此问题仅存在于将 RTS & DTR 引脚直接连接到 EN & GPIO0 引脚上的情况。更多详细信息，参见 [esptool documentation](#)。

**注解：** 验证通讯正常后关闭串口终端。下一步，我们将使用另一个应用程序来上传 ESP32。此应用程序在终端打开时将无法访问串口。

如您在安装用于 ESP32 开发的软件时，从 [第六步：连接设备](#) 小节跳转到了这里，请返回到 [第七步：配置](#) 小节继续阅读。

### 通过 make 指令创建和烧录项目 (传统 GNU Make)

**注解：** Since ESP-IDF V4.0, the default build system is based on CMake. This documentation is for the legacy build system based on GNU Make. Support for this build system may be removed in future major releases.

**寻找项目** 和 [esp-idf-template](#) 项目一样，ESP-IDF 在 Github 上的 [examples](#) 目录下也有示例项目。

找到需要的项目后，切换到其目录，然后可以对其进行配置和构建。

### 配置项目

```
make menuconfig
```

### 编译项目

```
make all
```

…该命令将配置 app 和 bootloader 并根据配置生成分区表。

**烧录项目** 当 make all 结束后，系统将打印一命令行提示您如何使用 `esptool.py` 烧录芯片。用户也可以通过以下指令进行烧录：

```
make flash
```

这种方法将烧录整个项目（包括 app, bootloader 和分割表）到芯片中。通过命令 `make menuconfig` 可以配置串口。

运行 `make flash` 之前无需运行 `make all`。运行 `make flash` 将自动重建烧录所需的一切。

**仅编译和烧录应用程序** 在最初的烧录之后，用户可以仅创建烧录 app，不烧录 bootloader 和分区表：

- `make app` - 仅创建应用程序。
- `make app-flash` - 仅烧录应用程序。

需要时 `make app-flash` 指令将自动重建 app。

如果 bootloader 和分区表不变的话，对他们进行重新烧录并不会有负面影响。

**分区表** 编译完项目后，“build”目录将包含一个名为“my\_app.bin”的二进制文件。这是一个可由 bootloader 加载的 ESP32 映像二进制文件。

一个 ESP32 flash 可以包含多个应用程序，以及多种数据（校准数据，文件系统，参数存储等）。因此，分区表烧录在 flash 偏移地址 0x8000 的地方。

分区表中的每个条目都有一个名称（标签），类型（app，数据或其他），子类型和闪存中分区表被存放的偏移量。

使用分区表最简单的方法是 `make menuconfig` 并选择一个简单的预定义分区表：

- “Single factory app, no OTA”
- “Factory app, two OTA definitions”

在这两种情况下，出厂应用程序的烧录偏移为 0x10000。运行 `make partition_table`，可以打印分区表摘要。

更多关于分区表的信息，以及如何创建自定义分区表，可以查看[文档](#)。

### Eclipse IDE 的创建和烧录指南 (传统 GNU Make)

**注解：** Since ESP-IDF V4.0, the default build system is based on CMake. This documentation is for the legacy build system based on GNU Make. Support for this build system may be removed in future major releases.

**安装 Eclipse IDE** Eclipse IDE 是一个可视化的集成开发环境，可用于编写、编译和调试 ESP-IDF 项目。

- 首先，请在您的平台上安装相应的 ESP-IDF，具体步骤请参考适用于 Windows、OS X 和 Linux 的相应安装步骤。
- 我们建议，您应首先使用命令行创建一个项目，大致熟悉项目的创建流程。此外，您还需要使用命令行 (`make menuconfig`) 对您的 ESP-IDF 项目进行配置。目前，Eclipse 尚不支持对 ESP-IDF 项目进行配置。
- 下载相应版本的 Eclipse Installer 至您的平台，点击 [eclipse.org](http://eclipse.org)。
- 运行 Eclipse Installer，选择“Eclipse for C/C++ Development”（有的版本也可能显示为 CDT）。

**配置 Eclipse IDE** 请打开安装好的 Eclipse IDE，并按照以下步骤进行操作：

#### 导入新项目

- Eclipse IDE 需使用 ESP-IDF 的 Makefile 功能。因此，在使用 Eclipse 前，您需要先创建一个 ESP-IDF 项目。在创建 ESP-IDF 项目时，您可以使用 GitHub 中的 `idf-template` 项目模版，或从 `esp-idf` 子目录中选择一个 `example`。
- 运行 Eclipse，选择“File”->“Import...”。
- 在弹出的对话框中选择“C/C++”->“Existing Code as Makefile Project”，然后点击“Next”。
- 在下个界面中“Existing Code Location”位置输入您的 IDF 项目的路径。注意，这里应输入 ESP-IDF 项目的路径，而非 ESP-IDF 本身的路径（这个稍后再填）。此外，您指定的目标路径中应包含名为 `Makefile`（项目 Makefile）的文件。
- 在本界面，找到“Toolchain for Indexer Settings”，选择“Cross GCC”，最后点击“Finish”。

## 项目属性

- 新项目将出现在“Project Explorer”下。请右键选择该项目，并在菜单中选择“Properties”。
- 点击“C/C++ Build”下的“Environment”属性页，选择“Add...”，并在对应位置输入 BATCH\_BUILD 和 1。
- 再次点击“Add...”，并在“IDF\_PATH”中输入 ESP-IDF 所在的完整安装路径。
- 选择“PATH”环境变量，不要改变默认值。如果 Xtensa 工具链的路径尚不在“PATH”列表中，则应将该路径 (something/xtensa-esp32-elf/bin) 增加至列表，工具链的典型路径类似于 /home/user-name/esp/xtensa-esp32-elf/bin。请注意您需要在附加路径前添加冒号 :。Windows 用户需要将 C:\msys32\mingw32\bin;C:\msys32\opt\xtensa-esp32-elf\bin;C:\msys32\usr\bin 添加到 PATH 环境变量的靠前位置（如果您将 msys32 安装到了其它目录，则需要更改对应的路径以匹配您的本地环境）。
- 在 macOS 平台上，增加一个“PYTHONPATH”环境变量，并将其设置为 /Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/site-packages，保证系统中预先安装的 Python（需安装 pyserial 模块）可以覆盖 Eclipse 内置的任何 Python。
- 前往“C/C++ General”->“Preprocessor Include Paths”属性页面。
  - 点击“Providers”选项卡。
    - \* 从“Providers”列表中选择“CDT Cross GCC Built-in Compiler Settings”，将“Command to get compiler specs”修改为 xtensa-esp32-elf-gcc \${FLAGS} -std=c++11 -E -P -v -dD "\${INPUTS}"
    - \* 从“Providers”列表中选择“CDT GCC Build Output Parser”，将“Compiler command pattern”修改为 xtensa-esp32-elf-(gcc|g\+\+|c\+\+|cc|cpp|clang)
- 前往“C/C++ General”->“Indexer”属性页面。
  - 去除“Allow heuristic resolution of includes”勾选。启用此选项时，Eclipse 有时无法找到正确的头文件目录。

点击“C/C++ General”->“Indexer”属性页。

- 选择“Enable project specific settings”以启用本页上的其他设置。

---

**注解：**取消选中“Allow heuristic resolution of includes”。因为启用此选项时，有时会导致 Eclipse 无法找到正确的头文件目录。

---

点击“C/C++ Build”->“Behavior”属性页。

- 选中“Enable parallel build”以启用多任务并行构建。

**在 Eclipse IDE 中创建项目** 在首次创建项目前，Eclipse IDE 可能会显示大量有关未定义值的错误和警告，主要原因在于项目编译过程中所需的一些源文件是在 ESP-IDF 项目创建过程中自动生成的。因此，这些错误和警告将在 ESP-IDF 项目生成完成后消失。

- 点击“OK”，关闭 Eclipse IDE 中的“Properties”对话框。
- 在 Eclipse IDE 界面外，打开命令管理器。进入项目目录，并通过 `make menuconfig` 命令对您的 ESP-IDF 项目进行配置。现阶段，您还无法在 Eclipse 中完成本操作。

如果您未进行最开始的配置步骤，ESP-IDF 将提示在命令行中进行配置 - 但由于 Eclipse 暂时不支持相关功能，因此该项目将挂起或创建失败。

- 返回 Eclipse IDE 界面中，选择“Project”->“Build”创建您的项目。

**提示：**如果您已经在 Eclipse IDE 环境外创建了项目，则可能需要选择“Project”->“Clean before choosing Project”->“Build”，允许 Eclipse 查看所有源文件的编译器参数，并借此确定头文件包含路径。

**在 Eclipse IDE 中烧录项目** 您可以将 `make flash` 目标放在 Eclipse 项目中，通过 Eclipse UI 调用 `esptool.py` 进行烧录：

- 打开“Project Explorer”，并右击您的项目（请注意右击项目本身，而非项目下的子文件，否则 Eclipse 可能会找到错误的 Makefile）。
- 从菜单中选择“Build Targets”->“Create”。
- 输入“flash”为目标名称，其他选项使用默认值。

- 选择 “Project” -> “Build Target” -> “Build (快捷键: Shift + F9)”, 创建自定义烧录目标, 用于编译、烧录项目。

注意, 您将需要通过 `make menuconfig`, 设置串行端口和其他烧录选项。`make menuconfig` 仍需通过命令行操作 (请见平台的对应指南)。

如有需要, 请按照相同步骤添加 `bootloader` 和 `partition_table`。

**Customized Setup of Toolchain (Legacy GNU Make)** Instead of downloading binary toolchain from Espressif website (see [第一步: 设置工具链](#)) you may build the toolchain yourself.

---

**注解:** Since ESP-IDF V4.0, the default build system is based on CMake. This documentation is for the legacy build system based on GNU Make. Support for this build system may be removed in future major releases.

---

If you can't think of a reason why you need to build it yourself, then probably it's better to stick with the binary version. However, here are some of the reasons why you might want to compile it from source:

- if you want to customize toolchain build configuration
- if you want to use a different GCC version (such as 4.8.5)
- if you want to hack gcc or newlib or libstdc++
- if you are curious and/or have time to spare
- if you don't trust binaries downloaded from the Internet

In any case, here are the instructions to compile the toolchain yourself.

### 从零开始设置 Linux 环境下的工具链(传统 GNU Make)

---

**注解:** Since ESP-IDF V4.0, the default build system is based on CMake. This documentation is for the legacy build system based on GNU Make. Support for this build system may be removed in future major releases.

---

---

**注解:** 安装工具链的标准流程可以通过阅读文档[Linux 平台工具链的标准设置](#)来获得, [工具链的自定义设置](#) 章节会介绍哪些情况下我们必须重新定义工具链。

---

**安装必要的工具** 要想使用 ESP-IDF 进行编译, 您需要获取以下软件包:

- Ubuntu 和 Debian:

```
sudo apt-get install gcc git wget make libncurses-dev flex bison gperf python-  
↳python-pip python-setuptools python-serial python-cryptography python-future-  
↳python-pyparsing python-pyelftools libffi-dev libssl-dev
```

- Arch:

```
sudo pacman -S --needed gcc git make ncurses flex bison gperf python-pyserial-  
↳python-cryptography python-future python-pyparsing python-pyelftools
```

---

**注解:** 一些旧的 (2014 年之前) Linux 发行版中使用的 `pyserial` 版本可能是 2.x, ESP-IDF 并不支持。在这种情况下, 请参考[安装依赖的 Python 软件包](#) 章节, 通过 `pip` 工具来安装支持的版本。

---

### 从源代码编译工具链

- 安装依赖:
  - CentOS 7:

```
sudo yum install gawk gperf grep gettext ncurses-devel python python-devel_
↳automake bison flex texinfo help2man libtool
```

– Ubuntu pre-16.04:

```
sudo apt-get install gawk gperf grep gettext libncurses-dev python python-
↳dev automake bison flex texinfo help2man libtool
```

– Ubuntu 16.04:

```
sudo apt-get install gawk gperf grep gettext python python-dev automake_
↳bison flex texinfo help2man libtool libtool-bin
```

– Debian 9:

```
sudo apt-get install gawk gperf grep gettext libncurses-dev python python-
↳dev automake bison flex texinfo help2man libtool libtool-bin
```

– Arch:

```
TODO
```

新建工作目录，然后进入:

```
mkdir -p ~/esp
cd ~/esp
```

下载 crosstool-NG 然后编译:

```
git clone https://github.com/espressif/crosstool-NG.git
cd crosstool-NG
git checkout esp-2020r3
git submodule update --init
./bootstrap && ./configure --enable-local && make
```

编译工具链:

```
./ct-ng xtensa-esp32-elf
./ct-ng build
chmod -R u+w builds/xtensa-esp32-elf
```

编译得到的工具链会被保存到 ~/esp/crosstool-NG/builds/xtensa-esp32-elf。根据[Linux 下设置环境变量的标准方法](#)中的介绍，将工具链添加到 PATH 中。

**下一步** 继续设置开发环境，请前往[获取 ESP-IDF](#) 章节。

# Chapter 2

## API 参考

### 2.1 蓝牙 API

#### 2.1.1 Controller & VHCI

##### Overview

##### Instructions

##### Application Example

Check `bluetooth/bluedroid/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/bluedroid/hci/controller\\_vhci\\_ble\\_adv](#).

##### API Reference

##### Header File

- `bt/include/esp_bt.h`

##### Functions

`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.

**Return** ESP\_OK - success, other - failed

##### Parameters

- `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_ble_power_type_t power_type)`

Get BLE TX power Connection Tx power should only be get after connection created.

**Return**  $\geq 0$  - Power level,  $< 0$  - Invalid

##### Parameters

- `power_type`: : The type of which tx power, could set Advertising/Connection/Default and etc

`esp_err_t esp_bredr_tx_power_set (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`.

**Return** `ESP_OK` - success, other - failed

**Parameters**

- `min_power_level`: The minimum power level
- `max_power_level`: The maximum power level

*esp\_err\_t* **esp\_bredr\_tx\_power\_get** (*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.

**Return** `ESP_OK` - success, other - failed

**Parameters**

- `min_power_level`: The minimum power level
- `max_power_level`: The maximum power level

*esp\_err\_t* **esp\_bredr\_sco\_datapath\_set** (*esp\_sco\_data\_path\_t* `data_path`)

set default SCO data path Should be called after controller is enabled, and before (e)SCO link is established

**Return** `ESP_OK` - success, other - failed

**Parameters**

- `data_path`: SCO data path

*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.

**Return** `ESP_OK` - success, other - failed

**Parameters**

- `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.

*esp\_err\_t* **esp\_bt\_controller\_deinit** (void)

De-initialize BT controller to free resource and delete task.

This function should be called only once, after any other BT functions are called.

**Return** `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.

**Return** `ESP_OK` - success, other - failed

**Parameters**

- `mode`: : the mode(BLE/BT/BTDM) to enable. For compatible of API, retain this argument. This mode must be equal as the mode in “`cfg`” of `esp_bt_controller_init()`.

*esp\_err\_t* **esp\_bt\_controller\_disable** (void)

Disable BT controller.

**Return** `ESP_OK` - success, other - failed

*esp\_bt\_controller\_status\_t* **esp\_bt\_controller\_get\_status** (void)

Get BT controller is initialised/de-initialised/enabled/disabled.

**Return** status value

uint8\_t \***esp\_bt\_get\_mac** (void)

Get BT MAC address.

**Return** Array pointer of length 6 storing MAC address value.

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.

**Return** true for ready to send, false means cannot send packet

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.

**Parameters**

- 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.

**Return** ESP\_OK - success, ESP\_FAIL - failed

**Parameters**

- 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

**Return** ESP\_OK - success, other - failed

**Parameters**

- 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:

```
esp_bluedroid_disable();
esp_bluedroid_deinit();
esp_bt_controller_disable();
esp_bt_controller_deinit();
esp_bt_mem_release(ESP_BT_MODE_BTDM);
```

**Note** 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.

**Return** `ESP_OK` - success, other - failed

**Parameters**

- `mode`: : the mode whose memory is to be released

*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_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`” .

**Return**

- `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.

**Return**

- `ESP_OK` : success
- other : failed

*esp\_err\_t* **esp\_ble\_scan\_duplicate\_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`.

**Note** This function name is incorrectly spelled, it will be fixed in release 5.x version.

**Return**

- `ESP_OK` : success
- other : failed

## 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)

```
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:*

```
ESP_BT_MODE_IDLE = 0x00
    Bluetooth is not running
ESP_BT_MODE_BLE = 0x01
    Run BLE mode
ESP_BT_MODE_CLASSIC_BT = 0x02
    Run Classic BT mode
ESP_BT_MODE_BTDM = 0x03
    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:*

```
ESP_BLE_SCA_500PPM = 0
    BLE SCA at 500ppm
ESP_BLE_SCA_250PPM
    BLE SCA at 250ppm
ESP_BLE_SCA_150PPM
    BLE SCA at 150ppm
ESP_BLE_SCA_100PPM
    BLE SCA at 100ppm
ESP_BLE_SCA_75PPM
    BLE SCA at 75ppm
ESP_BLE_SCA_50PPM
    BLE SCA at 50ppm
```

**ESP\_BLE\_SCA\_30PPM**  
BLE SCA at 30ppm

**ESP\_BLE\_SCA\_20PPM**  
BLE SCA at 20ppm

**enum esp\_bt\_controller\_status\_t**

Bluetooth controller enable/disable/initialised/de-initialised status.

*Values:*

**ESP\_BT\_CONTROLLER\_STATUS\_IDLE** = 0

**ESP\_BT\_CONTROLLER\_STATUS\_INITED**

**ESP\_BT\_CONTROLLER\_STATUS\_ENABLED**

**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\_ADV : for advertising/scan response. ESP\_BLE\_PWR\_TYPE\_SCAN : for scan. 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:*

**ESP\_BLE\_PWR\_TYPE\_CONN\_HDL0** = 0  
For connection handle 0

**ESP\_BLE\_PWR\_TYPE\_CONN\_HDL1** = 1  
For connection handle 1

**ESP\_BLE\_PWR\_TYPE\_CONN\_HDL2** = 2  
For connection handle 2

**ESP\_BLE\_PWR\_TYPE\_CONN\_HDL3** = 3  
For connection handle 3

**ESP\_BLE\_PWR\_TYPE\_CONN\_HDL4** = 4  
For connection handle 4

**ESP\_BLE\_PWR\_TYPE\_CONN\_HDL5** = 5  
For connection handle 5

**ESP\_BLE\_PWR\_TYPE\_CONN\_HDL6** = 6  
For connection handle 6

**ESP\_BLE\_PWR\_TYPE\_CONN\_HDL7** = 7  
For connection handle 7

**ESP\_BLE\_PWR\_TYPE\_CONN\_HDL8** = 8  
For connection handle 8

**ESP\_BLE\_PWR\_TYPE\_ADV** = 9  
For advertising

**ESP\_BLE\_PWR\_TYPE\_SCAN** = 10  
For scan

**ESP\_BLE\_PWR\_TYPE\_DEFAULT** = 11  
For default, if not set other, it will use default value

**ESP\_BLE\_PWR\_TYPE\_NUM** = 12  
TYPE numbers

**enum esp\_power\_level\_t**

Bluetooth TX power level(index), it's just a index corresponding to power(dbm).

*Values:*

**ESP\_PWR\_LVL\_N12** = 0  
Corresponding to -12dbm

**ESP\_PWR\_LVL\_N9** = 1  
Corresponding to -9dbm

**ESP\_PWR\_LVL\_N6** = 2  
Corresponding to -6dbm

**ESP\_PWR\_LVL\_N3** = 3  
Corresponding to -3dbm

**ESP\_PWR\_LVL\_N0** = 4  
Corresponding to 0dbm

**ESP\_PWR\_LVL\_P3** = 5  
Corresponding to +3dbm

**ESP\_PWR\_LVL\_P6** = 6  
Corresponding to +6dbm

**ESP\_PWR\_LVL\_P9** = 7  
Corresponding to +9dbm

**ESP\_PWR\_LVL\_N14** = [ESP\\_PWR\\_LVL\\_N12](#)  
Backward compatibility! Setting to -14dbm will actually result to -12dbm

**ESP\_PWR\_LVL\_N11** = [ESP\\_PWR\\_LVL\\_N9](#)  
Backward compatibility! Setting to -11dbm will actually result to -9dbm

**ESP\_PWR\_LVL\_N8** = [ESP\\_PWR\\_LVL\\_N6](#)  
Backward compatibility! Setting to -8dbm will actually result to -6dbm

**ESP\_PWR\_LVL\_N5** = [ESP\\_PWR\\_LVL\\_N3](#)  
Backward compatibility! Setting to -5dbm will actually result to -3dbm

**ESP\_PWR\_LVL\_N2** = [ESP\\_PWR\\_LVL\\_N0](#)  
Backward compatibility! Setting to -2dbm will actually result to 0dbm

**ESP\_PWR\_LVL\_P1** = [ESP\\_PWR\\_LVL\\_P3](#)  
Backward compatibility! Setting to +1dbm will actually result to +3dbm

**ESP\_PWR\_LVL\_P4** = [ESP\\_PWR\\_LVL\\_P6](#)  
Backward compatibility! Setting to +4dbm will actually result to +6dbm

**ESP\_PWR\_LVL\_P7** = [ESP\\_PWR\\_LVL\\_P9](#)  
Backward compatibility! Setting to +7dbm will actually result to +9dbm

**enum esp\_sco\_data\_path\_t**

Bluetooth audio data transport path.

*Values:*

**ESP\_SCO\_DATA\_PATH\_HCI** = 0  
data over HCI transport

**ESP\_SCO\_DATA\_PATH\_PCM** = 1  
data over PCM interface

## 2.1.2 BT COMMON

### BT GENERIC DEFINES

**Overview** [Instructions](#)

**Application Example** [Instructions](#)

## API Reference

### Header File

- [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

**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\_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.

**ESP\_APP\_ID\_MAX**

Maximum of the application id.

**ESP\_BD\_ADDR\_STR****ESP\_BD\_ADDR\_HEX** (addr)**Type Definitions**

```
typedef uint8_t esp_bt_octet16_t[ESP_BT_OCTET16_LEN]
```

```
typedef uint8_t esp_bt_octet8_t[ESP_BT_OCTET8_LEN]
```

```
typedef uint8_t esp_link_key[ESP_BT_OCTET16_LEN]
```

```
typedef uint8_t esp_bd_addr_t[ESP_BT_ADDR_LEN]
```

Bluetooth device address.

```
typedef uint8_t esp_ble_key_mask_t
```

**Enumerations**

```
enum esp_bt_status_t
```

Status Return Value.

*Values:*

```
ESP_BT_STATUS_SUCCESS = 0
```

```
ESP_BT_STATUS_FAIL
```

```
ESP_BT_STATUS_NOT_READY
```

```
ESP_BT_STATUS_NOMEM
```

```
ESP_BT_STATUS_BUSY
```

```
ESP_BT_STATUS_DONE = 5
```

```
ESP_BT_STATUS_UNSUPPORTED
```

```
ESP_BT_STATUS_PARM_INVALID
```

```
ESP_BT_STATUS_UNHANDLED
```

```
ESP_BT_STATUS_AUTH_FAILURE
```

```
ESP_BT_STATUS_RMT_DEV_DOWN = 10
```

```
ESP_BT_STATUS_AUTH_REJECTED
```

```
ESP_BT_STATUS_INVALID_STATIC_RAND_ADDR
```

```
ESP_BT_STATUS_PENDING
```

```
ESP_BT_STATUS_UNACCEPT_CONN_INTERVAL
```

```

ESP_BT_STATUS_PARAM_OUT_OF_RANGE
ESP_BT_STATUS_TIMEOUT
ESP_BT_STATUS_PEER_LE_DATA_LEN_UNSUPPORTED
ESP_BT_STATUS_CONTROL_LE_DATA_LEN_UNSUPPORTED
ESP_BT_STATUS_ERR_ILLEGAL_PARAMETER_FMT
ESP_BT_STATUS_MEMORY_FULL = 20
ESP_BT_STATUS_EIR_TOO_LARGE
enum esp_bt_dev_type_t
Bluetooth device type.

Values:
ESP_BT_DEVICE_TYPE_BREDR = 0x01
ESP_BT_DEVICE_TYPE_BLE = 0x02
ESP_BT_DEVICE_TYPE_DUMO = 0x03

enum esp_ble_addr_type_t
BLE device address type.

Values:
BLE_ADDR_TYPE_PUBLIC = 0x00
BLE_ADDR_TYPE_RANDOM = 0x01
BLE_ADDR_TYPE_RPA_PUBLIC = 0x02
BLE_ADDR_TYPE_RPA_RANDOM = 0x03

enum esp_ble_wl_addr_type_t
white list address type

Values:
BLE_WL_ADDR_TYPE_PUBLIC = 0x00
BLE_WL_ADDR_TYPE_RANDOM = 0x01

```

## BT MAIN API

**Overview** [Instructions](#)

**Application Example** [Instructions](#)

## API Reference

### Header File

- [bt/host/bluedroid/api/include/api/esp\\_bt\\_main.h](#)

### Functions

*esp\_bluedroid\_status\_t* **esp\_bluedroid\_get\_status** (void)  
Get bluetooth stack status.

**Return** Bluetooth stack status

*esp\_err\_t* **esp\_bluedroid\_enable** (void)  
Enable bluetooth, must after esp\_bluedroid\_init()



**Return**

- ESP\_OK : Succeed
- Other : Failed

*esp\_err\_t* **esp\_bluedroid\_disable** (void)

Disable bluetooth, must prior to esp\_bluedroid\_deinit()

**Return**

- 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.

**Return**

- 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.

**Return**

- ESP\_OK : Succeed
- Other : Failed

**Enumerations**

**enum esp\_bluedroid\_status\_t**

Bluetooth stack status type, to indicate whether the bluetooth stack is ready.

*Values:*

**ESP\_BLUEDROID\_STATUS\_UNINITIALIZED** = 0

Bluetooth not initialized

**ESP\_BLUEDROID\_STATUS\_INITIALIZED**

Bluetooth initialized but not enabled

**ESP\_BLUEDROID\_STATUS\_ENABLED**

Bluetooth initialized and enabled

**BT DEVICE APIs**

**Overview** Bluetooth device reference APIs.

[Instructions](#)

**Application Example** [Instructions](#)

**API Reference****Header File**

- [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.

**Return**

- ESP\_OK : Succeed
- ESP\_ERR\_INVALID\_ARG : if name is NULL pointer or empty, or string length out of limit
- ESP\_ERR\_INVALID\_STATE : if bluetooth stack is not yet enabled
- ESP\_FAIL : others

**Parameters**

- [in] name: : device name to be set

## 2.1.3 BT LE

### GAP API

#### Overview Instructions

**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](#)

### API Reference

#### Header File

- `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.

**Return**

- ESP\_OK : success
- other : failed

**Parameters**

- [in] callback: callback function

*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.

**Return**

- ESP\_OK : success
- other : failed

**Parameters**

- [in] adv\_data: Pointer to User defined ADV data structure. This memory space can not be freed until callback of `config_adv_data` is received.

*esp\_err\_t* **esp\_ble\_gap\_set\_scan\_params** (*esp\_ble\_scan\_params\_t* \*scan\_params)

This function is called to set scan parameters.

**Return**

- ESP\_OK : success
- other : failed

**Parameters**

- [in] scan\_params: Pointer to User defined scan\_params data structure. This memory space can not be freed until callback of set\_scan\_params

*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.

**Return**

- ESP\_OK : success
- other : failed

**Parameters**

- [in] duration: Keeping the scanning time, the unit is second.

*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.

**Return**

- 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.

**Return**

- ESP\_OK : success
- other : failed

**Parameters**

- [in] adv\_params: pointer to User defined adv\_params data structure.

*esp\_err\_t* **esp\_ble\_gap\_stop\_advertising** (void)

This function is called to stop advertising.

**Return**

- 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.

**Return**

- ESP\_OK : success
- other : failed

**Parameters**

- [in] params: - connection update parameters

*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.

**Return**

- 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.

**Return**

- ESP\_OK : success
- other : failed

**Parameters**

- [in] `rand_addr`: the random address which should be setting

`esp_err_t esp_ble_gap_clear_rand_addr` (void)

This function clears the random address for the application.

**Return**

- ESP\_OK : success
- other : failed

`esp_err_t esp_ble_gap_config_local_privacy` (bool *privacy\_enable*)

Enable/disable privacy on the local device.

**Return**

- ESP\_OK : success
- other : failed

**Parameters**

- [in] `privacy_enable`: - enable/disable privacy on remote device.

`esp_err_t esp_ble_gap_config_local_icon` (uint16\_t *icon*)

set local gap appearance icon

**Return**

- ESP\_OK : success
- other : failed

**Parameters**

- [in] `icon`: - External appearance value, these values are defined by the Bluetooth SIG, please refer to <https://www.bluetooth.com/specifications/gatt/viewer?attributeXmlFile=org.bluetooth.characteristic.gap.appearance.xml>

`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.

**Return**

- ESP\_OK : success
- other : failed

**Parameters**

- [in] `add_remove`: the value is true if added the ble device to the white list, and false remove to the white list.
- [in] `remote_bda`: the remote device address add/remove from the white list.
- [in] `wl_addr_type`: whitelist address type

`esp_err_t esp_ble_gap_get_whitelist_size` (uint16\_t \**length*)

Get the whitelist size in the controller.

**Return**

- ESP\_OK : success
- other : failed

**Parameters**

- [out] `length`: the white list length.

`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.

**Return**

- ESP\_OK : success
- other : failed

**Parameters**

- [in] `bd_addr`: BD address of the peripheral
- [in] `min_conn_int`: minimum preferred connection interval

- [in] `max_conn_int`: maximum preferred connection interval
- [in] `slave_latency`: preferred slave latency
- [in] `supervision_tout`: preferred supervision timeout

*esp\_err\_t* **esp\_ble\_gap\_set\_device\_name** (const char \*name)

Set device name to the local device.

**Return**

- ESP\_OK : success
- other : failed

**Parameters**

- [in] `name`: - device name.

*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.

**Return** - ESP\_OK : success

- other : failed

**Parameters**

- [in] `local_used_addr`: - current local used ble address (six bytes)
- [in] `addr_type`: - ble address type

uint8\_t \***esp\_ble\_resolve\_adv\_data** (uint8\_t \*adv\_data, uint8\_t type, uint8\_t \*length)

This function is called to get ADV data for a specific type.

**Return** pointer of ADV data

**Parameters**

- [in] `adv_data`: - pointer of ADV data which to be resolved
- [in] `type`: - finding ADV data type
- [out] `length`: - return the length of ADV data not including type

*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.

**Return**

- ESP\_OK : success
- other : failed

**Parameters**

- [in] `raw_data`: : raw advertising data
- [in] `raw_data_len`: : raw advertising data length , less than 31 bytes

*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.

**Return**

- ESP\_OK : success
- other : failed

**Parameters**

- [in] `raw_data`: : raw scan response data
- [in] `raw_data_len`: : raw scan response data length , less than 31 bytes

*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.

**Return**

- ESP\_OK : success
- other : failed

**Parameters**

- [in] `remote_addr`: : The remote connection device address.

*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.

#### Return

- ESP\_OK : success
- other : failed

#### Parameters

- [in] type: 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.
- [in] device\_info: the device information.

*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.

#### Return

- ESP\_OK : success
- other : failed

#### Parameters

- [in] type: 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.
- [in] device\_info: the device information.

*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.

#### Return

- ESP\_OK : success
- other : failed

#### Parameters

- [in] list\_type: duplicate scan exceptional list type, the value can be one or more of *esp\_duplicate\_scan\_exceptional\_list\_type\_t*.

*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 following:

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).

**Return** - ESP\_OK : success

- other : failed

#### Parameters

- [in] param\_type: : the type of the param which to be set
- [in] value: : the param value
- [in] len: : the length of the param value

*esp\_err\_t esp\_ble\_gap\_security\_rsp* (*esp\_bd\_addr\_t* *bd\_addr*, *bool* *accept*)

Grant security request access.

**Return** - ESP\_OK : success

- other : failed

**Parameters**

- [in] *bd\_addr*: : BD address of the peer
- [in] *accept*: : accept the security request or not

*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.

**Return** - ESP\_OK : success

- other : failed

**Parameters**

- [in] *bd\_addr*: : the address of the peer device need to encryption
- [in] *sec\_act*: : This is the security action to indicate what kind of BLE security level is required for the BLE link if the BLE is supported

*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.

**Return** - ESP\_OK : success

- other : failed

**Parameters**

- [in] *bd\_addr*: : BD address of the peer
- [in] *accept*: : passkey entry successful or declined.
- [in] *passkey*: : passkey value, must be a 6 digit number, can be lead by 0.

*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.

**Return** - ESP\_OK : success

- other : failed

**Parameters**

- [in] *bd\_addr*: : BD address of the peer device
- [in] *accept*: : numbers to compare are the same or different.

*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.

**Return** - ESP\_OK : success

- other : failed

**Parameters**

- [in] *bd\_addr*: : BD address of the peer device

*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.

**Return** -  $\geq 0$  : bonded devices number.

- ESP\_FAIL : failed

*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.

**Return** - ESP\_OK : success

- other : failed

**Parameters**

- [inout] *dev\_num*: 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*() .
- [out] *dev\_list*: 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.

*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.

**Return** - ESP\_OK : success

- other : failed

**Parameters**

- [in] *bd\_addr*: BD address of the peer device.
- [in] *TK*: TK value, the TK value shall be a 128-bit random number
- [in] *len*: length of tk, should always be 128-bit

*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\_gatt\_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.

**Return** - ESP\_OK : success

- other : failed

**Parameters**

- [in] *remote\_device*: : BD address of the peer device

*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.

**Return** - ESP\_OK : success

- other : failed

**Parameters**

- [in] *bd\_addr*: BD address of the peer device.
- [out] *conn\_params*: the connection parameters information

*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)

**Return** - ESP\_OK : success

- other : failed

**Parameters**

- [in] *bd\_addr*: BD address of the peer device.
- [out] *authorize*: Authorized the link or not.

## Unions

**union esp\_ble\_key\_value\_t**

*#include <esp\_gap\_ble\_api.h>* union type of the security key value

## Public Members

*esp\_ble\_penc\_keys\_t* **penc\_key**

received peer encryption key

*esp\_ble\_pcsrkeys\_t* **pcsrkey**

received peer device SRK

*esp\_ble\_pidkeys\_t* **pid\_key**

peer device ID key

*esp\_ble\_lenckeys\_t* **lenc\_key**

local encryption reproduction keys LTK = d1(ER,DIV,0)

*esp\_ble\_lcsrkeys\_t* **lcsrkey**

local device CSRK = d1(ER,DIV,1)



**union esp\_ble\_sec\_t**

#include <esp\_gap\_ble\_api.h> union associated with ble security

**Public Members**

*esp\_ble\_sec\_key\_notif\_t* **key\_notif**

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\_lenth\_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_except
    Event parameter of ESP_GAP_BLE_UPDATE_DUPLICATE_EXCEPTIONAL_LIST_COMPLETE_EVT

struct ble_adv_data_cmpl_evt_param
    #include <esp_gap_ble_api.h> ESP_GAP_BLE_ADV_DATA_SET_COMPLETE_EVT.

```

**Public Members***esp\_bt\_status\_t* **status**

Indicate the set advertising data operation success status

```

struct ble_adv_data_raw_cmpl_evt_param
    #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

```

struct ble_adv_start_cmpl_evt_param
    #include <esp_gap_ble_api.h> ESP_GAP_BLE_ADV_START_COMPLETE_EVT.

```

**Public Members***esp\_bt\_status\_t* **status**

Indicate advertising start operation success status

```

struct ble_adv_stop_cmpl_evt_param
    #include <esp_gap_ble_api.h> ESP_GAP_BLE_ADV_STOP_COMPLETE_EVT.

```

**Public Members***esp\_bt\_status\_t* **status**

Indicate adv stop operation success status

```

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 operation success status

**struct ble\_get\_bond\_dev\_cmpl\_evt\_param**

#include &lt;esp\_gap\_ble\_api.h&gt; 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 &lt;esp\_gap\_ble\_api.h&gt; 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\_pkt\_data\_length\_cmpl\_evt\_param**

#include &lt;esp\_gap\_ble\_api.h&gt; 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\_rssi\_cmpl\_evt\_param**

#include &lt;esp\_gap\_ble\_api.h&gt; ESP\_GAP\_BLE\_READ\_RSSI\_COMPLETE\_EVT.

**Public Members***esp\_bt\_status\_t* **status**

Indicate the read adv tx power operation success status

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 &lt;esp\_gap\_ble\_api.h&gt; 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 &lt;esp\_gap\_ble\_api.h&gt; 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\_result\_evt\_param**

#include &lt;esp\_gap\_ble\_api.h&gt; 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 **rss\_i**

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 &lt;esp\_gap\_ble\_api.h&gt; 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**

Indicate the set raw advertising data operation success status

```
struct ble_scan_start_cmpl_evt_param
#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

```
struct ble_scan_stop_cmpl_evt_param
#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

```
struct ble_set_rand_cmpl_evt_param
#include <esp_gap_ble_api.h> ESP_GAP_BLE_SET_STATIC_RAND_ADDR_EVT.
```

### Public Members

*esp\_bt\_status\_t* **status**

Indicate set static rand address operation success status

```
struct ble_update_conn_params_evt_param
#include <esp_gap_ble_api.h> ESP_GAP_BLE_UPDATE_CONN_PARAMS_EVT.
```

### Public Members

*esp\_bt\_status\_t* **status**

Indicate update connection parameters success status

*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 conn\_int**  
Current connection interval

**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**  
*#include <esp\_gap\_ble\_api.h>* ESP\_GAP\_BLE\_UPDATE\_DUPLICATE\_EXCEPTIONAL\_LIST\_COMPLETE\_EVT.

### Public Members

*esp\_bt\_status\_t* **status**  
Indicate update duplicate scan exceptional list operation success status

**uint8\_t subcode**  
Define in *esp\_bt\_duplicate\_exceptional\_subcode\_type\_t*

**uint16\_t length**  
The length of *device\_info*

*esp\_duplicate\_info\_t* **device\_info**  
device information, when subcode is ESP\_BLE\_DUPLICATE\_EXCEPTIONAL\_LIST\_CLEAN, the value is invalid

**struct ble\_update\_whitelist\_cmpl\_evt\_param**  
*#include <esp\_gap\_ble\_api.h>* ESP\_GAP\_BLE\_UPDATE\_WHITELIST\_COMPLETE\_EVT.

### Public Members

*esp\_bt\_status\_t* **status**  
Indicate the add or remove whitelist operation success status

*esp\_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

**struct esp\_ble\_adv\_params\_t**  
Advertising parameters.

### Public Members

**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

**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

*esp\_ble\_adv\_type\_t* **adv\_type**  
Advertising type

*esp\_ble\_addr\_type\_t* **own\_addr\_type**  
Owner bluetooth device address type

*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

**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:  $\text{connIntervalmin} = \text{Conn\_Interval\_Min} * 1.25 \text{ ms}$  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:  $\text{connIntervalmax} = \text{Conn\_Interval\_Max} * 1.25 \text{ ms}$  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

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 data 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

`esp_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\_pcsrkeys\_t**  
BLE CSRK keys.

**Public Members**

`uint32_t counter`  
The counter

`esp_bt_octet16_t csrkey`  
The csrkey

`uint8_t sec_level`  
The security level

**struct esp\_ble\_pidkeys\_t**  
BLE pid keys.

**Public Members**

`esp_bt_octet16_t irk`  
The irk value

`esp_ble_addr_type_t addr_type`  
The address type

`esp_bd_addr_t static_addr`  
The static address

**struct esp\_ble\_lenc\_keys\_t**  
BLE Encryption reproduction keys.

**Public Members***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\_lcsrkeys**

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* **csrkey**

The csrkey 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***esp\_bd\_addr\_t* **bd\_addr**

peer address

**struct esp\_ble\_bond\_key\_info\_t**

struct type of the bond key information value

**Public Members***esp\_ble\_key\_mask\_t* **key\_mask**

the key mask to indicate which key is present

*esp\_ble\_penc\_keys\_t* **penc\_key**

received peer encryption key

*esp\_ble\_pcsr\_key\_t* **pcsr\_key**  
received peer device SRK

*esp\_ble\_pid\_key\_t* **pid\_key**  
peer device ID key

**struct esp\_ble\_bond\_dev\_t**  
struct type of the bond device value

### Public Members

*esp\_bd\_addr\_t* **bd\_addr**  
peer address

*esp\_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

*esp\_bd\_addr\_t* **bd\_addr**  
peer address

*esp\_ble\_key\_type\_t* **key\_type**  
key type of the security link

*esp\_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

### 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**

**ESP\_LE\_KEY\_PENC**

**ESP\_LE\_KEY\_PID**

**ESP\_LE\_KEY\_PCSRK**

**ESP\_LE\_KEY\_PLK**

**ESP\_LE\_KEY\_LLK**

**ESP\_LE\_KEY\_LENC**

**ESP\_LE\_KEY\_LID**

**ESP\_LE\_KEY\_LCSRK**

**ESP\_LE\_AUTH\_NO\_BOND**

**ESP\_LE\_AUTH\_BOND**

**ESP\_LE\_AUTH\_REQ\_MITM**

**ESP\_LE\_AUTH\_REQ\_BOND\_MITM**  
0101

**ESP\_LE\_AUTH\_REQ\_SC\_ONLY**

**ESP\_LE\_AUTH\_REQ\_SC\_BOND**

**ESP\_LE\_AUTH\_REQ\_SC\_MITM**

**ESP\_LE\_AUTH\_REQ\_SC\_MITM\_BOND**

**ESP\_BLE\_ONLY\_ACCEPT\_SPECIFIED\_AUTH\_DISABLE**

**ESP\_BLE\_ONLY\_ACCEPT\_SPECIFIED\_AUTH\_ENABLE**

**ESP\_BLE\_OOB\_DISABLE**

**ESP\_BLE\_OOB\_ENABLE**

**ESP\_IO\_CAP\_OUT**

ESP\_IO\_CAP\_IO  
ESP\_IO\_CAP\_IN  
ESP\_IO\_CAP\_NONE  
ESP\_IO\_CAP\_KBDISP  
ESP\_BLE\_APPEARANCE\_UNKNOWN  
ESP\_BLE\_APPEARANCE\_GENERIC\_PHONE  
ESP\_BLE\_APPEARANCE\_GENERIC\_COMPUTER  
ESP\_BLE\_APPEARANCE\_GENERIC\_WATCH  
ESP\_BLE\_APPEARANCE\_SPORTS\_WATCH  
ESP\_BLE\_APPEARANCE\_GENERIC\_CLOCK  
ESP\_BLE\_APPEARANCE\_GENERIC\_DISPLAY  
ESP\_BLE\_APPEARANCE\_GENERIC\_REMOTE  
ESP\_BLE\_APPEARANCE\_GENERIC\_EYEGLASSES  
ESP\_BLE\_APPEARANCE\_GENERIC\_TAG  
ESP\_BLE\_APPEARANCE\_GENERIC\_KEYRING  
ESP\_BLE\_APPEARANCE\_GENERIC\_MEDIA\_PLAYER  
ESP\_BLE\_APPEARANCE\_GENERIC\_BARCODE\_SCANNER  
ESP\_BLE\_APPEARANCE\_GENERIC\_THERMOMETER  
ESP\_BLE\_APPEARANCE\_THERMOMETER\_EAR  
ESP\_BLE\_APPEARANCE\_GENERIC\_HEART\_RATE  
ESP\_BLE\_APPEARANCE\_HEART\_RATE\_BELT  
ESP\_BLE\_APPEARANCE\_GENERIC\_BLOOD\_PRESSURE  
ESP\_BLE\_APPEARANCE\_BLOOD\_PRESSURE\_ARM  
ESP\_BLE\_APPEARANCE\_BLOOD\_PRESSURE\_WRIST  
ESP\_BLE\_APPEARANCE\_GENERIC\_HID  
ESP\_BLE\_APPEARANCE\_HID\_KEYBOARD  
ESP\_BLE\_APPEARANCE\_HID\_MOUSE  
ESP\_BLE\_APPEARANCE\_HID\_JOYSTICK  
ESP\_BLE\_APPEARANCE\_HID\_GAMEPAD  
ESP\_BLE\_APPEARANCE\_HID\_DIGITIZER\_TABLET  
ESP\_BLE\_APPEARANCE\_HID\_CARD\_READER  
ESP\_BLE\_APPEARANCE\_HID\_DIGITAL\_PEN  
ESP\_BLE\_APPEARANCE\_HID\_BARCODE\_SCANNER  
ESP\_BLE\_APPEARANCE\_GENERIC\_GLUCOSE  
ESP\_BLE\_APPEARANCE\_GENERIC\_WALKING  
ESP\_BLE\_APPEARANCE\_WALKING\_IN\_SHOE  
ESP\_BLE\_APPEARANCE\_WALKING\_ON\_SHOE  
ESP\_BLE\_APPEARANCE\_WALKING\_ON\_HIP  
ESP\_BLE\_APPEARANCE\_GENERIC\_CYCLING

**ESP\_BLE\_APPEARANCE\_CYCLING\_COMPUTER**  
**ESP\_BLE\_APPEARANCE\_CYCLING\_SPEED**  
**ESP\_BLE\_APPEARANCE\_CYCLING\_CADENCE**  
**ESP\_BLE\_APPEARANCE\_CYCLING\_POWER**  
**ESP\_BLE\_APPEARANCE\_CYCLING\_SPEED\_CADENCE**  
**ESP\_BLE\_APPEARANCE\_GENERIC\_PULSE\_OXIMETER**  
**ESP\_BLE\_APPEARANCE\_PULSE\_OXIMETER\_FINGERTIP**  
**ESP\_BLE\_APPEARANCE\_PULSE\_OXIMETER\_WRIST**  
**ESP\_BLE\_APPEARANCE\_GENERIC\_WEIGHT**  
**ESP\_BLE\_APPEARANCE\_GENERIC\_PERSONAL\_MOBILITY\_DEVICE**  
**ESP\_BLE\_APPEARANCE\_POWERED\_WHEELCHAIR**  
**ESP\_BLE\_APPEARANCE\_MOBILITY\_SCOOTER**  
**ESP\_BLE\_APPEARANCE\_GENERIC\_CONTINUOUS\_GLUCOSE\_MONITOR**  
**ESP\_BLE\_APPEARANCE\_GENERIC\_INSULIN\_PUMP**  
**ESP\_BLE\_APPEARANCE\_INSULIN\_PUMP\_DURABLE\_PUMP**  
**ESP\_BLE\_APPEARANCE\_INSULIN\_PUMP\_PATCH\_PUMP**  
**ESP\_BLE\_APPEARANCE\_INSULIN\_PEN**  
**ESP\_BLE\_APPEARANCE\_GENERIC\_MEDICATION\_DELIVERY**  
**ESP\_BLE\_APPEARANCE\_GENERIC\_OUTDOOR\_SPORTS**  
**ESP\_BLE\_APPEARANCE\_OUTDOOR\_SPORTS\_LOCATION**  
**ESP\_BLE\_APPEARANCE\_OUTDOOR\_SPORTS\_LOCATION\_AND\_NAV**  
**ESP\_BLE\_APPEARANCE\_OUTDOOR\_SPORTS\_LOCATION\_POD**  
**ESP\_BLE\_APPEARANCE\_OUTDOOR\_SPORTS\_LOCATION\_POD\_AND\_NAV**  
**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)

### 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_duplicate_info_t[ESP_BD_ADDR_LEN]

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.
  
```

### Parameters

- *event*: : Event type

- param: : Point to callback parameter, currently is union type

### Enumerations

**enum esp\_gap\_ble\_cb\_event\_t**

GAP BLE callback event type.

*Values:*

**ESP\_GAP\_BLE\_ADV\_DATA\_SET\_COMPLETE\_EVT = 0**

When advertising data set complete, the event comes

**ESP\_GAP\_BLE\_SCAN\_RSP\_DATA\_SET\_COMPLETE\_EVT**

When scan response data set complete, the event comes

**ESP\_GAP\_BLE\_SCAN\_PARAM\_SET\_COMPLETE\_EVT**

When scan parameters set complete, the event comes

**ESP\_GAP\_BLE\_SCAN\_RESULT\_EVT**

When one scan result ready, the event comes each time

**ESP\_GAP\_BLE\_ADV\_DATA\_RAW\_SET\_COMPLETE\_EVT**

When raw advertising data set complete, the event comes

**ESP\_GAP\_BLE\_SCAN\_RSP\_DATA\_RAW\_SET\_COMPLETE\_EVT**

When raw advertising data set complete, the event comes

**ESP\_GAP\_BLE\_ADV\_START\_COMPLETE\_EVT**

When start advertising complete, the event comes

**ESP\_GAP\_BLE\_SCAN\_START\_COMPLETE\_EVT**

When start scan complete, the event comes

**ESP\_GAP\_BLE\_AUTH\_CMPL\_EVT**

**ESP\_GAP\_BLE\_KEY\_EVT**

**ESP\_GAP\_BLE\_SEC\_REQ\_EVT**

**ESP\_GAP\_BLE\_PASSKEY\_NOTIF\_EVT**

**ESP\_GAP\_BLE\_PASSKEY\_REQ\_EVT**

**ESP\_GAP\_BLE\_OOB\_REQ\_EVT**

**ESP\_GAP\_BLE\_LOCAL\_IR\_EVT**

**ESP\_GAP\_BLE\_LOCAL\_ER\_EVT**

**ESP\_GAP\_BLE\_NC\_REQ\_EVT**

**ESP\_GAP\_BLE\_ADV\_STOP\_COMPLETE\_EVT**

When stop adv complete, the event comes

**ESP\_GAP\_BLE\_SCAN\_STOP\_COMPLETE\_EVT**

When stop scan complete, the event comes

**ESP\_GAP\_BLE\_SET\_STATIC\_RAND\_ADDR\_EVT**

When set the static rand address complete, the event comes

**ESP\_GAP\_BLE\_UPDATE\_CONN\_PARAMS\_EVT**

When update connection parameters complete, the event comes

**ESP\_GAP\_BLE\_SET\_PKT\_LENGTH\_COMPLETE\_EVT**

When set pkt length complete, the event comes

**ESP\_GAP\_BLE\_SET\_LOCAL\_PRIVACY\_COMPLETE\_EVT**

When Enable/disable privacy on the local device complete, the event comes

**ESP\_GAP\_BLE\_REMOVE\_BOND\_DEV\_COMPLETE\_EVT**

When remove the bond device complete, the event comes

**ESP\_GAP\_BLE\_CLEAR\_BOND\_DEV\_COMPLETE\_EVT**

When clear the bond device clear complete, the event comes

**ESP\_GAP\_BLE\_GET\_BOND\_DEV\_COMPLETE\_EVT**

When get the bond device list complete, the event comes

**ESP\_GAP\_BLE\_READ\_RSSI\_COMPLETE\_EVT**

When read the rssi complete, the event comes

**ESP\_GAP\_BLE\_UPDATE\_WHITELIST\_COMPLETE\_EVT**

When add or remove whitelist complete, the event comes

**ESP\_GAP\_BLE\_UPDATE\_DUPLICATE\_EXCEPTIONAL\_LIST\_COMPLETE\_EVT**

When update duplicate exceptional list complete, the event comes

**ESP\_GAP\_BLE\_EVT\_MAX**

**enum esp\_ble\_adv\_data\_type**

The type of advertising data(not adv\_type)

*Values:*

**ESP\_BLE\_AD\_TYPE\_FLAG** = 0x01

**ESP\_BLE\_AD\_TYPE\_16SRV\_PART** = 0x02

**ESP\_BLE\_AD\_TYPE\_16SRV\_CMPL** = 0x03

**ESP\_BLE\_AD\_TYPE\_32SRV\_PART** = 0x04

**ESP\_BLE\_AD\_TYPE\_32SRV\_CMPL** = 0x05

**ESP\_BLE\_AD\_TYPE\_128SRV\_PART** = 0x06

**ESP\_BLE\_AD\_TYPE\_128SRV\_CMPL** = 0x07

**ESP\_BLE\_AD\_TYPE\_NAME\_SHORT** = 0x08

**ESP\_BLE\_AD\_TYPE\_NAME\_CMPL** = 0x09

**ESP\_BLE\_AD\_TYPE\_TX\_PWR** = 0x0A

**ESP\_BLE\_AD\_TYPE\_DEV\_CLASS** = 0x0D

**ESP\_BLE\_AD\_TYPE\_SM\_TK** = 0x10

**ESP\_BLE\_AD\_TYPE\_SM\_OOB\_FLAG** = 0x11

**ESP\_BLE\_AD\_TYPE\_INT\_RANGE** = 0x12

**ESP\_BLE\_AD\_TYPE\_SOL\_SRV\_UUID** = 0x14

**ESP\_BLE\_AD\_TYPE\_128SOL\_SRV\_UUID** = 0x15

**ESP\_BLE\_AD\_TYPE\_SERVICE\_DATA** = 0x16

**ESP\_BLE\_AD\_TYPE\_PUBLIC\_TARGET** = 0x17

**ESP\_BLE\_AD\_TYPE\_RANDOM\_TARGET** = 0x18

**ESP\_BLE\_AD\_TYPE\_APPEARANCE** = 0x19

**ESP\_BLE\_AD\_TYPE\_ADV\_INT** = 0x1A

**ESP\_BLE\_AD\_TYPE\_LE\_DEV\_ADDR** = 0x1b

**ESP\_BLE\_AD\_TYPE\_LE\_ROLE** = 0x1c

**ESP\_BLE\_AD\_TYPE\_SPAIR\_C256** = 0x1d

**ESP\_BLE\_AD\_TYPE\_SPAIR\_R256** = 0x1e

**ESP\_BLE\_AD\_TYPE\_32SOL\_SRV\_UUID** = 0x1f

**ESP\_BLE\_AD\_TYPE\_32SERVICE\_DATA** = 0x20



```

ESP_BLE_AD_TYPE_128SERVICE_DATA = 0x21
ESP_BLE_AD_TYPE_LE_SECURE_CONFIRM = 0x22
ESP_BLE_AD_TYPE_LE_SECURE_RANDOM = 0x23
ESP_BLE_AD_TYPE_URI = 0x24
ESP_BLE_AD_TYPE_INDOOR_POSITION = 0x25
ESP_BLE_AD_TYPE_TRANS_DISC_DATA = 0x26
ESP_BLE_AD_TYPE_LE_SUPPORT_FEATURE = 0x27
ESP_BLE_AD_TYPE_CHAN_MAP_UPDATE = 0x28
ESP_BLE_AD_MANUFACTURER_SPECIFIC_TYPE = 0xFF

```

```
enum esp_ble_adv_type_t
```

Advertising mode.

*Values:*

```

ADV_TYPE_IND = 0x00
ADV_TYPE_DIRECT_IND_HIGH = 0x01
ADV_TYPE_SCAN_IND = 0x02
ADV_TYPE_NONCONN_IND = 0x03
ADV_TYPE_DIRECT_IND_LOW = 0x04

```

```
enum esp_ble_adv_channel_t
```

Advertising channel mask.

*Values:*

```

ADV_CHNL_37 = 0x01
ADV_CHNL_38 = 0x02
ADV_CHNL_39 = 0x04
ADV_CHNL_ALL = 0x07

```

```
enum esp_ble_adv_filter_t
```

*Values:*

```

ADV_FILTER_ALLOW_SCAN_ANY_CON_ANY = 0x00
    Allow both scan and connection requests from anyone.
ADV_FILTER_ALLOW_SCAN_WLST_CON_ANY
    Allow both scan req from White List devices only and connection req from anyone.
ADV_FILTER_ALLOW_SCAN_ANY_CON_WLST
    Allow both scan req from anyone and connection req from White List devices only.
ADV_FILTER_ALLOW_SCAN_WLST_CON_WLST
    Allow scan and connection requests from White List devices only.

```

```
enum esp_ble_sec_act_t
```

*Values:*

```

ESP_BLE_SEC_ENCRYPT = 1
ESP_BLE_SEC_ENCRYPT_NO_MITM
ESP_BLE_SEC_ENCRYPT_MITM

```

```
enum esp_ble_sm_param_t
```

*Values:*

```
ESP_BLE_SM_PASSKEY = 0
```

```

ESP_BLE_SM_AUTHEN_REQ_MODE
ESP_BLE_SM_IOCAP_MODE
ESP_BLE_SM_SET_INIT_KEY
ESP_BLE_SM_SET_RSP_KEY
ESP_BLE_SM_MAX_KEY_SIZE
ESP_BLE_SM_MIN_KEY_SIZE
ESP_BLE_SM_SET_STATIC_PASSKEY
ESP_BLE_SM_CLEAR_STATIC_PASSKEY
ESP_BLE_SM_ONLY_ACCEPT_SPECIFIED_SEC_AUTH
ESP_BLE_SM_OOB_SUPPORT
ESP_BLE_APP_ENC_KEY_SIZE
ESP_BLE_SM_MAX_PARAM

```

```
enum esp_ble_scan_type_t
```

Ble scan type.

*Values:*

```
BLE_SCAN_TYPE_PASSIVE = 0x0
```

Passive scan

```
BLE_SCAN_TYPE_ACTIVE = 0x1
```

Active scan

```
enum esp_ble_scan_filter_t
```

Ble scan filter type.

*Values:*

```
BLE_SCAN_FILTER_ALLOW_ALL = 0x0
```

Accept all :

1. advertisement packets except directed advertising packets not addressed to this device (default).

```
BLE_SCAN_FILTER_ALLOW_ONLY_WLST = 0x1
```

Accept only :

1. advertisement packets from devices where the advertiser' s address is in the White list.
2. Directed advertising packets which are not addressed for this device shall be ignored.

```
BLE_SCAN_FILTER_ALLOW_UND_RPA_DIR = 0x2
```

Accept all :

1. undirected advertisement packets, and
2. directed advertising packets where the initiator address is a resolvable private address, and
3. directed advertising packets addressed to this device.

```
BLE_SCAN_FILTER_ALLOW_WLIST_PRA_DIR = 0x3
```

Accept all :

1. advertisement packets from devices where the advertiser' s address is in the White list, and
2. directed advertising packets where the initiator address is a resolvable private address, and
3. directed advertising packets addressed to this device.

```
enum esp_ble_scan_duplicate_t
```

Ble scan duplicate type.

*Values:*

```
BLE_SCAN_DUPLICATE_DISABLE = 0x0
```

the Link Layer should generate advertising reports to the host for each packet received

**BLE\_SCAN\_DUPLICATE\_ENABLE** = 0x1

the Link Layer should filter out duplicate advertising reports to the Host

**BLE\_SCAN\_DUPLICATE\_MAX** = 0x2

0x02–0xFF, Reserved for future use

**enum esp\_gap\_search\_evt\_t**

Sub Event of ESP\_GAP\_BLE\_SCAN\_RESULT\_EVT.

*Values:*

**ESP\_GAP\_SEARCH\_INQ\_RES\_EVT** = 0

Inquiry result for a peer device.

**ESP\_GAP\_SEARCH\_INQ\_CMPL\_EVT** = 1

Inquiry complete.

**ESP\_GAP\_SEARCH\_DISC\_RES\_EVT** = 2

Discovery result for a peer device.

**ESP\_GAP\_SEARCH\_DISC\_BLE\_RES\_EVT** = 3

Discovery result for BLE GATT based service on a peer device.

**ESP\_GAP\_SEARCH\_DISC\_CMPL\_EVT** = 4

Discovery complete.

**ESP\_GAP\_SEARCH\_DI\_DISC\_CMPL\_EVT** = 5

Discovery complete.

**ESP\_GAP\_SEARCH\_SEARCH\_CANCEL\_CMPL\_EVT** = 6

Search cancelled

**ESP\_GAP\_SEARCH\_INQ\_DISCARD\_NUM\_EVT** = 7

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:*

**ESP\_BLE\_EVT\_CONN\_ADV** = 0x00

Connectable undirected advertising (ADV\_IND)

**ESP\_BLE\_EVT\_CONN\_DIR\_ADV** = 0x01

Connectable directed advertising (ADV\_DIRECT\_IND)

**ESP\_BLE\_EVT\_DISC\_ADV** = 0x02

Scannable undirected advertising (ADV\_SCAN\_IND)

**ESP\_BLE\_EVT\_NON\_CONN\_ADV** = 0x03

Non connectable undirected advertising (ADV\_NONCONN\_IND)

**ESP\_BLE\_EVT\_SCAN\_RSP** = 0x04

Scan Response (SCAN\_RSP)

**enum esp\_ble\_wl\_opration\_t**

*Values:*

**ESP\_BLE\_WHITELIST\_REMOVE** = 0X00

remove mac from whitelist

**ESP\_BLE\_WHITELIST\_ADD** = 0X01

add address to whitelist

**enum esp\_bt\_duplicate\_exceptional\_subcode\_type\_t**

*Values:*

**ESP\_BLE\_DUPLICATE\_EXCEPTIONAL\_LIST\_ADD** = 0

Add device info into duplicate scan exceptional list

**ESP\_BLE\_DUPLICATE\_EXCEPTIONAL\_LIST\_REMOVE**

Remove device info from duplicate scan exceptional list

**ESP\_BLE\_DUPLICATE\_EXCEPTIONAL\_LIST\_CLEAN**

Clean duplicate scan exceptional list

**enum esp\_ble\_duplicate\_exceptional\_info\_type\_t**

*Values:*

**ESP\_BLE\_DUPLICATE\_SCAN\_EXCEPTIONAL\_INFO\_ADV\_ADDR = 0**

BLE advertising address, device info will be added into ESP\_BLE\_DUPLICATE\_SCAN\_EXCEPTIONAL\_ADDR\_LIST

**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

**ESP\_BLE\_DUPLICATE\_SCAN\_EXCEPTIONAL\_INFO\_MESH\_BEACON\_TYPE**

BLE mesh beacon AD type, the format is | Len | 0x2B | Beacon Type | Beacon Data |

**ESP\_BLE\_DUPLICATE\_SCAN\_EXCEPTIONAL\_INFO\_MESH\_PROV\_SRV\_ADV**

BLE mesh provisioning service uuid, the format is | 0x02 | 0x01 | flags | 0x03 | 0x03 | 0x1827 | ... |

**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:*

**ESP\_BLE\_DUPLICATE\_SCAN\_EXCEPTIONAL\_ADDR\_LIST = BLE\_BIT(0)**

duplicate scan exceptional addr list

**ESP\_BLE\_DUPLICATE\_SCAN\_EXCEPTIONAL\_MESH\_LINK\_ID\_LIST = BLE\_BIT(1)**

duplicate scan exceptional mesh link ID list

**ESP\_BLE\_DUPLICATE\_SCAN\_EXCEPTIONAL\_MESH\_BEACON\_TYPE\_LIST = BLE\_BIT(2)**

duplicate scan exceptional mesh beacon type list

**ESP\_BLE\_DUPLICATE\_SCAN\_EXCEPTIONAL\_MESH\_PROV\_SRV\_ADV\_LIST = BLE\_BIT(3)**

duplicate scan exceptional mesh adv with provisioning service uuid

**ESP\_BLE\_DUPLICATE\_SCAN\_EXCEPTIONAL\_MESH\_PROXY\_SRV\_ADV\_LIST = BLE\_BIT(4)**

duplicate scan exceptional mesh adv with provisioning service uuid

**ESP\_BLE\_DUPLICATE\_SCAN\_EXCEPTIONAL\_ALL\_LIST = 0xFFFF**

duplicate scan exceptional all list

## GATT DEFINES

**Overview** [Instructions](#)

**Application Example** [Instructions](#)

## API Reference

### Header File

- [bt/host/bluetooth/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.

**Public Members**

*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.

**Public Members**

*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)

**Public Members**

**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

**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 attr\_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

**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

**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

**struct esp\_gattc\_multi\_t**  
read multiple attribute

**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

***esp\_bt\_uuid\_t* uuid**

The uuid of the service

**struct esp\_gattc\_char\_elem\_t**  
characteristic element

**Public Members****uint16\_t char\_handle**

The characteristic handle

***esp\_gatt\_char\_prop\_t* properties**

The characteristic properties

***esp\_bt\_uuid\_t* uuid**

The characteristic uuid

**struct esp\_gattc\_descr\_elem\_t**  
descriptor element

**Public Members****uint16\_t handle**

The characteristic descriptor handle

***esp\_bt\_uuid\_t* uuid**

The characteristic descriptor uuid

**struct esp\_gattc\_incl\_svc\_elem\_t**  
include service element

**Public Members****uint16\_t handle**

The include service current attribute handle

**uint16\_t incl\_srvc\_s\_handle**

The start handle of the service which has been included

**uint16\_t incl\_srvc\_e\_handle**

The end handle of the service which has been included



*esp\_bt\_uuid\_t* **uuid**

The include service uuid

### Macros

**ESP\_GATT\_UUID\_IMMEDIATE\_ALERT\_SVC**

All “ESP\_GATT\_UUID\_XXX” is attribute types

**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**

**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**  
**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\_INDICATE**

**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:*

**ESP\_GATT\_PREP\_WRITE\_CANCEL** = 0x00

Prepare write cancel

**ESP\_GATT\_PREP\_WRITE\_EXEC** = 0x01

Prepare write execute

**enum** **esp\_gatt\_status\_t**

GATT success code and error codes.

*Values:*

**ESP\_GATT\_OK** = 0x0

**ESP\_GATT\_INVALID\_HANDLE** = 0x01

**ESP\_GATT\_READ\_NOT\_PERMIT** = 0x02

**ESP\_GATT\_WRITE\_NOT\_PERMIT** = 0x03

**ESP\_GATT\_INVALID\_PDU** = 0x04

**ESP\_GATT\_INSUF\_AUTHENTICATION** = 0x05

**ESP\_GATT\_REQ\_NOT\_SUPPORTED** = 0x06

**ESP\_GATT\_INVALID\_OFFSET** = 0x07

**ESP\_GATT\_INSUF\_AUTHORIZATION** = 0x08

**ESP\_GATT\_PREPARE\_Q\_FULL** = 0x09

**ESP\_GATT\_NOT\_FOUND** = 0x0a

```
ESP_GATT_NOT_LONG = 0x0b
ESP_GATT_INSUF_KEY_SIZE = 0x0c
ESP_GATT_INVALID_ATTR_LEN = 0x0d
ESP_GATT_ERR_UNLIKELY = 0x0e
ESP_GATT_INSUF_ENCRYPTION = 0x0f
ESP_GATT_UNSUPPORT_GRP_TYPE = 0x10
ESP_GATT_INSUF_RESOURCE = 0x11
ESP_GATT_NO_RESOURCES = 0x80
ESP_GATT_INTERNAL_ERROR = 0x81
ESP_GATT_WRONG_STATE = 0x82
ESP_GATT_DB_FULL = 0x83
ESP_GATT_BUSY = 0x84
ESP_GATT_ERROR = 0x85
ESP_GATT_CMD_STARTED = 0x86
ESP_GATT_ILLEGAL_PARAMETER = 0x87
ESP_GATT_PENDING = 0x88
ESP_GATT_AUTH_FAIL = 0x89
ESP_GATT_MORE = 0x8a
ESP_GATT_INVALID_CFG = 0x8b
ESP_GATT_SERVICE_STARTED = 0x8c
ESP_GATT_ENCRYPED_MITM = ESP_GATT_OK
ESP_GATT_ENCRYPED_NO_MITM = 0x8d
ESP_GATT_NOT_ENCRYPTED = 0x8e
ESP_GATT_CONGESTED = 0x8f
ESP_GATT_DUP_REG = 0x90
ESP_GATT_ALREADY_OPEN = 0x91
ESP_GATT_CANCEL = 0x92
ESP_GATT_STACK_RSP = 0xe0
ESP_GATT_APP_RSP = 0xe1
ESP_GATT_UNKNOWN_ERROR = 0xef
ESP_GATT_CCC_CFG_ERR = 0xfd
ESP_GATT_PRC_IN_PROGRESS = 0xfe
ESP_GATT_OUT_OF_RANGE = 0xff
```

```
enum esp_gatt_conn_reason_t
    Gatt Connection reason enum.
```

*Values:*

```
ESP_GATT_CONN_UNKNOWN = 0
    Gatt connection unknown
ESP_GATT_CONN_L2C_FAILURE = 1
    General L2cap failure
```

**ESP\_GATT\_CONN\_TIMEOUT** = 0x08  
Connection timeout

**ESP\_GATT\_CONN\_TERMINATE\_PEER\_USER** = 0x13  
Connection terminate by peer user

**ESP\_GATT\_CONN\_TERMINATE\_LOCAL\_HOST** = 0x16  
Connection terminated by local host

**ESP\_GATT\_CONN\_FAIL\_ESTABLISH** = 0x3e  
Connection fail to establish

**ESP\_GATT\_CONN\_LMP\_TIMEOUT** = 0x22  
Connection fail for LMP response tout

**ESP\_GATT\_CONN\_CONN\_CANCEL** = 0x0100  
L2CAP connection cancelled

**ESP\_GATT\_CONN\_NONE** = 0x0101  
No connection to cancel

**enum esp\_gatt\_auth\_req\_t**  
Gatt authentication request type.

*Values:*

**ESP\_GATT\_AUTH\_REQ\_NONE** = 0

**ESP\_GATT\_AUTH\_REQ\_NO\_MITM** = 1

**ESP\_GATT\_AUTH\_REQ\_MITM** = 2

**ESP\_GATT\_AUTH\_REQ\_SIGNED\_NO\_MITM** = 3

**ESP\_GATT\_AUTH\_REQ\_SIGNED\_MITM** = 4

**enum esp\_service\_source\_t**

*Values:*

**ESP\_GATT\_SERVICE\_FROM\_REMOTE\_DEVICE** = 0

**ESP\_GATT\_SERVICE\_FROM\_NVS\_FLASH** = 1

**ESP\_GATT\_SERVICE\_FROM\_UNKNOWN** = 2

**enum esp\_gatt\_write\_type\_t**

Gatt write type.

*Values:*

**ESP\_GATT\_WRITE\_TYPE\_NO\_RSP** = 1  
Gatt write attribute need no response

**ESP\_GATT\_WRITE\_TYPE\_RSP**  
Gatt write attribute need remote response

**enum esp\_gatt\_db\_attr\_type\_t**

the type of attribute element

*Values:*

**ESP\_GATT\_DB\_PRIMARY\_SERVICE**  
Gattc primary service attribute type in the cache

**ESP\_GATT\_DB\_SECONDARY\_SERVICE**  
Gattc secondary service attribute type in the cache

**ESP\_GATT\_DB\_CHARACTERISTIC**  
Gattc characteristic attribute type in the cache

**ESP\_GATT\_DB\_DESCRIPTOR**  
Gattc characteristic descriptor attribute type in the cache

**ESP\_GATT\_DB\_INCLUDED\_SERVICE**

Gattc include service attribute type in the cache

**ESP\_GATT\_DB\_ALL**

Gattc all the attribute (primary service & secondary service & include service & char & descriptor) type in the cache

**GATT SERVER API****Overview** Instructions

**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**

- [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.

**Return**

- 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.

**Return**

- ESP\_OK : success
- other : failed

*esp\_err\_t* **esp\_ble\_gatts\_app\_unregister** (*esp\_gatt\_if\_t* gatts\_if)

unregister with GATT Server.

**Return**

- ESP\_OK : success
- other : failed

**Parameters**

- [in] gatts\_if: GATT server access interface

*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.

#### Return

- ESP\_OK : success
- other : failed

#### Parameters

- [in] gatts\_if: GATT server access interface
- [in] service\_id: service ID.
- [in] num\_handle: number of handle requested for this service.

```
esp_err_t esp_ble_gatts_create_attr_tab(const esp_gatts_attr_db_t *gatts_attr_db,
                                        esp_gatt_if_t gatts_if, uint8_t max_nb_attr, uint8_t
                                        srvc_inst_id)
```

Create a service attribute tab.

#### Return

- ESP\_OK : success
- other : failed

#### Parameters

- [in] gatts\_attr\_db: the pointer to the service attr tab
- [in] gatts\_if: GATT server access interface
- [in] max\_nb\_attr: the number of attribute to be added to the service database.
- [in] srvc\_inst\_id: the instance id of the service

```
esp_err_t esp_ble_gatts_add_included_service(uint16_t service_handle, uint16_t in-
                                             cluded_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\_SRVC\_EVT is reported the included service ID.

#### Return

- ESP\_OK : success
- other : failed

#### Parameters

- [in] service\_handle: service handle to which this included service is to be added.
- [in] included\_service\_handle: the service ID to be included.

```
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.

#### Return

- ESP\_OK : success
- other : failed

#### Parameters

- [in] service\_handle: service handle to which this included service is to be added.
- [in] char\_uuid: : Characteristic UUID.
- [in] perm: : Characteristic value declaration attribute permission.
- [in] property: : Characteristic Properties
- [in] char\_val: : Characteristic value
- [in] control: : attribute response control byte

```
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\_GATTS\_ADD\_DESCR\_EVT is called to report the status and an ID number for this descriptor.

#### Return

- ESP\_OK : success
- other : failed



**Parameters**

- [in] `service_handle`: service handle to which this characteristic descriptor is to be added.
- [in] `perm`: descriptor access permission.
- [in] `descr_uuid`: descriptor UUID.
- [in] `char_descr_val`: : Characteristic descriptor value
- [in] `control`: : attribute response control byte

*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\_GATTS\_DELETE\_EVT is report with the status.

**Return**

- ESP\_OK : success
- other : failed

**Parameters**

- [in] `service_handle`: service\_handle to be deleted.

*esp\_err\_t* **esp\_ble\_gatts\_start\_service** (uint16\_t *service\_handle*)

This function is called to start a service.

**Return**

- ESP\_OK : success
- other : failed

**Parameters**

- [in] `service_handle`: the service handle to be started.

*esp\_err\_t* **esp\_ble\_gatts\_stop\_service** (uint16\_t *service\_handle*)

This function is called to stop a service.

**Return**

- ESP\_OK : success
- other : failed

**Parameters**

- [in] `service_handle`: - service to be topped.

*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.

**Return**

- ESP\_OK : success
- other : failed

**Parameters**

- [in] `gatts_if`: GATT server access interface
- [in] `conn_id`: - connection id to indicate.
- [in] `attr_handle`: - attribute handle to indicate.
- [in] `value_len`: - indicate value length.
- [in] `value`: value to indicate.
- [in] `need_confirm`: - Whether a confirmation is required. false sends a GATT notification, true sends a GATT indication.

*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.

**Return**

- ESP\_OK : success
- other : failed

**Parameters**

- [in] `gatts_if`: GATT server access interface
- [in] `conn_id`: - connection identifier.

- [in] `trans_id`: - transfer id
- [in] `status`: - response status
- [in] `rsp`: - response data.

*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.

#### Return

- `ESP_OK` : success
- other : failed

#### Parameters

- [in] `attr_handle`: the attribute handle which to be set
- [in] `length`: the value length
- [in] `value`: the pointer to the attribute value

*esp\_gatt\_status\_t* **esp\_ble\_gatts\_get\_attr\_value** (uint16\_t *attr\_handle*, uint16\_t \**length*, const uint8\_t \*\**value*)

Retrieve attribute value.

#### Return

- `ESP_GATT_OK` : success
- other : failed

#### Parameters

- [in] `attr_handle`: Attribute handle.
- [out] `length`: pointer to the attribute value length
- [out] `value`: Pointer to attribute value payload, the value cannot be modified by user

*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.

#### Return

- `ESP_OK` : success
- other : failed

#### Parameters

- [in] `gatts_if`: GATT server access interface
- [in] `remote_bda`: remote device bluetooth device address.
- [in] `is_direct`: direct connection or background auto connection

*esp\_err\_t* **esp\_ble\_gatts\_close** (*esp\_gatt\_if\_t* *gatts\_if*, uint16\_t *conn\_id*)

Close a connection a remote device.

#### Return

- `ESP_OK` : success
- other : failed

#### Parameters

- [in] `gatts_if`: GATT server access interface
- [in] `conn_id`: connection ID to be closed.

*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.

#### Return

- `ESP_OK` : success
- other : failed

#### Parameters

- [in] `gatts_if`: GATT server access interface
- [in] `remote_bda`: 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

## Unions

**union esp\_ble\_gatts\_cb\_param\_t**

#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_GATTS_CANCEL_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

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

*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.

#### Parameters

- **event**: : Event type
- **gatts\_if**: : GATT server access interface, normally different **gatts\_if** correspond to different profile
- **param**: : Point to callback parameter, currently is union type

### Enumerations

**enum esp\_gatts\_cb\_event\_t**

GATT Server callback function events.

*Values:***ESP\_GATTS\_REG\_EVT = 0**

When register application id, the event comes

**ESP\_GATTS\_READ\_EVT = 1**

When gatt client request read operation, the event comes

**ESP\_GATTS\_WRITE\_EVT = 2**

When gatt client request write operation, the event comes

**ESP\_GATTS\_EXEC\_WRITE\_EVT = 3**

When gatt client request execute write, the event comes

**ESP\_GATTS\_MTU\_EVT = 4**

When set mtu complete, the event comes

**ESP\_GATTS\_CONF\_EVT = 5**

When receive confirm, the event comes

**ESP\_GATTS\_UNREG\_EVT = 6**

When unregister application id, the event comes

**ESP\_GATTS\_CREATE\_EVT = 7**

When create service complete, the event comes

**ESP\_GATTS\_ADD\_INCL\_SRVC\_EVT = 8**

When add included service complete, the event comes

**ESP\_GATTS\_ADD\_CHAR\_EVT = 9**

When add characteristic complete, the event comes

**ESP\_GATTS\_ADD\_CHAR\_DESCR\_EVT = 10**

When add descriptor complete, the event comes

**ESP\_GATTS\_DELETE\_EVT = 11**

When delete service complete, the event comes

**ESP\_GATTS\_START\_EVT = 12**

When start service complete, the event comes

**ESP\_GATTS\_STOP\_EVT = 13**

When stop service complete, the event comes

**ESP\_GATTS\_CONNECT\_EVT = 14**

When gatt client connect, the event comes

**ESP\_GATTS\_DISCONNECT\_EVT = 15**

When gatt client disconnect, the event comes

**ESP\_GATTS\_OPEN\_EVT = 16**

When connect to peer, the event comes

**ESP\_GATTS\_CANCEL\_OPEN\_EVT = 17**

When disconnect from peer, the event comes

**ESP\_GATTS\_CLOSE\_EVT = 18**

When gatt server close, the event comes

**ESP\_GATTS\_LISTEN\_EVT = 19**

When gatt listen to be connected the event comes

**ESP\_GATTS\_CONGEST\_EVT = 20**

When congest happen, the event comes

**ESP\_GATTS\_RESPONSE\_EVT = 21**

When gatt send response complete, the event comes

**ESP\_GATTS\_CREAT\_ATTR\_TAB\_EVT = 22**

When gatt create table complete, the event comes

**ESP\_GATTS\_SET\_ATTR\_VAL\_EVT = 23**

When gatt set attr value complete, the event comes

**ESP\_GATTS\_SEND\_SERVICE\_CHANGE\_EVT = 24**

When gatt send service change indication complete, the event comes

## GATT CLIENT API

### Overview [Instructions](#)

**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

- [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.

#### Return

- ESP\_OK: success
- other: failed

#### Parameters

- [in] callback: : pointer to the application callback function.

*esp\_err\_t* **esp\_ble\_gattc\_app\_register** (uint16\_t app\_id)

This function is called to register application callbacks with GATTC module.

#### Return

- ESP\_OK: success
- other: failed

#### Parameters

- [in] app\_id: : Application Identify (UUID), for different application

*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.

#### Return

- ESP\_OK: success
- other: failed

**Parameters**

- [in] `gattc_if`: Gatt client access interface.

`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.

**Return**

- ESP\_OK: success
- other: failed

**Parameters**

- [in] `gattc_if`: Gatt client access interface.
- [in] `remote_bda`: remote device bluetooth device address.
- [in] `remote_addr_type`: remote device bluetooth device the address type.
- [in] `is_direct`: direct connection or background auto connection(by now, background auto connection is not supported).

`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)`.

**Return**

- ESP\_OK: success
- other: failed

**Parameters**

- [in] `gattc_if`: Gatt client access interface.
- [in] `conn_id`: connection ID to be closed.

`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.

**Return**

- ESP\_OK: success
- other: failed

**Parameters**

- [in] `gattc_if`: Gatt client access interface.
- [in] `conn_id`: connection ID.

`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.

**Return**

- ESP\_OK: success
- other: failed

**Parameters**

- [in] `gattc_if`: Gatt client access interface.
- [in] `conn_id`: connection ID.
- [in] `filter_uuid`: a UUID of the service application is interested in. If Null, discover for all services.

`esp_gatt_status_t esp_ble_gattc_get_service(esp_gatt_if_t gattc_if, uint16_t conn_id, esp_bt_uuid_t *svc_uuid, esp_gattc_service_elem_t *result, uint16_t *count, uint16_t offset)`

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_search_service`.

**Return**

- ESP\_OK: success

- other: failed

#### Parameters

- [in] `gattc_if`: Gatt client access interface.
- [in] `conn_id`: connection ID which identify the server.
- [in] `svc_uuid`: the pointer to the service uuid.
- [out] `result`: The pointer to the service which has been found in the gattc cache.
- [inout] `count`: 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.
- [in] `offset`: Offset of the service position to get.

```
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.

#### Return

- ESP\_OK: success
- other: failed

#### Parameters

- [in] `gattc_if`: Gatt client access interface.
- [in] `conn_id`: connection ID which identify the server.
- [in] `start_handle`: the attribute start handle.
- [in] `end_handle`: the attribute end handle
- [out] `result`: The pointer to the characteristic in the service.
- [inout] `count`: 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.
- [in] `offset`: Offset of the characteristic position to get.

```
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.

#### Return

- ESP\_OK: success
- other: failed

#### Parameters

- [in] `gattc_if`: Gatt client access interface.
- [in] `conn_id`: connection ID which identify the server.
- [in] `char_handle`: the given characteristic handle
- [out] `result`: The pointer to the descriptor in the characteristic.
- [inout] `count`: 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.
- [in] `offset`: Offset of the descriptor position to get.

```
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.

#### Return

- ESP\_OK: success
- other: failed

#### Parameters

- [in] `gattc_if`: Gatt client access interface.
- [in] `conn_id`: connection ID which identify the server.

- [in] start\_handle: the attribute start handle
- [in] end\_handle: the attribute end handle
- [in] char\_uuid: the characteristic uuid
- [out] result: The pointer to the characteristic in the service.
- [inout] count: 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.

```
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_bt_uuid_t
char_uuid, esp_bt_uuid_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.

#### Return

- ESP\_OK: success
- other: failed

#### Parameters

- [in] gattc\_if: Gatt client access interface.
- [in] conn\_id: connection ID which identify the server.
- [in] start\_handle: the attribute start handle
- [in] end\_handle: the attribute end handle
- [in] char\_uuid: the characteristic uuid.
- [in] descr\_uuid: the descriptor uuid.
- [out] result: The pointer to the descriptor in the given characteristic.
- [inout] count: 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.

```
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_bt_uuid_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.

#### Return

- ESP\_OK: success
- other: failed

#### Parameters

- [in] gattc\_if: Gatt client access interface.
- [in] conn\_id: connection ID which identify the server.
- [in] char\_handle: the characteristic handle.
- [in] descr\_uuid: the descriptor uuid.
- [out] result: The pointer to the descriptor in the given characteristic.
- [inout] count: 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.

```
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_bt_uuid_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.

#### Return

- ESP\_OK: success
- other: failed

#### Parameters

- [in] `gattc_if`: Gatt client access interface.
- [in] `conn_id`: connection ID which identify the server.
- [in] `start_handle`: the attribute start handle
- [in] `end_handle`: the attribute end handle
- [in] `incl_uuid`: the include service uuid
- [out] `result`: The pointer to the include service in the given service.
- [inout] `count`: 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.

```
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.

#### Return

- ESP\_OK: success
- other: failed

#### Parameters

- [in] `gattc_if`: Gatt client access interface.
- [in] `conn_id`: connection ID which identify the server.
- [in] `type`: the attribute type.
- [in] `start_handle`: the attribute start handle, if the type is ESP\_GATT\_DB\_DESCRIPTOR, this parameter should be ignore
- [in] `end_handle`: the attribute end handle, if the type is ESP\_GATT\_DB\_DESCRIPTOR, this parameter should be ignore
- [in] `char_handle`: 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.
- [out] `count`: output the number of attribute has been found in the gattc cache with the given attribute type.

```
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.

#### Return

- ESP\_OK: success
- other: failed

#### Parameters

- [in] `gattc_if`: Gatt client access interface.
- [in] `start_handle`: the attribute start handle
- [in] `end_handle`: the attribute end handle
- [in] `conn_id`: connection ID which identify the server.
- [in] `db`: output parameter which will contain the GATT database copy. Caller is responsible for freeing it.
- [in] `count`: number of elements in database.

```
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.

#### Return

- ESP\_OK: success
- other: failed

#### Parameters

- [in] `gattc_if`: Gatt client access interface.
- [in] `conn_id`: connection ID.
- [in] `handle`: characteristic handle to read.
- [in] `auth_req`: authenticate request type

```
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.

#### Return

- ESP\_OK: success
- other: failed

#### Parameters

- [in] gattc\_if: Gatt client access interface.
- [in] conn\_id: : connection ID.
- [in] start\_handle: : the attribute start handle.
- [in] end\_handle: : the attribute end handle
- [in] uuid: : The UUID of attribute which will be read.
- [in] auth\_req: : authenticate request type

```
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.

#### Return

- ESP\_OK: success
- other: failed

#### Parameters

- [in] gattc\_if: Gatt client access interface.
- [in] conn\_id: : connection ID.
- [in] read\_multi: : pointer to the read multiple parameter.
- [in] auth\_req: : authenticate request type

```
esp_err_t esp_ble_gattc_read_char_descr(esp_gatt_if_t gattc_if, uint16_t conn_id, uint16_t han-
dle, esp_gatt_auth_req_t auth_req)
```

This function is called to read a characteristics descriptor.

#### Return

- ESP\_OK: success
- other: failed

#### Parameters

- [in] gattc\_if: Gatt client access interface.
- [in] conn\_id: : connection ID.
- [in] handle: : descriptor handle to read.
- [in] auth\_req: : authenticate request type

```
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.

#### Return

- ESP\_OK: success
- other: failed

#### Parameters

- [in] gattc\_if: Gatt client access interface.
- [in] conn\_id: : connection ID.
- [in] handle: : characteristic handle to write.
- [in] value\_len: length of the value to be written.
- [in] value: : the value to be written.
- [in] write\_type: : the type of attribute write operation.
- [in] auth\_req: : authentication request.



```
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.

#### Return

- ESP\_OK: success
- other: failed

#### Parameters

- [in] gattc\_if: Gatt client access interface.
- [in] conn\_id: : connection ID
- [in] handle: : descriptor handle to write.
- [in] value\_len: length of the value to be written.
- [in] value: : the value to be written.
- [in] write\_type: : the type of attribute write operation.
- [in] auth\_req: : authentication request.

```
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.

#### Return

- ESP\_OK: success
- other: failed

#### Parameters

- [in] gattc\_if: Gatt client access interface.
- [in] conn\_id: : connection ID.
- [in] handle: : characteristic handle to prepare write.
- [in] offset: : offset of the write value.
- [in] value\_len: length of the value to be written.
- [in] value: : the value to be written.
- [in] auth\_req: : authentication request.

```
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.

#### Return

- ESP\_OK: success
- other: failed

#### Parameters

- [in] gattc\_if: Gatt client access interface.
- [in] conn\_id: : connection ID.
- [in] handle: : characteristic descriptor handle to prepare write.
- [in] offset: : offset of the write value.
- [in] value\_len: length of the value to be written.
- [in] value: : the value to be written.
- [in] auth\_req: : authentication request.

```
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.

#### Return

- ESP\_OK: success
- other: failed

#### Parameters

- [in] gattc\_if: Gatt client access interface.
- [in] conn\_id: : connection ID.

- [in] `is_execute`: : execute or cancel.

`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.

#### Return

- `ESP_OK`: registration succeeds
- other: failed

#### Parameters

- [in] `gattc_if`: Gatt client access interface.
- [in] `server_bda`: : target GATT server.
- [in] `handle`: : GATT characteristic handle.

`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.

#### Return

- `ESP_OK`: unregister succeeds
- other: failed

#### Parameters

- [in] `gattc_if`: Gatt client access interface.
- [in] `server_bda`: : target GATT server.
- [in] `handle`: : GATT characteristic handle.

`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.

#### Return

- `ESP_OK`: success
- other: failed

#### Parameters

- [in] `remote_bda`: remote device BD address.

`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 means that device want to use the database has stored in the local cache. The source address means that device want to share the database to the associated address device.

#### Return

- `ESP_OK`: success
- other: failed

#### Parameters

- [in] `gattc_if`: Gatt client access interface.
- [in] `src_addr`: the source address which provide the attribute table.
- [in] `assoc_addr`: the associated device address which went to share the attribute table with the source address.
- [in] `is_assoc`: true add the associated device address, false remove the associated device address.

`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.

#### Return

- `ESP_OK`: success
- other: failed

**Parameters**

- [in] `gattc_if`: Gatt client access interface.

`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,.

**Return**

- `ESP_OK`: success
- other: failed

**Parameters**

- [in] `remote_bda`: remote device BD address.

**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**

```

esp_gatt_status_t status
    Operation status

uint16_t conn_id
    Connection id

esp_bd_addr_t remote_bda
    Remote bluetooth device address

esp_gatt_conn_reason_t reason
    The reason of gatt connection close

struct gattc_congest_evt_param
    #include <esp_gattc_api.h> ESP_GATTC_CONGEST_EVT.

```

**Public Members**

```

uint16_t conn_id
    Connection id

bool congested
    Congested or not

struct gattc_connect_evt_param
    #include <esp_gattc_api.h> ESP_GATTC_CONNECT_EVT.

```

**Public Members**

`uint16_t conn_id`  
Connection id

`esp_bd_addr_t remote_bda`  
Remote bluetooth device address

`esp_gatt_conn_params_t conn_params`  
current connection parameters

**struct gattc\_dis\_srvc\_cmpl\_evt\_param**  
*#include <esp\_gattc\_api.h>* ESP\_GATTC\_DIS\_SRVC\_CMPL\_EVT.

**Public Members**

`esp_gatt_status_t status`  
Operation status

`uint16_t conn_id`  
Connection id

**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

`esp_bd_addr_t remote_bda`  
Remote bluetooth device address

**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

**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

`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

`esp_gatt_status_t status`  
Operation status

`uint16_t conn_id`  
Connection id

`bool is_full`  
The gattc command queue is full or not

```
struct gattc_read_char_evt_param  
#include <esp_gattc_api.h> ESP_GATTC_READ_CHAR_EVT, ESP_GATTC_READ_DESCR_EVT.
```

**Public Members**

*esp\_gatt\_status\_t* **status**  
Operation status

uint16\_t **conn\_id**  
Connection id

uint16\_t **handle**  
Characteristic handle

uint8\_t \***value**  
Characteristic value

uint16\_t **value\_len**  
Characteristic value length

**struct gattc\_reg\_evt\_param**  
*#include <esp\_gattc\_api.h>* ESP\_GATTC\_REG\_EVT.

**Public Members**

*esp\_gatt\_status\_t* **status**  
Operation status

uint16\_t **app\_id**  
Application id which input in register API

**struct gattc\_reg\_for\_notify\_evt\_param**  
*#include <esp\_gattc\_api.h>* ESP\_GATTC\_REG\_FOR\_NOTIFY\_EVT.

**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 srvc_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**

`esp_gatt_status_t status`  
Operation status

**struct gattc\_srvc\_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.

#### Parameters

- `event`: : Event type
- `gattc_if`: : GATT client access interface, normally different `gattc_if` correspond to different profile
- `param`: : Point to callback parameter, currently is union type

### Enumerations

```
enum esp_gattc_cb_event_t
```

GATT Client callback function events.

*Values:*

```
ESP_GATTC_REG_EVT = 0
```

When GATT client is registered, the event comes

```
ESP_GATTC_UNREG_EVT = 1
```

When GATT client is unregistered, the event comes

```
ESP_GATTC_OPEN_EVT = 2
```

When GATT virtual connection is set up, the event comes

```
ESP_GATTC_READ_CHAR_EVT = 3
```

When GATT characteristic is read, the event comes

```
ESP_GATTC_WRITE_CHAR_EVT = 4
```

When GATT characteristic write operation completes, the event comes

```
ESP_GATTC_CLOSE_EVT = 5
```

When GATT virtual connection is closed, the event comes

```
ESP_GATTC_SEARCH_CMPL_EVT = 6
```

When GATT service discovery is completed, the event comes

```
ESP_GATTC_SEARCH_RES_EVT = 7
```

When GATT service discovery result is got, the event comes

```
ESP_GATTC_READ_DESCR_EVT = 8
```

When GATT characteristic descriptor read completes, the event comes

```
ESP_GATTC_WRITE_DESCR_EVT = 9
```

When GATT characteristic descriptor write completes, the event comes

```
ESP_GATTC_NOTIFY_EVT = 10
```

When GATT notification or indication arrives, the event comes

```
ESP_GATTC_PREP_WRITE_EVT = 11
```

When GATT prepare-write operation completes, the event comes

```
ESP_GATTC_EXEC_EVT = 12
```

When write execution completes, the event comes

```
ESP_GATTC_ACL_EVT = 13
```

When ACL connection is up, the event comes

```
ESP_GATTC_CANCEL_OPEN_EVT = 14
```

When GATT client ongoing connection is cancelled, the event comes

- ESP\_GATTC\_SRVC\_CHG\_EVT = 15**  
When “service changed” occurs, the event comes
- ESP\_GATTC\_ENC\_CMPL\_CB\_EVT = 17**  
When encryption procedure completes, the event comes
- ESP\_GATTC\_CFG\_MTU\_EVT = 18**  
When configuration of MTU completes, the event comes
- ESP\_GATTC\_ADV\_DATA\_EVT = 19**  
When advertising of data, the event comes
- ESP\_GATTC\_MULT\_ADV\_ENB\_EVT = 20**  
When multi-advertising is enabled, the event comes
- ESP\_GATTC\_MULT\_ADV\_UPD\_EVT = 21**  
When multi-advertising parameters are updated, the event comes
- ESP\_GATTC\_MULT\_ADV\_DATA\_EVT = 22**  
When multi-advertising data arrives, the event comes
- ESP\_GATTC\_MULT\_ADV\_DIS\_EVT = 23**  
When multi-advertising is disabled, the event comes
- ESP\_GATTC\_CONGEST\_EVT = 24**  
When GATT connection congestion comes, the event comes
- ESP\_GATTC\_BTH\_SCAN\_ENB\_EVT = 25**  
When batch scan is enabled, the event comes
- ESP\_GATTC\_BTH\_SCAN\_CFG\_EVT = 26**  
When batch scan storage is configured, the event comes
- ESP\_GATTC\_BTH\_SCAN\_RD\_EVT = 27**  
When Batch scan read event is reported, the event comes
- ESP\_GATTC\_BTH\_SCAN\_THR\_EVT = 28**  
When Batch scan threshold is set, the event comes
- ESP\_GATTC\_BTH\_SCAN\_PARAM\_EVT = 29**  
When Batch scan parameters are set, the event comes
- ESP\_GATTC\_BTH\_SCAN\_DIS\_EVT = 30**  
When Batch scan is disabled, the event comes
- ESP\_GATTC\_SCAN\_FLT\_CFG\_EVT = 31**  
When Scan filter configuration completes, the event comes
- ESP\_GATTC\_SCAN\_FLT\_PARAM\_EVT = 32**  
When Scan filter parameters are set, the event comes
- ESP\_GATTC\_SCAN\_FLT\_STATUS\_EVT = 33**  
When Scan filter status is reported, the event comes
- ESP\_GATTC\_ADV\_VSC\_EVT = 34**  
When advertising vendor spec content event is reported, the event comes
- ESP\_GATTC\_REG\_FOR\_NOTIFY\_EVT = 38**  
When register for notification of a service completes, the event comes
- ESP\_GATTC\_UNREG\_FOR\_NOTIFY\_EVT = 39**  
When unregister for notification of a service completes, the event comes
- ESP\_GATTC\_CONNECT\_EVT = 40**  
When the ble physical connection is set up, the event comes
- ESP\_GATTC\_DISCONNECT\_EVT = 41**  
When the ble physical connection disconnected, the event comes

**ESP\_GATTC\_READ\_MULTIPLE\_EVT = 42**

When the ble characteristic or descriptor multiple complete, the event comes

**ESP\_GATTC\_QUEUE\_FULL\_EVT = 43**

When the gattc command queue full, the event comes

**ESP\_GATTC\_SET\_ASSOC\_EVT = 44**

When the ble gattc set the associated address complete, the event comes

**ESP\_GATTC\_GET\_ADDR\_LIST\_EVT = 45**

When the ble get gattc address list in cache finish, the event comes

**ESP\_GATTC\_DIS\_SRVC\_CMPL\_EVT = 46**

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/bluedroid/ble](#) 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/bluedroid/ble/blufi](#)

## API Reference

### Header File

- [bt/host/bluedroid/api/include/api/esp\\_blufi\\_api.h](#)

### Functions

*esp\_err\_t* **esp\_blufi\_register\_callbacks** (*esp\_blufi\_callbacks\_t* \*callbacks)

This function is called to receive blufi callback event.

**Return** ESP\_OK - success, other - failed

#### Parameters

- [in] *callbacks*: callback functions

*esp\_err\_t* **esp\_blufi\_profile\_init** (void)

This function is called to initialize blufi\_profile.

**Return** ESP\_OK - success, other - failed

*esp\_err\_t* **esp\_blufi\_profile\_deinit** (void)

This function is called to de-initialize blufi\_profile.

**Return** 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 connection report.

**Return** ESP\_OK - success, other - failed

#### Parameters

- *opmode*: : wifi opmode

- `sta_conn_state`: : station is already in connection or not
- `softap_conn_num`: : softap connection number
- `extra_info`: : extra information, such as `sta_ssid`, `softap_ssid` and etc.

`esp_err_t esp_blufi_send_wifi_list` (`uint16_t apCount`, `esp_blufi_ap_record_t *list`)

This function is called to send wifi list.

**Return** `ESP_OK` - success, other - failed

**Parameters**

- `apCount`: : wifi list count
- `list`: : wifi list

`uint16_t esp_blufi_get_version` (`void`)

Get BLUFI profile version.

**Return** Most 8bit significant is Great version, Least 8bit is Sub version

`esp_err_t esp_blufi_close` (`esp_gatt_if_t gatts_if`, `uint16_t conn_id`)

Close a connection a remote device.

**Return**

- `ESP_OK` : success
- other : failed

**Parameters**

- [in] `gatts_if`: GATT server access interface
- [in] `conn_id`: connection ID to be closed.

`esp_err_t esp_blufi_send_error_info` (`esp_blufi_error_state_t state`)

This function is called to send blufi error information.

**Return** `ESP_OK` - success, other - failed

**Parameters**

- `state`: : error state

`esp_err_t esp_blufi_send_custom_data` (`uint8_t *data`, `uint32_t data_len`)

This function is called to custom data.

**Return** `ESP_OK` - success, other - failed

**Parameters**

- `data`: : custom data value
- `data_len`: : the length of custom data

## Unions

`union esp_blufi_cb_param_t`

`#include <esp_blufi_api.h>` BLUFI callback parameters union.

### Public Members

`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_rcv_sta_bssid_evt_param sta_bssid
    Blufi callback param of ESP_BLUFI_EVENT_RECV_STA_BSSID
struct esp_blufi_cb_param_t::blufi_rcv_sta_ssid_evt_param sta_ssid
    Blufi callback param of ESP_BLUFI_EVENT_RECV_STA_SSID
struct esp_blufi_cb_param_t::blufi_rcv_sta_passwd_evt_param sta_passwd
    Blufi callback param of ESP_BLUFI_EVENT_RECV_STA_PASSWD
struct esp_blufi_cb_param_t::blufi_rcv_softap_ssid_evt_param softap_ssid
    Blufi callback param of ESP_BLUFI_EVENT_RECV_SOFTAP_SSID
struct esp_blufi_cb_param_t::blufi_rcv_softap_passwd_evt_param softap_passwd
    Blufi callback param of ESP_BLUFI_EVENT_RECV_SOFTAP_PASSWD
struct esp_blufi_cb_param_t::blufi_rcv_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_rcv_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_rcv_softap_channel_evt_param softap_channel
    Blufi callback param of ESP_BLUFI_EVENT_RECV_SOFTAP_CHANNEL
struct esp_blufi_cb_param_t::blufi_rcv_username_evt_param username
    Blufi callback param of ESP_BLUFI_EVENT_RECV_USERNAME
struct esp_blufi_cb_param_t::blufi_rcv_ca_evt_param ca
    Blufi callback param of ESP_BLUFI_EVENT_RECV_CA_CERT
struct esp_blufi_cb_param_t::blufi_rcv_client_cert_evt_param client_cert
    Blufi callback param of ESP_BLUFI_EVENT_RECV_CLIENT_CERT
struct esp_blufi_cb_param_t::blufi_rcv_server_cert_evt_param server_cert
    Blufi callback param of ESP_BLUFI_EVENT_RECV_SERVER_CERT
struct esp_blufi_cb_param_t::blufi_rcv_client_pkey_evt_param client_pkey
    Blufi callback param of ESP_BLUFI_EVENT_RECV_CLIENT_PRIV_KEY
struct esp_blufi_cb_param_t::blufi_rcv_server_pkey_evt_param server_pkey
    Blufi callback param of ESP_BLUFI_EVENT_RECV_SERVER_PRIV_KEY
struct esp_blufi_cb_param_t::blufi_get_error_evt_param report_error
    Blufi callback param of ESP_BLUFI_EVENT_REPORT_ERROR
struct esp_blufi_cb_param_t::blufi_rcv_custom_data_evt_param custom_data
    Blufi callback param of ESP_BLUFI_EVENT_RECV_CUSTOM_DATA

struct blufi_connect_evt_param
    #include <esp_blufi_api.h> ESP_BLUFI_EVENT_CONNECT.

```

### Public Members

```

esp_bd_addr_t remote_bda
    Blufi Remote bluetooth device address

uint8_t server_if
    server interface

uint16_t conn_id
    Connection id

struct blufi_deinit_finish_evt_param
    #include <esp_blufi_api.h> ESP_BLUFI_EVENT_DEINIT_FINISH.

```

**Public Members**

*esp\_blufi\_deinit\_state\_t* **state**  
De-initial status

**struct blufi\_disconnect\_evt\_param**  
*#include <esp\_blufi\_api.h>* ESP\_BLUFI\_EVENT\_DISCONNECT.

**Public Members**

*esp\_bd\_addr\_t* **remote\_bda**  
Blufi Remote bluetooth device address

**struct blufi\_get\_error\_evt\_param**  
*#include <esp\_blufi\_api.h>* ESP\_BLUFI\_EVENT\_REPORT\_ERROR.

**Public Members**

*esp\_blufi\_error\_state\_t* **state**  
Blufi error state

**struct blufi\_init\_finish\_evt\_param**  
*#include <esp\_blufi\_api.h>* ESP\_BLUFI\_EVENT\_INIT\_FINISH.

**Public Members**

*esp\_blufi\_init\_state\_t* **state**  
Initial status

**struct blufi\_recv\_ca\_evt\_param**  
*#include <esp\_blufi\_api.h>* ESP\_BLUFI\_EVENT\_RECV\_CA\_CERT.

**Public Members**

*uint8\_t* \***cert**  
CA certificate point

int **cert\_len**  
CA certificate length

**struct blufi\_recv\_client\_cert\_evt\_param**  
*#include <esp\_blufi\_api.h>* ESP\_BLUFI\_EVENT\_RECV\_CLIENT\_CERT

**Public Members**

*uint8\_t* \***cert**  
Client certificate point

int **cert\_len**  
Client certificate length

**struct blufi\_recv\_client\_pkey\_evt\_param**  
*#include <esp\_blufi\_api.h>* ESP\_BLUFI\_EVENT\_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**

`wifi_auth_mode_t auth_mode`  
Authentication mode

**struct blufi\_recv\_softap\_channel\_evt\_param**  
*#include <esp\_blufi\_api.h> ESP\_BLUFI\_EVENT\_RECV\_SOFTAP\_CHANNEL.*

**Public Members**

`uint8_t channel`  
Authentication mode

```
struct blufi_recv_softap_max_conn_num_evt_param  
#include <esp_blufi_api.h> ESP_BLUFI_EVENT_RECV_SOFTAP_MAX_CONN_NUM.
```

#### Public Members

```
int max_conn_num  
SSID
```

```
struct blufi_recv_softap_passwd_evt_param  
#include <esp_blufi_api.h> ESP_BLUFI_EVENT_RECV_SOFTAP_PASSWD.
```

#### Public Members

```
uint8_t *passwd  
Password  
  
int passwd_len  
Password Length
```

```
struct blufi_recv_softap_ssid_evt_param  
#include <esp_blufi_api.h> ESP_BLUFI_EVENT_RECV_SOFTAP_SSID.
```

#### Public Members

```
uint8_t *ssid  
SSID  
  
int ssid_len  
SSID length
```

```
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

**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

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

*esp\_blufi\_decrypt\_func\_t* **decrypt\_func**  
BLUFI decrypt data function with share key generated by negotiate\_data\_handler

*esp\_blufi\_checksum\_func\_t* **checksum\_func**  
BLUFI check sum function (FCS)

### Type Definitions

**typedef** void (\***esp\_blufi\_event\_cb\_t**) (*esp\_blufi\_cb\_event\_t* event, *esp\_blufi\_cb\_param\_t* \*param)

BLUFI event callback function type.

### Parameters

- event: : Event type
- 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.

**Parameters**

- `data`: : data from phone
- `len`: : length of data from phone
- `output_data`: : data want to send to phone
- `output_len`: : length of data want to send to phone
- `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.

**Return** Nonnegative number is encrypted length, if error, return negative number;

**Parameters**

- `iv8`: : initial vector(8bit), normally, blufi core will input packet sequence number
- `crypt_data`: : plain text and encrypted data, the encrypt function must support autochthonous encrypt
- `crypt_len`: : length of plain text

**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.

**Return** Nonnegative number is decrypted length, if error, return negative number;

**Parameters**

- `iv8`: : initial vector(8bit), normally, blufi core will input packet sequence number
- `crypt_data`: : encrypted data and plain text, the encrypt function must support autochthonous decrypt
- `crypt_len`: : length of encrypted text

**typedef** uint16\_t (\***esp\_blufi\_checksum\_func\_t**) (uint8\_t iv8, uint8\_t \*data, int len)  
BLUFI checksum.

**Parameters**

- `iv8`: : initial vector(8bit), normally, blufi core will input packet sequence number
- `data`: : data need to checksum
- `len`: : length of data

**Enumerations**

**enum** **esp\_blufi\_cb\_event\_t**

*Values:*

```
ESP_BLUFI_EVENT_INIT_FINISH = 0
ESP_BLUFI_EVENT_DEINIT_FINISH
ESP_BLUFI_EVENT_SET_WIFI_OPMODE
ESP_BLUFI_EVENT_BLE_CONNECT
ESP_BLUFI_EVENT_BLE_DISCONNECT
ESP_BLUFI_EVENT_REQ_CONNECT_TO_AP
ESP_BLUFI_EVENT_REQ_DISCONNECT_FROM_AP
ESP_BLUFI_EVENT_GET_WIFI_STATUS
ESP_BLUFI_EVENT_DEAUTHENTICATE_STA
ESP_BLUFI_EVENT_RECV_STA_BSSID
ESP_BLUFI_EVENT_RECV_STA_SSID
ESP_BLUFI_EVENT_RECV_STA_PASSWD
ESP_BLUFI_EVENT_RECV_SOFTAP_SSID
ESP_BLUFI_EVENT_RECV_SOFTAP_PASSWD
ESP_BLUFI_EVENT_RECV_SOFTAP_MAX_CONN_NUM
```

```
ESP_BLUFI_EVENT_RECV_SOFTAP_AUTH_MODE
ESP_BLUFI_EVENT_RECV_SOFTAP_CHANNEL
ESP_BLUFI_EVENT_RECV_USERNAME
ESP_BLUFI_EVENT_RECV_CA_CERT
ESP_BLUFI_EVENT_RECV_CLIENT_CERT
ESP_BLUFI_EVENT_RECV_SERVER_CERT
ESP_BLUFI_EVENT_RECV_CLIENT_PRIV_KEY
ESP_BLUFI_EVENT_RECV_SERVER_PRIV_KEY
ESP_BLUFI_EVENT_RECV_SLAVE_DISCONNECT_BLE
ESP_BLUFI_EVENT_GET_WIFI_LIST
ESP_BLUFI_EVENT_REPORT_ERROR
ESP_BLUFI_EVENT_RECV_CUSTOM_DATA
enum esp_blufi_sta_conn_state_t
    BLUFI config status.
```

*Values:*

```
ESP_BLUFI_STA_CONN_SUCCESS = 0x00
```

```
ESP_BLUFI_STA_CONN_FAIL = 0x01
```

```
enum esp_blufi_init_state_t
    BLUFI init status.
```

*Values:*

```
ESP_BLUFI_INIT_OK = 0
```

```
ESP_BLUFI_INIT_FAILED
```

```
enum esp_blufi_deinit_state_t
    BLUFI deinit status.
```

*Values:*

```
ESP_BLUFI_DEINIT_OK = 0
```

```
ESP_BLUFI_DEINIT_FAILED
```

```
enum esp_blufi_error_state_t
```

*Values:*

```
ESP_BLUFI_SEQUENCE_ERROR = 0
```

```
ESP_BLUFI_CHECKSUM_ERROR
```

```
ESP_BLUFI_DECRYPT_ERROR
```

```
ESP_BLUFI_ENCRYPT_ERROR
```

```
ESP_BLUFI_INIT_SECURITY_ERROR
```

```
ESP_BLUFI_DH_MALLOC_ERROR
```

```
ESP_BLUFI_DH_PARAM_ERROR
```

```
ESP_BLUFI_READ_PARAM_ERROR
```

```
ESP_BLUFI_MAKE_PUBLIC_ERROR
```

```
ESP_BLUFI_DATA_FORMAT_ERROR
```

## 2.1.4 CLASSIC BT

### CLASSIC BLUETOOTH GAP API

[Overview](#) [Instructions](#)

[Application Example](#) [Instructions](#)

#### API Reference

##### Header File

- [bt/host/bluedroid/api/include/api/esp\\_gap\\_bt\\_api.h](#)

##### Functions

**static** uint32\_t **esp\_bt\_gap\_get\_cod\_srvc** (uint32\_t *cod*)  
get major service field of COD

**Return** major service bits

**Parameters**

- [in] *cod*: Class of Device

**static** uint32\_t **esp\_bt\_gap\_get\_cod\_major\_dev** (uint32\_t *cod*)  
get major device field of COD

**Return** major device bits

**Parameters**

- [in] *cod*: Class of Device

**static** uint32\_t **esp\_bt\_gap\_get\_cod\_minor\_dev** (uint32\_t *cod*)  
get minor service field of COD

**Return** minor service bits

**Parameters**

- [in] *cod*: Class of Device

**static** uint32\_t **esp\_bt\_gap\_get\_cod\_format\_type** (uint32\_t *cod*)  
get format type of COD

**Return** format type

**Parameters**

- [in] *cod*: Class of Device

**static** bool **esp\_bt\_gap\_is\_valid\_cod** (uint32\_t *cod*)  
decide the integrity of COD

**Return**

- true if *cod* is valid
- false otherwise

**Parameters**

- [in] *cod*: Class of Device

*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

**Return**

- 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.

**Return**

- ESP\_OK : Succeed
- ESP\_ERR\_INVALID\_ARG: if argument invalid
- ESP\_ERR\_INVALID\_STATE: if bluetooth stack is not yet enabled
- ESP\_FAIL: others

**Parameters**

- [in] *c\_mode*: : one of the enums of `esp_bt_connection_mode_t`
- [in] *d\_mode*: : one of the enums of `esp_bt_discovery_mode_t`

*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. It 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.

**Return**

- 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

**Parameters**

- [in] *mode*: - Inquiry mode
- [in] *inq\_len*: - 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.
- [in] *num\_rsps*: - Number of responses that can be received before the Inquiry is halted, value 0 indicates an unlimited number of responses.

*esp\_err\_t* **esp\_bt\_gap\_cancel\_discovery** (void)

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.

**Return**

- 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.

**Return**

- 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

**Return**

- ESP\_OK : Succeed
- ESP\_ERR\_INVALID\_STATE: if bluetooth stack is not yet enabled
- ESP\_FAIL: others

`uint8_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.

**Return** pointer of starting position of eir data excluding eir data type, NULL if not found

**Parameters**

- [in] eir: - pointer of raw eir data to be resolved
- [in] type: - specific EIR data type
- [out] length: - return the length of EIR data excluding fields of length and data type

`esp_err_t esp_bt_gap_config_eir_data (esp_bt_eir_data_t *eir_data)`

This function is called to config EIR data.

esp\_bt\_gap\_cb\_t will be called with ESP\_BT\_GAP\_CONFIG\_EIR\_DATA\_EVT after config EIR ends.

**Return**

- 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

**Parameters**

- [in] eir\_data: - pointer of EIR data content

`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. esp\_bt\_gap\_cb\_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.

**Return**

- 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

**Parameters**

- [in] cod: - class of device
- [in] mode: - setting mode

`esp_err_t esp_bt_gap_get_cod (esp_bt_cod_t *cod)`

This function is called to get class of device.

**Return**

- ESP\_OK : Succeed
- ESP\_FAIL: others

**Parameters**

- [out] cod: - class of device

`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.

**Return**

- ESP\_OK : Succeed
- ESP\_FAIL: others

**Parameters**

- [in] remote\_addr: - remote device address, corresponding to a certain connection handle.

`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.

**Return** - ESP\_OK : success

- ESP\_FAIL : failed

**Parameters**

- [in] `bd_addr`: : BD address of the peer device

int `esp_bt_gap_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.

**Return** - `>= 0` : bonded devices number.

- `ESP_FAIL` : failed

*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.

**Return**

- `ESP_OK` : Succeed
- `ESP_ERR_INVALID_STATE`: if bluetooth stack is not yet enabled
- `ESP_FAIL`: others

**Parameters**

- [inout] `dev_num`: 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()`.
- [out] `dev_list`: 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.

*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.

**Return** - `ESP_OK` : success

- `ESP_ERR_INVALID_STATE`: if bluetooth stack is not yet enabled
- other : failed

**Parameters**

- [in] `pin_type`: 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.
- [in] `pin_code_len`: Length of `pin_code`
- [in] `pin_code`: Pin\_code

*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.

**Return** - `ESP_OK` : success

- `ESP_ERR_INVALID_STATE`: if bluetooth stack is not yet enabled
- other : failed

**Parameters**

- [in] `bd_addr`: BD address of the peer
- [in] `accept`: Pin\_code reply successful or declined.
- [in] `pin_code_len`: Length of `pin_code`
- [in] `pin_code`: Pin\_code

*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.

**Return** - `ESP_OK` : success

- `ESP_ERR_INVALID_STATE`: if bluetooth stack is not yet enabled
- other : failed

**Parameters**

- [in] `param_type`: : the type of the param which is to be set
- [in] `value`: : the param value
- [in] `len`: : the length of the param value



*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.

**Return** - ESP\_OK : success

- ESP\_ERR\_INVALID\_STATE: if bluetooth stack is not yet enabled
- other : failed

**Parameters**

- [in] *bd\_addr*: : BD address of the peer
- [in] *accept*: : passkey entry successful or declined.
- [in] *passkey*: : passkey value, must be a 6 digit number, can be lead by 0.

*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.

**Return** - ESP\_OK : success

- ESP\_ERR\_INVALID\_STATE: if bluetooth stack is not yet enabled
- other : failed

**Parameters**

- [in] *bd\_addr*: : BD address of the peer device
- [in] *accept*: : numbers to compare are the same or different.

*esp\_err\_t* **esp\_bt\_gap\_set\_qos** (*esp\_bd\_addr\_t* *remote\_bda*, uint32\_t *t\_poll*)

Config Quality of service.

**Return** - ESP\_OK : success

- ESP\_ERR\_INVALID\_STATE: if bluetooth stack is not yet enabled
- other : failed

**Parameters**

- [in] *remote\_bda*: The remote device' s address
- [in] *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

## Unions

**union** *esp\_bt\_gap\_cb\_param\_t*

*#include <esp\_gap\_bt\_api.h>* A2DP 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\_srvc\_rec\_param* **rmt\_srvc\_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::key_notif_param key_notif
    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::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**

```

esp_bd_addr_t bda
    remote bluetooth device address

uint32_t num_val
    the numeric value for comparison.

struct config_eir_data_param
    #include <esp_gap_bt_api.h> ESP_BT_GAP_CONFIG_EIR_DATA_EVT *.

```

**Public Members**

```

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

```

`uint8_t eir_type_num`  
the number of EIR types in EIR type

`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

`esp_bd_addr_t bda`  
remote bluetooth device address

`int num_prop`  
number of properties got

`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

`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.

**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

**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

`bool min_16_digit`  
TRUE if the pin returned must be at least 16 digits

```
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

*esp\_bd\_addr\_t* **bda**  
remote bluetooth device address

*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.

```
struct read_rssi_delta_param  
#include <esp_gap_bt_api.h> ESP_BT_GAP_READ_RSSI_DELTA_EVT *.
```

### Public Members

*esp\_bd\_addr\_t* **bda**  
remote bluetooth device address

*esp\_bt\_status\_t* **stat**  
read rssi status

*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

```
struct rmt_srvc_rec_param  
#include <esp_gap_bt_api.h> ESP_BT_GAP_RMT_SRVC_REC_EVT.
```

### Public Members

*esp\_bd\_addr\_t* **bda**  
remote bluetooth device address

*esp\_bt\_status\_t* **stat**  
service search status

```
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

## Structures

**struct esp\_bt\_cod\_t**

Class of device.

### Public Members

uint32\_t **reserved\_2** : 2

undefined

uint32\_t **minor** : 6

minor class

uint32\_t **major** : 5

major class

uint32\_t **service** : 11

service class

uint32\_t **reserved\_8** : 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

**ESP\_BT\_GAP\_RSSI\_LOW\_THRLD**

Low RSSI threshold

**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\_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_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 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

#### Parameters

- event: : Event type
- param: : Pointer to callback parameter

### Enumerations

```
enum esp_bt_cod_mode_t
```

class of device settings

Values:

**ESP\_BT\_SET\_COD\_MAJOR\_MINOR** = 0x01  
 overwrite major, minor class

**ESP\_BT\_SET\_COD\_SERVICE\_CLASS** = 0x02  
 set the bits in the input, the current bit will remain

**ESP\_BT\_CLR\_COD\_SERVICE\_CLASS** = 0x04  
 clear the bits in the input, others will remain

**ESP\_BT\_SET\_COD\_ALL** = 0x08  
 overwrite major, minor, set the bits in service class

**ESP\_BT\_INIT\_COD** = 0x0a  
 overwrite major, minor, and service class

**enum esp\_bt\_connection\_mode\_t**  
 Discoverability and Connectability mode.

*Values:*

**ESP\_BT\_NON\_CONNECTABLE**  
 Non-connectable

**ESP\_BT\_CONNECTABLE**  
 Connectable

**enum esp\_bt\_discovery\_mode\_t**

*Values:*

**ESP\_BT\_NON\_DISCOVERABLE**  
 Non-discoverable

**ESP\_BT\_LIMITED\_DISCOVERABLE**  
 Limited Discoverable

**ESP\_BT\_GENERAL\_DISCOVERABLE**  
 General Discoverable

**enum esp\_bt\_gap\_dev\_prop\_type\_t**

Bluetooth Device Property type.

*Values:*

**ESP\_BT\_GAP\_DEV\_PROP\_BDNAME** = 1  
 Bluetooth device name, value type is int8\_t []

**ESP\_BT\_GAP\_DEV\_PROP\_COD**  
 Class of Device, value type is uint32\_t

**ESP\_BT\_GAP\_DEV\_PROP\_RSSI**  
 Received Signal strength Indication, value type is int8\_t, ranging from -128 to 127

**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, mutiple bits can be set.

*Values:*

**ESP\_BT\_COD\_SRVC\_NONE** = 0  
 None indicates an invalid value

**ESP\_BT\_COD\_SRVC\_LMTD\_DISCOVER** = 0x1  
 Limited Discoverable Mode

**ESP\_BT\_COD\_SRVC\_POSITIONING** = 0x8  
 Positioning (Location identification)

**ESP\_BT\_COD\_SRVC\_NETWORKING** = 0x10  
 Networking, e.g. LAN, Ad hoc



**ESP\_BT\_COD\_SRVC\_RENDERING** = 0x20  
Rendering, e.g. Printing, Speakers

**ESP\_BT\_COD\_SRVC\_CAPTURING** = 0x40  
Capturing, e.g. Scanner, Microphone

**ESP\_BT\_COD\_SRVC\_OBJ\_TRANSFER** = 0x80  
Object Transfer, e.g. v-Inbox, v-Folder

**ESP\_BT\_COD\_SRVC\_AUDIO** = 0x100  
Audio, e.g. Speaker, Microphone, Headset service

**ESP\_BT\_COD\_SRVC\_TELEPHONY** = 0x200  
Telephony, e.g. Cordless telephony, Modem, Headset service

**ESP\_BT\_COD\_SRVC\_INFORMATION** = 0x400  
Information, e.g., WEB-server, WAP-server

**enum esp\_bt\_pin\_type\_t**

*Values:*

**ESP\_BT\_PIN\_TYPE\_VARIABLE** = 0  
Refer to BTM\_PIN\_TYPE\_VARIABLE

**ESP\_BT\_PIN\_TYPE\_FIXED** = 1  
Refer to BTM\_PIN\_TYPE\_FIXED

**enum esp\_bt\_sp\_param\_t**

*Values:*

**ESP\_BT\_SP\_IOCAP\_MODE** = 0  
Set IO mode

**enum esp\_bt\_cod\_major\_dev\_t**

Major device class field of Class of Device.

*Values:*

**ESP\_BT\_COD\_MAJOR\_DEV\_MISC** = 0  
Miscellaneous

**ESP\_BT\_COD\_MAJOR\_DEV\_COMPUTER** = 1  
Computer

**ESP\_BT\_COD\_MAJOR\_DEV\_PHONE** = 2  
Phone(cellular, cordless, pay phone, modem)

**ESP\_BT\_COD\_MAJOR\_DEV\_LAN\_NAP** = 3  
LAN, Network Access Point

**ESP\_BT\_COD\_MAJOR\_DEV\_AV** = 4  
Audio/Video(headset, speaker, stereo, video display, VCR)

**ESP\_BT\_COD\_MAJOR\_DEV\_PERIPHERAL** = 5  
Peripheral(mouse, joystick, keyboard)

**ESP\_BT\_COD\_MAJOR\_DEV\_IMAGING** = 6  
Imaging(printer, scanner, camera, display)

**ESP\_BT\_COD\_MAJOR\_DEV\_WEARABLE** = 7  
Wearable

**ESP\_BT\_COD\_MAJOR\_DEV\_TOY** = 8  
Toy

**ESP\_BT\_COD\_MAJOR\_DEV\_HEALTH** = 9  
Health

**ESP\_BT\_COD\_MAJOR\_DEV\_UNCATEGORIZED** = 31  
Uncategorized: device not specified

**enum esp\_bt\_gap\_discovery\_state\_t**  
Bluetooth Device Discovery state

*Values:*

**ESP\_BT\_GAP\_DISCOVERY\_STOPPED**  
device discovery stopped

**ESP\_BT\_GAP\_DISCOVERY\_STARTED**  
device discovery started

**enum esp\_bt\_gap\_cb\_event\_t**  
BT GAP callback events.

*Values:*

**ESP\_BT\_GAP\_DISC\_RES\_EVT = 0**  
device discovery result event

**ESP\_BT\_GAP\_DISC\_STATE\_CHANGED\_EVT**  
discovery state changed event

**ESP\_BT\_GAP\_RMT\_SRVCS\_EVT**  
get remote services event

**ESP\_BT\_GAP\_RMT\_SRVC\_REC\_EVT**  
get remote service record event

**ESP\_BT\_GAP\_AUTH\_CMPL\_EVT**  
AUTH complete event

**ESP\_BT\_GAP\_PIN\_REQ\_EVT**  
Legacy Pairing Pin code request

**ESP\_BT\_GAP\_CFM\_REQ\_EVT**  
Simple Pairing User Confirmation request.

**ESP\_BT\_GAP\_KEY\_NOTIF\_EVT**  
Simple Pairing Passkey Notification

**ESP\_BT\_GAP\_KEY\_REQ\_EVT**  
Simple Pairing Passkey request

**ESP\_BT\_GAP\_READ\_RSSI\_DELTA\_EVT**  
read rssi event

**ESP\_BT\_GAP\_CONFIG\_EIR\_DATA\_EVT**  
config EIR data event

**ESP\_BT\_GAP\_REMOVE\_BOND\_DEV\_COMPLETE\_EVT**  
remove bond device complete event

**ESP\_BT\_GAP\_QOS\_CMPL\_EVT**  
QOS complete event

**ESP\_BT\_GAP\_EVT\_MAX**

**enum esp\_bt\_inq\_mode\_t**  
Inquiry Mode

*Values:*

**ESP\_BT\_INQ\_MODE\_GENERAL\_INQUIRY**  
General inquiry mode

**ESP\_BT\_INQ\_MODE\_LIMITED\_INQUIRY**  
Limited inquiry mode

## Bluetooth A2DP API

### Overview [Instructions](#)

**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

- `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.

##### Return

- `ESP_OK`: success
- `ESP_INVALID_STATE`: if bluetooth stack is not yet enabled
- `ESP_FAIL`: if callback is a NULL function pointer

##### Parameters

- [in] `callback`: A2DP event callback function

*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`.

##### Return

- `ESP_OK`: success
- `ESP_INVALID_STATE`: if bluetooth stack is not yet enabled
- `ESP_FAIL`: if callback is a NULL function pointer

##### Parameters

- [in] `callback`: A2DP sink data callback function

*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.

##### Return

- `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.

##### Return

- `ESP_OK`: success
- `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, must after esp\_a2d\_sink\_init()

**Return**

- ESP\_OK: connect request is sent to lower layer
- ESP\_INVALID\_STATE: if bluetooth stack is not yet enabled
- ESP\_FAIL: others

**Parameters**

- [in] remote\_bda: remote bluetooth device address

*esp\_err\_t* **esp\_a2d\_sink\_disconnect** (*esp\_bd\_addr\_t* remote\_bda)

Disconnect from the remote A2DP source device.

**Return**

- ESP\_OK: disconnect request is sent to lower layer
- ESP\_INVALID\_STATE: if bluetooth stack is not yet enabled
- ESP\_FAIL: others

**Parameters**

- [in] remote\_bda: remote bluetooth device address

*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

**Return**

- ESP\_OK: control command is sent to lower layer
- ESP\_INVALID\_STATE: if bluetooth stack is not yet enabled
- ESP\_FAIL: others

**Parameters**

- [in] ctrl: control commands for A2DP data channel

*esp\_err\_t* **esp\_a2d\_source\_init** (void)

Initialize the bluetooth A2DP source 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.

**Return**

- 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\_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.

**Return**

- 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 is 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.

**Return**

- ESP\_OK: success
- ESP\_INVALID\_STATE: if bluetooth stack is not yet enabled
- ESP\_FAIL: if callback is a NULL function pointer

**Parameters**

- [in] callback: A2DP source data callback function

*esp\_err\_t* **esp\_a2d\_source\_connect** (*esp\_bd\_addr\_t* remote\_bda)  
Connect to remote A2DP sink device, must after esp\_a2d\_source\_init()

**Return**

- ESP\_OK: connect request is sent to lower layer
- ESP\_INVALID\_STATE: if bluetooth stack is not yet enabled
- ESP\_FAIL: others

**Parameters**

- [in] remote\_bda: remote bluetooth device address

*esp\_err\_t* **esp\_a2d\_source\_disconnect** (*esp\_bd\_addr\_t* remote\_bda)  
Disconnect from the remote A2DP sink device.

**Return**

- ESP\_OK: disconnect request is sent to lower layer
- ESP\_INVALID\_STATE: if bluetooth stack is not yet enabled
- ESP\_FAIL: others

**Parameters**

- [in] remote\_bda: remote bluetooth device address

**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 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\_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 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

**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****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** void (\***esp\_a2d\_cb\_t**) (*esp\_a2d\_cb\_event\_t* event, *esp\_a2d\_cb\_param\_t* \*param)

A2DP profile callback function type.

#### Parameters

- **event**: : Event type
- **param**: : Pointer to callback parameter

**typedef** void (\***esp\_a2d\_sink\_data\_cb\_t**) (**const** uint8\_t \*buf, uint32\_t len)

A2DP profile data callback function.

#### Parameters

- [**in**] **buf**: : data received from A2DP source device and is PCM format decoder from SBC decoder; buf references to a static memory block and can be overwritten by upcoming data
- [**in**] **len**: : 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.

**Return** size of bytes read successfully, if the argument len is -1, this value is ignored.

#### Parameters

- [**in**] **buf**: : buffer to be filled with PCM data stream from higher layer
- [**in**] **len**: : size(in bytes) of data block to be copied to buf. -1 is an indication to user that data buffer shall be flushed

### Enumerations

**enum** **esp\_a2d\_connection\_state\_t**

Bluetooth A2DP connection states.

*Values:***ESP\_A2D\_CONNECTION\_STATE\_DISCONNECTED** = 0

connection released

**ESP\_A2D\_CONNECTION\_STATE\_CONNECTING**

connecting remote device

**ESP\_A2D\_CONNECTION\_STATE\_CONNECTED**

connection established

**ESP\_A2D\_CONNECTION\_STATE\_DISCONNECTING**

disconnecting remote device

**enum** **esp\_a2d\_disc\_rsn\_t**

Bluetooth A2DP disconnection reason.

*Values:***ESP\_A2D\_DISC\_RSN\_NORMAL** = 0

Finished disconnection that is initiated by local or remote device

**ESP\_A2D\_DISC\_RSN\_ABNORMAL**

Abnormal disconnection caused by signal loss

**enum esp\_a2d\_audio\_state\_t**

Bluetooth A2DP datapath states.

*Values:*

**ESP\_A2D\_AUDIO\_STATE\_REMOTE\_SUSPEND = 0**

audio stream datapath suspended by remote device

**ESP\_A2D\_AUDIO\_STATE\_STOPPED**

audio stream datapath stopped

**ESP\_A2D\_AUDIO\_STATE\_STARTED**

audio stream datapath started

**enum esp\_a2d\_media\_ctrl\_ack\_t**

A2DP media control command acknowledgement code.

*Values:*

**ESP\_A2D\_MEDIA\_CTRL\_ACK\_SUCCESS = 0**

media control command is acknowledged with success

**ESP\_A2D\_MEDIA\_CTRL\_ACK\_FAILURE**

media control command is acknowledged with failure

**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:*

**ESP\_A2D\_MEDIA\_CTRL\_NONE = 0**

Not for application use, use inside stack only.

**ESP\_A2D\_MEDIA\_CTRL\_CHECK\_SRC\_RDY**

check whether AVDTP is connected, only used in A2DP source

**ESP\_A2D\_MEDIA\_CTRL\_START**

command to set up media transmission channel

**ESP\_A2D\_MEDIA\_CTRL\_STOP**

command to stop media transmission

**ESP\_A2D\_MEDIA\_CTRL\_SUSPEND**

command to suspend media transmission

**enum esp\_a2d\_init\_state\_t**

Bluetooth A2DP Initiation states.

*Values:*

**ESP\_A2D\_DEINIT\_SUCCESS = 0**

A2DP profile deinit successful event

**ESP\_A2D\_INIT\_SUCCESS**

A2DP profile deinit successful event

**enum esp\_a2d\_cb\_event\_t**

A2DP callback events.

*Values:*

**ESP\_A2D\_CONNECTION\_STATE\_EVT = 0**

connection state changed event



- ESP\_A2D\_AUDIO\_STATE\_EVT**  
audio stream transmission state changed event
- ESP\_A2D\_AUDIO\_CFG\_EVT**  
audio codec is configured, only used for A2DP SINK
- ESP\_A2D\_MEDIA\_CTRL\_ACK\_EVT**  
acknowledge event in response to media control commands
- ESP\_A2D\_PROF\_STATE\_EVT**  
indicate a2dp init&deinit complete

## BT AVRCP APIs

**Overview** Bluetooth AVRCP reference APIs.

[Instructions](#)

**Application Example** [Instructions](#)

## API Reference

### Header File

- [bt/host/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.

#### Return

- ESP\_OK: success
- ESP\_ERR\_INVALID\_STATE: if bluetooth stack is not yet enabled
- ESP\_FAIL: others

#### Parameters

- [in] `callback`: AVRCP controller callback function

*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.

#### Return

- 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 `esp_bluedroid_enable()` completes successfully. Note: AVRC cannot work independently, AVRC should be used along with A2DP and AVRC should be deinitialized before A2DP.

#### Return

- 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 AVRCP connection is established.

**Return**

- ESP\_OK: success
- ESP\_ERR\_INVALID\_STATE: if bluetooth stack is not yet enabled
- ESP\_FAIL: others

**Parameters**

- [in] *tl*: transaction label, 0 to 15, consecutive commands should use different values.
- [in] *attr\_id*: player application setting attribute IDs from one of *esp\_avrc\_ps\_attr\_ids\_t*
- [in] *value\_id*: attribute value defined for the specific player application setting attribute

*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 AVRCP connection is established.

**Return**

- ESP\_OK: success
- ESP\_ERR\_INVALID\_STATE: if bluetooth stack is not yet enabled
- ESP\_FAIL: others

**Parameters**

- [in] *tl*: transaction label, 0 to 15, consecutive commands should use different values.

*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.

**Return**

- 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

**Parameters**

- [in] *tl*: transaction label, 0 to 15, consecutive commands should use different values.
- [in] *event\_id*: id of events, e.g. ESP\_AVRC\_RN\_PLAY\_STATUS\_CHANGE, ESP\_AVRC\_RN\_TRACK\_CHANGE, etc.
- [in] *event\_parameter*: playback interval for ESP\_AVRC\_RN\_PLAY\_POS\_CHANGED; For other events, value of this parameter is ignored.

*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.

**Return**

- 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

**Parameters**

- [in] *tl*: transaction label, 0 to 15, consecutive commands should use different values.
- [in] *volume*: volume, 0 to 0x7f, means 0% to 100%

*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.

**Return**

- ESP\_OK: success
- ESP\_ERR\_INVALID\_STATE: if bluetooth stack is not yet enabled

- ESP\_FAIL: others

#### Parameters

- [in] `tl`: transaction label, 0 to 15, consecutive commands should use different values.
- [in] `attr_mask`: mask of attributes, e.g. `ESP_AVRC_MD_ATTR_ID_TITLE` | `ESP_AVRC_MD_ATTR_ID_ARTIST`.

*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.

#### Return

- ESP\_OK: success
- ESP\_ERR\_INVALID\_STATE: if bluetooth stack is not yet enabled
- ESP\_FAIL: others

#### Parameters

- [in] `tl`: transaction label, 0 to 15, consecutive commands should use different values.
- [in] `key_code`: passthrough command code, e.g. `ESP_AVRC_PT_CMD_PLAY`, `ESP_AVRC_PT_CMD_STOP`, etc.
- [in] `key_state`: passthrough command key state, `ESP_AVRC_PT_CMD_STATE_PRESSED` or `ESP_AVRC_PT_CMD_STATE_RELEASED`

*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.

#### Return

- ESP\_OK: success
- ESP\_ERR\_INVALID\_STATE: if bluetooth stack is not yet enabled
- ESP\_FAIL: others

#### Parameters

- [in] `callback`: AVRCP target callback function

*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: AVRCP cannot work independently, AVRCP should be used along with A2DP and AVRCP should be initialized before A2DP.

#### Return

- 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 `esp_bluedroid_enable()` completes successfully. Note: AVRCP cannot work independently, AVRCP should be used along with A2DP and AVRCP should be deinitialized before A2DP.

#### Return

- ESP\_OK: success
- ESP\_ERR\_INVALID\_STATE: if bluetooth stack is not yet enabled
- ESP\_FAIL: others

*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 AVRCP 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()`

#### 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 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.

#### Return

- 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 unallowed commands

**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.

**Return** 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.

#### Parameters

- [in] op: operation requested on the bit mask field
- [in] psth: pointer to passthrough command bit mask structure
- [in] cmd: passthrough command code

*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()`.

#### 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 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()`.

#### Return

- 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.

**Return** 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.

#### Parameters

- [in] `op`: operation requested on the bit mask field
- [in] `events`: pointer to event notification capability bit mask structure
- [in] `event_id`: notification event code

```
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()`.

#### Return

- `ESP_OK`: success
- `ESP_ERR_INVALID_STATE`: if bluetooth stack is not enabled or AVRCP TG is not initialized
- `ESP_ERR_INVALID_ARG`: if `evt_set` is NULL

#### Parameters

- [in] `event_id`: notification event ID that remote AVRCP CT registers
- [in] `rsp`: notification response code
- [in] `param`: parameters included in the specific notification

## Unions

```
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
```

```
union esp_avrc_ct_cb_param_t
#include <esp_avrc_api.h> AVRCP 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

    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 t1**  
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.

**Public Members**

**uint8\_t volume**  
the volume which has actually been set, range is 0 to 0x7f, means 0% to 100%

**union esp\_avrc\_tg\_cb\_param\_t**  
*#include <esp\_avrc\_api.h>* AVRC target callback parameters.

**Public Members**

**struct esp\_avrc\_tg\_cb\_param\_t::avrc\_tg\_conn\_stat\_param** **conn\_stat**  
AVRC connection status

**struct esp\_avrc\_tg\_cb\_param\_t::avrc\_tg\_rmt\_feats\_param** **rmt\_feats**  
AVRC features discovered through SDP

**struct esp\_avrc\_tg\_cb\_param\_t::avrc\_tg\_psth\_cmd\_param** **psth\_cmd**  
passthrough command

**struct esp\_avrc\_tg\_cb\_param\_t::avrc\_tg\_set\_abs\_vol\_param** **set\_abs\_vol**  
set absolute volume command targeted on audio sink

**struct esp\_avrc\_tg\_cb\_param\_t::avrc\_tg\_reg\_ntf\_param** **reg\_ntf**  
register notification

**struct avrc\_tg\_conn\_stat\_param**  
*#include <esp\_avrc\_api.h>* ESP\_AVRC\_TG\_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\_tg\_psth\_cmd\_param**  
*#include <esp\_avrc\_api.h>* ESP\_AVRC\_TG\_PASSTHROUGH\_CMD\_EVT.

**Public Members**

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

**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

**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.

**Parameters**

- `event`: : Event type
- `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.

**Parameters**

- `event`: : Event type
- `param`: : Pointer to callback parameter union

**Enumerations**

**enum esp\_avrc\_features\_t**  
 AVRC feature bit mask.

*Values:*

**ESP\_AVRC\_FEAT\_RCTG** = 0x0001  
 remote control target

**ESP\_AVRC\_FEAT\_RCCT** = 0x0002  
 remote control controller

**ESP\_AVRC\_FEAT\_VENDOR** = 0x0008  
 remote control vendor dependent commands

**ESP\_AVRC\_FEAT\_BROWSE** = 0x0010  
 use browsing channel

**ESP\_AVRC\_FEAT\_META\_DATA** = 0x0040  
 remote control metadata transfer command/response

**ESP\_AVRC\_FEAT\_ADV\_CTRL** = 0x0200  
 remote control advanced control command/response

**enum esp\_avrc\_feature\_flag\_t**  
 AVRC supported features flag retrieved in SDP record.

*Values:*

```
ESP_AVRC_FEAT_FLAG_CAT1 = 0x0001
    category 1
ESP_AVRC_FEAT_FLAG_CAT2 = 0x0002
    category 2
ESP_AVRC_FEAT_FLAG_CAT3 = 0x0004
    category 3
ESP_AVRC_FEAT_FLAG_CAT4 = 0x0008
    category 4
ESP_AVRC_FEAT_FLAG_BROWSING = 0x0040
    browsing
ESP_AVRC_FEAT_FLAG_COVER_ART_GET_IMAGE_PROP = 0x0080
    Cover Art GetImageProperties
ESP_AVRC_FEAT_FLAG_COVER_ART_GET_IMAGE = 0x0100
    Cover Art GetImage
ESP_AVRC_FEAT_FLAG_COVER_ART_GET_LINKED_THUMBNAIL = 0x0200
    Cover Art GetLinkedThumbnail
```

```
enum esp_avrc_pt_cmd_t
```

AVRC passthrough command code.

*Values:*

```
ESP_AVRC_PT_CMD_SELECT = 0x00
    select
ESP_AVRC_PT_CMD_UP = 0x01
    up
ESP_AVRC_PT_CMD_DOWN = 0x02
    down
ESP_AVRC_PT_CMD_LEFT = 0x03
    left
ESP_AVRC_PT_CMD_RIGHT = 0x04
    right
ESP_AVRC_PT_CMD_RIGHT_UP = 0x05
    right-up
ESP_AVRC_PT_CMD_RIGHT_DOWN = 0x06
    right-down
ESP_AVRC_PT_CMD_LEFT_UP = 0x07
    left-up
ESP_AVRC_PT_CMD_LEFT_DOWN = 0x08
    left-down
ESP_AVRC_PT_CMD_ROOT_MENU = 0x09
    root menu
ESP_AVRC_PT_CMD_SETUP_MENU = 0x0A
    setup menu
ESP_AVRC_PT_CMD_CONT_MENU = 0x0B
    contents menu
ESP_AVRC_PT_CMD_FAV_MENU = 0x0C
    favorite menu
ESP_AVRC_PT_CMD_EXIT = 0x0D
    exit
```

**ESP\_AVRC\_PT\_CMD\_0** = 0x20  
0

**ESP\_AVRC\_PT\_CMD\_1** = 0x21  
1

**ESP\_AVRC\_PT\_CMD\_2** = 0x22  
2

**ESP\_AVRC\_PT\_CMD\_3** = 0x23  
3

**ESP\_AVRC\_PT\_CMD\_4** = 0x24  
4

**ESP\_AVRC\_PT\_CMD\_5** = 0x25  
5

**ESP\_AVRC\_PT\_CMD\_6** = 0x26  
6

**ESP\_AVRC\_PT\_CMD\_7** = 0x27  
7

**ESP\_AVRC\_PT\_CMD\_8** = 0x28  
8

**ESP\_AVRC\_PT\_CMD\_9** = 0x29  
9

**ESP\_AVRC\_PT\_CMD\_DOT** = 0x2A  
dot

**ESP\_AVRC\_PT\_CMD\_ENTER** = 0x2B  
enter

**ESP\_AVRC\_PT\_CMD\_CLEAR** = 0x2C  
clear

**ESP\_AVRC\_PT\_CMD\_CHAN\_UP** = 0x30  
channel up

**ESP\_AVRC\_PT\_CMD\_CHAN\_DOWN** = 0x31  
channel down

**ESP\_AVRC\_PT\_CMD\_PREV\_CHAN** = 0x32  
previous channel

**ESP\_AVRC\_PT\_CMD\_SOUND\_SEL** = 0x33  
sound select

**ESP\_AVRC\_PT\_CMD\_INPUT\_SEL** = 0x34  
input select

**ESP\_AVRC\_PT\_CMD\_DISP\_INFO** = 0x35  
display information

**ESP\_AVRC\_PT\_CMD\_HELP** = 0x36  
help

**ESP\_AVRC\_PT\_CMD\_PAGE\_UP** = 0x37  
page up

**ESP\_AVRC\_PT\_CMD\_PAGE\_DOWN** = 0x38  
page down

**ESP\_AVRC\_PT\_CMD\_POWER** = 0x40  
power

**ESP\_AVRC\_PT\_CMD\_VOL\_UP** = 0x41  
volume up

**ESP\_AVRC\_PT\_CMD\_VOL\_DOWN** = 0x42  
volume down

**ESP\_AVRC\_PT\_CMD\_MUTE** = 0x43  
mute

**ESP\_AVRC\_PT\_CMD\_PLAY** = 0x44  
play

**ESP\_AVRC\_PT\_CMD\_STOP** = 0x45  
stop

**ESP\_AVRC\_PT\_CMD\_PAUSE** = 0x46  
pause

**ESP\_AVRC\_PT\_CMD\_RECORD** = 0x47  
record

**ESP\_AVRC\_PT\_CMD\_REWIND** = 0x48  
rewind

**ESP\_AVRC\_PT\_CMD\_FAST\_FORWARD** = 0x49  
fast forward

**ESP\_AVRC\_PT\_CMD\_EJECT** = 0x4A  
eject

**ESP\_AVRC\_PT\_CMD\_FORWARD** = 0x4B  
forward

**ESP\_AVRC\_PT\_CMD\_BACKWARD** = 0x4C  
backward

**ESP\_AVRC\_PT\_CMD\_ANGLE** = 0x50  
angle

**ESP\_AVRC\_PT\_CMD\_SUBPICT** = 0x51  
subpicture

**ESP\_AVRC\_PT\_CMD\_F1** = 0x71  
F1

**ESP\_AVRC\_PT\_CMD\_F2** = 0x72  
F2

**ESP\_AVRC\_PT\_CMD\_F3** = 0x73  
F3

**ESP\_AVRC\_PT\_CMD\_F4** = 0x74  
F4

**ESP\_AVRC\_PT\_CMD\_F5** = 0x75  
F5

**ESP\_AVRC\_PT\_CMD\_VENDOR** = 0x7E  
vendor unique

**enum esp\_avrc\_psth\_filter\_t**  
AVRC passthrough command filter.

*Values:*

**ESP\_AVRC\_PSTH\_FILTER\_ALLOWED\_CMD** = 0  
all of the PASSTHROUGH commands that can possibly be used, immutable

**ESP\_AVRC\_PSTH\_FILTER\_SUPPORTED\_CMD** = 1  
PASSTHROUGH commands selectively supported according to the current configuration

**ESP\_AVRC\_PSTH\_FILTER\_SUPPORT\_MAX**

**enum esp\_avrc\_bit\_mask\_op\_t**

*Values:*

**ESP\_AVRC\_BIT\_MASK\_OP\_TEST = 0**  
operation code to test a specific bit

**ESP\_AVRC\_BIT\_MASK\_OP\_SET = 1**  
operation code to set a specific bit

**ESP\_AVRC\_BIT\_MASK\_OP\_CLEAR = 2**  
operation code to clear a specific bit

**enum esp\_avrc\_pt\_cmd\_state\_t**

AVRC passthrough command state.

*Values:*

**ESP\_AVRC\_PT\_CMD\_STATE\_PRESSED = 0**  
key pressed

**ESP\_AVRC\_PT\_CMD\_STATE\_RELEASED = 1**  
key released

**enum esp\_avrc\_ct\_cb\_event\_t**

AVRC Controller callback events.

*Values:*

**ESP\_AVRC\_CT\_CONNECTION\_STATE\_EVT = 0**  
connection state changed event

**ESP\_AVRC\_CT\_PASSTHROUGH\_RSP\_EVT = 1**  
passthrough response event

**ESP\_AVRC\_CT\_METADATA\_RSP\_EVT = 2**  
metadata response event

**ESP\_AVRC\_CT\_PLAY\_STATUS\_RSP\_EVT = 3**  
play status response event

**ESP\_AVRC\_CT\_CHANGE\_NOTIFY\_EVT = 4**  
notification event

**ESP\_AVRC\_CT\_REMOTE\_FEATURES\_EVT = 5**  
feature of remote device indication event

**ESP\_AVRC\_CT\_GET\_RN\_CAPABILITIES\_RSP\_EVT = 6**  
supported notification events capability of peer device

**ESP\_AVRC\_CT\_SET\_ABSOLUTE\_VOLUME\_RSP\_EVT = 7**  
set absolute volume response event

**enum esp\_avrc\_tg\_cb\_event\_t**

AVRC Target callback events.

*Values:*

**ESP\_AVRC\_TG\_CONNECTION\_STATE\_EVT = 0**  
connection state changed event

**ESP\_AVRC\_TG\_REMOTE\_FEATURES\_EVT = 1**  
feature of remote device indication event

**ESP\_AVRC\_TG\_PASSTHROUGH\_CMD\_EVT = 2**  
passthrough command event

**ESP\_AVRC\_TG\_SET\_ABSOLUTE\_VOLUME\_CMD\_EVT = 3**  
set absolute volume command from remote device

**ESP\_AVRC\_TG\_REGISTER\_NOTIFICATION\_EVT = 4**  
register notification event

**enum esp\_avrc\_md\_attr\_mask\_t**  
AVRC metadata attribute mask.

*Values:*

**ESP\_AVRC\_MD\_ATTR\_TITLE = 0x1**  
title of the playing track

**ESP\_AVRC\_MD\_ATTR\_ARTIST = 0x2**  
track artist

**ESP\_AVRC\_MD\_ATTR\_ALBUM = 0x4**  
album name

**ESP\_AVRC\_MD\_ATTR\_TRACK\_NUM = 0x8**  
track position on the album

**ESP\_AVRC\_MD\_ATTR\_NUM\_TRACKS = 0x10**  
number of tracks on the album

**ESP\_AVRC\_MD\_ATTR\_GENRE = 0x20**  
track genre

**ESP\_AVRC\_MD\_ATTR\_PLAYING\_TIME = 0x40**  
total album playing time in milliseconds

**enum esp\_avrc\_rn\_event\_ids\_t**  
AVRC event notification ids.

*Values:*

**ESP\_AVRC\_RN\_PLAY\_STATUS\_CHANGE = 0x01**  
track status change, eg. from playing to paused

**ESP\_AVRC\_RN\_TRACK\_CHANGE = 0x02**  
new track is loaded

**ESP\_AVRC\_RN\_TRACK\_REACHED\_END = 0x03**  
current track reached end

**ESP\_AVRC\_RN\_TRACK\_REACHED\_START = 0x04**  
current track reached start position

**ESP\_AVRC\_RN\_PLAY\_POS\_CHANGED = 0x05**  
track playing position changed

**ESP\_AVRC\_RN\_BATTERY\_STATUS\_CHANGE = 0x06**  
battery status changed

**ESP\_AVRC\_RN\_SYSTEM\_STATUS\_CHANGE = 0x07**  
system status changed

**ESP\_AVRC\_RN\_APP\_SETTING\_CHANGE = 0x08**  
application settings changed

**ESP\_AVRC\_RN\_NOW\_PLAYING\_CHANGE = 0x09**  
now playing content changed

**ESP\_AVRC\_RN\_AVAILABLE\_PLAYERS\_CHANGE = 0x0a**  
available players changed

**ESP\_AVRC\_RN\_ADDRESSED\_PLAYER\_CHANGE = 0x0b**  
the addressed player changed

**ESP\_AVRC\_RN\_UIDS\_CHANGE = 0x0c**  
UIDs changed

**ESP\_AVRC\_RN\_VOLUME\_CHANGE** = 0x0d  
volume changed locally on TG

**ESP\_AVRC\_RN\_MAX\_EVT**

**enum esp\_avrc\_rn\_evt\_cap\_t**  
AVRC target notification event notification capability.

*Values:*

**ESP\_AVRC\_RN\_CAP\_ALLOWED\_EVT** = 0  
all of the notification events that can possibly be supported, immutable

**ESP\_AVRC\_RN\_CAP\_SUPPORTED\_EVT** = 1  
notification events selectively supported according to the current configuration

**ESP\_AVRC\_RN\_CAP\_MAX**

**enum esp\_avrc\_rn\_rsp\_t**  
AVRC notification response type.

*Values:*

**ESP\_AVRC\_RN\_RSP\_INTERIM** = 13  
initial response to RegisterNotification, should be sent T\_mtp(1000ms) from receiving the command

**ESP\_AVRC\_RN\_RSP\_CHANGED** = 15  
final response to RegisterNotification command

**enum esp\_avrc\_ps\_attr\_ids\_t**  
AVRC player setting ids.

*Values:*

**ESP\_AVRC\_PS\_EQUALIZER** = 0x01  
equalizer, on or off

**ESP\_AVRC\_PS\_REPEAT\_MODE** = 0x02  
repeat mode

**ESP\_AVRC\_PS\_SHUFFLE\_MODE** = 0x03  
shuffle mode

**ESP\_AVRC\_PS\_SCAN\_MODE** = 0x04  
scan mode on or off

**ESP\_AVRC\_PS\_MAX\_ATTR**

**enum esp\_avrc\_ps\_eq\_value\_ids\_t**  
AVRC equalizer modes.

*Values:*

**ESP\_AVRC\_PS\_EQUALIZER\_OFF** = 0x1  
equalizer OFF

**ESP\_AVRC\_PS\_EQUALIZER\_ON** = 0x2  
equalizer ON

**enum esp\_avrc\_ps\_rpt\_value\_ids\_t**  
AVRC repeat modes.

*Values:*

**ESP\_AVRC\_PS\_REPEAT\_OFF** = 0x1  
repeat mode off

**ESP\_AVRC\_PS\_REPEAT\_SINGLE** = 0x2  
single track repeat

**ESP\_AVRC\_PS\_REPEAT\_GROUP** = 0x3  
group repeat

**enum esp\_avrc\_ps\_shf\_value\_ids\_t**  
AVRC shuffle modes.

*Values:*

**ESP\_AVRC\_PS\_SHUFFLE\_OFF** = 0x1

**ESP\_AVRC\_PS\_SHUFFLE\_ALL** = 0x2

**ESP\_AVRC\_PS\_SHUFFLE\_GROUP** = 0x3

**enum esp\_avrc\_ps\_scn\_value\_ids\_t**  
AVRC scan modes.

*Values:*

**ESP\_AVRC\_PS\_SCAN\_OFF** = 0x1  
scan off

**ESP\_AVRC\_PS\_SCAN\_ALL** = 0x2  
all tracks scan

**ESP\_AVRC\_PS\_SCAN\_GROUP** = 0x3  
group scan

**enum esp\_avrc\_rsp\_t**  
AVCTP response codes.

*Values:*

**ESP\_AVRC\_RSP\_NOT\_IMPL** = 8  
not implemented

**ESP\_AVRC\_RSP\_ACCEPT** = 9  
accept

**ESP\_AVRC\_RSP\_REJECT** = 10  
reject

**ESP\_AVRC\_RSP\_IN\_TRANS** = 11  
in transition

**ESP\_AVRC\_RSP\_IMPL\_STBL** = 12  
implemented/stable

**ESP\_AVRC\_RSP\_CHANGED** = 13  
changed

**ESP\_AVRC\_RSP\_INTERIM** = 15  
interim

**enum esp\_avrc\_batt\_stat\_t**  
AVRCP battery status.

*Values:*

**ESP\_AVRC\_BATT\_NORMAL** = 0  
normal state

**ESP\_AVRC\_BATT\_WARNING** = 1  
unable to operate soon

**ESP\_AVRC\_BATT\_CRITICAL** = 2  
cannot operate any more

**ESP\_AVRC\_BATT\_EXTERNAL** = 3  
plugged to external power supply



**ESP\_AVRC\_BATT\_FULL\_CHARGE** = 4  
when completely charged from external power supply

**enum esp\_avrc\_playback\_stat\_t**  
AVRCP current status of playback.

*Values:*

**ESP\_AVRC\_PLAYBACK\_STOPPED** = 0  
stopped

**ESP\_AVRC\_PLAYBACK\_PLAYING** = 1  
playing

**ESP\_AVRC\_PLAYBACK\_PAUSED** = 2  
paused

**ESP\_AVRC\_PLAYBACK\_FWD\_SEEK** = 3  
forward seek

**ESP\_AVRC\_PLAYBACK\_REV\_SEEK** = 4  
reverse seek

**ESP\_AVRC\_PLAYBACK\_ERROR** = 0xFF  
error

## SPP API

### Overview [Instructions](#)

**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

- [bt/host/bluedroid/api/include/api/esp\\_spp\\_api.h](#)

#### Functions

*esp\_err\_t* **esp\_spp\_register\_callback** (*esp\_spp\_cb\_t* callback)

This function is called to init callbacks with SPP module.

##### Return

- ESP\_OK: success
- other: failed

##### Parameters

- [in] callback: pointer to the init callback function.

*esp\_err\_t* **esp\_spp\_init** (*esp\_spp\_mode\_t* mode)

This function is called to init SPP.

##### Return

- ESP\_OK: success
- other: failed

##### Parameters

- [in] mode: Choose the mode of SPP, ESP\_SPP\_MODE\_CB or ESP\_SPP\_MODE\_VFS.

***esp\_err\_t* esp\_spp\_deinit ()**

This function is called to uninit SPP.

**Return**

- ESP\_OK: success
- other: failed

***esp\_err\_t* esp\_spp\_start\_discovery (*esp\_bd\_addr\_t* bd\_addr)**

This function is called to performs service discovery for the services provided by the given peer device. When the operation is complete the callback function will be called with a ESP\_SPP\_DISCOVERY\_COMP\_EVT.

**Return**

- ESP\_OK: success
- other: failed

**Parameters**

- [in] bd\_addr: Remote device bluetooth device address.

***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.

**Return**

- ESP\_OK: success
- other: failed

**Parameters**

- [in] sec\_mask: Security Setting Mask. Suggest to use ESP\_SPP\_SEC\_NONE, ESP\_SPP\_SEC\_AUTHORIZE or ESP\_SPP\_SEC\_AUTHENTICATE only.
- [in] role: Master or slave.
- [in] remote\_scn: Remote device bluetooth device SCN.
- [in] peer\_bd\_addr: Remote device bluetooth device address.

***esp\_err\_t* esp\_spp\_disconnect (uint32\_t handle)**

This function closes an SPP connection.

**Return**

- ESP\_OK: success
- other: failed

**Parameters**

- [in] handle: The connection handle.

***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.

**Return**

- ESP\_OK: success
- other: failed

**Parameters**

- [in] sec\_mask: Security Setting Mask. Suggest to use ESP\_SPP\_SEC\_NONE, ESP\_SPP\_SEC\_AUTHORIZE or ESP\_SPP\_SEC\_AUTHENTICATE only.
- [in] role: Master or slave.
- [in] local\_scn: The specific channel you want to get. If channel is 0, means get any channel.
- [in] name: Server' s name.

***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()` successful and before `esp_spp_deinit()`.

#### Return

- ESP\_OK: success
- other: failed

*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()` successful and before `esp_spp_deinit()`.

#### Return

- ESP\_OK: success
- other: failed

#### Parameters

- [in] *scn*: Server channel number.

*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.

#### Return

- ESP\_OK: success
- other: failed

#### Parameters

- [in] *handle*: The connection handle.
- [in] *len*: The length of the data written.
- [in] *p\_data*: The data written.

*esp\_err\_t* **esp\_spp\_vfs\_register** (void)

This function is used to register VFS.

#### Return

- ESP\_OK: success
- other: failed

## Unions

**union esp\_spp\_cb\_param\_t**

*#include <esp\_spp\_api.h>* 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

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

*esp\_spp\_status\_t* **status**  
status

uint8\_t **scn\_num**  
The num of scn\_num

uint8\_t **scn**[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

*esp\_spp\_status\_t* **status**  
status

**struct spp\_open\_evt\_param**  
*#include <esp\_spp\_api.h>* ESP\_SPP\_OPEN\_EVT.

### Public Members

*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

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**  
status

```
struct spp_write_evt_param  
#include <esp_spp_api.h> ESP_SPP_WRITE_EVT.
```

### Public Members

*esp\_spp\_status\_t* **status**  
status

uint32\_t **handle**  
The connection handle

int **len**  
The length of the data written.

bool **cong**  
congestion status

### Macros

**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

**ESP\_SPP\_MAX\_MTU**  
SPP max MTU

**ESP\_SPP\_MAX\_SCN**  
SPP max SCN

### 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.
```

#### Parameters

- event: Event type
- param: Point to callback parameter, currently is union type

### Enumerations

```
enum esp_spp_status_t
Values:

ESP_SPP_SUCCESS = 0
Successful operation.

ESP_SPP_FAILURE
Generic failure.
```

**ESP\_SPP\_BUSY**  
Temporarily can not handle this request.

**ESP\_SPP\_NO\_DATA**  
No data

**ESP\_SPP\_NO\_RESOURCE**  
No more resource

**ESP\_SPP\_NEED\_INIT**  
SPP module shall init first

**ESP\_SPP\_NEED\_DEINIT**  
SPP module shall deinit first

**ESP\_SPP\_NO\_CONNECTION**  
Connection may have been closed

**ESP\_SPP\_NO\_SERVER**  
No SPP server

**enum esp\_spp\_role\_t**  
*Values:*

**ESP\_SPP\_ROLE\_MASTER = 0**  
Role: master

**ESP\_SPP\_ROLE\_SLAVE = 1**  
Role: slave

**enum esp\_spp\_mode\_t**  
*Values:*

**ESP\_SPP\_MODE\_CB = 0**  
When data is coming, a callback will come with data

**ESP\_SPP\_MODE\_VFS = 1**  
Use VFS to write/read data

**enum esp\_spp\_cb\_event\_t**  
SPP callback function events.

*Values:*

**ESP\_SPP\_INIT\_EVT = 0**  
When SPP is inited, the event comes

**ESP\_SPP\_UNINIT\_EVT = 1**  
When SPP is uninited, the event comes

**ESP\_SPP\_DISCOVERY\_COMP\_EVT = 8**  
When SDP discovery complete, the event comes

**ESP\_SPP\_OPEN\_EVT = 26**  
When SPP Client connection open, the event comes

**ESP\_SPP\_CLOSE\_EVT = 27**  
When SPP connection closed, the event comes

**ESP\_SPP\_START\_EVT = 28**  
When SPP server started, the event comes

**ESP\_SPP\_CL\_INIT\_EVT = 29**  
When SPP client initiated a connection, the event comes

**ESP\_SPP\_DATA\_IND\_EVT = 30**  
When SPP connection received data, the event comes, only for ESP\_SPP\_MODE\_CB

**ESP\_SPP\_CONG\_EVT = 31**  
When SPP connection congestion status changed, the event comes, only for ESP\_SPP\_MODE\_CB



**ESP\_SPP\_WRITE\_EVT = 33**

When SPP write operation completes, the event comes, only for ESP\_SPP\_MODE\_CB

**ESP\_SPP\_SRV\_OPEN\_EVT = 34**

When SPP Server connection open, the event comes

**ESP\_SPP\_SRV\_STOP\_EVT = 35**

When SPP server stopped, the event comes

## HFP DEFINES

### Overview [Instructions](#)

### API Reference

#### Header File

- [bt/host/bluetooth/api/include/api/esp\\_hf\\_defs.h](#)

#### Enumerations

**enum esp\_hf\_volume\_control\_target\_t**

Bluetooth HFP audio volume control target.

*Values:*

**ESP\_HF\_VOLUME\_CONTROL\_TARGET\_SPK = 0**  
speaker

**ESP\_HF\_VOLUME\_CONTROL\_TARGET\_MIC**  
microphone

**enum esp\_hf\_roaming\_status\_t**

+CIND roaming status indicator values

*Values:*

**ESP\_HF\_ROAMING\_STATUS\_INACTIVE = 0**  
roaming is not active

**ESP\_HF\_ROAMING\_STATUS\_ACTIVE**  
a roaming is active

**enum esp\_hf\_call\_status\_t**

+CIND call status indicator values

*Values:*

**ESP\_HF\_CALL\_STATUS\_NO\_CALLS = 0**  
no call in progress

**ESP\_HF\_CALL\_STATUS\_CALL\_IN\_PROGRESS = 1**  
call is present(active or held)

**enum esp\_hf\_call\_setup\_status\_t**

+CIND call setup status indicator values

*Values:*

**ESP\_HF\_CALL\_SETUP\_STATUS\_NONE = 0**  
no call setup in progress

**ESP\_HF\_CALL\_SETUP\_STATUS\_INCOMING = 1**  
incoming call setup in progress

**ESP\_HF\_CALL\_SETUP\_STATUS\_OUTGOING\_DIALING = 2**  
outgoing call setup in dialing state

**ESP\_HF\_CALL\_SETUP\_STATUS\_OUTGOING\_ALERTING = 3**  
outgoing call setup in alerting state

**enum esp\_hf\_call\_held\_status\_t**  
+CIND call held indicator values

*Values:*

**ESP\_HF\_CALL\_HELD\_STATUS\_NONE = 0**  
no calls held

**ESP\_HF\_CALL\_HELD\_STATUS\_HELD\_AND\_ACTIVE = 1**  
both active and held call

**ESP\_HF\_CALL\_HELD\_STATUS\_HELD = 2**  
call on hold, no active call

**enum esp\_hf\_service\_availability\_status\_t**  
+CIND network service availability status

*Values:*

**ESP\_HF\_SERVICE\_AVAILABILITY\_STATUS\_UNAVAILABLE = 0**  
service not available

**ESP\_HF\_SERVICE\_AVAILABILITY\_STATUS\_AVAILABLE**  
service available

**enum esp\_hf\_current\_call\_status\_t**  
+CLCC status of the call

*Values:*

**ESP\_HF\_CURRENT\_CALL\_STATUS\_ACTIVE = 0**  
active

**ESP\_HF\_CURRENT\_CALL\_STATUS\_HELD = 1**  
held

**ESP\_HF\_CURRENT\_CALL\_STATUS\_DIALING = 2**  
dialing (outgoing calls only)

**ESP\_HF\_CURRENT\_CALL\_STATUS\_ALERTING = 3**  
alerting (outgoing calls only)

**ESP\_HF\_CURRENT\_CALL\_STATUS\_INCOMING = 4**  
incoming (incoming calls only)

**ESP\_HF\_CURRENT\_CALL\_STATUS\_WAITING = 5**  
waiting (incoming calls only)

**ESP\_HF\_CURRENT\_CALL\_STATUS\_HELD\_BY\_RESP\_HOLD = 6**  
call held by response and hold

**enum esp\_hf\_current\_call\_direction\_t**  
+CLCC direction of the call

*Values:*

**ESP\_HF\_CURRENT\_CALL\_DIRECTION\_OUTGOING = 0**  
outgoing

**ESP\_HF\_CURRENT\_CALL\_DIRECTION\_INCOMING = 1**  
incoming

**enum esp\_hf\_current\_call\_mpty\_type\_t**

+CLCC multi-party call flag

*Values:***ESP\_HF\_CURRENT\_CALL\_MPTY\_TYPE\_SINGLE** = 0

not a member of a multi-party call

**ESP\_HF\_CURRENT\_CALL\_MPTY\_TYPE\_MULTI** = 1

member of a multi-party call

**enum esp\_hf\_current\_call\_mode\_t**

+CLCC call mode

*Values:***ESP\_HF\_CURRENT\_CALL\_MODE\_VOICE** = 0**ESP\_HF\_CURRENT\_CALL\_MODE\_DATA** = 1**ESP\_HF\_CURRENT\_CALL\_MODE\_FAX** = 2**enum esp\_hf\_call\_addr\_type\_t**

+CLCC address type

*Values:***ESP\_HF\_CALL\_ADDR\_TYPE\_UNKNOWN** = 0x81

unknown address type

**ESP\_HF\_CALL\_ADDR\_TYPE\_INTERNATIONAL** = 0x91

international address

**enum esp\_hf\_subscriber\_service\_type\_t**

+CNUM service type of the phone number

*Values:***ESP\_HF\_SUBSCRIBER\_SERVICE\_TYPE\_UNKNOWN** = 0

unknown

**ESP\_HF\_SUBSCRIBER\_SERVICE\_TYPE\_VOICE**

voice service

**ESP\_HF\_SUBSCRIBER\_SERVICE\_TYPE\_FAX**

fax service

**enum esp\_hf\_btrh\_status\_t**

+BTRH response and hold result code

*Values:***ESP\_HF\_BTRH\_STATUS\_HELD** = 0

incoming call is put on held in AG

**ESP\_HF\_BTRH\_STATUS\_ACCEPTED**

held incoming call is accepted in AG

**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:***ESP\_HF\_BTRH\_CMD\_HOLD** = 0

put the incoming call on hold

**ESP\_HF\_BTRH\_CMD\_ACCEPT** = 1

accept a held incoming call

**ESP\_HF\_BTRH\_CMD\_REJECT = 2**  
reject a held incoming call

**enum esp\_hf\_at\_response\_code\_t**  
response indication codes for AT commands

*Values:*

**ESP\_HF\_AT\_RESPONSE\_CODE\_OK = 0**  
acknowledges execution of a command line

**ESP\_HF\_AT\_RESPONSE\_CODE\_ERR**  
command not accepted

**ESP\_HF\_AT\_RESPONSE\_CODE\_NO\_CARRIER**  
connection terminated

**ESP\_HF\_AT\_RESPONSE\_CODE\_BUSY**  
busy signal detected

**ESP\_HF\_AT\_RESPONSE\_CODE\_NO\_ANSWER**  
connection completion timeout

**ESP\_HF\_AT\_RESPONSE\_CODE\_DELAYED**  
delayed

**ESP\_HF\_AT\_RESPONSE\_CODE\_BLACKLISTED**  
blacklisted

**ESP\_HF\_AT\_RESPONSE\_CODE\_CME**  
CME error

**enum esp\_hf\_vr\_state\_t**  
voice recognition state

*Values:*

**ESP\_HF\_VR\_STATE\_DISABLED = 0**  
voice recognition disabled

**ESP\_HF\_VR\_STATE\_ENABLED**  
voice recognition enabled

**enum esp\_hf\_chld\_type\_t**  
AT+CHLD command values.

*Values:*

**ESP\_HF\_CHLD\_TYPE\_REL = 0**  
<0>, Terminate all held or set UDUB( “busy” ) to a waiting call

**ESP\_HF\_CHLD\_TYPE\_REL\_ACC**  
<1>, Terminate all active calls and accepts a waiting/held call

**ESP\_HF\_CHLD\_TYPE\_HOLD\_ACC**  
<2>, Hold all active calls and accepts a waiting/held call

**ESP\_HF\_CHLD\_TYPE\_MERGE**  
<3>, Add all held calls to a conference

**ESP\_HF\_CHLD\_TYPE\_MERGE\_DETACH**  
<4>, connect the two calls and disconnects the subscriber from both calls

**ESP\_HF\_CHLD\_TYPE\_REL\_X**  
<1x>, releases specified calls only

**ESP\_HF\_CHLD\_TYPE\_PRIV\_X**  
<2x>, request private consultation mode with specified call

**enum esp\_hf\_cme\_err\_t**

Extended Audio Gateway Error Result Code Response.

*Values:*

- ESP\_HF\_CME\_AG\_FAILURE = 0**  
ag failure
- ESP\_HF\_CME\_NO\_CONNECTION\_TO\_PHONE = 1**  
no connection to phone
- ESP\_HF\_CME\_OPERATION\_NOT\_ALLOWED = 3**  
operation not allowed
- ESP\_HF\_CME\_OPERATION\_NOT\_SUPPORTED = 4**  
operation not supported
- ESP\_HF\_CME\_PH\_SIM\_PIN\_REQUIRED = 5**  
PH-SIM PIN Required
- ESP\_HF\_CME\_SIM\_NOT\_INSERTED = 10**  
SIM not inserted
- ESP\_HF\_CME\_SIM\_PIN\_REQUIRED = 11**  
SIM PIN required
- ESP\_HF\_CME\_SIM\_PUK\_REQUIRED = 12**  
SIM PUK required
- ESP\_HF\_CME\_SIM\_FAILURE = 13**  
SIM failure
- ESP\_HF\_CME\_SIM\_BUSY = 14**  
SIM busy
- ESP\_HF\_CME\_INCORRECT\_PASSWORD = 16**  
incorrect password
- ESP\_HF\_CME\_SIM\_PIN2\_REQUIRED = 17**  
SIM PIN2 required
- ESP\_HF\_CME\_SIM\_PUK2\_REQUIRED = 18**  
SIM PUK2 required
- ESP\_HF\_CME\_MEMEORY\_FULL = 20**  
memory full
- ESP\_HF\_CME\_INVALID\_INDEX = 21**  
invalid index
- ESP\_HF\_CME\_MEMEORY\_FAILURE = 23**  
memory failure
- ESP\_HF\_CME\_TEXT\_STRING\_TOO\_LONG = 24**  
test string too long
- ESP\_HF\_CME\_INVALID\_CHARACTERS\_IN\_TEXT\_STRING = 25**  
invalid characters in text string
- ESP\_HF\_CME\_DIAL\_STRING\_TOO\_LONG = 26**  
dial string too long
- ESP\_HF\_CME\_INVALID\_CHARACTERS\_IN\_DIAL\_STRING = 27**  
invalid characters in dial string
- ESP\_HF\_CME\_NO\_NETWORK\_SERVICE = 30**  
no network service
- ESP\_HF\_CME\_NETWORK\_TIMEOUT = 31**  
network timeout

**ESP\_HF\_CME\_NETWORK\_NOT\_ALLOWED = 32**  
network not allowed emergency calls only

## HFP CLIENT API

**Overview** [Instructions](#)

### API Reference

#### Header File

- [bt/host/bluedroid/api/include/api/esp\\_hf\\_client\\_api.h](#)

#### Functions

*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, used by HFP client.

##### Return

- ESP\_OK: success
- ESP\_INVALID\_STATE: if bluetooth stack is not yet enabled
- ESP\_FAIL: if callback is a NULL function pointer

##### Parameters

- [in] callback: HFP client event callback function

*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.

##### Return

- 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\_client\_deinit** (void)

De-initialize for HFP client module. This function should be called only after `esp_bluedroid_enable()` completes successfully.

##### Return

- 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)

Connect to remote bluetooth HFP audio gateway(AG) device, must after `esp_hf_client_init()`

##### Return

- ESP\_OK: connect request is sent to lower layer
- ESP\_INVALID\_STATE: if bluetooth stack is not yet enabled
- ESP\_FAIL: others

##### Parameters

- [in] remote\_bda: remote bluetooth device address

*esp\_err\_t* **esp\_hf\_client\_disconnect** (*esp\_bd\_addr\_t* remote\_bda)

Disconnect from the remote HFP audio gateway.

##### Return

- ESP\_OK: disconnect request is sent to lower layer
- ESP\_INVALID\_STATE: if bluetooth stack is not yet enabled
- ESP\_FAIL: others

##### Parameters

- [in] remote\_bda: remote bluetooth device address

*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.

**Return**

- ESP\_OK: disconnect request is sent to lower layer
- ESP\_INVALID\_STATE: if bluetooth stack is not yet enabled
- ESP\_FAIL: others

**Parameters**

- [in] remote\_bda: remote bluetooth device address

*esp\_err\_t* **esp\_hf\_client\_disconnect\_audio** (*esp\_bd\_addr\_t* remote\_bda)

Release the established audio connection with remote HFP AG.

**Return**

- ESP\_OK: disconnect request is sent to lower layer
- ESP\_INVALID\_STATE: if bluetooth stack is not yet enabled
- ESP\_FAIL: others

**Parameters**

- [in] remote\_bda: remote bluetooth device address

*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.

**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\_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.

**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\_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.

**Return**

- ESP\_OK: disconnect request is sent to lower layer
- ESP\_INVALID\_STATE: if bluetooth stack is not yet enabled
- ESP\_FAIL: others

**Parameters**

- [in] type: volume control target, speaker or microphone
- [in] volume: gain of the speaker of microphone, ranges 0 to 15

*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.

**Return**

- ESP\_OK: disconnect request is sent to lower layer
- ESP\_INVALID\_STATE: if bluetooth stack is not yet enabled
- ESP\_FAIL: others

**Parameters**

- [in] number: number string of the call. If NULL, the last number is called(aka re-dial)

***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.

**Return**

- ESP\_OK: disconnect request is sent to lower layer
- ESP\_INVALID\_STATE: if bluetooth stack is not yet enabled
- ESP\_FAIL: others

**Parameters**

- [in] *location*: location of the number in the memory

***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.

**Return**

- ESP\_OK: disconnect request is sent to lower layer
- ESP\_INVALID\_STATE: if bluetooth stack is not yet enabled
- ESP\_FAIL: others

**Parameters**

- [in] *chld*: AT+CHLD call hold and multiparty handling AT command.
- [in] *idx*: used in Enhanced Call Control Mechanisms, used if *chld* is ESP\_HF\_CHLD\_TYPE\_REL\_X or ESP\_HF\_CHLD\_TYPE\_PRIV\_X

***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.

**Return**

- ESP\_OK: disconnect request is sent to lower layer
- ESP\_INVALID\_STATE: if bluetooth stack is not yet enabled
- ESP\_FAIL: others

**Parameters**

- [in] *btrh*: response and hold action to send

***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.

**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\_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.

**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\_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.

**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\_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.

#### 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\_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.

#### 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\_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.

#### Return

- ESP\_OK: disconnect request is sent to lower layer
- ESP\_INVALID\_STATE: if bluetooth stack is not yet enabled
- ESP\_FAIL: others

#### Parameters

- [in] code: dtmf code, single ascii character in the set 0-9, #, \*, A-D

*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.

#### 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\_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.

#### Return

- ESP\_OK: success
- ESP\_INVALID\_STATE: if bluetooth stack is not yet enabled
- ESP\_FAIL: if callback is a NULL function pointer

#### Parameters

- [in] recv: HFP client incoming data callback function
- [in] send: HFP client outgoing data callback function

void **esp\_hf\_client\_outgoing\_data\_ready** (void)

Trigger the lower-layer to fetch and send audio data. This function is only used in the case that Voice Over HCI is enabled. Precondition is that the HFP audio connection is connected. After this function is called, lower layer will invoke *esp\_hf\_client\_outgoing\_data\_cb\_t* to fetch data.

void **esp\_hf\_client\_pcm\_resample\_init** (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.

#### Parameters

- [in] src\_sps: original samples per second(source audio data, i.e. 48000, 32000, 16000, 44100, 22050, 11025)
- [in] bits: number of bits per pcm sample (16)
- [in] channels: number of channels (i.e. mono(1), stereo(2)...) )

void **esp\_hf\_client\_pcm\_resample\_deinit** (void)

Deinitialize the down sampling converter.

`int32_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.

**Return** number of samples converted

**Parameters**

- [in] src: pointer to the buffer where the original sampling PCM are stored
- [in] in\_bytes: length of the input PCM sample buffer in byte
- [in] dst: pointer to the buffer which is to be used to store the converted PCM samples

## 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.
```

### Public Members

```
int value
    battery charge value, ranges from 0 to 5
```

```
struct hf_client_binp_param
    #include <esp_hf_client_api.h> ESP_HF_CLIENT_BINP_EVT.
```

### Public Members

```
const char *number
    phone number corresponding to the last voice tag in the HF
```

```
struct hf_client_bsirparam
    #include <esp_hf_client_api.h> ESP_HF_CLIENT_BSIR_EVT.
```

### Public Members

```
esp_hf_client_in_band_ring_state_t state
    setting state of in-band ring tone
```

```
struct hf_client_btrh_param  
    #include <esp_hf_client_api.h> ESP_HF_CLIENT_BTRH_EVT.
```

### Public Members

*esp\_hf\_btrh\_status\_t* **status**  
call hold and response status result code

```
struct hf_client_bvra_param  
    #include <esp_hf_client_api.h> ESP_HF_CLIENT_BVRA_EVT.
```

### Public Members

*esp\_hf\_vr\_state\_t* **value**  
voice recognition state

```
struct hf_client_call_held_ind_param  
    #include <esp_hf_client_api.h> ESP_HF_CLIENT_CIND_CALL_HELD_EVT.
```

### Public Members

*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

**struct hf\_client\_cnum\_param**  
*#include <esp\_hf\_client\_api.h>* ESP\_HF\_CLIENT\_CNUM\_EVT.

**Public Members**

**const** char \***number**  
phone number string

*esp\_hf\_subscriber\_service\_type\_t* **type**  
service type that the phone number relates to

**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

uint32\_t **peer\_feat**  
AG supported features

uint32\_t **chld\_feat**  
AG supported features on call hold and multiparty services

*esp\_bd\_addr\_t* **remote\_bda**  
remote bluetooth device address

**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

**struct hf\_client\_network\_roaming\_param**  
*#include <esp\_hf\_client\_api.h>* ESP\_HF\_CLIENT\_CIND\_ROAMING\_STATUS\_EVT.

### Public Members

*esp\_hf\_roaming\_status\_t* **status**  
roaming status

**struct hf\_client\_service\_availability\_param**  
*#include <esp\_hf\_client\_api.h>* ESP\_HF\_CLIENT\_CIND\_SERVICE\_AVAILABILITY\_EVT.

### Public Members

*esp\_hf\_service\_availability\_status\_t* **status**  
service availability status

**struct hf\_client\_signal\_strength\_ind\_param**  
*#include <esp\_hf\_client\_api.h>* ESP\_HF\_CLIENT\_CIND\_SIGNAL\_STRENGTH\_EVT.

### Public Members

**int value**  
signal strength value, ranges from 0 to 5

**struct hf\_client\_volume\_control\_param**  
*#include <esp\_hf\_client\_api.h>* ESP\_HF\_CLIENT\_VOLUME\_CONTROL\_EVT.

### Public Members

*esp\_hf\_volume\_control\_target\_t* **type**  
volume control target, speaker or microphone

**int volume**  
gain, ranges from 0 to 15

### Macros

ESP\_BT\_HF\_CLIENT\_NUMBER\_LEN  
ESP\_BT\_HF\_CLIENT\_OPERATOR\_NAME\_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_EXTERR
ESP_HF_CLIENT_PEER_FEAT_CODEC
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

```

### 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.

#### Parameters

- [in] buf: : 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.
- [in] len: : 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.

**Return** length of data successfully read

#### Parameters

- [in] buf: : 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.
- [in] len: : size(in bytes) in buf

**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.

#### Parameters

- event: : Event type
- param: : Pointer to callback parameter

### Enumerations

**enum** **esp\_hf\_client\_connection\_state\_t**

Bluetooth HFP RFCOMM connection and service level connection status.

*Values:*

**ESP\_HF\_CLIENT\_CONNECTION\_STATE\_DISCONNECTED** = 0  
RFCOMM data link channel released

**ESP\_HF\_CLIENT\_CONNECTION\_STATE\_CONNECTING**  
connecting remote device on the RFCOMM data link

**ESP\_HF\_CLIENT\_CONNECTION\_STATE\_CONNECTED**  
RFCOMM connection established

**ESP\_HF\_CLIENT\_CONNECTION\_STATE\_SLC\_CONNECTED**  
service level connection established

**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:***ESP\_HF\_CLIENT\_AUDIO\_STATE\_DISCONNECTED = 0**  
audio connection released**ESP\_HF\_CLIENT\_AUDIO\_STATE\_CONNECTING**  
audio connection has been initiated**ESP\_HF\_CLIENT\_AUDIO\_STATE\_CONNECTED**  
audio connection is established**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:***ESP\_HF\_CLIENT\_IN\_BAND\_RINGTONE\_NOT\_PROVIDED = 0****ESP\_HF\_CLIENT\_IN\_BAND\_RINGTONE\_PROVIDED****enum esp\_hf\_client\_cb\_event\_t**

HF CLIENT callback events.

*Values:***ESP\_HF\_CLIENT\_CONNECTION\_STATE\_EVT = 0**  
connection state changed event**ESP\_HF\_CLIENT\_AUDIO\_STATE\_EVT**  
audio connection state change event**ESP\_HF\_CLIENT\_BVRA\_EVT**  
voice recognition state change event**ESP\_HF\_CLIENT\_CIND\_CALL\_EVT**  
call indication**ESP\_HF\_CLIENT\_CIND\_CALL\_SETUP\_EVT**  
call setup indication**ESP\_HF\_CLIENT\_CIND\_CALL\_HELD\_EVT**  
call held indication**ESP\_HF\_CLIENT\_CIND\_SERVICE\_AVAILABILITY\_EVT**  
network service availability indication**ESP\_HF\_CLIENT\_CIND\_SIGNAL\_STRENGTH\_EVT**  
signal strength indication**ESP\_HF\_CLIENT\_CIND\_ROAMING\_STATUS\_EVT**  
roaming status indication**ESP\_HF\_CLIENT\_CIND\_BATTERY\_LEVEL\_EVT**  
battery level indication**ESP\_HF\_CLIENT\_COPS\_CURRENT\_OPERATOR\_EVT**  
current operator information**ESP\_HF\_CLIENT\_BTRH\_EVT**  
call response and hold event**ESP\_HF\_CLIENT\_CLIP\_EVT**  
Calling Line Identification notification



<b>ESP_HF_CLIENT_CCWA_EVT</b>	call waiting notification
<b>ESP_HF_CLIENT_CLCC_EVT</b>	list of current calls notification
<b>ESP_HF_CLIENT_VOLUME_CONTROL_EVT</b>	audio volume control command from AG, provided by +VGM or +VGS message
<b>ESP_HF_CLIENT_AT_RESPONSE_EVT</b>	AT command response event
<b>ESP_HF_CLIENT_CNUM_EVT</b>	subscriber information response from AG
<b>ESP_HF_CLIENT_BSIR_EVT</b>	setting of in-band ring tone
<b>ESP_HF_CLIENT_BINP_EVT</b>	requested number of last voice tag from AG
<b>ESP_HF_CLIENT_RING_IND_EVT</b>	ring indication event

## 2.1.5 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 Blueroid) 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. However, 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 requirements.

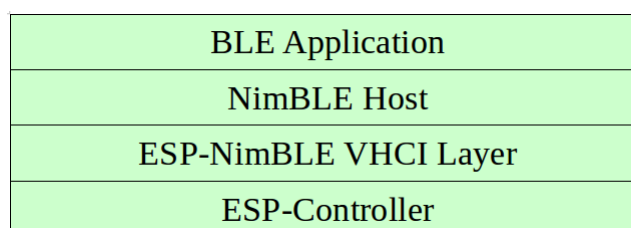


图 1: ESP NimBLE Stack

### Threading Model

The NimBLE host can run inside the application thread or can have its own independent thread. This flexibility is inherently provided by NimBLE design. By default, a thread is spawned by the porting function `nim-`

`ble_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.
- Call `esp_nimble_hci_and_controller_init()` to initialize ESP controller as well as transport layer. This will also link the host and controller modules together. Alternatively, if ESP controller is already initialized, then `esp_nimble_hci_init()` can be called for the remaining initialization.
- Initialize the host 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

- [bt/host/nimble/esp-hci/include/esp\\_nimble\\_hci.h](#)

### Functions

`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.

#### Return

- 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_and_controller_init` (void)

Initialize ESP Bluetooth controller(link layer) and VHCI transport layer between NimBLE Host and ESP Bluetooth controller.

This function initializes ESP controller in BLE only mode and the transport buffers to be exchanged between NimBLE host and ESP controller. It also registers required host callbacks with the controller.

Below is the sequence of APIs to be called to init/enable NimBLE host and ESP controller:

```
void ble_host_task(void *param)
{
    nimble_port_run(); //This function will return only when nimble_port_
    ↪stop() is executed.
    nimble_port_freertos_deinit();
}

int ret = esp_nimble_hci_and_controller_init();
if (ret != ESP_OK) {
    ESP_LOGE(TAG, "esp_nimble_hci_and_controller_init() failed with error: %d
    ↪", ret);
    return;
}
```

(下页继续)

```
nimble_port_init();

//Initialize the NimBLE Host configuration

nimble_port_freertos_init(ble_host_task);
```

`nimble_port_freertos_init()` is an optional call that creates a new task in which the NimBLE host will run. The task function should have a call to `nimble_port_run()`. If a separate task is not required, calling `nimble_port_run()` will run the NimBLE host in the current task.

**Return**

- 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.

**Note** This function should be called after the NimBLE host is deinitialized.

**Return**

- ESP\_OK if the deinitialization is successful
- Appropriate error codes from `esp_err_t` in case of an error

*esp\_err\_t* **esp\_nimble\_hci\_and\_controller\_deinit** (void)

Deinitialize VHCI transport layer between NimBLE Host and ESP Bluetooth controller and disable and deinitialize the controller.

Below is the sequence of APIs to be called to disable/deinit NimBLE host and ESP controller:

**Note** This function should not be executed in the context of Bluetooth host task.

**Note** This function should be called after the NimBLE host is deinitialized.

```
int ret = nimble_port_stop();
if (ret == 0) {
    nimble_port_deinit();

    ret = esp_nimble_hci_and_controller_deinit();
    if (ret != ESP_OK) {
        ESP_LOGE(TAG, "esp_nimble_hci_and_controller_deinit() failed with_
->error: %d", ret);
    }
}
```

If `nimble_port_freertos_init()` is used during initialization, then `nimble_port_freertos_deinit()` should be called in the host task after `nimble_port_run()`.

**Return**

- 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**

## 2.1.6 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 unprovisioned 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 [Getting Started with ESP-BLE-MESH](#). 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 Examples](#) and [ESP-BLE-MESH Demo Videos](#).

### 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

- [bt/esp\\_ble\\_mesh/api/esp\\_ble\\_mesh\\_defs.h](#)

### Unions

```
union esp_ble_mesh_prov_cb_param_t
#include <esp_ble_mesh_defs.h> BLE Mesh Node/Provisioner callback parameters union.
```

### Public Members

```
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
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_num_comp
    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_rcv_unprov_adv_pkt_param provisioner_rcv_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_evt
    Event parameter of ESP_BLE_MESH_PROVISIONER_PROV_LINK_OPEN_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_prov_read_oob_pub_key_evt_param provisioner_prov_read_oob_pub_key_evt
    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_evt
    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_link_close_evt_param provisioner_prov_link_clos
    Event parameter of ESP_BLE_MESH_PROVISIONER_PROV_LINK_CLOSE_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_prov_comp_param 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_unpr
    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_
    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_c
    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_d
    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_prov_data_info_comp_param provisioner_set_pr
    Event parameter of ESP_BLE_MESH_PROVISIONER_SET_PROV_DATA_INFO_COMP_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_set_static_oob_val_comp_param provisioner_set_sta
    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
    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_pr
    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_inp
    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_inpu
    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
    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_ap
    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_upda
    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_
    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 provisioner_add_ne
    Event parameter of 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 provisioner_updat
    Event parameter of 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 provisioner_sto
    Event parameter of 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 provisioner_del
    Event parameter of 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 provisioner_del
    Event parameter of ESP_BLE_MESH_PROVISIONER_DELETE_NODE_WITH_ADDR_COMP_EVT

int err_code
    Indicate the result of enabling/disabling to receive heartbeat messages by the Provisioner

    Indicate the result of setting the heartbeat filter type by the Provisioner

```

Indicate the result of setting the heartbeat filter address by the Provisioner

Indicate the result of directly erasing settings by the Provisioner

Indicate the result of opening settings with index by the Provisioner

Indicate the result of opening settings with user id by the Provisioner

Indicate the result of closing settings with index by the Provisioner

Indicate the result of closing settings with user id by the Provisioner

Indicate the result of deleting settings with index by the Provisioner

Indicate the result of deleting settings with user id by the Provisioner

bool **enable**

Indicate enabling or disabling receiving heartbeat messages

**struct** *esp\_ble\_mesh\_prov\_cb\_param\_t::[anonymous]* **provisioner\_enable\_heartbeat\_recv\_comp**  
ESP\_BLE\_MESH\_PROVISIONER\_ENABLE\_HEARTBEAT\_RECV\_COMP\_EVT.

Event parameters of ESP\_BLE\_MESH\_PROVISIONER\_ENABLE\_HEARTBEAT\_RECV\_COMP\_EVT

uint8\_t **type**

Type of the filter used for receiving heartbeat messages

**struct** *esp\_ble\_mesh\_prov\_cb\_param\_t::[anonymous]* **provisioner\_set\_heartbeat\_filter\_type\_comp**  
ESP\_BLE\_MESH\_PROVISIONER\_SET\_HEARTBEAT\_FILTER\_TYPE\_COMP\_EVT.

Event parameters of ESP\_BLE\_MESH\_PROVISIONER\_SET\_HEARTBEAT\_FILTER\_TYPE\_COMP\_EVT

uint8\_t **op**

Operation (add, remove, clean)

uint16\_t **hb\_src**

Heartbeat source address

uint16\_t **hb\_dst**

Heartbeat destination address

**struct** *esp\_ble\_mesh\_prov\_cb\_param\_t::[anonymous]* **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 **init\_ttl**

Heartbeat InitTTL

uint8\_t **rx\_ttl**

Heartbeat RxTTL

uint8\_t **hops**

Heartbeat hops (InitTTL - RxTTL + 1)

uint16\_t **feature**

Bit field of currently active features of the node

int8\_t **rss\_i**

RSSI of the heartbeat message

**struct** *esp\_ble\_mesh\_prov\_cb\_param\_t::[anonymous]* **provisioner\_rcv\_heartbeat**  
ESP\_BLE\_MESH\_PROVISIONER\_RECV\_HEARTBEAT\_MESSAGE\_EVT.

Event parameters of ESP\_BLE\_MESH\_PROVISIONER\_RECV\_HEARTBEAT\_MESSAGE\_EVT

**struct** *esp\_ble\_mesh\_prov\_cb\_param\_t::[anonymous]* **provisioner\_direct\_erase\_settings\_comp**  
ESP\_BLE\_MESH\_PROVISIONER\_DRIECT\_ERASE\_SETTINGS\_COMP\_EVT.

Event parameters of ESP\_BLE\_MESH\_PROVISIONER\_DRIECT\_ERASE\_SETTINGS\_COMP\_EVT

**uint8\_t index**

Index of Provisioner settings

**struct esp\_ble\_mesh\_prov\_cb\_param\_t::[anonymous] 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

char **uid**[ESP\_BLE\_MESH\_SETTINGS\_UID\_SIZE + 1]

Provisioner settings user id

**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\_UID\_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\_UID\_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\_rcv\_param heartbeat\_msg\_rcv**  
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\_rcv\_adv\_pkt\_param proxy\_client\_rcv\_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_rcv_filter_status_param proxy_client_rcv_filt
    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
    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_filt
    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_fi
    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_remo
    Event parameter of ESP_BLE_MESH_PROXY_CLIENT_REMOVE_FILTER_ADDR_COMP_EVT

struct esp_ble_mesh_prov_cb_param_t::ble_mesh_model_sub_group_addr_comp_param model_sub_group_addr_co
    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_add
    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

struct ble_mesh_deinit_mesh_comp_param
    #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:

```

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

```

ESP_BLE_MESH_FRND_FRIENDSHIP_TERMINATE_POLL_TIMEOUT

```

Friendship is established, PollTimeout timer expires and no Friend Poll/Sub Add/Sub Remove is received

**ESP\_BLE\_MESH\_FRND\_FRIENDSHIP\_TERMINATE\_RECV\_FRND\_REQ**

Receive Friend Request from existing Low Power Node

**ESP\_BLE\_MESH\_FRND\_FRIENDSHIP\_TERMINATE\_RECV\_FRND\_CLEAR**

Receive Friend Clear from other friend node

**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

*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 reason

**struct ble\_mesh\_heartbeat\_msg\_rcv\_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.*

### Public Members

**int err\_code**

Indicate the result of inputting number

**struct ble\_mesh\_input\_string\_comp\_param**

*#include <esp\_ble\_mesh\_defs.h> ESP\_BLE\_MESH\_NODE\_PROV\_INPUT\_STR\_COMP\_EVT.*

### Public Members

**int err\_code**

Indicate the result of inputting string

**struct ble\_mesh\_link\_close\_evt\_param**

*#include <esp\_ble\_mesh\_defs.h> ESP\_BLE\_MESH\_NODE\_PROV\_LINK\_CLOSE\_EVT.*

### Public Members

*esp\_ble\_mesh\_prov\_bearer\_t* **bearer**

Type of the bearer used when device link is closed

```
struct ble_mesh_link_open_evt_param  
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_NODE_PROV_LINK_OPEN_EVT.
```

### Public Members

*esp\_ble\_mesh\_prov\_bearer\_t* **bearer**

Type of the bearer used when device link is open

```
struct ble_mesh_lpn_disable_comp_param  
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_LPN_DISABLE_COMP_EVT.
```

### Public Members

int **err\_code**

Indicate the result of disabling LPN functionality

```
struct ble_mesh_lpn_enable_comp_param  
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_LPN_ENABLE_COMP_EVT.
```

### Public Members

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.

**Public Members**

int **err\_code**  
Indicate the result of local model unsubscribing 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\_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**

int **err\_code**  
Indicate the result of adding local AppKey by the node

uint16\_t **net\_idx**  
NetKey Index

uint16\_t **app\_idx**  
AppKey Index

**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**

int **err\_code**  
Indicate the result of adding local NetKey by the node

uint16\_t **net\_idx**  
NetKey Index

```
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

```
int err_code  
    Indicate the result of enabling BLE Mesh device
```

```
struct ble_mesh_prov_register_comp_param  
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROV_REGISTER_COMP_EVT.
```

**Public Members**

int **err\_code**  
Indicate the result of BLE Mesh initialization

**struct ble\_mesh\_provision\_complete\_evt\_param**  
*#include <esp\_ble\_mesh\_defs.h>* ESP\_BLE\_MESH\_NODE\_PROV\_COMPLETE\_EVT.

**Public Members**

uint16\_t **net\_idx**  
NetKey Index

uint8\_t **net\_key**[16]  
NetKey

uint16\_t **addr**  
Primary address

uint8\_t **flags**  
Flags

uint32\_t **iv\_index**  
IV Index

**struct ble\_mesh\_provision\_reset\_param**  
*#include <esp\_ble\_mesh\_defs.h>* ESP\_BLE\_MESH\_NODE\_PROV\_RESET\_EVT.

**struct ble\_mesh\_provisioner\_add\_local\_app\_key\_comp\_param**  
*#include <esp\_ble\_mesh\_defs.h>* ESP\_BLE\_MESH\_PROVISIONER\_ADD\_LOCAL\_APP\_KEY\_COMP\_EVT.

**Public Members**

int **err\_code**  
Indicate the result of adding local AppKey by the Provisioner

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**

int **err\_code**  
Indicate the result of deleting device by the Provisioner

**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**

int **err\_code**  
Indicate the result of deleting node with unicast address by the Provisioner

uint16\_t **unicast\_addr**  
Node unicast address

**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**

int **err\_code**  
Indicate the result of deleting node with uuid by the Provisioner

uint8\_t **uuid**[16]  
Node device uuid

**struct ble\_mesh\_provisioner\_link\_close\_evt\_param**  
*#include <esp\_ble\_mesh\_defs.h>* ESP\_BLE\_MESH\_PROVISIONER\_PROV\_LINK\_CLOSE\_EVT.

**Public Members*****esp\_ble\_mesh\_prov\_bearer\_t* bearer**

Type of the bearer used when Provisioner link is closed

**uint8\_t reason**

Reason of the closed provisioning link

**struct ble\_mesh\_provisioner\_link\_open\_evt\_param***#include <esp\_ble\_mesh\_defs.h>* ESP\_BLE\_MESH\_PROVISIONER\_PROV\_LINK\_OPEN\_EVT.**Public Members*****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

**struct ble\_mesh\_provisioner\_prov\_input\_num\_comp\_param**  
*#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

**struct ble\_mesh\_provisioner\_prov\_input\_str\_comp\_param**  
*#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

**struct ble\_mesh\_provisioner\_prov\_output\_evt\_param**  
*#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

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_PUB_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_PUB_KEY_EVT.
```

### Public Members

```
uint8_t link_idx
    Index of the provisioning link
```

```
struct ble_mesh_provisioner_rcv_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_t 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
#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
#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***#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***#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***#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

uint16\_t **net\_idx**  
Corresponding NetKey Index

uint8\_t **reason**  
Proxy disconnect reason

**struct ble\_mesh\_proxy\_client\_rcv\_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\_rcv\_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_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\_rcv\_publish\_msg\_param* **client\_rcv\_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**  
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\_rcv\_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

`esp_ble_mesh_model_t *model`  
Pointer to the model which sends the message

`esp_ble_mesh_msg_ctx_t *ctx`  
Context of the message

**struct ble\_mesh\_server\_model\_update\_state\_comp\_param**  
*#include <esp\_ble\_mesh\_defs.h>* ESP\_BLE\_MESH\_SERVER\_MODEL\_UPDATE\_STATE\_COMP\_EVT.

### Public Members

int **err\_code**  
Indicate the result of updating server model state

*esp\_ble\_mesh\_model\_t* \***model**  
Pointer to the server model which state value is updated

*esp\_ble\_mesh\_server\_state\_type\_t* **type**  
Type of the updated server state

### Structures

**struct esp\_ble\_mesh\_deinit\_param\_t**  
BLE Mesh deinit parameters

### Public Members

bool **erase\_flash**  
Indicate if erasing flash when deinit mesh stack

**struct 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`

### Public Members

uint16\_t **element\_addr**  
Element Address, assigned during provisioning.

**const** uint16\_t **location**  
Location Descriptor (GATT Bluetooth Namespace Descriptors)

**const** uint8\_t **sig\_model\_count**  
SIG Model count

**const** uint8\_t **vnd\_model\_count**  
Vendor Model count

*esp\_ble\_mesh\_model\_t* \***sig\_models**  
SIG Models

*esp\_ble\_mesh\_model\_t* \***vnd\_models**  
Vendor Models

**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** : 12  
Publish AppKey Index.

uint16\_t **cred** : 1  
Friendship Credentials Flag.

`uint16_t send_rel` : 1  
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` : 4  
Divisor for the Period.

`uint8_t fast_period` : 1  
Use FastPeriodDivisor

`uint8_t count` : 3  
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**

`union esp_ble_mesh_model::[anonymous] [anonymous]`

Model ID

`uint8_t element_idx`

Internal information, mainly for persistent storage Belongs to Nth element

`uint8_t model_idx`

Is the Nth model in the element

`uint16_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**

Message 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 : 7`

Received TTL value. Not used for sending.

`uint8_t send_rel : 1`

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

`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.

### 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

**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 : 1**  
BLE Mesh Server Response Option.

1. 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;
2. 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;
3. 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;
4. 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;
5. 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;
6. 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 : 1**  
Response control for Client Set messages

**uint8\_t status\_auto\_rsp : 1**  
Response control for Server Status messages

**Macros****ESP\_BLE\_HOST\_STATUS\_ENABLED****ESP\_BLE\_HOST\_STATUS\_CHECK** (status)

The maximum length of a BLE Mesh message, including Opcode, Payload and TransMIC

**ESP\_BLE\_MESH\_SDU\_MAX\_LEN**

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**

**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.

**Note** 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).

**Return** BLE Mesh transmit value that can be used e.g. for the default values of the Configuration Model data.

**Parameters**

- `count`: Number of retransmissions (first transmission is excluded).
- `int_ms`: Interval steps in milliseconds. Must be greater than 0 and a multiple of 10.

**ESP\_BLE\_MESH\_GET\_TRANSMIT\_COUNT** (transmit)

Decode transmit count from a transmit value.

**Return** Transmission count (actual transmissions equal to  $N + 1$ ).

**Parameters**

- `transmit`: Encoded transmit count & interval value.

**ESP\_BLE\_MESH\_GET\_TRANSMIT\_INTERVAL** (transmit)

Decode transmit interval from a transmit value.

**Return** Transmission interval in milliseconds.

**Parameters**

- `transmit`: Encoded transmit count & interval value.

**ESP\_BLE\_MESH\_PUBLISH\_TRANSMIT** (count, int\_ms)

Encode Publish Retransmit count & interval steps.

**Return** BLE Mesh transmit value that can be used e.g. for the default values of the Configuration Model data.

**Parameters**

- `count`: Number of retransmissions (first transmission is excluded).
- `int_ms`: Interval steps in milliseconds. Must be greater than 0 and a multiple of 50.

**ESP\_BLE\_MESH\_GET\_PUBLISH\_TRANSMIT\_COUNT** (transmit)

Decode Publish Retransmit count from a given value.

**Return** Retransmission count (actual transmissions equal to  $N + 1$ ).

**Parameters**

- `transmit`: Encoded Publish Retransmit count & interval value.

**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)

**Return** Transmission interval in milliseconds.

**Parameters**

- `transmit`: Encoded Publish Retransmit count & interval value.

**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.

**Note** This macro is associated with BLE\_MESH\_ELEM in mesh\_access.h

**Parameters**

- `_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.

**Parameters**

- `_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.

**Parameters**

- `_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  
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  
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\_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  
**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  
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  
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  
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\_DEFAULT\_GET  
ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_CTL\_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\_STATUS  
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  
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\_IC\_MODE\_GET  
ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_IC\_MODE\_SET  
ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_IC\_MODE\_SET\_UNACK  
ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_IC\_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**  
**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** *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 the Config Model messages. 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:

**ESP\_BLE\_MESH\_TYPE\_PROV\_CB**

**ESP\_BLE\_MESH\_TYPE\_OUTPUT\_NUM\_CB**

```
ESP_BLE_MESH_TYPE_OUTPUT_STR_CB
ESP_BLE_MESH_TYPE_INTPUT_CB
ESP_BLE_MESH_TYPE_LINK_OPEN_CB
ESP_BLE_MESH_TYPE_LINK_CLOSE_CB
ESP_BLE_MESH_TYPE_COMPLETE_CB
ESP_BLE_MESH_TYPE_RESET_CB
enum esp_ble_mesh_oob_method_t
  Values:
    ESP_BLE_MESH_NO_OOB
    ESP_BLE_MESH_STATIC_OOB
    ESP_BLE_MESH_OUTPUT_OOB
    ESP_BLE_MESH_INPUT_OOB
enum esp_ble_mesh_output_action_t
  Values:
    ESP_BLE_MESH_NO_OUTPUT = 0
    ESP_BLE_MESH_BLINK = BIT(0)
    ESP_BLE_MESH_BEEP = BIT(1)
    ESP_BLE_MESH_VIBRATE = BIT(2)
    ESP_BLE_MESH_DISPLAY_NUMBER = BIT(3)
    ESP_BLE_MESH_DISPLAY_STRING = BIT(4)
enum esp_ble_mesh_input_action_t
  Values:
    ESP_BLE_MESH_NO_INPUT = 0
    ESP_BLE_MESH_PUSH = BIT(0)
    ESP_BLE_MESH_TWIST = BIT(1)
    ESP_BLE_MESH_ENTER_NUMBER = BIT(2)
    ESP_BLE_MESH_ENTER_STRING = BIT(3)
enum esp_ble_mesh_prov_bearer_t
  Values:
    ESP_BLE_MESH_PROV_ADV = BIT(0)
    ESP_BLE_MESH_PROV_GATT = BIT(1)
enum esp_ble_mesh_prov_oob_info_t
  Values:
    ESP_BLE_MESH_PROV_OOB_OTHER = BIT(0)
    ESP_BLE_MESH_PROV_OOB_URI = BIT(1)
    ESP_BLE_MESH_PROV_OOB_2D_CODE = BIT(2)
    ESP_BLE_MESH_PROV_OOB_BAR_CODE = BIT(3)
    ESP_BLE_MESH_PROV_OOB_NFC = BIT(4)
    ESP_BLE_MESH_PROV_OOB_NUMBER = BIT(5)
    ESP_BLE_MESH_PROV_OOB_STRING = BIT(6)
    ESP_BLE_MESH_PROV_OOB_ON_BOX = BIT(11)
```

```
ESP_BLE_MESH_PROV_OOB_IN_BOX = BIT(12)
ESP_BLE_MESH_PROV_OOB_ON_PAPER = BIT(13)
ESP_BLE_MESH_PROV_OOB_IN_MANUAL = BIT(14)
ESP_BLE_MESH_PROV_OOB_ON_DEV = BIT(15)
```

```
enum esp_ble_mesh_dev_role_t
```

*Values:*

```
ROLE_NODE = 0
ROLE_PROVISIONER
ROLE_FAST_PROV
```

```
enum esp_ble_mesh_fast_prov_action_t
```

*Values:*

```
FAST_PROV_ACT_NONE
FAST_PROV_ACT_ENTER
FAST_PROV_ACT_SUSPEND
FAST_PROV_ACT_EXIT
FAST_PROV_ACT_MAX
```

```
enum esp_ble_mesh_proxy_filter_type_t
```

*Values:*

```
PROXY_FILTER_WHITELIST
PROXY_FILTER_BLACKLIST
```

```
enum esp_ble_mesh_prov_cb_event_t
```

*Values:*

```
ESP_BLE_MESH_PROV_REGISTER_COMP_EVT
    Initialize BLE Mesh provisioning capabilities and internal data information completion event

ESP_BLE_MESH_NODE_SET_UNPROV_DEV_NAME_COMP_EVT
    Set the unprovisioned device name completion event

ESP_BLE_MESH_NODE_PROV_ENABLE_COMP_EVT
    Enable node provisioning functionality completion event

ESP_BLE_MESH_NODE_PROV_DISABLE_COMP_EVT
    Disable node provisioning functionality completion event

ESP_BLE_MESH_NODE_PROV_LINK_OPEN_EVT
    Establish a BLE Mesh link event

ESP_BLE_MESH_NODE_PROV_LINK_CLOSE_EVT
    Close a BLE Mesh link event

ESP_BLE_MESH_NODE_PROV_OOB_PUB_KEY_EVT
    Generate Node input OOB public key event

ESP_BLE_MESH_NODE_PROV_OUTPUT_NUMBER_EVT
    Generate Node Output Number event

ESP_BLE_MESH_NODE_PROV_OUTPUT_STRING_EVT
    Generate Node Output String event

ESP_BLE_MESH_NODE_PROV_INPUT_EVT
    Event requiring the user to input a number or string

ESP_BLE_MESH_NODE_PROV_COMPLETE_EVT
    Provisioning done event
```

<b>ESP_BLE_MESH_NODE_PROV_RESET_EVT</b>	Provisioning reset event
<b>ESP_BLE_MESH_NODE_PROV_SET_OOB_PUB_KEY_COMP_EVT</b>	Node set oob public key completion event
<b>ESP_BLE_MESH_NODE_PROV_INPUT_NUMBER_COMP_EVT</b>	Node input number completion event
<b>ESP_BLE_MESH_NODE_PROV_INPUT_STRING_COMP_EVT</b>	Node input string completion event
<b>ESP_BLE_MESH_NODE_PROXY_IDENTITY_ENABLE_COMP_EVT</b>	Enable BLE Mesh Proxy Identity advertising completion event
<b>ESP_BLE_MESH_NODE_PROXY_GATT_ENABLE_COMP_EVT</b>	Enable BLE Mesh GATT Proxy Service completion event
<b>ESP_BLE_MESH_NODE_PROXY_GATT_DISABLE_COMP_EVT</b>	Disable BLE Mesh GATT Proxy Service completion event
<b>ESP_BLE_MESH_NODE_ADD_LOCAL_NET_KEY_COMP_EVT</b>	Node add NetKey locally completion event
<b>ESP_BLE_MESH_NODE_ADD_LOCAL_APP_KEY_COMP_EVT</b>	Node add AppKey locally completion event
<b>ESP_BLE_MESH_NODE_BIND_APP_KEY_TO_MODEL_COMP_EVT</b>	Node bind AppKey to model locally completion event
<b>ESP_BLE_MESH_PROVISIONER_PROV_ENABLE_COMP_EVT</b>	Provisioner enable provisioning functionality completion event
<b>ESP_BLE_MESH_PROVISIONER_PROV_DISABLE_COMP_EVT</b>	Provisioner disable provisioning functionality completion event
<b>ESP_BLE_MESH_PROVISIONER_RECV_UNPROV_ADV_PKT_EVT</b>	Provisioner receives unprovisioned device beacon event
<b>ESP_BLE_MESH_PROVISIONER_PROV_READ_OOB_PUB_KEY_EVT</b>	Provisioner read unprovisioned device OOB public key event
<b>ESP_BLE_MESH_PROVISIONER_PROV_INPUT_EVT</b>	Provisioner input value for provisioning procedure event
<b>ESP_BLE_MESH_PROVISIONER_PROV_OUTPUT_EVT</b>	Provisioner output value for provisioning procedure event
<b>ESP_BLE_MESH_PROVISIONER_PROV_LINK_OPEN_EVT</b>	Provisioner establish a BLE Mesh link event
<b>ESP_BLE_MESH_PROVISIONER_PROV_LINK_CLOSE_EVT</b>	Provisioner close a BLE Mesh link event
<b>ESP_BLE_MESH_PROVISIONER_PROV_COMPLETE_EVT</b>	Provisioner provisioning done event
<b>ESP_BLE_MESH_PROVISIONER_ADD_UNPROV_DEV_COMP_EVT</b>	Provisioner add a device to the list which contains devices that are waiting/going to be provisioned completion event
<b>ESP_BLE_MESH_PROVISIONER_PROV_DEV_WITH_ADDR_COMP_EVT</b>	Provisioner start to provision an unprovisioned device completion event
<b>ESP_BLE_MESH_PROVISIONER_DELETE_DEV_COMP_EVT</b>	Provisioner delete a device from the list, close provisioning link with the device completion event
<b>ESP_BLE_MESH_PROVISIONER_SET_DEV_UUID_MATCH_COMP_EVT</b>	Provisioner set the value to be compared with part of the unprovisioned device UUID completion event

<b>ESP_BLE_MESH_PROVISIONER_SET_PROV_DATA_INFO_COMP_EVT</b>
Provisioner set net_idx/flags/iv_index used for provisioning completion event
<b>ESP_BLE_MESH_PROVISIONER_SET_STATIC_OOB_VALUE_COMP_EVT</b>
Provisioner set static oob value used for provisioning completion event
<b>ESP_BLE_MESH_PROVISIONER_SET_PRIMARY_ELEM_ADDR_COMP_EVT</b>
Provisioner set unicast address of primary element completion event
<b>ESP_BLE_MESH_PROVISIONER_PROV_READ_OOB_PUB_KEY_COMP_EVT</b>
Provisioner read unprovisioned device OOB public key completion event
<b>ESP_BLE_MESH_PROVISIONER_PROV_INPUT_NUMBER_COMP_EVT</b>
Provisioner input number completion event
<b>ESP_BLE_MESH_PROVISIONER_PROV_INPUT_STRING_COMP_EVT</b>
Provisioner input string completion event
<b>ESP_BLE_MESH_PROVISIONER_SET_NODE_NAME_COMP_EVT</b>
Provisioner set node name completion event
<b>ESP_BLE_MESH_PROVISIONER_ADD_LOCAL_APP_KEY_COMP_EVT</b>
Provisioner add local app key completion event
<b>ESP_BLE_MESH_PROVISIONER_UPDATE_LOCAL_APP_KEY_COMP_EVT</b>
Provisioner update local app key completion event
<b>ESP_BLE_MESH_PROVISIONER_BIND_APP_KEY_TO_MODEL_COMP_EVT</b>
Provisioner bind local model with local app key completion event
<b>ESP_BLE_MESH_PROVISIONER_ADD_LOCAL_NET_KEY_COMP_EVT</b>
Provisioner add local network key completion event
<b>ESP_BLE_MESH_PROVISIONER_UPDATE_LOCAL_NET_KEY_COMP_EVT</b>
Provisioner update local network key completion event
<b>ESP_BLE_MESH_PROVISIONER_STORE_NODE_COMP_DATA_COMP_EVT</b>
Provisioner store node composition data completion event
<b>ESP_BLE_MESH_PROVISIONER_DELETE_NODE_WITH_UUID_COMP_EVT</b>
Provisioner delete node with uuid completion event
<b>ESP_BLE_MESH_PROVISIONER_DELETE_NODE_WITH_ADDR_COMP_EVT</b>
Provisioner delete node with unicast address completion event
<b>ESP_BLE_MESH_PROVISIONER_ENABLE_HEARTBEAT_RECV_COMP_EVT</b>
Provisioner start to receive heartbeat message completion event
<b>ESP_BLE_MESH_PROVISIONER_SET_HEARTBEAT_FILTER_TYPE_COMP_EVT</b>
Provisioner set the heartbeat filter type completion event
<b>ESP_BLE_MESH_PROVISIONER_SET_HEARTBEAT_FILTER_INFO_COMP_EVT</b>
Provisioner set the heartbeat filter information completion event
<b>ESP_BLE_MESH_PROVISIONER_RECV_HEARTBEAT_MESSAGE_EVT</b>
Provisioner receive heartbeat message event
<b>ESP_BLE_MESH_PROVISIONER_DIRECT_ERASE_SETTINGS_COMP_EVT</b>
Provisioner directly erase settings completion event
<b>ESP_BLE_MESH_PROVISIONER_OPEN_SETTINGS_WITH_INDEX_COMP_EVT</b>
Provisioner open settings with index completion event
<b>ESP_BLE_MESH_PROVISIONER_OPEN_SETTINGS_WITH_UID_COMP_EVT</b>
Provisioner open settings with user id completion event
<b>ESP_BLE_MESH_PROVISIONER_CLOSE_SETTINGS_WITH_INDEX_COMP_EVT</b>
Provisioner close settings with index completion event

- ESP\_BLE\_MESH\_PROVISIONER\_CLOSE\_SETTINGS\_WITH\_UID\_COMP\_EVT**  
Provisioner close settings with user id completion event
- ESP\_BLE\_MESH\_PROVISIONER\_DELETE\_SETTINGS\_WITH\_INDEX\_COMP\_EVT**  
Provisioner delete settings with index completion event
- ESP\_BLE\_MESH\_PROVISIONER\_DELETE\_SETTINGS\_WITH\_UID\_COMP\_EVT**  
Provisioner delete settings with user id completion event
- ESP\_BLE\_MESH\_SET\_FAST\_PROV\_INFO\_COMP\_EVT**  
Set fast provisioning information (e.g. unicast address range, net\_idx, etc.) completion event
- ESP\_BLE\_MESH\_SET\_FAST\_PROV\_ACTION\_COMP\_EVT**  
Set fast provisioning action completion event
- ESP\_BLE\_MESH\_HEARTBEAT\_MESSAGE\_RECV\_EVT**  
Receive Heartbeat message event
- ESP\_BLE\_MESH\_LPN\_ENABLE\_COMP\_EVT**  
Enable Low Power Node completion event
- ESP\_BLE\_MESH\_LPN\_DISABLE\_COMP\_EVT**  
Disable Low Power Node completion event
- ESP\_BLE\_MESH\_LPN\_POLL\_COMP\_EVT**  
Low Power Node send Friend Poll completion event
- ESP\_BLE\_MESH\_LPN\_FRIENDSHIP\_ESTABLISH\_EVT**  
Low Power Node establishes friendship event
- ESP\_BLE\_MESH\_LPN\_FRIENDSHIP\_TERMINATE\_EVT**  
Low Power Node terminates friendship event
- ESP\_BLE\_MESH\_FRIEND\_FRIENDSHIP\_ESTABLISH\_EVT**  
Friend Node establishes friendship event
- ESP\_BLE\_MESH\_FRIEND\_FRIENDSHIP\_TERMINATE\_EVT**  
Friend Node terminates friendship event
- ESP\_BLE\_MESH\_PROXY\_CLIENT\_RECV\_ADV\_PKT\_EVT**  
Proxy Client receives Network ID advertising packet event
- ESP\_BLE\_MESH\_PROXY\_CLIENT\_CONNECTED\_EVT**  
Proxy Client establishes connection successfully event
- ESP\_BLE\_MESH\_PROXY\_CLIENT\_DISCONNECTED\_EVT**  
Proxy Client terminates connection successfully event
- ESP\_BLE\_MESH\_PROXY\_CLIENT\_RECV\_FILTER\_STATUS\_EVT**  
Proxy Client receives Proxy Filter Status event
- ESP\_BLE\_MESH\_PROXY\_CLIENT\_CONNECT\_COMP\_EVT**  
Proxy Client connect completion event
- ESP\_BLE\_MESH\_PROXY\_CLIENT\_DISCONNECT\_COMP\_EVT**  
Proxy Client disconnect completion event
- ESP\_BLE\_MESH\_PROXY\_CLIENT\_SET\_FILTER\_TYPE\_COMP\_EVT**  
Proxy Client set filter type completion event
- ESP\_BLE\_MESH\_PROXY\_CLIENT\_ADD\_FILTER\_ADDR\_COMP\_EVT**  
Proxy Client add filter address completion event
- ESP\_BLE\_MESH\_PROXY\_CLIENT\_REMOVE\_FILTER\_ADDR\_COMP\_EVT**  
Proxy Client remove filter address completion event
- ESP\_BLE\_MESH\_MODEL\_SUBSCRIBE\_GROUP\_ADDR\_COMP\_EVT**  
Local model subscribes group address completion event

**ESP\_BLE\_MESH\_MODEL\_UNSUBSCRIBE\_GROUP\_ADDR\_COMP\_EVT**  
Local model unsubscribes group address completion event

**ESP\_BLE\_MESH\_DEINIT\_MESH\_COMP\_EVT**  
De-initialize BLE Mesh stack completion event

**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:*

**ESP\_BLE\_MESH\_SERVER\_TRANS\_TIMER\_START**

**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:*

**ESP\_BLE\_MESH\_GENERIC\_ONOFF\_STATE**

**ESP\_BLE\_MESH\_GENERIC\_LEVEL\_STATE**

**ESP\_BLE\_MESH\_GENERIC\_ONPOWERUP\_STATE**

**ESP\_BLE\_MESH\_GENERIC\_POWER\_ACTUAL\_STATE**

**ESP\_BLE\_MESH\_LIGHT\_LIGHTNESS\_ACTUAL\_STATE**

**ESP\_BLE\_MESH\_LIGHT\_LIGHTNESS\_LINEAR\_STATE**

**ESP\_BLE\_MESH\_LIGHT\_CTL\_LIGHTNESS\_STATE**

**ESP\_BLE\_MESH\_LIGHT\_CTL\_TEMP\_DELTA\_UV\_STATE**

**ESP\_BLE\_MESH\_LIGHT\_HSL\_STATE**

**ESP\_BLE\_MESH\_LIGHT\_HSL\_LIGHTNESS\_STATE**

**ESP\_BLE\_MESH\_LIGHT\_HSL\_HUE\_STATE**

**ESP\_BLE\_MESH\_LIGHT\_HSL\_SATURATION\_STATE**

**ESP\_BLE\_MESH\_LIGHT\_XYL\_LIGHTNESS\_STATE**

**ESP\_BLE\_MESH\_LIGHT\_LC\_LIGHT\_ONOFF\_STATE**

**ESP\_BLE\_MESH\_SERVER\_MODEL\_STATE\_MAX**

**enum esp\_ble\_mesh\_model\_cb\_event\_t**

*Values:*

**ESP\_BLE\_MESH\_MODEL\_OPERATION\_EVT**

User-defined models receive messages from peer devices (e.g. get, set, status, etc) event

**ESP\_BLE\_MESH\_MODEL\_SEND\_COMP\_EVT**

User-defined models send messages completion event

**ESP\_BLE\_MESH\_MODEL\_PUBLISH\_COMP\_EVT**

User-defined models publish messages completion event

**ESP\_BLE\_MESH\_CLIENT\_MODEL\_RECV\_PUBLISH\_MSG\_EVT**

User-defined client models receive publish messages event

**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



**ESP\_BLE\_MESH\_MODEL\_PUBLISH\_UPDATE\_EVT**

When a model is configured to publish messages periodically, this event will occur during every publish period

**ESP\_BLE\_MESH\_SERVER\_MODEL\_UPDATE\_STATE\_COMP\_EVT**

Server models update state value completion event

**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*
- *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

- [bt/esp\\_ble\\_mesh/api/core/include/esp\\_ble\\_mesh\\_common\\_api.h](#)

### Functions

*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.

**Note** After calling this API, the device needs to call `esp_ble_mesh_prov_enable()` to enable provisioning functionality again.

**Return** ESP\_OK on success or error code otherwise.

#### Parameters

- [in] prov: Pointer to the device provisioning capabilities. This pointer must remain valid during the lifetime of the BLE Mesh device.
- [in] comp: Pointer to the device composition data information. This pointer must remain valid during the lifetime of the BLE Mesh device.

*esp\_err\_t* **esp\_ble\_mesh\_deinit** (*esp\_ble\_mesh\_deinit\_param\_t* \*param)

De-initialize BLE Mesh module.

**Note** This function shall be invoked after `esp_ble_mesh_client_model_deinit()`.

**Return** ESP\_OK on success or error code otherwise.

#### Parameters

- [in] param: Pointer to the structure of BLE Mesh deinit parameters.

## Reading of Local Data Information

### Header File

- [bt/esp\\_ble\\_mesh/api/core/include/esp\\_ble\\_mesh\\_local\\_data\\_operation\\_api.h](#)

**Functions**

`int32_t esp_ble_mesh_get_model_publish_period(esp_ble_mesh_model_t *model)`

Get the model publish period, the unit is ms.

**Return** Publish period value on success, 0 or (negative) error code from `errno.h` on failure.

**Parameters**

- [in] `model`: Model instance pointer.

`uint16_t esp_ble_mesh_get_primary_element_address(void)`

Get the address of the primary element.

**Return** 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)`

Check if the model has subscribed to the given group address. Note: E.g., once a status message is received and the destination address is a group address, the model uses this API to check if it is successfully subscribed to the given group address.

**Return** Pointer to the group address within the Subscription List of the model on success, or `NULL` on failure which means the model has not subscribed to the given group address. Note: With the pointer to the group address returned, you can reset the group address to `0x0000` in order to unsubscribe the model from the group.

**Parameters**

- [in] `model`: Pointer to the model.
- [in] `group_addr`: Group address.

`esp_ble_mesh_elem_t *esp_ble_mesh_find_element(uint16_t element_addr)`

Find the BLE Mesh element pointer via the element address.

**Return** Pointer to the element on success, or `NULL` on failure.

**Parameters**

- [in] `element_addr`: Element address.

`uint8_t esp_ble_mesh_get_element_count(void)`

Get the number of elements that have been registered.

**Return** Number of elements.

`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)`

Find the Vendor specific model with the given element, the company ID and the Vendor Model ID.

**Return** Pointer to the Vendor Model on success, or `NULL` on failure which means the Vendor Model is not found.

**Parameters**

- [in] `element`: Element to which the model belongs.
- [in] `company_id`: A 16-bit company identifier assigned by the Bluetooth SIG.
- [in] `model_id`: A 16-bit vendor-assigned model identifier.

`esp_ble_mesh_model_t *esp_ble_mesh_find_sig_model(const esp_ble_mesh_elem_t *element, uint16_t model_id)`

Find the SIG model with the given element and Model id.

**Return** Pointer to the SIG Model on success, or `NULL` on failure which means the SIG Model is not found.

**Parameters**

- [in] `element`: Element to which the model belongs.
- [in] `model_id`: SIG model identifier.

`const esp_ble_mesh_comp_t *esp_ble_mesh_get_composition_data(void)`

Get the Composition data which has been registered.

**Return** Pointer to the Composition data on success, or `NULL` on failure which means the Composition data is not initialized.

*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*)

A local model of node or Provisioner subscribes a group address.

**Note** This function shall not be invoked before node is provisioned or Provisioner is enabled.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] *element\_addr*: Unicast address of the element to which the model belongs.
- [in] *company\_id*: A 16-bit company identifier.
- [in] *model\_id*: A 16-bit model identifier.
- [in] *group\_addr*: The group address to be subscribed.

*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.

**Note** This function shall not be invoked before node is provisioned or Provisioner is enabled.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] *element\_addr*: Unicast address of the element to which the model belongs.
- [in] *company\_id*: A 16-bit company identifier.
- [in] *model\_id*: A 16-bit model identifier.
- [in] *group\_addr*: The subscribed 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.

**Return** NetKey on success, or NULL on failure.

**Parameters**

- [in] *net\_idx*: NetKey index.

**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.

**Return** AppKey on success, or NULL on failure.

**Parameters**

- [in] *app\_idx*: AppKey index.

*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.

**Note** This function can only be called after the device is provisioned.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] *net\_key*: NetKey to be added.
- [in] *net\_idx*: NetKey Index.

*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.

**Note** The *net\_idx* must be an existing one. This function can only be called after the device is provisioned.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] *app\_key*: AppKey to be added.
- [in] *net\_idx*: NetKey Index.
- [in] *app\_idx*: AppKey Index.

*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.

**Note** 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.

**Return** `ESP_OK` on success or error code otherwise.

#### Parameters

- [in] `element_addr`: Node local element address
- [in] `company_id`: Node local company id
- [in] `model_id`: Node local model id
- [in] `app_idx`: Node local appkey index

## Low Power Operation (Updating)

### Header File

- [bt/esp\\_ble\\_mesh/api/core/include/esp\\_ble\\_mesh\\_low\\_power\\_api.h](#)

### Functions

*esp\_err\_t* **esp\_ble\_mesh\_lpn\_enable** (void)

Enable BLE Mesh device LPN functionality.

**Note** This API enables LPN functionality. Once called, the proper Friend Request will be sent.

**Return** `ESP_OK` on success or error code otherwise.

*esp\_err\_t* **esp\_ble\_mesh\_lpn\_disable** (bool *force*)

Disable BLE Mesh device LPN functionality.

**Return** `ESP_OK` on success or error code otherwise.

#### Parameters

- [in] `force`: 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.

*esp\_err\_t* **esp\_ble\_mesh\_lpn\_poll** (void)

LPN tries to poll messages from the Friend Node.

**Note** 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.

**Return** `ESP_OK` on success or error code otherwise.

## Send/Publish Messages, add Local AppKey, etc.

### Header File

- [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.

**Note** 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.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] `callback`: Pointer to the callback function.

*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.

**Note** This API is only used to set the opcode of the message.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] `data`: Pointer to the message data.
- [in] `opcode`: The message opcode.

*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` variables within the `user_data` (defined using `esp_ble_mesh_client_t`) of the client model need to be initialized.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] `model`: BLE Mesh Client model to which the message belongs.

*esp\_err\_t* `esp_ble_mesh_client_model_deinit` (*esp\_ble\_mesh\_model\_t* \*model)

De-initialize the user-defined client model.

**Note** This function shall be invoked before `esp_ble_mesh_deinit()` is called.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] `model`: Pointer of the Client model.

*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).

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] `model`: BLE Mesh Server Model to which the message belongs.
- [in] `ctx`: Message context, includes keys, TTL, etc.
- [in] `opcode`: Message opcode.
- [in] `length`: Message length (exclude the message opcode).
- [in] `data`: Parameters of Access Payload (exclude the message opcode) to be sent.

*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 message (such as model get, set, etc).

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] `model`: BLE Mesh Client Model to which the message belongs.

- [in] ctx: Message context, includes keys, TTL, etc.
- [in] opcode: Message opcode.
- [in] length: Message length (exclude the message opcode).
- [in] data: Parameters of the Access Payload (exclude the message opcode) to be sent.
- [in] msg\_timeout: Time to get response to the message (in milliseconds).
- [in] need\_rsp: TRUE if the opcode requires the peer device to reply, FALSE otherwise.
- [in] device\_role: Role of the device (Node/Provisioner) that sends the message.

*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.

**Note** 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.

**Return** ESP\_OK on success or error code otherwise.

#### Parameters

- [in] model: Mesh (client) Model publishing the message.
- [in] opcode: Message opcode.
- [in] length: Message length (exclude the message opcode).
- [in] data: Parameters of the Access Payload (exclude the message opcode) to be sent.
- [in] device\_role: Role of the device (node/provisioner) publishing the message of the type *esp\_ble\_mesh\_dev\_role\_t*.

*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.

**Note** Currently this API is used to update bound state value, not for all server model states.

**Return** ESP\_OK on success or error code otherwise.

#### Parameters

- [in] model: Server model which is going to update the state.
- [in] type: Server model state type.
- [in] value: Server model state value.

*esp\_err\_t* **esp\_ble\_mesh\_node\_local\_reset** (void)

Reset the provisioning procedure of the local BLE Mesh node.

**Note** 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.

**Return** ESP\_OK on success or error code otherwise.

*esp\_err\_t* **esp\_ble\_mesh\_provisioner\_set\_node\_name** (uint16\_t index, const char \*name)

This function is called to set the node (provisioned device) name.

**Note** index is obtained from the parameters of ESP\_BLE\_MESH\_PROVISIONER\_PROV\_COMPLETE\_EVT.

**Return** ESP\_OK on success or error code otherwise.

#### Parameters

- [in] index: Index of the node in the node queue.
- [in] name: Name (end by '\0') to be set for the node.

const char \***esp\_ble\_mesh\_provisioner\_get\_node\_name** (uint16\_t index)

This function is called to get the node (provisioned device) name.

**Note** index is obtained from the parameters of ESP\_BLE\_MESH\_PROVISIONER\_PROV\_COMPLETE\_EVT.

**Return** Node name on success, or NULL on failure.

#### Parameters

- [in] index: Index of the node in the node queue.

`uint16_t esp_ble_mesh_provisioner_get_node_index (const char *name)`

This function is called to get the node (provisioned device) index.

**Return** Node index on success, or an invalid value (0xFFFF) on failure.

**Parameters**

- [in] name: Name of the node (end by ‘\0’).

`esp_err_t 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.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] unicast\_addr: Element address of the node
- [in] data: Pointer of Composition Data
- [in] length: Length of Composition Data

`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.

**Return** Pointer of the node info struct or NULL on failure.

**Parameters**

- [in] uuid: Device UUID of the node

`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.

**Return** Pointer of the node info struct or NULL on failure.

**Parameters**

- [in] unicast\_addr: Unicast address of the node

`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.

**Return** Pointer of the node info struct or NULL on failure.

**Parameters**

- [in] name: Name of the node (end by ‘\0’).

`uint16_t esp_ble_mesh_provisioner_get_prov_node_count (void)`

This function is called by Provisioner to get provisioned node count.

**Return** 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.

**Note** 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) { ... } }` “

**Return** 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.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] uuid: Device UUID of the node

*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.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] *unicast\_addr*: Unicast address of the node

*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.

**Note** *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.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] *app\_key*: The app key to be set for the local BLE Mesh stack.
- [in] *net\_idx*: The network key index.
- [in] *app\_idx*: The app key index.

*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.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] *app\_key*: Value of the AppKey.
- [in] *net\_idx*: Corresponding NetKey Index.
- [in] *app\_idx*: The AppKey Index

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.

**Return** App key on success, or NULL on failure.

**Parameters**

- [in] *net\_idx*: Network key index.
- [in] *app\_idx*: Application key index.

*esp\_err\_t* **esp\_ble\_mesh\_provisioner\_bind\_app\_key\_to\_local\_model** (uint16\_t *el-*  
*ement\_addr*,  
uint16\_t *app\_idx*,  
uint16\_t  
*model\_id*,  
uint16\_t *com-*  
*pany\_id*)

This function is called by Provisioner to bind own model with proper app key.

**Note** *company\_id*: If going to bind *app\_key* with local vendor model, *company\_id* should be set to 0xFFFF.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] *element\_addr*: Provisioner local element address
- [in] *app\_idx*: Provisioner local appkey index
- [in] *model\_id*: Provisioner local model id
- [in] *company\_id*: Provisioner local company id

*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.

**Note** *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.



**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] net\_key: The network key to be added to the Provisioner local BLE Mesh stack.
- [in] net\_idx: The network key index.

*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.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] net\_key: Value of the NetKey.
- [in] net\_idx: The NetKey Index.

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.

**Return** Network key on success, or NULL on failure.

**Parameters**

- [in] net\_idx: Network key index.

*esp\_err\_t* **esp\_ble\_mesh\_provisioner\_recv\_heartbeat** (bool enable)

This function is called by Provisioner to enable or disable receiving heartbeat messages.

**Note** 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.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] enable: Enable or disable receiving heartbeat messages.

*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.

**Note** 1. If the filter type is not the same with the current value, then all the filter entries will be cleaned.

1. 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.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] type: Heartbeat filter type (acceptlist or rejectlist).

*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.

1. 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.

**Note** 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.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] op: Add or REMOVE

- [in] info: Heartbeat filter entry information, including: hb\_src - Heartbeat source address; hb\_dst - Heartbeat destination address;

*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.

**Note** This function can be invoked when the mesh stack is not initialized or has been de-initialized.

**Return** 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.

**Note** Before open another nvs namespace, the previously opened nvs namespace must be closed firstly.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] index: Settings index.

*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.

**Note** Before open another nvs namespace, the previously opened nvs namespace must be closed firstly.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] uid: Settings user id.

*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.

**Note** 1. Before closing the nvs namespace, it must be open.

1. 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.
2. 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.
3. 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) ……

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] index: Settings index.
- [in] erase: Indicate if erasing mesh information.

*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.

**Note** 1. Before closing the nvs namespace, it must be open.

1. 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.
2. 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.
3. 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) ……

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] uid: Settings user id.
- [in] erase: Indicate if erasing mesh information.

*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.

**Note** 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.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] *index*: Settings index.

*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.

**Note** 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.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] *uid*: Settings user id.

const char \***esp\_ble\_mesh\_provisioner\_get\_settings\_uid** (uint8\_t *index*)

This function is called by Provisioner to get settings user id.

**Return** Setting user id on success or NULL on failure.

**Parameters**

- [in] *index*: Settings index.

uint8\_t **esp\_ble\_mesh\_provisioner\_get\_settings\_index** (const char \**uid*)

This function is called by Provisioner to get settings index.

**Return** Settings index.

**Parameters**

- [in] *uid*: Settings user id.

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.

**Return** 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.

**Return** Application key on success, or NULL on failure.

**Parameters**

- [in] *net\_idx*: Network key index.
- [in] *app\_idx*: Application key index.

## 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

- [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.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] `callback`: Pointer to the callback function.

bool **esp\_ble\_mesh\_node\_is\_provisioned** (void)

Check if a device has been provisioned.

**Return** 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.

**Note** PB-ADV: send unprovisioned device beacon. PB-GATT: send connectable advertising packets.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- *bearers*: Bit-wise OR of provisioning bearers.

*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.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- *bearers*: Bit-wise OR of provisioning bearers.

*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.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] *pub\_key\_x*: Unprovisioned device' s Public Key X
- [in] *pub\_key\_y*: Unprovisioned device' s Public Key Y
- [in] *private\_key*: Unprovisioned device' s Private Key

*esp\_err\_t* **esp\_ble\_mesh\_node\_input\_number** (uint32\_t *number*)

Provide provisioning input OOB number.

**Note** 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.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] *number*: Number input by device.

*esp\_err\_t* **esp\_ble\_mesh\_node\_input\_string** (const char \**string*)

Provide provisioning input OOB string.

**Note** 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.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] *string*: String input by device.

*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.

**Note** 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.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] *name*: Unprovisioned device name

*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.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] link\_idx: The provisioning link index
- [in] pub\_key\_x: Unprovisioned device's Public Key X
- [in] pub\_key\_y: Unprovisioned device's Public Key Y

*esp\_err\_t* **esp\_ble\_mesh\_provisioner\_input\_string** (const char \*string, uint8\_t link\_idx)

Provide provisioning input OOB string.

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.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] string: String input by Provisioner.
- [in] link\_idx: The provisioning link index.

*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.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] number: Number input by Provisioner.
- [in] link\_idx: The provisioning link index.

*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.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] bearers: Bit-wise OR of provisioning bearers.

*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.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] bearers: Bit-wise OR of provisioning bearers.

*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.

**Return** ESP\_OK on success or error code otherwise.

**Note** : 1. Currently address type only supports public address and static random address.

1. 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.
2. 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.
3. 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` can not be enabled at the same time.

#### Parameters

- [in] `add_dev`: Pointer to a struct containing the device information
- [in] `flags`: 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)

```
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.

**Return** Zero on success or (negative) error code otherwise.

**Note** : 1. Currently address type only supports public address and static random address.

1. 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.
2. 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.
3. 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.

#### Parameters

- [in] `uuid`: Device UUID of the unprovisioned device
- [in] `addr`: Device address of the unprovisioned device
- [in] `addr_type`: Device address type of the unprovisioned device
- [in] `bearer`: Provisioning bearer going to be used by Provisioner
- [in] `oob_info`: OOB info of the unprovisioned device
- [in] `unicast_addr`: Unicast address going to be allocated for the unprovisioned device

```
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.
```

**Note** 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.

**Return** `ESP_OK` on success or error code otherwise.

#### Parameters

- [in] `del_dev`: Pointer to a struct containing the device information.

*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.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] match\_val: Value to be compared with the part of the device UUID.
- [in] match\_len: Length of the compared match value.
- [in] offset: Offset of the device UUID to be compared (based on zero).
- [in] prov\_after\_match: Flag used to indicate whether provisioner should start to provision the device immediately if the part of the UUID matches.

*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.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] prov\_data\_info: Pointer to a struct containing net\_idx or flags or iv\_index.

*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.

**Note** 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).

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] value: Pointer to the static oob value.
- [in] length: 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.

**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.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] addr: Unicast address of the Primary element of Provisioner.

*esp\_err\_t* **esp\_ble\_mesh\_set\_fast\_prov\_info** (*esp\_ble\_mesh\_fast\_prov\_info\_t* \*fast\_prov\_info)

This function is called to set provisioning data information before starting fast provisioning.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] fast\_prov\_info: Pointer to a struct containing unicast address range, net\_idx, etc.

*esp\_err\_t* **esp\_ble\_mesh\_set\_fast\_prov\_action** (*esp\_ble\_mesh\_fast\_prov\_action\_t* action)

This function is called to start/suspend/exit fast provisioning.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] action: fast provisioning action (i.e. enter, suspend, exit).

## 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_prov_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, uint16_t oob_info,
                                             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

- [in] addr: Pointer to the unprovisioned device address.
- [in] addr\_type: Unprovisioned device address type.
- [in] adv\_type: Adv packet type(ADV\_IND or ADV\_NONCONN\_IND).
- [in] dev\_uuid: Unprovisioned device UUID pointer.
- [in] oob\_info: OOB information of the unprovisioned device.
- [in] bearer: Adv packet received from PB-GATT or PB-ADV bearer.

## ESP-BLE-MESH GATT Proxy Server

### Header File

- [bt/esp\\_ble\\_mesh/api/core/include/esp\\_ble\\_mesh\\_proxy\\_api.h](#)

### Functions

*esp\_err\_t* **esp\_ble\_mesh\_proxy\_identity\_enable** (void)

Enable advertising with Node Identity.

**Note** This API requires that GATT Proxy support be enabled. Once called, each subnet starts advertising using Node Identity for the next 60 seconds, and after 60s Network ID will be advertised. Under normal conditions, the BLE Mesh Proxy Node Identity and Network ID advertising will be enabled automatically by BLE Mesh stack after the device is provisioned.

**Return** ESP\_OK on success or error code otherwise.

*esp\_err\_t* **esp\_ble\_mesh\_proxy\_gatt\_enable** (void)

Enable BLE Mesh GATT Proxy Service.

**Return** ESP\_OK on success or error code otherwise.

*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.

**Return** ESP\_OK on success or error code otherwise.

*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.

**Return** ESP\_OK on success or error code otherwise.

#### Parameters

- [in] addr: Device address of the Proxy Server.
- [in] addr\_type: Device address type(public or static random).
- [in] net\_idx: NetKey Index related with Network ID in the Mesh Proxy advertising packet.

*esp\_err\_t* **esp\_ble\_mesh\_proxy\_client\_disconnect** (uint8\_t conn\_handle)

Proxy Client terminates a connection with the Proxy Server.

**Return** ESP\_OK on success or error code otherwise.



**Parameters**

- [in] `conn_handle`: Proxy connection handle.

`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.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] `conn_handle`: Proxy connection handle.
- [in] `net_idx`: Corresponding NetKey Index.
- [in] `filter_type`: whitelist or blacklist.

`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.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] `conn_handle`: Proxy connection handle.
- [in] `net_idx`: Corresponding NetKey Index.
- [in] `addr`: Pointer to the filter address.
- [in] `addr_num`: Number of the filter address.

`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.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] `conn_handle`: Proxy connection handle.
- [in] `net_idx`: Corresponding NetKey Index.
- [in] `addr`: Pointer to the filter address.
- [in] `addr_num`: Number of the filter address.

**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*

---

注解: Definitions related to Server Models are being updated, and will be released soon.

---

**Configuration Client/Server Models****Header File**

- `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.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] callback: Pointer to the callback function.

*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.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] callback: Pointer to the callback function.

*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.

**Note** 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

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] params: Pointer to BLE Mesh common client parameters.
- [in] get\_state: Pointer to a union, each kind of opcode corresponds to one structure inside. Shall not be set to NULL.

*esp\_err\_t* **esp\_ble\_mesh\_config\_client\_set\_state** (*esp\_ble\_mesh\_client\_common\_param\_t* \*params, *esp\_ble\_mesh\_cfg\_client\_set\_state\_t* \*set\_state)

Set the value of the Configuration Server Model states using the Config Client Model set messages.

**Note** 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

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] params: Pointer to BLE Mesh common client parameters.
- [in] set\_state: Pointer to a union, each kind of opcode corresponds to one structure inside. Shall not be set to NULL.

**Unions**

**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  
 the get\_state parameter in the esp\_ble\_mesh\_config\_client\_get\_state function should not be set to NULL.

**Public Members**

*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.

*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\_kr\_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  
ESP\_BLE\_MESH\_MODEL\_OP\_MODEL\_SUB\_DELETE ESP\_BLE\_MESH\_MODEL\_OP\_MODEL\_SUB\_VIRTUAL\_ADDR  
ESP\_BLE\_MESH\_MODEL\_OP\_MODEL\_SUB\_OVERWRITE ESP\_BLE\_MESH\_MODEL\_OP\_MODEL\_SUB\_VIRTUAL\_ADDR  
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

*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 model_app_unbind
    For ESP_BLE_MESH_MODEL_OP_MODEL_APP_UNBIND

esp_ble_mesh_cfg_kr_phase_set_t kr_phase_set
    For ESP_BLE_MESH_MODEL_OP_KEY_REFRESH_PHASE_SET

esp_ble_mesh_cfg_net_transmit_set_t net_transmit_set
    For ESP_BLE_MESH_MODEL_OP_NETWORK_TRANSMIT_SET

```

```

union esp_ble_mesh_cfg_client_common_cb_param_t
#include <esp_ble_mesh_config_model_api.h> Configuration Client Model received message union.

```

### Public Members

```

esp_ble_mesh_cfg_beacon_status_cb_t beacon_status
    The beacon status value

esp_ble_mesh_cfg_comp_data_status_cb_t comp_data_status
    The composition data status value

esp_ble_mesh_cfg_default_ttl_status_cb_t default_ttl_status
    The default_ttl status value

esp_ble_mesh_cfg_gatt_proxy_status_cb_t gatt_proxy_status
    The gatt_proxy status value

```

*esp\_ble\_mesh\_cfg\_relay\_status\_cb\_t* **relay\_status**

The relay status value

*esp\_ble\_mesh\_cfg\_model\_pub\_status\_cb\_t* **model\_pub\_status**

The model publication status value

*esp\_ble\_mesh\_cfg\_model\_sub\_status\_cb\_t* **model\_sub\_status**

The model subscription status value

*esp\_ble\_mesh\_cfg\_net\_key\_status\_cb\_t* **netkey\_status**

The netkey status value

*esp\_ble\_mesh\_cfg\_app\_key\_status\_cb\_t* **appkey\_status**

The appkey status value

*esp\_ble\_mesh\_cfg\_mod\_app\_status\_cb\_t* **model\_app\_status**

The model app status value

*esp\_ble\_mesh\_cfg\_friend\_status\_cb\_t* **friend\_status**

The friend status value

*esp\_ble\_mesh\_cfg\_hb\_pub\_status\_cb\_t* **heartbeat\_pub\_status**

The heartbeat publication status value

*esp\_ble\_mesh\_cfg\_hb\_sub\_status\_cb\_t* **heartbeat\_sub\_status**

The heartbeat subscription status value

*esp\_ble\_mesh\_cfg\_net\_trans\_status\_cb\_t* **net\_transmit\_status**

The network transmit status value

*esp\_ble\_mesh\_cfg\_model\_sub\_list\_cb\_t* **model\_sub\_list**

The model subscription list value

*esp\_ble\_mesh\_cfg\_net\_key\_list\_cb\_t* **netkey\_list**

The network key index list value

*esp\_ble\_mesh\_cfg\_app\_key\_list\_cb\_t* **appkey\_list**

The application key index list value

*esp\_ble\_mesh\_cfg\_node\_id\_status\_cb\_t* **node\_identity\_status**

The node identity status value

*esp\_ble\_mesh\_cfg\_model\_app\_list\_cb\_t* **model\_app\_list**

The model application key index list value

*esp\_ble\_mesh\_cfg\_kr\_phase\_status\_cb\_t* **kr\_phase\_status**

The key refresh phase status value

*esp\_ble\_mesh\_cfg\_lpn\_pollto\_status\_cb\_t* **lpn\_timeout\_status**

The low power node poll timeout status value

**union esp\_ble\_mesh\_cfg\_server\_state\_change\_t**

*#include <esp\_ble\_mesh\_config\_model\_api.h>* Configuration Server model state change value union.

### Public Members

*esp\_ble\_mesh\_state\_change\_cfg\_mod\_pub\_set\_t* **mod\_pub\_set**

The `recv_op` in `ctx` can be used to decide which state is changed. Config Model Publication Set

*esp\_ble\_mesh\_state\_change\_cfg\_model\_sub\_add\_t* **mod\_sub\_add**

Config Model Subscription Add

*esp\_ble\_mesh\_state\_change\_cfg\_model\_sub\_delete\_t* **mod\_sub\_delete**

Config Model Subscription Delete

*esp\_ble\_mesh\_state\_change\_cfg\_netkey\_add\_t* **netkey\_add**

Config NetKey Add

*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 max\_hops**

Maximum hops when receiving Heartbeat messages

**esp\_ble\_mesh\_cb\_t heartbeat\_rcv\_cb**

Optional heartbeat subscription tracking function

**struct esp\_ble\_mesh\_cfg\_srv::[anonymous] heartbeat\_sub**

Heartbeat Subscription

**struct esp\_ble\_mesh\_cfg\_composition\_data\_get\_t**

Parameters of Config Composition Data Get.

### Public Members

**uint8\_t page**

Page number of the Composition Data.

**struct esp\_ble\_mesh\_cfg\_model\_pub\_get\_t**

Parameters of Config Model Publication Get.

### 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\_sig\_model\_sub\_get\_t**

Parameters of Config SIG Model Subscription Get.

**Public Members**

`uint16_t element_addr`  
The element address

`uint16_t model_id`  
The model id

**struct esp\_ble\_mesh\_cfg\_vnd\_model\_sub\_get\_t**  
Parameters of Config Vendor Model Subscription Get.

**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

`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

**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

**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

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

**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

**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** : 3  
Number of transmissions for each Network PDU originating from the node

**uint8\_t net\_trans\_step** : 5  
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 \*sub\_addr**  
A block of all addresses from the Subscription List

**struct esp\_ble\_mesh\_cfg\_net\_key\_list\_cb\_t**  
Parameter of Config NetKey List

### Public Members

**struct net\_buf\_simple \*net\_idx**  
A list of NetKey Indexes known to the node

**struct esp\_ble\_mesh\_cfg\_app\_key\_list\_cb\_t**  
Parameters of Config AppKey List

### Public Members

**uint8\_t status**  
Status Code for the request message

**uint16\_t net\_idx**  
NetKey Index of the NetKey that the AppKeys are bound to

**struct net\_buf\_simple \*app\_idx**  
A list of AppKey indexes that are bound to the NetKey identified by NetKeyIndex

**struct esp\_ble\_mesh\_cfg\_node\_id\_status\_cb\_t**  
Parameters of Config Node Identity Status

### Public Members

**uint8\_t status**  
Status Code for the request message

**uint16\_t net\_idx**  
Index of the NetKey

**uint8\_t identity**  
Node Identity state

**struct esp\_ble\_mesh\_cfg\_model\_app\_list\_cb\_t**  
Parameters of Config SIG/Vendor Model App 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 \*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

`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\_model\_sub\_delete\_t**  
Parameters of Config Model Subscription Delete

### Public Members

`uint16_t element_addr`  
Element Address

`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

`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

uint16\_t **app\_idx**  
AppKey Index

**struct esp\_ble\_mesh\_state\_change\_cfg\_model\_app\_bind\_t**  
Parameters of Config Model App Bind

### Public Members

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 esp\_ble\_mesh\_state\_change\_cfg\_model\_app\_unbind\_t**  
Parameters of Config Model App Unbind

### Public Members

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 esp\_ble\_mesh\_state\_change\_cfg\_kr\_phase\_set\_t**  
Parameters of Config Key Refresh Phase Set

### Public Members

uint16\_t **net\_idx**  
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.

**Note** The Config Server Model can only be included by a Primary Element.

**Return** New Config Server Model instance.

#### Parameters

- *srv\_data*: Pointer to a unique Config Server Model user\_data.

**ESP\_BLE\_MESH\_MODEL\_CFG\_CLI** (cli\_data)  
Define a new Config Client Model.

**Note** The Config Client Model can only be included by a Primary Element.

**Return** New Config Client Model instance.

#### Parameters

- *cli\_data*: Pointer to a unique struct *esp\_ble\_mesh\_client\_t*.

### 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

#### Parameters

- *event*: Event type
- *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.



**Parameters**

- `event`: Event type
- `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:

**ESP\_BLE\_MESH\_CFG\_CLIENT\_GET\_STATE\_EVT**

**ESP\_BLE\_MESH\_CFG\_CLIENT\_SET\_STATE\_EVT**

**ESP\_BLE\_MESH\_CFG\_CLIENT\_PUBLISH\_EVT**

**ESP\_BLE\_MESH\_CFG\_CLIENT\_TIMEOUT\_EVT**

**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:

**ESP\_BLE\_MESH\_CFG\_SERVER\_STATE\_CHANGE\_EVT**

**ESP\_BLE\_MESH\_CFG\_SERVER\_EVT\_MAX**

**Health Client/Server Models****Header File**

- [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.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] `callback`: Pointer to the callback function.

**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.

**Note** 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`

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] `params`: Pointer to BLE Mesh common client parameters.
- [in] `get_state`: Pointer to generic get message value. Shall not be set to NULL.

**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.

**Note** 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`

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] `params`: Pointer to BLE Mesh common client parameters.
- [in] `set_state`: Pointer to generic set message value. Shall not be set to NULL.

`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.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] `callback`: Pointer to the callback function.

## 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

esp_ble_mesh_gen_admin_property_set_t admin_property_set
    For ESP_BLE_MESH_MODEL_OP_GEN_ADMIN_PROPERTY_SET &
    ESP_BLE_MESH_MODEL_OP_GEN_ADMIN_PROPERTY_SET_UNACK

esp_ble_mesh_gen_manufacturer_property_set_t manufacturer_property_set
    For ESP_BLE_MESH_MODEL_OP_GEN_MANUFACTURER_PROPERTY_SET &
    ESP_BLE_MESH_MODEL_OP_GEN_MANUFACTURER_PROPERTY_SET_UNACK

```

```

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

esp_ble_mesh_state_change_gen_power_range_set_t power_range_set
    Generic Power Range Set

esp_ble_mesh_state_change_gen_loc_global_set_t loc_global_set
    Generic Location Global Set

esp_ble_mesh_state_change_gen_loc_local_set_t loc_local_set
    Generic Location Local Set

esp_ble_mesh_state_change_gen_user_property_set_t user_property_set
    Generic User Property Set

esp_ble_mesh_state_change_gen_admin_property_set_t admin_property_set
    Generic Admin Property Set

esp_ble_mesh_state_change_gen_manu_property_set_t manu_property_set
    Generic Manufacturer Property Set

```

```

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**

*esp\_ble\_mesh\_server\_rcv\_gen\_user\_property\_get\_t* **user\_property**  
Generic User Property Get

*esp\_ble\_mesh\_server\_rcv\_gen\_admin\_property\_get\_t* **admin\_property**  
Generic Admin Property Get

*esp\_ble\_mesh\_server\_rcv\_gen\_manufacturer\_property\_get\_t* **manu\_property**  
Generic Manufacturer Property Get

*esp\_ble\_mesh\_server\_rcv\_gen\_client\_properties\_get\_t* **client\_properties**  
Generic Client Properties Get

**union esp\_ble\_mesh\_generic\_server\_rcv\_set\_msg\_t**  
#include <esp\_ble\_mesh\_generic\_model\_api.h> Generic Server Model received set message union.

**Public Members**

*esp\_ble\_mesh\_server\_rcv\_gen\_onoff\_set\_t* **onoff**  
Generic OnOff Set/Generic OnOff Set Unack

*esp\_ble\_mesh\_server\_rcv\_gen\_level\_set\_t* **level**  
Generic Level Set/Generic Level Set Unack

*esp\_ble\_mesh\_server\_rcv\_gen\_delta\_set\_t* **delta**  
Generic Delta Set/Generic Delta Set Unack

*esp\_ble\_mesh\_server\_rcv\_gen\_move\_set\_t* **move**  
Generic Move Set/Generic Move Set Unack

*esp\_ble\_mesh\_server\_rcv\_gen\_def\_trans\_time\_set\_t* **def\_trans\_time**  
Generic Default Transition Time Set/Generic Default Transition Time Set Unack

*esp\_ble\_mesh\_server\_rcv\_gen\_onpowerup\_set\_t* **onpowerup**  
Generic OnPowerUp Set/Generic OnPowerUp Set Unack

*esp\_ble\_mesh\_server\_rcv\_gen\_power\_level\_set\_t* **power\_level**  
Generic Power Level Set/Generic Power Level Set Unack

*esp\_ble\_mesh\_server\_rcv\_gen\_power\_default\_set\_t* **power\_default**  
Generic Power Default Set/Generic Power Default Set Unack

*esp\_ble\_mesh\_server\_rcv\_gen\_power\_range\_set\_t* **power\_range**  
Generic Power Range Set/Generic Power Range Set Unack

*esp\_ble\_mesh\_server\_rcv\_gen\_loc\_global\_set\_t* **location\_global**  
Generic Location Global Set/Generic Location Global Set Unack

*esp\_ble\_mesh\_server\_rcv\_gen\_loc\_local\_set\_t* **location\_local**  
Generic Location Local Set/Generic Location Local Set Unack

*esp\_ble\_mesh\_server\_rcv\_gen\_user\_property\_set\_t* **user\_property**  
Generic User Property Set/Generic User Property Set Unack

*esp\_ble\_mesh\_server\_rcv\_gen\_admin\_property\_set\_t* **admin\_property**  
Generic Admin Property Set/Generic Admin Property Set Unack

*esp\_ble\_mesh\_server\_rcv\_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

*esp\_ble\_mesh\_generic\_server\_state\_change\_t* **state\_change**  
ESP\_BLE\_MESH\_GENERIC\_SERVER\_STATE\_CHANGE\_EVT

*esp\_ble\_mesh\_generic\_server\_rcv\_get\_msg\_t* **get**  
ESP\_BLE\_MESH\_GENERIC\_SERVER\_RECV\_GET\_MSG\_EVT

*esp\_ble\_mesh\_generic\_server\_rcv\_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

**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\_set\_t**  
Parameters of Generic Location Local Set.

### 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

**struct esp\_ble\_mesh\_gen\_user\_property\_get\_t**  
Parameter of Generic User Property Get.



**Public Members**

`uint16_t property_id`  
Property ID identifying a Generic User Property

`struct esp_ble_mesh_gen_user_property_set_t`  
Parameters of Generic User Property Set.

**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_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****uint16\_t property\_id**

A starting Client Property ID present within an element

**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****bool op\_en**

Indicate if optional parameters are included

**uint8\_t present\_onoff**

Current value of Generic OnOff state

**uint8\_t target\_onoff**

Target value of Generic OnOff state (optional)

**uint8\_t remain\_time**

Time to complete state transition (C.1)

**struct esp\_ble\_mesh\_gen\_level\_status\_cb\_t**

Parameters of Generic Level Status.

**Public Members****bool op\_en**

Indicate if optional parameters are included

**int16\_t present\_level**

Current value of Generic Level state

**int16\_t target\_level**

Target value of the Generic Level state (optional)

**uint8\_t remain\_time**

Time to complete state transition (C.1)

**struct esp\_ble\_mesh\_gen\_def\_trans\_time\_status\_cb\_t**

Parameter of Generic Default Transition Time Status.

**Public Members****uint8\_t trans\_time**

The value of the Generic Default Transition Time state

**struct esp\_ble\_mesh\_gen\_onpowerup\_status\_cb\_t**

Parameter of Generic OnPowerUp Status.

**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 : 8**  
Value of Generic Battery Level state

**uint32\_t time\_to\_discharge : 24**  
Value of Generic Battery Time to Discharge state

**uint32\_t time\_to\_charge : 24**  
Value of Generic Battery Time to Charge state

**uint32\_t flags : 8**  
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

**struct esp\_ble\_mesh\_gen\_user\_properties\_status\_cb\_t**  
Parameter of Generic User Properties Status.

#### Public Members

**struct net\_buf\_simple \*property\_ids**  
Buffer contains a sequence of N User Property IDs

**struct esp\_ble\_mesh\_gen\_user\_property\_status\_cb\_t**  
Parameters of Generic User Property Status.

#### Public Members

**bool op\_en**  
Indicate if optional parameters are included

**uint16\_t property\_id**  
Property ID identifying a Generic User Property

**uint8\_t user\_access**  
Enumeration indicating user access (optional)

**struct net\_buf\_simple \*property\_value**  
Raw value for the User Property (C.1)

**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

***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\_level, 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* **transition**  
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**  
Parameter 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

*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**

Pointer 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 tt\_delta\_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 : 8  
The value of the Generic Battery Level state

uint32\_t time\_to\_discharge : 24  
The value of the Generic Battery Time to Discharge state

uint32\_t time\_to\_charge : 24  
The value of the Generic Battery Time to Charge state

uint32\_t battery\_flags : 8  
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\_state\_t**  
Parameters of Generic Location state

**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\_srv\_t**  
User data of Generic Location Server Model

**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

**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

**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\_rcv\_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\_rcv\_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_rcv_gen_manufacturer_property_get_t`  
Context of the received Generic Manufacturer Property message

**Public Members**

`uint16_t property_id`  
Property ID identifying a Generic Manufacturer Property

`struct esp_ble_mesh_server_rcv_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_rcv_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_rcv_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

`uint8_t trans_time`  
Time to complete state transition (optional)

`uint8_t delay`  
Indicate message execution delay (C.1)

**struct esp\_ble\_mesh\_server\_rcv\_gen\_delta\_set\_t**  
Context of the received Generic Delta Set message

#### 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)

**struct esp\_ble\_mesh\_server\_rcv\_gen\_move\_set\_t**  
Context of the received Generic Move Set message

#### 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\_server\_rcv\_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\_rcv\_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\_rcv\_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_rcv_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_rcv_gen_power_range_set_t`  
Context of the received Generic Power Range Set message

**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_server_rcv_gen_loc_global_set_t`  
Context of the received Generic Location Global Set message

**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_server_rcv_gen_loc_local_set_t`  
Context of the received Generic Location Local Set message

**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

**struct esp\_ble\_mesh\_server\_rcv\_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\_rcv\_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\_rcv\_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.

**Note** This API needs to be called for each element on which the application needs to have a Generic OnOff Client Model.**Return** New Generic OnOff Client Model instance.**Parameters**

- 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\_GEN\_LEVEL\_CLI** (cli\_pub, cli\_data)

Define a new Generic Level Client Model.

**Note** This API needs to be called for each element on which the application needs to have a Generic Level Client Model.**Return** New Generic Level Client Model instance.**Parameters**

- 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\_GEN\_DEF\_TRANS\_TIME\_CLI** (cli\_pub, cli\_data)

Define a new Generic Default Transition Time Client Model.

**Note** This API needs to be called for each element on which the application needs to have a Generic Default Transition Time Client Model.**Return** New Generic Default Transition Time Client Model instance.**Parameters**

- 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\_GEN\_POWER\_ONOFF\_CLI** (cli\_pub, cli\_data)

Define a new Generic Power OnOff Client Model.

**Note** This API needs to be called for each element on which the application needs to have a Generic Power OnOff Client Model.**Return** New Generic Power OnOff Client Model instance.**Parameters**

- 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\_GEN\_POWER\_LEVEL\_CLI** (cli\_pub, cli\_data)

Define a new Generic Power Level Client Model.

**Note** This API needs to be called for each element on which the application needs to have a Generic Power Level Client Model.**Return** New Generic Power Level Client Model instance.**Parameters**

- 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\_GEN\_BATTERY\_CLI** (cli\_pub, cli\_data)

Define a new Generic Battery Client Model.

**Note** This API needs to be called for each element on which the application needs to have a Generic Battery Client Model.**Return** New Generic Battery Client Model instance.**Parameters**

- 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\_GEN\_LOCATION\_CLI** (cli\_pub, cli\_data)

Define a new Generic Location Client Model.

**Note** This API needs to be called for each element on which the application needs to have a Generic Location Client Model.

**Return** New Generic Location Client Model instance.

**Parameters**

- `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\_GEN\_PROPERTY\_CLI** (`cli_pub`, `cli_data`)

Define a new Generic Property Client Model.

**Note** This API needs to be called for each element on which the application needs to have a Generic Property Client Model.

**Return** New Generic Location Client Model instance.

**Parameters**

- `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\_GEN\_ONOFF\_SRV** (`srv_pub`, `srv_data`)

Generic Server Models related context.

Define a new Generic OnOff Server Model.

**Note** 1. The Generic OnOff Server Model is a root model.

1. This model shall support model publication and model subscription.

**Return** New Generic OnOff Server Model instance.

**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\_onoff\_srv\_t*.

**ESP\_BLE\_MESH\_MODEL\_GEN\_LEVEL\_SRV** (`srv_pub`, `srv_data`)

Define a new Generic Level Server Model.

**Note** 1. The Generic Level Server Model is a root model.

1. This model shall support model publication and model subscription.

**Return** New Generic Level Server Model instance.

**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\_level\_srv\_t*.

**ESP\_BLE\_MESH\_MODEL\_GEN\_DEF\_TRANS\_TIME\_SRV** (`srv_pub`, `srv_data`)

Define a new Generic Default Transition Time Server Model.

**Note** 1. The Generic Default Transition Time Server Model is a root model.

1. This model shall support model publication and model subscription.

**Return** New Generic Default Transition Time Server Model instance.

**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\_def\_trans\_time\_srv\_t*.

**ESP\_BLE\_MESH\_MODEL\_GEN\_POWER\_ONOFF\_SRV** (`srv_pub`, `srv_data`)

Define a new Generic Power OnOff Server Model.

**Note** 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.

1. 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.
2. This model shall support model publication and model subscription.

**Return** New Generic Power OnOff Server Model instance.

**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\_onoff\_srv\_t*.

**ESP\_BLE\_MESH\_MODEL\_GEN\_POWER\_ONOFF\_SETUP\_SRV** (srv\_pub, srv\_data)

Define a new Generic Power OnOff Setup Server Model.

**Note** 1. The Generic Power OnOff Setup Server model extends the Generic Power OnOff Server model and the Generic Default Transition Time Server model.

1. This model shall support model subscription.

**Return** New Generic Power OnOff Setup Server Model instance.

**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\_onoff\_setup\_srv\_t*.

**ESP\_BLE\_MESH\_MODEL\_GEN\_POWER\_LEVEL\_SRV** (srv\_pub, srv\_data)

Define a new Generic Power Level Server Model.

**Note** 1. The Generic Power Level 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 Generic Power Level Setup Server model shall also be present.

1. This model shall support model publication and model subscription.

**Return** New Generic Power Level Server Model instance.

**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*.

**ESP\_BLE\_MESH\_MODEL\_GEN\_POWER\_LEVEL\_SETUP\_SRV** (srv\_pub, srv\_data)

Define a new Generic Power Level Setup Server Model.

**Note** 1. The Generic Power Level Setup Server model extends the Generic Power Level Server model and the Generic Power OnOff Setup Server model.

1. This model shall support model subscription.

**Return** New Generic Power Level Setup Server Model instance.

**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*.

**ESP\_BLE\_MESH\_MODEL\_GEN\_BATTERY\_SRV** (srv\_pub, srv\_data)

Define a new Generic Battery Server Model.

**Note** 1. The Generic Battery Server Model is a root model.

1. This model shall support model publication and model subscription.

2. The model may be used to represent an element that is powered by a battery.

**Return** New Generic Battery Server Model instance.

**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*.

**ESP\_BLE\_MESH\_MODEL\_GEN\_LOCATION\_SRV** (srv\_pub, srv\_data)

Define a new Generic Location Server Model.

**Note** 1. The Generic Location Server model is a root model. When this model is present on an Element, the corresponding Generic Location Setup Server model shall also be present.

1. This model shall support model publication and model subscription.

2. The model may be used to represent an element that knows its location (global or local).

**Return** New Generic Location Server Model instance.

**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*.

**ESP\_BLE\_MESH\_MODEL\_GEN\_LOCATION\_SETUP\_SRV** (srv\_pub, srv\_data)

Define a new Generic Location Setup Server Model.

**Note** 1. The Generic Location Setup Server model extends the Generic Location Server model.

1. This model shall support model subscription.

**Return** New Generic Location Setup Server Model instance.

**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_setup_srv_t`.

**ESP\_BLE\_MESH\_MODEL\_GEN\_USER\_PROP\_SRV** (`srv_pub`, `srv_data`)

Define a new Generic User Property Server Model.

**Note** 1. The Generic User Property Server model is a root model.

1. This model shall support model publication and model subscription.

**Return** New Generic User Property Server Model instance.

**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_user_prop_srv_t`.

**ESP\_BLE\_MESH\_MODEL\_GEN\_ADMIN\_PROP\_SRV** (`srv_pub`, `srv_data`)

Define a new Generic Admin Property Server Model.

**Note** 1. The Generic Admin Property Server model extends the Generic User Property Server model.

1. This model shall support model publication and model subscription.

**Return** New Generic Admin Property Server Model instance.

**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_admin_prop_srv_t`.

**ESP\_BLE\_MESH\_MODEL\_GEN\_MANUFACTURER\_PROP\_SRV** (`srv_pub`, `srv_data`)

Define a new Generic Manufacturer Property Server Model.

**Note** 1. The Generic Manufacturer Property Server model extends the Generic User Property Server model.

1. This model shall support model publication and model subscription.

**Return** New Generic Manufacturer Property Server Model instance.

**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_manu_prop_srv_t`.

**ESP\_BLE\_MESH\_MODEL\_GEN\_CLIENT\_PROP\_SRV** (`srv_pub`, `srv_data`)

Define a new Generic User Property Server Model.

**Note** 1. The Generic Client Property Server model is a root model.

1. This model shall support model publication and model subscription.

**Return** New Generic Client Property Server Model instance.

**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`.

## Type Definitions

```
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

**Parameters**

- `event`: Event type
- `param`: Pointer to callback parameter

```
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

**Parameters**

- event: Event type
- param: Pointer to callback parameter

### Enumerations

**enum esp\_ble\_mesh\_generic\_client\_cb\_event\_t**

This enum value is the event of Generic Client Model

*Values:*

**ESP\_BLE\_MESH\_GENERIC\_CLIENT\_GET\_STATE\_EVT**

**ESP\_BLE\_MESH\_GENERIC\_CLIENT\_SET\_STATE\_EVT**

**ESP\_BLE\_MESH\_GENERIC\_CLIENT\_PUBLISH\_EVT**

**ESP\_BLE\_MESH\_GENERIC\_CLIENT\_TIMEOUT\_EVT**

**ESP\_BLE\_MESH\_GENERIC\_CLIENT\_EVT\_MAX**

**enum esp\_ble\_mesh\_gen\_user\_prop\_access\_t**

This enum value is the access value of Generic User Property

*Values:*

**ESP\_BLE\_MESH\_GEN\_USER\_ACCESS\_PROHIBIT**

**ESP\_BLE\_MESH\_GEN\_USER\_ACCESS\_READ**

**ESP\_BLE\_MESH\_GEN\_USER\_ACCESS\_WRITE**

**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:*

**ESP\_BLE\_MESH\_GEN\_ADMIN\_NOT\_USER\_PROP**

**ESP\_BLE\_MESH\_GEN\_ADMIN\_ACCESS\_READ**

**ESP\_BLE\_MESH\_GEN\_ADMIN\_ACCESS\_WRITE**

**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:*

**ESP\_BLE\_MESH\_GEN\_MANU\_NOT\_USER\_PROP**

**ESP\_BLE\_MESH\_GEN\_MANU\_ACCESS\_READ**

**enum esp\_ble\_mesh\_generic\_server\_cb\_event\_t**

This enum value is the event of Generic Server Model

*Values:*

**ESP\_BLE\_MESH\_GENERIC\_SERVER\_STATE\_CHANGE\_EVT**

1. When `get_auto_rsp` is set to `ESP_BLE_MESH_SERVER_AUTO_RSP`, no event will be callback to the application layer when Generic Get messages are received.
2. When `set_auto_rsp` is set to `ESP_BLE_MESH_SERVER_AUTO_RSP`, this event will be callback to the application layer when Generic Set/Set Unack messages are received.

**ESP\_BLE\_MESH\_GENERIC\_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 Generic Get messages are received.

**ESP\_BLE\_MESH\_GENERIC\_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 Generic Set/Set Unack messages are received.

**ESP\_BLE\_MESH\_GENERIC\_SERVER\_EVT\_MAX****Generic Client/Server Models****Header File**

- [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.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] `callback`: Pointer to the callback function.

*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.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] `callback`: Pointer to the callback function.

*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.

**Note** 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`

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] `params`: Pointer to BLE Mesh common client parameters.
- [in] `get_state`: Pointer to a union, each kind of opcode corresponds to one structure inside. Shall not be set to NULL.

*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.

**Note** 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`

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] `params`: Pointer to BLE Mesh common client parameters.
- [in] `set_state`: Pointer to a union, each kind of opcode corresponds to one structure inside. Shall not be set to NULL.

*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.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] `element`: The element to which the Health Server Model belongs.



**Unions****union esp\_ble\_mesh\_health\_client\_get\_state\_t**

*#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***esp\_ble\_mesh\_health\_fault\_get\_t* **fault\_get**

For ESP\_BLE\_MESH\_MODEL\_OP\_HEALTH\_FAULT\_GET.

**union esp\_ble\_mesh\_health\_client\_set\_state\_t**

*#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 ESP\_BLE\_MESH\_MODEL\_OP\_HEALTH\_FAULT\_TEST\_UNACK ESP\_BLE\_MESH\_MODEL\_OP\_HEALTH\_PERIOD\_SET ESP\_BLE\_MESH\_MODEL\_OP\_HEALTH\_PERIOD\_SET\_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

esp_ble_mesh_health_attention_on_cb_t attention_on
    ESP_BLE_MESH_HEALTH_SERVER_ATTENTION_ON_EVT

esp_ble_mesh_health_attention_off_cb_t attention_off
    ESP_BLE_MESH_HEALTH_SERVER_ATTENTION_OFF_EVT

```

**Structures**

```

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.

```

```

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

**uint16\_t** `company_id`  
Bluetooth assigned 16-bit Company ID

**uint8\_t** `test_id`  
ID of a specific test to be performed

**struct** `esp_ble_mesh_health_fault_clear_t`  
Parameter of Health Fault Clear

### Public Members

**uint16\_t** `company_id`  
Bluetooth assigned 16-bit Company ID

**struct** `esp_ble_mesh_health_current_status_cb_t`  
Parameters of Health Current Status

**Public Members****uint8\_t test\_id**

ID of a most recently performed test

**uint16\_t company\_id**

Bluetooth assigned 16-bit Company ID

**struct net\_buf\_simple \*fault\_array**

FaultArray field contains a sequence of 1-octet fault values

**struct esp\_ble\_mesh\_health\_fault\_status\_cb\_t**

Parameters of Health Fault Status

**Public Members****uint8\_t test\_id**

ID of a most recently performed test

**uint16\_t company\_id**

Bluetooth assigned 16-bit Company ID

**struct net\_buf\_simple \*fault\_array**

FaultArray field contains a sequence of 1-octet fault values

**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

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.

**Note** The Health Server Model can only be included by a Primary Element.

**Return** New Health Server Model instance.

**Parameters**

- `srv`: Pointer to the unique struct `esp_ble_mesh_health_srv_t`.
- `pub`: Pointer to the unique struct `esp_ble_mesh_model_pub_t`.

**ESP\_BLE\_MESH\_MODEL\_HEALTH\_CLI** (`cli_data`)

Define a new Health Client Model.

**Note** This API needs to be called for each element on which the application needs to have a Health Client Model.

**Return** New Health Client Model instance.

**Parameters**

- `cli_data`: Pointer to the unique struct `esp_ble_mesh_client_t`.

**ESP\_BLE\_MESH\_HEALTH\_PUB\_DEFINE** (`_name`, `_max`, `_role`)

A helper to define a health publication context

**Parameters**

- `_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**  
**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

#### Parameters

- event: Event type
- 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.

#### Parameters

- event: Event type

- 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:

**ESP\_BLE\_MESH\_HEALTH\_CLIENT\_GET\_STATE\_EVT**

**ESP\_BLE\_MESH\_HEALTH\_CLIENT\_SET\_STATE\_EVT**

**ESP\_BLE\_MESH\_HEALTH\_CLIENT\_PUBLISH\_EVT**

**ESP\_BLE\_MESH\_HEALTH\_CLIENT\_TIMEOUT\_EVT**

**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:

**ESP\_BLE\_MESH\_HEALTH\_SERVER\_FAULT\_UPDATE\_COMP\_EVT**

**ESP\_BLE\_MESH\_HEALTH\_SERVER\_FAULT\_CLEAR\_EVT**

**ESP\_BLE\_MESH\_HEALTH\_SERVER\_FAULT\_TEST\_EVT**

**ESP\_BLE\_MESH\_HEALTH\_SERVER\_ATTENTION\_ON\_EVT**

**ESP\_BLE\_MESH\_HEALTH\_SERVER\_ATTENTION\_OFF\_EVT**

**ESP\_BLE\_MESH\_HEALTH\_SERVER\_EVT\_MAX**

### Sensor Client/Server Models

#### Header File

- [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.

**Return** ESP\_OK on success or error code otherwise.

#### Parameters

- [in] callback: pointer to the callback function.

**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.

**Note** 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

**Return** ESP\_OK on success or error code otherwise.

#### Parameters

- [in] params: Pointer to BLE Mesh common client parameters.
- [in] get\_state: Pointer of light get message value. Shall not be set to NULL.

**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.



**Note** 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`

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] `params`: Pointer to BLE Mesh common client parameters.
- [in] `set_state`: Pointer of light set message value. Shall not be set to NULL.

`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.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] `callback`: Pointer to the callback function.

## 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

```

esp_ble_mesh_light_hsl_hue_set_t hsl_hue_set
    For ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_HUE_SET & ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_HUE_SET_UNACK

esp_ble_mesh_light_hsl_saturation_set_t hsl_saturation_set
    For ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_SATURATION_SET &
    ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_SATURATION_SET_UNACK

esp_ble_mesh_light_hsl_default_set_t hsl_default_set
    For ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_DEFAULT_SET &
    ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_DEFAULT_SET_UNACK

esp_ble_mesh_light_hsl_range_set_t hsl_range_set
    For ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_RANGE_SET &
    ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_RANGE_SET_UNACK

esp_ble_mesh_light_xyl_set_t xyl_set
    For ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_SET & ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_SET_UNACK

esp_ble_mesh_light_xyl_default_set_t xyl_default_set
    For ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_DEFAULT_SET &
    ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_DEFAULT_SET_UNACK

esp_ble_mesh_light_xyl_range_set_t xyl_range_set
    For ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_RANGE_SET &
    ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_RANGE_SET_UNACK

esp_ble_mesh_light_lc_mode_set_t lc_mode_set
    For ESP_BLE_MESH_MODEL_OP_LIGHT_LC_MODE_SET &
    ESP_BLE_MESH_MODEL_OP_LIGHT_LC_MODE_SET_UNACK

esp_ble_mesh_light_lc_om_set_t lc_om_set
    For ESP_BLE_MESH_MODEL_OP_LIGHT_LC_OM_SET & ESP_BLE_MESH_MODEL_OP_LIGHT_LC_OM_SET_UNACK

esp_ble_mesh_light_lc_light_onoff_set_t lc_light_onoff_set
    For ESP_BLE_MESH_MODEL_OP_LIGHT_LC_LIGHT_ONOFF_SET &
    ESP_BLE_MESH_MODEL_OP_LIGHT_LC_LIGHT_ONOFF_SET_UNACK

esp_ble_mesh_light_lc_property_set_t lc_property_set
    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

```

esp_ble_mesh_light_lightness_status_cb_t lightness_status
    For ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_STATUS

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

*esp\_ble\_mesh\_light\_xyl\_default\_status\_cb\_t* **xyl\_default\_status**  
For ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_XYL\_DEFAULT\_STATUS

*esp\_ble\_mesh\_light\_xyl\_range\_status\_cb\_t* **xyl\_range\_status**  
For ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_XYL\_RANGE\_STATUS

*esp\_ble\_mesh\_light\_lc\_mode\_status\_cb\_t* **lc\_mode\_status**  
For ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_LC\_MODE\_STATUS

*esp\_ble\_mesh\_light\_lc\_om\_status\_cb\_t* **lc\_om\_status**  
For ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_LC\_OM\_STATUS

*esp\_ble\_mesh\_light\_lc\_light\_onoff\_status\_cb\_t* **lc\_light\_onoff\_status**  
For ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_LC\_LIGHT\_ONOFF\_STATUS

*esp\_ble\_mesh\_light\_lc\_property\_status\_cb\_t* **lc\_property\_status**  
For ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_LC\_PROPERTY\_STATUS

**union esp\_ble\_mesh\_lighting\_server\_state\_change\_t**  
*#include <esp\_ble\_mesh\_lighting\_model\_api.h>* Lighting Server Model state change value union.

### Public Members

*esp\_ble\_mesh\_state\_change\_light\_lightness\_set\_t* **lightness\_set**  
The recv\_op in ctx can be used to decide which state is changed. Light Lightness Set

*esp\_ble\_mesh\_state\_change\_light\_lightness\_linear\_set\_t* **lightness\_linear\_set**  
Light Lightness Linear Set

*esp\_ble\_mesh\_state\_change\_light\_lightness\_default\_set\_t* **lightness\_default\_set**  
Light Lightness Default Set

*esp\_ble\_mesh\_state\_change\_light\_lightness\_range\_set\_t* **lightness\_range\_set**  
Light Lightness Range Set

*esp\_ble\_mesh\_state\_change\_light\_ctl\_set\_t* **ctl\_set**  
Light CTL Set

*esp\_ble\_mesh\_state\_change\_light\_ctl\_temperature\_set\_t* **ctl\_temp\_set**  
Light CTL Temperature Set

*esp\_ble\_mesh\_state\_change\_light\_ctl\_temperature\_range\_set\_t* **ctl\_temp\_range\_set**  
Light CTL Temperature Range Set

*esp\_ble\_mesh\_state\_change\_light\_ctl\_default\_set\_t* **ctl\_default\_set**  
Light CTL Default Set

*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

*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

*esp\_ble\_mesh\_server\_recv\_light\_lightness\_set\_t* **lightness**  
Light Lightness Set/Light Lightness Set Unack

*esp\_ble\_mesh\_server\_recv\_light\_lightness\_linear\_set\_t* **lightness\_linear**  
Light Lightness Linear Set/Light Lightness Linear Set Unack

```

esp_ble_mesh_server_rcv_light_lightness_default_set_t lightness_default
    Light Lightness Default Set/Light Lightness Default Set Unack

esp_ble_mesh_server_rcv_light_lightness_range_set_t lightness_range
    Light Lightness Range Set/Light Lightness Range Set Unack

esp_ble_mesh_server_rcv_light_ctl_set_t ctl
    Light CTL Set/Light CTL Set Unack

esp_ble_mesh_server_rcv_light_ctl_temperature_set_t ctl_temp
    Light CTL Temperature Set/Light CTL Temperature Set Unack

esp_ble_mesh_server_rcv_light_ctl_temperature_range_set_t ctl_temp_range
    Light CTL Temperature Range Set/Light CTL Temperature Range Set Unack

esp_ble_mesh_server_rcv_light_ctl_default_set_t ctl_default
    Light CTL Default Set/Light CTL Default Set Unack

esp_ble_mesh_server_rcv_light_hsl_set_t hsl
    Light HSL Set/Light HSL Set Unack

esp_ble_mesh_server_rcv_light_hsl_hue_set_t hsl_hue
    Light HSL Hue Set/Light HSL Hue Set Unack

esp_ble_mesh_server_rcv_light_hsl_saturation_set_t hsl_saturation
    Light HSL Saturation Set/Light HSL Saturation Set Unack

esp_ble_mesh_server_rcv_light_hsl_default_set_t hsl_default
    Light HSL Default Set/Light HSL Default Set Unack

esp_ble_mesh_server_rcv_light_hsl_range_set_t hsl_range
    Light HSL Range Set/Light HSL Range Set Unack

esp_ble_mesh_server_rcv_light_xyl_set_t xyl
    Light xyL Set/Light xyL Set Unack

esp_ble_mesh_server_rcv_light_xyl_default_set_t xyl_default
    Light xyL Default Set/Light xyL Default Set Unack

esp_ble_mesh_server_rcv_light_xyl_range_set_t xyl_range
    Light xyL Range Set/Light xyL Range Set Unack

esp_ble_mesh_server_rcv_light_lc_mode_set_t lc_mode
    Light LC Mode Set/Light LC Mode Set Unack

esp_ble_mesh_server_rcv_light_lc_om_set_t lc_om
    Light LC OM Set/Light LC OM Set Unack

esp_ble_mesh_server_rcv_light_lc_light_onoff_set_t lc_light_onoff
    Light LC Light OnOff Set/Light LC Light OnOff Set Unack

esp_ble_mesh_server_rcv_light_lc_property_set_t lc_property
    Light LC Property Set/Light LC Property Set Unack

```

```

union esp_ble_mesh_lighting_server_rcv_status_msg_t
    #include <esp_ble_mesh_lighting_model_api.h> Lighting Server Model received status message union.

```

### Public Members

```

esp_ble_mesh_server_rcv_sensor_status_t sensor_status
    Sensor Status

```

```

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

*esp\_ble\_mesh\_lighting\_server\_state\_change\_t* **state\_change**  
ESP\_BLE\_MESH\_LIGHTING\_SERVER\_STATE\_CHANGE\_EVT

*esp\_ble\_mesh\_lighting\_server\_rcv\_get\_msg\_t* **get**  
ESP\_BLE\_MESH\_LIGHTING\_SERVER\_RECV\_GET\_MSG\_EVT

*esp\_ble\_mesh\_lighting\_server\_rcv\_set\_msg\_t* **set**  
ESP\_BLE\_MESH\_LIGHTING\_SERVER\_RECV\_SET\_MSG\_EVT

*esp\_ble\_mesh\_lighting\_server\_rcv\_status\_msg\_t* **status**  
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 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\_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 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\_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****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\_set\_t**

Parameters of Light CTL Set

**Public Members****bool op\_en**

Indicate if optional parameters are included

**uint16\_t ctl\_lightness**

Target value of light ctl lightness state

**uint16\_t ctl\_temperatrue**

Target value of light ctl temperature state

**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\_set\_t**

Parameters of Light CTL Temperature Set

**Public Members****bool op\_en**

Indicate if optional parameters are included

**uint16\_t ctl\_temperatrue**

Target value of light ctl temperature state

**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****uint16\_t property\_id**

Property ID identifying a Light LC Property

**struct esp\_ble\_mesh\_light\_lc\_property\_set\_t**

Parameters of Light LC Property Set

**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\_lightness\_status\_cb\_t**

Bluetooth Mesh Light Lightness Client Model Get and Set callback parameters structure.

Parameters of Light Lightness Status

**Public Members**

**bool op\_en**  
Indicate if optional parameters are included

**uint16\_t present\_lightness**  
Current value of light lightness actual state

**uint16\_t target\_lightness**  
Target value of light lightness actual state (optional)

**uint8\_t remain\_time**  
Time to complete state transition (C.1)

**struct esp\_ble\_mesh\_light\_lightness\_linear\_status\_cb\_t**  
Parameters of Light Lightness Linear Status

**Public Members**

**bool op\_en**  
Indicate if optional parameters are included

**uint16\_t present\_lightness**  
Current value of light lightness linear state

**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

**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\_status\_cb\_t**  
Parameters of Light HSL Range Status

### Public Members

**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

**struct esp\_ble\_mesh\_light\_xyl\_status\_cb\_t**  
Parameters of Light xyL Status

**Public Members**

**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 remain\_time**  
Time to complete state transition (optional)

**struct esp\_ble\_mesh\_light\_xyl\_target\_status\_cb\_t**  
Parameters of Light xyL Target Status

**Public Members**

**bool op\_en**  
Indicate whether optional parameters included

**uint16\_t target\_xyl\_lightness**  
The target value of the Light xyL Lightness state

**uint16\_t target\_xyl\_x**  
The target value of the Light xyL x state

**uint16\_t target\_xyl\_y**  
The target value of the Light xyL y state

**uint8\_t remain\_time**  
Time to complete state transition (optional)

**struct esp\_ble\_mesh\_light\_xyl\_default\_status\_cb\_t**  
Parameters of Light xyL Default Status

**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\_status\_cb\_t**  
Parameters of Light xyL Range Status

**Public Members**

**uint8\_t status\_code**  
Status Code for the requesting message

**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\_status\_cb\_t**

Parameter of Light LC Mode Status

### Public Members

`uint8_t mode`

The present value of the Light LC Mode state

**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

**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)

**struct esp\_ble\_mesh\_light\_lc\_property\_status\_cb\_t**

Parameters of Light LC Property Status

### 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

**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

**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 statue 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\_setup\_srv\_t**

User data of Light CTL Setup 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.

*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

`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\_saturation**

Delta change value of saturation state transition

**struct esp\_ble\_mesh\_light\_xyl\_state\_t**

Parameters of Light xyL state

### Public Members

uint16\_t **lightness**

The present value of Light xyL Lightness state

uint16\_t **target\_lightness**

The target value of Light xyL Lightness state

uint16\_t **x**

The present value of Light xyL x state

uint16\_t **target\_x**

The target value of Light xyL x state

uint16\_t **y**

The present value of Light xyL y state

uint16\_t **target\_y**

The target value of Light xyL y state

uint16\_t **lightness\_default**

The value of Light Lightness Default state

uint16\_t **x\_default**

The value of Light xyL x Default state

uint16\_t **y\_default**

The value of Light xyL y Default state

uint8\_t **status\_code**

The status code of setting Light xyL x & y Range state

uint16\_t **x\_range\_min**

The minimum value of Light xyL x Range state

uint16\_t **x\_range\_max**

The maximum value of Light xyL x Range state

uint16\_t **y\_range\_min**

The minimum value of Light xyL y Range state

uint16\_t **y\_range\_max**

The maximum value of Light xyL y Range state

**struct esp\_ble\_mesh\_light\_xyl\_srv\_t**

User data of Light xyL Server Model

### Public Members

*esp\_ble\_mesh\_model\_t* \***model**

Pointer to the Lighting xyL 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\_xyl\_state\_t* \***state**

Parameters of the Light xyL 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\_x**

Delta change value of x state transition

int32\_t **tt\_delta\_y**

Delta change value of y state transition

**struct esp\_ble\_mesh\_light\_xyl\_setup\_srv\_t**

User data of Light xyL Setup Server Model

### Public Members

*esp\_ble\_mesh\_model\_t* \***model**

Pointer to the Lighting xyL 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\_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** : 1

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** : 1

The value of Light LC Occupancy Mode state

uint32\_t **light\_onoff** : 1

The present value of Light LC Light OnOff state

uint32\_t **target\_light\_onoff** : 1

The target value of Light LC Light OnOff state

uint32\_t **occupancy** : 1

The value of Light LC Occupancy state

uint32\_t **ambient\_luxlevel** : 24

The value of Light LC Ambient LuxLevel state

uint16\_t **linear\_output**

1. Light LC Linear Output =  $\max((\text{Lightness Out})^2/65535, \text{Regulator Output})$
2. 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
3. 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 perceptible 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 Ob1 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 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\_light\_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* **transition**  
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\_set\_t**  
Parameter of Light Lightness Actual state change event

#### Public Members

uint16\_t **lightness**  
The value of Light Lightness Actual state

**struct esp\_ble\_mesh\_state\_change\_light\_lightness\_linear\_set\_t**  
Parameter of Light Lightness Linear state change event

**Public Members****uint16\_t lightness**

The value of Light Lightness Linear state

**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 range\_min**

The minimum value of Light Lightness Range state

**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****uint16\_t lightness**

The value of Light CTL Lightness state

**uint16\_t temperature**

The value of Light CTL Temperature state

**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****uint16\_t temperature**

The value of Light CTL Temperature state

**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****uint16\_t range\_min**

The minimum value of Light CTL Temperature Range state

**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**

**uint16\_t lightness**  
The value of Light Lightness Default state

**uint16\_t x**  
The value of Light xyL x Default state

**uint16\_t y**  
The value of Light xyL y Default state

**struct esp\_ble\_mesh\_state\_change\_light\_xyl\_range\_set\_t**  
Parameters of Light xyL Range state change event

**Public Members**

**uint16\_t x\_range\_min**  
The minimum value of Light xyL x Range state

**uint16\_t x\_range\_max**  
The maximum value of Light xyL x Range state

**uint16\_t y\_range\_min**  
The minimum value of Light xyL y Range state

**uint16\_t y\_range\_max**  
The maximum value of Light xyL y Range state

**struct esp\_ble\_mesh\_state\_change\_light\_lc\_mode\_set\_t**  
Parameter of Light LC Mode state change event

**Public Members****uint8\_t mode**

The value of Light LC Mode state

**struct esp\_ble\_mesh\_state\_change\_light\_lc\_om\_set\_t**

Parameter of Light LC Occupancy Mode state change event

**Public Members****uint8\_t mode**

The value of Light LC Occupancy Mode state

**struct esp\_ble\_mesh\_state\_change\_light\_lc\_light\_onoff\_set\_t**

Parameter of Light LC Light OnOff state change event

**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\_rcv\_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\_rcv\_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\_rcv\_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\_rcv\_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\_rcv\_light\_lightness\_range\_set\_t**  
Context of the received Light Lightness Range Set message

**Public Members**

`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\_server\_rcv\_light\_ctl\_set\_t**  
Context of the received Light CTL Set message

### Public Members

**bool op\_en**  
Indicate if optional parameters are included

**uint16\_t lightness**  
Target value of light ctl lightness state

**uint16\_t temperature**  
Target value of light ctl temperature state

**int16\_t 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\_server\_rcv\_light\_ctl\_temperature\_set\_t**  
Context of the received Light CTL Temperature Set message

### Public Members

**bool op\_en**  
Indicate if optional parameters are included

**uint16\_t temperature**  
Target value of light ctl temperature state

**int16\_t 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\_server\_rcv\_light\_ctl\_temperature\_range\_set\_t**  
Context of the received Light CTL Temperature Range Set message

### 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\_server\_rcv\_light\_ctl\_default\_set\_t**  
Context of the received Light CTL Default Set message

#### 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\_server\_rcv\_light\_hsl\_set\_t**  
Context of the received Light HSL Set message

#### Public Members

**bool op\_en**  
Indicate if optional parameters are included

**uint16\_t lightness**  
Target value of light hsl lightness state

**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)

**struct esp\_ble\_mesh\_server\_rcv\_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)

**struct esp\_ble\_mesh\_server\_rcv\_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\_rcv\_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\_rcv\_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\_rcv\_light\_xyl\_set\_t**  
Context of the received Light xyL Set message

**Public Members**

bool **op\_en**  
Indicate whether optional parameters included

uint16\_t **lightness**  
The target value of the Light xyL Lightness state

uint16\_t **x**  
The target value of the Light xyL x state

**uint16\_t 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\_server\_rcv\_light\_xyl\_default\_set\_t**  
Context of the received Light xyL Default Set message

### Public Members

**uint16\_t lightness**  
The value of the Light Lightness Default state

**uint16\_t x**  
The value of the Light xyL x Default state

**uint16\_t y**  
The value of the Light xyL y Default state

**struct esp\_ble\_mesh\_server\_rcv\_light\_xyl\_range\_set\_t**  
Context of the received Light xyL Range Set message

### Public Members

**uint16\_t x\_range\_min**  
The value of the xyL x Range Min field of the Light xyL x Range state

**uint16\_t x\_range\_max**  
The value of the xyL x Range Max field of the Light xyL x Range state

**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\_rcv\_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\_rcv\_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\_rcv\_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\_rcv\_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\_rcv\_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.

**Note** This API needs to be called for each element on which the application needs to have a Light Lightness Client Model.

**Return** New Light Lightness Client Model instance.

**Parameters**

- **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\_CTL\_CLI** (cli\_pub, cli\_data)

Define a new Light CTL Client Model.

**Note** This API needs to be called for each element on which the application needs to have a Light CTL Client Model.

**Return** New Light CTL Client Model instance.

**Parameters**

- 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.

**Note** This API needs to be called for each element on which the application needs to have a Light HSL Client Model.

**Return** New Light HSL Client Model instance.

**Parameters**

- 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.

**Note** This API needs to be called for each element on which the application needs to have a Light xyL Client Model.

**Return** New Light xyL Client Model instance.

**Parameters**

- 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.

**Note** This API needs to be called for each element on which the application needs to have a Light LC Client Model.

**Return** New Light LC Client Model instance.

**Parameters**

- 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.

**Note** 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.

1. This model shall support model publication and model subscription.

**Return** New Light Lightness Server Model instance.

**Parameters**

- 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*.

**ESP\_BLE\_MESH\_MODEL\_LIGHT\_LIGHTNESS\_SETUP\_SRV** (srv\_pub, srv\_data)

Define a new Light Lightness Setup Server Model.

**Note** 1. The Light Lightness Setup Server model extends the Light Lightness Server model and the Generic Power OnOff Setup Server model.

1. This model shall support model subscription.

**Return** New Light Lightness Setup Server Model instance.

**Parameters**

- 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\_setup\_srv\_t*.

**ESP\_BLE\_MESH\_MODEL\_LIGHT\_CTL\_SRV** (srv\_pub, srv\_data)

Define a new Light CTL Server Model.

**Note** 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.

1. This model shall support model publication and model subscription.
2. 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.

**Return** New Light CTL Server Model instance.

**Parameters**

- `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_srv_t`.

**ESP\_BLE\_MESH\_MODEL\_LIGHT\_CTL\_SETUP\_SRV** (srv\_pub, srv\_data)

Define a new Light CTL Setup Server Model.

**Note** 1. The Light CTL Setup Server model extends the Light CTL Server and the Light Lightness Setup Server.

1. This model shall support model subscription.

**Return** New Light CTL Setup Server Model instance.

**Parameters**

- `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_setup_srv_t`.

**ESP\_BLE\_MESH\_MODEL\_LIGHT\_CTL\_TEMP\_SRV** (srv\_pub, srv\_data)

Define a new Light CTL Temperature Server Model.

**Note** 1. The Light CTL Temperature Server model extends the Generic Level Server model.

1. This model shall support model publication and model subscription.

**Return** New Light CTL Temperature Server Model instance.

**Parameters**

- `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`.

**ESP\_BLE\_MESH\_MODEL\_LIGHT\_HSL\_SRV** (srv\_pub, srv\_data)

Define a new Light HSL Server Model.

**Note** 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.

1. This model shall support model publication and model subscription.
2. 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.

**Return** New Light HSL Server Model instance.

**Parameters**

- `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`.

**ESP\_BLE\_MESH\_MODEL\_LIGHT\_HSL\_SETUP\_SRV** (srv\_pub, srv\_data)

Define a new Light HSL Setup Server Model.

**Note** 1. The Light HSL Setup Server model extends the Light HSL Server and the Light Lightness Setup Server.

1. This model shall support model subscription.

**Return** New Light HSL Setup Server Model instance.

**Parameters**

- `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*.

**ESP\_BLE\_MESH\_MODEL\_LIGHT\_HSL\_HUE\_SRV** (`srv_pub`, `srv_data`)

Define a new Light HSL Hue Server Model.

**Note** 1. The Light HSL Hue Server model extends the Generic Level Server model. This model is associated with the Light HSL Server model.

1. This model shall support model publication and model subscription.

**Return** New Light HSL Hue Server Model instance.

**Parameters**

- `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*.

**ESP\_BLE\_MESH\_MODEL\_LIGHT\_HSL\_SAT\_SRV** (`srv_pub`, `srv_data`)

Define a new Light HSL Saturation Server Model.

**Note** 1. The Light HSL Saturation Server model extends the Generic Level Server model. This model is associated with the Light HSL Server model.

1. This model shall support model publication and model subscription.

**Return** New Light HSL Saturation Server Model instance.

**Parameters**

- `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\_sat\_srv\_t*.

**ESP\_BLE\_MESH\_MODEL\_LIGHT\_XYL\_SRV** (`srv_pub`, `srv_data`)

Define a new Light xyL Server Model.

**Note** 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.

1. This model shall support model publication and model subscription.

**Return** New Light xyL Server Model instance.

**Parameters**

- `srv_pub`: Pointer to the unique struct *esp\_ble\_mesh\_model\_pub\_t*.
- `srv_data`: Pointer to the unique struct *esp\_ble\_mesh\_light\_xyl\_srv\_t*.

**ESP\_BLE\_MESH\_MODEL\_LIGHT\_XYL\_SETUP\_SRV** (`srv_pub`, `srv_data`)

Define a new Light xyL Setup Server Model.

**Note** 1. The Light xyL Setup Server model extends the Light xyL Server and the Light Lightness Setup Server.

1. This model shall support model subscription.

**Return** New Light xyL Setup Server Model instance.

**Parameters**

- `srv_pub`: Pointer to the unique struct *esp\_ble\_mesh\_model\_pub\_t*.
- `srv_data`: Pointer to the unique struct *esp\_ble\_mesh\_light\_xyl\_setup\_srv\_t*.

**ESP\_BLE\_MESH\_MODEL\_LIGHT\_LC\_SRV** (`srv_pub`, `srv_data`)

Define a new Light LC Server Model.

**Note** 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.

1. This model shall support model publication and model subscription.
2. 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.

**Return** New Light LC Server Model instance.

**Parameters**

- `srv_pub`: Pointer to the unique struct *esp\_ble\_mesh\_model\_pub\_t*.
- `srv_data`: Pointer to the unique struct *esp\_ble\_mesh\_light\_lc\_srv\_t*.

**ESP\_BLE\_MESH\_MODEL\_LIGHT\_LC\_SETUP\_SRV** (`srv_pub`, `srv_data`)

Define a new Light LC Setup Server Model.



- Note**
1. The Light LC (Lightness Control) Setup model extends the Light LC Server model.
    1. This model shall support model publication and model subscription.
    2. This model may be used to configure setup parameters for the Light LC Server model.

**Return** New Light LC Setup Server Model instance.

**Parameters**

- `srv_pub`: Pointer to the unique struct `esp_ble_mesh_model_pub_t`.
- `srv_data`: Pointer to the unique struct `esp_ble_mesh_light_lc_setup_srv_t`.

### Type Definitions

```
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

**Parameters**

- `event`: Event type
- `param`: Pointer to callback parameter

```
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

**Parameters**

- `event`: Event type
- `param`: Pointer to callback parameter

### Enumerations

```
enum esp_ble_mesh_light_client_cb_event_t
This enum value is the event of Lighting Client Model
```

*Values:*

```
ESP_BLE_MESH_LIGHT_CLIENT_GET_STATE_EVT
ESP_BLE_MESH_LIGHT_CLIENT_SET_STATE_EVT
ESP_BLE_MESH_LIGHT_CLIENT_PUBLISH_EVT
ESP_BLE_MESH_LIGHT_CLIENT_TIMEOUT_EVT
ESP_BLE_MESH_LIGHT_CLIENT_EVT_MAX
```

```
enum esp_ble_mesh_lc_state_t
This enum value is the Light LC State Machine states
```

*Values:*

```
ESP_BLE_MESH_LC_OFF
ESP_BLE_MESH_LC_STANDBY
ESP_BLE_MESH_LC_FADE_ON
ESP_BLE_MESH_LC_RUN
ESP_BLE_MESH_LC_FADE
ESP_BLE_MESH_LC_PROLONG
ESP_BLE_MESH_LC_FADE_STANDBY_AUTO
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:

**ESP\_BLE\_MESH\_LIGHTING\_SERVER\_STATE\_CHANGE\_EVT**

1. 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.
2. 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.

**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.

**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.

**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.

**ESP\_BLE\_MESH\_LIGHTING\_SERVER\_EVT\_MAX****Time and Scenes Client/Server Models****Header File**

- [bt/esp\\_ble\\_mesh/api/models/include/esp\\_ble\\_mesh\\_sensor\\_model\\_api.h](#)

**Functions**

*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.

**Return** `ESP_OK` on success or error code otherwise.

**Parameters**

- [in] `callback`: Pointer to the callback function.

*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.

**Note** 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`

**Return** `ESP_OK` on success or error code otherwise.

**Parameters**

- [in] `params`: Pointer to BLE Mesh common client parameters.
- [in] `get_state`: Pointer to sensor get message value. Shall not be set to NULL.

*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.

**Note** 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`

**Return** `ESP_OK` on success or error code otherwise.

**Parameters**

- [in] `params`: Pointer to BLE Mesh common client parameters.
- [in] `set_state`: Pointer to sensor set message value. Shall not be set to NULL.

*esp\_err\_t* **esp\_ble\_mesh\_register\_sensor\_server\_callback** (*esp\_ble\_mesh\_sensor\_server\_cb\_t* callback)

Register BLE Mesh Sensor Server Model callback.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] callback: Pointer to the callback function.

## Unions

**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

*esp\_ble\_mesh\_sensor\_descriptor\_get\_t* **descriptor\_get**  
For ESP\_BLE\_MESH\_MODEL\_OP\_SENSOR\_DESCRIPTOR\_GET

*esp\_ble\_mesh\_sensor\_cadence\_get\_t* **cadence\_get**  
For ESP\_BLE\_MESH\_MODEL\_OP\_SENSOR\_CADENCE\_GET

*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 rcv\_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\_rcv\_get\_msg\_t**  
*#include <esp\_ble\_mesh\_sensor\_model\_api.h>* Sensor Server Model received get message union.

### Public Members

*esp\_ble\_mesh\_server\_rcv\_sensor\_descriptor\_get\_t* **sensor\_descriptor**  
Sensor Descriptor Get

*esp\_ble\_mesh\_server\_rcv\_sensor\_cadence\_get\_t* **sensor\_cadence**  
Sensor Cadence Get

*esp\_ble\_mesh\_server\_rcv\_sensor\_settings\_get\_t* **sensor\_settings**  
Sensor Settings Get

*esp\_ble\_mesh\_server\_rcv\_sensor\_setting\_get\_t* **sensor\_setting**  
Sensor Setting Get

*esp\_ble\_mesh\_server\_rcv\_sensor\_get\_t* **sensor\_data**  
Sensor Get

*esp\_ble\_mesh\_server\_rcv\_sensor\_column\_get\_t* **sensor\_column**  
Sensor Column Get

*esp\_ble\_mesh\_server\_rcv\_sensor\_series\_get\_t* **sensor\_series**  
Sensor Series Get

**union esp\_ble\_mesh\_sensor\_server\_rcv\_set\_msg\_t**  
*#include <esp\_ble\_mesh\_sensor\_model\_api.h>* Sensor Server Model received set message union.

### Public Members

*esp\_ble\_mesh\_server\_rcv\_sensor\_cadence\_set\_t* **sensor\_cadence**  
Sensor Cadence Set

*esp\_ble\_mesh\_server\_rcv\_sensor\_setting\_set\_t* **sensor\_setting**  
Sensor Setting Set

**union esp\_ble\_mesh\_sensor\_server\_cb\_value\_t**  
*#include <esp\_ble\_mesh\_sensor\_model\_api.h>* Sensor Server Model callback value union.

### Public Members

*esp\_ble\_mesh\_sensor\_server\_state\_change\_t* **state\_change**  
ESP\_BLE\_MESH\_SENSOR\_SERVER\_STATE\_CHANGE\_EVT

*esp\_ble\_mesh\_sensor\_server\_rcv\_get\_msg\_t* **get**  
ESP\_BLE\_MESH\_SENSOR\_SERVER\_RECV\_GET\_MSG\_EVT

*esp\_ble\_mesh\_sensor\_server\_rcv\_set\_msg\_t* **set**  
ESP\_BLE\_MESH\_SENSOR\_SERVER\_RECV\_SET\_MSG\_EVT

### Structures

**struct esp\_ble\_mesh\_sensor\_descriptor\_get\_t**  
Bluetooth Mesh Sensor Client Model Get and Set parameters structure.  
Parameters of Sensor Descriptor Get

### 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\_sensor\_cadence\_get\_t**  
Parameter of Sensor Cadence Get

### Public Members

uint16\_t **property\_id**  
Property ID of a sensor

**struct esp\_ble\_mesh\_sensor\_cadence\_set\_t**  
Parameters of Sensor Cadence Set

### Public Members

uint16\_t **property\_id**  
Property ID for the sensor

uint8\_t **fast\_cadence\_period\_divisor** : 7  
Divisor for the publish period

uint8\_t **status\_trigger\_type** : 1  
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

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

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

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

`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`.

`esp_ble_mesh_client_common_param_t *params`  
The client common parameters.

`esp_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 : 12`  
The value of Sensor Positive Tolerance field



**uint32\_t negative\_tolerance** : 12  
The value of Sensor Negative Tolerance field

**uint32\_t sampling\_function** : 8  
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

**uint8\_t period\_divisor** : 7  
The value of Fast Cadence Period Divisor field

**uint8\_t trigger\_type** : 1  
The value of Status Trigger Type field

**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

**struct net\_buf\_simple \*trigger\_delta\_up**  
The value of Status Trigger Delta Up field

**uint8\_t min\_interval**  
The value of Status Min Interval field

**struct net\_buf\_simple \*fast\_cadence\_low**  
The value of Fast Cadence Low field

**struct net\_buf\_simple \*fast\_cadence\_high**  
The value of Fast Cadence High field

**struct esp\_ble\_mesh\_sensor\_data\_t**  
Parameters of Sensor Data state

### Public Members

**uint8\_t format** : 1  
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

**uint8\_t length** : 7  
The value of the Sensor Data length

**struct net\_buf\_simple \*raw\_value**  
The value of Sensor Data raw value

**struct esp\_ble\_mesh\_sensor\_series\_column\_t**  
Parameters of Sensor Series Column state

### Public Members

**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** : 7  
The value of Fast Cadence Period Divisor state

uint8\_t **trigger\_type** : 1  
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\_rcv\_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\_rcv\_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\_rcv\_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\_rcv\_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\_rcv\_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\_rcv\_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\_rcv\_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\_rcv\_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\_rcv\_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.

**Note** This API needs to be called for each element on which the application needs to have a Sensor Client Model.**Return** New Sensor Client Model instance.**Parameters**

- 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\_SENSOR\_SRV** (srv\_pub, srv\_data)

Sensor Server Models related context.

Define a new Sensor Server Model.

**Note** 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.

1. This model shall support model publication and model subscription.

**Return** New Sensor Server Model instance.**Parameters**

- 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*.

**ESP\_BLE\_MESH\_MODEL\_SENSOR\_SETUP\_SRV** (srv\_pub, srv\_data)

Define a new Sensor Setup Server Model.

**Note** 1. The Sensor Setup Server model extends the Sensor Server model.

1. This model shall support model publication and model subscription.

**Return** New Sensor Setup Server Model instance.**Parameters**

- 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*.

**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 &amp; 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\_MP\_ID\_LEN**

MPID length of Sensor Data Format A

**ESP\_BLE\_MESH\_SENSOR\_DATA\_FORMAT\_B\_MP\_ID\_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\_GET\_SENSOR\_DATA\_FORMAT** (\_data)

Get format of the sensor data.

**Note** Multiple sensor data may be concatenated. Make sure the \_data pointer is updated before getting the format of the corresponding sensor data.

**Return** Format of the sensor data.

**Parameters**

- \_data: Pointer to the start of the sensor data.

**ESP\_BLE\_MESH\_GET\_SENSOR\_DATA\_LENGTH** (\_data, \_fmt)

Get length of the sensor data.

**Note** Multiple sensor data may be concatenated. Make sure the \_data pointer is updated before getting the length of the corresponding sensor data.

**Return** Length (zero-based) of the sensor data.

**Parameters**

- \_data: Pointer to the start of the sensor data.
- \_fmt: Format of the sensor data.

**ESP\_BLE\_MESH\_GET\_SENSOR\_DATA\_PROPERTY\_ID** (\_data, \_fmt)

Get Sensor Property ID of the sensor data.

**Note** Multiple sensor data may be concatenated. Make sure the \_data pointer is updated before getting Sensor Property ID of the corresponding sensor data.

**Return** Sensor Property ID of the sensor data.

**Parameters**

- \_data: Pointer to the start of the sensor data.
- \_fmt: Format of the sensor data.

**ESP\_BLE\_MESH\_SENSOR\_DATA\_FORMAT\_A\_MPID** (*\_len*, *\_id*)

Generate a MPID value for sensor data with Format A.

**Note** 1. The Format field is 0b0 and indicates that Format A is used.

1. The Length field is a 1-based uint4 value (valid range 0x0–0xF, representing range of 1–16).
2. The Property ID is an 11-bit bit field representing 11 LSb of a Property ID.
3. This format may be used for Property Values that are not longer than 16 octets and for Property IDs less than 0x0800.

**Return** 2-octet MPID value for sensor data with Format A.

**Parameters**

- *\_len*: Length of Sensor Raw value.
- *\_id*: Sensor Property ID.

**ESP\_BLE\_MESH\_SENSOR\_DATA\_FORMAT\_B\_MPID** (*\_len*, *\_id*)

Generate a MPID value for sensor data with Format B.

**Note** 1. The Format field is 0b1 and indicates Format B is used.

1. 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.
2. The Property ID is a 16-bit bit field representing a Property ID.
3. 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.
4. Exclude the generated 1-octet value, the 2-octet Sensor Property ID

**Return** 3-octet MPID value for sensor data with Format B.

**Parameters**

- *\_len*: Length of Sensor Raw value.
- *\_id*: Sensor Property ID.

**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

**Parameters**

- *event*: Event type
- *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

**Parameters**

- *event*: Event type
- *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:*

```
ESP_BLE_MESH_SENSOR_CLIENT_GET_STATE_EVT
```

```
ESP_BLE_MESH_SENSOR_CLIENT_SET_STATE_EVT
```

```
ESP_BLE_MESH_SENSOR_CLIENT_PUBLISH_EVT
```

```
ESP_BLE_MESH_SENSOR_CLIENT_TIMEOUT_EVT
```



**ESP\_BLE\_MESH\_SENSOR\_CLIENT\_EVT\_MAX**  
**enum esp\_ble\_mesh\_sensor\_sample\_func**  
 This enum value is value of Sensor Sampling Function

*Values:*

**ESP\_BLE\_MESH\_SAMPLE\_FUNC\_UNSPECIFIED**  
**ESP\_BLE\_MESH\_SAMPLE\_FUNC\_INSTANTANEOUS**  
**ESP\_BLE\_MESH\_SAMPLE\_FUNC\_ARITHMETIC\_MEAN**  
**ESP\_BLE\_MESH\_SAMPLE\_FUNC\_RMS**  
**ESP\_BLE\_MESH\_SAMPLE\_FUNC\_MAXIMUM**  
**ESP\_BLE\_MESH\_SAMPLE\_FUNC\_MINIMUM**  
**ESP\_BLE\_MESH\_SAMPLE\_FUNC\_ACCUMULATED**  
**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:*

**ESP\_BLE\_MESH\_SENSOR\_SERVER\_STATE\_CHANGE\_EVT**

1. 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.
2. 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.

**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.

**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.

**ESP\_BLE\_MESH\_SENSOR\_SERVER\_EVT\_MAX**

## Lighting Client/Server Models

### Header File

- `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.

**Return** `ESP_OK` on success or error code otherwise.

#### Parameters

- `[in] callback`: Pointer to the callback function.

`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.

**Note** If you want to know the opcodes and corresponding meanings accepted by this API, please refer to `esp_ble_mesh_time_scene_message_opcode_t` in `esp_ble_mesh_defs.h`

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] `params`: Pointer to BLE Mesh common client parameters.
- [in] `get_state`: Pointer to time scene get message value. Shall not be set to NULL.

```
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.

**Note** If you want to know the opcodes and corresponding meanings accepted by this API, please refer to `esp_ble_mesh_time_scene_message_opcode_t` in `esp_ble_mesh_defs.h`

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] `params`: Pointer to BLE Mesh common client parameters.
- [in] `set_state`: Pointer to time scene set message value. Shall not be set to NULL.

```
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.

**Return** ESP\_OK on success or error code otherwise.

**Parameters**

- [in] `callback`: Pointer to the callback function.

## Unions

```
union 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

```
esp_ble_mesh_scheduler_act_get_t scheduler_act_get
For ESP_BLE_MESH_MODEL_OP_SCHEDULER_ACT_GET
union 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

esp_ble_mesh_scheduler_status_cb_t scheduler_status
    For ESP_BLE_MESH_MODEL_OP_SCHEDULER_STATUS

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

```
esp_ble_mesh_state_change_time_set_t time_set
    The rcv_op in ctx can be used to decide which state is changed. Time Set

esp_ble_mesh_state_change_time_status_t time_status
    Time Status

esp_ble_mesh_state_change_time_zone_set_t time_zone_set
    Time Zone Set

esp_ble_mesh_state_change_tai_utc_delta_set_t tai_utc_delta_set
    TAI UTC Delta Set

esp_ble_mesh_state_change_time_role_set_t time_role_set
    Time Role Set

esp_ble_mesh_state_change_scene_store_t scene_store
    Scene Store

esp_ble_mesh_state_change_scene_recall_t scene_recall
    Scene Recall

esp_ble_mesh_state_change_scene_delete_t scene_delete
    Scene Delete

esp_ble_mesh_state_change_scheduler_act_set_t scheduler_act_set
    Scheduler Action Set
```

**union esp\_ble\_mesh\_time\_scene\_server\_rcv\_get\_msg\_t**  
*#include <esp\_ble\_mesh\_time\_scene\_model\_api.h>* Time Scene Server Model received get message union.

### Public Members

*esp\_ble\_mesh\_server\_rcv\_scheduler\_act\_get\_t* **scheduler\_act**  
 Scheduler Action Get

**union esp\_ble\_mesh\_time\_scene\_server\_rcv\_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\_rcv\_time\_set\_t* **time**  
 Time Set

*esp\_ble\_mesh\_server\_rcv\_time\_zone\_set\_t* **time\_zone**  
 Time Zone Set

*esp\_ble\_mesh\_server\_rcv\_tai\_utc\_delta\_set\_t* **tai\_utc\_delta**  
 TAI-UTC Delta Set

*esp\_ble\_mesh\_server\_rcv\_time\_role\_set\_t* **time\_role**  
 Time Role Set

*esp\_ble\_mesh\_server\_rcv\_scene\_store\_t* **scene\_store**  
 Scene Store/Scene Store Unack

*esp\_ble\_mesh\_server\_rcv\_scene\_recall\_t* **scene\_recall**  
 Scene Recall/Scene Recall Unack

*esp\_ble\_mesh\_server\_rcv\_scene\_delete\_t* **scene\_delete**  
 Scene Delete/Scene Delete Unack

*esp\_ble\_mesh\_server\_rcv\_scheduler\_act\_set\_t* **scheduler\_act**  
 Scheduler Action Set/Scheduler Action Set Unack

**union esp\_ble\_mesh\_time\_scene\_server\_rcv\_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\_rcv\_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\_rcv\_get\_msg\_t* **get**  
 ESP\_BLE\_MESH\_TIME\_SCENE\_SERVER\_RECV\_GET\_MSG\_EVT

*esp\_ble\_mesh\_time\_scene\_server\_rcv\_set\_msg\_t* **set**  
 ESP\_BLE\_MESH\_TIME\_SCENE\_SERVER\_RECV\_SET\_MSG\_EVT

*esp\_ble\_mesh\_time\_scene\_server\_rcv\_status\_msg\_t* **status**  
 ESP\_BLE\_MESH\_TIME\_SCENE\_SERVER\_RECV\_STATUS\_MSG\_EVT

## Structures

### **struct esp\_ble\_mesh\_time\_set\_t**

Bluetooth Mesh Time Scene Client Model Get and Set parameters structure.

Parameters of Time Set

#### **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 : 1**

0 = No Time Authority, 1 = Time Authority

**uint16\_t tai\_utc\_delta : 15**

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\_set\_t**

Parameters of Time Zone Set

#### **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\_tai\_utc\_delta\_set\_t**

Parameters of TAI-UTC Delta Set

#### **Public Members**

**uint16\_t tai\_utc\_delta\_new : 15**

Upcoming difference between TAI and UTC in seconds

**uint16\_t padding : 1**

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\_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

**Public Members**

`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**

`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**

`uint64_t index : 4`  
Index of the Schedule Register entry to set

`uint64_t year : 7`  
Scheduled year for the action

`uint64_t month : 12`  
Scheduled month for the action

`uint64_t day : 5`  
Scheduled day of the month for the action

`uint64_t hour : 5`  
Scheduled hour for the action

`uint64_t minute` : 6  
Scheduled minute for the action

`uint64_t second` : 6  
Scheduled second for the action

`uint64_t day_of_week` : 7  
Schedule days of the week for the action

`uint64_t action` : 4  
Action to be performed at the scheduled time

`uint64_t trans_time` : 8  
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` : 1  
0 = No Time Authority, 1 = Time Authority

`uint16_t tai_utc_delta` : 15  
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 Seconds 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` : 15  
Current difference between TAI and UTC in seconds

`uint16_t padding_1 : 1`  
Always 0b0. Other values are Prohibited.

`uint16_t tai_utc_delta_new : 15`  
Upcoming difference between TAI and UTC in seconds

`uint16_t padding_2 : 1`  
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

`uint16_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** : 4  
Enumerates (selects) a Schedule Register entry

uint64\_t **year** : 7  
Scheduled year for the action

uint64\_t **month** : 12  
Scheduled month for the action

uint64\_t **day** : 5  
Scheduled day of the month for the action

uint64\_t **hour** : 5  
Scheduled hour for the action

uint64\_t **minute** : 6  
Scheduled minute for the action

uint64\_t **second** : 6  
Scheduled second for the action

uint64\_t **day\_of\_week** : 7  
Schedule days of the week for the action

uint64\_t **action** : 4  
Action to be performed at the scheduled time

uint64\_t **trans\_time** : 8  
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 time\_zone\_offset\_curr**  
The value of the Time Zone Offset Current field

**uint8\_t time\_zone\_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 : 1**  
The value of the Time Authority bit

**uint16\_t tai\_utc\_delta\_curr : 15**  
The value of the TAI-UTC Delta Current state

**uint16\_t tai\_utc\_delta\_new : 15**  
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**

1. Scene Store is an operation of storing values of a present state of an element.
2. The structure and meaning of the stored state is determined by a model. States to be stored are specified by each model.
3. 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.

4. 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

`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

`uint8_t status_code`

The status code of the last scene operation

`bool in_progress`

Indicate if the scene transition is in progress

`struct esp_ble_mesh_scene_srv_t`

User data of Scene Server Model

### Public Members

`esp_ble_mesh_model_t *model`

Pointer to the Scene 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

`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_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 : 7`

The value of Scheduled year for the action

`uint64_t month : 12`

The value of Scheduled month for the action

`uint64_t day : 5`

The value of Scheduled day of the month for the action

`uint64_t hour : 5`

The value of Scheduled hour for the action

`uint64_t minute` : 6  
The value of Scheduled minute for the action

`uint64_t second` : 6  
The value of Scheduled second for the action

`uint64_t day_of_week` : 7  
The value of Schedule days of the week for the action

`uint64_t action` : 4  
The value of Action to be performed at the scheduled time

`uint64_t trans_time` : 8  
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

`const uint8_t schedule_count`  
Scheduler count

`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

`esp_ble_mesh_model_t *model`  
Pointer to the Scheduler Server Model. Initialized internally.

`esp_ble_mesh_server_rsp_ctrl_t rsp_ctrl`  
Response control of the server model received messages

`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

`esp_ble_mesh_model_t *model`  
Pointer to the Scheduler 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_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**

`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 : 1`  
0 = No Time Authority, 1 = Time Authority

`uint16_t tai_utc_delta_curr : 15`  
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 : 1`  
0 = No Time Authority, 1 = Time Authority

`uint16_t tai_utc_delta_curr : 15`  
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 tai_zone_change[5]`  
TAI Seconds time of the upcoming Time Zone Offset change

**struct esp\_ble\_mesh\_state\_change\_tai\_utc\_delta\_set\_t**  
Parameters of TAI UTC Delta Set state change event

**Public Members**

`uint16_t tai_utc_delta_new : 15`  
Upcoming difference between TAI and UTC in seconds

`uint8_t tai_delta_change[5]`  
TAI Seconds time of the upcoming TAI-UTC Delta change

**struct esp\_ble\_mesh\_state\_change\_time\_role\_set\_t**  
Parameter of Time Role Set state change event

### Public Members

`uint8_t time_role`  
The Time Role for the element

**struct esp\_ble\_mesh\_state\_change\_scene\_store\_t**  
Parameter of Scene Store state change event

### Public Members

`uint16_t scene_number`  
The number of scenes to be stored

**struct esp\_ble\_mesh\_state\_change\_scene\_recall\_t**  
Parameter of Scene Recall state change event

### Public Members

`uint16_t scene_number`  
The number of scenes to be recalled

**struct esp\_ble\_mesh\_state\_change\_scene\_delete\_t**  
Parameter of Scene Delete state change event

### Public Members

`uint16_t 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 : 4`  
Index of the Schedule Register entry to set

`uint64_t year : 7`  
Scheduled year for the action

`uint64_t month : 12`  
Scheduled month for the action

`uint64_t day : 5`  
Scheduled day of the month for the action

`uint64_t hour : 5`  
Scheduled hour for the action

`uint64_t minute : 6`  
Scheduled minute for the action

`uint64_t second : 6`  
Scheduled second for the action

`uint64_t day_of_week` : 7  
Schedule days of the week for the action

`uint64_t action` : 4  
Action to be performed at the scheduled time

`uint64_t trans_time` : 8  
Transition time for this action

`uint16_t scene_number`  
Scene number to be used for some actions

**struct esp\_ble\_mesh\_server\_rcv\_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\_rcv\_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` : 1  
0 = No Time Authority, 1 = Time Authority

`uint16_t tai_utc_delta` : 15  
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\_rcv\_time\_zone\_set\_t**  
Context of the received Time Zone 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\_rcv\_tai\_utc\_delta\_set\_t**  
Context of the received TAI UTC Delta Set message

### Public Members

`uint16_t tai_utc_delta_new` : 15  
Upcoming difference between TAI and UTC in seconds



`uint16_t padding : 1`  
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\_rcv\_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\_rcv\_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\_rcv\_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\_rcv\_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\_rcv\_scheduler\_act\_set\_t**  
Context of the received Scheduler Action Set message

### Public Members

`uint64_t index : 4`  
Index of the Schedule Register entry to set

`uint64_t year : 7`  
Scheduled year for the action

`uint64_t month` : 12  
Scheduled month for the action

`uint64_t day` : 5  
Scheduled day of the month for the action

`uint64_t hour` : 5  
Scheduled hour for the action

`uint64_t minute` : 6  
Scheduled minute for the action

`uint64_t second` : 6  
Scheduled second for the action

`uint64_t day_of_week` : 7  
Schedule days of the week for the action

`uint64_t action` : 4  
Action to be performed at the scheduled time

`uint64_t trans_time` : 8  
Transition time for this action

`uint16_t scene_number`  
Scene number to be used for some actions

**struct esp\_ble\_mesh\_server\_rcv\_time\_status\_t**  
Context of the received Time Status 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` : 1  
0 = No Time Authority, 1 = Time Authority

`uint16_t tai_utc_delta` : 15  
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\_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.

**Note** This API needs to be called for each element on which the application needs to have a Time Client Model.**Return** New Time Client Model instance.**Parameters**

- 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\_SCENE\_CLI** (cli\_pub, cli\_data)

Define a new Scene Client Model.

**Note** This API needs to be called for each element on which the application needs to have a Scene Client Model.**Return** New Scene Client Model instance.**Parameters**

- 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\_SCHEDULER\_CLI** (cli\_pub, cli\_data)

Define a new Scheduler Client Model.

**Note** This API needs to be called for each element on which the application needs to have a Scheduler Client Model.**Return** New Scheduler Client Model instance.**Parameters**

- 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\_TIME\_SRV** (srv\_pub, srv\_data)

Time Scene Server Models related context.

Define a new Time Server Model.

**Note** 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.

1. This model shall support model publication and model subscription.

**Return** New Time Server Model instance.**Parameters**

- 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*.

**ESP\_BLE\_MESH\_MODEL\_TIME\_SETUP\_SRV** (srv\_data)

Define a new Time Setup Server Model.

**Note** 1. The Time Setup Server model extends the Time Server model. Time is sensitive information that is propagated across a mesh network.

1. 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.
2. This model does not support subscribing nor publishing.

**Return** New Time Setup Server Model instance.**Parameters**

- srv\_data: Pointer to the unique struct *esp\_ble\_mesh\_time\_setup\_srv\_t*.

**ESP\_BLE\_MESH\_MODEL\_SCENE\_SRV** (srv\_pub, srv\_data)

Define a new Scene Server Model.

**Note** 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.

1. This model shall support model publication and model subscription.
2. The model may be present only on the Primary element of a node.

**Return** New Scene Server Model instance.

**Parameters**

- `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`.

**ESP\_BLE\_MESH\_MODEL\_SCENE\_SETUP\_SRV** (`srv_pub`, `srv_data`)

Define a new Scene Setup Server Model.

**Note** 1. The Scene Setup Server model extends the Scene Server model and the Generic Default Transition Time Server model.

1. This model shall support model subscription.
2. The model may be present only on the Primary element of a node.

**Return** New Scene Setup Server Model instance.

**Parameters**

- `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`.

**ESP\_BLE\_MESH\_MODEL\_SCHEDULER\_SRV** (`srv_pub`, `srv_data`)

Define a new Scheduler Server Model.

**Note** 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.

1. This model shall support model publication and model subscription.
2. The model may be present only on the Primary element of a node.
3. The model requires the Time Server model shall be present on the element.

**Return** New Scheduler Server Model instance.

**Parameters**

- `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`.

**ESP\_BLE\_MESH\_MODEL\_SCHEDULER\_SETUP\_SRV** (`srv_pub`, `srv_data`)

Define a new Scheduler Setup Server Model.

**Note** 1. The Scheduler Setup Server model extends the Scheduler Server and the Scene Setup Server models.

1. This model shall support model subscription.
2. The model may be present only on the Primary element of a node.

**Return** New Scheduler Setup Server Model instance.

**Parameters**

- `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`.

**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\_CLINET**  
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

```
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

#### Parameters

- event: Event type
- param: Pointer to callback parameter

```
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

#### Parameters

- event: Event type
- param: Pointer to callback parameter

### Enumerations

```
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**

```
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**

1. 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.
2. 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.

**ESP\_BLE\_MESH\_TIME\_SCENE\_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 Time Status message is received.

**ESP\_BLE\_MESH\_TIME\_SCENE\_SERVER\_EVT\_MAX**

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 [bluetooth/bluedroid](#) 示例目录下，请查看。

下面的示例给出了详细介绍：

- [GATT 客户端示例](#)
- [GATT 服务端服务表格示例](#)
- [GATT 服务端示例](#)
- [GATT 客户端安全性示例](#)
- [GATT 服务端安全性示例](#)
- [GATT 客户端多连接示例](#)

## 2.2 连网 API

### 2.2.1 Wi-Fi

#### 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：

- 基站模式（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。
- AP-STA 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 示例项目的 `wifi` 目录下包含以下应用程序：

- Wi-Fi 示例代码；
- 另外一个简单的应用程序 `esp-idf-template`，演示了如何将 ESP32 模组连接到 AP。

#### API 参考

##### Header File

- [esp\\_wifi/include/esp\\_wifi.h](#)

##### Functions

`esp_err_t esp_wifi_init (const wifi_init_config_t *config)`

Init WiFi Alloc resource for WiFi driver, such as WiFi control structure, RX/TX buffer, WiFi NVS structure etc, this WiFi also start WiFi task.

**Attention** 1. This API must be called before all other WiFi API can be called

**Attention 2.** Always use `WIFI_INIT_CONFIG_DEFAULT` macro to init the config to default values, this can guarantee all the fields got 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`!

**Return**

- `ESP_OK`: succeed
- `ESP_ERR_NO_MEM`: out of memory
- others: refer to error code `esp_err.h`

**Parameters**

- `config`: pointer to WiFi init configuration structure; can point to a temporary variable.

`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

**Return**

- `ESP_OK`: succeed
- `ESP_ERR_WIFI_NOT_INIT`: WiFi is not initialized by `esp_wifi_init`

`esp_err_t esp_wifi_set_mode` (`wifi_mode_t mode`)

Set the WiFi operating mode.

Set the WiFi operating mode as station, soft-AP or station+soft-AP, The default mode is soft-AP mode.

**Return**

- `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`

**Parameters**

- `mode`: WiFi operating mode

`esp_err_t esp_wifi_get_mode` (`wifi_mode_t *mode`)

Get current operating mode of WiFi.

**Return**

- `ESP_OK`: succeed
- `ESP_ERR_WIFI_NOT_INIT`: WiFi is not initialized by `esp_wifi_init`
- `ESP_ERR_INVALID_ARG`: invalid argument

**Parameters**

- [out] `mode`: store current WiFi mode

`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.

**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_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.

**Return**

- `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

**Return**

- 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\_start\_scan() 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 re-connection 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.

**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\_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.

**Return**

- 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.

**Return**

- ESP\_OK: succeed
- others: fail

**esp\_err\_t esp\_wifi\_deauth\_sta** (uint16\_t aid)

deauthenticate all stations or associated id equals to aid

**Return**

- 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

**Parameters**

- aid: when aid is 0, deauthenticate all stations, otherwise deauthenticate station whose associated id is aid

**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.

**Return**

- 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`

**Parameters**

- `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

`esp_err_t esp_wifi_scan_stop` (void)

Stop the scan in process.

**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_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.

**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

**Parameters**

- [out] `number`: store number of APIs found in last scan

`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.

**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

**Parameters**

- [inout] `number`: 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

`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

**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

**Parameters**

- `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

`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`.

**Return** `ESP_OK`: succeed

**Parameters**

- `type`: power save type

`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`.

**Return** `ESP_OK`: succeed

**Parameters**

- `[out] type`: store current power save type

`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_11B|WIFI_PROTOCOL_11G|WIFI_PROTOCOL_11N`)

**Attention** Currently we only support 802.11b or 802.11g or 802.11n mode

**Return**

- `ESP_OK`: succeed
- `ESP_ERR_WIFI_NOT_INIT`: WiFi is not initialized by `esp_wifi_init`
- `ESP_ERR_WIFI_IF`: invalid interface
- others: refer to error codes in `esp_err.h`

**Parameters**

- `ifx`: interfaces
- `protocol_bitmap`: WiFi protocol bitmap

`esp_err_t esp_wifi_get_protocol (wifi_interface_t ifx, uint8_t *protocol_bitmap)`

Get the current protocol bitmap of the specified interface.

**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
- others: refer to error codes in `esp_err.h`

**Parameters**

- `ifx`: interface
- `[out] protocol_bitmap`: store current WiFi protocol bitmap of interface `ifx`

`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

**Attention** 2. `WIFI_BW_HT40` is supported only when the interface support 11N

**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
- others: refer to error codes in `esp_err.h`

**Parameters**

- `ifx`: interface to be configured
- `bw`: bandwidth

`esp_err_t esp_wifi_get_bandwidth (wifi_interface_t ifx, wifi_bandwidth_t *bw)`

Get the bandwidth of ESP32 specified interface.

**Attention** 1. API return false if try to get a interface that is not enable

**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

**Parameters**

- ifx: interface to be configured
- [out] bw: store bandwidth of interface ifx

*esp\_err\_t* **esp\_wifi\_set\_channel** (uint8\_t primary, wifi\_second\_chan\_t second)

Set primary/secondary channel of ESP32.

**Attention** 1. This API should be called after esp\_wifi\_start()

**Attention** 2. When ESP32 is in STA mode, this API should not be called when STA is scanning or connecting to an external AP

**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

**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

**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

*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

**Return**

- ESP\_OK: succeed
- ESP\_ERR\_WIFI\_NOT\_INIT: WiFi is not initialized by esp\_wifi\_init
- ESP\_ERR\_INVALID\_ARG: invalid argument

**Parameters**

- primary: store current primary channel
- [out] second: store current second channel

*esp\_err\_t* **esp\_wifi\_set\_country** (const wifi\_country\_t \*country)

configure country info

**Attention** 1. The default country is {.cc=" CN" , .schan=1, .nchan=13, policy=WIFI\_COUNTRY\_POLICY\_AUTO}

**Attention** 2. 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=" USA" , .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 back to the country info of the station automatically, {.cc=" US" , .schan=1, .nchan=11} in the example.

**Attention** 3. When the country policy is WIFI\_COUNTRY\_POLICY\_MANUAL, always use the configured country info.

**Attention** 4. 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 changed also.

**Attention** 5. The country configuration is not stored into flash

**Attention** 6. This API doesn't validate the per-country rules, it's up to the user to fill in all fields according to local regulations.

**Return**

- ESP\_OK: succeed

- ESP\_ERR\_WIFI\_NOT\_INIT: WiFi is not initialized by esp\_wifi\_init
- ESP\_ERR\_INVALID\_ARG: invalid argument

**Parameters**

- `country`: the configured country info

*esp\_err\_t* **esp\_wifi\_get\_country** (*wifi\_country\_t* \*country)

get the current country info

**Return**

- ESP\_OK: succeed
- ESP\_ERR\_WIFI\_NOT\_INIT: WiFi is not initialized by esp\_wifi\_init
- ESP\_ERR\_INVALID\_ARG: invalid argument

**Parameters**

- `country`: country info

*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” .

**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\_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

**Parameters**

- `ifx`: interface
- `mac`: the MAC address

*esp\_err\_t* **esp\_wifi\_get\_mac** (*wifi\_interface\_t* ifx, uint8\_t mac[6])

Get mac of specified interface.

**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\_IF: invalid interface

**Parameters**

- `ifx`: interface
- [out] `mac`: store mac of the interface ifx

*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.

**Return**

- ESP\_OK: succeed
- ESP\_ERR\_WIFI\_NOT\_INIT: WiFi is not initialized by esp\_wifi\_init

**Parameters**

- `cb`: callback

*esp\_err\_t* **esp\_wifi\_set\_promiscuous** (bool en)

Enable the promiscuous mode.

**Return**

- ESP\_OK: succeed
- ESP\_ERR\_WIFI\_NOT\_INIT: WiFi is not initialized by esp\_wifi\_init

**Parameters**

- en: false - disable, true - enable

*esp\_err\_t* **esp\_wifi\_get\_promiscuous** (bool \*en)

Get the promiscuous mode.

**Return**

- ESP\_OK: succeed
- ESP\_ERR\_WIFI\_NOT\_INIT: WiFi is not initialized by esp\_wifi\_init
- ESP\_ERR\_INVALID\_ARG: invalid argument

**Parameters**

- [out] en: store the current status of promiscuous mode

*esp\_err\_t* **esp\_wifi\_set\_promiscuous\_filter** (const *wifi\_promiscuous\_filter\_t* \*filter)

Enable the promiscuous mode packet type filter.

**Note** The default filter is to filter all packets except WIFI\_PKT\_MISC

**Return**

- ESP\_OK: succeed
- ESP\_ERR\_WIFI\_NOT\_INIT: WiFi is not initialized by esp\_wifi\_init

**Parameters**

- filter: the packet type filtered in promiscuous mode.

*esp\_err\_t* **esp\_wifi\_get\_promiscuous\_filter** (*wifi\_promiscuous\_filter\_t* \*filter)

Get the promiscuous filter.

**Return**

- ESP\_OK: succeed
- ESP\_ERR\_WIFI\_NOT\_INIT: WiFi is not initialized by esp\_wifi\_init
- ESP\_ERR\_INVALID\_ARG: invalid argument

**Parameters**

- [out] filter: store the current status of promiscuous filter

*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.

**Note** The default filter is to filter none control packet.

**Return**

- ESP\_OK: succeed
- ESP\_ERR\_WIFI\_NOT\_INIT: WiFi is not initialized by esp\_wifi\_init

**Parameters**

- filter: the subtype of the control packet filtered in promiscuous mode.

*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.

**Return**

- ESP\_OK: succeed
- ESP\_ERR\_WIFI\_NOT\_INIT: WiFi is not initialized by esp\_wifi\_init
- ESP\_ERR\_WIFI\_ARG: invalid argument

**Parameters**

- [out] filter: store the current status of subtype filter of the control packet in promiscuous mode

*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.

**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\_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

**Parameters**

- interface: interface
- conf: station or soft-AP configuration

*esp\_err\_t* **esp\_wifi\_get\_config** (*wifi\_interface\_t* interface, *wifi\_config\_t* \*conf)

Get configuration of specified interface.

**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\_IF: invalid interface

**Parameters**

- interface: interface
- [out] conf: station or soft-AP configuration

*esp\_err\_t* **esp\_wifi\_ap\_get\_sta\_list** (*wifi\_sta\_list\_t* \*sta)

Get STAs associated with soft-AP.

**Attention** SSC only 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

**Parameters**

- [out] sta: station list ap can get the connected sta' s phy mode info through the struct member phy\_11b, phy\_11g, phy\_11n, phy\_1r in the *wifi\_sta\_info\_t* struct. For example, phy\_11b = 1 imply that sta support 802.11b mode

*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.

**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\_MODE: WiFi mode is wrong
- ESP\_ERR\_WIFI\_CONN: WiFi internal error, the station/soft-AP control block is invalid

**Parameters**

- mac: STA' s mac address
- [out] aid: Store the AID corresponding to STA mac

*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

**Return**

- ESP\_OK: succeed
- ESP\_ERR\_WIFI\_NOT\_INIT: WiFi is not initialized by esp\_wifi\_init
- ESP\_ERR\_INVALID\_ARG: invalid argument

**Parameters**

- storage: : storage type

*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.

#### 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

#### Parameters

- *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 *wifi\_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.

*esp\_err\_t* **esp\_wifi\_set\_vendor\_ie\_cb** (*esp\_vendor\_ie\_cb\_t* *cb*, void \**ctx*)

Register Vendor-Specific Information Element monitoring callback.

#### Return

- ESP\_OK: succeed
- ESP\_ERR\_WIFI\_NOT\_INIT: WiFi is not initialized by esp\_wifi\_init

#### Parameters

- *cb*: Callback function
- *ctx*: Context argument, passed to callback function.

*esp\_err\_t* **esp\_wifi\_set\_max\_tx\_power** (int8\_t *power*)

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. Param 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],34}, {[44, 51],44}, {[52, 55],52}, {[56, 59],56}, {[60, 65],60}, {[66, 71],66}, {[72, 79],72}, {[80, 84],80}}.

#### Return

- ESP\_OK: succeed
- ESP\_ERR\_WIFI\_NOT\_INIT: WiFi is not initialized by esp\_wifi\_init
- ESP\_ERR\_WIFI\_NOT\_START: WiFi is not started by esp\_wifi\_start
- ESP\_ERR\_WIFI\_ARG: invalid argument, e.g. parameter is out of range

#### Parameters

- *power*: Maximum WiFi transmitting power.

*esp\_err\_t* **esp\_wifi\_get\_max\_tx\_power** (int8\_t \**power*)

Get maximum transmitting power after WiFi start.

#### Return

- ESP\_OK: succeed
- ESP\_ERR\_WIFI\_NOT\_INIT: WiFi is not initialized by esp\_wifi\_init
- ESP\_ERR\_WIFI\_NOT\_START: WiFi is not started by esp\_wifi\_start
- ESP\_ERR\_WIFI\_ARG: invalid argument

#### Parameters

- *power*: Maximum WiFi transmitting power, unit is 0.25dBm.

*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.

**Return**

- `ESP_OK`: succeed
- `ESP_ERR_WIFI_NOT_INIT`: WiFi is not initialized by `esp_wifi_init`

**Parameters**

- `mask`: WiFi event mask.

`esp_err_t esp_wifi_get_event_mask (uint32_t *mask)`

Get mask of WiFi events.

**Return**

- `ESP_OK`: succeed
- `ESP_ERR_WIFI_NOT_INIT`: WiFi is not initialized by `esp_wifi_init`
- `ESP_ERR_WIFI_ARG`: invalid argument

**Parameters**

- `mask`: WiFi event mask.

`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

**Return**

- `ESP_OK`: success
- `ESP_ERR_WIFI_IF`: Invalid interface
- `ESP_ERR_INVALID_ARG`: Invalid parameter
- `ESP_ERR_WIFI_NO_MEM`: out of memory

**Parameters**

- `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.

`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.

**Return**

- `ESP_OK`: succeed
- `ESP_ERR_WIFI_NOT_INIT`: WiFi is not initialized by `esp_wifi_init`

**Parameters**

- `cb`: callback
- `ctx`: context argument, passed to callback function

`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\_START: WiFi is not started by esp\_wifi\_start or promiscuous mode is not enabled
- ESP\_ERR\_INVALID\_ARG: invalid argument

**Parameters**

- 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\_START: WiFi is not started by esp\_wifi\_start or promiscuous mode is not enabled
- ESP\_ERR\_INVALID\_ARG: invalid argument

**Parameters**

- en: true - enable, false - disable

*esp\_err\_t* esp\_wifi\_set\_ant\_gpio (const *wifi\_ant\_gpio\_config\_t* \*config)

Set antenna GPIO configuration.

**Return**

- 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

**Parameters**

- config: Antenna GPIO configuration.

*esp\_err\_t* esp\_wifi\_get\_ant\_gpio (*wifi\_ant\_gpio\_config\_t* \*config)

Get current antenna GPIO configuration.

**Return**

- 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

**Parameters**

- config: Antenna GPIO configuration.

*esp\_err\_t* esp\_wifi\_set\_ant (const *wifi\_ant\_config\_t* \*config)

Set antenna configuration.

**Return**

- 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

**Parameters**

- config: Antenna configuration.

*esp\_err\_t* esp\_wifi\_get\_ant (*wifi\_ant\_config\_t* \*config)

Get current antenna configuration.

**Return**

- 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

**Parameters**

- config: Antenna configuration.

*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 death the STA. Default is 300s.

**Attention** 3. The inactive time configuration is not stored into flash

#### 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\_ARG: invalid argument, For Station, if sec is less than 3. For SoftAP, if sec is less than 10.

#### Parameters

- *ifx*: interface to be configured.
- *sec*: Inactive time. Unit seconds.

*esp\_err\_t* **esp\_wifi\_get\_inactive\_time** (*wifi\_interface\_t* *ifx*, uint16\_t \**sec*)

Get inactive time of specified interface.

#### Return

- ESP\_OK: succeed
- ESP\_ERR\_WIFI\_NOT\_INIT: WiFi is not initialized by esp\_wifi\_init
- ESP\_ERR\_WIFI\_ARG: invalid argument

#### Parameters

- *ifx*: Interface to be configured.
- *sec*: Inactive time. Unit seconds.

*esp\_err\_t* **esp\_wifi\_statdump** (uint32\_t *modules*)

Dump WiFi statistics.

#### Return

- ESP\_OK: succeed
- others: failed

#### Parameters

- *modules*: statistic modules to be dumped

## Structures

**struct wifi\_init\_config\_t**

WiFi stack configuration parameters passed to esp\_wifi\_init call.

### Public Members

*system\_event\_handler\_t* **event\_handler**

WiFi event handler

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 nvs\_enable**  
WiFi NVS flash enable flag

**int nano\_enable**  
Nano option for printf/scan family enable flag

**int rx\_ba\_win**  
WiFi Block Ack RX window size

**int wifi\_task\_core\_id**  
WiFi Task Core ID

**int beacon\_max\_len**  
WiFi softAP maximum length of the beacon

**int mgmt\_sbuf\_num**  
WiFi management short buffer number, the minimum value is 6, the maximum value is 32

**uint64\_t feature\_caps**  
Enables additional WiFi features and capabilities

**int magic**  
WiFi init magic number, it should be the last field

### Macros

**ESP\_ERR\_WIFI\_NOT\_INIT**  
WiFi driver was not installed by esp\_wifi\_init

**ESP\_ERR\_WIFI\_NOT\_STARTED**  
WiFi driver was not started by esp\_wifi\_start

**ESP\_ERR\_WIFI\_NOT\_STOPPED**  
WiFi driver was not stopped by esp\_wifi\_stop

**ESP\_ERR\_WIFI\_IF**  
WiFi interface error

**ESP\_ERR\_WIFI\_MODE**  
WiFi mode error

**ESP\_ERR\_WIFI\_STATE**  
WiFi internal state error

**ESP\_ERR\_WIFI\_CONN**  
WiFi internal control block of station or soft-AP error

**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**

**WIFI\_AMPDU\_TX\_ENABLED**

**WIFI\_NVS\_ENABLED**

**WIFI\_NANO\_FORMAT\_ENABLED**

**WIFI\_INIT\_CONFIG\_MAGIC**

**WIFI\_DEFAULT\_RX\_BA\_WIN**

**WIFI\_TASK\_CORE\_ID**

**WIFI\_SOFTAP\_BEACON\_MAX\_LEN**

**WIFI\_MGMT\_SBUF\_NUM**

**CONFIG\_FEATURE\_WPA3\_SAE\_BIT**

**CONFIG\_FEATURE\_CACHE\_TX\_BUF\_BIT**

**WIFI\_INIT\_CONFIG\_DEFAULT()**

### Type Definitions

**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.

### Parameters

- `buf`: Data received. Type of data in buffer (`wifi_promiscuous_pkt_t` or `wifi_pkt_rx_ctrl_t`) indicated by 'type' parameter.
- `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.

#### Parameters

- `ctx`: Context argument, as passed to `esp_wifi_set_vendor_ie_cb()` when registering callback.
- `type`: Information element type, based on frame type received.
- `sa`: Source 802.11 address.
- `vnd_ie`: Pointer to the vendor specific element data received.
- `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.

#### Parameters

- `ctx`: context argument, passed to `esp_wifi_set_csi_rx_cb()` when registering callback function.
- `data`: CSI data received. The memory that it points to will be deallocated after callback function returns.

### Header File

- [esp\\_wifi/include/esp\\_wifi\\_types.h](#)

### Unions

```
union wifi_config_t
```

```
#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

```
wifi_ap_config_t ap  
configuration of AP
```

```
wifi_sta_config_t sta  
configuration of STA
```

### Structures

```
struct wifi_country_t
```

Structure describing WiFi country-based regional restrictions.

#### Public Members

```
char cc[3]  
country code string
```

```
uint8_t schan  
start channel
```

```
uint8_t nchan  
total channel number
```

`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

`wifi_scan_type_t scan_type`

scan type, active or passive

`wifi_scan_time_t scan_time`

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 second`  
secondary channel of AP

`int8_t rssi`  
signal strength of AP

`wifi_auth_mode_t authmode`  
authmode of AP

`wifi_cipher_type_t pairwise_cipher`  
pairwise cipher of AP

`wifi_cipher_type_t group_cipher`  
group cipher of AP

`wifi_ant_t ant`  
antenna used to receive beacon from AP

`uint32_t phy_11b : 1`  
bit: 0 flag to identify if 11b mode is enabled or not

`uint32_t phy_11g : 1`  
bit: 1 flag to identify if 11g mode is enabled or not

`uint32_t phy_11n : 1`  
bit: 2 flag to identify if 11n mode is enabled or not

`uint32_t phy_1r : 1`  
bit: 3 flag to identify if low rate is enabled or not

`uint32_t wps : 1`  
bit: 4 flag to identify if WPS is supported or not

`uint32_t reserved : 27`  
bit: 5..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

**struct wifi\_pmf\_config\_t**  
Configuration structure for Protected Management Frame

### Public Members

`bool capable`  
Advertizes support for Protected Management Frame. Device will prefer to 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

**struct wifi\_sta\_config\_t**

STA configuration settings for the ESP32.

### Public Members

uint8\_t **ssid**[32]

SSID of target AP.

uint8\_t **password**[64]

Password of target AP.

*wifi\_scan\_method\_t* **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.

uint8\_t **bssid**[6]

MAC address of target AP

uint8\_t **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.

uint16\_t **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.

*wifi\_sort\_method\_t* **sort\_method**  
sort the connect AP in the list by rssi or security mode

*wifi\_scan\_threshold\_t* **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.

*wifi\_pmf\_config\_t* **pmf\_cfg**  
Configuration for Protected Management Frame. Will be advertised in RSN Capabilities in RSN IE.

**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** : 1  
bit: 0 flag to identify if 11b mode is enabled or not

uint32\_t **phy\_11g** : 1  
bit: 1 flag to identify if 11g mode is enabled or not

uint32\_t **phy\_11n** : 1  
bit: 2 flag to identify if 11n mode is enabled or not

uint32\_t **phy\_lr** : 1  
bit: 3 flag to identify if low rate is enabled or not

uint32\_t **reserved** : 28  
bit: 4..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** : 8

Received Signal Strength Indicator(RSSI) of packet. unit: dBm

unsigned **rate** : 5

PHY rate encoding of the packet. Only valid for non HT(11bg) packet

unsigned **\_\_pad0\_\_** : 1

reserved

unsigned **sig\_mode** : 2

0: non HT(11bg) packet; 1: HT(11n) packet; 3: VHT(11ac) packet

unsigned **\_\_pad1\_\_** : 16

reserved

unsigned **mcs** : 7

Modulation Coding Scheme. If is HT(11n) packet, shows the modulation, range from 0 to 76(MSC0 ~ MCS76)

unsigned **cwb** : 1

Channel Bandwidth of the packet. 0: 20MHz; 1: 40MHz

unsigned **\_\_pad2\_\_** : 16

reserved

unsigned **smoothing** : 1

reserved

unsigned **not\_sounding** : 1

reserved

unsigned **\_\_pad3\_\_** : 1

reserved

unsigned **aggregation** : 1

Aggregation. 0: MPDU packet; 1: AMPDU packet

unsigned **stbc** : 2

Space Time Block Code(STBC). 0: non STBC packet; 1: STBC packet

unsigned **fec\_coding** : 1

Flag is set for 11n packets which are LDPC

unsigned **sgi** : 1

Short Guide Interval(SGI). 0: Long GI; 1: Short GI

signed **noise\_floor** : 8

noise floor of Radio Frequency Module(RF). unit: 0.25dBm

unsigned **ampdu\_cnt** : 8

ampdu cnt

unsigned **channel** : 4

primary channel on which this packet is received

unsigned **secondary\_channel** : 4  
secondary channel on which this packet is received. 0: none; 1: above; 2: below

unsigned **\_\_pad4\_\_** : 8  
reserved

unsigned **timestamp** : 32  
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\_\_** : 32  
reserved

unsigned **\_\_pad6\_\_** : 31  
reserved

unsigned **ant** : 1  
antenna number from which this packet is received. 0: WiFi antenna 0; 1: WiFi antenna 1

unsigned **sig\_len** : 12  
length of packet including Frame Check Sequence(FCS)

unsigned **\_\_pad7\_\_** : 12  
reserved

unsigned **rx\_state** : 8  
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 htlft 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** : 1  
Whether this GPIO is connected to external antenna switch

uint8\_t **gpio\_num** : 7  
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** : 4

Index (in antenna GPIO configuration) of enabled WIFI\_ANT\_MODE\_ANT0

uint8\_t **enabled\_ant1** : 4

Index (in antenna GPIO configuration) of enabled WIFI\_ANT\_MODE\_ANT1

**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

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

**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

**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

**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

**struct wifi\_event\_ap\_probe\_req\_rx\_t**

Argument structure for WIFI\_EVENT\_AP\_PROBEREQRCVED event

### Public Members

int **rss**i  
Received probe request signal strength

uint8\_t **mac**[6]  
MAC address of the station which send probe request

### Macros

**WIFI\_IF\_STA**

**WIFI\_IF\_AP**

**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 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

**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\_PROBEREQRECVD**  
mask SYSTEM\_EVENT\_AP\_PROBEREQRECVD 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\_ALL**

### Type Definitions

```
typedef esp_interface_t wifi_interface_t
```

### Enumerations

```
enum wifi_mode_t
```

*Values:*

**WIFI\_MODE\_NULL** = 0  
null mode

**WIFI\_MODE\_STA**  
WiFi station mode

**WIFI\_MODE\_AP**  
WiFi soft-AP mode

**WIFI\_MODE\_APSTA**  
WiFi station + soft-AP mode

**WIFI\_MODE\_MAX**

```
enum wifi_country_policy_t
```

*Values:*

**WIFI\_COUNTRY\_POLICY\_AUTO**  
Country policy is auto, use the country info of AP to which the station is connected

**WIFI\_COUNTRY\_POLICY\_MANUAL**  
Country policy is manual, always use the configured country info

```
enum wifi_auth_mode_t
```

*Values:*

**WIFI\_AUTH\_OPEN** = 0  
authenticate mode : open

**WIFI\_AUTH\_WEP**  
authenticate mode : WEP

**WIFI\_AUTH\_WPA\_PSK**  
authenticate mode : WPA\_PSK

**WIFI\_AUTH\_WPA2\_PSK**  
authenticate mode : WPA2\_PSK

**WIFI\_AUTH\_WPA\_WPA2\_PSK**  
authenticate mode : WPA\_WPA2\_PSK

**WIFI\_AUTH\_WPA2\_ENTERPRISE**  
authenticate mode : WPA2\_ENTERPRISE

**WIFI\_AUTH\_WPA3\_PSK**  
authenticate mode : WPA3\_PSK

**WIFI\_AUTH\_WPA2\_WPA3\_PSK**  
authenticate mode : WPA2\_WPA3\_PSK

**WIFI\_AUTH\_MAX**

**enum wifi\_err\_reason\_t**

*Values:*

**WIFI\_REASON\_UNSPECIFIED** = 1

**WIFI\_REASON\_AUTH\_EXPIRE** = 2

**WIFI\_REASON\_AUTH\_LEAVE** = 3

**WIFI\_REASON\_ASSOC\_EXPIRE** = 4

**WIFI\_REASON\_ASSOC\_TOOMANY** = 5

**WIFI\_REASON\_NOT\_AUTHED** = 6

**WIFI\_REASON\_NOT\_ASSOCED** = 7

**WIFI\_REASON\_ASSOC\_LEAVE** = 8

**WIFI\_REASON\_ASSOC\_NOT\_AUTHED** = 9

**WIFI\_REASON\_DISASSOC\_PWRCAP\_BAD** = 10

**WIFI\_REASON\_DISASSOC\_SUPCHAN\_BAD** = 11

**WIFI\_REASON\_IE\_INVALID** = 13

**WIFI\_REASON\_MIC\_FAILURE** = 14

**WIFI\_REASON\_4WAY\_HANDSHAKE\_TIMEOUT** = 15

**WIFI\_REASON\_GROUP\_KEY\_UPDATE\_TIMEOUT** = 16

**WIFI\_REASON\_IE\_IN\_4WAY\_DIFFERS** = 17

**WIFI\_REASON\_GROUP\_CIPHER\_INVALID** = 18

**WIFI\_REASON\_PAIRWISE\_CIPHER\_INVALID** = 19

**WIFI\_REASON\_AKMP\_INVALID** = 20

**WIFI\_REASON\_UNSUPP\_RSN\_IE\_VERSION** = 21

**WIFI\_REASON\_INVALID\_RSN\_IE\_CAP** = 22

**WIFI\_REASON\_802\_1X\_AUTH\_FAILED** = 23

**WIFI\_REASON\_CIPHER\_SUITE\_REJECTED** = 24

**WIFI\_REASON\_INVALID\_PMKID** = 53  
**WIFI\_REASON\_BEACON\_TIMEOUT** = 200  
**WIFI\_REASON\_NO\_AP\_FOUND** = 201  
**WIFI\_REASON\_AUTH\_FAIL** = 202  
**WIFI\_REASON\_ASSOC\_FAIL** = 203  
**WIFI\_REASON\_HANDSHAKE\_TIMEOUT** = 204  
**WIFI\_REASON\_CONNECTION\_FAIL** = 205  
**WIFI\_REASON\_AP\_TSF\_RESET** = 206

**enum wifi\_second\_chan\_t**

*Values:*

**WIFI\_SECOND\_CHAN\_NONE** = 0  
the channel width is HT20  
**WIFI\_SECOND\_CHAN\_ABOVE**  
the channel width is HT40 and the secondary channel is above the primary channel  
**WIFI\_SECOND\_CHAN\_BELOW**  
the channel width is HT40 and the secondary channel is below the primary channel

**enum wifi\_scan\_type\_t**

*Values:*

**WIFI\_SCAN\_TYPE\_ACTIVE** = 0  
active scan  
**WIFI\_SCAN\_TYPE\_PASSIVE**  
passive scan

**enum wifi\_cipher\_type\_t**

*Values:*

**WIFI\_CIPHER\_TYPE\_NONE** = 0  
the cipher type is none  
**WIFI\_CIPHER\_TYPE\_WEP40**  
the cipher type is WEP40  
**WIFI\_CIPHER\_TYPE\_WEP104**  
the cipher type is WEP104  
**WIFI\_CIPHER\_TYPE\_TKIP**  
the cipher type is TKIP  
**WIFI\_CIPHER\_TYPE\_CCMP**  
the cipher type is CCMP  
**WIFI\_CIPHER\_TYPE\_TKIP\_CCMP**  
the cipher type is TKIP and CCMP  
**WIFI\_CIPHER\_TYPE\_AES\_CMAC128**  
the cipher type is AES-CMAC-128  
**WIFI\_CIPHER\_TYPE\_UNKNOWN**  
the cipher type is unknown

**enum wifi\_ant\_t**

WiFi antenna.

*Values:*

**WIFI\_ANT\_ANT0**  
WiFi antenna 0

**WIFI\_ANT\_ANT1**  
WiFi antenna 1

**WIFI\_ANT\_MAX**  
Invalid WiFi antenna

**enum wifi\_scan\_method\_t**  
*Values:*

**WIFI\_FAST\_SCAN = 0**  
Do fast scan, scan will end after find SSID match AP

**WIFI\_ALL\_CHANNEL\_SCAN**  
All channel scan, scan will end after scan all the channel

**enum wifi\_sort\_method\_t**  
*Values:*

**WIFI\_CONNECT\_AP\_BY\_SIGNAL = 0**  
Sort match AP in scan list by RSSI

**WIFI\_CONNECT\_AP\_BY\_SECURITY**  
Sort match AP in scan list by security mode

**enum wifi\_ps\_type\_t**  
*Values:*

**WIFI\_PS\_NONE**  
No power save

**WIFI\_PS\_MIN\_MODEM**  
Minimum modem power saving. In this mode, station wakes up to receive beacon every DTIM period

**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:*

**WIFI\_BW\_HT20 = 1**

**WIFI\_BW\_HT40**

**enum wifi\_storage\_t**  
*Values:*

**WIFI\_STORAGE\_FLASH**  
all configuration will store in both memory and flash

**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:*

**WIFI\_VND\_IE\_TYPE\_BEACON**

**WIFI\_VND\_IE\_TYPE\_PROBE\_REQ**

**WIFI\_VND\_IE\_TYPE\_PROBE\_RESP**

**WIFI\_VND\_IE\_TYPE\_ASSOC\_REQ**

**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:*

**WIFI\_VND\_IE\_ID\_0**

**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:*

**WIFI\_PKT\_MGMT**

Management frame, indicates ‘buf’ argument is *wifi\_promiscuous\_pkt\_t*

**WIFI\_PKT\_CTRL**

Control frame, indicates ‘buf’ argument is *wifi\_promiscuous\_pkt\_t*

**WIFI\_PKT\_DATA**

Data frame, indicates ‘buf’ argument is *wifi\_promiscuous\_pkt\_t*

**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:*

**WIFI\_ANT\_MODE\_ANT0**

Enable WiFi antenna 0 only

**WIFI\_ANT\_MODE\_ANT1**

Enable WiFi antenna 1 only

**WIFI\_ANT\_MODE\_AUTO**

Enable WiFi antenna 0 and 1, automatically select an antenna

**WIFI\_ANT\_MODE\_MAX**

Invalid WiFi enabled antenna

**enum wifi\_phy\_rate\_t**

WiFi PHY rate encodings.

*Values:*

**WIFI\_PHY\_RATE\_1M\_L** = 0x00

1 Mbps with long preamble

**WIFI\_PHY\_RATE\_2M\_L** = 0x01

2 Mbps with long preamble

**WIFI\_PHY\_RATE\_5M\_L** = 0x02

5.5 Mbps with long preamble

**WIFI\_PHY\_RATE\_11M\_L** = 0x03

11 Mbps with long preamble

**WIFI\_PHY\_RATE\_2M\_S** = 0x05

2 Mbps with short preamble

**WIFI\_PHY\_RATE\_5M\_S** = 0x06

5.5 Mbps with short preamble

**WIFI\_PHY\_RATE\_11M\_S** = 0x07  
11 Mbps with short preamble

**WIFI\_PHY\_RATE\_48M** = 0x08  
48 Mbps

**WIFI\_PHY\_RATE\_24M** = 0x09  
24 Mbps

**WIFI\_PHY\_RATE\_12M** = 0x0A  
12 Mbps

**WIFI\_PHY\_RATE\_6M** = 0x0B  
6 Mbps

**WIFI\_PHY\_RATE\_54M** = 0x0C  
54 Mbps

**WIFI\_PHY\_RATE\_36M** = 0x0D  
36 Mbps

**WIFI\_PHY\_RATE\_18M** = 0x0E  
18 Mbps

**WIFI\_PHY\_RATE\_9M** = 0x0F  
9 Mbps

**WIFI\_PHY\_RATE\_MCS0\_LGI** = 0x10  
MCS0 with long GI, 6.5 Mbps for 20MHz, 13.5 Mbps for 40MHz

**WIFI\_PHY\_RATE\_MCS1\_LGI** = 0x11  
MCS1 with long GI, 13 Mbps for 20MHz, 27 Mbps for 40MHz

**WIFI\_PHY\_RATE\_MCS2\_LGI** = 0x12  
MCS2 with long GI, 19.5 Mbps for 20MHz, 40.5 Mbps for 40MHz

**WIFI\_PHY\_RATE\_MCS3\_LGI** = 0x13  
MCS3 with long GI, 26 Mbps for 20MHz, 54 Mbps for 40MHz

**WIFI\_PHY\_RATE\_MCS4\_LGI** = 0x14  
MCS4 with long GI, 39 Mbps for 20MHz, 81 Mbps for 40MHz

**WIFI\_PHY\_RATE\_MCS5\_LGI** = 0x15  
MCS5 with long GI, 52 Mbps for 20MHz, 108 Mbps for 40MHz

**WIFI\_PHY\_RATE\_MCS6\_LGI** = 0x16  
MCS6 with long GI, 58.5 Mbps for 20MHz, 121.5 Mbps for 40MHz

**WIFI\_PHY\_RATE\_MCS7\_LGI** = 0x17  
MCS7 with long GI, 65 Mbps for 20MHz, 135 Mbps for 40MHz

**WIFI\_PHY\_RATE\_MCS0\_SGI** = 0x18  
MCS0 with short GI, 7.2 Mbps for 20MHz, 15 Mbps for 40MHz

**WIFI\_PHY\_RATE\_MCS1\_SGI** = 0x19  
MCS1 with short GI, 14.4 Mbps for 20MHz, 30 Mbps for 40MHz

**WIFI\_PHY\_RATE\_MCS2\_SGI** = 0x1A  
MCS2 with short GI, 21.7 Mbps for 20MHz, 45 Mbps for 40MHz

**WIFI\_PHY\_RATE\_MCS3\_SGI** = 0x1B  
MCS3 with short GI, 28.9 Mbps for 20MHz, 60 Mbps for 40MHz

**WIFI\_PHY\_RATE\_MCS4\_SGI** = 0x1C  
MCS4 with short GI, 43.3 Mbps for 20MHz, 90 Mbps for 40MHz

**WIFI\_PHY\_RATE\_MCS5\_SGI** = 0x1D  
MCS5 with short GI, 57.8 Mbps for 20MHz, 120 Mbps for 40MHz

**WIFI\_PHY\_RATE\_MCS6\_SGI** = 0x1E  
MCS6 with short GI, 65 Mbps for 20MHz, 135 Mbps for 40MHz

**WIFI\_PHY\_RATE\_MCS7\_SGI** = 0x1F  
MCS7 with short GI, 72.2 Mbps for 20MHz, 150 Mbps for 40MHz

**WIFI\_PHY\_RATE\_LORA\_250K** = 0x29  
250 Kbps

**WIFI\_PHY\_RATE\_LORA\_500K** = 0x2A  
500 Kbps

**WIFI\_PHY\_RATE\_MAX**

**enum wifi\_event\_t**

WiFi event declarations

*Values:*

**WIFI\_EVENT\_WIFI\_READY** = 0  
ESP32 WiFi ready

**WIFI\_EVENT\_SCAN\_DONE**  
ESP32 finish scanning AP

**WIFI\_EVENT\_STA\_START**  
ESP32 station start

**WIFI\_EVENT\_STA\_STOP**  
ESP32 station stop

**WIFI\_EVENT\_STA\_CONNECTED**  
ESP32 station connected to AP

**WIFI\_EVENT\_STA\_DISCONNECTED**  
ESP32 station disconnected from AP

**WIFI\_EVENT\_STA\_AUTHMODE\_CHANGE**  
the auth mode of AP connected by ESP32 station changed

**WIFI\_EVENT\_STA\_WPS\_ER\_SUCCESS**  
ESP32 station wps succeeds in enrollee mode

**WIFI\_EVENT\_STA\_WPS\_ER\_FAILED**  
ESP32 station wps fails in enrollee mode

**WIFI\_EVENT\_STA\_WPS\_ER\_TIMEOUT**  
ESP32 station wps timeout in enrollee mode

**WIFI\_EVENT\_STA\_WPS\_ER\_PIN**  
ESP32 station wps pin code in enrollee mode

**WIFI\_EVENT\_STA\_WPS\_ER\_PBC\_OVERLAP**  
ESP32 station wps overlap in enrollee mode

**WIFI\_EVENT\_AP\_START**  
ESP32 soft-AP start

**WIFI\_EVENT\_AP\_STOP**  
ESP32 soft-AP stop

**WIFI\_EVENT\_AP\_STACONNECTED**  
a station connected to ESP32 soft-AP

**WIFI\_EVENT\_AP\_STADISCONNECTED**  
a station disconnected from ESP32 soft-AP

**WIFI\_EVENT\_AP\_PROBEREQRCVD**  
Receive probe request packet in soft-AP interface

**WIFI\_EVENT\_STA\_BEACON\_TIMEOUT**

ESP32 station beacon timeout

**enum wifi\_event\_sta\_wps\_fail\_reason\_t**

Argument structure for WIFI\_EVENT\_STA\_WPS\_ER\_FAILED event

*Values:*

**WPS\_FAIL\_REASON\_NORMAL = 0**

ESP32 WPS normal fail reason

**WPS\_FAIL\_REASON\_RECV\_M2D**

ESP32 WPS receive M2D frame

**WPS\_FAIL\_REASON\_MAX**

## SmartConfig

### API 参考

#### Header File

- [esp\\_wifi/include/esp\\_smartconfig.h](#)

#### Functions

**const** char \***esp\_smartconfig\_get\_version** (void)

Get the version of SmartConfig.

##### Return

- 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.

##### Return

- ESP\_OK: succeed
- others: fail

##### Parameters

- config: pointer to smartconfig start configure structure

*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`.

##### Return

- 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.

##### Return

- ESP\_OK: succeed
- others: fail

##### Parameters

- time\_s: range 15s~255s, offset:45s.



*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`.

**Return**

- ESP\_OK: succeed
- others: fail

**Parameters**

- type: Choose from the `smartconfig_type_t`.

*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`.

**Attention** 2. Fast mode have corresponding APP(phone).

**Attention** 3. Two mode is compatible.

**Return**

- ESP\_OK: succeed
- others: fail

**Parameters**

- enable: false-disable(default); true-enable;

## 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.

*smartconfig\_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.

## Macros

**SMARTCONFIG\_START\_CONFIG\_DEFAULT** ()

**Enumerations****enum smartconfig\_type\_t**

Values:

**SC\_TYPE\_ESPTOUCH = 0**

protocol: ESPTouch

**SC\_TYPE\_AIRKISS**

protocol: AirKiss

**SC\_TYPE\_ESPTOUCH\_AIRKISS**

protocol: ESPTouch and AirKiss

**enum smartconfig\_event\_t**

Smartconfig event declarations

Values:

**SC\_EVENT\_SCAN\_DONE**

ESP32 station smartconfig has finished to scan for APs

**SC\_EVENT\_FOUND\_CHANNEL**

ESP32 station smartconfig has found the channel of the target AP

**SC\_EVENT\_GOT\_SSID\_PSWD**

ESP32 station smartconfig got the SSID and password

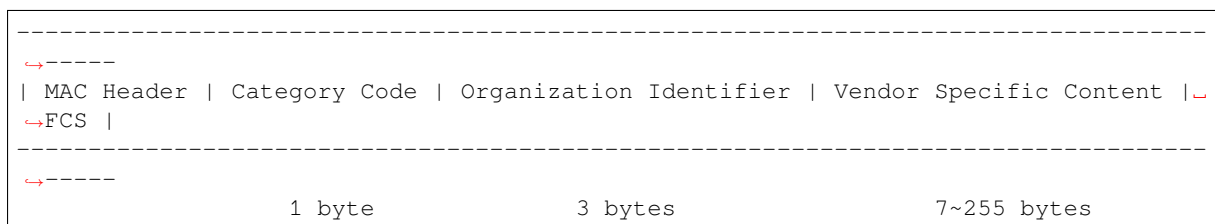
**SC\_EVENT\_SEND\_ACK\_DONE**

ESP32 station smartconfig has sent ACK to cellphone

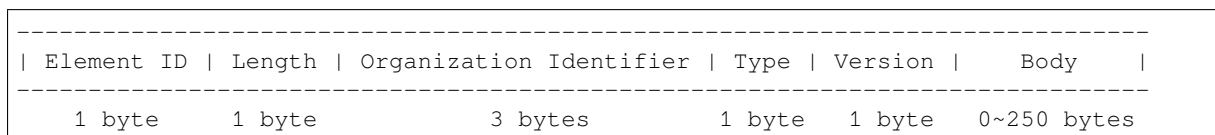
**ESP-NOW**

**Overview** ESP-NOW is a kind of connectionless WiFi communication protocol which is defined by Espressif. In ESP-NOW, application data is encapsulated in vendor-specific action frame and then transmitted from one WiFi device to another without connection. CTR with CBC-MAC Protocol(CCMP) is used to protect the action frame for security. ESP-NOW is widely used in smart light, remote controlling, sensor, etc.

**Frame Format** ESP-NOW uses vendor-specific action frame to transmit ESP-NOW data. The format of vendor-specific action frame is as follows:



- Category Code: The Category field is set to the value(127) indicating the vendor-specific category.
- **Organization Identifier: The Organization Identifier contains a unique identifier(0x18fe34) which is the first three bytes of MAC address applied by Espressif.**
- Vendor Specific Content: The Vendor Specific Content contains vendor-specific field as follows:



- Element ID: The Element ID field is set to the value(221) indicating the vendor-specific element.
- Length: The length is the total length of Organization Identifier, Type, Version and Body.
- **Organization Identifier: The Organization Identifier contains a unique identifier(0x18fe34) which is the first three bytes of MAC address applied by Espressif.**
- Type: The Type field is set to the value(4) indicating ESP-NOW.

- Version: The Version field is set to the version of ESP-NOW.
- Body: The Body contains the ESP-NOW data.

As ESP-NOW is connectionless, the MAC header is a little different from that of standard frames. The FromDS and ToDS bits of FrameControl field are both 0. The first address field is set to the destination address. The second address field is set to the source address. The third address field is set to broadcast address(0xff:0xff:0xff:0xff:0xff:0xff).

**Security** ESP-NOW use CCMP method which can be referenced in IEEE Std. 802.11-2012 to protect the vendor-specific action frame. The WiFi device maintains a Primary Master Key(PMK) and several Local Master Keys(LMK). The lengths of them are 16 bytes. PMK is used to encrypt LMK with AES-128 algorithm. Call `esp_now_set_pmk()` to set PMK. If PMK is not set, a default PMK will be used. If LMK of the paired device is set, it will be used to encrypt the vendor-specific action frame with CCMP method. The maximum number of different LMKs is six. Do not support encrypting multicast vendor-specific action frame.

**Initialization and De-initialization** Call `esp_now_init()` to initialize ESP-NOW and `esp_now_deinit()` to de-initialize ESP-NOW. ESP-NOW data must be transmitted after WiFi is started, so it is recommended to start WiFi before initializing ESP-NOW and stop WiFi after de-initializing ESP-NOW. When `esp_now_deinit()` is called, all of the information of paired devices will be deleted.

**Add Paired Device** Before sending data to other device, call `esp_now_add_peer()` to add it to the paired device list first. The maximum number of paired devices is twenty. If security is enabled, the LMK must be set. ESP-NOW data can be sent from station or softap interface. Make sure that the interface is enabled before sending ESP-NOW data. A device with broadcast MAC address must be added before sending broadcast data. The range of the channel of paired device is from 0 to 14. If the channel is set to 0, data will be sent on the current channel. Otherwise, the channel must be set as the channel that the local device is on.

**Send ESP-NOW Data** Call `esp_now_send()` to send ESP-NOW data and `esp_now_register_send_cb` to register sending callback function. It will return `ESP_NOW_SEND_SUCCESS` in sending callback function if the data is received successfully on MAC layer. Otherwise, it will return `ESP_NOW_SEND_FAIL`. There are several reasons failing to send ESP-NOW data, for example, the destination device doesn't exist, the channels of the devices are not the same, the action frame is lost when transmitting on the air, etc. It is not guaranteed that application layer can receive the data. If necessary, send back ack data when receiving ESP-NOW data. If receiving ack data timeout happens, retransmit the ESP-NOW data. A sequence number can also be assigned to ESP-NOW data to drop the duplicated data.

If there is a lot of ESP-NOW data to send, call `esp_now_send()` to send less than or equal to 250 bytes of data once a time. Note that too short interval between sending two ESP-NOW datas may lead to disorder of sending callback function. So, it is recommended that sending the next ESP-NOW data after the sending callback function of previous sending has returned. The sending callback function runs from a high-priority WiFi task. So, do not do lengthy operations in the callback function. Instead, post necessary data to a queue and handle it from a lower priority task.

**Receiving ESP-NOW Data** Call `esp_now_register_rcv_cb` to register receiving callback function. When receiving ESP-NOW data, receiving callback function is called. The receiving callback function also runs from WiFi task. So, do not do lengthy operations in the callback function. Instead, post necessary data to a queue and handle it from a lower priority task.

## API Reference

### Header File

- [esp\\_wifi/include/esp\\_now.h](#)

**Functions*****esp\_err\_t* esp\_now\_init** (void)

Initialize ESPNOW function.

**Return**

- ESP\_OK : succeed
- ESP\_ERR\_ESPNOW\_INTERNAL : Internal error

***esp\_err\_t* esp\_now\_deinit** (void)

De-initialize ESPNOW function.

**Return**

- ESP\_OK : succeed

***esp\_err\_t* esp\_now\_get\_version** (uint32\_t \**version*)

Get the version of ESPNOW.

**Return**

- ESP\_OK : succeed
- ESP\_ERR\_ESPNOW\_ARG : invalid argument

**Parameters**

- *version*: ESPNOW version

***esp\_err\_t* esp\_now\_register\_recv\_cb** (*esp\_now\_recv\_cb\_t* *cb*)

Register callback function of receiving ESPNOW data.

**Return**

- ESP\_OK : succeed
- ESP\_ERR\_ESPNOW\_NOT\_INIT : ESPNOW is not initialized
- ESP\_ERR\_ESPNOW\_INTERNAL : internal error

**Parameters**

- *cb*: callback function of receiving ESPNOW data

***esp\_err\_t* esp\_now\_unregister\_recv\_cb** (void)

Unregister callback function of receiving ESPNOW data.

**Return**

- ESP\_OK : succeed
- ESP\_ERR\_ESPNOW\_NOT\_INIT : ESPNOW is not initialized

***esp\_err\_t* esp\_now\_register\_send\_cb** (*esp\_now\_send\_cb\_t* *cb*)

Register callback function of sending ESPNOW data.

**Return**

- ESP\_OK : succeed
- ESP\_ERR\_ESPNOW\_NOT\_INIT : ESPNOW is not initialized
- ESP\_ERR\_ESPNOW\_INTERNAL : internal error

**Parameters**

- *cb*: callback function of sending ESPNOW data

***esp\_err\_t* esp\_now\_unregister\_send\_cb** (void)

Unregister callback function of sending ESPNOW data.

**Return**

- ESP\_OK : succeed
- ESP\_ERR\_ESPNOW\_NOT\_INIT : ESPNOW is not initialized

***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**Attention** 3. The maximum length of *data* must be less than ESP\_NOW\_MAX\_DATA\_LEN**Attention** 4. The buffer pointed to by *data* argument does not need to be valid after *esp\_now\_send* returns**Return**

- 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
- `ESP_ERR_ESPNOW_NOT_FOUND` : peer is not found
- `ESP_ERR_ESPNOW_IF` : current WiFi interface doesn't match that of peer

**Parameters**

- `peer_addr`: peer MAC address
- `data`: data to send
- `len`: length of data

`esp_err_t esp_now_add_peer (const esp_now_peer_info_t *peer)`

Add a peer to peer list.

**Return**

- `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

**Parameters**

- `peer`: peer information

`esp_err_t esp_now_del_peer (const uint8_t *peer_addr)`

Delete a peer from peer list.

**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

**Parameters**

- `peer_addr`: peer MAC address

`esp_err_t esp_now_mod_peer (const esp_now_peer_info_t *peer)`

Modify a peer.

**Return**

- `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

**Parameters**

- `peer`: peer information

`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.

**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

**Parameters**

- `peer_addr`: peer MAC address
- `peer`: peer information

`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.

**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

**Parameters**

- `from_head`: fetch from head of list or not
- `peer`: peer information

bool **esp\_now\_is\_peer\_exist** (const uint8\_t \*peer\_addr)

Peer exists or not.

**Return**

- true : peer exists
- false : peer not exists

**Parameters**

- `peer_addr`: peer MAC address

*esp\_err\_t* **esp\_now\_get\_peer\_num** (*esp\_now\_peer\_num\_t* \*num)

Get the number of peers.

**Return**

- ESP\_OK : succeed
- ESP\_ERR\_ESPNOW\_NOT\_INIT : ESPNOW is not initialized
- ESP\_ERR\_ESPNOW\_ARG : invalid argument

**Parameters**

- `num`: number of peers

*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

**Return**

- ESP\_OK : succeed
- ESP\_ERR\_ESPNOW\_NOT\_INIT : ESPNOW is not initialized
- ESP\_ERR\_ESPNOW\_ARG : invalid argument

**Parameters**

- `pmk`: primary master key

**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

**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.

**Parameters**

- `mac_addr`: peer MAC address
- `data`: received data
- `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.

#### Parameters

- `mac_addr`: peer MAC address
- `status`: status of sending ESPNOW data (succeed or fail)

#### Enumerations

```
enum esp_now_send_status_t
```

Status of sending ESPNOW data .

Values:

```
ESP_NOW_SEND_SUCCESS = 0
```

Send ESPNOW data successfully

```
ESP_NOW_SEND_FAIL
```

Send ESPNOW data fail

### ESP-MESH Programming Guide

This is a programming guide for ESP-MESH, including the API reference and coding examples. This guide is split into the following parts:

1. [ESP-MESH Programming Model](#)
2. [Writing an ESP-MESH Application](#)
3. [Self Organized Networking](#)
4. [Application Examples](#)
5. [API Reference](#)

For documentation regarding the ESP-MESH protocol, please see the [ESP-MESH API Guide](#).

### ESP-MESH Programming Model

**Software Stack** The ESP-MESH software stack is built atop the Wi-Fi Driver/FreeRTOS and may use the LwIP Stack in some instances (i.e. the root node). The following diagram illustrates the ESP-MESH software stack.

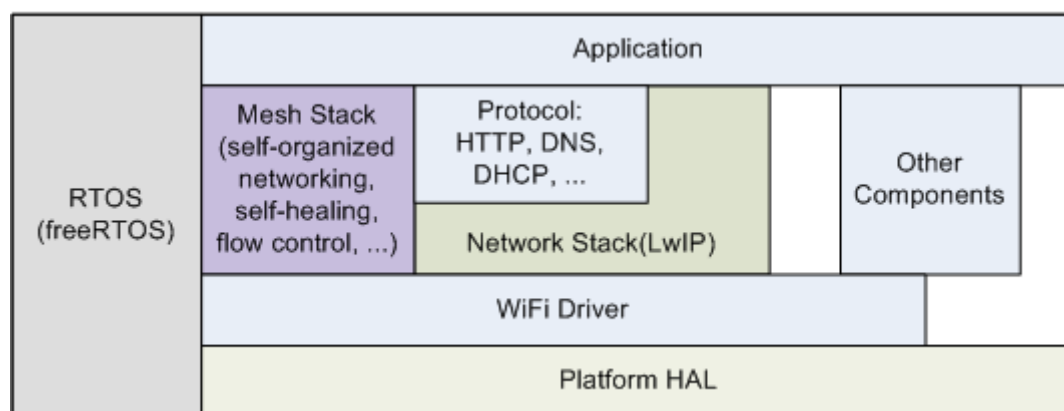


图 2: ESP-MESH Software Stack



**System Events** An application interfaces with ESP-MESH via **ESP-MESH Events**. Since ESP-MESH is built atop the Wi-Fi stack, it is also possible for the application to interface with the Wi-Fi driver via the **Wi-Fi Event Task**. The following diagram illustrates the interfaces for the various System Events in an ESP-MESH application.

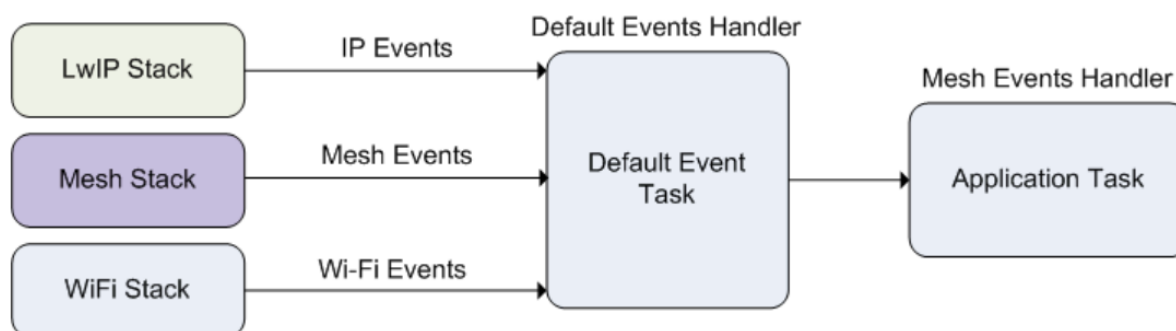


图 3: ESP-MESH System Events Delivery

The `mesh_event_id_t` defines all possible ESP-MESH events and can indicate events such as the connection/disconnection of parent/child. Before ESP-MESH events can be used, the application must register a **Mesh Events handler** via `esp_event_handler_register()` to the default event task. Should contain handlers for each event relevant to the application.

Typical use cases of mesh events include using events such as `MESH_EVENT_PARENT_CONNECTED` and `MESH_EVENT_CHILD_CONNECTED` to indicate when a node can begin transmitting data upstream and downstream respectively. Likewise, `IP_EVENT_STA_GOT_IP` and `IP_EVENT_STA_LOST_IP` can be used to indicate when the root node can and cannot transmit data to the external IP network.

**警告:** When using ESP-MESH under self-organized mode, users must ensure that no calls to Wi-Fi API are made. This is due to the fact that the self-organizing mode will internally make Wi-Fi API calls to connect/disconnect/scan etc. **Any Wi-Fi calls from the application (including calls from callbacks and handlers of Wi-Fi events) may interfere with ESP-MESH's self-organizing behavior.** Therefore, user's should not call Wi-Fi APIs after `esp_mesh_start()` is called, and before `esp_mesh_stop()` is called.

**LwIP & ESP-MESH** The application can access the ESP-MESH stack directly without having to go through the LwIP stack. The LwIP stack is only required by the root node to transmit/receive data to/from an external IP network. However, since every node can potentially become the root node (due to automatic root node selection), each node must still initialize the LwIP stack.

**Each node is required to initialize LwIP by calling `tcpip_adapter_init()`.** In order to prevent non-root node access to LwIP, the application should stop the following services after LwIP initialization:

- DHCP server service on the softAP interface.
- DHCP client service on the station interface.

The following code snippet demonstrates how to initialize LwIP for ESP-MESH applications.

```

/* tcpip initialization */
tcpip_adapter_init();
/*
 * for mesh
 * stop DHCP server on softAP interface by default
 * stop DHCP client on station interface by default
 */
ESP_ERROR_CHECK(tcpip_adapter_dhcps_stop(TCPIP_ADAPTER_IF_AP));
ESP_ERROR_CHECK(tcpip_adapter_dhcpc_stop(TCPIP_ADAPTER_IF_STA));
  
```

**注解:** ESP-MESH requires a root node to be connected with a router. Therefore, in the event that a node becomes

the root, **the corresponding handler must start the DHCP client service and immediately obtain an IP address.** Doing so will allow other nodes to begin transmitting/receiving packets to/from the external IP network. However, this step is unnecessary if static IP settings are used.

**Writing an ESP-MESH Application** The prerequisites for starting ESP-MESH is to initialize LwIP and Wi-Fi. The following code snippet demonstrates the necessary prerequisite steps before ESP-MESH itself can be initialized.

```
tcpip_adapter_init();
/*
 * for mesh
 * stop DHCP server on softAP interface by default
 * stop DHCP client on station interface by default
 */
ESP_ERROR_CHECK(tcpip_adapter_dhcps_stop(TCPIP_ADAPTER_IF_AP));
ESP_ERROR_CHECK(tcpip_adapter_dhcpc_stop(TCPIP_ADAPTER_IF_STA));

/* event initialization */
ESP_ERROR_CHECK(esp_event_loop_create_default());

/* Wi-Fi initialization */
wifi_init_config_t config = WIFI_INIT_CONFIG_DEFAULT();
ESP_ERROR_CHECK(esp_wifi_init(&config));
/* register IP events handler */
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());
```

After initializing LwIP and Wi-Fi, the process of getting an ESP-MESH network up and running can be summarized into the following three steps:

1. *Initialize Mesh*
2. *Configuring an ESP-MESH Network*
3. *Start Mesh*

**Initialize Mesh** The following code snippet demonstrates how to initialize ESP-MESH

```
/* mesh initialization */
ESP_ERROR_CHECK(esp_mesh_init());
/* register mesh events handler */
ESP_ERROR_CHECK(esp_event_handler_register(MESH_EVENT, ESP_EVENT_ANY_ID, &mesh_
↪event_handler, NULL));
```

**Configuring an ESP-MESH Network** ESP-MESH is configured via `esp_mesh_set_config()` which receives its arguments using the `mesh_cfg_t` structure. The structure contains the following parameters used to configure ESP-MESH:

Parameter	Description
Channel	Range from 1 to 14
Mesh ID	ID of ESP-MESH Network, see <code>mesh_addr_t</code>
Router	Router Configuration, see <code>mesh_router_t</code>
Mesh AP	Mesh AP Configuration, see <code>mesh_ap_cfg_t</code>
Crypto Functions	Crypto Functions for Mesh IE, see <code>mesh_crypto_funcs_t</code>

The following code snippet demonstrates how to configure ESP-MESH.

```

/* Enable the Mesh IE encryption by default */
mesh_cfg_t cfg = MESH_INIT_CONFIG_DEFAULT();
/* mesh ID */
memcpy((uint8_t *) &cfg.mesh_id, MESH_ID, 6);
/* channel (must match the router's channel) */
cfg.channel = CONFIG_MESH_CHANNEL;
/* router */
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));

```

**Start Mesh** The following code snippet demonstrates how to start ESP-MESH.

```

/* mesh start */
ESP_ERROR_CHECK(esp_mesh_start());

```

After starting ESP-MESH, the application should check for ESP-MESH events to determine when it has connected to the network. After connecting, the application can start transmitting and receiving packets over the ESP-MESH network using `esp_mesh_send()` and `esp_mesh_recv()`.

**Self Organized Networking** Self organized networking is a feature of ESP-MESH where nodes can autonomously scan/select/connect/reconnect to other nodes and routers. This feature allows an ESP-MESH network to operate with high degree of autonomy by making the network robust to dynamic network topologies and conditions. With self organized networking enabled, nodes in an ESP-MESH network are able to carryout the following actions without autonomously:

- Selection or election of the root node (see **Automatic Root Node Selection** in *Building a Network*)
- Selection of a preferred parent node (see **Parent Node Selection** in *Building a Network*)
- Automatic reconnection upon detecting a disconnection (see **Intermediate Parent Node Failure** in *Managing a Network*)

When self organized networking is enabled, the ESP-MESH stack will internally make calls to Wi-Fi driver APIs. Therefore, **the application layer should not make any calls to Wi-Fi driver APIs whilst self organized networking is enabled as doing so would risk interfering with ESP-MESH.**

**Toggling Self Organized Networking** Self organized networking can be enabled or disabled by the application at runtime by calling the `esp_mesh_set_self_organized()` function. The function has the two following parameters:

- `bool enable` specifies whether to enable or disable self organized networking.
- `bool select_parent` specifies whether a new parent node should be selected when enabling self organized networking. Selecting a new parent has different effects depending the node type and the node's current state. This parameter is unused when disabling self organized networking.

**Disabling Self Organized Networking** The following code snippet demonstrates how to disable self organized networking.

```

//Disable self organized networking
esp_mesh_set_self_organized(false, false);

```

ESP-MESH will attempt to maintain the node's current Wi-Fi state when disabling self organized networking.

- If the node was previously connected to other nodes, it will remain connected.
- If the node was previously disconnected and was scanning for a parent node or router, it will stop scanning.
- If the node was previously attempting to reconnect to a parent node or router, it will stop reconnecting.

**Enabling Self Organized Networking** ESP-MESH will attempt to maintain the node's current Wi-Fi state when enabling self organized networking. However, depending on the node type and whether a new parent is selected, the Wi-Fi state of the node can change. The following table shows effects of enabling self organized networking.

Select Parent	Is Root Node	Effects
N	N	<ul style="list-style-type: none"> <li>• Nodes already connected to a parent node will remain connected.</li> <li>• Nodes previously scanning for a parent nodes will stop scanning. Call <code>esp_mesh_connect ()</code> to restart.</li> </ul>
	Y	<ul style="list-style-type: none"> <li>• A root node already connected to router will stay connected.</li> <li>• A root node disconnected from router will need to call <code>esp_mesh_connect ()</code> to reconnect.</li> </ul>
Y	N	<ul style="list-style-type: none"> <li>• Nodes without a parent node will automatically select a preferred parent and connect.</li> <li>• Nodes already connected to a parent node will disconnect, reselect a preferred parent node, and connect.</li> </ul>
	Y	<ul style="list-style-type: none"> <li>• For a root node to connect to a parent node, it must give up its role as root. Therefore, a root node will disconnect from the router and all child nodes, select a preferred parent node, and connect.</li> </ul>

The following code snippet demonstrates how to enable self organized networking.

```
//Enable self organized networking and select a new parent
esp_mesh_set_self_organized(true, true);

...

//Enable self organized networking and manually reconnect
esp_mesh_set_self_organized(true, false);
esp_mesh_connect ();
```

**Calling Wi-Fi Driver API** There can be instances in which an application may want to directly call Wi-Fi driver API whilst using ESP-MESH. For example, an application may want to manually scan for neighboring APs. However, **self organized networking must be disabled before the application calls any Wi-Fi driver APIs**. This will prevent the ESP-MESH stack from attempting to call any Wi-Fi driver APIs and potentially interfering with the application's calls.

Therefore, application calls to Wi-Fi driver APIs should be placed in between calls of `esp_mesh_set_self_organized()` which disable and enable self organized networking. The following code snippet demonstrates how an application can safely call `esp_wifi_scan_start()` whilst using ESP-MESH.

```
//Disable self organized networking
esp_mesh_set_self_organized(0, 0);

//Stop any scans already in progress
esp_wifi_scan_stop();
//Manually start scan. Will automatically stop when run to completion
esp_wifi_scan_start();

//Process scan results

...

//Re-enable self organized networking if still connected
esp_mesh_set_self_organized(1, 0);

...

//Re-enable self organized networking if non-root and disconnected
esp_mesh_set_self_organized(1, 1);

...

//Re-enable self organized networking if root and disconnected
esp_mesh_set_self_organized(1, 0); //Don't select new parent
esp_mesh_connect(); //Manually reconnect to router
```

**Application Examples** ESP-IDF contains these ESP-MESH example projects:

[The Internal Communication Example](#) demonstrates how to setup a ESP-MESH network and have the root node send a data packet to every node within the network.

[The Manual Networking Example](#) demonstrates how to use ESP-MESH without the self-organizing features. This example shows how to program a node to manually scan for a list of potential parent nodes and select a parent node based on custom criteria.

## API Reference

### Header File

- [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.

**Return**

- ESP\_OK
- ESP\_FAIL

*esp\_err\_t* **esp\_mesh\_send** (const *mesh\_addr\_t* \*to, const *mesh\_data\_t* \*data, int flag, const *mesh\_opt\_t* 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.

**Return**

- 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

#### Parameters

- [in] `to`: 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.
- [in] `data`: 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).
- [in] `flag`: 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_rcv_toDS()`. Generally, we suggest `esp_mesh_rcv_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_rcv_toDS()` which avoids unnecessary memory copy.
- [in] `opt`: 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.
- [in] `opt_count`: option count
  - Currently, this API only takes one option, so `opt_count` is only supported to be 1.

*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.

#### Return

- ESP\_OK

#### Parameters

- [in] `time_ms`: blocking time of `esp_mesh_send()`, unit:ms

*esp\_err\_t* **esp\_mesh\_rcv** (*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.

#### Return

- ESP\_OK
- ESP\_ERR\_MESH\_ARGUMENT
- ESP\_ERR\_MESH\_NOT\_START

- ESP\_ERR\_MESH\_TIMEOUT
- ESP\_ERR\_MESH\_DISCARD

**Parameters**

- [out] from: the address of the original source of the packet
- [out] data: 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.
- [in] timeout\_ms: wait time if a packet isn't immediately available (0:no wait, port-MAX\_DELAY:wait forever)
- [out] flag: bitmap for data received
  - MESH\_DATA\_FROMDS represents data from external IP network
  - MESH\_DATA\_TODS represents data directed upward within the mesh network

**Parameters**

- [out] opt: options desired to receive
  - MESH\_OPT\_RECV\_DS\_ADDR attaches the DS address
- [in] opt\_count: option count desired to receive
  - Currently, this API only takes one option, so opt\_count is only supported to be 1.

*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.

**Return**

- ESP\_OK
- ESP\_ERR\_MESH\_ARGUMENT
- ESP\_ERR\_MESH\_NOT\_START
- ESP\_ERR\_MESH\_TIMEOUT
- ESP\_ERR\_MESH\_DISCARD

**Parameters**

- [out] from: the address of the original source of the packet
- [out] to: the address contains remote IP address and port (IPv4:PORT)
- [out] data: pointer to the received packet
  - Contain the protocol and applications should follow it to parse the data.
- [in] timeout\_ms: wait time if a packet isn't immediately available (0:no wait, port-MAX\_DELAY:wait forever)
- [out] flag: 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.

**Parameters**

- [out] opt: options desired to receive
- [in] opt\_count: option count desired to receive

*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 de-



fault.

- 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.

**Return**

- ESP\_OK
- ESP\_ERR\_MESH\_ARGUMENT
- ESP\_ERR\_MESH\_NOT\_ALLOWED

**Parameters**

- [in] `config`: pointer to mesh stack configuration

*esp\_err\_t* **esp\_mesh\_get\_config** (*mesh\_cfg\_t* \**config*)

Get mesh stack configuration.

**Return**

- ESP\_OK
- ESP\_ERR\_MESH\_ARGUMENT

**Parameters**

- [out] `config`: pointer to mesh stack configuration

*esp\_err\_t* **esp\_mesh\_set\_router** (**const** *mesh\_router\_t* \**router*)

Get router configuration.

**Attention** This API is used to dynamically modify the router configuration after mesh is configured.

**Return**

- ESP\_OK
- ESP\_ERR\_MESH\_ARGUMENT

**Parameters**

- [in] `router`: pointer to router configuration

*esp\_err\_t* **esp\_mesh\_get\_router** (*mesh\_router\_t* \**router*)

Get router configuration.

**Return**

- ESP\_OK
- ESP\_ERR\_MESH\_ARGUMENT

**Parameters**

- [out] `router`: pointer to router configuration

*esp\_err\_t* **esp\_mesh\_set\_id** (**const** *mesh\_addr\_t* \**id*)

Set mesh network ID.

**Attention** This API is used to dynamically modify the mesh network ID.

**Return**

- ESP\_OK
- ESP\_ERR\_MESH\_ARGUMENT: invalid argument

**Parameters**

- [in] `id`: pointer to mesh network ID

*esp\_err\_t* **esp\_mesh\_get\_id** (*mesh\_addr\_t* \*id)

Get mesh network ID.

**Return**

- ESP\_OK
- ESP\_ERR\_MESH\_ARGUMENT

**Parameters**

- [out] id: pointer to mesh network ID

*esp\_err\_t* **esp\_mesh\_set\_type** (*mesh\_type\_t* type)

Designate device type over the mesh network.

- MESH\_ROOT: designates the root node for a mesh network
- MESH\_LEAF: designates a device as a standalone Wi-Fi station

**Return**

- ESP\_OK
- ESP\_ERR\_MESH\_NOT\_ALLOWED

**Parameters**

- [in] type: device type

*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.

**Return** mesh type

*esp\_err\_t* **esp\_mesh\_set\_max\_layer** (int max\_layer)

Set network max layer value (max:25, default:25)

- Network max layer limits the max hop count.

**Attention** This API shall be called before mesh is started.

**Return**

- ESP\_OK
- ESP\_ERR\_MESH\_ARGUMENT
- ESP\_ERR\_MESH\_NOT\_ALLOWED

**Parameters**

- [in] max\_layer: max layer value

int **esp\_mesh\_get\_max\_layer** (void)

Get max layer value.

**Return** 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.

**Return**

- ESP\_OK
- ESP\_ERR\_MESH\_ARGUMENT
- ESP\_ERR\_MESH\_NOT\_ALLOWED

**Parameters**

- [in] pwd: pointer to the password
- [in] len: password length

*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.

**Return**

- ESP\_OK
- ESP\_ERR\_MESH\_ARGUMENT
- ESP\_ERR\_MESH\_NOT\_ALLOWED

**Parameters**

- [in] `authmode`: authentication mode

*wifi\_auth\_mode\_t* **esp\_mesh\_get\_ap\_authmode** (void)

Get mesh softAP authentication mode.

**Return** authentication mode

*esp\_err\_t* **esp\_mesh\_set\_ap\_connections** (int *connections*)

Set mesh softAP max connection value.

**Attention** This API shall be called before mesh is started.

**Return**

- ESP\_OK
- ESP\_ERR\_MESH\_ARGUMENT

**Parameters**

- [in] `connections`: the number of max connections

int **esp\_mesh\_get\_ap\_connections** (void)

Get mesh softAP max connection configuration.

**Return** the number of 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.

**Return** 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.

**Return**

- ESP\_OK
- ESP\_FAIL

**Parameters**

- [out] `bssid`: pointer to parent BSSID

bool **esp\_mesh\_is\_root** (void)

Return whether the device is the root node of the network.

**Return** 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.

**Return**

- ESP\_OK
- ESP\_FAIL

**Parameters**

- [in] `enable`: enable or disable self-organized networking
- [in] `select_parent`: 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.

bool **esp\_mesh\_get\_self\_organized** (void)

Return whether enable self-organized networking or not.

**Return** 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.

**Return**

- ESP\_OK
- ESP\_ERR\_MESH\_QUEUE\_FULL
- ESP\_ERR\_MESH\_DISCARD
- ESP\_FAIL

**Parameters**

- [in] `vote`: 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.
- [in] `reason`: only accept MESH\_VOTE\_REASON\_ROOT\_INITIATED for now

`esp_err_t esp_mesh_set_vote_percentage (float percentage)`

Set vote percentage threshold for approval of being a root.

- 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.

**Return**

- ESP\_OK
- ESP\_FAIL

**Parameters**

- [in] `percentage`: vote percentage threshold

float `esp_mesh_get_vote_percentage (void)`

Get vote percentage threshold for approval of being a root.

**Return** 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.

**Return**

- ESP\_OK
- ESP\_FAIL

**Parameters**

- [in] `seconds`: the expired time

int `esp_mesh_get_ap_assoc_expire (void)`

Get mesh softAP associate expired time.

**Return** 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.

**Return** 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)

**Return** 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)

**Return**

- ESP\_OK
- ESP\_ERR\_MESH\_ARGUMENT

**Parameters**

- [out] mac: pointer to routing table
- [in] len: routing table size(in bytes)
- [out] size: pointer to the number of devices in routing table (including itself)

*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.

**Return**

- ESP\_OK
- ESP\_FAIL

**Parameters**

- [in] reachable: this state represents whether the root is able to access external IP network

*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.

**Return**

- ESP\_OK
- ESP\_FAIL

**Parameters**

- [out] pending: pointer to the TX pending

*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.

**Return**

- ESP\_OK
- ESP\_FAIL

**Parameters**

- [out] pending: pointer to the RX pending

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.

**Return** the number of upQ for a certain address

**Parameters**

- [in] addr: self address or an associate children address
- [out] xseqno\_in: sequence number of the last received packet from the specified address

*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.

**Return**

- ESP\_OK

- ESP\_FAIL

**Parameters**

- [in] qsize: default:32 (min:16)

int **esp\_mesh\_get\_xon\_qsize** (void)

Get queue size.

**Return** the number of queue

*esp\_err\_t* **esp\_mesh\_allow\_root\_conflicts** (bool *allowed*)

Set whether allow more than one root existing in one network.

**Return**

- ESP\_OK
- ESP\_WIFI\_ERR\_NOT\_INIT
- ESP\_WIFI\_ERR\_NOT\_START

**Parameters**

- [in] allowed: allow or not

bool **esp\_mesh\_is\_root\_conflicts\_allowed** (void)

Check whether allow more than one root to exist in one network.

**Return** true/false

*esp\_err\_t* **esp\_mesh\_set\_group\_id** (const *mesh\_addr\_t* \**addr*, int *num*)

Set group ID addresses.

**Return**

- ESP\_OK
- ESP\_MESH\_ERR\_ARGUMENT

**Parameters**

- [in] addr: pointer to new group ID addresses
- [in] num: the number of group ID addresses

*esp\_err\_t* **esp\_mesh\_delete\_group\_id** (const *mesh\_addr\_t* \**addr*, int *num*)

Delete group ID addresses.

**Return**

- ESP\_OK
- ESP\_MESH\_ERR\_ARGUMENT

**Parameters**

- [in] addr: pointer to deleted group ID address
- [in] num: the number of group ID addresses

int **esp\_mesh\_get\_group\_num** (void)

Get the number of group ID addresses.

**Return** the number of group ID addresses

*esp\_err\_t* **esp\_mesh\_get\_group\_list** (*mesh\_addr\_t* \**addr*, int *num*)

Get group ID addresses.

**Return**

- ESP\_OK
- ESP\_MESH\_ERR\_ARGUMENT

**Parameters**

- [out] addr: pointer to group ID addresses
- [in] num: the number of group ID addresses

bool **esp\_mesh\_is\_my\_group** (const *mesh\_addr\_t* \**addr*)

Check whether the specified group address is my group.

**Return** 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.

**Return**

- ESP\_OK
- ESP\_ERR\_MESH\_NOT\_ALLOWED
- ESP\_MESH\_ERR\_ARGUMENT

**Parameters**

- [in] num: mesh network capacity

int **esp\_mesh\_get\_capacity\_num** (void)

Get mesh network capacity.

**Return** 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.

**Return**

- ESP\_OK

**Parameters**

- [in] crypto\_funcs: crypto functions for mesh IE
  - If crypto\_funcs is set to NULL, mesh IE is no longer encrypted.

*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.

**Return**

- ESP\_OK
- ESP\_MESH\_ERR\_ARGUMENT

**Parameters**

- [in] key: ASCII crypto key
- [in] len: length in bytes, range:8-64

*esp\_err\_t* **esp\_mesh\_get\_ie\_crypto\_key** (char \*key, int len)

Get mesh IE crypto key.

**Return**

- ESP\_OK
- ESP\_MESH\_ERR\_ARGUMENT

**Parameters**

- [out] key: ASCII crypto key
- [in] len: length in bytes, range:8-64

*esp\_err\_t* **esp\_mesh\_set\_root\_healing\_delay** (int delay\_ms)

Set delay time before starting root healing.

**Return**

- ESP\_OK

**Parameters**

- [in] delay\_ms: delay time in milliseconds

int **esp\_mesh\_get\_root\_healing\_delay** (void)

Get delay time before network starts root healing.

**Return** delay time in milliseconds

*esp\_err\_t* **esp\_mesh\_fix\_root** (bool enable)

Enable network Fixed Root Setting.

- Enabling fixed root disables 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.

**Return**

- ESP\_OK

**Parameters**

- [in] enable: enable or not

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.

**Return** 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.

**Return**

- ESP\_OK
- ESP\_ERR\_ARGUMENT
- ESP\_ERR\_MESH\_NOT\_CONFIG

**Parameters**

- [in] parent: 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.
- [in] parent\_mesh\_id: parent mesh ID,
  - If this value is not set, the original mesh ID is used.
- [in] my\_type: mesh type
  - 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.
- [in] my\_layer: 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.

*esp\_err\_t* **esp\_mesh\_scan\_get\_ap\_ie\_len** (int \*len)

Get mesh networking IE length of one AP.

**Return**

- ESP\_OK
- ESP\_ERR\_WIFI\_NOT\_INIT
- ESP\_ERR\_WIFI\_ARG
- ESP\_ERR\_WIFI\_FAIL

**Parameters**

- [out] len: mesh networking IE length

*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.

**Return**

- ESP\_OK
- ESP\_ERR\_WIFI\_NOT\_INIT
- ESP\_ERR\_WIFI\_ARG
- ESP\_ERR\_WIFI\_FAIL

**Parameters**

- [out] ap\_record: pointer to one AP record
- [out] buffer: pointer to the mesh networking IE of this AP

*esp\_err\_t* **esp\_mesh\_flush\_upstream\_packets** (void)

Flush upstream packets pending in to\_parent queue and to\_parent\_p2p queue.

**Return**



- ESP\_OK

*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.

**Return**

- ESP\_OK
- ESP\_ERR\_MESH\_NOT\_START
- ESP\_ERR\_MESH\_ARGUMENT

**Parameters**

- [in] child\_mac: an associated child address of this device
- [out] nodes\_num: pointer to the number of nodes in the subnet of a specific child

*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.

**Return**

- ESP\_OK
- ESP\_ERR\_MESH\_NOT\_START
- ESP\_ERR\_MESH\_ARGUMENT

**Parameters**

- [in] child\_mac: an associated child address of this device
- [out] nodes: pointer to nodes in the subnet of a specific child
- [in] nodes\_num: the number of nodes in the subnet of a specific child

*esp\_err\_t* **esp\_mesh\_disconnect** (void)

Disconnect from current parent.

**Return**

- ESP\_OK

*esp\_err\_t* **esp\_mesh\_connect** (void)

Connect to current parent.

**Return**

- ESP\_OK

*esp\_err\_t* **esp\_mesh\_flush\_scan\_result** (void)

Flush scan result.

**Return**

- ESP\_OK

*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.

**Return**

- ESP\_OK

**Parameters**

- [in] new\_bssid: the new router BSSID if the router changes
- [in] csa\_newchan: the new channel number to which the whole network is moving
- [in] csa\_count: channel switch period (beacon count), unit is based on beacon interval of its softAP, the default value is 15.

*esp\_err\_t* **esp\_mesh\_get\_router\_bssid** (uint8\_t \*router\_bssid)

Get the router BSSID.

**Return**

- ESP\_OK
- ESP\_ERR\_WIFI\_NOT\_INIT

- ESP\_ERR\_WIFI\_ARG

**Parameters**

- [out] router\_bssid: pointer to the router BSSID

int64\_t **esp\_mesh\_get\_tsf\_time** (void)

Get the TSF time.

**Return** the TSF time

**Unions**

**union mesh\_addr\_t**

*#include <esp\_mesh.h>* Mesh address.

**Public Members**

uint8\_t **addr**[6]  
mac address

*mip\_t* **mip**  
mip address

**union mesh\_event\_info\_t**

*#include <esp\_mesh.h>* Mesh event information.

**Public Members**

*mesh\_event\_channel\_switch\_t* **channel\_switch**  
channel switch

*mesh\_event\_child\_connected\_t* **child\_connected**  
child connected

*mesh\_event\_child\_disconnected\_t* **child\_disconnected**  
child disconnected

*mesh\_event\_routing\_table\_change\_t* **routing\_table**  
routing table change

*mesh\_event\_connected\_t* **connected**  
parent connected

*mesh\_event\_disconnected\_t* **disconnected**  
parent disconnected

*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\_got\_ip\_t* **got\_ip**  
root obtains IP address

*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

**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

*system\_event\_sta\_connected\_t* **connected**  
parent information, same as Wi-Fi event `SYSTEM_EVENT_STA_CONNECTED` does

uint8\_t **self\_layer**  
layer

**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

uint8\_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 requester

**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_opt_t`  
Mesh option.

**Public Members**

`uint8_t type`  
option type

`uint16_t len`  
option length

`uint8_t *val`  
option value

**struct mesh\_data\_t**

Mesh data for esp\_mesh\_send() and esp\_mesh\_recv()

**Public Members****uint8\_t \*data**

data

**uint16\_t size**

data size

**mesh\_proto\_t proto**

data protocol

**mesh\_tos\_t tos**

data type of service

**struct mesh\_router\_t**

Router configuration.

**Public Members****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.

**struct mesh\_ap\_cfg\_t**

Mesh softAP configuration.

**Public Members****uint8\_t password[64]**

mesh softAP password

**uint8\_t max\_connection**

max number of stations allowed to connect in, max 10

**struct mesh\_cfg\_t**

Mesh initialization configuration.

**Public Members****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

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

**MESH\_DATA\_ENC**

Flags bitmap for esp\_mesh\_send() and esp\_mesh\_rcv()

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\_rcv()

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\_rcv()

**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\_INIT\_CONFIG\_DEFAULT ( )****Type Definitions**

```
typedef system_event_sta_got_ip_t mesh_event_root_got_ip_t  
IP settings from LwIP stack.
```

```
typedef mesh_addr_t mesh_event_root_address_t  
Root address.
```

```
typedef system_event_sta_disconnected_t mesh_event_disconnected_t  
Parent disconnected information.
```

**typedef** *system\_event\_ap\_staconnected\_t* **mesh\_event\_child\_connected\_t**  
Child connected information.

**typedef** *system\_event\_ap\_stadisconnected\_t* **mesh\_event\_child\_disconnected\_t**  
Child disconnected information.

**typedef** *system\_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:*

**MESH\_EVENT\_STARTED**

mesh is started

**MESH\_EVENT\_STOPPED**

mesh is stopped

**MESH\_EVENT\_CHANNEL\_SWITCH**

channel switch

**MESH\_EVENT\_CHILD\_CONNECTED**

a child is connected on softAP interface

**MESH\_EVENT\_CHILD\_DISCONNECTED**

a child is disconnected on softAP interface

**MESH\_EVENT\_ROUTING\_TABLE\_ADD**

routing table is changed by adding newly joined children

**MESH\_EVENT\_ROUTING\_TABLE\_REMOVE**

routing table is changed by removing leave children

**MESH\_EVENT\_PARENT\_CONNECTED**

parent is connected on station interface

**MESH\_EVENT\_PARENT\_DISCONNECTED**

parent is disconnected on station interface

**MESH\_EVENT\_NO\_PARENT\_FOUND**

no parent found

**MESH\_EVENT\_LAYER\_CHANGE**

layer changes over the mesh network

**MESH\_EVENT\_TODS\_STATE**

state represents whether the root is able to access external IP network

**MESH\_EVENT\_VOTE\_STARTED**

the process of voting a new root is started either by children or by the root

**MESH\_EVENT\_VOTE\_STOPPED**

the process of voting a new root is stopped

**MESH\_EVENT\_ROOT\_ADDRESS**

the root address is obtained. It is posted by mesh stack automatically.

**MESH\_EVENT\_ROOT\_SWITCH\_REQ**

root switch request sent from a new voted root candidate

**MESH\_EVENT\_ROOT\_SWITCH\_ACK**

root switch acknowledgment responds the above request sent from current root

**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.

**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.

**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.

**MESH\_EVENT\_NETWORK\_STATE**

network state, such as whether current mesh network has a root.

**MESH\_EVENT\_STOP\_RECONNECTION**

the root stops reconnecting to the router and non-root devices stop reconnecting to their parents.

**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.

**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.

**MESH\_EVENT\_MAX**

**enum mesh\_type\_t**

Device type.

*Values:*

**MESH\_IDLE**

hasn't joined the mesh network yet

**MESH\_ROOT**

the only sink of the mesh network. Has the ability to access external IP network

**MESH\_NODE**

intermediate device. Has the ability to forward packets over the mesh network

**MESH\_LEAF**

has no forwarding ability

**enum mesh\_proto\_t**

Protocol of transmitted application data.

*Values:*

**MESH\_PROTO\_BIN**

binary

**MESH\_PROTO\_HTTP**

HTTP protocol

**MESH\_PROTO\_JSON**

JSON format

**MESH\_PROTO\_MQTT**

MQTT protocol

**enum mesh\_tos\_t**

For reliable transmission, mesh stack provides three type of services.

*Values:*

**MESH\_TOS\_P2P**

provide P2P (point-to-point) retransmission on mesh stack by default

**MESH\_TOS\_E2E**

provide E2E (end-to-end) retransmission on mesh stack (Unimplemented)

**MESH\_TOS\_DEF**

no retransmission on mesh stack

**enum mesh\_vote\_reason\_t**

Vote reason.

*Values:***MESH\_VOTE\_REASON\_ROOT\_INITIATED = 1**

vote is initiated by the root

**MESH\_VOTE\_REASON\_CHILD\_INITIATED**

vote is initiated by children

**enum mesh\_disconnect\_reason\_t**

Mesh disconnect reason code.

*Values:***MESH\_REASON\_CYCLIC = 100**

cyclic is detected

**MESH\_REASON\_PARENT\_IDLE**

parent is idle

**MESH\_REASON\_LEAF**

the connected device is changed to a leaf

**MESH\_REASON\_DIFF\_ID**

in different mesh ID

**MESH\_REASON\_ROOTS**

root conflict is detected

**MESH\_REASON\_PARENT\_STOPPED**

parent has stopped the mesh

**MESH\_REASON\_SCAN\_FAIL**

scan fail

**MESH\_REASON\_IE\_UNKNOWN**

unknown IE

**MESH\_REASON\_WAIVE\_ROOT**

waive root

**MESH\_REASON\_PARENT\_WORSE**

parent with very poor RSSI

**MESH\_REASON\_EMPTY\_PASSWORD**

use an empty password to connect to an encrypted parent

**MESH\_REASON\_PARENT\_UNENCRYPTED**

connect to an unencrypted parent/router

**enum mesh\_event\_toDS\_state\_t**

The reachability of the root to a DS (distribute system)

*Values:***MESH\_TODS\_UNREACHABLE**

the root isn't able to access external IP network

**MESH\_TODS\_REACHABLE**

the root is able to access external IP network

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-MESH 的示例代码存放在 ESP-IDF 示例项目的 [mesh](#) 目录下。

## 2.2.2 以太网

### Ethernet

#### Application Example

- Ethernet basic example: [ethernet/ethernet](#).
- Ethernet iperf example: [ethernet/iperf](#).

#### Ethernet Driver Model

- [esp\\_eth/include/esp\\_eth.h](#)

#### Ethernet Common Interface

- [esp\\_eth/include/esp\\_eth\\_com.h](#)

#### Ethernet MAC Interface

- [esp\\_eth/include/esp\\_eth\\_mac.h](#)

#### Ethernet PHY Interface

- [esp\\_eth/include/esp\\_eth\\_phy.h](#)

#### Ethernet PHY Common Registers

- [esp\\_eth/include/eth\\_phy\\_regs\\_struct.h](#)

### API Reference - Driver Model

#### Header File

- [esp\\_eth/include/esp\\_eth.h](#)

#### Functions

*esp\_err\_t* **esp\_eth\_driver\_install** (*const esp\_eth\_config\_t* \*config, *esp\_eth\_handle\_t* \*out\_hdl)

Install Ethernet driver.

#### Return

- ESP\_OK: install esp\_eth driver successfully
- ESP\_ERR\_INVALID\_ARG: install esp\_eth driver failed because of some invalid argument
- ESP\_ERR\_NO\_MEM: install esp\_eth driver failed because there's no memory for driver
- ESP\_FAIL: install esp\_eth driver failed because some other error occurred

#### Parameters

- [in] config: configuration of the Ethernet driver
- [out] out\_hdl: handle of Ethernet driver

*esp\_err\_t* **esp\_eth\_driver\_uninstall** (*esp\_eth\_handle\_t* hdl)

Uninstall Ethernet driver.

**Note** 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.

**Return**

- 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

**Parameters**

- [in] hdl: handle of Ethernet driver

*esp\_err\_t* **esp\_eth\_start** (*esp\_eth\_handle\_t* hdl)

Start Ethernet driver.

**Note** This API will start driver state machine and internal software timer (for checking link status).

**Return**

- 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

**Parameters**

- [in] hdl: handle of Ethernet driver

*esp\_err\_t* **esp\_eth\_stop** (*esp\_eth\_handle\_t* hdl)

Stop Ethernet driver.

**Note** This function does the oppsite operation of esp\_eth\_start.

**Return**

- 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

**Parameters**

- [in] hdl: handle of Ethernet driver

*esp\_err\_t* **esp\_eth\_transmit** (*esp\_eth\_handle\_t* hdl, *uint8\_t \*buf*, *uint32\_t length*)

General Transmit.

**Return**

- ESP\_OK: transmit frame buffer successfully
- ESP\_ERR\_INVALID\_ARG: transmit frame buffer failed because of some invalid argument
- ESP\_FAIL: transmit frame buffer failed because some other error occurred

**Parameters**

- [in] hdl: handle of Ethernet driver
- [in] buf: buffer of the packet to transfer
- [in] length: length of the buffer to transfer

*esp\_err\_t* **esp\_eth\_receive** (*esp\_eth\_handle\_t* hdl, *uint8\_t \*buf*, *uint32\_t \*length*)

General Receive.

**Return**

- ESP\_OK: receive frame buffer successfully
- ESP\_ERR\_INVALID\_ARG: receive frame buffer failed because of some invalid argument
- ESP\_FAIL: receive frame buffer failed because some other error occurred

**Parameters**

- [in] hdl: handle of Ethernet driver
- [out] buf: buffer to preserve the received packet
- [out] length: length of the received packet

*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.

**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

**Parameters**

- [in] hdl: handle of Ethernet driver
- [in] cmd: IO control command
- [in] data: specified data for command

*esp\_err\_t* **esp\_eth\_increase\_reference** (*esp\_eth\_handle\_t* hdl)

Increase Ethernet driver reference.

**Note** Ethernet driver handle can be obtained by os timer, netif, etc. It's dangerous when thread A is using Ethernet but thread B uninstalls 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.

**Return**

- ESP\_OK: increase reference successfully
- ESP\_ERR\_INVALID\_ARG: increase reference failed because of some invalid argument

**Parameters**

- [in] hdl: handle of Ethernet driver

*esp\_err\_t* **esp\_eth\_decrease\_reference** (*esp\_eth\_handle\_t* hdl)

Decrease Ethernet driver reference.

**Return**

- ESP\_OK: increase reference successfully
- ESP\_ERR\_INVALID\_ARG: increase reference failed because of some invalid argument

**Parameters**

- [in] hdl: handle of Ethernet driver

**Structures**

**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)

Input frame buffer to user's stack.

**Return**

- ESP\_OK: input frame buffer to upper stack successfully
- ESP\_FAIL: error occurred when inputting buffer to upper stack

**Parameters**

- [in] eth\_handle: handle of Ethernet driver
- [in] buffer: frame buffer that will get input to upper stack
- [in] length: length of the frame buffer

*esp\_err\_t* (**\*on\_lowlevel\_init\_done**) (*esp\_eth\_handle\_t* eth\_handle)

Callback function invoked when lowlevel initialization is finished.

**Return**

- ESP\_OK: process extra lowlevel initialization successfully

- ESP\_FAIL: error occurred when processing extra lowlevel initialization

**Parameters**

- [in] eth\_handle: handle of Ethernet driver

*esp\_err\_t* (\***on\_lowlevel\_deinit\_done**)(*esp\_eth\_handle\_t* eth\_handle)

Callback function invoked when lowlevel deinitialization is finished.

**Return**

- ESP\_OK: process extra lowlevel deinitialization successfully
- ESP\_FAIL: error occurred when processing extra lowlevel deinitialization

**Parameters**

- [in] eth\_handle: handle of Ethernet driver

**Macros**

**ETH\_DEFAULT\_CONFIG** (emac, ephy)

Default configuration for Ethernet driver.

**Type Definitions**

**typedef** void \***esp\_eth\_handle\_t**

Handle of Ethernet driver.

**API Reference - Common Interface****Header File**

- [esp\\_eth/include/esp\\_eth\\_com.h](#)

**Structures**

**struct** **esp\_eth\_mediator\_s**

Ethernet mediator.

**Public Members**

*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.

**Return**

- ESP\_OK: read PHY register successfully
- ESP\_FAIL: read PHY register failed because some error occurred

**Parameters**

- [in] eth: mediator of Ethernet driver
- [in] phy\_addr: PHY Chip address (0~31)
- [in] phy\_reg: PHY register index code
- [out] reg\_value: PHY register value

*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.

**Return**

- ESP\_OK: write PHY register successfully
- ESP\_FAIL: write PHY register failed because some error occurred

**Parameters**

- [in] eth: mediator of Ethernet driver
- [in] phy\_addr: PHY Chip address (0~31)
- [in] phy\_reg: PHY register index code



- [in] `reg_value`: PHY register value

`esp_err_t` (**\*stack\_input**) (`esp_eth_mediator_t` \*eth, uint8\_t \*buffer, uint32\_t length)  
Deliver packet to upper stack.

#### Return

- ESP\_OK: deliver packet to upper stack successfully
- ESP\_FAIL: deliver packet failed because some error occurred

#### Parameters

- [in] `eth`: mediator of Ethernet driver
- [in] `buffer`: packet buffer
- [in] `length`: length of the packet

`esp_err_t` (**\*on\_state\_changed**) (`esp_eth_mediator_t` \*eth, `esp_eth_state_t` state, void \*args)  
Callback on Ethernet state changed.

#### Return

- ESP\_OK: process the new state successfully
- ESP\_FAIL: process the new state failed because some error occurred

#### Parameters

- [in] `eth`: mediator of Ethernet driver
- [in] `state`: new state
- [in] `args`: optional argument for the new state

## Macros

### ETH\_MAX\_PAYLOAD\_LEN

Maximum Ethernet payload size.

### ETH\_MIN\_PAYLOAD\_LEN

Minimum Ethernet payload size.

### ETH\_HEADER\_LEN

Ethernet frame header size: Dest addr(6 Bytes) + Src addr(6 Bytes) + length/type(2 Bytes)

### ETH\_CRC\_LEN

Ethernet frame CRC length.

### ETH\_VLAN\_TAG\_LEN

Optional 802.1q VLAN Tag length.

### ETH\_JUMBO\_FRAME\_PAYLOAD\_LEN

Jumbo frame payload size.

### ETH\_MAX\_PACKET\_SIZE

Maximum frame size (1522 Bytes)

### ETH\_MIN\_PACKET\_SIZE

Minimum frame size (64 Bytes)

## Type Definitions

`typedef struct esp_eth_mediator_s esp_eth_mediator_t`  
Ethernet mediator.

## Enumerations

### enum esp\_eth\_state\_t

Ethernet driver state.

Values:

#### ETH\_STATE\_LLINIT

Lowlevel init done

#### ETH\_STATE\_DEINIT

Deinit done

**ETH\_STATE\_LINK**  
Link status changed

**ETH\_STATE\_SPEED**  
Speed updated

**ETH\_STATE\_DUPLEX**  
Duplex updated

**enum esp\_eth\_io\_cmd\_t**  
Command list for ioctl API.

*Values:*

**ETH\_CMD\_G\_MAC\_ADDR**  
Get MAC address

**ETH\_CMD\_S\_MAC\_ADDR**  
Set MAC address

**ETH\_CMD\_G\_PHY\_ADDR**  
Get PHY address

**ETH\_CMD\_S\_PHY\_ADDR**  
Set PHY address

**ETH\_CMD\_G\_SPEED**  
Get Speed

**ETH\_CMD\_S\_PROMISCUOUS**  
Set promiscuous mode

**enum eth\_link\_t**  
Ethernet link status.

*Values:*

**ETH\_LINK\_UP**  
Ethernet link is up

**ETH\_LINK\_DOWN**  
Ethernet link is down

**enum eth\_speed\_t**  
Ethernet speed.

*Values:*

**ETH\_SPEED\_10M**  
Ethernet speed is 10Mbps

**ETH\_SPEED\_100M**  
Ethernet speed is 100Mbps

**enum eth\_duplex\_t**  
Ethernet duplex mode.

*Values:*

**ETH\_DUPLEX\_HALF**  
Ethernet is in half duplex

**ETH\_DUPLEX\_FULL**  
Ethernet is in full duplex

**enum eth\_event\_t**  
Ethernet event declarations.

*Values:*

**ETHERNET\_EVENT\_START**

Ethernet driver start

**ETHERNET\_EVENT\_STOP**

Ethernet driver stop

**ETHERNET\_EVENT\_CONNECTED**

Ethernet got a valid link

**ETHERNET\_EVENT\_DISCONNECTED**

Ethernet lost a valid link

**API Reference - MAC Interface****Header File**

- [esp\\_eth/include/esp\\_eth\\_mac.h](#)

**Structures****struct esp\_eth\_mac\_s**

Ethernet MAC.

**Public Members***esp\_err\_t* (**\*set\_mediator**) (*esp\_eth\_mac\_t* \*mac, *esp\_eth\_mediator\_t* \*eth)

Set mediator for Ethernet MAC.

**Return**

- ESP\_OK: set mediator for Ethernet MAC successfully
- ESP\_ERR\_INVALID\_ARG: set mediator for Ethernet MAC failed because of invalid argument

**Parameters**

- [in] mac: Ethernet MAC instance
- [in] eth: Ethernet mediator

*esp\_err\_t* (**\*init**) (*esp\_eth\_mac\_t* \*mac)

Initialize Ethernet MAC.

**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

**Parameters**

- [in] mac: Ethernet MAC instance

*esp\_err\_t* (**\*deinit**) (*esp\_eth\_mac\_t* \*mac)

Deinitialize Ethernet MAC.

**Return**

- ESP\_OK: deinitialize Ethernet MAC successfully
- ESP\_FAIL: deinitialize Ethernet MAC failed because some error occurred

**Parameters**

- [in] mac: Ethernet MAC instance

*esp\_err\_t* (**\*start**) (*esp\_eth\_mac\_t* \*mac)

Start Ethernet MAC.

**Return**

- ESP\_OK: start Ethernet MAC successfully
- ESP\_FAIL: start Ethernet MAC failed because some other error occurred

**Parameters**

- [in] mac: Ethernet MAC instance

*esp\_err\_t* (\***stop**) (*esp\_eth\_mac\_t* \*mac)

Stop Ethernet MAC.

#### Return

- ESP\_OK: stop Ethernet MAC successfully
- ESP\_FAIL: stop Ethernet MAC failed because some error occurred

#### Parameters

- [in] mac: Ethernet MAC instance

*esp\_err\_t* (\***transmit**) (*esp\_eth\_mac\_t* \*mac, uint8\_t \*buf, uint32\_t length)

Transmit packet from Ethernet MAC.

#### Return

- ESP\_OK: transmit packet successfully
- ESP\_ERR\_INVALID\_ARG: transmit packet failed because of invalid argument
- ESP\_ERR\_INVALID\_STATE: transmit packet failed because of wrong state of MAC
- ESP\_FAIL: transmit packet failed because some other error occurred

#### Parameters

- [in] mac: Ethernet MAC instance
- [in] buf: packet buffer to transmit
- [in] length: length of packet

*esp\_err\_t* (\***receive**) (*esp\_eth\_mac\_t* \*mac, uint8\_t \*buf, uint32\_t \*length)

Receive packet from Ethernet MAC.

**Note** Memory of buf is allocated in the Layer2, make sure it get free after process.

#### Return

- ESP\_OK: receive packet successfully
- ESP\_ERR\_INVALID\_ARG: receive packet failed because of invalid argument
- ESP\_FAIL: receive packet failed because some other error occurred

#### Parameters

- [in] mac: Ethernet MAC instance
- [out] buf: packet buffer which will preserve the received frame
- [out] length: length of the received packet

*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.

#### 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

#### Parameters

- [in] mac: Ethernet MAC instance
- [in] phy\_addr: PHY chip address (0~31)
- [in] phy\_reg: PHY register index code
- [out] reg\_value: PHY register value

*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.

#### 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

#### Parameters

- [in] mac: Ethernet MAC instance
- [in] phy\_addr: PHY chip address (0~31)
- [in] phy\_reg: PHY register index code
- [in] reg\_value: PHY register value

*esp\_err\_t* (\***set\_addr**) (*esp\_eth\_mac\_t* \*mac, uint8\_t \*addr)  
Set 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

#### Parameters

- [in] mac: Ethernet MAC instance
- [in] addr: MAC address

*esp\_err\_t* (\***get\_addr**) (*esp\_eth\_mac\_t* \*mac, uint8\_t \*addr)  
Get 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

#### Parameters

- [in] mac: Ethernet MAC instance
- [out] addr: MAC address

*esp\_err\_t* (\***set\_speed**) (*esp\_eth\_mac\_t* \*mac, *eth\_speed\_t* speed)  
Set speed of MAC.

#### 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

#### Parameters

- [in] mac: Ethernet MAC instance
- [in] speed: MAC speed

*esp\_err\_t* (\***set\_duplex**) (*esp\_eth\_mac\_t* \*mac, *eth\_duplex\_t* duplex)  
Set duplex mode of MAC.

#### 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

#### Parameters

- [in] mac: Ethernet MAC instance
- [in] duplex: MAC duplex

*esp\_err\_t* (\***set\_link**) (*esp\_eth\_mac\_t* \*mac, *eth\_link\_t* link)  
Set link status of MAC.

#### 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

#### Parameters

- [in] mac: Ethernet MAC instance
- [in] link: Link status

*esp\_err\_t* (\***set\_promiscuous**) (*esp\_eth\_mac\_t* \*mac, bool enable)  
Set promiscuous of MAC.

#### Return

- ESP\_OK: set promiscuous mode successfully
- ESP\_FAIL: set promiscuous mode failed because some error occurred

**Parameters**

- [in] mac: Ethernet MAC instance
- [in] enable: set true to enable promiscuous mode; set false to disable promiscuous mode

*esp\_err\_t* (\*del) (*esp\_eth\_mac\_t* \*mac)

Free memory of Ethernet MAC.

**Return**

- ESP\_OK: free Ethernet MAC instance successfully
- ESP\_FAIL: free Ethernet MAC instance failed because some error occurred

**Parameters**

- [in] mac: Ethernet MAC instance

**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

int **smi\_mdc\_gpio\_num**

SMI MDC GPIO number

int **smi\_mdio\_gpio\_num**

SMI MDIO GPIO number

uint32\_t **flags**

Flags that specify extra capability for mac driver

**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.

**Type Definitions**

**typedef struct esp\_eth\_mac\_s esp\_eth\_mac\_t**

Ethernet MAC.

**API Reference - PHY Interface****Header File**

- [esp\\_eth/include/esp\\_eth\\_phy.h](#)

**Functions**

*esp\_eth\_phy\_t* \***esp\_eth\_phy\_new\_ip101** (**const** *eth\_phy\_config\_t* \**config*)

Create a PHY instance of IP101.

**Return**

- instance: create PHY instance successfully
- NULL: create PHY instance failed because some error occurred

**Parameters**

- [in] *config*: configuration of PHY

*esp\_eth\_phy\_t* \***esp\_eth\_phy\_new\_rt18201** (**const** *eth\_phy\_config\_t* \**config*)

Create a PHY instance of RTL8201.

**Return**

- instance: create PHY instance successfully
- NULL: create PHY instance failed because some error occurred

**Parameters**

- [in] *config*: configuration of PHY

*esp\_eth\_phy\_t* \***esp\_eth\_phy\_new\_lan8720** (**const** *eth\_phy\_config\_t* \**config*)

Create a PHY instance of LAN8720.

**Return**

- instance: create PHY instance successfully
- NULL: create PHY instance failed because some error occurred

**Parameters**

- [in] *config*: configuration of PHY

*esp\_eth\_phy\_t* \***esp\_eth\_phy\_new\_dp83848** (**const** *eth\_phy\_config\_t* \**config*)

Create a PHY instance of DP83848.

**Return**

- instance: create PHY instance successfully
- NULL: create PHY instance failed because some error occurred

**Parameters**

- [in] *config*: configuration of PHY

**Structures**

**struct esp\_eth\_phy\_s**

Ethernet PHY.

**Public Members**

*esp\_err\_t* (\***set\_mediator**) (*esp\_eth\_phy\_t* \**phy*, *esp\_eth\_mediator\_t* \**mediator*)

Set mediator for PHY.

**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

**Parameters**

- [in] *phy*: Ethernet PHY instance
- [in] *mediator*: mediator of Ethernet driver

*esp\_err\_t* (\***reset**) (*esp\_eth\_phy\_t* \**phy*)

Software Reset Ethernet PHY.

**Return**

- ESP\_OK: reset Ethernet PHY successfully
- ESP\_FAIL: reset Ethernet PHY failed because some error occurred

**Parameters**

- [in] *phy*: Ethernet PHY instance

*esp\_err\_t* (\***reset\_hw**)(*esp\_eth\_phy\_t* \*phy)

Hardware Reset Ethernet PHY.

**Note** Hardware reset is mostly done by pull down and up PHY's nRST pin

**Return**

- ESP\_OK: reset Ethernet PHY successfully
- ESP\_FAIL: reset Ethernet PHY failed because some error occurred

**Parameters**

- [in] phy: Ethernet PHY instance

*esp\_err\_t* (\***init**)(*esp\_eth\_phy\_t* \*phy)

Initialize Ethernet PHY.

**Return**

- ESP\_OK: initialize Ethernet PHY successfully
- ESP\_FAIL: initialize Ethernet PHY failed because some error occurred

**Parameters**

- [in] phy: Ethernet PHY instance

*esp\_err\_t* (\***deinit**)(*esp\_eth\_phy\_t* \*phy)

Deinitialize Ethernet PHY.

**Return**

- ESP\_OK: deinitialize Ethernet PHY successfully
- ESP\_FAIL: deinitialize Ethernet PHY failed because some error occurred

**Parameters**

- [in] phy: Ethernet PHY instance

*esp\_err\_t* (\***negotiate**)(*esp\_eth\_phy\_t* \*phy)

Start auto negotiation.

**Return**

- ESP\_OK: restart auto negotiation successfully
- ESP\_FAIL: restart auto negotiation failed because some error occurred

**Parameters**

- [in] phy: Ethernet PHY instance

*esp\_err\_t* (\***get\_link**)(*esp\_eth\_phy\_t* \*phy)

Get Ethernet PHY link status.

**Return**

- ESP\_OK: get Ethernet PHY link status successfully
- ESP\_FAIL: get Ethernet PHY link status failed because some error occurred

**Parameters**

- [in] phy: Ethernet PHY instance

*esp\_err\_t* (\***pwrctrl**)(*esp\_eth\_phy\_t* \*phy, bool enable)

Power control of Ethernet PHY.

**Return**

- ESP\_OK: control Ethernet PHY power successfully
- ESP\_FAIL: control Ethernet PHY power failed because some error occurred

**Parameters**

- [in] phy: Ethernet PHY instance
- [in] enable: set true to power on Ethernet PHY; set false to power off Ethernet PHY

*esp\_err\_t* (\***set\_addr**)(*esp\_eth\_phy\_t* \*phy, uint32\_t addr)

Set PHY chip address.

**Return**

- ESP\_OK: set Ethernet PHY address successfully
- ESP\_FAIL: set Ethernet PHY address failed because some error occurred

**Parameters**

- [in] phy: Ethernet PHY instance
- [in] addr: PHY chip address



```
esp_err_t (*get_addr) (esp_eth_phy_t *phy, uint32_t *addr)
```

Get PHY chip address.

#### Return

- ESP\_OK: get Ethernet PHY address successfully
- ESP\_ERR\_INVALID\_ARG: get Ethernet PHY address failed because of invalid argument

#### Parameters

- [in] phy: Ethernet PHY instance
- [out] addr: PHY chip address

```
esp_err_t (*del) (esp_eth_phy_t *phy)
```

Free memory of Ethernet PHY instance.

#### Return

- ESP\_OK: free PHY instance successfully
- ESP\_FAIL: free PHY instance failed because some error occurred

#### Parameters

- [in] phy: Ethernet PHY instance

```
struct eth_phy_config_t
```

Ethernet PHY configuration.

### Public Members

```
uint32_t phy_addr
```

PHY address

```
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.

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。

## 2.2.3 IP 网络层协议

### TCP/IP Adapter

The purpose of TCP/IP Adapter 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 TCP/IP Adapter for the lwIP TCP/IP stack only. However, the adapter itself is TCP/IP implementation agnostic and different implementations are possible.

Some TCP/IP Adapter 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 TCP/IP Adapter APIs directly as they are called from the default network event handlers.

## API Reference

### Header File

- [tcpip\\_adapter/include/tcpip\\_adapter.h](#)

### Functions

void **tcpip\_adapter\_init** (void)

Initialize the underlying TCP/IP stack.

**Note** This function should be called exactly once from application code, when the application starts up.

*esp\_err\_t* **tcpip\_adapter\_eth\_start** (uint8\_t \*mac, tcpip\_adapter\_ip\_info\_t \*ip\_info, void \*args)

Cause the TCP/IP stack to start the Ethernet interface with specified MAC and IP.

**Note** This function should be called after the Ethernet MAC hardware is initialized. In the default configuration, application code does not need to call this function - it is called automatically by the default handler for the SYSTEM\_EVENT\_ETH\_START event.

#### Return

- ESP\_OK
- ESP\_ERR\_TCPIP\_ADAPTER\_INVALID\_PARAMS
- ESP\_ERR\_NO\_MEM

#### Parameters

- [in] mac: Set MAC address of this interface
- [in] ip\_info: Set IP address of this interface
- [in] args: extra args passed to tcpip\_adapter

*esp\_err\_t* **tcpip\_adapter\_sta\_start** (uint8\_t \*mac, tcpip\_adapter\_ip\_info\_t \*ip\_info)

Cause the TCP/IP stack to start the Wi-Fi station interface with specified MAC and IP.

**Note** This function should be called after the Wi-Fi Station hardware is initialized. In the default configuration, application code does not need to call this function - it is called automatically by the default handler for the SYSTEM\_EVENT\_STA\_START event.

#### Return

- ESP\_OK
- ESP\_ERR\_TCPIP\_ADAPTER\_INVALID\_PARAMS
- ESP\_ERR\_NO\_MEM

#### Parameters

- [in] mac: Set MAC address of this interface
- [in] ip\_info: Set IP address of this interface

*esp\_err\_t* **tcpip\_adapter\_ap\_start** (uint8\_t \*mac, tcpip\_adapter\_ip\_info\_t \*ip\_info)

Cause the TCP/IP stack to start the Wi-Fi AP interface with specified MAC and IP.

DHCP server will be started automatically when this function is called.

**Note** This function should be called after the Wi-Fi AP hardware is initialized. In the default configuration, application code does not need to call this function - it is called automatically by the default handler for the SYSTEM\_EVENT\_AP\_START event.

#### Return

- ESP\_OK
- ESP\_ERR\_TCPIP\_ADAPTER\_INVALID\_PARAMS
- ESP\_ERR\_NO\_MEM

#### Parameters

- [in] mac: Set MAC address of this interface

- [in] ip\_info: Set IP address of this interface

*esp\_err\_t* **tcpip\_adapter\_stop** (*tcpip\_adapter\_if\_t* tcpip\_if)

Cause the TCP/IP stack to stop a network interface.

Causes TCP/IP stack to clean up this interface. This includes stopping the DHCP server or client, if they are started.

**Note** This API is called by the default Wi-Fi and Ethernet event handlers if the underlying network driver reports that the interface has stopped.

**Note** To stop an interface from application code, call the network-specific API (`esp_wifi_stop()` or `esp_eth_stop()`). The driver layer will then send a stop event and the event handler should call this API. Otherwise, the driver and MAC layer will remain started.

**Return**

- ESP\_OK
- ESP\_ERR\_TCPIP\_ADAPTER\_INVALID\_PARAMS
- ESP\_ERR\_TCPIP\_ADAPTER\_IF\_NOT\_READY

**Parameters**

- [in] tcpip\_if: Interface which will be stopped

*esp\_err\_t* **tcpip\_adapter\_up** (*tcpip\_adapter\_if\_t* tcpip\_if)

Cause the TCP/IP stack to bring up an interface.

**Note** This function is called automatically by the default event handlers for the Wi-Fi Station and Ethernet interfaces, in response to the SYSTEM\_EVENT\_STA\_CONNECTED and SYSTEM\_EVENT\_ETH\_CONNECTED events, respectively.

**Note** This function is not normally used with Wi-Fi AP interface. If the AP interface is started, it is up.

**Return**

- ESP\_OK
- ESP\_ERR\_TCPIP\_ADAPTER\_IF\_NOT\_READY

**Parameters**

- [in] tcpip\_if: Interface to bring up

*esp\_err\_t* **tcpip\_adapter\_down** (*tcpip\_adapter\_if\_t* tcpip\_if)

Cause the TCP/IP stack to shutdown an interface.

**Note** This function is called automatically by the default event handlers for the Wi-Fi Station and Ethernet interfaces, in response to the SYSTEM\_EVENT\_STA\_DISCONNECTED and SYSTEM\_EVENT\_ETH\_DISCONNECTED events, respectively.

**Note** This function is not normally used with Wi-Fi AP interface. If the AP interface is stopped, it is down.

**Return**

- ESP\_OK
- ESP\_ERR\_TCPIP\_ADAPTER\_IF\_NOT\_READY

**Parameters**

- [in] tcpip\_if: Interface to shutdown

*esp\_err\_t* **tcpip\_adapter\_get\_ip\_info** (*tcpip\_adapter\_if\_t* tcpip\_if, *tcpip\_adapter\_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 TCP/IP adapter library itself.

**Return**

- ESP\_OK
- ESP\_ERR\_TCPIP\_ADAPTER\_INVALID\_PARAMS

**Parameters**

- [in] tcpip\_if: Interface to get IP information
- [out] ip\_info: If successful, IP information will be returned in this argument.

*esp\_err\_t* **tcpip\_adapter\_set\_ip\_info** (*tcpip\_adapter\_if\_t* tcpip\_if, *tcpip\_adapter\_ip\_info\_t* \*ip\_info, **const**)

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 TCP/IP adapter library is also updated (this copy is returned if the IP is queried while the interface is still down.)

**Note** DHCP client/server must be stopped before setting new IP information.

**Note** Calling this interface for the Wi-Fi STA or Ethernet interfaces may generate a SYSTEM\_EVENT\_STA\_GOT\_IP or SYSTEM\_EVENT\_ETH\_GOT\_IP event.

#### Return

- ESP\_OK
- ESP\_ERR\_TCPIP\_ADAPTER\_INVALID\_PARAMS
- ESP\_ERR\_TCPIP\_ADAPTER\_DHCP\_NOT\_STOPPED If DHCP server or client is still running

#### Parameters

- [in] `tcpip_if`: Interface to set IP information
- [in] `ip_info`: IP information to set on the specified interface

*esp\_err\_t tcpip\_adapter\_set\_dns\_info(tcpip\_adapter\_if\_t tcpip\_if, tcpip\_adapter\_dns\_type\_t type, tcpip\_adapter\_dns\_info\_t \*dns)*

Set DNS Server information.

This function behaves differently for different interfaces:

- For Wi-Fi Station interface and Ethernet interface, up to three types of DNS server can be set (in order of priority):
  - Main DNS Server (TCPIP\_ADAPTER\_DNS\_MAIN)
  - Backup DNS Server (TCPIP\_ADAPTER\_DNS\_BACKUP)
  - Fallback DNS Server (TCPIP\_ADAPTER\_DNS\_FALLBACK)

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.

- For Wi-Fi AP interface, 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 the IP of the Wi-Fi AP interface itself.
  - This function can override it by setting server type TCPIP\_ADAPTER\_DNS\_MAIN.
  - Other DNS Server types are not supported for the Wi-Fi AP interface.

#### Return

- ESP\_OK on success
- ESP\_ERR\_TCPIP\_ADAPTER\_INVALID\_PARAMS invalid params

#### Parameters

- [in] `tcpip_if`: Interface to set DNS Server information
- [in] `type`: Type of DNS Server to set: TCPIP\_ADAPTER\_DNS\_MAIN, TCPIP\_ADAPTER\_DNS\_BACKUP, TCPIP\_ADAPTER\_DNS\_FALLBACK
- [in] `dns`: DNS Server address to set

*esp\_err\_t tcpip\_adapter\_get\_dns\_info(tcpip\_adapter\_if\_t tcpip\_if, tcpip\_adapter\_dns\_type\_t type, tcpip\_adapter\_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 `tcpip_adapter_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.

#### Return

- ESP\_OK on success
- ESP\_ERR\_TCPIP\_ADAPTER\_INVALID\_PARAMS invalid params

#### Parameters

- [in] `tcpip_if`: Interface to get DNS Server information
- [in] `type`: Type of DNS Server to get: TCPIP\_ADAPTER\_DNS\_MAIN, TCPIP\_ADAPTER\_DNS\_BACKUP, TCPIP\_ADAPTER\_DNS\_FALLBACK

- [out] dns: DNS Server result is written here on success

`esp_err_t tcpip_adapter_get_old_ip_info(tcpip_adapter_if_t tcpip_if, tcpip_adapter_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.

#### Return

- ESP\_OK
- ESP\_ERR\_TCPIP\_ADAPTER\_INVALID\_PARAMS

#### Parameters

- [in] tcpip\_if: Interface to get old IP information
- [out] ip\_info: If successful, IP information will be returned in this argument.

`esp_err_t tcpip_adapter_set_old_ip_info(tcpip_adapter_if_t tcpip_if, const tcpip_adapter_ip_info_t *ip_info)`

Set interface old IP information.

This function is called from the DHCP client for the Wi-Fi STA and Ethernet interfaces, 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.

#### Return

- ESP\_OK
- ESP\_ERR\_TCPIP\_ADAPTER\_INVALID\_PARAMS

#### Parameters

- [in] tcpip\_if: Interface to set old IP information
- [in] ip\_info: Store the old IP information for the specified interface

`esp_err_t tcpip_adapter_create_ip6_linklocal(tcpip_adapter_if_t tcpip_if)`

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.

#### Return

- ESP\_OK
- ESP\_ERR\_TCPIP\_ADAPTER\_INVALID\_PARAMS

#### Parameters

- [in] tcpip\_if: Interface to create a link-local IPv6 address

`esp_err_t tcpip_adapter_get_ip6_linklocal(tcpip_adapter_if_t tcpip_if, 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.

#### Return

- 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.

#### Parameters

- [in] tcpip\_if: Interface to get link-local IPv6 address
- [out] if\_ip6: IPv6 information will be returned in this argument if successful.

*esp\_err\_t* **tcpip\_adapter\_get\_ip6\_global** (*tcpip\_adapter\_if\_t* *tcpip\_if*, *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.

**Return**

- 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.

**Parameters**

- [in] *tcpip\_if*: Interface to get global IPv6 address
- [out] *if\_ip6*: IPv6 information will be returned in this argument if successful.

*esp\_err\_t* **tcpip\_adapter\_dhcps\_get\_status** (*tcpip\_adapter\_if\_t* *tcpip\_if*, *tcpip\_adapter\_dhcp\_status\_t* *\*status*)

Get DHCP Server status.

**Return**

- ESP\_OK

**Parameters**

- [in] *tcpip\_if*: Interface to get status of DHCP server.
- [out] *status*: If successful, the status of the DHCP server will be returned in this argument.

*esp\_err\_t* **tcpip\_adapter\_dhcps\_option** (*tcpip\_adapter\_dhcp\_option\_mode\_t* *opt\_op*, *tcpip\_adapter\_dhcp\_option\_id\_t* *opt\_id*, *void* *\*opt\_val*, *uint32\_t* *opt\_len*)

Set or Get DHCP server option.

**Return**

- ESP\_OK
- ESP\_ERR\_TCPIP\_ADAPTER\_INVALID\_PARAMS
- ESP\_ERR\_TCPIP\_ADAPTER\_DHCP\_ALREADY\_STOPPED
- ESP\_ERR\_TCPIP\_ADAPTER\_DHCP\_ALREADY\_STARTED

**Parameters**

- [in] *opt\_op*: TCPIP\_ADAPTER\_OP\_SET to set an option, TCPIP\_ADAPTER\_OP\_GET to get an option.
- [in] *opt\_id*: Option index to get or set, must be one of the supported enum values.
- [inout] *opt\_val*: Pointer to the option parameter.
- [in] *opt\_len*: Length of the option parameter.

*esp\_err\_t* **tcpip\_adapter\_dhcps\_start** (*tcpip\_adapter\_if\_t* *tcpip\_if*)

Start DHCP server.

**Note** Currently DHCP server is only supported on the Wi-Fi AP interface.

**Return**

- ESP\_OK
- ESP\_ERR\_TCPIP\_ADAPTER\_INVALID\_PARAMS
- ESP\_ERR\_TCPIP\_ADAPTER\_DHCP\_ALREADY\_STARTED

**Parameters**

- [in] *tcpip\_if*: Interface to start DHCP server. Must be TCPIP\_ADAPTER\_IF\_AP.

*esp\_err\_t* **tcpip\_adapter\_dhcps\_stop** (*tcpip\_adapter\_if\_t* *tcpip\_if*)

Stop DHCP server.

**Note** Currently DHCP server is only supported on the Wi-Fi AP interface.

**Return**

- ESP\_OK
- ESP\_ERR\_TCPIP\_ADAPTER\_INVALID\_PARAMS
- ESP\_ERR\_TCPIP\_ADAPTER\_DHCP\_ALREADY\_STOPPED
- ESP\_ERR\_TCPIP\_ADAPTER\_IF\_NOT\_READY

**Parameters**

- [in] *tcpip\_if*: Interface to stop DHCP server. Must be TCPIP\_ADAPTER\_IF\_AP.

*esp\_err\_t* **tcPIP\_adapter\_dhcpc\_get\_status** (*tcPIP\_adapter\_if\_t* *tcPIP\_adapter\_dhcpc\_status\_t* \*status) *tcPIP\_if,*

Get DHCP client status.

**Return**

- ESP\_OK

**Parameters**

- [in] *tcPIP\_if*: Interface to get status of DHCP client
- [out] *status*: If successful, the status of DHCP client will be returned in this argument.

*esp\_err\_t* **tcPIP\_adapter\_dhcpc\_option** (*tcPIP\_adapter\_dhcpc\_option\_mode\_t* *tcPIP\_adapter\_dhcpc\_option\_id\_t* *opt\_id*, void \**opt\_val*, *uint32\_t* *opt\_len*)

Set or Get DHCP client's option.

**Note** This function is not yet implemented

**Return**

- ESP\_ERR\_NOT\_SUPPORTED (not implemented)

**Parameters**

- [in] *opt\_op*: TCPIP\_ADAPTER\_OP\_SET to set an option, TCPIP\_ADAPTER\_OP\_GET to get an option.
- [in] *opt\_id*: Option index to get or set, must be one of the supported enum values.
- [inout] *opt\_val*: Pointer to the option parameter.
- [in] *opt\_len*: Length of the option parameter.

*esp\_err\_t* **tcPIP\_adapter\_dhcpc\_start** (*tcPIP\_adapter\_if\_t* *tcPIP\_if*)

Start DHCP client.

**Note** DHCP Client is only supported for the Wi-Fi station and Ethernet interfaces.

**Note** The default event handlers for the SYSTEM\_EVENT\_STA\_CONNECTED and SYSTEM\_EVENT\_ETH\_CONNECTED events call this function.

**Return**

- ESP\_OK
- ESP\_ERR\_TCPIP\_ADAPTER\_INVALID\_PARAMS
- ESP\_ERR\_TCPIP\_ADAPTER\_DHCP\_ALREADY\_STARTED
- ESP\_ERR\_TCPIP\_ADAPTER\_DHCPC\_START\_FAILED

**Parameters**

- [in] *tcPIP\_if*: Interface to start the DHCP client

*esp\_err\_t* **tcPIP\_adapter\_dhcpc\_stop** (*tcPIP\_adapter\_if\_t* *tcPIP\_if*)

Stop DHCP client.

**Note** DHCP Client is only supported for the Wi-Fi station and Ethernet interfaces.

**Note** Calling *tcPIP\_adapter\_stop()* or *tcPIP\_adapter\_down()* will also stop the DHCP Client if it is running.

**Return**

- ESP\_OK
- ESP\_ERR\_TCPIP\_ADAPTER\_INVALID\_PARAMS
- ESP\_ERR\_TCPIP\_ADAPTER\_DHCP\_ALREADY\_STOPPED
- ESP\_ERR\_TCPIP\_ADAPTER\_IF\_NOT\_READY

**Parameters**

- [in] *tcPIP\_if*: Interface to stop the DHCP client

*esp\_err\_t* **tcPIP\_adapter\_eth\_input** (void \**buffer*, *uint16\_t* *len*, void \**eb*)

Receive an Ethernet frame from the Ethernet interface.

This function will automatically be installed by *esp\_eth\_init()*. The Ethernet driver layer will then call this function to forward frames to the TCP/IP stack.

**Note** Application code does not usually need to use this function directly.

**Return**

- ESP\_OK

**Parameters**

- [in] *buffer*: Received data

- [in] len: Length of the data frame
- [in] eb: Pointer to internal Wi-Fi buffer (ignored for Ethernet)

*esp\_err\_t* **tcpip\_adapter\_sta\_input** (void \*buffer, uint16\_t len, void \*eb)

Receive an 802.11 data frame from the Wi-Fi Station interface.

This function should be installed by calling `esp_wifi_reg_rxcb()`. The Wi-Fi driver layer will then call this function to forward frames to the TCP/IP stack.

**Note** Installation happens automatically in the default handler for the `SYSTEM_EVENT_STA_CONNECTED` event.

**Note** Application code does not usually need to call this function directly.

**Return**

- ESP\_OK

**Parameters**

- [in] buffer: Received data
- [in] len: Length of the data frame
- [in] eb: Pointer to internal Wi-Fi buffer

*esp\_err\_t* **tcpip\_adapter\_ap\_input** (void \*buffer, uint16\_t len, void \*eb)

Receive an 802.11 data frame from the Wi-Fi AP interface.

This function should be installed by calling `esp_wifi_reg_rxcb()`. The Wi-Fi driver layer will then call this function to forward frames to the TCP/IP stack.

**Note** Installation happens automatically in the default handler for the `SYSTEM_EVENT_AP_START` event.

**Note** Application code does not usually need to call this function directly.

**Return**

- ESP\_OK

**Parameters**

- [in] buffer: Received data
- [in] len: Length of the data frame
- [in] eb: Pointer to internal Wi-Fi buffer

*esp\_interface\_t* **tcpip\_adapter\_get\_esp\_if** (void \*dev)

Get network interface index.

Get network interface from TCP/IP implementation-specific interface pointer.

**Return**

- ESP\_IF\_WIFI\_STA
- ESP\_IF\_WIFI\_AP
- ESP\_IF\_ETH
- ESP\_IF\_MAX - invalid parameter

**Parameters**

- [in] dev: Implementation-specific TCP/IP stack interface pointer.

*esp\_err\_t* **tcpip\_adapter\_get\_sta\_list** (const *wifi\_sta\_list\_t* \*wifi\_sta\_list, *tcpip\_adapter\_sta\_list\_t* \*tcpip\_sta\_list)

Get IP information for stations connected to the Wi-Fi AP interface.

**Return**

- ESP\_OK
- ESP\_ERR\_TCPIP\_ADAPTER\_NO\_MEM
- ESP\_ERR\_TCPIP\_ADAPTER\_INVALID\_PARAMS

**Parameters**

- [in] wifi\_sta\_list: Wi-Fi station info list, returned from `esp_wifi_ap_get_sta_list()`
- [out] tcpip\_sta\_list: IP layer station info list, corresponding to MAC addresses provided in `wifi_sta_list`

*esp\_err\_t* **tcpip\_adapter\_set\_hostname** (*tcpip\_adapter\_if\_t* tcpip\_if, const char \*hostname)

Set the hostname of an interface.

**Return**

- ESP\_OK - success



- ESP\_ERR\_TCPIP\_ADAPTER\_IF\_NOT\_READY - interface status error
- ESP\_ERR\_TCPIP\_ADAPTER\_INVALID\_PARAMS - parameter error

**Parameters**

- [in] `tcpip_if`: Interface to set the hostname
- [in] `hostname`: New hostname for the interface. Maximum length 32 bytes.

*esp\_err\_t* **tcpip\_adapter\_get\_hostname** (*tcpip\_adapter\_if\_t* `tcpip_if`, **const** char **\*\***`hostname`)

Get interface hostname.

**Return**

- ESP\_OK - success
- ESP\_ERR\_TCPIP\_ADAPTER\_IF\_NOT\_READY - interface status error
- ESP\_ERR\_TCPIP\_ADAPTER\_INVALID\_PARAMS - parameter error

**Parameters**

- [in] `tcpip_if`: Interface to get the hostname
- [out] `hostname`: 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).

*esp\_err\_t* **tcpip\_adapter\_get\_netif** (*tcpip\_adapter\_if\_t* `tcpip_if`, void **\*\***`netif`)

Get the TCP/IP stack-specific interface that is assigned to a given interface.

**Note** For lwIP, this returns a pointer to a `netif` structure.

**Return**

- ESP\_OK - success
- ESP\_ERR\_TCPIP\_ADAPTER\_IF\_NOT\_READY - interface status error
- ESP\_ERR\_TCPIP\_ADAPTER\_INVALID\_PARAMS - parameter error

**Parameters**

- [in] `tcpip_if`: Interface to get the implementation-specific interface
- [out] `netif`: Pointer to the implementation-specific interface

bool **tcpip\_adapter\_is\_netif\_up** (*tcpip\_adapter\_if\_t* `tcpip_if`)

Test if supplied interface is up or down.

**Return**

- true - Interface is up
- false - Interface is down

**Parameters**

- [in] `tcpip_if`: Interface to test up/down status

*esp\_err\_t* **tcpip\_adapter\_test\_start** (uint8\_t `*mac`, *tcpip\_adapter\_ip\_info\_t* `*ip_info`)

Cause the TCP/IP stack to start the test interface with specified MAC and IP. Test interface is used to exercise network stack with injected packets from SW.

**Return**

- ESP\_OK
- ESP\_ERR\_TCPIP\_ADAPTER\_INVALID\_PARAMS
- ESP\_ERR\_NO\_MEM

**Parameters**

- [in] `mac`: Set MAC address of this interface
- [in] `ip_info`: Set IP address of this interface

*esp\_err\_t* **tcpip\_adapter\_set\_default\_eth\_handlers** ()

Install default event handlers for Ethernet interface.

**Return**

- ESP\_OK on success
- one of the errors from `esp_event` on failure

*esp\_err\_t* **tcpip\_adapter\_clear\_default\_eth\_handlers** ()

Uninstall default event handlers for Ethernet interface.

**Return**

- ESP\_OK on success
- one of the errors from `esp_event` on failure

*esp\_err\_t* **tcpip\_adapter\_set\_default\_wifi\_handlers** ()

Install default event handlers for Wi-Fi interfaces (station and AP)

**Return**

- ESP\_OK on success
- one of the errors from esp\_event on failure

*esp\_err\_t* **tcpip\_adapter\_clear\_default\_wifi\_handlers** ()

Uninstall default event handlers for Wi-Fi interfaces (station and AP)

**Return**

- ESP\_OK on success
- one of the errors from esp\_event on failure

int **tcpip\_adapter\_get\_netif\_index** (*tcpip\_adapter\_if\_t* tcpip\_if)

Search netif index through netif interface.

**Return**

- netif\_index on success
- -1 if an invalid parameter is supplied

**Parameters**

- [in] tcpip\_if: Interface to search for netif index

void **esp\_netif\_netstack\_buf\_ref** (void \**netstack\_buf*)

increase the reference counter of net stack buffer

**Parameters**

- [in] netstack\_buf: the net stack buffer

void **esp\_netif\_netstack\_buf\_free** (void \**netstack\_buf*)

free the netstack buffer

**Parameters**

- [in] netstack\_buf: the net stack buffer

## Structures

**struct tcpip\_adapter\_dns\_info\_t**

DNS server info.

## Public Members

ip\_addr\_t **ip**

IPV4 address of DNS server

**struct ip\_event\_got\_ip\_t**

Event structure for IP\_EVENT\_STA\_GOT\_IP, IP\_EVENT\_ETH\_GOT\_IP events

## Public Members

*tcpip\_adapter\_if\_t* **if\_index**

Interface for which the event is received

tcpip\_adapter\_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

*tcpip\_adapter\_if\_t* **if\_index**

Interface for which the event is received

*tcpip\_adapter\_ip6\_info\_t* **ip6\_info**

IPv6 address of the interface

**struct ip\_event\_ap\_staassigned\_t**

Event structure for IP\_EVENT\_AP\_STAIPASSIGNED event

### Public Members

*ip4\_addr\_t* **ip**

IP address which was assigned to the station

### Macros

**ESP\_ERR\_TCPIP\_ADAPTER\_BASE**

**ESP\_ERR\_TCPIP\_ADAPTER\_INVALID\_PARAMS**

**ESP\_ERR\_TCPIP\_ADAPTER\_IF\_NOT\_READY**

**ESP\_ERR\_TCPIP\_ADAPTER\_DHCP\_START\_FAILED**

**ESP\_ERR\_TCPIP\_ADAPTER\_DHCP\_ALREADY\_STARTED**

**ESP\_ERR\_TCPIP\_ADAPTER\_DHCP\_ALREADY\_STOPPED**

**ESP\_ERR\_TCPIP\_ADAPTER\_NO\_MEM**

**ESP\_ERR\_TCPIP\_ADAPTER\_DHCP\_NOT\_STOPPED**

**TCPIP\_HOSTNAME\_MAX\_SIZE**

### Type Definitions

**typedef** *tcpip\_adapter\_dhcp\_option\_mode\_t* **tcpip\_adapter\_option\_mode\_t**

**typedef** *tcpip\_adapter\_dhcp\_option\_id\_t* **tcpip\_adapter\_option\_id\_t**

### Enumerations

**enum tcpip\_adapter\_if\_t**

*Values:*

**TCPIP\_ADAPTER\_IF\_STA** = 0

Wi-Fi STA (station) interface

**TCPIP\_ADAPTER\_IF\_AP**

Wi-Fi soft-AP interface

**TCPIP\_ADAPTER\_IF\_ETH**

Ethernet interface

**TCPIP\_ADAPTER\_IF\_TEST**

tcpip stack test interface

**TCPIP\_ADAPTER\_IF\_MAX**

**enum tcpip\_adapter\_dns\_type\_t**

Type of DNS server.

*Values:*

**TCPIP\_ADAPTER\_DNS\_MAIN** = 0

DNS main server address

**TCPIP\_ADAPTER\_DNS\_BACKUP**

DNS backup server address (Wi-Fi STA and Ethernet only)

**TCPIP\_ADAPTER\_DNS\_FALLBACK**

DNS fallback server address (Wi-Fi STA and Ethernet only)

**TCPIP\_ADAPTER\_DNS\_MAX**

**enum tcpip\_adapter\_dhcp\_status\_t**

Status of DHCP client or DHCP server.

*Values:*

**TCPIP\_ADAPTER\_DHCP\_INIT = 0**

DHCP client/server is in initial state (not yet started)

**TCPIP\_ADAPTER\_DHCP\_STARTED**

DHCP client/server has been started

**TCPIP\_ADAPTER\_DHCP\_STOPPED**

DHCP client/server has been stopped

**TCPIP\_ADAPTER\_DHCP\_STATUS\_MAX**

**enum tcpip\_adapter\_dhcp\_option\_mode\_t**

Mode for DHCP client or DHCP server option functions.

*Values:*

**TCPIP\_ADAPTER\_OP\_START = 0**

**TCPIP\_ADAPTER\_OP\_SET**

Set option

**TCPIP\_ADAPTER\_OP\_GET**

Get option

**TCPIP\_ADAPTER\_OP\_MAX**

**enum tcpip\_adapter\_dhcp\_option\_id\_t**

Supported options for DHCP client or DHCP server.

*Values:*

**TCPIP\_ADAPTER\_DOMAIN\_NAME\_SERVER = 6**

Domain name server

**TCPIP\_ADAPTER\_ROUTER\_SOLICITATION\_ADDRESS = 32**

Solicitation router address

**TCPIP\_ADAPTER\_REQUESTED\_IP\_ADDRESS = 50**

Request specific IP address

**TCPIP\_ADAPTER\_IP\_ADDRESS\_LEASE\_TIME = 51**

Request IP address lease time

**TCPIP\_ADAPTER\_IP\_REQUEST\_RETRY\_TIME = 52**

Request IP address retry counter

**enum ip\_event\_t**

IP event declarations

*Values:*

**IP\_EVENT\_STA\_GOT\_IP**

ESP32 station got IP from connected AP

**IP\_EVENT\_STA\_LOST\_IP**

ESP32 station lost IP and the IP is reset to 0

**IP\_EVENT\_AP\_STAIPASSIGNED**

ESP32 soft-AP assign an IP to a connected station

**IP\_EVENT\_GOT\_IP6**

ESP32 station or ap or ethernet interface v6IP addr is preferred

**IP\_EVENT\_ETH\_GOT\_IP**

ESP32 ethernet got IP from connected AP

TCP/IP 套接字 API 的示例代码存放在 ESP-IDF 示例项目的 [protocols/sockets](#) 目录下。

## 2.2.4 应用层协议

应用层网络协议（IP 网络层协议之上）的相关文档存放在 [应用层协议](#)。

## 2.3 外设 API

### 2.3.1 Analog to Digital Converter

#### Overview

The ESP32 integrates two 12-bit SAR ([Successive Approximation Register](#)) ADCs supporting a total of 18 measurement channels (analog enabled pins).

The ADC driver API supports ADC1 (8 channels, attached to GPIOs 32 - 39), and ADC2 (10 channels, attached to GPIOs 0, 2, 4, 12 - 15 and 25 - 27). However, the usage of ADC2 has some restrictions for the application:

1. ADC2 is used by the Wi-Fi driver. Therefore the application can only use ADC2 when the Wi-Fi driver has not started.
2. Some of the ADC2 pins are used as strapping pins (GPIO 0, 2, 15) thus cannot be used freely. Such is the case in the following official Development Kits:
  - [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.

#### Configuration and Reading ADC

The ADC should be configured before reading is taken.

- For ADC1, configure desired precision and attenuation by calling functions [adc1\\_config\\_width\(\)](#) and [adc1\\_config\\_channel\\_atten\(\)](#).
- For ADC2, configure the attenuation by [adc2\\_config\\_channel\\_atten\(\)](#). The reading width of ADC2 is configured every time you take the reading.

Attenuation configuration is done per channel, see [adc1\\_channel\\_t](#) and [adc2\\_channel\\_t](#), set as a parameter of above functions.

Then it is possible to read ADC conversion result with [adc1\\_get\\_raw\(\)](#) and [adc2\\_get\\_raw\(\)](#). Reading width of ADC2 should be set as a parameter of [adc2\\_get\\_raw\(\)](#) instead of in the configuration functions.

---

**注解:** Since the ADC2 is shared with the WIFI module, which has higher priority, reading operation of [adc2\\_get\\_raw\(\)](#) will fail between [esp\\_wifi\\_start\(\)](#) and [esp\\_wifi\\_stop\(\)](#). Use the return code to see whether the reading is successful.

---

It is also possible to read the internal hall effect sensor via ADC1 by calling dedicated function [hall\\_sensor\\_read\(\)](#). Note that even the hall sensor is internal to ESP32, reading from it uses channels 0 and 3 of ADC1 (GPIO 36 and 39). Do not connect anything else to these pins and do not change their configuration. Otherwise it may affect the measurement of low value signal from the sensor.

This API provides convenient way to configure ADC1 for reading from *ULP*. To do so, call function `adc1_ulp_enable()` and then set precision and attenuation as discussed above.

There is another specific function `adc2_vref_to_gpio()` used to route internal reference voltage to a GPIO pin. It comes handy to calibrate ADC reading and this is discussed in section *Minimizing Noise*.

### Application Examples

Reading voltage on ADC1 channel 0 (GPIO 36):

```
#include <driver/adc.h>

...

adc1_config_width(ADC_WIDTH_BIT_12);
adc1_config_channel_atten(ADC1_CHANNEL_0, ADC_ATTEN_DB_0);
int val = adc1_get_raw(ADC1_CHANNEL_0);
```

The input voltage in above example is from 0 to 1.1V (0 dB attenuation). The input range can be extended by setting higher attenuation, see `adc_atten_t`. An example using the ADC driver including calibration (discussed below) is available in esp-idf: [peripherals/adc](#)

Reading voltage on ADC2 channel 7 (GPIO 27):

```
#include <driver/adc.h>

...

int read_raw;
adc2_config_channel_atten( ADC2_CHANNEL_7, ADC_ATTEN_0db );

esp_err_t r = adc2_get_raw( ADC2_CHANNEL_7, ADC_WIDTH_12Bit, &read_raw);
if ( r == ESP_OK ) {
    printf("%d\n", read_raw );
} else if ( r == ESP_ERR_TIMEOUT ) {
    printf("ADC2 used by Wi-Fi.\n");
}
```

The reading may fail due to collision with Wi-Fi, should check it. An example using the ADC2 driver to read the output of DAC is available in esp-idf: [peripherals/adc2](#)

Reading the internal hall effect sensor:

```
#include <driver/adc.h>

...

adc1_config_width(ADC_WIDTH_BIT_12);
int val = hall_sensor_read();
```

The value read in both these examples is 12 bits wide (range 0-4095).

### Minimizing Noise

The ESP32 ADC can be sensitive to noise leading to large discrepancies in ADC readings. To minimize noise, users may connect a 0.1uF capacitor to the ADC input pad in use. Multisampling may also be used to further mitigate the effects of noise.

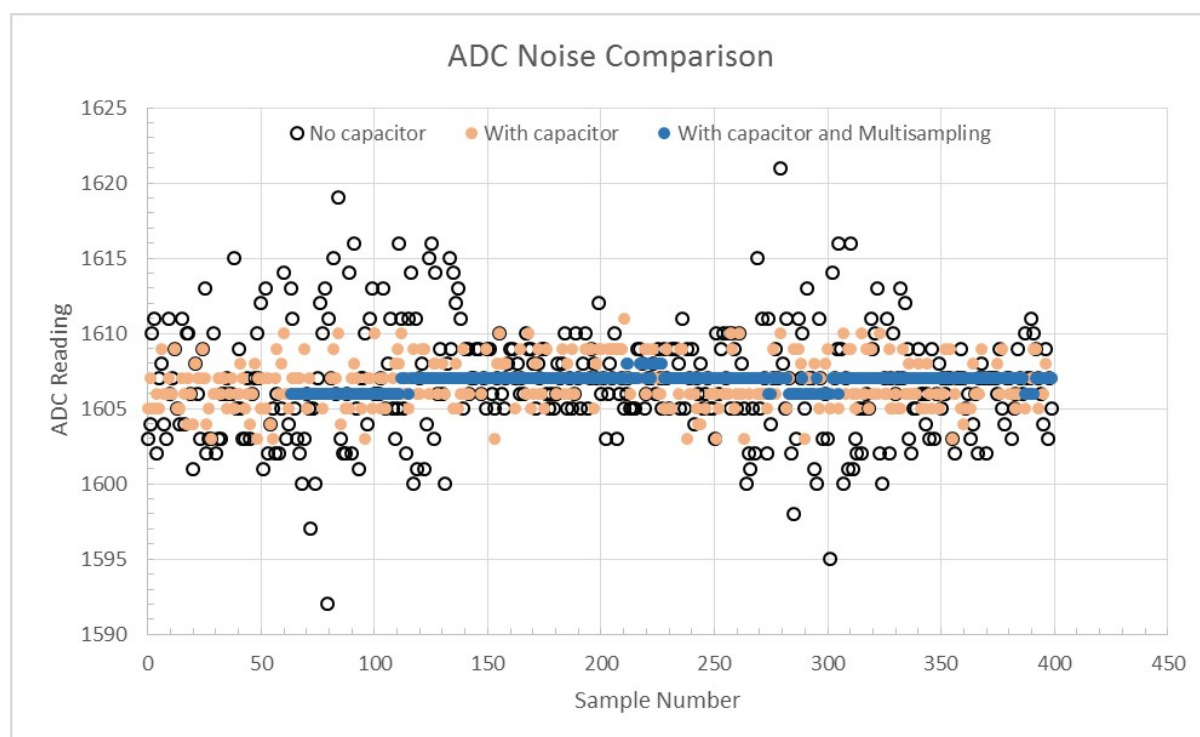


图 4: Graph illustrating noise mitigation using capacitor and multisampling of 64 samples.

### ADC Calibration

The `esp_adc_cal/include/esp_adc_cal.h` API provides functions to correct for differences in measured voltages caused by variation of ADC reference voltages ( $V_{ref}$ ) between chips. Per design the ADC reference voltage is 1100mV, however the true reference voltage can range from 1000mV to 1200mV amongst different ESP32s.

Correcting ADC readings using this API involves characterizing one of the ADCs at a given attenuation to obtain a characteristics curve (ADC-Voltage curve) that takes into account the difference in ADC reference voltage. The characteristics curve is in the form of  $y = \text{coeff\_a} * x + \text{coeff\_b}$  and is used to convert ADC readings to voltages in mV. Calculation of the characteristics curve is based on calibration values which can be stored in eFuse or provided by the user.

**Calibration Values** Calibration values are used to generate characteristic curves that account for the unique ADC reference voltage of a particular ESP32. There are currently three sources of calibration values. The availability of these calibration values will depend on the type and production date of the ESP32 chip/module.

- **Two Point** values represent each of the ADCs' readings at 150mV and 850mV. To obtain more accurate calibration results these values should be measured by user and burned into eFuse BLOCK3.
- **eFuse  $V_{ref}$**  represents the true ADC reference voltage. This value is measured and burned into eFuse BLOCK0 during factory calibration.
- **Default  $V_{ref}$**  is an estimate of the ADC reference voltage provided by the user as a parameter during characterization. If Two Point or eFuse  $V_{ref}$  values are unavailable, **Default  $V_{ref}$**  will be used.

Individual measurement and burning of the **eFuse  $V_{ref}$**  has been applied to ESP32-D0WD and ESP32-D0WDQ6 chips produced on/after the 1st week of 2018. Such chips may be recognized by date codes on/later than 012018 (see Line 4 on figure below).

If you would like to purchase chips or modules with calibration, double check with distributor or Espressif directly.

If you are unable to check the date code (i.e. the chip may be enclosed inside a canned module, etc.), you can still verify if **eFuse  $V_{ref}$**  is present by running `espefuse.py` tool with `adc_info` parameter

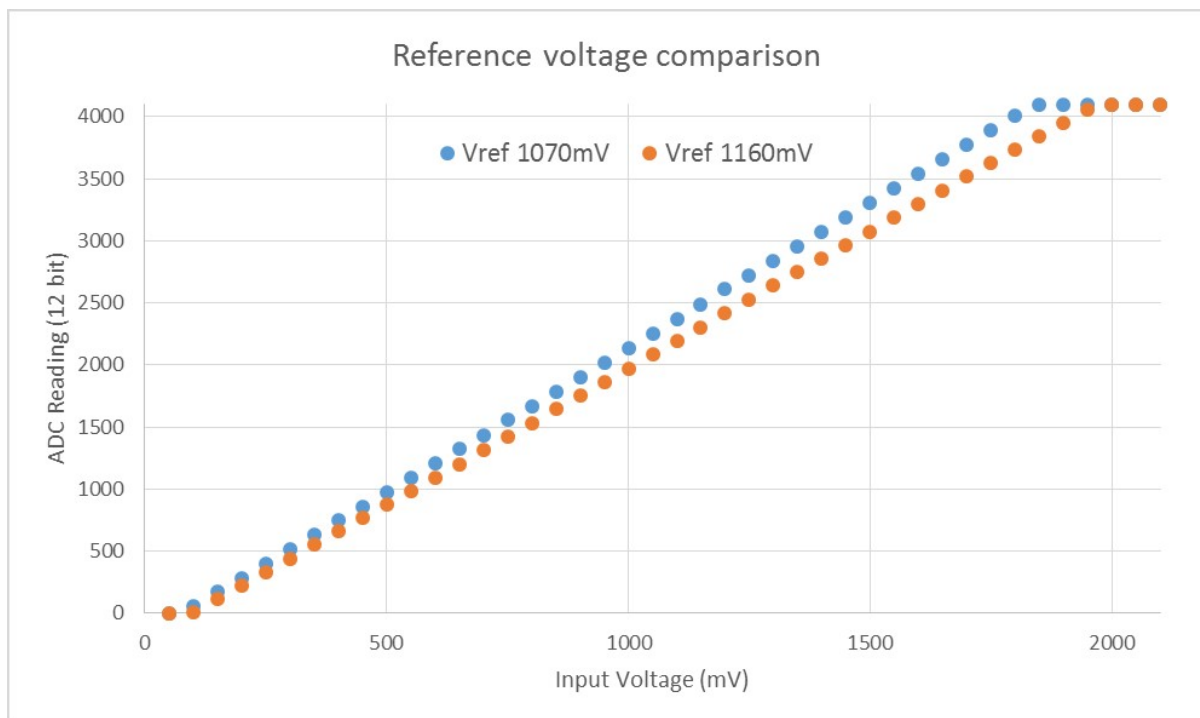


图 5: Graph illustrating effect of differing reference voltages on the ADC voltage curve.

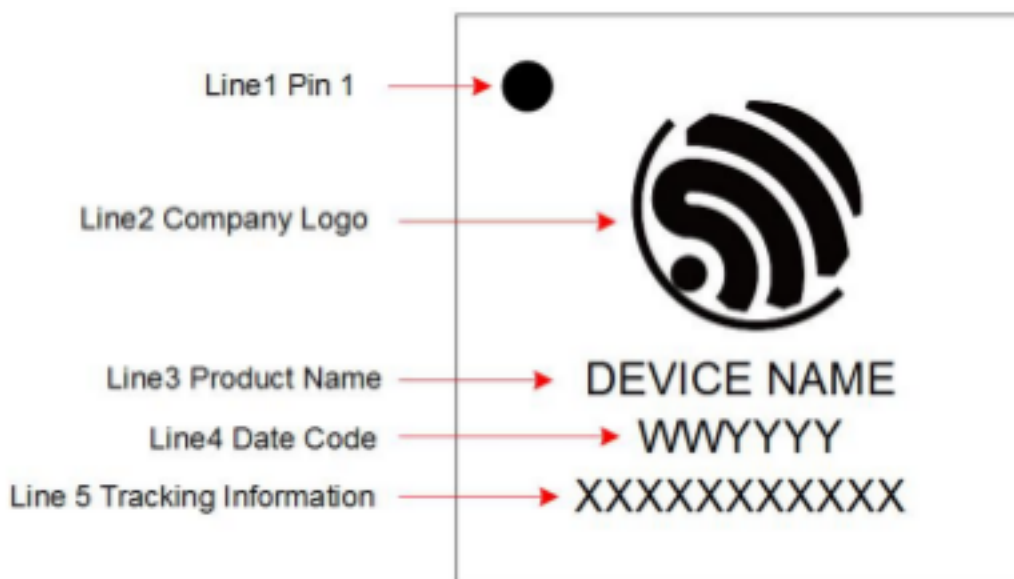


图 6: ESP32 Chip Surface Marking



```
$IDF_PATH/components/esptool_py/esptool/espefuse.py --port /dev/ttyUSB0 adc_info
```

Replace `/dev/ttyUSB0` with ESP32 board's port name.

A chip that has specific **eFuse Vref** value programmed (in this case 1093mV) will be reported as follows:

```
ADC VRef calibration: 1093mV
```

In another example below the **eFuse Vref** is not programmed:

```
ADC VRef calibration: None (1100mV nominal)
```

For a chip with two point calibration the message will look similar to:

```
ADC VRef calibration: 1149mV
ADC readings stored in efuse BLK3:
  ADC1 Low reading (150mV): 306
  ADC1 High reading (850mV): 3153
  ADC2 Low reading (150mV): 389
  ADC2 High reading (850mV): 3206
```

**Application Example** For a full example see esp-idf: [peripherals/adc](#)

Characterizing an ADC at a particular attenuation:

```
#include "driver/adc.h"
#include "esp_adc_cal.h"

...

    //Characterize ADC at particular atten
    esp_adc_cal_characteristics_t *adc_chars = calloc(1, sizeof(esp_adc_cal_
↪characteristics_t));
    esp_adc_cal_value_t val_type = esp_adc_cal_characterize(unit, atten, ADC_WIDTH_
↪BIT_12, DEFAULT_VREF, adc_chars);
    //Check type of calibration value used to characterize ADC
    if (val_type == ESP_ADC_CAL_VAL_EFUSE_VREF) {
        printf("eFuse Vref");
    } else if (val_type == ESP_ADC_CAL_VAL_EFUSE_TP) {
        printf("Two Point");
    } else {
        printf("Default");
    }
}
```

Reading an ADC then converting the reading to a voltage:

```
#include "driver/adc.h"
#include "esp_adc_cal.h"

...

    uint32_t reading = adc1_get_raw(ADC1_CHANNEL_5);
    uint32_t voltage = esp_adc_cal_raw_to_voltage(reading, adc_chars);
```

Routing ADC reference voltage to GPIO, so it can be manually measured (for **Default Vref**):

```
#include "driver/adc.h"

...

    esp_err_t status = adc2_vref_to_gpio(GPIO_NUM_25);
    if (status == ESP_OK) {
```

(下页继续)

```

    printf("v_ref routed to GPIO\n");
} else {
    printf("failed to route v_ref\n");
}

```

## GPIO Lookup Macros

There are macros available to specify the GPIO number of a ADC channel, or vice versa. e.g.

1. ADC1\_CHANNEL\_0\_GPIO\_NUM is the GPIO number of ADC1 channel 0 (36);
2. ADC1\_GPIO32\_CHANNEL is the ADC1 channel number of GPIO 32 (ADC1 channel 4).

## API Reference

This reference covers three components:

- [ADC driver](#)
- [ADC Calibration](#)
- [GPIO Lookup Macros](#)

## ADC driver

### Header File

- [driver/include/driver/adc.h](#)

### Functions

*esp\_err\_t* **adc1\_pad\_get\_io\_num** (*adc1\_channel\_t* channel, *gpio\_num\_t* \*gpio\_num)

Get the gpio number of a specific ADC1 channel.

#### Return

- ESP\_OK if success
- ESP\_ERR\_INVALID\_ARG if channel not valid

#### Parameters

- channel: Channel to get the gpio number
- gpio\_num: output buffer to hold the gpio number

*esp\_err\_t* **adc1\_config\_width** (*adc\_bits\_width\_t* width\_bit)

Configure ADC1 capture width, meanwhile enable output invert for ADC1. The configuration is for all channels of ADC1.

#### Return

- ESP\_OK success
- ESP\_ERR\_INVALID\_ARG Parameter error

#### Parameters

- width\_bit: Bit capture width for ADC1

*esp\_err\_t* **adc\_set\_data\_width** (*adc\_unit\_t* adc\_unit, *adc\_bits\_width\_t* width\_bit)

Configure ADC capture width.

#### Return

- ESP\_OK success
- ESP\_ERR\_INVALID\_ARG Parameter error

#### Parameters

- adc\_unit: ADC unit index
- width\_bit: Bit capture width for ADC unit.

`esp_err_t adc1_config_channel_atten(adc1_channel_t channel, adc_atten_t atten)`

Set the attenuation of a particular channel on ADC1, and configure its associated GPIO pin mux.

The default ADC full-scale voltage is 1.1V. To read higher voltages (up to the pin maximum voltage, usually 3.3V) requires setting >0dB signal attenuation for that ADC channel.

**Note** For any given channel, this function must be called before the first time `adc1_get_raw()` is called for that channel.

**Note** This function can be called multiple times to configure multiple ADC channels simultaneously. `adc1_get_raw()` can then be called for any configured channel.

When VDD\_A is 3.3V:

- 0dB attenuation (ADC\_ATTEN\_DB\_0) gives full-scale voltage 1.1V
- 2.5dB attenuation (ADC\_ATTEN\_DB\_2\_5) gives full-scale voltage 1.5V
- 6dB attenuation (ADC\_ATTEN\_DB\_6) gives full-scale voltage 2.2V
- 11dB attenuation (ADC\_ATTEN\_DB\_11) gives full-scale voltage 3.9V (see note below)

Due to ADC characteristics, most accurate results are obtained within the following approximate voltage ranges:

**Note** The full-scale voltage is the voltage corresponding to a maximum reading (depending on ADC1 configured bit width, this value is: 4095 for 12-bits, 2047 for 11-bits, 1023 for 10-bits, 511 for 9 bits.)

**Note** At 11dB attenuation the maximum voltage is limited by VDD\_A, not the full scale voltage.

- 0dB attenuation (ADC\_ATTEN\_DB\_0) between 100 and 950mV
- 2.5dB attenuation (ADC\_ATTEN\_DB\_2\_5) between 100 and 1250mV
- 6dB attenuation (ADC\_ATTEN\_DB\_6) between 150 to 1750mV
- 11dB attenuation (ADC\_ATTEN\_DB\_11) between 150 to 2450mV

For maximum accuracy, use the ADC calibration APIs and measure voltages within these recommended ranges.

#### Return

- ESP\_OK success
- ESP\_ERR\_INVALID\_ARG Parameter error

#### Parameters

- `channel`: ADC1 channel to configure
- `atten`: Attenuation level

`int adc1_get_raw(adc1_channel_t channel)`

Take an ADC1 reading from a single channel.

**Note** When the power switch of SARADC1, SARADC2, HALL sensor and AMP sensor is turned on, the input of GPIO36 and GPIO39 will be pulled down for about 80ns. When enabling power for any of these peripherals, ignore input from GPIO36 and GPIO39. Please refer to section 3.11 of ‘ECO\_and\_Workarounds\_for\_Bugs\_in\_ESP32’ for the description of this issue.

**Note** Call `adc1_config_width()` before the first time this function is called.

**Note** For any given channel, `adc1_config_channel_atten(channel)` must be called before the first time this function is called. Configuring a new channel does not prevent a previously configured channel from being read.

#### Return

- -1: Parameter error
- Other: ADC1 channel reading.

#### Parameters

- `channel`: ADC1 channel to read

`void adc_power_on(void)`

Enable ADC power.

`void adc_power_off(void)`

Power off SAR ADC.

`void adc_power_acquire(void)`

Increment the usage counter for ADC module. ADC will stay powered on while the counter is greater than 0.

Call `adc_power_release` when done using the ADC.

void **adc\_power\_release** (void)

Decrement the usage counter for ADC module. ADC will stay powered on while the counter is greater than 0. Call this function when done using the ADC.

*esp\_err\_t* **adc\_gpio\_init** (*adc\_unit\_t* *adc\_unit*, *adc\_channel\_t* *channel*)

Initialize ADC pad.

**Return**

- ESP\_OK success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- *adc\_unit*: ADC unit index
- *channel*: ADC channel index

*esp\_err\_t* **adc\_set\_data\_inv** (*adc\_unit\_t* *adc\_unit*, bool *inv\_en*)

Set ADC data invert.

**Return**

- ESP\_OK success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- *adc\_unit*: ADC unit index
- *inv\_en*: whether enable data invert

*esp\_err\_t* **adc\_set\_clk\_div** (uint8\_t *clk\_div*)

Set ADC source clock.

**Return**

- ESP\_OK success

**Parameters**

- *clk\_div*: ADC clock divider, ADC clock is divided from APB clock

*esp\_err\_t* **adc\_set\_i2s\_data\_source** (*adc\_i2s\_source\_t* *src*)

Set I2S data source.

**Return**

- ESP\_OK success

**Parameters**

- *src*: I2S DMA data source, I2S DMA can get data from digital signals or from ADC.

*esp\_err\_t* **adc\_i2s\_mode\_init** (*adc\_unit\_t* *adc\_unit*, *adc\_channel\_t* *channel*)

Initialize I2S ADC mode.

**Return**

- ESP\_OK success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- *adc\_unit*: ADC unit index
- *channel*: ADC channel index

void **adc1\_ulp\_enable** ()

Configure ADC1 to be usable by the ULP.

This function reconfigures ADC1 to be controlled by the ULP. Effect of this function can be reverted using `adc1_get_raw` function.

Note that `adc1_config_channel_atten`, `adc1_config_width` functions need to be called to configure ADC1 channels, before ADC1 is used by the ULP.

int **hall\_sensor\_read** ()

Read Hall Sensor.

**Note** When the power switch of SARADC1, SARADC2, HALL sensor and AMP sensor is turned on, the input of GPIO36 and GPIO39 will be pulled down for about 80ns. When enabling power for

any of these peripherals, ignore input from GPIO36 and GPIO39. Please refer to section 3.11 of ‘ECO\_and\_Workarounds\_for\_Bugs\_in\_ESP32’ for the description of this issue.

**Note** The Hall Sensor uses channels 0 and 3 of ADC1. Do not configure these channels for use as ADC channels.

**Note** The ADC1 module must be enabled by calling `adc1_config_width()` before calling `hall_sensor_read()`. ADC1 should be configured for 12 bit readings, as the hall sensor readings are low values and do not cover the full range of the ADC.

**Return** The hall sensor reading.

*esp\_err\_t* `adc2_pad_get_io_num` (*adc2\_channel\_t* channel, *gpio\_num\_t* \*gpio\_num)

Get the gpio number of a specific ADC2 channel.

**Return**

- ESP\_OK if success
- ESP\_ERR\_INVALID\_ARG if channel not valid

**Parameters**

- channel: Channel to get the gpio number
- gpio\_num: output buffer to hold the gpio number

*esp\_err\_t* `adc2_config_channel_atten` (*adc2\_channel\_t* channel, *adc\_atten\_t* atten)

Configure the ADC2 channel, including setting attenuation.

The default ADC full-scale voltage is 1.1V. To read higher voltages (up to the pin maximum voltage, usually 3.3V) requires setting >0dB signal attenuation for that ADC channel.

**Note** This function also configures the input GPIO pin mux to connect it to the ADC2 channel. It must be called before calling `adc2_get_raw()` for this channel.

When VDD\_A is 3.3V:

- 0dB attenuation (ADC\_ATTEN\_0db) gives full-scale voltage 1.1V
- 2.5dB attenuation (ADC\_ATTEN\_2\_5db) gives full-scale voltage 1.5V
- 6dB attenuation (ADC\_ATTEN\_6db) gives full-scale voltage 2.2V
- 11dB attenuation (ADC\_ATTEN\_11db) gives full-scale voltage 3.9V (see note below)

**Note** The full-scale voltage is the voltage corresponding to a maximum reading (depending on ADC2 configured bit width, this value is: 4095 for 12-bits, 2047 for 11-bits, 1023 for 10-bits, 511 for 9 bits.)

**Note** At 11dB attenuation the maximum voltage is limited by VDD\_A, not the full scale voltage.

**Return**

- ESP\_OK success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- channel: ADC2 channel to configure
- atten: Attenuation level

*esp\_err\_t* `adc2_get_raw` (*adc2\_channel\_t* channel, *adc\_bits\_width\_t* width\_bit, int \*raw\_out)

Take an ADC2 reading on a single channel.

**Note** When the power switch of SARADC1, SARADC2, HALL sensor and AMP sensor is turned on, the input of GPIO36 and GPIO39 will be pulled down for about 80ns. When enabling power for any of these peripherals, ignore input from GPIO36 and GPIO39. Please refer to section 3.11 of ‘ECO\_and\_Workarounds\_for\_Bugs\_in\_ESP32’ for the description of this issue.

**Note** For a given channel, `adc2_config_channel_atten()` must be called before the first time this function is called. If Wi-Fi is started via `esp_wifi_start()`, this function will always fail with `ESP_ERR_TIMEOUT`.

**Return**

- ESP\_OK if success
- ESP\_ERR\_TIMEOUT the WIFI is started, using the ADC2

**Parameters**

- channel: ADC2 channel to read
- width\_bit: Bit capture width for ADC2
- raw\_out: the variable to hold the output data.

`esp_err_t adc2_vref_to_gpio(gpio_num_t gpio)`

Output ADC2 reference voltage to gpio 25 or 26 or 27.

This function utilizes the testing mux exclusive to ADC 2 to route the reference voltage one of ADC2's channels. Supported gpios are gpios 25, 26, and 27. This reference voltage can be manually read from the pin and used in the `esp_adc_cal` component.

#### Return

- ESP\_OK: v\_ref successfully routed to selected gpio
- ESP\_ERR\_INVALID\_ARG: Unsupported gpio

#### Parameters

- [in] gpio: GPIO number (gpios 25,26,27 supported)

#### Macros

`ADC_ATTEN_0db`

`ADC_ATTEN_2_5db`

`ADC_ATTEN_6db`

`ADC_ATTEN_11db`

`ADC_WIDTH_9Bit`

`ADC_WIDTH_10Bit`

`ADC_WIDTH_11Bit`

`ADC_WIDTH_12Bit`

#### Enumerations

`enum adc_atten_t`

*Values:*

`ADC_ATTEN_DB_0 = 0`

The input voltage of ADC will be reduced to about 1/1

`ADC_ATTEN_DB_2_5 = 1`

The input voltage of ADC will be reduced to about 1/1.34

`ADC_ATTEN_DB_6 = 2`

The input voltage of ADC will be reduced to about 1/2

`ADC_ATTEN_DB_11 = 3`

The input voltage of ADC will be reduced to about 1/3.6

`ADC_ATTEN_MAX`

`enum adc_bits_width_t`

*Values:*

`ADC_WIDTH_BIT_9 = 0`

ADC capture width is 9Bit

`ADC_WIDTH_BIT_10 = 1`

ADC capture width is 10Bit

`ADC_WIDTH_BIT_11 = 2`

ADC capture width is 11Bit

`ADC_WIDTH_BIT_12 = 3`

ADC capture width is 12Bit

`ADC_WIDTH_MAX`

`enum adc1_channel_t`

*Values:*

**ADC1\_CHANNEL\_0** = 0  
ADC1 channel 0 is GPIO36

**ADC1\_CHANNEL\_1**  
ADC1 channel 1 is GPIO37

**ADC1\_CHANNEL\_2**  
ADC1 channel 2 is GPIO38

**ADC1\_CHANNEL\_3**  
ADC1 channel 3 is GPIO39

**ADC1\_CHANNEL\_4**  
ADC1 channel 4 is GPIO32

**ADC1\_CHANNEL\_5**  
ADC1 channel 5 is GPIO33

**ADC1\_CHANNEL\_6**  
ADC1 channel 6 is GPIO34

**ADC1\_CHANNEL\_7**  
ADC1 channel 7 is GPIO35

**ADC1\_CHANNEL\_MAX**

**enum adc2\_channel\_t**

*Values:*

**ADC2\_CHANNEL\_0** = 0  
ADC2 channel 0 is GPIO4

**ADC2\_CHANNEL\_1**  
ADC2 channel 1 is GPIO0

**ADC2\_CHANNEL\_2**  
ADC2 channel 2 is GPIO2

**ADC2\_CHANNEL\_3**  
ADC2 channel 3 is GPIO15

**ADC2\_CHANNEL\_4**  
ADC2 channel 4 is GPIO13

**ADC2\_CHANNEL\_5**  
ADC2 channel 5 is GPIO12

**ADC2\_CHANNEL\_6**  
ADC2 channel 6 is GPIO14

**ADC2\_CHANNEL\_7**  
ADC2 channel 7 is GPIO27

**ADC2\_CHANNEL\_8**  
ADC2 channel 8 is GPIO25

**ADC2\_CHANNEL\_9**  
ADC2 channel 9 is GPIO26

**ADC2\_CHANNEL\_MAX**

**enum adc\_channel\_t**

*Values:*

**ADC\_CHANNEL\_0** = 0  
ADC channel

**ADC\_CHANNEL\_1**  
ADC channel

**ADC\_CHANNEL\_2**  
ADC channel

**ADC\_CHANNEL\_3**  
ADC channel

**ADC\_CHANNEL\_4**  
ADC channel

**ADC\_CHANNEL\_5**  
ADC channel

**ADC\_CHANNEL\_6**  
ADC channel

**ADC\_CHANNEL\_7**  
ADC channel

**ADC\_CHANNEL\_8**  
ADC channel

**ADC\_CHANNEL\_9**  
ADC channel

**ADC\_CHANNEL\_MAX**

**enum adc\_unit\_t**

*Values:*

**ADC\_UNIT\_1 = 1**  
SAR ADC 1

**ADC\_UNIT\_2 = 2**  
SAR ADC 2, not supported yet

**ADC\_UNIT\_BOTH = 3**  
SAR ADC 1 and 2, not supported yet

**ADC\_UNIT\_ALTER = 7**  
SAR ADC 1 and 2 alternative mode, not supported yet

**ADC\_UNIT\_MAX**

**enum adc\_i2s\_encode\_t**

*Values:*

**ADC\_ENCODE\_12BIT**  
ADC to I2S data format, [15:12]-channel [11:0]-12 bits ADC data

**ADC\_ENCODE\_11BIT**  
ADC to I2S data format, [15]-1 [14:11]-channel [10:0]-11 bits ADC data

**ADC\_ENCODE\_MAX**

**enum adc\_i2s\_source\_t**

*Values:*

**ADC\_I2S\_DATA\_SRC\_IO\_SIG = 0**  
I2S data from GPIO matrix signal

**ADC\_I2S\_DATA\_SRC\_ADC = 1**  
I2S data from ADC

**ADC\_I2S\_DATA\_SRC\_MAX**

## ADC Calibration



**Header File**

- [esp\\_adc\\_cal/include/esp\\_adc\\_cal.h](#)

**Functions**

*esp\_err\_t* **esp\_adc\_cal\_check\_efuse** (*esp\_adc\_cal\_value\_t* value\_type)

Checks if ADC calibration values are burned into eFuse.

This function checks if ADC reference voltage or Two Point values have been burned to the eFuse of the current ESP32

**Return**

- ESP\_OK: The calibration mode is supported in eFuse
- ESP\_ERR\_NOT\_SUPPORTED: Error, eFuse values are not burned
- ESP\_ERR\_INVALID\_ARG: Error, invalid argument (ESP\_ADC\_CAL\_VAL\_DEFAULT\_VREF)

**Parameters**

- value\_type: Type of calibration value (ESP\_ADC\_CAL\_VAL\_EFUSE\_VREF or ESP\_ADC\_CAL\_VAL\_EFUSE\_TP)

*esp\_adc\_cal\_value\_t* **esp\_adc\_cal\_characterize** (*adc\_unit\_t* adc\_num, *adc\_atten\_t* atten, *adc\_bits\_width\_t* bit\_width, *uint32\_t* default\_vref, *esp\_adc\_cal\_characteristics\_t* \*chars)

Characterize an ADC at a particular attenuation.

This function will characterize the ADC at a particular attenuation and generate the ADC-Voltage curve in the form of  $[y = \text{coeff\_a} * x + \text{coeff\_b}]$ . Characterization can be based on Two Point values, eFuse Vref, or default Vref and the calibration values will be prioritized in that order.

**Note** Two Point values and eFuse Vref can be enabled/disabled using menuconfig.

**Return**

- ESP\_ADC\_CAL\_VAL\_EFUSE\_VREF: eFuse Vref used for characterization
- ESP\_ADC\_CAL\_VAL\_EFUSE\_TP: Two Point value used for characterization (only in Linear Mode)
- ESP\_ADC\_CAL\_VAL\_DEFAULT\_VREF: Default Vref used for characterization

**Parameters**

- [in] adc\_num: ADC to characterize (ADC\_UNIT\_1 or ADC\_UNIT\_2)
- [in] atten: Attenuation to characterize
- [in] bit\_width: Bit width configuration of ADC
- [in] default\_vref: Default ADC reference voltage in mV (used if eFuse values is not available)
- [out] chars: Pointer to empty structure used to store ADC characteristics

*uint32\_t* **esp\_adc\_cal\_raw\_to\_voltage** (*uint32\_t* adc\_reading, *const* *esp\_adc\_cal\_characteristics\_t* \*chars)

Convert an ADC reading to voltage in mV.

This function converts an ADC reading to a voltage in mV based on the ADC's characteristics.

**Note** Characteristics structure must be initialized before this function is called (call esp\_adc\_cal\_characterize())

**Return** Voltage in mV

**Parameters**

- [in] adc\_reading: ADC reading
- [in] chars: Pointer to initialized structure containing ADC characteristics

*esp\_err\_t* **esp\_adc\_cal\_get\_voltage** (*adc\_channel\_t* channel, *const* *esp\_adc\_cal\_characteristics\_t* \*chars, *uint32\_t* \*voltage)

Reads an ADC and converts the reading to a voltage in mV.

This function reads an ADC then converts the raw reading to a voltage in mV based on the characteristics provided. The ADC that is read is also determined by the characteristics.

**Note** The Characteristics structure must be initialized before this function is called (call esp\_adc\_cal\_characterize())

**Return**

- ESP\_OK: ADC read and converted to mV
- ESP\_ERR\_TIMEOUT: Error, timed out attempting to read ADC
- ESP\_ERR\_INVALID\_ARG: Error due to invalid arguments

**Parameters**

- [in] channel: ADC Channel to read
- [in] chars: Pointer to initialized ADC characteristics structure
- [out] voltage: Pointer to store converted voltage

**Structures****struct esp\_adc\_cal\_characteristics\_t**

Structure storing characteristics of an ADC.

**Note** Call esp\_adc\_cal\_characterize() to initialize the structure**Public Members***adc\_unit\_t* **adc\_num**

ADC number

*adc\_atten\_t* **atten**

ADC attenuation

*adc\_bits\_width\_t* **bit\_width**

ADC bit width

uint32\_t **coeff\_a**

Gradient of ADC-Voltage curve

uint32\_t **coeff\_b**

Offset of ADC-Voltage curve

uint32\_t **vref**

Vref used by lookup table

**const** uint32\_t \***low\_curve**

Pointer to low Vref curve of lookup table (NULL if unused)

**const** uint32\_t \***high\_curve**

Pointer to high Vref curve of lookup table (NULL if unused)

**Enumerations****enum esp\_adc\_cal\_value\_t**

Type of calibration value used in characterization.

*Values:***ESP\_ADC\_CAL\_VAL\_EFUSE\_VREF = 0**

Characterization based on reference voltage stored in eFuse

**ESP\_ADC\_CAL\_VAL\_EFUSE\_TP = 1**

Characterization based on Two Point values stored in eFuse

**ESP\_ADC\_CAL\_VAL\_DEFAULT\_VREF = 2**

Characterization based on default reference voltage

**GPIO Lookup Macros****Header File**

- [soc/esp32/include/soc/adc\\_channel.h](#)

**Macros**

ADC1\_GPIO36\_CHANNEL  
ADC1\_CHANNEL\_0\_GPIO\_NUM  
ADC1\_GPIO37\_CHANNEL  
ADC1\_CHANNEL\_1\_GPIO\_NUM  
ADC1\_GPIO38\_CHANNEL  
ADC1\_CHANNEL\_2\_GPIO\_NUM  
ADC1\_GPIO39\_CHANNEL  
ADC1\_CHANNEL\_3\_GPIO\_NUM  
ADC1\_GPIO32\_CHANNEL  
ADC1\_CHANNEL\_4\_GPIO\_NUM  
ADC1\_GPIO33\_CHANNEL  
ADC1\_CHANNEL\_5\_GPIO\_NUM  
ADC1\_GPIO34\_CHANNEL  
ADC1\_CHANNEL\_6\_GPIO\_NUM  
ADC1\_GPIO35\_CHANNEL  
ADC1\_CHANNEL\_7\_GPIO\_NUM  
ADC2\_GPIO4\_CHANNEL  
ADC2\_CHANNEL\_0\_GPIO\_NUM  
ADC2\_GPIO0\_CHANNEL  
ADC2\_CHANNEL\_1\_GPIO\_NUM  
ADC2\_GPIO2\_CHANNEL  
ADC2\_CHANNEL\_2\_GPIO\_NUM  
ADC2\_GPIO15\_CHANNEL  
ADC2\_CHANNEL\_3\_GPIO\_NUM  
ADC2\_GPIO13\_CHANNEL  
ADC2\_CHANNEL\_4\_GPIO\_NUM  
ADC2\_GPIO12\_CHANNEL  
ADC2\_CHANNEL\_5\_GPIO\_NUM  
ADC2\_GPIO14\_CHANNEL  
ADC2\_CHANNEL\_6\_GPIO\_NUM  
ADC2\_GPIO27\_CHANNEL  
ADC2\_CHANNEL\_7\_GPIO\_NUM  
ADC2\_GPIO25\_CHANNEL  
ADC2\_CHANNEL\_8\_GPIO\_NUM  
ADC2\_GPIO26\_CHANNEL  
ADC2\_CHANNEL\_9\_GPIO\_NUM

## 2.3.2 Controller Area Network (CAN)

### Overview

The ESP32's peripherals contains a CAN Controller that supports Standard Frame Format (11-bit ID) and Extended Frame Format (29-bit ID) of the CAN2.0B specification.

**警告:** The ESP32 CAN controller is not compatible with CAN FD frames and will interpret such frames as errors.

This programming guide is split into the following sections:

1. [Basic CAN Concepts](#)
2. [Signals Lines and Transceiver](#)
3. [Configuration](#)
4. [Driver Operation](#)
5. [Examples](#)

### Basic CAN Concepts

---

**注解:** The following section only covers the basic aspects of CAN. For full details, see the CAN2.0B specification

---

The CAN protocol is a multi-master, multi-cast communication protocol with error detection/signalling and inbuilt message prioritization. The CAN protocol is commonly used as a communication bus in automotive applications.

**Multi-master:** Any node in a CAN bus is allowed initiate the transfer of data.

**Multi-cast:** When a node transmits a message, all nodes are able to receive the message (broadcast). However some nodes can selective choose which messages to accept via the use of acceptance filtering (multi-cast).

**Error Detection and Signalling:** Every CAN node will constantly monitor the CAN bus. When any node detects an error, it will signal the error 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:** If two nodes attempt to transmit simultaneously, the node transmitting the message with the lower ID will win arbitration. All other nodes will become receivers ensuring there is at most one transmitter at any time.

**CAN Message Frames** The CAN2.0B specification contains two frame formats known as **Extended Frame** and **Standard Frame** which contain 29-bit IDs and 11-bit IDs respectively. A CAN message consists of the following components

- 29-bit or 11-bit ID
- Data Length Code (DLC) between 0 to 8
- Up to 8 bytes of data (should match DLC)

**Error States and Counters** The CAN2.0B specification implements fault confinement by requiring every CAN 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 used to determine a CAN node's **error state**, and the counters are incremented and decremented following a set of rules (see CAN2.0B specification). These error states are known as **Error Active**, **Error Passive**, and **Bus-Off**.

**Error Active:** A CAN node is Error Active when **both TEC and REC are less than 128** and indicates a CAN node is operating normally. Error Active nodes are allowed to participate in CAN bus activities, and will actively signal any error conditions it detects by transmitting an **Active Error Flag** over the CAN bus.

**Error Passive:** A CAN 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 CAN bus activities, but will instead transmit a **Passive Error Flag** upon detection of an error.

**Bus-Off:** A CAN node becomes Bus-Off when the **TEC becomes greater than or equal to 256**. A Bus-Off node is unable take part in CAN bus activity and will remain so until it undergoes bus recovery.

### Signals Lines and Transceiver

The CAN controller does not contain a internal transceiver and therefore **requires an external transceiver** to operate. The type of external transceiver will depend on the application' s physical layer specification (e.g. using SN65HVD23X transceivers for ISO 11898-2 compatibility).

The CAN 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 GPIOs.

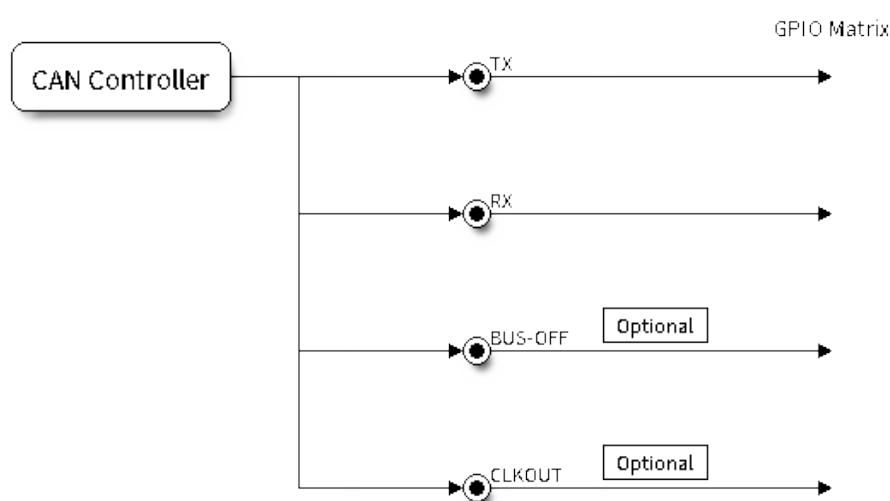


图 7: Signal lines of the CAN controller

**TX and RX:** The TX and RX signal lines are required to interface with an external CAN 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 CAN 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 CAN controller' s source clock (APB Clock).

---

**注解:** An external transceiver **must internally tie the TX input and the RX output** such that a change in logic level to the TX signal line can be observed on the RX line. Failing to do so will cause the CAN controller to interpret differences in logic levels between the two signal lines as a lost in arbitration or a bit error.

---

### Configuration

**Operating Modes** The CAN driver supports the following modes of operations:

**Normal Mode:** The normal operating mode allows the CAN controller to take part in bus activities such as transmitting and receiving messages/error frames. Acknowledgement from another CAN node is required when transmitting message frames.

**No Ack Mode:** The No Acknowledgement mode is similar to normal mode, however acknowledgements are not required when transmitting message frames. This mode is useful when self testing the CAN controller.

**Listen Only Mode:** This mode will prevent the CAN controller from taking part in bus activities. Therefore transmissions of messages/acknowledgement/error frames will be disabled. However the the CAN controller will still be able to receive messages (without acknowledging). This mode is suited for applications such as CAN bus monitoring.

**Alerts** The CAN driver contains an alert feature which is used to notify the application level of certain CAN driver events. Alerts are selectively enabled when the CAN driver is installed, but can be reconfigured during runtime by calling `can_reconfigure_alerts()`. The application can then wait for any enabled alerts to occur by calling `can_read_alerts()`. The CAN driver supports the following alerts:

Alert	Description
CAN_ALERT_TX_IDLE	No more messages queued for transmission
CAN_ALERT_TX_SUCCESS	The previous transmission was successful
CAN_ALERT_BELOW_ERR_WARN	Both error counters have dropped below error warning limit
CAN_ALERT_ERR_ACTIVE	CAN controller has become error active
CAN_ALERT_RECOVERY_IN_PROGRESS	CAN controller is undergoing bus recovery
CAN_ALERT_BUS_RECOVERED	CAN controller has successfully completed bus recovery
CAN_ALERT_ARB_LOST	The previous transmission lost arbitration
CAN_ALERT_ABOVE_ERR_WARN	One of the error counters have exceeded the error warning limit
CAN_ALERT_BUS_ERROR	A (Bit, Stuff, CRC, Form, ACK) error has occurred on the bus
CAN_ALERT_TX_FAILED	The previous transmission has failed
CAN_ALERT_RX_QUEUE_FULL	The RX queue is full causing a received frame to be lost
CAN_ALERT_ERR_PASS	CAN controller has become error passive
CAN_ALERT_BUS_OFF	Bus-off condition occurred. CAN controller can no longer influence bus

**注解:** The **error warning limit** can be used to preemptively warn the application of bus errors before the error passive state is reached. By default the CAN driver sets the **error warning limit** to **96**. The CAN\_ALERT\_ABOVE\_ERR\_WARN is raised when the TEC or REC becomes larger then or equal to the error warning limit. The CAN\_ALERT\_BELOW\_ERR\_WARN is raised when both TEC and REC return back to values below **96**.

**注解:** When enabling alerts, the CAN\_ALERT\_AND\_LOG flag can be used to cause the CAN driver to log any raised alerts to UART. However, alert logging is disabled and CAN\_ALERT\_AND\_LOG if the `CONFIG_CAN_ISR_IN_IRAM` option is enabled (see *Placing ISR into IRAM*).

**注解:** The CAN\_ALERT\_ALL and CAN\_ALERT\_NONE macros can also be used to enable/disable all alerts during configuration/reconfiguration.

**Bit Timing** The operating bit rate of the CAN controller is configured using the `can_timing_config_t` structure. The period of each bit is made up of multiple **time quanta**, and the period of a **time quanta** is determined by a prescaled version of the CAN controller's source clock. A single bit contains the following segments in the following order:

1. The **Synchronization Segment** consists of a single time quanta
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 quanta by dividing the CAN controller's source clock (80 MHz APB clock). The `brp` can be **any even number from 2 to 128**. 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.

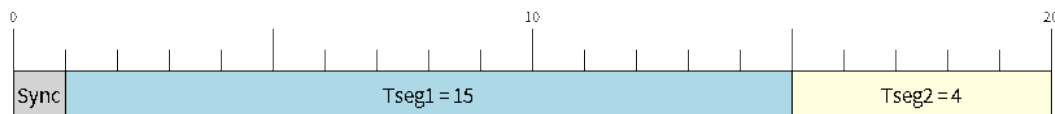


图 8: Bit timing configuration for 500kbit/s given BRP = 8

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**.

---

**注解:** 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 CAN 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 CAN bus bit rates. The following macro initializers are provided by the CAN driver.

- `CAN_TIMING_CONFIG_12_5KBIT` ()
- `CAN_TIMING_CONFIG_16KBIT` ()
- `CAN_TIMING_CONFIG_20KBIT` ()
- `CAN_TIMING_CONFIG_25KBIT` ()
- `CAN_TIMING_CONFIG_50KBIT` ()
- `CAN_TIMING_CONFIG_100KBIT` ()
- `CAN_TIMING_CONFIG_125KBIT` ()
- `CAN_TIMING_CONFIG_250KBIT` ()
- `CAN_TIMING_CONFIG_500KBIT` ()
- `CAN_TIMING_CONFIG_800KBIT` ()
- `CAN_TIMING_CONFIG_1MBIT` ()

---

**注解:** The macro initializers for 12.5K, 16K, and 20K bit rates are only available for ESP32 revision 2 or later.

---

**Acceptance Filter** The CAN controller contains a hardware acceptance filter which can be used to filter CAN 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 CAN bus that are irrelevant to the CAN node in question. The CAN controller's acceptance filter is configured using two 32-bit values within `can_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 CAN 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 CAN frames respectively).

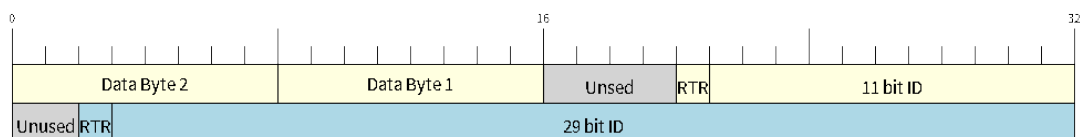


图 9: Bit layout of single filter mode (Right side MSBit)

**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 illustrates how the 32-bit acceptance code and mask will be interpreted under **Dual Filter Mode** (Note: The yellow and blue fields represent standard and extended CAN frames respectively).

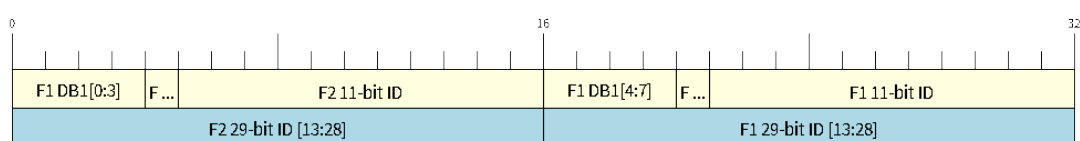


图 10: Bit layout of dual filter mode (Right side MSBit)

**Disabling TX Queue** The TX queue can be disabled during configuration by setting the `tx_queue_len` member of `can_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 CAN driver.

**Placing ISR into IRAM** The CAN 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 CAN 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 CAN driver's ISR, users must do the following:

- Enable the `CONFIG_CAN_ISR_IN_IRAM` option using `idf.py menuconfig`.
- When calling `can_driver_install()`, the `intr_flags` member of `can_general_config_t` should set the `ESP_INTR_FLAG_IRAM` set.

---

**注解:** When the `CONFIG_CAN_ISR_IN_IRAM` option is enabled, the CAN driver will no longer log any alerts (i.e., the `CAN_ALERT_AND_LOG` flag will not have any effect).

---

## Driver Operation

The CAN 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.



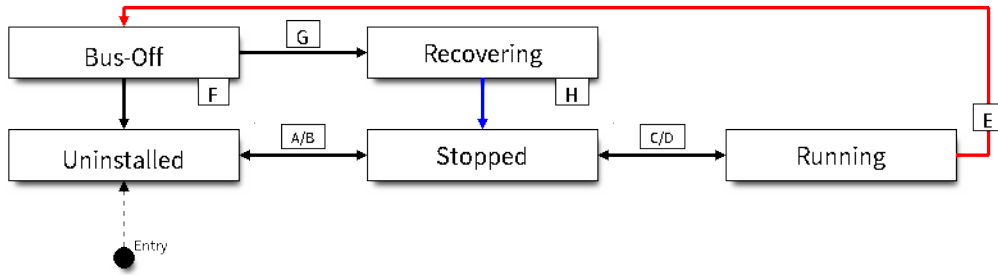


图 11: State transition diagram of the CAN driver (see table below)

Label	Transition	Action/Condition
A	Uninstalled -> Stopped	<code>can_driver_install()</code>
B	Stopped -> Uninstalled	<code>can_driver_uninstall()</code>
C	Stopped -> Running	<code>can_start()</code>
D	Running -> Stopped	<code>can_stop()</code>
E	Running -> Bus-Off	Transmit Error Counter $\geq$ 256
F	Bus-Off -> Uninstalled	<code>can_driver_uninstall()</code>
G	Bus-Off -> Recovering	<code>can_initiate_recovery()</code>
H	Recovering -> Stopped	128 occurrences of bus-free signal

**Driver States Uninstalled:** In the uninstalled state, no memory is allocated for the driver and the CAN controller is powered OFF.

**Stopped:** In this state, the CAN controller is powered ON and the CAN driver has been installed. However the CAN controller will be unable to take part in any CAN bus activities such as transmitting, receiving, or acknowledging messages.

**Running:** In the running state, the CAN controller is able to take part in bus activities. Therefore messages can be transmitted/received/acknowledged. Furthermore the CAN controller will be able to transmit error frames upon detection of errors on the CAN bus.

**Bus-Off:** The bus-off state is automatically entered when the CAN controller's Transmit Error Counter becomes greater than or equal to 256 (see CAN2.0B specification regarding error counter rules). The bus-off state indicates the occurrence of severe errors on the CAN bus or in the CAN controller. Whilst in the bus-off state, the CAN controller will be unable to take part in any CAN bus activities. To exit the bus-off state, the CAN controller must undergo the bus recovery process.

**Recovering:** The recovering state is entered when the CAN driver undergoes bus recovery. The CAN driver/controller will remain in the recovering state until the 128 occurrences of the bus-free signal (see CAN2.0B specification) is observed on the CAN bus.

**Message Flags** The CAN driver distinguishes different types of CAN messages by using the message flags in the `flags` field of `can_message_t`. These flags help distinguish whether a message is in standard or extended format, an RTR, and the type of transmission to use when transmitting such a message. The CAN driver supports the following flags:

Flag	Description
CAN_MSG_FLAG_EXTD	Message is in Extended Frame Format (29bit ID)
CAN_MSG_FLAG_RTR	Message is a Remote Transmit Request
CAN_MSG_FLAG_SS	Transmit message using Single Shot Transmission (Message will not be retransmitted upon error or loss of arbitration)
CAN_MSG_FLAG_SELF	Transmit message using Self Reception Request (Transmitted message will also be received by the same node)
CAN_MSG_FLAG_DLC_NONE	Message's Data length code is larger than 8. This will break compliance with CAN2.0B

**注解:** The CAN\_MSG\_FLAG\_NONE flag can be used for Standard Frame Format messages

## Examples

**Configuration & Installation** The following code snippet demonstrates how to configure, install, and start the CAN driver via the use of the various configuration structures, macro initializers, the `can_driver_install()` function, and the `can_start()` function.

```
#include "driver/gpio.h"
#include "driver/can.h"

void app_main()
{
    //Initialize configuration structures using macro initializers
    can_general_config_t g_config = CAN_GENERAL_CONFIG_DEFAULT(GPIO_NUM_21, GPIO_
    ↪NUM_22, CAN_MODE_NORMAL);
    can_timing_config_t t_config = CAN_TIMING_CONFIG_500KBITS();
    can_filter_config_t f_config = CAN_FILTER_CONFIG_ACCEPT_ALL();

    //Install CAN driver
    if (can_driver_install(&g_config, &t_config, &f_config) == ESP_OK) {
        printf("Driver installed\n");
    } else {
        printf("Failed to install driver\n");
        return;
    }

    //Start CAN driver
    if (can_start() == ESP_OK) {
        printf("Driver started\n");
    } else {
        printf("Failed to start driver\n");
        return;
    }

    ...
}
```

The usage of macro initializers are 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 `can_message_t` type and `can_transmit()` function.

```
#include "driver/can.h"
```

(下页继续)

(续上页)

```
...

//Configure message to transmit
can_message_t message;
message.identifier = 0xAAAA;
message.flags = CAN_MSG_FLAG_EXTD;
message.data_length_code = 4;
for (int i = 0; i < 4; i++) {
    message.data[i] = 0;
}

//Queue message for transmission
if (can_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 `can_message_t` type and `can_receive()` function.

```
#include "driver/can.h"

...

//Wait for message to be received
can_message_t message;
if (can_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.flags & CAN_MSG_FLAG_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.flags & CAN_MSG_FLAG_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 CAN driver alerts via the use of the `can_reconfigure_alerts()` and `can_read_alerts()` functions.

```
#include "driver/can.h"

...

//Reconfigure alerts to detect Error Passive and Bus-Off error states
uint32_t alerts_to_enable = CAN_ALERT_ERR_PASS | CAN_ALERT_BUS_OFF;
if (can_reconfigure_alerts(alerts_to_enable, NULL) == ESP_OK) {
    printf("Alerts reconfigured\n");
}
```

(下页继续)

```

} else {
    printf("Failed to reconfigure alerts");
}

//Block indefinitely until an alert occurs
uint32_t alerts_triggered;
can_read_alerts(&alerts_triggered, portMAX_DELAY);

```

**Stop and Uninstall** The following code demonstrates how to stop and uninstall the CAN driver via the use of the `can_stop()` and `can_driver_uninstall()` functions.

```

#include "driver/can.h"

...

//Stop the CAN driver
if (can_stop() == ESP_OK) {
    printf("Driver stopped\n");
} else {
    printf("Failed to stop driver\n");
    return;
}

//Uninstall the CAN driver
if (can_driver_uninstall() == ESP_OK) {
    printf("Driver uninstalled\n");
} else {
    printf("Failed to uninstall driver\n");
    return;
}

```

**Multiple ID Filter Configuration** The acceptance mask in `can_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 CAN Network example demonstrates communication between two ESP32s using the CAN driver API. One CAN node acts as a network master initiate and ceasing the transfer of a data from another CAN node acting as a network slave. The example can be found via [peripherals/can/can\\_network](#).

**Alert and Recovery Example:** This example demonstrates how to use the CAN driver's alert and bus recovery API. The example purposely introduces errors on the CAN bus to put the CAN 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/can/can\\_alert\\_and\\_recovery](#).

**Self Test Example:** This example uses the No Acknowledge Mode and Self Reception Request to cause the CAN controller to send and simultaneously receive a series of messages. This example can be used to verify if the connections between the CAN controller and the external transceiver are working correctly. The example can be found via [peripherals/can/can\\_self\\_test](#).

## API Reference

### Header File

- [driver/include/driver/can.h](#)

### Functions

*esp\_err\_t* **can\_driver\_install** (**const** *can\_general\_config\_t* \*g\_config, **const** *can\_timing\_config\_t* \*t\_config, **const** *can\_filter\_config\_t* \*f\_config)

Install CAN driver.

This function installs the CAN driver using three configuration structures. The required memory is allocated and the CAN driver is placed in the stopped state after running this function.

**Note** Macro initializers are available for the configuration structures (see documentation)

**Note** To reinstall the CAN driver, call `can_driver_uninstall()` first

#### Return

- ESP\_OK: Successfully installed CAN driver
- ESP\_ERR\_INVALID\_ARG: Arguments are invalid
- ESP\_ERR\_NO\_MEM: Insufficient memory
- ESP\_ERR\_INVALID\_STATE: Driver is already installed

#### Parameters

- [in] g\_config: General configuration structure
- [in] t\_config: Timing configuration structure
- [in] f\_config: Filter configuration structure

*esp\_err\_t* **can\_driver\_uninstall** ()

Uninstall the CAN driver.

This function uninstalls the CAN 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.

**Warning** The application must ensure that no tasks are blocked on TX/RX queues or alerts when this function is called.

#### Return

- ESP\_OK: Successfully uninstalled CAN driver
- ESP\_ERR\_INVALID\_STATE: Driver is not in stopped/bus-off state, or is not installed

*esp\_err\_t* **can\_start** ()

Start the CAN driver.

This function starts the CAN driver, putting the CAN driver into the running state. This allows the CAN driver to participate in CAN bus activities such as transmitting/receiving messages. The RX queue is reset in this function, clearing any unread messages. This function can only be called when the CAN driver is in the stopped state.

#### Return

- ESP\_OK: CAN driver is now running
- ESP\_ERR\_INVALID\_STATE: Driver is not in stopped state, or is not installed

*esp\_err\_t* **can\_stop** ()

Stop the CAN driver.

This function stops the CAN driver, preventing any further message from being transmitted or received until `can_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 CAN driver is in the running state.

**Warning** A message currently being transmitted/received on the CAN bus will be ceased immediately. This may lead to other CAN nodes interpreting the unfinished message as an error.

#### Return

- ESP\_OK: CAN driver is now Stopped
- ESP\_ERR\_INVALID\_STATE: Driver is not in running state, or is not installed

*esp\_err\_t* **can\_transmit** (*const can\_message\_t* \*message, TickType\_t ticks\_to\_wait)

Transmit a CAN message.

This function queues a CAN 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 timesout. 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 CAN driver is in the running state and cannot be called under Listen Only Mode.

**Note** This function does not guarantee that the transmission is successful. The TX\_SUCCESS/TX\_FAILED alert can be enabled to alert the application upon the success/failure of a transmission.

**Note** The TX\_IDLE alert can be used to alert the application when no other messages are awaiting transmission.

**Return**

- 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: CAN driver is not in running state, or is not installed
- ESP\_ERR\_NOT\_SUPPORTED: Listen Only Mode does not support transmissions

**Parameters**

- [in] message: Message to transmit
- [in] ticks\_to\_wait: Number of FreeRTOS ticks to block on the TX queue

*esp\_err\_t* **can\_receive** (*can\_message\_t* \*message, TickType\_t ticks\_to\_wait)

Receive a CAN 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

**Warning** The flags field of the received message should be checked to determine if the received message contains any data bytes.

**Return**

- 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: CAN driver is not installed

**Parameters**

- [out] message: Received message
- [in] ticks\_to\_wait: Number of FreeRTOS ticks to block on RX queue

*esp\_err\_t* **can\_read\_alerts** (*uint32\_t* \*alerts, TickType\_t ticks\_to\_wait)

Read CAN driver alerts.

This function will read the alerts raised by the CAN driver. If no alert has been when this function is called, this function will block until an alert occurs or until it timeouts.

**Note** Multiple alerts can be raised simultaneously. The application should check for all alerts that have been enabled.

**Return**

- ESP\_OK: Alerts read
- ESP\_ERR\_TIMEOUT: Timed out waiting for alerts
- ESP\_ERR\_INVALID\_ARG: Arguments are invalid
- ESP\_ERR\_INVALID\_STATE: CAN driver is not installed

**Parameters**

- [out] alerts: Bit field of raised alerts (see documentation for alert flags)
- [in] ticks\_to\_wait: Number of FreeRTOS ticks to block for alert

*esp\_err\_t* **can\_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.

**Return**

- ESP\_OK: Alerts reconfigured
- ESP\_ERR\_INVALID\_STATE: CAN driver is not installed

**Parameters**

- [in] `alerts_enabled`: Bit field of alerts to enable (see documentation for alert flags)
- [out] `current_alerts`: Bit field of currently raised alerts. Set to NULL if unused

**`esp_err_t can_initiate_recovery()`**

Start the bus recovery process.

This function initiates the bus recovery process when the CAN driver is in the bus-off state. Once initiated, the CAN driver will enter the recovering state and wait for 128 occurrences of the bus-free signal on the CAN bus before returning to the stopped state. This function will reset the TX queue, clearing any messages pending transmission.

**Note** The BUS\_RECOVERED alert can be enabled to alert the application when the bus recovery process completes.

**Return**

- ESP\_OK: Bus recovery started
- ESP\_ERR\_INVALID\_STATE: CAN driver is not in the bus-off state, or is not installed

**`esp_err_t can_get_status_info(can_status_info_t *status_info)`**

Get current status information of the CAN driver.

**Return**

- ESP\_OK: Status information retrieved
- ESP\_ERR\_INVALID\_ARG: Arguments are invalid
- ESP\_ERR\_INVALID\_STATE: CAN driver is not installed

**Parameters**

- [out] `status_info`: Status information

**`esp_err_t can_clear_transmit_queue()`**

Clear the transmit queue.

This function will clear the transmit queue of all messages.

**Note** The transmit queue is automatically cleared when `can_stop()` or `can_initiate_recovery()` is called.

**Return**

- ESP\_OK: Transmit queue cleared
- ESP\_ERR\_INVALID\_STATE: CAN driver is not installed or TX queue is disabled

**`esp_err_t can_clear_receive_queue()`**

Clear the receive queue.

This function will clear the receive queue of all messages.

**Note** The receive queue is automatically cleared when `can_start()` is called.

**Return**

- ESP\_OK: Transmit queue cleared
- ESP\_ERR\_INVALID\_STATE: CAN driver is not installed

**Structures****`struct can_general_config_t`**

Structure for general configuration of the CAN driver.

**Note** Macro initializers are available for this structure

**Public Members****`can_mode_t mode`**

Mode of CAN 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_CAN_ISR_IN_IRAM` option should be enabled first.

**struct can\_timing\_config\_t**  
Structure for bit timing configuration of the CAN driver.

**Note** Macro initializers are available for this structure

### Public Members

`uint32_t brp`  
Baudrate prescaler (i.e., APB clock divider) can be any even number from 2 to 128. For ESP32 Rev 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 CAN controller samples a bit

**struct can\_filter\_config\_t**  
Structure for acceptance filter configuration of the CAN driver (see documentation)

**Note** 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)



**struct can\_status\_info\_t**

Structure to store status information of CAN driver.

**Public Members****can\_state\_t state**

Current state of CAN 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

**uint32\_t arb\_lost\_count**

Number of instances arbitration was lost

**uint32\_t bus\_error\_count**

Number of instances a bus error has occurred

**struct can\_message\_t**

Structure to store a CAN message.

**Note** The flags member is used to control the message type, and transmission type (see documentation for message flags)

**Public Members****uint32\_t flags**

Bit field of message flags indicates frame/transmission type (see documentation)

**uint32\_t identifier**

11 or 29 bit identifier

**uint8\_t data\_length\_code**

Data length code

**uint8\_t data[CAN\_MAX\_DATA\_LEN]**

Data bytes (not relevant in RTR frame)

**Enumerations****enum can\_mode\_t**

CAN driver operating modes.

*Values:*

**CAN\_MODE\_NORMAL**

Normal operating mode where CAN controller can send/receive/acknowledge messages

**CAN\_MODE\_NO\_ACK**

Transmission does not require acknowledgment. Use this mode for self testing

**CAN\_MODE\_LISTEN\_ONLY**

The CAN controller will not influence the bus (No transmissions or acknowledgments) but can receive messages

**enum can\_state\_t**

CAN driver states.

*Values:*

**CAN\_STATE\_STOPPED**

Stopped state. The CAN controller will not participate in any CAN bus activities

**CAN\_STATE\_RUNNING**

Running state. The CAN controller can transmit and receive messages

**CAN\_STATE\_BUS\_OFF**

Bus-off state. The CAN controller cannot participate in bus activities until it has recovered

**CAN\_STATE\_RECOVERING**

Recovering state. The CAN controller is undergoing bus recovery

### 2.3.3 Digital To Analog Converter

#### Overview

ESP32 has two 8-bit DAC (digital to analog converter) channels, connected to GPIO25 (Channel 1) and GPIO26 (Channel 2).

The DAC driver allows these channels to be set to arbitrary voltages.

The DAC channels can also be driven with DMA-style written sample data, via the *I2S driver* when using the “built-in DAC mode” .

For other analog output options, see the *Sigma-delta Modulation module* and the *LED Control module*. Both these modules produce high frequency PWM output, which can be hardware low-pass filtered in order to generate a lower frequency analog output.

#### Application Example

Setting DAC channel 1 (GPIO 25) voltage to approx 0.78 of VDD\_A voltage ( $VDD * 200 / 255$ ). For VDD\_A 3.3V, this is 2.59V:

```
#include <driver/dac.h>

...

dac_output_enable(DAC_CHANNEL_1);
dac_output_voltage(DAC_CHANNEL_1, 200);
```

#### API Reference

##### Header File

- [driver/include/driver/dac.h](#)

##### Functions

*esp\_err\_t* **dac\_pad\_get\_io\_num** (*dac\_channel\_t* channel, *gpio\_num\_t* \*gpio\_num)

Get the gpio number of a specific DAC channel.

##### Return

- ESP\_OK if success

- ESP\_ERR\_INVALID\_ARG if channel not valid

**Parameters**

- `channel`: Channel to get the gpio number
- `gpio_num`: output buffer to hold the gpio number

*esp\_err\_t* **dac\_output\_voltage** (*dac\_channel\_t* channel, uint8\_t dac\_value)

Set DAC output voltage.

DAC output is 8-bit. Maximum (255) corresponds to VDD.

**Note** Need to configure DAC pad before calling this function. DAC channel 1 is attached to GPIO25, DAC channel 2 is attached to GPIO26

**Return**

- ESP\_OK success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- `channel`: DAC channel
- `dac_value`: DAC output value

*esp\_err\_t* **dac\_output\_enable** (*dac\_channel\_t* channel)

DAC pad output enable.

**Note** DAC channel 1 is attached to GPIO25, DAC channel 2 is attached to GPIO26 I2S left channel will be mapped to DAC channel 2 I2S right channel will be mapped to DAC channel 1

**Parameters**

- `channel`: DAC channel

*esp\_err\_t* **dac\_output\_disable** (*dac\_channel\_t* channel)

DAC pad output disable.

**Note** DAC channel 1 is attached to GPIO25, DAC channel 2 is attached to GPIO26

**Parameters**

- `channel`: DAC channel

*esp\_err\_t* **dac\_i2s\_enable** ()

Enable DAC output data from I2S.

*esp\_err\_t* **dac\_i2s\_disable** ()

Disable DAC output data from I2S.

**Enumerations**

**enum** `dac_channel_t`

Values:

**DAC\_CHANNEL\_1** = 1  
DAC channel 1 is GPIO25

**DAC\_CHANNEL\_2**  
DAC channel 2 is GPIO26

**DAC\_CHANNEL\_MAX**

**GPIO Lookup Macros** Some useful macros can be used to specified the GPIO number of a DAC channel, or vice versa. e.g.

1. `DAC_CHANNEL_1_GPIO_NUM` is the GPIO number of channel 1 (25);
2. `DAC_GPIO26_CHANNEL` is the channel number of GPIO 26 (channel 2).

**Header File**

- `soc/esp32/include/soc/dac_channel.h`

**Macros**

`DAC_GPIO25_CHANNEL`  
`DAC_CHANNEL_1_GPIO_NUM`  
`DAC_GPIO26_CHANNEL`  
`DAC_CHANNEL_2_GPIO_NUM`

**2.3.4 GPIO & RTC GPIO****Overview**

The ESP32 chip features 40 physical GPIO pads. Some GPIO pads cannot be used or do not have the corresponding pin on the chip package (refer to technical reference manual). Each pad can be used as a general purpose I/O or can be connected to an internal peripheral signal.

- Note that GPIO6-11 are usually used for SPI flash.
- GPIO34-39 can only be set as input mode and do not have software pullup or pulldown functions.

There is also separate “RTC GPIO” support, which functions when GPIOs are routed to the “RTC” low-power and analog subsystem. These pin functions can be used when in deep sleep, when the *Ultra Low Power co-processor* is running, or when analog functions such as ADC/DAC/etc are in use.

**Application Example**

GPIO output and input interrupt example: [peripherals/gpio](#).

**API Reference - Normal GPIO****Header File**

- [driver/include/driver/gpio.h](#)

**Functions**

`esp_err_t gpio_config (const gpio_config_t *pGPIOConfig)`

GPIO common configuration.

Configure GPIO's Mode, pull-up, PullDown, IntrType

**Return**

- ESP\_OK success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- pGPIOConfig: Pointer to GPIO configure struct

`esp_err_t gpio_reset_pin (gpio_num_t gpio_num)`

Reset an gpio to default state (select gpio function, enable pullup and disable input and output).

**Note** This function also configures the IOMUX for this pin to the GPIO function, and disconnects any other peripheral output configured via GPIO Matrix.

**Return** Always return ESP\_OK.

**Parameters**

- gpio\_num: GPIO number.

`esp_err_t gpio_set_intr_type (gpio_num_t gpio_num, gpio_int_type_t intr_type)`

GPIO set interrupt trigger type.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**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`

*esp\_err\_t* **gpio\_intr\_enable** (*gpio\_num\_t* *gpio\_num*)

Enable GPIO module interrupt signal.

**Note** Please do not use the interrupt of GPIO36 and GPIO39 when using ADC or Wi-Fi with sleep mode enabled. Please refer to the comments of `adc1_get_raw`. Please refer to section 3.11 of ‘ECO\_and\_Workarounds\_for\_Bugs\_in\_ESP32’ for the description of this issue. As a workaround, call `adc_power_acquire()` in the app. This will result in higher power consumption (by ~1mA), but will remove the glitches on GPIO36 and GPIO39.

**Return**

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error

**Parameters**

- `gpio_num`: GPIO number. If you want to enable an interrupt on e.g. GPIO16, `gpio_num` should be `GPIO_NUM_16` (16);

*esp\_err\_t* **gpio\_intr\_disable** (*gpio\_num\_t* *gpio\_num*)

Disable GPIO module interrupt signal.

**Return**

- `ESP_OK` success
- `ESP_ERR_INVALID_ARG` Parameter error

**Parameters**

- `gpio_num`: GPIO number. If you want to disable the interrupt of e.g. GPIO16, `gpio_num` should be `GPIO_NUM_16` (16);

*esp\_err\_t* **gpio\_set\_level** (*gpio\_num\_t* *gpio\_num*, *uint32\_t* *level*)

GPIO set output level.

**Return**

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` GPIO number error

**Parameters**

- `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

*int* **gpio\_get\_level** (*gpio\_num\_t* *gpio\_num*)

GPIO get input level.

**Warning** If the pad is not configured for input (or input and output) the returned value is always 0.

**Return**

- 0 the GPIO input level is 0
- 1 the GPIO input level is 1

**Parameters**

- `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);

*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`

**Return**

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` GPIO error

**Parameters**

- `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\_err\_t* **gpio\_set\_pull\_mode** (*gpio\_num\_t* gpio\_num, *gpio\_pull\_mode\_t* pull)

Configure GPIO pull-up/pull-down resistors.

Only pins that support both input & output have integrated pull-up and pull-down resistors. Input-only GPIOs 34-39 do not.

#### Return

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG : Parameter error

#### Parameters

- 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.

*esp\_err\_t* **gpio\_wakeup\_enable** (*gpio\_num\_t* gpio\_num, *gpio\_int\_type\_t* intr\_type)

Enable GPIO wake-up function.

#### Return

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

#### Parameters

- gpio\_num: GPIO number.
- intr\_type: GPIO wake-up type. Only GPIO\_INTR\_LOW\_LEVEL or GPIO\_INTR\_HIGH\_LEVEL can be used.

*esp\_err\_t* **gpio\_wakeup\_disable** (*gpio\_num\_t* gpio\_num)

Disable GPIO wake-up function.

#### Return

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

#### Parameters

- gpio\_num: GPIO number

*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*.

#### Parameters

- fn: Interrupt 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.
- arg: Parameter for handler function
- handle: Pointer to return handle. If non-NULL, a handle for the interrupt will be returned here.

#### Return

- 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.

#### Return

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

#### Parameters

- gpio\_num: GPIO number

*esp\_err\_t* **gpio\_pullup\_dis** (*gpio\_num\_t* *gpio\_num*)

Disable pull-up on GPIO.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- *gpio\_num*: GPIO number

*esp\_err\_t* **gpio\_pulldown\_en** (*gpio\_num\_t* *gpio\_num*)

Enable pull-down on GPIO.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- *gpio\_num*: GPIO number

*esp\_err\_t* **gpio\_pulldown\_dis** (*gpio\_num\_t* *gpio\_num*)

Disable pull-down on GPIO.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- *gpio\_num*: GPIO number

*esp\_err\_t* **gpio\_install\_isr\_service** (int *intr\_alloc\_flags*)

Install the driver's GPIO 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.

**Return**

- 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

**Parameters**

- *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.

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.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_STATE Wrong state, the ISR service has not been initialized.
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- *gpio\_num*: GPIO number

- `isr_handler`: ISR handler function for the corresponding GPIO number.
- `args`: parameter for ISR handler.

*esp\_err\_t* `gpio_isr_handler_remove` (*gpio\_num\_t* `gpio_num`)

Remove ISR handler for the corresponding GPIO pin.

**Return**

- `ESP_OK` Success
- `ESP_ERR_INVALID_STATE` Wrong state, the ISR service has not been initialized.
- `ESP_ERR_INVALID_ARG` Parameter error

**Parameters**

- `gpio_num`: GPIO number

*esp\_err\_t* `gpio_set_drive_capability` (*gpio\_num\_t* `gpio_num`, *gpio\_drive\_cap\_t* `strength`)

Set GPIO pad drive capability.

**Return**

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error

**Parameters**

- `gpio_num`: GPIO number, only support output GPIOs
- `strength`: Drive capability of the pad

*esp\_err\_t* `gpio_get_drive_capability` (*gpio\_num\_t* `gpio_num`, *gpio\_drive\_cap\_t* \*`strength`)

Get GPIO pad drive capability.

**Return**

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error

**Parameters**

- `gpio_num`: GPIO number, only support output GPIOs
- `strength`: Pointer to accept drive capability of the pad

*esp\_err\_t* `gpio_hold_en` (*gpio\_num\_t* `gpio_num`)

Enable gpio pad hold function.

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: the input value read will not change, regardless the changes of input signal.

The state of digital gpio cannot be held during Deep-sleep, and it will resume the hold function 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.

**Return**

- `ESP_OK` Success
- `ESP_ERR_NOT_SUPPORTED` Not support pad hold function

**Parameters**

- `gpio_num`: GPIO number, only support output-capable GPIOs

*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`.

**Return**

- `ESP_OK` Success
- `ESP_ERR_NOT_SUPPORTED` Not support pad hold function



**Parameters**

- `gpio_num`: GPIO number, only support output-capable GPIOs

void **gpio\_deep\_sleep\_hold\_en** (void)

Enable all digital gpio pad hold function during Deep-sleep.

When the chip is in Deep-sleep mode, all digital gpio will hold the state before sleep, and when the chip is woken up, the status of digital gpio will not be held. Note that the pad hold feature only works when the chip is in Deep-sleep mode, when not in sleep mode, the digital gpio state can be changed even you have called this function.

Power down or call `gpio_hold_dis` will disable this function, otherwise, the digital gpio hold feature works as long as the chip enter Deep-sleep.

void **gpio\_deep\_sleep\_hold\_dis** (void)

Disable all digital gpio pad 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.

**Parameters**

- `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.

**Parameters**

- `gpio_num`: `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 inversed, otherwise False.

**Structures**

**struct gpio\_config\_t**

Configuration parameters of GPIO pad for `gpio_config` function.

**Public Members**

uint64\_t **pin\_bit\_mask**

GPIO pin: set with bit mask, 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\_SEL\_0**

Pin 0 selected

**GPIO\_SEL\_1**

Pin 1 selected

**GPIO\_SEL\_2**

Pin 2 selected

**Note** There are more macros like that up to pin 39, excluding pins 20, 24 and 28..31. They are not shown here to reduce redundant information.

**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

**Type Definitions**

```
typedef void (*gpio_isr_t) (void *)
```

```
typedef intr_handle_t gpio_isr_handle_t
```

**Enumerations****enum gpio\_num\_t***Values:***GPIO\_NUM\_NC** = -1

Use to signal not connected to S/W

**GPIO\_NUM\_0** = 0

GPIO0, input and output

**GPIO\_NUM\_1** = 1

GPIO1, input and output

**GPIO\_NUM\_2** = 2

GPIO2, input and output

**Note** There are more enumerations like that up to GPIO39, excluding GPIO20, GPIO24 and GPIO28..31. They are not shown here to reduce redundant information.

**Note** GPIO34..39 are input mode only.

**enum gpio\_int\_type\_t***Values:***GPIO\_INTR\_DISABLE** = 0

Disable GPIO interrupt

**GPIO\_INTR\_POSEDGE** = 1

GPIO interrupt type : rising edge

**GPIO\_INTR\_NEGEDGE** = 2

GPIO interrupt type : falling edge

**GPIO\_INTR\_ANYEDGE** = 3

GPIO interrupt type : both rising and falling edge

**GPIO\_INTR\_LOW\_LEVEL** = 4

GPIO interrupt type : input low level trigger

**GPIO\_INTR\_HIGH\_LEVEL** = 5

GPIO interrupt type : input high level trigger

**GPIO\_INTR\_MAX****enum gpio\_mode\_t***Values:***GPIO\_MODE\_DISABLE** = GPIO\_MODE\_DEF\_DISABLE

GPIO mode : disable input and output

**GPIO\_MODE\_INPUT** = GPIO\_MODE\_DEF\_INPUT

GPIO mode : input only

**GPIO\_MODE\_OUTPUT** = GPIO\_MODE\_DEF\_OUTPUT

GPIO mode : output only mode

**GPIO\_MODE\_OUTPUT\_OD** = ((GPIO\_MODE\_DEF\_OUTPUT) | (GPIO\_MODE\_DEF\_OD))

GPIO mode : output only with open-drain mode

**GPIO\_MODE\_INPUT\_OUTPUT\_OD** = ((GPIO\_MODE\_DEF\_INPUT) | (GPIO\_MODE\_DEF\_OUTPUT) | (GPIO\_MODE\_

GPIO mode : output and input with open-drain mode

**GPIO\_MODE\_INPUT\_OUTPUT** = ((GPIO\_MODE\_DEF\_INPUT) | (GPIO\_MODE\_DEF\_OUTPUT))

GPIO mode : output and input mode

**enum gpio\_pullup\_t**

*Values:*

**GPIO\_PULLUP\_DISABLE** = 0x0

Disable GPIO pull-up resistor

**GPIO\_PULLUP\_ENABLE** = 0x1

Enable GPIO pull-up resistor

**enum gpiopulldown\_t**

*Values:*

**GPIO\_PULLDOWN\_DISABLE** = 0x0

Disable GPIO pull-down resistor

**GPIO\_PULLDOWN\_ENABLE** = 0x1

Enable GPIO pull-down resistor

**enum gpio\_pull\_mode\_t**

*Values:*

**GPIO\_PULLUP\_ONLY**

Pad pull up

**GPIO\_PULLDOWN\_ONLY**

Pad pull down

**GPIO\_PULLUP\_PULLDOWN**

Pad pull up + pull down

**GPIO\_FLOATING**

Pad floating

**enum gpio\_drive\_cap\_t**

*Values:*

**GPIO\_DRIVE\_CAP\_0** = 0

Pad drive capability: weak

**GPIO\_DRIVE\_CAP\_1** = 1

Pad drive capability: stronger

**GPIO\_DRIVE\_CAP\_2** = 2

Pad drive capability: default value

**GPIO\_DRIVE\_CAP\_DEFAULT** = 2

Pad drive capability: default value

**GPIO\_DRIVE\_CAP\_3** = 3

Pad drive capability: strongest

**GPIO\_DRIVE\_CAP\_MAX**

## API Reference - RTC GPIO

### Header File

- [driver/include/driver/rtc\\_io.h](#)

### Functions

**static bool** `rtc_gpio_is_valid_gpio` (*gpio\_num\_t* gpio\_num)

Determine if the specified GPIO is a valid RTC GPIO.

**Return** true if GPIO is valid for RTC GPIO use. false otherwise.

**Parameters**

- `gpio_num`: GPIO number

*esp\_err\_t* `rtc_gpio_init` (*gpio\_num\_t* gpio\_num)

Init a GPIO as RTC GPIO.

This function must be called when initializing a pad for an analog function.

**Return**

- `ESP_OK` success
- `ESP_ERR_INVALID_ARG` GPIO is not an RTC IO

**Parameters**

- `gpio_num`: GPIO number (e.g. `GPIO_NUM_12`)

*esp\_err\_t* `rtc_gpio_deinit` (*gpio\_num\_t* gpio\_num)

Init a GPIO as digital GPIO.

**Return**

- `ESP_OK` success
- `ESP_ERR_INVALID_ARG` GPIO is not an RTC IO

**Parameters**

- `gpio_num`: GPIO number (e.g. `GPIO_NUM_12`)

*uint32\_t* `rtc_gpio_get_level` (*gpio\_num\_t* gpio\_num)

Get the RTC IO input level.

**Return**

- 1 High level
- 0 Low level
- `ESP_ERR_INVALID_ARG` GPIO is not an RTC IO

**Parameters**

- `gpio_num`: GPIO number (e.g. `GPIO_NUM_12`)

*esp\_err\_t* `rtc_gpio_set_level` (*gpio\_num\_t* gpio\_num, *uint32\_t* level)

Set the RTC IO output level.

**Return**

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` GPIO is not an RTC IO

**Parameters**

- `gpio_num`: GPIO number (e.g. `GPIO_NUM_12`)
- `level`: output level

*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.

**Return**

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` GPIO is not an RTC IO

**Parameters**

- `gpio_num`: GPIO number (e.g. `GPIO_NUM_12`)
- `mode`: GPIO direction

*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.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG GPIO is not an RTC IO

**Parameters**

- `gpio_num`: GPIO number (e.g. GPIO\_NUM\_12)

*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.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG GPIO is not an RTC IO

**Parameters**

- `gpio_num`: GPIO number (e.g. GPIO\_NUM\_12)

*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.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG GPIO is not an RTC IO

**Parameters**

- `gpio_num`: GPIO number (e.g. GPIO\_NUM\_12)

*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.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG GPIO is not an RTC IO

**Parameters**

- `gpio_num`: GPIO number (e.g. GPIO\_NUM\_12)

*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.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG GPIO is not an RTC IO

**Parameters**

- `gpio_num`: GPIO number (e.g. GPIO\_NUM\_12)

*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.

**Return**

- ESP\_OK Success

- ESP\_ERR\_INVALID\_ARG GPIO is not an RTC IO

**Parameters**

- `gpio_num`: GPIO number (e.g. GPIO\_NUM\_12)

*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.

**Return**

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG if GPIO is not an RTC IO

**Parameters**

- `gpio_num`: GPIO number (e.g. GPIO\_NUM\_12).

`void rtc_gpio_force_hold_dis_all ()`

Disable 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\_set\_drive\_capability (gpio\_num\_t gpio\_num, gpio\_drive\_cap\_t strength)*

Set RTC GPIO pad drive capability.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- `gpio_num`: GPIO number, only support output GPIOs
- `strength`: Drive capability of the pad

*esp\_err\_t rtc\_gpio\_get\_drive\_capability (gpio\_num\_t gpio\_num, gpio\_drive\_cap\_t \*strength)*

Get RTC GPIO pad drive capability.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- `gpio_num`: GPIO number, only support output GPIOs
- `strength`: Pointer to accept drive capability of the pad

*esp\_err\_t rtc\_gpio\_wakeup\_enable (gpio\_num\_t gpio\_num, gpio\_intr\_type\_t intr\_type)*

Enable wakeup from sleep mode using specific GPIO.

**Return**

- 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.

**Parameters**

- `gpio_num`: GPIO number
- `intr_type`: Wakeup on high level (GPIO\_INTR\_HIGH\_LEVEL) or low 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.

**Return**

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG if `gpio_num` is not an RTC IO

**Parameters**

- `gpio_num`: GPIO number

## Macros

`RTC_GPIO_IS_VALID_GPIO` (`gpio_num`)

## Enumerations

`enum rtc_gpio_mode_t`

Values:

`RTC_GPIO_MODE_INPUT_ONLY`

Pad input

`RTC_GPIO_MODE_OUTPUT_ONLY`

Pad output

`RTC_GPIO_MODE_INPUT_OUTPUT`

Pad pull input + output

`RTC_GPIO_MODE_DISABLED`

Pad (output + input) disable

## 2.3.5 I2C

An I2C (Inter-Integrated Circuit) bus can be used for communication with several external devices connected to the same bus as ESP32. There are two I2C controllers on board of the ESP32, each of which can be set to master mode or slave mode.

### Overview

The following sections will walk you through typical steps to configure and operate the I2C driver:

1. *Configure Driver* - select driver's parameters like master or slave mode, set specific GPIO pins to act as SDA and SCL, set the clock speed, etc.
2. *Install Driver* - activate driver in master or slave mode to operate on one of the two I2C controllers available on ESP32.
3. *Run Communication*:
  - a) *Master Mode* - run communication acting as a master
  - b) *Slave Mode* - get slave responding to messages from the master
4. *Interrupt Handling* - configure and service I2C interrupts.
5. *Going Beyond Defaults* - adjust timing, pin configuration and other parameters of the I2C communication.
6. *Error Handling* - how to recognize and handle driver configuration and communication errors.
7. *Delete Driver* - on communication end to free resources used by the I2C driver.

The top level identification of an I2C driver is one of the two port numbers selected from `i2c_port_t`. The mode of operation for a given port is provided during driver configuration by selecting either “master” or “slave” from `i2c_mode_t`.

**Configure Driver** The first step to establishing I2C communication is to configure the driver. This is done by setting several parameters contained in `i2c_config_t` structure:

- **I2C mode** - select either slave or master from `i2c_mode_t`
- Settings of the **communication pins**:
  - GPIO pin numbers assigned to the SDA and SCL signals
  - Whether to enable ESP32's internal pull up for respective pins
- **I2C clock speed**, if this configuration concerns the master mode
- If this configuration concerns the slave mode:
  - Whether **10 bit address mode** should be enabled
  - The **slave address**

Then, to initialize configuration for a given I2C port, call function `i2c_param_config()` with the port number and `i2c_config_t` structure as the function call parameters.

At this stage `i2c_param_config()` also sets “behind the scenes” couple of other I2C configuration parameters to commonly used default values. To check what are the values and how to change them, see [Going Beyond Defaults](#).

**Install Driver** Having the configuration initialized, the next step is to install the I2C driver by calling `i2c_driver_install()`. This function call requires the following parameters:

- The port number, one of the two ports available, selected from `i2c_port_t`
- The I2C mode, slave or master, selected from `i2c_mode_t`
- Sizes of buffers that will be allocated for sending and receiving data **in the slave mode**
- Flags used to allocate the interrupt

**Run Communication** With the I2C driver installed, ESP32 is ready to communicate with other I2C devices. Programming of communication depends on whether selected I2C port operates in a master or a slave mode.

**Master Mode** ESP32’s I2C port working in the master mode is responsible for establishing communication with slave I2C devices and sending commands to trigger actions by slaves, like doing a measurement and sending back a result.

To organize this process the driver provides a container, called a “command link”, that should be populated with a sequence of commands and then passed to the I2C controller for execution.

### Master Write

An example of building a command link for I2C master sending n bytes to slave is shown below:

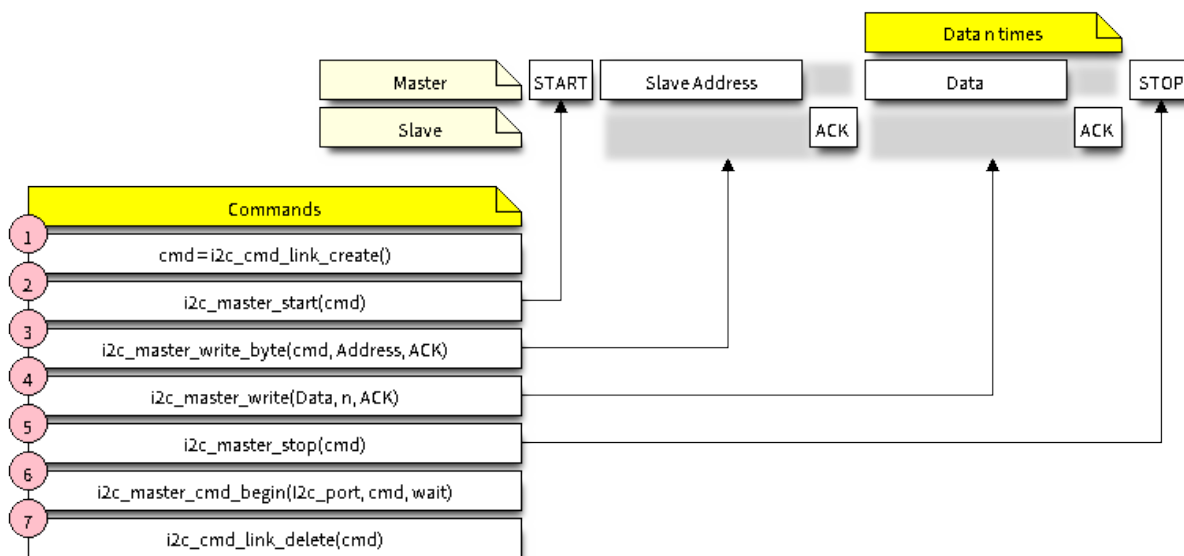


图 12: I2C command link - master write example

The following describes how the command link for a “master write” is set up and what comes inside:

1. The first step is to create a command link with `i2c_cmd_link_create()`. Then the command link is populated with series of data to be sent to the slave:
  2. **Start bit** - `i2c_master_start()`
  3. **Single byte slave address** - `i2c_master_write_byte()`. The address is provided as an argument of this function call.
  4. One or more bytes of **data** as an argument of `i2c_master_write()`.
  5. **Stop bit** - `i2c_master_stop()`

Both `i2c_master_write_byte()` and `i2c_master_write()` commands have additional argument defining whether slave should **acknowledge** received data or not.



6. Execution of command link by I2C controller is triggered by calling `i2c_master_cmd_begin()`.
7. As the last step, after sending of the commands is finished, the resources used by the command link are released by calling `i2c_cmd_link_delete()`.

### Master Read

There is a similar sequence of steps for the master to read the data from a slave.

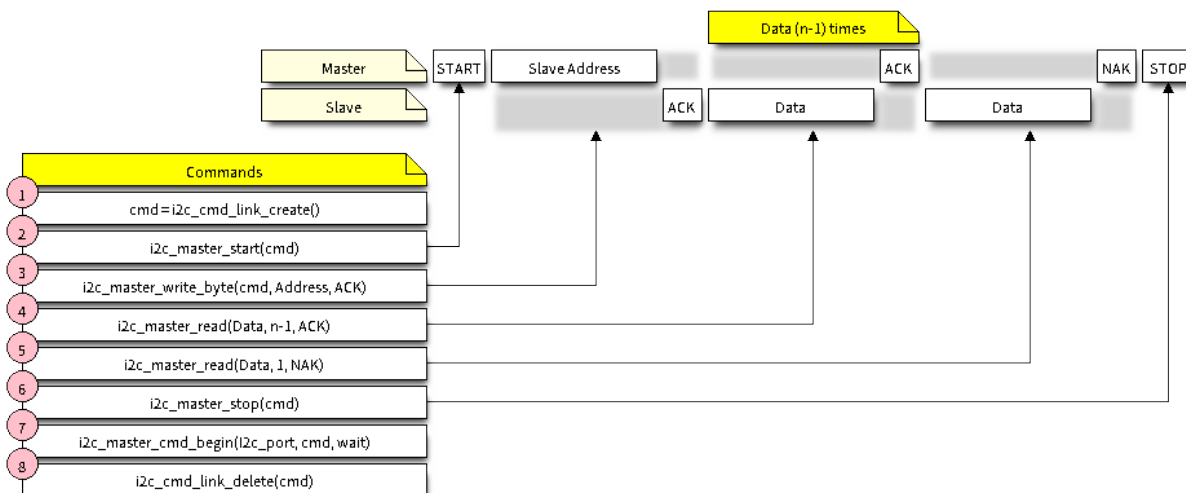


图 13: I2C command link - master read example

When reading the data, instead of “`i2c_master_read ...`”, the command link is populated with `i2c_master_read_byte()` and / or `i2c_master_read()`. Also, the last read is configured for not providing an acknowledge by the master.

### Master Write or Read?

After sending a slave’s address, see step 3 on pictures above, the master either writes to or reads from the slave. The information what the master will actually do is hidden in the least significant bit of the slave’s address.

Therefore the command link instructing the slave that the master will write the data contains the address like `(ESP_SLAVE_ADDR << 1) | I2C_MASTER_WRITE` and looks as follows:

```
i2c_master_write_byte(cmd, (ESP_SLAVE_ADDR << 1) | I2C_MASTER_WRITE, ACK_CHECK_EN)
```

By similar token the command link to read from the slave looks as follows:

```
i2c_master_write_byte(cmd, (ESP_SLAVE_ADDR << 1) | I2C_MASTER_READ, ACK_CHECK_EN)
```

**Slave Mode** The API provides functions to read and write data by the slave - \* `i2c_slave_read_buffer()` and `i2c_slave_write_buffer()`. An example of using these functions is provided in [peripherals/i2c](#).

**Interrupt Handling** To register an interrupt handler, call function `i2c_isr_register()`, to delete the handler call `i2c_isr_free()`. Description of interrupts triggered by I2C controller is provided in the [ESP32 Technical Reference Manual \(PDF\)](#).

**Going Beyond Defaults** There are couple of I2C communication parameters setup during driver configuration (when calling `i2c_param_config()`, see [Configure Driver](#)), to some default commonly used values. Some parameters are also already configured in registers of the I2C controller. These parameters can be changed to user defined values by calling dedicated functions:

- Period of SCL pulses being high and low - `i2c_set_period()`
- SCL and SDA signal timing used during generation of start / stop signals - `i2c_set_start_timing()` / `i2c_set_stop_timing()`

- Timing relationship between SCL and SDA signals when sampling by slave, as well as when transmitting by master - `i2c_set_data_timing()`
- I2C timeout - `i2c_set_timeout()`

---

**注解:** The timing values are defined in APB clock cycles. The frequency of APB is specified in `I2C_APB_CLK_FREQ`.

---

- What bit, LSB or MSB, is transmitted / received first - `i2c_set_data_mode()` selectable out of modes defined in `i2c_trans_mode_t`

Each one of the above functions has a `_get_` counterpart to check the currently set value.

To see the default values of parameters setup during driver configuration, please refer to file `driver/i2c.c` looking up defines with `_DEFAULT` suffix.

With function `i2c_set_pin()` it is also possible to select different SDA and SCL pins and alter configuration of pull ups, changing what has been already entered with `i2c_param_config()`.

---

**注解:** ESP32's internal pull ups are in the range of some tens of kOhm, and as such in most cases insufficient for use as I2C pull ups by themselves. We suggest to add external pull ups as well, with values as described in the I2C standard.

---

**Error Handling** Most of driver's function return the `ESP_OK` on successful completion or a specific error code on a failure. It is a good practice to always check the returned values and implement the error handling. The driver is also printing out log messages, when e.g. checking the correctness of entered configuration, that contain explanation of errors. For details please refer to file `driver/i2c.c` looking up defines with `_ERR_STR` suffix.

Use dedicated interrupts to capture communication failures. For instance there is `I2C_TIME_OUT_INT` interrupt triggered when I2C takes too long to receive data. See *Interrupt Handling* for related information.

To reset internal hardware buffers in case of communication failure, you can use `i2c_reset_tx_fifo()` and `i2c_reset_rx_fifo()`.

**Delete Driver** If the I2C communication is established with `i2c_driver_install()` for some specific period of time and then not required, the driver may be removed to free allocated resources by calling `i2c_driver_delete()`.

## Application Example

I2C master and slave example: [peripherals/i2c](#).

## API Reference

### Header File

- `driver/include/driver/i2c.h`

### Functions

`esp_err_t 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)`

I2C driver install.

**Note** Only slave mode will use this value, driver will ignore this value in master mode.

**Note** Only slave mode will use this value, driver will ignore this value in master mode.

**Note** 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.

**Return**

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error
- `ESP_FAIL` Driver install error

**Parameters**

- `i2c_num`: I2C port number
- `mode`: I2C mode ( master or slave )
- `slv_rx_buf_len`: receiving buffer size for slave mode

**Parameters**

- `slv_tx_buf_len`: sending buffer size for slave mode

**Parameters**

- `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.

`esp_err_t i2c_driver_delete (i2c_port_t i2c_num)`

I2C driver delete.

**Return**

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error

**Parameters**

- `i2c_num`: I2C port number

`esp_err_t i2c_param_config (i2c_port_t i2c_num, const i2c_config_t *i2c_conf)`

I2C parameter initialization.

**Return**

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error

**Parameters**

- `i2c_num`: I2C port number
- `i2c_conf`: pointer to I2C parameter settings

`esp_err_t i2c_reset_tx_fifo (i2c_port_t i2c_num)`

reset I2C tx hardware fifo

**Return**

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error

**Parameters**

- `i2c_num`: I2C port number

`esp_err_t i2c_reset_rx_fifo (i2c_port_t i2c_num)`

reset I2C rx fifo

**Return**

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error

**Parameters**

- `i2c_num`: I2C port number

`esp_err_t i2c_isr_register (i2c_port_t i2c_num, void (*fn)) void *`  
, void \*arg, int intr\_alloc\_flags, intr\_handle\_t \*handle I2C isr handler register.

**Return**

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error

**Parameters**

- `i2c_num`: I2C port number
- `fn`: isr handler function

- `arg`: parameter for isr 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`: handle return from `esp_intr_alloc`.

*esp\_err\_t* **i2c\_isr\_free** (*intr\_handle\_t* handle)

to delete and free I2C isr.

**Return**

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error

**Parameters**

- `handle`: handle of isr.

*esp\_err\_t* **i2c\_set\_pin** (*i2c\_port\_t* i2c\_num, int sda\_io\_num, int scl\_io\_num, *gpio\_pullup\_t* sda\_pullup\_en, *gpio\_pullup\_t* scl\_pullup\_en, *i2c\_mode\_t* mode)

Configure GPIO signal for I2C sck and sda.

**Return**

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error

**Parameters**

- `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`: Whether to enable the internal pullup for sda pin
- `scl_pullup_en`: Whether to enable the internal pullup for scl pin
- `mode`: I2C mode

*i2c\_cmd\_handle\_t* **i2c\_cmd\_link\_create** ()

Create and init I2C command link.

**Note** Before we build I2C command link, we need to call `i2c_cmd_link_create()` to create a command link. After we finish sending the commands, we need to call `i2c_cmd_link_delete()` to release and return the resources.

**Return** i2c command link handler

void **i2c\_cmd\_link\_delete** (*i2c\_cmd\_handle\_t* cmd\_handle)

Free I2C command link.

**Note** Before we build I2C command link, we need to call `i2c_cmd_link_create()` to create a command link. After we finish sending the commands, we need to call `i2c_cmd_link_delete()` to release and return the resources.

**Parameters**

- `cmd_handle`: I2C command handle

*esp\_err\_t* **i2c\_master\_start** (*i2c\_cmd\_handle\_t* cmd\_handle)

Queue command for I2C master to generate a start signal.

**Note** Only call this function in I2C master mode Call `i2c_master_cmd_begin()` to send all queued commands

**Return**

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error

**Parameters**

- `cmd_handle`: I2C cmd link

*esp\_err\_t* **i2c\_master\_write\_byte** (*i2c\_cmd\_handle\_t* cmd\_handle, uint8\_t data, bool ack\_en)

Queue command for I2C master to write one byte to I2C bus.

**Note** Only call this function in I2C master mode Call `i2c_master_cmd_begin()` to send all queued commands

**Return**

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error

**Parameters**

- `cmd_handle`: I2C cmd link
- `data`: I2C one byte command to write to bus
- `ack_en`: enable ack check for master

`esp_err_t i2c_master_write(i2c_cmd_handle_t cmd_handle, uint8_t *data, size_t data_len, bool ack_en)`

Queue command for I2C master to write buffer to I2C bus.

**Note** Only call this function in I2C master mode Call `i2c_master_cmd_begin()` to send all queued commands

**Note** If the psram is enabled and `intr_flag` is `ESP_INTR_FLAG_IRAM`, please use the memory allocated from internal RAM.

**Return**

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error

**Parameters**

- `cmd_handle`: I2C cmd link
- `data`: data to send

**Parameters**

- `data_len`: data length
- `ack_en`: enable ack check for master

`esp_err_t i2c_master_read_byte(i2c_cmd_handle_t cmd_handle, uint8_t *data, i2c_ack_type_t ack)`

Queue command for I2C master to read one byte from I2C bus.

**Note** Only call this function in I2C master mode Call `i2c_master_cmd_begin()` to send all queued commands

**Note** If the psram is enabled and `intr_flag` is `ESP_INTR_FLAG_IRAM`, please use the memory allocated from internal RAM.

**Return**

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error

**Parameters**

- `cmd_handle`: I2C cmd link
- `data`: pointer accept the data byte

**Parameters**

- `ack`: ack value for read command

`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 command for I2C master to read data from I2C bus.

**Note** Only call this function in I2C master mode Call `i2c_master_cmd_begin()` to send all queued commands

**Note** If the psram is enabled and `intr_flag` is `ESP_INTR_FLAG_IRAM`, please use the memory allocated from internal RAM.

**Return**

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error

**Parameters**

- `cmd_handle`: I2C cmd link
- `data`: data buffer to accept the data from bus

**Parameters**

- `data_len`: read data length
- `ack`: ack value for read command

`esp_err_t i2c_master_stop(i2c_cmd_handle_t cmd_handle)`

Queue command for I2C master to generate a stop signal.

**Note** Only call this function in I2C master mode Call `i2c_master_cmd_begin()` to send all queued commands

**Return**

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error

**Parameters**

- `cmd_handle`: I2C cmd link

*esp\_err\_t* **i2c\_master\_cmd\_begin** (*i2c\_port\_t* *i2c\_num*, *i2c\_cmd\_handle\_t* *cmd\_handle*, TickType\_t *ticks\_to\_wait*)

I2C master send queued commands. This function will trigger sending all queued commands. The task will be blocked until all the commands have been sent out. The I2C APIs are not thread-safe, if you want to use one I2C port in different tasks, you need to take care of the multi-thread issue.

**Note** Only call this function in I2C master mode

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_FAIL Sending command error, slave doesn'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.

**Parameters**

- *i2c\_num*: I2C port number
- *cmd\_handle*: I2C command handler
- *ticks\_to\_wait*: maximum wait ticks.

int **i2c\_slave\_write\_buffer** (*i2c\_port\_t* *i2c\_num*, uint8\_t \**data*, int *size*, TickType\_t *ticks\_to\_wait*)

I2C slave write data to internal ringbuffer, when tx fifo empty, isr will fill the hardware fifo from the internal ringbuffer.

**Note** Only call this function in I2C slave mode

**Return**

- ESP\_FAIL(-1) Parameter error
- Others(>=0) The number of data bytes that pushed to the I2C slave buffer.

**Parameters**

- *i2c\_num*: I2C port number
- *data*: data pointer to write into internal buffer
- *size*: data size
- *ticks\_to\_wait*: Maximum waiting ticks

int **i2c\_slave\_read\_buffer** (*i2c\_port\_t* *i2c\_num*, uint8\_t \**data*, size\_t *max\_size*, TickType\_t *ticks\_to\_wait*)

I2C slave read data from internal buffer. When I2C slave receive data, isr will copy received data from hardware rx fifo to internal ringbuffer. Then users can read from internal ringbuffer.

**Note** Only call this function in I2C slave mode

**Return**

- ESP\_FAIL(-1) Parameter error
- Others(>=0) The number of data bytes that read from I2C slave buffer.

**Parameters**

- *i2c\_num*: I2C port number
- *data*: data pointer to accept data from internal buffer
- *max\_size*: Maximum data size to read
- *ticks\_to\_wait*: Maximum waiting ticks

*esp\_err\_t* **i2c\_set\_period** (*i2c\_port\_t* *i2c\_num*, int *high\_period*, int *low\_period*)  
set I2C master clock period

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- *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

*esp\_err\_t* **i2c\_get\_period** (*i2c\_port\_t* *i2c\_num*, int \**high\_period*, int \**low\_period*)  
get I2C master clock period

**Return**

- ESP\_OK Success

- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- `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

*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 broken. enable hardware filter can filter out high frequency interference and make the master more stable.

**Note** Enable filter will slow the SCL clock.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- `i2c_num`: I2C port number
- `cyc_num`: the APB cycles need to be filtered ( $0 \leq \text{cyc\_num} \leq 7$ ). When the period of a pulse is less than  $\text{cyc\_num} * \text{APB\_cycle}$ , the I2C controller will ignore this pulse.

*esp\_err\_t* **i2c\_filter\_disable** (*i2c\_port\_t* *i2c\_num*)

disable filter on I2C bus

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- `i2c_num`: I2C port number

*esp\_err\_t* **i2c\_set\_start\_timing** (*i2c\_port\_t* *i2c\_num*, *int* *setup\_time*, *int* *hold\_time*)

set I2C master start signal timing

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- `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 num between the falling-edge of SDA and falling-edge of SCL for start mark, it' s a 10-bit value.

*esp\_err\_t* **i2c\_get\_start\_timing** (*i2c\_port\_t* *i2c\_num*, *int* \**setup\_time*, *int* \**hold\_time*)

get I2C master start signal timing

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- `i2c_num`: I2C port number
- `setup_time`: pointer to get setup time
- `hold_time`: pointer to get hold time

*esp\_err\_t* **i2c\_set\_stop\_timing** (*i2c\_port\_t* *i2c\_num*, *int* *setup\_time*, *int* *hold\_time*)

set I2C master stop signal timing

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- `i2c_num`: I2C port number
- `setup_time`: clock num 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.

`esp_err_t i2c_get_stop_timing (i2c_port_t i2c_num, int *setup_time, int *hold_time)`  
get I2C master stop signal timing

**Return**

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error

**Parameters**

- `i2c_num`: I2C port number
- `setup_time`: pointer to get setup time.
- `hold_time`: pointer to get hold time.

`esp_err_t i2c_set_data_timing (i2c_port_t i2c_num, int sample_time, int hold_time)`  
set I2C data signal timing

**Return**

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error

**Parameters**

- `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

`esp_err_t i2c_get_data_timing (i2c_port_t i2c_num, int *sample_time, int *hold_time)`  
get I2C data signal timing

**Return**

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error

**Parameters**

- `i2c_num`: I2C port number
- `sample_time`: pointer to get sample time
- `hold_time`: pointer to get hold time

`esp_err_t i2c_set_timeout (i2c_port_t i2c_num, int timeout)`  
set I2C timeout value

**Return**

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error

**Parameters**

- `i2c_num`: I2C port number
- `timeout`: timeout value for I2C bus (unit: APB 80Mhz clock cycle)

`esp_err_t i2c_get_timeout (i2c_port_t i2c_num, int *timeout)`  
get I2C timeout value

**Return**

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error

**Parameters**

- `i2c_num`: I2C port number
- `timeout`: pointer to get timeout value

`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

**Return**

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error



**Parameters**

- `i2c_num`: I2C port number
- `tx_trans_mode`: I2C sending data mode
- `rx_trans_mode`: I2C receiving data mode

`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

**Return**

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error

**Parameters**

- `i2c_num`: I2C port number
- `tx_trans_mode`: pointer to get I2C sending data mode
- `rx_trans_mode`: pointer to get I2C receiving data mode

**Structures**

`struct i2c_config_t`

I2C initialization parameters.

**Public Members**

`i2c_mode_t mode`

I2C mode

`gpio_num_t sda_io_num`

GPIO number for I2C sda signal

`gpio_pullup_t sda_pullup_en`

Internal GPIO pull mode for I2C sda signal

`gpio_num_t scl_io_num`

GPIO number for I2C scl signal

`gpio_pullup_t 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)

`uint8_t addr_10bit_en`

I2C 10bit address mode enable for slave mode

`uint16_t slave_addr`

I2C address for slave mode

**Macros**

`I2C_APB_CLK_FREQ`

I2C source clock is APB clock, 80MHz

`I2C_FIFO_LEN`

I2C hardware fifo length

**Type Definitions**

`typedef void *i2c_cmd_handle_t`

I2C command handle

**Enumerations****enum i2c\_mode\_t***Values:***I2C\_MODE\_SLAVE** = 0  
I2C slave mode**I2C\_MODE\_MASTER**  
I2C master mode**I2C\_MODE\_MAX****enum i2c\_rw\_t***Values:***I2C\_MASTER\_WRITE** = 0  
I2C write data**I2C\_MASTER\_READ**  
I2C read data**enum i2c\_trans\_mode\_t***Values:***I2C\_DATA\_MODE\_MSB\_FIRST** = 0  
I2C data msb first**I2C\_DATA\_MODE\_LSB\_FIRST** = 1  
I2C data lsb first**I2C\_DATA\_MODE\_MAX****enum i2c\_opmode\_t***Values:***I2C\_CMD\_RESTART** = 0  
I2C restart command**I2C\_CMD\_WRITE**  
I2C write command**I2C\_CMD\_READ**  
I2C read command**I2C\_CMD\_STOP**  
I2C stop command**I2C\_CMD\_END**  
I2C end command**enum i2c\_port\_t***Values:***I2C\_NUM\_0** = 0  
I2C port 0**I2C\_NUM\_1**  
I2C port 1**I2C\_NUM\_MAX****enum i2c\_addr\_mode\_t***Values:***I2C\_ADDR\_BIT\_7** = 0  
I2C 7bit address for slave mode**I2C\_ADDR\_BIT\_10**  
I2C 10bit address for slave mode**I2C\_ADDR\_BIT\_MAX**

**enum i2c\_ack\_type\_t**

Values:

```

I2C_MASTER_ACK = 0x0
    I2C ack for each byte read

I2C_MASTER_NACK = 0x1
    I2C nack for each byte read

I2C_MASTER_LAST_NACK = 0x2
    I2C nack for the last byte

I2C_MASTER_ACK_MAX

```

**2.3.6 I2S****Overview**

ESP32 contains two I2S peripherals. These peripherals can be configured to input and output sample data via the I2S driver.

The I2S peripheral supports DMA meaning it can stream sample data without requiring each sample to be read or written by the CPU.

I2S output can also be routed directly to the Digital/Analog Converter output channels (GPIO 25 & GPIO 26) to produce analog output directly, rather than via an external I2S codec.

---

**注解:** For high accuracy clock applications, APLL clock source can be used with `.use_apll = true` and ESP32 will automatically calculate APLL parameter.

---



---

**注解:** If `use_apll = true` and `fixed_mclk > 0`, then the Master clock output for I2S is fixed and equal to the `fixed_mclk` value. The audio clock rate (LRCK) is always the MCLK divisor and  $0 < \text{MCLK}/\text{LRCK}/\text{channels}/\text{bits\_per\_sample} < 64$

---

**Application Example**

A full I2S example is available in esp-idf: [peripherals/i2s](#).

Short example of I2S configuration:

```

#include "driver/i2s.h"
#include "freertos/queue.h"

static const int i2s_num = 0; // i2s port number

static const i2s_config_t i2s_config = {
    .mode = I2S_MODE_MASTER | I2S_MODE_TX,
    .sample_rate = 44100,
    .bits_per_sample = 16,
    .channel_format = I2S_CHANNEL_FMT_RIGHT_LEFT,
    .communication_format = I2S_COMM_FORMAT_I2S | I2S_COMM_FORMAT_I2S_MSB,
    .intr_alloc_flags = 0, // default interrupt priority
    .dma_buf_count = 8,
    .dma_buf_len = 64,
    .use_apll = false
};

static const i2s_pin_config_t pin_config = {

```

(下页继续)

```

    .bck_io_num = 26,
    .ws_io_num = 25,
    .data_out_num = 22,
    .data_in_num = I2S_PIN_NO_CHANGE
};

...

    i2s_driver_install(i2s_num, &i2s_config, 0, NULL); //install and start i2s_
↳driver

    i2s_set_pin(i2s_num, &pin_config);

    i2s_set_sample_rates(i2s_num, 22050); //set sample rates

    i2s_driver_uninstall(i2s_num); //stop & destroy i2s driver

```

Short example configuring I2S to use internal DAC for analog output:

```

#include "driver/i2s.h"
#include "freertos/queue.h"

static const int i2s_num = 0; // i2s port number

static const i2s_config_t i2s_config = {
    .mode = I2S_MODE_MASTER | I2S_MODE_TX | I2S_MODE_DAC_BUILT_IN,
    .sample_rate = 44100,
    .bits_per_sample = 16, /* the DAC module will only take the 8bits from MSB */
    .channel_format = I2S_CHANNEL_FMT_RIGHT_LEFT,
    .communication_format = I2S_COMM_FORMAT_I2S_MSB,
    .intr_alloc_flags = 0, // default interrupt priority
    .dma_buf_count = 8,
    .dma_buf_len = 64,
    .use_apll = false
};

...

    i2s_driver_install(i2s_num, &i2s_config, 0, NULL); //install and start i2s_
↳driver

    i2s_set_pin(i2s_num, NULL); //for internal DAC, this will enable both of the_
↳internal channels

    //You can call i2s_set_dac_mode to set built-in DAC output mode.
    //i2s_set_dac_mode(I2S_DAC_CHANNEL_BOTH_EN);

    i2s_set_sample_rates(i2s_num, 22050); //set sample rates

    i2s_driver_uninstall(i2s_num); //stop & destroy i2s driver

```

## API Reference

### Header File

- [driver/include/driver/i2s.h](#)

### Functions

`esp_err_t i2s_set_pin(i2s_port_t i2s_num, const i2s_pin_config_t *pin)`  
Set I2S pin number.

Inside the pin configuration structure, set `I2S_PIN_NO_CHANGE` for any pin where the current configuration should not be changed.

**Note** The I2S peripheral output signals can be connected to multiple GPIO pads. However, the I2S peripheral input signal can only be connected to one GPIO pad.

#### Parameters

- `i2s_num`: I2S\_NUM\_0 or I2S\_NUM\_1
- `pin`: I2S Pin structure, or NULL to set 2-channel 8-bit internal DAC pin configuration (GPIO25 & GPIO26)

**Note** if `*pin` is set as NULL, this function will initialize both of the built-in DAC channels by default. if you don't want this to happen and you want to initialize only one of the DAC channels, you can call `i2s_set_dac_mode` instead.

#### Return

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_FAIL IO error

*esp\_err\_t* `i2s_set_pdm_rx_down_sample` (*i2s\_port\_t* *i2s\_num*, *i2s\_pdm\_dsr\_t* *dsr*)

Set PDM mode down-sample rate In PDM RX mode, there would be 2 rounds of downsample process in hardware. In the first downsample process, the sampling number can be 16 or 8. In the second downsample process, the sampling number is fixed as 8. So the clock frequency in PDM RX mode would be ( $f_{pcm} * 64$ ) or ( $f_{pcm} * 128$ ) accordingly.

**Note** After calling this function, it would call `i2s_set_clk` inside to update the clock frequency. Please call this function after I2S driver has been initialized.

#### Return

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_ERR\_NO\_MEM Out of memory

#### Parameters

- `i2s_num`: I2S\_NUM\_0, I2S\_NUM\_1
- `dsr`: i2s RX down sample rate for PDM mode.

*esp\_err\_t* `i2s_set_dac_mode` (*i2s\_dac\_mode\_t* *dac\_mode*)

Set I2S dac mode, I2S built-in DAC is disabled by default.

**Note** Built-in DAC functions are only supported on I2S0 for current ESP32 chip. If either of the built-in DAC channel are enabled, the other one can not be used as RTC DAC function at the same time.

#### Return

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

#### Parameters

- `dac_mode`: DAC mode configurations - see `i2s_dac_mode_t`

*esp\_err\_t* `i2s_driver_install` (*i2s\_port\_t* *i2s\_num*, `const` *i2s\_config\_t* \**i2s\_config*, `int` *queue\_size*, `void` \**i2s\_queue*)

Install and start I2S driver.

This function must be called before any I2S driver read/write operations.

#### Parameters

- `i2s_num`: I2S\_NUM\_0, I2S\_NUM\_1
- `i2s_config`: I2S configurations - see `i2s_config_t` struct
- `queue_size`: I2S event queue size/depth.
- `i2s_queue`: I2S event queue handle, if set NULL, driver will not use an event queue.

#### Return

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_ERR\_NO\_MEM Out of memory

*esp\_err\_t* `i2s_driver_uninstall` (*i2s\_port\_t* *i2s\_num*)

Uninstall I2S driver.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- `i2s_num`: I2S\_NUM\_0, I2S\_NUM\_1

`esp_err_t i2s_write(i2s_port_t i2s_num, const void *src, size_t size, size_t *bytes_written, TickType_t ticks_to_wait)`

Write data to I2S DMA transmit buffer.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- `i2s_num`: I2S\_NUM\_0, I2S\_NUM\_1
- `src`: Source address to write from
- `size`: Size of data in bytes
- [out] `bytes_written`: Number of bytes written, if timeout, the result will be less than the size passed in.
- `ticks_to_wait`: TX buffer wait timeout in RTOS ticks. If this many ticks pass without space becoming available in the DMA transmit buffer, then the function will return (note that if the data is written to the DMA buffer in pieces, the overall operation may still take longer than this timeout.) Pass `portMAX_DELAY` for no timeout.

`esp_err_t i2s_write_expand(i2s_port_t i2s_num, const void *src, size_t size, size_t src_bits, size_t aim_bits, size_t *bytes_written, TickType_t ticks_to_wait)`

Write data to I2S DMA transmit buffer while expanding the number of bits per sample. For example, expanding 16-bit PCM to 32-bit PCM.

Format of the data in source buffer is determined by the I2S configuration (see [i2s\\_config\\_t](#)).

**Parameters**

- `i2s_num`: I2S\_NUM\_0, I2S\_NUM\_1
- `src`: Source address to write from
- `size`: Size of data in bytes
- `src_bits`: Source audio bit
- `aim_bits`: Bit wanted, no more than 32, and must be greater than `src_bits`
- [out] `bytes_written`: Number of bytes written, if timeout, the result will be less than the size passed in.
- `ticks_to_wait`: TX buffer wait timeout in RTOS ticks. If this many ticks pass without space becoming available in the DMA transmit buffer, then the function will return (note that if the data is written to the DMA buffer in pieces, the overall operation may still take longer than this timeout.) Pass `portMAX_DELAY` for no timeout.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

`esp_err_t i2s_read(i2s_port_t i2s_num, void *dest, size_t size, size_t *bytes_read, TickType_t ticks_to_wait)`

Read data from I2S DMA receive buffer.

**Note** If the built-in ADC mode is enabled, we should call `i2s_adc_start` and `i2s_adc_stop` around the whole reading process, to prevent the data getting corrupted.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- `i2s_num`: I2S\_NUM\_0, I2S\_NUM\_1
- `dest`: Destination address to read into
- `size`: Size of data in bytes
- [out] `bytes_read`: Number of bytes read, if timeout, bytes read will be less than the size passed in.

- `ticks_to_wait`: RX buffer wait timeout in RTOS ticks. If this many ticks pass without bytes becoming available in the DMA receive buffer, then the function will return (note that if data is read from the DMA buffer in pieces, the overall operation may still take longer than this timeout.) Pass `portMAX_DELAY` for no timeout.

*esp\_err\_t* **i2s\_set\_sample\_rates** (*i2s\_port\_t* *i2s\_num*, *uint32\_t* *rate*)

Set sample rate used for I2S RX and TX.

The bit clock rate is determined by the sample rate and *i2s\_config\_t* configuration parameters (number of channels, `bits_per_sample`).

`bit_clock = rate * (number of channels) * bits_per_sample`

#### Return

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error
- `ESP_ERR_NO_MEM` Out of memory

#### Parameters

- `i2s_num`: `I2S_NUM_0`, `I2S_NUM_1`
- `rate`: I2S sample rate (ex: 8000, 44100...)

*esp\_err\_t* **i2s\_stop** (*i2s\_port\_t* *i2s\_num*)

Stop I2S driver.

Disables I2S TX/RX, until `i2s_start()` is called.

#### Return

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error

#### Parameters

- `i2s_num`: `I2S_NUM_0`, `I2S_NUM_1`

*esp\_err\_t* **i2s\_start** (*i2s\_port\_t* *i2s\_num*)

Start I2S driver.

It is not necessary to call this function after `i2s_driver_install()` (it is started automatically), however it is necessary to call it after `i2s_stop()`.

#### Return

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error

#### Parameters

- `i2s_num`: `I2S_NUM_0`, `I2S_NUM_1`

*esp\_err\_t* **i2s\_zero\_dma\_buffer** (*i2s\_port\_t* *i2s\_num*)

Zero the contents of the TX DMA buffer.

Pushes zero-byte samples into the TX DMA buffer, until it is full.

#### Return

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error

#### Parameters

- `i2s_num`: `I2S_NUM_0`, `I2S_NUM_1`

*esp\_err\_t* **i2s\_set\_clk** (*i2s\_port\_t* *i2s\_num*, *uint32\_t* *rate*, *i2s\_bits\_per\_sample\_t* *bits*, *i2s\_channel\_t* *ch*)

Set clock & bit width used for I2S RX and TX.

Similar to `i2s_set_sample_rates()`, but also sets bit width.

#### Return

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error
- `ESP_ERR_NO_MEM` Out of memory

#### Parameters

- `i2s_num`: `I2S_NUM_0`, `I2S_NUM_1`

- `rate`: I2S sample rate (ex: 8000, 44100...)
- `bits`: I2S bit width (I2S\_BITS\_PER\_SAMPLE\_16BIT, I2S\_BITS\_PER\_SAMPLE\_24BIT, I2S\_BITS\_PER\_SAMPLE\_32BIT)
- `ch`: I2S channel, (I2S\_CHANNEL\_MONO, I2S\_CHANNEL\_STEREO)

float **i2s\_get\_clk** (*i2s\_port\_t* *i2s\_num*)  
get clock set on particular port number.

**Return**

- actual clock set by i2s driver

**Parameters**

- `i2s_num`: I2S\_NUM\_0, I2S\_NUM\_1

*esp\_err\_t* **i2s\_set\_adc\_mode** (*adc\_unit\_t* *adc\_unit*, *adc1\_channel\_t* *adc\_channel*)

Set built-in ADC mode for I2S DMA, this function will initialize ADC pad, and set ADC parameters.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- `adc_unit`: SAR ADC unit index
- `adc_channel`: ADC channel index

*esp\_err\_t* **i2s\_adc\_enable** (*i2s\_port\_t* *i2s\_num*)

Start to use I2S built-in ADC mode.

**Note** This function would acquire the lock of ADC to prevent the data getting corrupted during the I2S peripheral is being used to do fully continuous ADC sampling.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_ERR\_INVALID\_STATE Driver state error

**Parameters**

- `i2s_num`: i2s port index

*esp\_err\_t* **i2s\_adc\_disable** (*i2s\_port\_t* *i2s\_num*)

Stop to use I2S built-in ADC mode.

**Note** This function would release the lock of ADC so that other tasks can use ADC.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_ERR\_INVALID\_STATE Driver state error

**Parameters**

- `i2s_num`: i2s port index

**Structures**

**struct i2s\_config\_t**

I2S configuration parameters for `i2s_param_config` function.

**Public Members**

*i2s\_mode\_t* **mode**

I2S work mode

int **sample\_rate**

I2S sample rate

*i2s\_bits\_per\_sample\_t* **bits\_per\_sample**

I2S bits per sample



*i2s\_channel\_fmt\_t* **channel\_format**

I2S channel format

*i2s\_comm\_format\_t* **communication\_format**

I2S communication format

int **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

int **dma\_buf\_count**

I2S DMA Buffer Count

int **dma\_buf\_len**

I2S DMA Buffer Length

bool **use\_apll**

I2S using APLL as main I2S clock, enable it to get accurate clock

bool **tx\_desc\_auto\_clear**

I2S auto clear tx descriptor if there is underflow condition (helps in avoiding noise in case of data unavailability)

int **fixed\_mclk**

I2S using fixed MCLK output. If use\_apll = true and fixed\_mclk > 0, then the clock output for i2s is fixed and equal to the fixed\_mclk value.

**struct i2s\_event\_t**

Event structure used in I2S event queue.

### Public Members

*i2s\_event\_type\_t* **type**

I2S event type

size\_t **size**

I2S data size for I2S\_DATA event

**struct i2s\_pin\_config\_t**

I2S pin number for i2s\_set\_pin.

### Public Members

int **bck\_io\_num**

BCK in out pin

int **ws\_io\_num**

WS in out pin

int **data\_out\_num**

DATA out pin

int **data\_in\_num**

DATA in pin

### Macros

**I2S\_PIN\_NO\_CHANGE**

Use in *i2s\_pin\_config\_t* for pins which should not be changed

### Type Definitions

**typedef intr\_handle\_t i2s\_isr\_handle\_t**

**Enumerations****enum i2s\_bits\_per\_sample\_t**

I2S bit width per sample.

*Values:***I2S\_BITS\_PER\_SAMPLE\_8BIT = 8**

I2S bits per sample: 8-bits

**I2S\_BITS\_PER\_SAMPLE\_16BIT = 16**

I2S bits per sample: 16-bits

**I2S\_BITS\_PER\_SAMPLE\_24BIT = 24**

I2S bits per sample: 24-bits

**I2S\_BITS\_PER\_SAMPLE\_32BIT = 32**

I2S bits per sample: 32-bits

**enum i2s\_channel\_t**

I2S channel.

*Values:***I2S\_CHANNEL\_MONO = 1**

I2S 1 channel (mono)

**I2S\_CHANNEL\_STEREO = 2**

I2S 2 channel (stereo)

**enum i2s\_comm\_format\_t**

I2S communication standard format.

*Values:***I2S\_COMM\_FORMAT\_I2S = 0x01**

I2S communication format I2S

**I2S\_COMM\_FORMAT\_I2S\_MSB = 0x02**

I2S format MSB

**I2S\_COMM\_FORMAT\_I2S\_LSB = 0x04**

I2S format LSB

**I2S\_COMM\_FORMAT\_PCM = 0x08**

I2S communication format PCM

**I2S\_COMM\_FORMAT\_PCM\_SHORT = 0x10**

PCM Short

**I2S\_COMM\_FORMAT\_PCM\_LONG = 0x20**

PCM Long

**enum i2s\_channel\_fmt\_t**

I2S channel format type.

*Values:***I2S\_CHANNEL\_FMT\_RIGHT\_LEFT = 0x00****I2S\_CHANNEL\_FMT\_ALL\_RIGHT****I2S\_CHANNEL\_FMT\_ALL\_LEFT****I2S\_CHANNEL\_FMT\_ONLY\_RIGHT****I2S\_CHANNEL\_FMT\_ONLY\_LEFT****enum pdm\_sample\_rate\_ratio\_t**

PDM sample rate ratio, measured in Hz.

*Values:*

**PDM\_SAMPLE\_RATE\_RATIO\_64**

**PDM\_SAMPLE\_RATE\_RATIO\_128**

**enum pdm\_pcm\_conv\_t**

PDM PCM convter enable/disable.

*Values:*

**PDM\_PCM\_CONV\_ENABLE**

**PDM\_PCM\_CONV\_DISABLE**

**enum i2s\_port\_t**

I2S Peripheral, 0 & 1.

*Values:*

**I2S\_NUM\_0** = 0x0

I2S 0

**I2S\_NUM\_1** = 0x1

I2S 1

**I2S\_NUM\_MAX**

**enum i2s\_mode\_t**

I2S Mode, default is I2S\_MODE\_MASTER | I2S\_MODE\_TX.

**Note** PDM and built-in DAC functions are only supported on I2S0 for current ESP32 chip.

*Values:*

**I2S\_MODE\_MASTER** = 1

**I2S\_MODE\_SLAVE** = 2

**I2S\_MODE\_TX** = 4

**I2S\_MODE\_RX** = 8

**I2S\_MODE\_DAC\_BUILT\_IN** = 16

Output I2S data to built-in DAC, no matter the data format is 16bit or 32 bit, the DAC module will only take the 8bits from MSB

**I2S\_MODE\_ADC\_BUILT\_IN** = 32

Input I2S data from built-in ADC, each data can be 12-bit width at most

**I2S\_MODE\_PDM** = 64

**enum i2s\_event\_type\_t**

I2S event types.

*Values:*

**I2S\_EVENT\_DMA\_ERROR**

**I2S\_EVENT\_TX\_DONE**

I2S DMA finish sent 1 buffer

**I2S\_EVENT\_RX\_DONE**

I2S DMA finish received 1 buffer

**I2S\_EVENT\_MAX**

I2S event max index

**enum i2s\_dac\_mode\_t**

I2S DAC mode for i2s\_set\_dac\_mode.

**Note** PDM and built-in DAC functions are only supported on I2S0 for current ESP32 chip.

*Values:*

```

I2S_DAC_CHANNEL_DISABLE = 0
    Disable I2S built-in DAC signals

I2S_DAC_CHANNEL_RIGHT_EN = 1
    Enable I2S built-in DAC right channel, maps to DAC channel 1 on GPIO25

I2S_DAC_CHANNEL_LEFT_EN = 2
    Enable I2S built-in DAC left channel, maps to DAC channel 2 on GPIO26

I2S_DAC_CHANNEL_BOTH_EN = 0x3
    Enable both of the I2S built-in DAC channels.

I2S_DAC_CHANNEL_MAX = 0x4
    I2S built-in DAC mode max index

```

```

enum i2s_pdm_dsr_t
    I2S PDM RX downsample mode.

```

*Values:*

```

I2S_PDM_DSR_8S = 0
    downsampling number is 8 for PDM RX mode

I2S_PDM_DSR_16S
    downsampling number is 16 for PDM RX mode

I2S_PDM_DSR_MAX

```

## 2.3.7 LED Control

### Introduction

The LED control (LEDC) peripheral is primarily designed to control the intensity of LEDs, although it can also be used to generate PWM signals for other purposes as well. It has 16 channels which can generate independent waveforms that can be used, for example, to drive RGB LED devices.

A half of LEDC's channels operate in high speed mode. This mode is implemented in hardware and offers automatic and glitch-free changing of the PWM duty cycle. The other half of channels operate in low speed mode, where the moment of change depends on the application software. Each group of channels is also able to use different clock sources, but this feature is not yet supported in the LEDC driver.

The PWM controller can automatically increase or decrease the duty cycle gradually, allowing for fades without any processor interference.

### Functionality Overview

Getting LEDC to work on a specific channel in either *high or low speed mode* is done in three steps:

1. *Configure Timer* by specifying the PWM signal's frequency and duty cycle resolution.
2. *Configure Channel* by associating it with the timer and GPIO to output the PWM signal.
3. *Change PWM Signal* that drives the output in order to change LED's intensity. This can be done under the full control of software or with hardware fading functions.

As an optional step, it is also possible to set up an interrupt on the fade end.

**Configure Timer** Setting the timer is done by calling the function `ledc_timer_config()` and passing to it a data structure `ledc_timer_config_t` that contains the following configuration settings:

- Timer number `ledc_timer_t`
- Speed mode `ledc_mode_t`
- PWM signal frequency
- Resolution of PWM duty

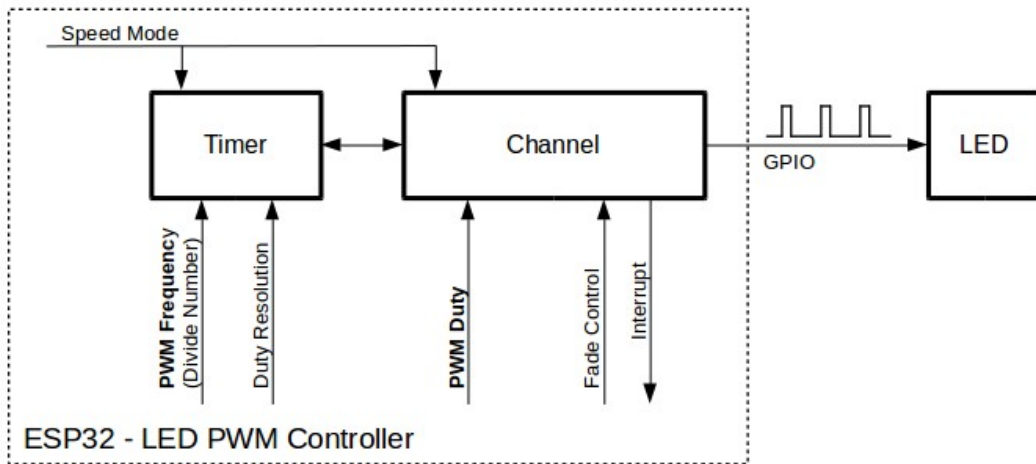


图 14: Key Settings of LED PWM Controller' s API

The frequency and the duty resolution are interdependent. The higher the PWM frequency, the lower duty resolution is available, and vice versa. This relationship might be important if you are planning to use this API for purposes other than changing the intensity of LEDs. For more details, see Section [Supported Range of Frequency and Duty Resolutions](#).

**Configure Channel** When the timer is set up, configure a selected channel (one out of `ledc_channel_t`). This is done by calling the function `ledc_channel_config()`.

Similar to the timer configuration, the channel setup function should be passed a structure `ledc_channel_config_t` that contains the channel' s configuration parameters.

At this point, the channel should start operating and generating the PWM signal on the selected GPIO, as configured in `ledc_channel_config_t`, with the frequency specified in the timer settings and the given duty cycle. The channel operation (signal generation) can be suspended at any time by calling the function `ledc_stop()`.

**Change PWM Signal** Once the channel starts operating and generating the PWM signal with the constant duty cycle and frequency, there are a couple of ways to change this signal. When driving LEDs, primarily the duty cycle is changed to vary the light intensity.

The following two sections describe how to change the duty cycle using software and hardware fading. If required, the signal' s frequency can also be changed; it is covered in Section [Change PWM Frequency](#).

**Change PWM Duty Cycle Using Software** To set the duty cycle, use the dedicated function `ledc_set_duty()`. After that, call `ledc_update_duty()` to activate the changes. To check the currently set value, use the corresponding `_get_` function `ledc_get_duty()`.

Another way to set the duty cycle, as well as some other channel parameters, is by calling `ledc_channel_config()` covered in Section [Configure Channel](#).

The range of the duty cycle values passed to functions depends on selected `duty_resolution` and should be from 0 to  $(2^{**} \text{duty\_resolution}) - 1$ . For example, if the selected duty resolution is 10, then the duty cycle values can range from 0 to 1023. This provides the resolution of ~0.1%.

**Change PWM Duty Cycle using Hardware** The LEDC hardware provides the means to gradually transition from one duty cycle value to another. To use this functionality, enable fading with `ledc_fade_func_install()` and then configure it by calling one of the available fading functions:

- `ledc_set_fade_with_time()`
- `ledc_set_fade_with_step()`
- `ledc_set_fade()`

Finally start fading with `ledc_fade_start()`.

If not required anymore, fading and an associated interrupt can be disabled with `ledc_fade_func_uninstall()`.

**Change PWM Frequency** The LEDC API provides several ways to change the PWM frequency “on the fly” :

- Set the frequency by calling `ledc_set_freq()`. There is a corresponding function `ledc_get_freq()` to check the current frequency.
- Change the frequency and the duty resolution by calling `ledc_bind_channel_timer()` to bind some other timer to the channel.
- Change the channel’s timer by calling `ledc_channel_config()`.

**More Control Over PWM** There are several lower level timer-specific functions that can be used to change PWM settings:

- `ledc_timer_set()`
- `ledc_timer_rst()`
- `ledc_timer_pause()`
- `ledc_timer_resume()`

The first two functions are called “behind the scenes” by `ledc_channel_config()` to provide a “clean” startup of a timer after it is configured.

**Use Interrupts** When configuring an LEDC channel, one of the parameters selected within `ledc_channel_config_t` is `ledc_intr_type_t` which triggers an interrupt on fade completion.

For registration of a handler to address this interrupt, call `ledc_isr_register()`.

### LEDC High and Low Speed Mode

Of the total 8 timers and 16 channels available in the LED PWM Controller, half of them are dedicated to operation in high speed mode and the other half in low speed mode. Selection of a low or high speed timer or channel is done with the parameter `ledc_mode_t` that can be found in applicable function calls.

The advantage of high speed mode is hardware-supported, glitch-free changeover of the timer settings. This means that if the timer settings are modified, the changes will be applied automatically on the next overflow interrupt of the timer. In contrast, when updating the low-speed timer, the change of settings should be explicitly triggered by software. The LEDC driver handles it in the background, e.g., when `ledc_timer_config()` or `ledc_timer_set()` is called.

For additional details regarding speed modes, refer to [ESP32 Technical Reference Manual \(PDF\)](#). Please note that the support for `SLOW_CLOCK` mentioned in this manual is not yet supported in the LEDC driver.

### Supported Range of Frequency and Duty Resolutions

The LED PWM Controller is designed primarily to drive LEDs. It provides a wide resolution for PWM duty cycle settings. For instance, the PWM frequency of 5 kHz can have the maximum duty resolution of 13 bits. It means that the duty can be set anywhere from 0 to 100% with a resolution of ~0.012% ( $2^{13} = 8192$  discrete levels of the LED intensity).

The LEDC can be used for generating signals at much higher frequencies that are sufficient enough to clock other devices, e.g., a digital camera module. In this case, the maximum available frequency is 40 MHz with duty resolution of 1 bit. This means that the duty cycle is fixed at 50% and cannot be adjusted.

The LEDC API is designed to report an error when trying to set a frequency and a duty resolution that exceed the range of LEDC's hardware. For example, an attempt to set the frequency to 20 MHz and the duty resolution to 3 bits will result in the following error reported on a serial monitor:

```
E (196) ledc: requested frequency and duty resolution cannot be achieved, try_
↔reducing freq_hz or duty_resolution. div_param=128
```

In such a situation, either the duty resolution or the frequency must be reduced. For example, setting the duty resolution to 2 will resolve this issue and will make it possible to set the duty cycle at 25% steps, i.e., at 25%, 50% or 75%.

The LEDC driver will also capture and report attempts to configure frequency / duty resolution combinations that are below the supported minimum, e.g.:

```
E (196) ledc: requested frequency and duty resolution cannot be achieved, try_
↔increasing freq_hz or duty_resolution. div_param=128000000
```

The duty resolution is normally set using `ledc_timer_bit_t`. This enumeration covers the range from 10 to 15 bits. If a smaller duty resolution is required (from 10 down to 1), enter the equivalent numeric values directly.

## Application Example

The LEDC change duty cycle and fading control example: [peripherals/ledc](#).

## API Reference

### Header File

- [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.

#### Return

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

#### Parameters

- ledc\_conf: Pointer of LEDC channel configure struct

`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.

#### Return

- 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.

#### Parameters

- timer\_conf: Pointer of LEDC timer configure struct

`esp_err_t ledc_update_duty(ledc_mode_t speed_mode, ledc_channel_t channel)`

LEDC update channel parameters.

**Note** Call this function to activate the LEDC updated parameters. After `ledc_set_duty`, we need to call this function to update the settings.

**Note** `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`

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- `speed_mode`: Select the LEDC `speed_mode`, high-speed mode and low-speed mode,
- `channel`: LEDC channel (0-7), select from `ledc_channel_t`

*esp\_err\_t* `ledc_set_pin` (*int* `gpio_num`, *ledc\_mode\_t* `speed_mode`, *ledc\_channel\_t* `ledc_channel`)

Set LEDC output gpio.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- `gpio_num`: The LEDC output gpio
- `speed_mode`: Select the LEDC `speed_mode`, high-speed mode and low-speed mode
- `ledc_channel`: LEDC channel (0-7), select from `ledc_channel_t`

*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.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- `speed_mode`: Select the LEDC `speed_mode`, high-speed mode and low-speed mode
- `channel`: LEDC channel (0-7), select from `ledc_channel_t`
- `idle_level`: Set output idle level after LEDC stops.

*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)

**Return**

- 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`.

**Parameters**

- `speed_mode`: Select the LEDC `speed_mode`, high-speed mode and low-speed mode
- `timer_num`: LEDC timer index (0-3), select from `ledc_timer_t`
- `freq_hz`: Set the LEDC frequency

*uint32\_t* `ledc_get_freq` (*ledc\_mode\_t* `speed_mode`, *ledc\_timer\_t* `timer_num`)

LEDC get channel frequency (Hz)

**Return**

- 0 error
- Others Current LEDC frequency

**Parameters**

- `speed_mode`: Select the LEDC `speed_mode`, high-speed mode and low-speed mode
- `timer_num`: LEDC timer index (0-3), select from `ledc_timer_t`

*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.

**Note** `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`



**Note** If a fade operation is running in progress on that channel, the driver would not allow it to be stopped. Other duty operations will have to wait until the fade operation has finished.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- `speed_mode`: Select the LEDC `speed_mode`, high-speed mode and low-speed mode
- `channel`: LEDC channel (0-7), select from `ledc_channel_t`
- `duty`: Set the LEDC duty, the range of duty setting is  $[0, (2^{**}duty\_resolution)]$
- `hpoint`: Set the LEDC hpoint value(max: 0xffff)

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.

**Return**

- LEDC\_ERR\_VAL if parameter error
- Others Current hpoint value of LEDC channel

**Parameters**

- `speed_mode`: Select the LEDC `speed_mode`, high-speed mode and low-speed mode
- `channel`: LEDC channel (0-7), select from `ledc_channel_t`

*`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.

**Note** `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`.

**Note** If a fade operation is running in progress on that channel, the driver would not allow it to be stopped. Other duty operations will have to wait until the fade operation has finished.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- `speed_mode`: Select the LEDC `speed_mode`, high-speed mode and low-speed mode
- `channel`: LEDC channel (0-7), select from `ledc_channel_t`
- `duty`: Set the LEDC duty, the range of duty setting is  $[0, (2^{**}duty\_resolution)]$

uint32\_t `ledc_get_duty` (*`ledc_mode_t speed_mode, ledc_channel_t channel`*)  
LEDC get duty.

**Return**

- LEDC\_ERR\_DUTY if parameter error
- Others Current LEDC duty

**Parameters**

- `speed_mode`: Select the LEDC `speed_mode`, high-speed mode and low-speed mode
- `channel`: LEDC channel (0-7), select from `ledc_channel_t`

*`esp_err_t ledc_set_fade`* (*`ledc_mode_t speed_mode, ledc_channel_t channel, uint32_t duty, ledc_duty_direction_t fade_direction, uint32_t step_num, uint32_t duty_cycle_num, uint32_t duty_scale`*)

LEDC set gradient Set LEDC gradient, After the function calls the `ledc_update_duty` function, the function can take effect.

**Note** If a fade operation is running in progress on that channel, the driver would not allow it to be stopped. Other duty operations will have to wait until the fade operation has finished.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- `speed_mode`: Select the LEDC `speed_mode`, high-speed mode and low-speed mode
- `channel`: LEDC channel (0-7), select from `ledc_channel_t`

- `duty`: Set the start of the gradient duty, the range of duty setting is  $[0, (2^{**}duty\_resolution)]$
- `fade_direction`: Set the direction of the gradient
- `step_num`: Set the number of the gradient
- `duty_cycle_num`: Set how many LEDC tick each time the gradient lasts
- `duty_scale`: Set gradient change amplitude

*esp\_err\_t* **ledc\_isr\_register** (void (\*fn)) void \*

, void \*arg, int intr\_alloc\_flags, *ledc\_isr\_handle\_t* \*handle Register LEDC interrupt handler, the handler is an ISR. The handler will be attached to the same CPU core that this function is running on.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Function pointer error.

**Parameters**

- `fn`: Interrupt handler function.
- `arg`: User-supplied argument passed to the 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.
- `arg`: Parameter for handler function
- `handle`: Pointer to return handle. If non-NULL, a handle for the interrupt will be returned here.

*esp\_err\_t* **ledc\_timer\_set** (*ledc\_mode\_t* speed\_mode, *ledc\_timer\_t* timer\_sel, uint32\_t clock\_divider, uint32\_t duty\_resolution, *ledc\_clk\_src\_t* clk\_src)

Configure LEDC settings.

**Return**

- (-1) Parameter error
- Other Current LEDC duty

**Parameters**

- `speed_mode`: Select the LEDC speed\_mode, high-speed mode and low-speed mode
- `timer_sel`: Timer index (0-3), there are 4 timers in LEDC module
- `clock_divider`: Timer clock divide value, the timer clock is divided from the selected clock source
- `duty_resolution`: Resolution of duty setting in number of bits. The range of duty values is  $[0, (2^{**}duty\_resolution)]$
- `clk_src`: Select LEDC source clock.

*esp\_err\_t* **ledc\_timer\_rst** (*ledc\_mode\_t* speed\_mode, uint32\_t timer\_sel)

Reset LEDC timer.

**Return**

- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_OK Success

**Parameters**

- `speed_mode`: Select the LEDC speed\_mode, high-speed mode and low-speed mode
- `timer_sel`: LEDC timer index (0-3), select from `ledc_timer_t`

*esp\_err\_t* **ledc\_timer\_pause** (*ledc\_mode\_t* speed\_mode, uint32\_t timer\_sel)

Pause LEDC timer counter.

**Return**

- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_OK Success

**Parameters**

- `speed_mode`: Select the LEDC speed\_mode, high-speed mode and low-speed mode
- `timer_sel`: LEDC timer index (0-3), select from `ledc_timer_t`

*esp\_err\_t* **ledc\_timer\_resume** (*ledc\_mode\_t* speed\_mode, uint32\_t timer\_sel)

Resume LEDC timer.

**Return**

- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_OK Success

**Parameters**

- `speed_mode`: Select the LEDC `speed_mode`, high-speed mode and low-speed mode
- `timer_sel`: LEDC timer index (0-3), select from `ledc_timer_t`

*esp\_err\_t* `ledc_bind_channel_timer` (*ledc\_mode\_t* `speed_mode`, *uint32\_t* `channel`, *uint32\_t* `timer_idx`)

Bind LEDC channel with the selected timer.

**Return**

- `ESP_ERR_INVALID_ARG` Parameter error
- `ESP_OK` Success

**Parameters**

- `speed_mode`: Select the LEDC `speed_mode`, high-speed mode and low-speed mode
- `channel`: LEDC channel index (0-7), select from `ledc_channel_t`
- `timer_idx`: LEDC timer index (0-3), select from `ledc_timer_t`

*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.

**Note** Call `ledc_fade_func_install()` once before calling this function. Call `ledc_fade_start()` after this to start fading.

**Note** `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`

**Note** If a fade operation is running in progress on that channel, the driver would not allow it to be stopped. Other duty operations will have to wait until the fade operation has finished.

**Return**

- `ESP_ERR_INVALID_ARG` Parameter error
- `ESP_OK` Success
- `ESP_ERR_INVALID_STATE` Fade function not installed.
- `ESP_FAIL` Fade function init error

**Parameters**

- `speed_mode`: Select the LEDC `speed_mode`, high-speed mode and low-speed mode,
- `channel`: LEDC channel index (0-7), 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

*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.

**Note** Call `ledc_fade_func_install()` once before calling this function. Call `ledc_fade_start()` after this to start fading.

**Note** `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`

**Note** If a fade operation is running in progress on that channel, the driver would not allow it to be stopped. Other duty operations will have to wait until the fade operation has finished.

**Return**

- `ESP_ERR_INVALID_ARG` Parameter error
- `ESP_OK` Success
- `ESP_ERR_INVALID_STATE` Fade function not installed.
- `ESP_FAIL` Fade function init error

**Parameters**

- `speed_mode`: Select the LEDC `speed_mode`, high-speed mode and low-speed mode,
- `channel`: LEDC channel index (0-7), 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 ).

*esp\_err\_t* `ledc_fade_func_install` (*int* `intr_alloc_flags`)

Install LEDC fade function. This function will occupy interrupt of LEDC module.

#### Return

- ESP\_OK Success
- ESP\_ERR\_INVALID\_STATE Fade function already installed.

#### Parameters

- `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.

```
void ledc_fade_func_uninstall ()
```

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.

**Note** 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.

**Note** If a fade operation is running in progress on that channel, the driver would not allow it to be stopped. Other duty operations will have to wait until the fade operation has finished.

#### Return

- ESP\_OK Success
- ESP\_ERR\_INVALID\_STATE Fade function not installed.
- ESP\_ERR\_INVALID\_ARG Parameter error.

#### Parameters

- `speed_mode`: Select the LEDC `speed_mode`, high-speed mode and low-speed mode
- `channel`: LEDC channel number
- `fade_mode`: Whether to block until fading done.

```
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.

**Note** If a fade operation is running in progress on that channel, the driver would not allow it to be stopped. Other duty operations will have to wait until the fade operation has finished.

#### Parameters

- `speed_mode`: Select the LEDC `speed_mode`, high-speed mode and low-speed mode
- `channel`: LEDC channel (0-7), select from `ledc_channel_t`
- `duty`: Set the LEDC duty, the range of duty setting is  $[0, (2^{**}duty\_resolution)]$
- `hpoint`: Set the LEDC `hpoint` value(max: 0xffff)

```
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.

**Note** Call `ledc_fade_func_install()` once, before calling this function.

**Note** If a fade operation is running in progress on that channel, the driver would not allow it to be stopped. Other duty operations will have to wait until the fade operation has finished.

#### Return

- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_OK Success
- ESP\_ERR\_INVALID\_STATE Fade function not installed.
- ESP\_FAIL Fade function init error

#### Parameters

- `speed_mode`: Select the LEDC `speed_mode`, high-speed mode and low-speed mode,
- `channel`: LEDC channel index (0-7), 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

*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.

**Note** Call ledc\_fade\_func\_install() once before calling this function.

**Note** If a fade operation is running in progress on that channel, the driver would not allow it to be stopped.

Other duty operations will have to wait until the fade operation has finished.

#### Return

- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_OK Success
- ESP\_ERR\_INVALID\_STATE Fade function not installed.
- ESP\_FAIL Fade function init error

#### Parameters

- speed\_mode: Select the LEDC speed\_mode, high-speed mode and low-speed mode,
- channel: LEDC channel index (0-7), 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

### Structures

**struct ledc\_channel\_config\_t**

Configuration parameters of LEDC channel for ledc\_channel\_config function.

#### Public Members

int **gpio\_num**

the LEDC output gpio\_num, if you want to use gpio16, gpio\_num = 16

*ledc\_mode\_t* **speed\_mode**

LEDC speed speed\_mode, high-speed mode or low-speed mode

*ledc\_channel\_t* **channel**

LEDC channel (0 - 7)

*ledc\_intr\_type\_t* **intr\_type**

configure interrupt, Fade interrupt enable or Fade interrupt disable

*ledc\_timer\_t* **timer\_sel**

Select the timer source of channel (0 - 3)

uint32\_t **duty**

LEDC channel duty, the range of duty setting is [0, (2\*\*duty\_resolution)]

int **hpoint**

LEDC channel hpoint value, the max value is 0xffff

**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. For low speed channels and high speed channels, you can specify the source clock using `LEDC_USE_REF_TICK`, `LEDC_USE_APB_CLK` or `LEDC_AUTO_CLK`. For low speed channels, you can also specify the source clock using `LEDC_USE_RTC8M_CLK`, in this case, all low speed channel's source clock must be `RTC8M_CLK`

### Macros

`LEDC_APB_CLK_HZ`

`LEDC_REF_CLK_HZ`

`LEDC_ERR_DUTY`

`LEDC_ERR_VAL`

### Type Definitions

`typedef intr_handle_t ledc_isr_handle_t`

### Enumerations

`enum ledc_mode_t`

*Values:*

`LEDC_HIGH_SPEED_MODE = 0`

LEDC high speed speed\_mode

`LEDC_LOW_SPEED_MODE`

LEDC low speed speed\_mode

`LEDC_SPEED_MODE_MAX`

LEDC speed limit

`enum ledc_intr_type_t`

*Values:*

`LEDC_INTR_DISABLE = 0`

Disable LEDC interrupt

`LEDC_INTR_FADE_END`

Enable LEDC interrupt

`enum ledc_duty_direction_t`

*Values:*

`LEDC_DUTY_DIR_DECREASE = 0`

LEDC duty decrease direction

`LEDC_DUTY_DIR_INCREASE = 1`

LEDC duty increase direction

`LEDC_DUTY_DIR_MAX`

`enum ledc_clk_cfg_t`

*Values:*

`LEDC_AUTO_CLK`

The driver will automatically select the source clock(`REF_TICK` or `APB`) based on the giving resolution and duty parameter when init the timer

`LEDC_USE_REF_TICK`

LEDC timer select `REF_TICK` clock as source clock

`LEDC_USE_APB_CLK`

LEDC timer select `APB` clock as source clock

**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

**enum ledc\_clk\_src\_t**

*Values:*

**LEDC\_REF\_TICK** = *LEDC\_USE\_REF\_TICK*

LEDC timer clock divided from reference tick (1Mhz)

**LEDC\_APB\_CLK** = *LEDC\_USE\_APB\_CLK*

LEDC timer clock divided from APB clock (80Mhz)

**enum ledc\_timer\_t**

*Values:*

**LEDC\_TIMER\_0** = 0

LEDC timer 0

**LEDC\_TIMER\_1**

LEDC timer 1

**LEDC\_TIMER\_2**

LEDC timer 2

**LEDC\_TIMER\_3**

LEDC timer 3

**LEDC\_TIMER\_MAX**

**enum ledc\_channel\_t**

*Values:*

**LEDC\_CHANNEL\_0** = 0

LEDC channel 0

**LEDC\_CHANNEL\_1**

LEDC channel 1

**LEDC\_CHANNEL\_2**

LEDC channel 2

**LEDC\_CHANNEL\_3**

LEDC channel 3

**LEDC\_CHANNEL\_4**

LEDC channel 4

**LEDC\_CHANNEL\_5**

LEDC channel 5

**LEDC\_CHANNEL\_6**

LEDC channel 6

**LEDC\_CHANNEL\_7**

LEDC channel 7

**LEDC\_CHANNEL\_MAX**

**enum ledc\_timer\_bit\_t**

*Values:*

**LEDC\_TIMER\_1\_BIT** = 1

LEDC PWM duty resolution of 1 bits

**LEDC\_TIMER\_2\_BIT**

LEDC PWM duty resolution of 2 bits

**LEDC\_TIMER\_3\_BIT**

LEDC PWM duty resolution of 3 bits

**LEDC\_TIMER\_4\_BIT**  
LEDC PWM duty resolution of 4 bits

**LEDC\_TIMER\_5\_BIT**  
LEDC PWM duty resolution of 5 bits

**LEDC\_TIMER\_6\_BIT**  
LEDC PWM duty resolution of 6 bits

**LEDC\_TIMER\_7\_BIT**  
LEDC PWM duty resolution of 7 bits

**LEDC\_TIMER\_8\_BIT**  
LEDC PWM duty resolution of 8 bits

**LEDC\_TIMER\_9\_BIT**  
LEDC PWM duty resolution of 9 bits

**LEDC\_TIMER\_10\_BIT**  
LEDC PWM duty resolution of 10 bits

**LEDC\_TIMER\_11\_BIT**  
LEDC PWM duty resolution of 11 bits

**LEDC\_TIMER\_12\_BIT**  
LEDC PWM duty resolution of 12 bits

**LEDC\_TIMER\_13\_BIT**  
LEDC PWM duty resolution of 13 bits

**LEDC\_TIMER\_14\_BIT**  
LEDC PWM duty resolution of 14 bits

**LEDC\_TIMER\_15\_BIT**  
LEDC PWM duty resolution of 15 bits

**LEDC\_TIMER\_16\_BIT**  
LEDC PWM duty resolution of 16 bits

**LEDC\_TIMER\_17\_BIT**  
LEDC PWM duty resolution of 17 bits

**LEDC\_TIMER\_18\_BIT**  
LEDC PWM duty resolution of 18 bits

**LEDC\_TIMER\_19\_BIT**  
LEDC PWM duty resolution of 19 bits

**LEDC\_TIMER\_20\_BIT**  
LEDC PWM duty resolution of 20 bits

**LEDC\_TIMER\_BIT\_MAX**

**enum ledc\_fade\_mode\_t**

*Values:*

**LEDC\_FADE\_NO\_WAIT = 0**  
LEDC fade function will return immediately

**LEDC\_FADE\_WAIT\_DONE**  
LEDC fade function will block until fading to the target duty

**LEDC\_FADE\_MAX**

### 2.3.8 MCPWM

ESP32 has two MCPWM units which can be used to control different types of motors. Each unit has three pairs of PWM outputs.



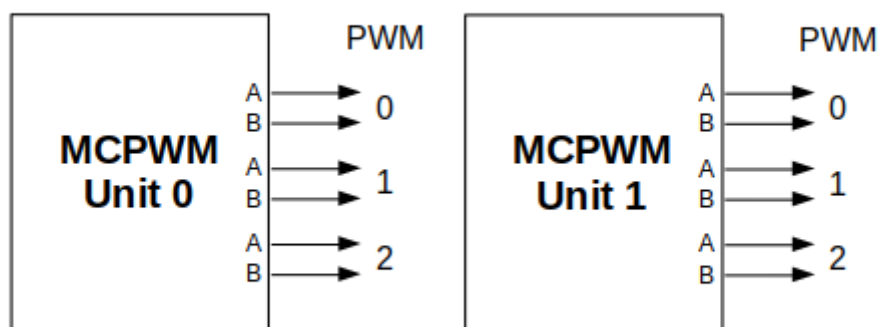


图 15: MCPWM Overview

Further in documentation the outputs of a single unit are labeled `PWMxA` / `PWMxB`.

More detailed block diagram of the MCPWM unit is shown below. Each A/B pair may be clocked by any one of the three timers Timer 0, 1 and 2. The same timer may be used to clock more than one pair of PWM outputs. Each unit is also able to collect inputs such as `SYNC SIGNALS`, detect `FAULT SIGNALS` like motor overcurrent or overvoltage, as well as obtain feedback with `CAPTURE SIGNALS` on e.g. a rotor position.

Description of this API starts with configuration of MCPWM's **Timer** and **Operator** submodules to provide the basic motor control functionality. Then it discusses more advanced submodules and functionalities of a **Fault Handler**, signal **Capture**, **Carrier** and **Interrupts**.

## Contents

- *Configure* a basic functionality of the outputs
- *Operate* the outputs to drive a motor
- *Adjust* how the motor is driven
- *Capture* external signals to provide additional control over the outputs
- Use *Fault Handler* to detect and manage faults
- Add a higher frequency *Carrier*, if output signals are passed through an isolation transformer
- Configuration and handling of *Interrupts*.

## Configure

The scope of configuration depends on the motor type, in particular how many outputs and inputs are required, and what will be the sequence of signals to drive the motor.

In this case we will describe a simple configuration to control a brushed DC motor that is using only some of the available MCPWM's resources. An example circuit is shown below. It includes a **H-Bridge** to switch polarization of a voltage applied to the motor (M) and to provide sufficient current to drive it.

Configuration covers the following steps:

1. Selection of a MPWN unit that will be used to drive the motor. There are two units available on-board of ESP32 and enumerated in `mcpwm_unit_t`.
2. Initialization of two GPIOs as output signals within selected unit by calling `mcpwm_gpio_init()`. The two output signals are typically used to command the motor to rotate right or left. All available signal options are listed in `mcpwm_io_signals_t`. To set more than a single pin at a time, use function `mcpwm_set_pin()` together with `mcpwm_pin_config_t`.

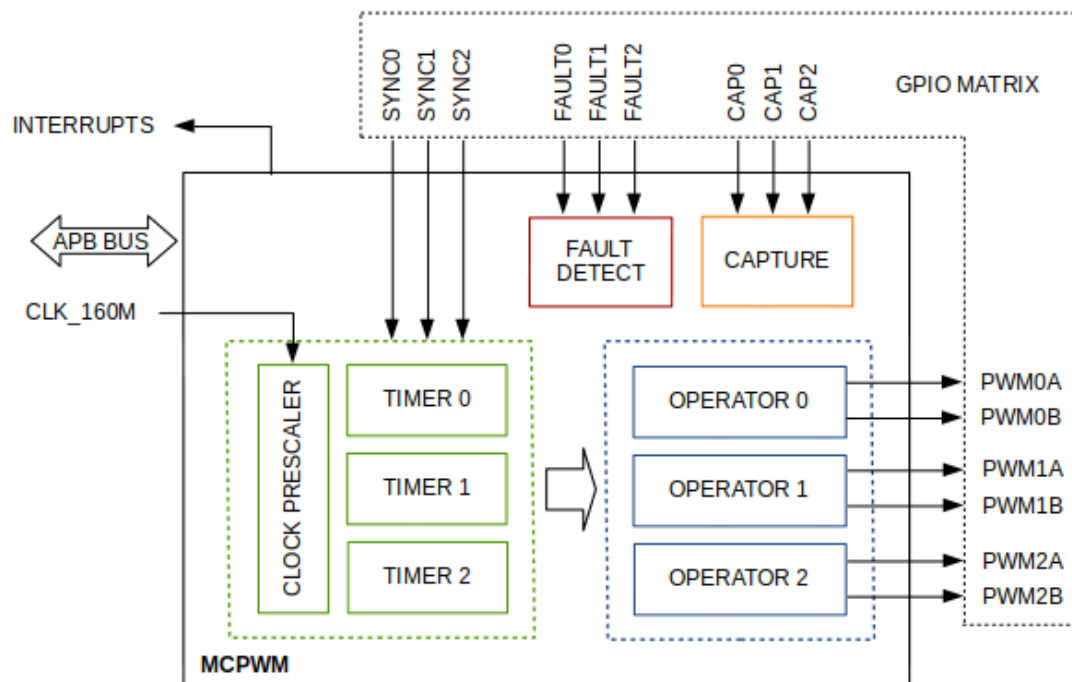


图 16: MCPWM Block Diagram

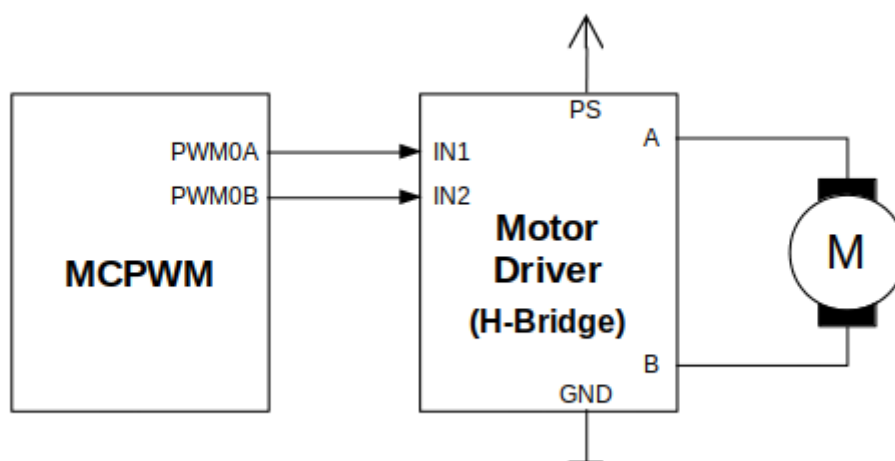


图 17: Example of Brushed DC Motor Control with MCPWM

3. Selection of a timer. There are three timers available within the unit. The timers are listed in `mcpwm_timer_t`.
4. Setting of the timer frequency and initial duty within `mcpwm_config_t` structure.
5. Calling of `mcpwm_init()` with the above parameters to make the configuration effective.

## Operate

To operate a motor connected to the MCPWM unit, e.g. turn it left or right, or vary the speed, we should apply some control signals to the unit's outputs. The outputs are organized into three pairs. Within a pair they are labeled "A" and "B" and driven by a submodule called an "Operator". To provide a PWM signal, the Operator itself should be clocked by one of three available Timers. To make the API simpler, each Timer is automatically associated by the API to drive an Operator of the same index, e.g. Timer 0 is associated with Operator 0.

There are the following basic ways to control the outputs:

- We can drive particular signal steady high or steady low with function `mcpwm_set_signal_high()` or `mcpwm_set_signal_low()`. This will make the motor to turn with a maximum speed or stop. Depending on selected output A or B the motor will rotate either right or left.
- Another option is to drive the outputs with the PWM signal by calling `mcpwm_start()` or `mcpwm_stop()`. The motor speed will be proportional to the PWM duty.
- To vary PWM's duty call `mcpwm_set_duty()` and provide the duty value in %. Optionally, you may call `mcpwm_set_duty_in_us()`, if you prefer to set the duty in microseconds. Checking of currently set value is possible by calling `mcpwm_get_duty()`. Phase of the PWM signal may be altered by calling `mcpwm_set_duty_type()`. The duty is set individually for each A and B output using `mcpwm_operator_t` in specific function calls. The duty value refers either to high or low output signal duration. This is configured when calling `mcpwm_init()`, as discussed in section *Configure*, and selecting one of options from `mcpwm_duty_type_t`.

---

**注解:** Call function `mcpwm_set_duty_type()` every time after `mcpwm_set_signal_high()` or `mcpwm_set_signal_low()` to resume with previously set duty cycle.

---

## Adjust

There are couple of ways to adjust a signal on the outputs and changing how the motor operates.

- Set specific PWM frequency by calling `mcpwm_set_frequency()`. This may be required to adjust to electrical or mechanical characteristics of particular motor and driver. To check what frequency is set, use function `mcpwm_get_frequency()`.
- Introduce a dead time between outputs A and B when they are changing the state to reverse direction of the motor rotation. This is to make up for on/off switching delay of the motor driver FETs. The dead time options are defined in `mcpwm_deadtime_type_t` and enabled by calling `mcpwm_deadtime_enable()`. To disable this functionality call `mcpwm_deadtime_disable()`.
- Synchronize outputs of operator submodules, e.g. to get raising edge of PWM0A/B and PWM1A/B to start exactly at the same time, or shift them between each other by a given phase. Synchronization is triggered by SYNC SIGNALS shown on the *block diagram* of the MCPWM above, and defined in `mcpwm_sync_signal_t`. To attach the signal to a GPIO call `mcpwm_gpio_init()`. You can then enable synchronization with function `mcpwm_sync_enable()`. As input parameters provide MCPWM unit, timer to synchronize, the synchronization signal and a phase to delay the timer.

---

**注解:** Synchronization signals are referred to using two different enumerations. First one `mcpwm_io_signals_t` is used together with function `mcpwm_gpio_init()` when selecting a GPIO as the signal input source. The second one `mcpwm_sync_signal_t` is used when enabling or disabling synchronization with `mcpwm_sync_enable()` or `mcpwm_sync_disable()`.

---

- Vary the pattern of the A/B output signals by getting MCPWM counters to count up, down and up/down (automatically changing the count direction). Respective configuration is done when calling `mcpwm_init()`,

as discussed in section [Configure](#), and selecting one of counter types from `mcpwm_counter_type_t`. For explanation of how A/B PWM output signals are generated please refer to [ESP32 Technical Reference Manual](#).

## Capture

One of requirements of BLDC (Brushless DC, see figure below) motor control is sensing of the rotor position. To facilitate this task each MCPWM unit provides three sensing inputs together with dedicated hardware. The hardware is able to detect the input signal's edge and measure time between signals. As result the control software is simpler and the CPU power may be used for other tasks.

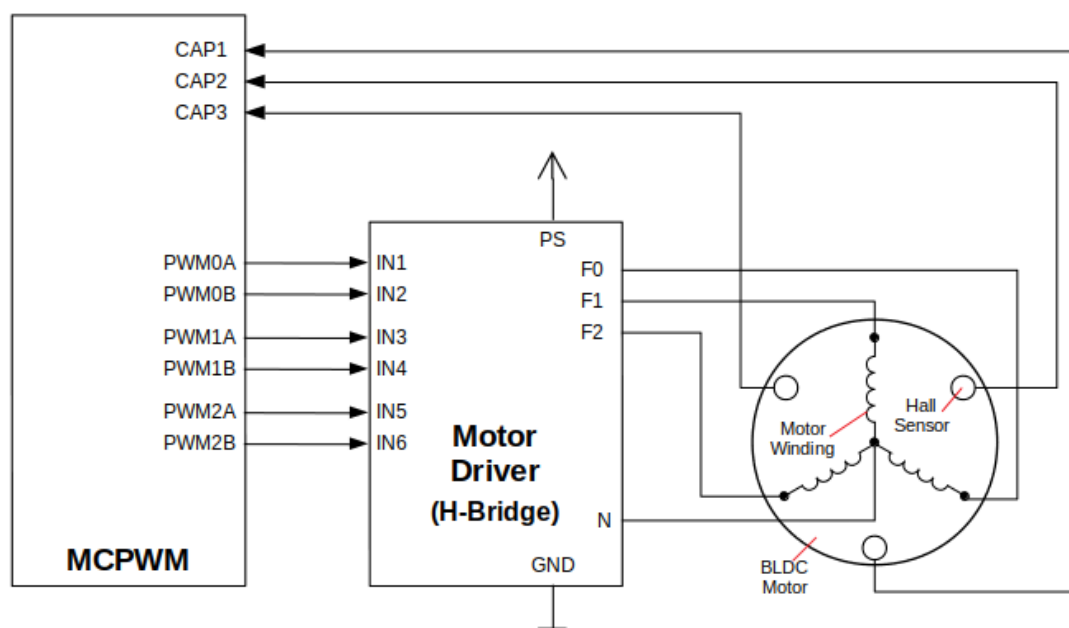


图 18: Example of Brushless DC Motor Control with MCPWM

The capture functionality may be used for other types of motors or tasks. The functionality is enabled in two steps:

1. Configuration of GPIOs to act as the capture signal inputs by calling functions `mcpwm_gpio_init()` or `mcpwm_set_pin()`, that were described in section [Configure](#).
2. Enabling of the functionality itself by invoking `mcpwm_capture_enable()`, selecting desired signal input from `mcpwm_capture_signal_t`, setting the signal edge with `mcpwm_capture_on_edge_t` and the signal count prescaler.

Within the second step above a 32-bit capture timer is enabled. The timer runs continuously driven by the APB clock. The clock frequency is typically 80 MHz. On each capture event the capture timer's value is stored in time-stamp register that may be then checked by calling `mcpwm_capture_signal_get_value()`. The edge of the last signal may be checked with `mcpwm_capture_signal_get_edge()`.

If not required anymore, the capture functionality may be disabled with `mcpwm_capture_disable()`.

## Fault Handler

Each unit of the MCPWM is able to sense external signals with information about failure of the motor, the motor driver or any other device connected to the MCPWM. There are three fault inputs per unit that may be routed to user selectable GPIOs. The MCPWM may be configured to perform one of four predefined actions on A/B outputs when a fault signal is received:

- lock current state of the output

- set the output low
- set the output high
- toggle the output

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.

The fault handler functionality is enabled in two steps:

1. Configuration of GPIOs to act as fault signal inputs. This is done in analogous way as described for capture signals in section above. It includes setting the signal level to trigger the fault as defined in `mcpwm_fault_input_level_t`.
2. Initialization of the fault handler by calling either `mcpwm_fault_set_oneshot_mode()` or `mcpwm_fault_set_cyc_mode()`. These functions set the mode that MCPWM should operate once fault signal becomes inactive. There are two modes possible:
  - State of MCPWM unit will be locked until reset - `mcpwm_fault_set_oneshot_mode()`.
  - The MCPWM will resume operation once fault signal becoming inactive - `mcpwm_fault_set_cyc_mode()`.

The function call parameters include selection of one of three fault inputs defined in `mcpwm_fault_signal_t` and specific action on outputs A and B defined in `mcpwm_action_on_pwmxa_t` and `mcpwm_action_on_pwmxb_t`.

Particular fault signal may be disabled at the runtime by calling `mcpwm_fault_deinit()`.

## Carrier

The MCPWM has a carrier submodule used if galvanic isolation from the motor driver is required by passing the A/B output signals through transformers. Any of A and B 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 use the carrier submodule, it should be first initialized by calling `mcpwm_carrier_init()`. The carrier parameters are defined in `mcpwm_carrier_config_t` structure invoked within the function call. Then the carrier functionality may be enabled by calling `mcpwm_carrier_enable()`.

The carrier parameters may be then alerted at a runtime by calling dedicated functions to change individual fields of the `mcpwm_carrier_config_t` structure, like `mcpwm_carrier_set_period()`, `mcpwm_carrier_set_duty_cycle()`, `mcpwm_carrier_output_invert()`, etc.

This includes enabling and setting duration of the first pulse of the career with `mcpwm_carrier_oneshot_mode_enable()`. For more details please refer to “PWM Carrier Submodule” section of the [ESP32 Technical Reference Manual](#).

To disable carrier functionality call `mcpwm_carrier_disable()`.

## Interrupts

Registering of the MCPWM interrupt handler is possible by calling `mcpwm_isr_register()`.

## Application Example

Examples of using MCPWM for motor control: [peripherals/mcpwm](#):

- Demonstration how to use each submodule of the MCPWM - [peripherals/mcpwm/mcpwm\\_basic\\_config](#)
- Control of BLDC (brushless DC) motor with hall sensor feedback - [peripherals/mcpwm/mcpwm\\_bldc\\_control](#)
- Brushed DC motor control - [peripherals/mcpwm/mcpwm\\_brushed\\_dc\\_control](#)
- Servo motor control - [peripherals/mcpwm/mcpwm\\_servo\\_control](#)

## API Reference

### Header File

- `driver/include/driver/mcpwm.h`

### Functions

`esp_err_t mcpwm_gpio_init(mcpwm_unit_t mcpwm_num, mcpwm_io_signals_t io_signal, int gpio_num)`

This function initializes each gpio signal for MCPWM.

**Note** This function initializes one gpio at a time.

#### Return

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

#### Parameters

- `mcpwm_num`: set MCPWM unit(0-1)
- `io_signal`: set MCPWM signals, each MCPWM unit has 6 output(MCPWMXA, MCPWMXB) and 9 input(SYNC\_X, FAULT\_X, CAP\_X) 'X' is timer\_num(0-2)
- `gpio_num`: set this to configure gpio for MCPWM, if you want to use gpio16, `gpio_num = 16`

`esp_err_t mcpwm_set_pin(mcpwm_unit_t mcpwm_num, const mcpwm_pin_config_t *mcpwm_pin)`

Initialize MCPWM gpio structure.

**Note** This function can be used to initialize more than one gpio at a time.

#### Return

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

#### Parameters

- `mcpwm_num`: set MCPWM unit(0-1)
- `mcpwm_pin`: MCPWM pin structure

`esp_err_t mcpwm_init(mcpwm_unit_t mcpwm_num, mcpwm_timer_t timer_num, const mcpwm_config_t *mcpwm_conf)`

Initialize MCPWM parameters.

#### Return

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

#### Parameters

- `mcpwm_num`: set MCPWM unit(0-1)
- `timer_num`: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers
- `mcpwm_conf`: configure structure `mcpwm_config_t`

`esp_err_t mcpwm_set_frequency(mcpwm_unit_t mcpwm_num, mcpwm_timer_t timer_num, uint32_t frequency)`

Set frequency(in Hz) of MCPWM timer.

#### Return

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

#### Parameters

- `mcpwm_num`: set MCPWM unit(0-1)
- `timer_num`: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers
- `frequency`: set the frequency in Hz of each timer

`esp_err_t mcpwm_set_duty(mcpwm_unit_t mcpwm_num, mcpwm_timer_t timer_num, mcpwm_operator_t op_num, float duty)`

Set duty cycle of each operator(MCPWMXA/MCPWMXB)

#### Return

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- `mcpwm_num`: set MCPWM unit(0-1)
- `timer_num`: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers
- `op_num`: set the operator(MCPWMXA/MCPWMXB), 'X' is timer number selected
- `duty`: set duty cycle in % (i.e for 62.3% duty cycle, `duty = 62.3`) of each operator

`esp_err_t mcpwm_set_duty_in_us` (`mcpwm_unit_t` `mcpwm_num`, `mcpwm_timer_t` `timer_num`, `mcpwm_operator_t` `op_num`, `uint32_t` `duty`)

Set duty cycle of each operator(MCPWMXA/MCPWMXB) in us.

**Return**

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error

**Parameters**

- `mcpwm_num`: set MCPWM unit(0-1)
- `timer_num`: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers
- `op_num`: set the operator(MCPWMXA/MCPWMXB), 'x' is timer number selected
- `duty`: set duty value in microseconds of each operator

`esp_err_t mcpwm_set_duty_type` (`mcpwm_unit_t` `mcpwm_num`, `mcpwm_timer_t` `timer_num`, `mcpwm_operator_t` `op_num`, `mcpwm_duty_type_t` `duty_num`)

Set duty either active high or active low(out of phase/inverted)

**Note** Call this function every time after `mcpwm_set_signal_high` or `mcpwm_set_signal_low` to resume with previously set duty cycle

**Return**

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error

**Parameters**

- `mcpwm_num`: set MCPWM unit(0-1)
- `timer_num`: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers
- `op_num`: set the operator(MCPWMXA/MCPWMXB), 'x' is timer number selected
- `duty_num`: set active low or active high duty type

`uint32_t mcpwm_get_frequency` (`mcpwm_unit_t` `mcpwm_num`, `mcpwm_timer_t` `timer_num`)

Get frequency of timer.

**Return**

- frequency of timer

**Parameters**

- `mcpwm_num`: set MCPWM unit(0-1)
- `timer_num`: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers

`float mcpwm_get_duty` (`mcpwm_unit_t` `mcpwm_num`, `mcpwm_timer_t` `timer_num`, `mcpwm_operator_t` `op_num`)

Get duty cycle of each operator.

**Return**

- duty cycle in % of each operator(56.7 means duty is 56.7%)

**Parameters**

- `mcpwm_num`: set MCPWM unit(0-1)
- `timer_num`: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers
- `op_num`: set the operator(MCPWMXA/MCPWMXB), 'x' is timer number selected

`esp_err_t mcpwm_set_signal_high` (`mcpwm_unit_t` `mcpwm_num`, `mcpwm_timer_t` `timer_num`, `mcpwm_operator_t` `op_num`)

Use this function to set MCPWM signal high.

**Return**

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error

**Parameters**

- `mcpwm_num`: set MCPWM unit(0-1)
- `timer_num`: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers

- `op_num`: set the operator(MCPWMA/MCPWMB), 'x' is timer number selected

*esp\_err\_t* **mcpwm\_set\_signal\_low** (*mcpwm\_unit\_t* *mcpwm\_num*, *mcpwm\_timer\_t* *timer\_num*, *mcpwm\_operator\_t* *op\_num*)

Use this function to set MCPWM signal low.

#### Return

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error

#### Parameters

- `mcpwm_num`: set MCPWM unit(0-1)
- `timer_num`: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers
- `op_num`: set the operator(MCPWMA/MCPWMB), 'x' is timer number selected

*esp\_err\_t* **mcpwm\_start** (*mcpwm\_unit\_t* *mcpwm\_num*, *mcpwm\_timer\_t* *timer\_num*)

Start MCPWM signal on timer 'x' .

#### Return

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error

#### Parameters

- `mcpwm_num`: set MCPWM unit(0-1)
- `timer_num`: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers

*esp\_err\_t* **mcpwm\_stop** (*mcpwm\_unit\_t* *mcpwm\_num*, *mcpwm\_timer\_t* *timer\_num*)

Start MCPWM signal on timer 'x' .

#### Return

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error

#### Parameters

- `mcpwm_num`: set MCPWM unit(0-1)
- `timer_num`: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers

*esp\_err\_t* **mcpwm\_carrier\_init** (*mcpwm\_unit\_t* *mcpwm\_num*, *mcpwm\_timer\_t* *timer\_num*, **const** *mcpwm\_carrier\_config\_t* \**carrier\_conf*)

Initialize carrier configuration.

#### Return

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error

#### Parameters

- `mcpwm_num`: set MCPWM unit(0-1)
- `timer_num`: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers
- `carrier_conf`: configure structure *mcpwm\_carrier\_config\_t*

*esp\_err\_t* **mcpwm\_carrier\_enable** (*mcpwm\_unit\_t* *mcpwm\_num*, *mcpwm\_timer\_t* *timer\_num*)

Enable MCPWM carrier submodule, for respective timer.

#### Return

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error

#### Parameters

- `mcpwm_num`: set MCPWM unit(0-1)
- `timer_num`: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers

*esp\_err\_t* **mcpwm\_carrier\_disable** (*mcpwm\_unit\_t* *mcpwm\_num*, *mcpwm\_timer\_t* *timer\_num*)

Disable MCPWM carrier submodule, for respective timer.

#### Return

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error

#### Parameters

- `mcpwm_num`: set MCPWM unit(0-1)



- `timer_num`: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers

`esp_err_t mcpwm_carrier_set_period` (*mcpwm\_unit\_t* *mcpwm\_num*, *mcpwm\_timer\_t* *timer\_num*, *uint8\_t* *carrier\_period*)

Set period of carrier.

#### Return

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

#### Parameters

- `mcpwm_num`: set MCPWM unit(0-1)
- `timer_num`: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers
- `carrier_period`: set the carrier period of each timer, carrier period = (carrier\_period + 1)\*800ns (carrier\_period <= 15)

`esp_err_t mcpwm_carrier_set_duty_cycle` (*mcpwm\_unit\_t* *mcpwm\_num*, *mcpwm\_timer\_t* *timer\_num*, *uint8\_t* *carrier\_duty*)

Set duty\_cycle of carrier.

#### Return

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

#### Parameters

- `mcpwm_num`: set MCPWM unit(0-1)
- `timer_num`: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers
- `carrier_duty`: set duty\_cycle of carrier , carrier duty cycle = carrier\_duty\*12.5% (chop\_duty <= 7)

`esp_err_t mcpwm_carrier_oneshot_mode_enable` (*mcpwm\_unit\_t* *mcpwm\_num*, *mcpwm\_timer\_t* *timer\_num*, *uint8\_t* *pulse\_width*)

Enable and set width of first pulse in carrier oneshot mode.

#### Return

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

#### Parameters

- `mcpwm_num`: set MCPWM unit(0-1)
- `timer_num`: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers
- `pulse_width`: set pulse width of first pulse in oneshot mode, width = (carrier period)\*(pulse\_width + 1) (pulse\_width <= 15)

`esp_err_t mcpwm_carrier_oneshot_mode_disable` (*mcpwm\_unit\_t* *mcpwm\_num*, *mcpwm\_timer\_t* *timer\_num*)

Disable oneshot mode, width of first pulse = carrier period.

#### Return

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

#### Parameters

- `mcpwm_num`: set MCPWM unit(0-1)
- `timer_num`: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers

`esp_err_t mcpwm_carrier_output_invert` (*mcpwm\_unit\_t* *mcpwm\_num*, *mcpwm\_timer\_t* *timer\_num*, *mcpwm\_carrier\_out\_ivt\_t* *carrier\_ivt\_mode*)

Enable or disable carrier output inversion.

#### Return

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

#### Parameters

- `mcpwm_num`: set MCPWM unit(0-1)
- `timer_num`: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers
- `carrier_ivt_mode`: enable or disable carrier output inversion

*esp\_err\_t* **mcpwm\_deadtime\_enable** (*mcpwm\_unit\_t* mcpwm\_num, *mcpwm\_timer\_t* timer\_num, *mcpwm\_deadtime\_type\_t* dt\_mode, uint32\_t red, uint32\_t fed)

Enable and initialize deadtime for each MCPWM timer.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- mcpwm\_num: set MCPWM unit(0-1)
- timer\_num: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers
- dt\_mode: set deadtime mode
- red: set rising edge delay = red\*100ns
- fed: set rising edge delay = fed\*100ns

*esp\_err\_t* **mcpwm\_deadtime\_disable** (*mcpwm\_unit\_t* mcpwm\_num, *mcpwm\_timer\_t* timer\_num)

Disable deadtime on MCPWM timer.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- mcpwm\_num: set MCPWM unit(0-1)
- timer\_num: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers

*esp\_err\_t* **mcpwm\_fault\_init** (*mcpwm\_unit\_t* mcpwm\_num, *mcpwm\_fault\_input\_level\_t* input\_level, *mcpwm\_fault\_signal\_t* fault\_sig)

Initialize fault submodule, currently low level triggering is not supported.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- mcpwm\_num: set MCPWM unit(0-1)
- input\_level: set fault signal level, which will cause fault to occur
- fault\_sig: set the fault pin, which needs to be enabled

*esp\_err\_t* **mcpwm\_fault\_set\_oneshot\_mode** (*mcpwm\_unit\_t* mcpwm\_num, *mcpwm\_timer\_t* timer\_num, *mcpwm\_fault\_signal\_t* fault\_sig, *mcpwm\_action\_on\_pwmxa\_t* action\_on\_pwmxa, *mcpwm\_action\_on\_pwmxb\_t* action\_on\_pwmxb)

Set oneshot mode on fault detection, once fault occur in oneshot mode reset is required to resume MCPWM signals.

**Note** currently low level triggering is not supported

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- mcpwm\_num: set MCPWM unit(0-1)
- timer\_num: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers
- fault\_sig: set the fault pin, which needs to be enabled for oneshot mode
- action\_on\_pwmxa: action to be taken on MCPWMXA when fault occurs, either no change or high or low or toggle
- action\_on\_pwmxb: action to be taken on MCPWMXB when fault occurs, either no change or high or low or toggle

*esp\_err\_t* **mcpwm\_fault\_set\_cyc\_mode** (*mcpwm\_unit\_t* mcpwm\_num, *mcpwm\_timer\_t* timer\_num, *mcpwm\_fault\_signal\_t* fault\_sig, *mcpwm\_action\_on\_pwmxa\_t* action\_on\_pwmxa, *mcpwm\_action\_on\_pwmxb\_t* action\_on\_pwmxb)

Set cycle-by-cycle mode on fault detection, once fault occur in cyc mode MCPWM signal resumes as soon as fault signal becomes inactive.

**Note** currently low level triggering is not supported

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- `mcpwm_num`: set MCPWM unit(0-1)
- `timer_num`: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers
- `fault_sig`: set the fault pin, which needs to be enabled for cyc mode
- `action_on_pwmxa`: action to be taken on MCPWMXA when fault occurs, either no change or high or low or toggle
- `action_on_pwmxb`: action to be taken on MCPWMXB when fault occurs, either no change or high or low or toggle

`esp_err_t mcpwm_fault_deinit` (`mcpwm_unit_t mcpwm_num`, `mcpwm_fault_signal_t fault_sig`)

Disable fault signal.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- `mcpwm_num`: set MCPWM unit(0-1)
- `fault_sig`: fault pin, which needs to be disabled

`esp_err_t mcpwm_capture_enable` (`mcpwm_unit_t mcpwm_num`, `mcpwm_capture_signal_t cap_sig`, `mcpwm_capture_on_edge_t cap_edge`, `uint32_t num_of_pulse`)

Initialize capture submodule.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- `mcpwm_num`: set MCPWM unit(0-1)
- `cap_edge`: set capture edge, BIT(0) - negative edge, BIT(1) - positive edge
- `cap_sig`: capture pin, which needs to be enabled
- `num_of_pulse`: count time between rising/falling edge between 2 \*(pulses mentioned), counter uses APB\_CLK

`esp_err_t mcpwm_capture_disable` (`mcpwm_unit_t mcpwm_num`, `mcpwm_capture_signal_t cap_sig`)

Disable capture signal.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- `mcpwm_num`: set MCPWM unit(0-1)
- `cap_sig`: capture pin, which needs to be disabled

`uint32_t mcpwm_capture_signal_get_value` (`mcpwm_unit_t mcpwm_num`, `mcpwm_capture_signal_t cap_sig`)

Get capture value.

**Return** Captured value

**Parameters**

- `mcpwm_num`: set MCPWM unit(0-1)
- `cap_sig`: capture pin on which value is to be measured

`uint32_t mcpwm_capture_signal_get_edge` (`mcpwm_unit_t mcpwm_num`, `mcpwm_capture_signal_t cap_sig`)

Get edge of capture signal.

**Return** Capture signal edge: 1 - positive edge, 2 - negative edge

**Parameters**

- `mcpwm_num`: set MCPWM unit(0-1)
- `cap_sig`: capture pin of whose edge is to be determined

*esp\_err\_t* **mcpwm\_sync\_enable** (*mcpwm\_unit\_t* *mcpwm\_num*, *mcpwm\_timer\_t* *timer\_num*, *mcpwm\_sync\_signal\_t* *sync\_sig*, *uint32\_t* *phase\_val*)

Initialize sync submodule.

#### Return

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

#### Parameters

- *mcpwm\_num*: set MCPWM unit(0-1)
- *timer\_num*: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers
- *sync\_sig*: set the synchronization pin, which needs to be enabled
- *phase\_val*: phase value in 1/1000 (for 86.7%, *phase\_val* = 867) which timer moves to on sync signal

*esp\_err\_t* **mcpwm\_sync\_disable** (*mcpwm\_unit\_t* *mcpwm\_num*, *mcpwm\_timer\_t* *timer\_num*)

Disable sync submodule on given timer.

#### Return

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

#### Parameters

- *mcpwm\_num*: set MCPWM unit(0-1)
- *timer\_num*: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers

*esp\_err\_t* **mcpwm\_isr\_register** (*mcpwm\_unit\_t* *mcpwm\_num*, void (\**fn*)) void \*

, void \**arg*, int *intr\_alloc\_flags*, *intr\_handle\_t* \**handle* Register MCPWM interrupt handler, the handler is an ISR. the handler will be attached to the same CPU core that this function is running on.

#### Return

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Function pointer error.

#### Parameters

- *mcpwm\_num*: set MCPWM unit(0-1)
- *fn*: interrupt handler function.
- *arg*: user-supplied argument passed to the 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.
- *arg*: parameter for handler function
- *handle*: pointer to return handle. If non-NULL, a handle for the interrupt will be returned here.

## Structures

**struct mcpwm\_pin\_config\_t**

MCPWM pin number for.

### Public Members

int **mcpwm0a\_out\_num**  
MCPWM0A out pin

int **mcpwm0b\_out\_num**  
MCPWM0A out pin

int **mcpwm1a\_out\_num**  
MCPWM0A out pin

int **mcpwm1b\_out\_num**  
MCPWM0A out pin

int **mcpwm2a\_out\_num**  
MCPWM0A out pin

```
int mcpwm2b_out_num  
    MCPWM0A out pin  
int mcpwm_sync0_in_num  
    SYNC0 in pin  
int mcpwm_sync1_in_num  
    SYNC1 in pin  
int mcpwm_sync2_in_num  
    SYNC2 in pin  
int mcpwm_fault0_in_num  
    FAULT0 in pin  
int mcpwm_fault1_in_num  
    FAULT1 in pin  
int mcpwm_fault2_in_num  
    FAULT2 in pin  
int mcpwm_cap0_in_num  
    CAP0 in pin  
int mcpwm_cap1_in_num  
    CAP1 in pin  
int mcpwm_cap2_in_num  
    CAP2 in pin  
struct mcpwm_config_t  
    MCPWM config structure.
```

### Public Members

```
uint32_t frequency  
    Set frequency of MCPWM in Hz  
float cmp_r_a  
    Set % duty cycle for operator a(MCPWMXA), i.e for 62.3% duty cycle, duty_a = 62.3  
float cmp_r_b  
    Set % duty cycle for operator b(MCPWMXB), i.e for 48% duty cycle, duty_b = 48.0  
mcpwm_duty_type_t duty_mode  
    Set type of duty cycle  
mcpwm_counter_type_t counter_mode  
    Set type of MCPWM counter  
struct mcpwm_carrier_config_t  
    MCPWM config carrier structure.
```

### Public Members

```
uint8_t carrier_period  
    Set carrier period = (carrier_period + 1)*800ns, carrier_period should be < 16  
uint8_t carrier_duty  
    Set carrier duty cycle, carrier_duty should be less than 8 (increment every 12.5%)  
uint8_t pulse_width_in_os  
    Set pulse width of first pulse in one shot mode = (carrier period)*(pulse_width_in_os + 1), should be less than 16
```

*mcpwm\_carrier\_os\_t* **carrier\_os\_mode**

Enable or disable carrier oneshot mode

*mcpwm\_carrier\_out\_ivt\_t* **carrier\_ivt\_mode**

Invert output of carrier

### Enumerations

**enum mcpwm\_io\_signals\_t**

IO signals for the MCPWM.

- 6 MCPWM output pins that generate PWM signals
- 3 MCPWM fault input pins to detect faults like overcurrent, overvoltage, etc.
- 3 MCPWM sync input pins to synchronize MCPWM outputs signals
- 3 MCPWM capture input pins to gather feedback from controlled motors, using e.g. hall sensors

*Values:*

**MCPWM0A** = 0

PWM0A output pin

**MCPWM0B**

PWM0B output pin

**MCPWM1A**

PWM1A output pin

**MCPWM1B**

PWM1B output pin

**MCPWM2A**

PWM2A output pin

**MCPWM2B**

PWM2B output pin

**MCPWM\_SYNC\_0**

SYNC0 input pin

**MCPWM\_SYNC\_1**

SYNC1 input pin

**MCPWM\_SYNC\_2**

SYNC2 input pin

**MCPWM\_FAULT\_0**

FAULT0 input pin

**MCPWM\_FAULT\_1**

FAULT1 input pin

**MCPWM\_FAULT\_2**

FAULT2 input pin

**MCPWM\_CAP\_0** = 84

CAP0 input pin

**MCPWM\_CAP\_1**

CAP1 input pin

**MCPWM\_CAP\_2**

CAP2 input pin

**enum mcpwm\_unit\_t**

Select MCPWM unit.

*Values:*

**MCPWM\_UNIT\_0 = 0**  
MCPWM unit0 selected

**MCPWM\_UNIT\_1**  
MCPWM unit1 selected

**MCPWM\_UNIT\_MAX**  
Num of MCPWM units on ESP32

**enum mcpwm\_timer\_t**  
Select MCPWM timer.

*Values:*

**MCPWM\_TIMER\_0 = 0**  
Select MCPWM timer0

**MCPWM\_TIMER\_1**  
Select MCPWM timer1

**MCPWM\_TIMER\_2**  
Select MCPWM timer2

**MCPWM\_TIMER\_MAX**  
Num of MCPWM timers on ESP32

**enum mcpwm\_operator\_t**  
Select MCPWM operator.

*Values:*

**MCPWM\_OPR\_A = 0**  
Select MCPWMXA, where 'X' is timer number

**MCPWM\_OPR\_B**  
Select MCPWMXB, where 'X' is timer number

**MCPWM\_OPR\_MAX**  
Num of operators to each timer of MCPWM

**enum mcpwm\_counter\_type\_t**  
Select type of MCPWM counter.

*Values:*

**MCPWM\_UP\_COUNTER = 1**  
For asymmetric MCPWM

**MCPWM\_DOWN\_COUNTER**  
For asymmetric MCPWM

**MCPWM\_UP\_DOWN\_COUNTER**  
For symmetric MCPWM, frequency is half of MCPWM frequency set

**MCPWM\_COUNTER\_MAX**  
Maximum counter mode

**enum mcpwm\_duty\_type\_t**  
Select type of MCPWM duty cycle mode.

*Values:*

**MCPWM\_DUTY\_MODE\_0 = 0**  
Active high duty, i.e. duty cycle proportional to high time for asymmetric MCPWM

**MCPWM\_DUTY\_MODE\_1**  
Active low duty, i.e. duty cycle proportional to low time for asymmetric MCPWM, out of phase(inverted) MCPWM

**MCPWM\_DUTY\_MODE\_MAX**

Num of duty cycle modes

**enum mcpwm\_carrier\_os\_t**

MCPWM carrier oneshot mode, in this mode the width of the first pulse of carrier can be programmed.

*Values:***MCPWM\_ONESHOT\_MODE\_DIS = 0**

Enable oneshot mode

**MCPWM\_ONESHOT\_MODE\_EN**

Disable oneshot mode

**enum mcpwm\_carrier\_out\_ivt\_t**

MCPWM carrier output inversion, high frequency carrier signal active with MCPWM signal is high.

*Values:***MCPWM\_CARRIER\_OUT\_IVT\_DIS = 0**

Enable carrier output inversion

**MCPWM\_CARRIER\_OUT\_IVT\_EN**

Disable carrier output inversion

**enum mcpwm\_sync\_signal\_t**

MCPWM select sync signal input.

*Values:***MCPWM\_SELECT\_SYNC0 = 4**

Select SYNC0 as input

**MCPWM\_SELECT\_SYNC1**

Select SYNC1 as input

**MCPWM\_SELECT\_SYNC2**

Select SYNC2 as input

**enum mcpwm\_fault\_signal\_t**

MCPWM select fault signal input.

*Values:***MCPWM\_SELECT\_F0 = 0**

Select F0 as input

**MCPWM\_SELECT\_F1**

Select F1 as input

**MCPWM\_SELECT\_F2**

Select F2 as input

**enum mcpwm\_fault\_input\_level\_t**

MCPWM select triggering level of fault signal.

*Values:***MCPWM\_LOW\_LEVEL\_TGR = 0**

Fault condition occurs when fault input signal goes from high to low, currently not supported

**MCPWM\_HIGH\_LEVEL\_TGR**

Fault condition occurs when fault input signal goes low to high

**enum mcpwm\_action\_on\_pwmxa\_t**

MCPWM select action to be taken on MCPWMXA when fault occurs.

*Values:***MCPWM\_NO\_CHANGE\_IN\_MCPWMXA = 0**

No change in MCPWMXA output



**MCPWM\_FORCE\_MCPWMXA\_LOW**  
Make MCPWMXA output low

**MCPWM\_FORCE\_MCPWMXA\_HIGH**  
Make MCPWMXA output high

**MCPWM\_TOG\_MCPWMXA**  
Make MCPWMXA output toggle

**enum mcpwm\_action\_on\_pwmxb\_t**  
MCPWM select action to be taken on MCPWMxB when fault occurs.

*Values:*

**MCPWM\_NO\_CHANGE\_IN\_MCPWMXB = 0**  
No change in MCPWMXB output

**MCPWM\_FORCE\_MCPWMXB\_LOW**  
Make MCPWMXB output low

**MCPWM\_FORCE\_MCPWMXB\_HIGH**  
Make MCPWMXB output high

**MCPWM\_TOG\_MCPWMXB**  
Make MCPWMXB output toggle

**enum mcpwm\_capture\_signal\_t**  
MCPWM select capture signal input.

*Values:*

**MCPWM\_SELECT\_CAP0 = 0**  
Select CAP0 as input

**MCPWM\_SELECT\_CAP1**  
Select CAP1 as input

**MCPWM\_SELECT\_CAP2**  
Select CAP2 as input

**enum mcpwm\_capture\_on\_edge\_t**  
MCPWM select capture starts from which edge.

*Values:*

**MCPWM\_NEG\_EDGE = 0**  
Capture starts from negative edge

**MCPWM\_POS\_EDGE**  
Capture starts from positive edge

**enum mcpwm\_deadtime\_type\_t**  
MCPWM deadtime types, used to generate deadtime, RED refers to rising edge delay and FED refers to falling edge delay.

*Values:*

**MCPWM\_BYPASS\_RED = 0**  
MCPWMXA = no change, MCPWMXB = falling edge delay

**MCPWM\_BYPASS\_FED**  
MCPWMXA = rising edge delay, MCPWMXB = no change

**MCPWM\_ACTIVE\_HIGH\_MODE**  
MCPWMXA = rising edge delay, MCPWMXB = falling edge delay

**MCPWM\_ACTIVE\_LOW\_MODE**  
MCPWMXA = compliment of rising edge delay, MCPWMXB = compliment of falling edge delay

**MCPWM\_ACTIVE\_HIGH\_COMPLIMENT\_MODE**

MCPWMXA = rising edge delay, MCPWMXB = compliment of falling edge delay

**MCPWM\_ACTIVE\_LOW\_COMPLIMENT\_MODE**

MCPWMXA = compliment of rising edge delay, MCPWMXB = falling edge delay

**MCPWM\_ACTIVE\_RED\_FED\_FROM\_PWMXA**

MCPWMXA = MCPWMXB = rising edge delay as well as falling edge delay, generated from MCPWMXA

**MCPWM\_ACTIVE\_RED\_FED\_FROM\_PWMXB**

MCPWMXA = MCPWMXB = rising edge delay as well as falling edge delay, generated from MCPWMXB

**MCPWM\_DEADTIME\_TYPE\_MAX**

## 2.3.9 Pulse Counter

### Introduction

The PCNT (Pulse Counter) module is designed to count the number of rising and/or falling edges of an input signal. Each pulse counter unit has a 16-bit signed counter register and two channels that can be configured to either increment or decrement the counter. Each channel has a signal input that accepts signal edges to be detected, as well as a control input that can be used to enable or disable the signal input. The inputs have optional filters that can be used to discard unwanted glitches in the signal.

### Functionality Overview

Description of functionality of this API has been broken down into four sections:

- *Configuration* - describes counter's configuration parameters and how to setup the counter.
- *Operating the Counter* - provides information on control functions to pause, measure and clear the counter.
- *Filtering Pulses* - describes options to filtering pulses and the counter control signals.
- *Using Interrupts* - presents how to trigger interrupts on specific states of the counter.

### Configuration

The PCNT module has eight independent counting “units” numbered from 0 to 7. In the API they are referred to using `pcnt_unit_t`. Each unit has two independent channels numbered as 0 and 1 and specified with `pcnt_channel_t`.

The configuration is provided separately per unit's channel using `pcnt_config_t` and covers:

- The unit and the channel number this configuration refers to.
- GPIO numbers of the pulse input and the pulse gate input.
- Two pairs of parameters: `pcnt_ctrl_mode_t` and `pcnt_count_mode_t` to define how the counter reacts depending on the the status of control signal and how counting is done positive / negative edge of the pulses.
- Two limit values (minimum / maximum) that are used to establish watchpoints and trigger interrupts when the pulse count is meeting particular limit.

Setting up of particular channel is then done by calling a function `pcnt_unit_config()` with above `pcnt_config_t` as the input parameter.

To disable the pulse or the control input pin in configuration, provide `PCNT_PIN_NOT_USED` instead of the GPIO number.

## Operating the Counter

After doing setup with `pcnt_unit_config()`, the counter immediately starts to operate. The accumulated pulse count can be checked by calling `pcnt_get_counter_value()`.

There are couple of functions that allow to control the counter's operation: `pcnt_counter_pause()`, `pcnt_counter_resume()` and `pcnt_counter_clear()`

It is also possible to dynamically change the previously set up counter modes with `pcnt_unit_config()` by calling `pcnt_set_mode()`.

If desired, the pulse input pin and the control input pin may be changed “on the fly” using `pcnt_set_pin()`. To disable particular input provide as a function parameter `PCNT_PIN_NOT_USED` instead of the GPIO number.

---

**注解:** For the counter not to miss any pulses, the pulse duration should be longer than one APB\_CLK cycle (12.5 ns). The pulses are sampled on the edges of the APB\_CLK clock and may be missed, if fall between the edges. This applies to counter operation with or without a *filer*.

---

## Filtering Pulses

The PCNT unit features filters on each of the pulse and control inputs, adding the option to ignore short glitches in the signals.

The length of ignored pulses is provided in APB\_CLK clock cycles by calling `pcnt_set_filter_value()`. The current filter setting may be checked with `pcnt_get_filter_value()`. The APB\_CLK clock is running at 80 MHz.

The filter is put into operation / suspended by calling `pcnt_filter_enable()` / `pcnt_filter_disable()`.

## Using Interrupts

There are five counter state watch events, defined in `pcnt_evt_type_t`, that are able to trigger an interrupt. The event happens on the pulse counter reaching specific values:

- Minimum or maximum count values: `counter_l_lim` or `counter_h_lim` provided in `pcnt_config_t` as discussed in *Configuration*
- Threshold 0 or Threshold 1 values set using function `pcnt_set_event_value()`.
- Pulse count = 0

To register, enable or disable an interrupt to service the above events, call `pcnt_isr_register()`, `pcnt_intr_enable()`. and `pcnt_intr_disable()`. To enable or disable events on reaching threshold values, you will also need to call functions `pcnt_event_enable()` and `pcnt_event_disable()`.

In order to check what are the threshold values currently set, use function `pcnt_get_event_value()`.

## Application Example

Pulse counter with control signal and event interrupt example: [peripherals/pcnt](#).

## API Reference

### Header File

- [driver/include/driver/pcnt.h](#)

**Functions**

*esp\_err\_t* **pcnt\_unit\_config** (*const pcnt\_config\_t* \**pcnt\_config*)

Configure Pulse Counter unit.

**Note** This function will disable three events: PCNT\_EVT\_L\_LIM, PCNT\_EVT\_H\_LIM, PCNT\_EVT\_ZERO.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- *pcnt\_config*: Pointer of Pulse Counter unit configure parameter

*esp\_err\_t* **pcnt\_get\_counter\_value** (*pcnt\_unit\_t* *pcnt\_unit*, *int16\_t* \**count*)

Get pulse counter value.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- *pcnt\_unit*: Pulse Counter unit number
- *count*: Pointer to accept counter value

*esp\_err\_t* **pcnt\_counter\_pause** (*pcnt\_unit\_t* *pcnt\_unit*)

Pause PCNT counter of PCNT unit.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- *pcnt\_unit*: PCNT unit number

*esp\_err\_t* **pcnt\_counter\_resume** (*pcnt\_unit\_t* *pcnt\_unit*)

Resume counting for PCNT counter.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- *pcnt\_unit*: PCNT unit number, select from *pcnt\_unit\_t*

*esp\_err\_t* **pcnt\_counter\_clear** (*pcnt\_unit\_t* *pcnt\_unit*)

Clear and reset PCNT counter value to zero.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- *pcnt\_unit*: PCNT unit number, select from *pcnt\_unit\_t*

*esp\_err\_t* **pcnt\_intr\_enable** (*pcnt\_unit\_t* *pcnt\_unit*)

Enable PCNT interrupt for PCNT unit.

**Note** Each Pulse counter unit has five watch point events that share the same interrupt. Configure events with *pcnt\_event\_enable()* and *pcnt\_event\_disable()*

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- *pcnt\_unit*: PCNT unit number

*esp\_err\_t* **pcnt\_intr\_disable** (*pcnt\_unit\_t* *pcnt\_unit*)

Disable PCNT interrupt for PCNT unit.

**Return**

- ESP\_OK Success

- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- `pcnt_unit`: PCNT unit number

*esp\_err\_t* **pcnt\_event\_enable** (*pcnt\_unit\_t* unit, *pcnt\_evt\_type\_t* evt\_type)

Enable PCNT event of PCNT unit.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- `unit`: PCNT unit number
- `evt_type`: Watch point event type. All enabled events share the same interrupt (one interrupt per pulse counter unit).

*esp\_err\_t* **pcnt\_event\_disable** (*pcnt\_unit\_t* unit, *pcnt\_evt\_type\_t* evt\_type)

Disable PCNT event of PCNT unit.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- `unit`: PCNT unit number
- `evt_type`: Watch point event type. All enabled events share the same interrupt (one interrupt per pulse counter unit).

*esp\_err\_t* **pcnt\_set\_event\_value** (*pcnt\_unit\_t* unit, *pcnt\_evt\_type\_t* evt\_type, int16\_t value)

Set PCNT event value of PCNT unit.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- `unit`: PCNT unit number
- `evt_type`: Watch point event type. All enabled events share the same interrupt (one interrupt per pulse counter unit).
- `value`: Counter value for PCNT event

*esp\_err\_t* **pcnt\_get\_event\_value** (*pcnt\_unit\_t* unit, *pcnt\_evt\_type\_t* evt\_type, int16\_t \*value)

Get PCNT event value of PCNT unit.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- `unit`: PCNT unit number
- `evt_type`: Watch point event type. All enabled events share the same interrupt (one interrupt per pulse counter unit).
- `value`: Pointer to accept counter value for PCNT event

*esp\_err\_t* **pcnt\_isr\_register** (void (\*fn)) void \*

, void \*arg, int intr\_alloc\_flags, *pcnt\_isr\_handle\_t* \*handle Register PCNT interrupt handler, the handler is an ISR. The handler will be attached to the same CPU core that this function is running on. Please do not use `pcnt_isr_service_install` if this function was called.

**Return**

- ESP\_OK Success
- ESP\_ERR\_NOT\_FOUND Can not find the interrupt that matches the flags.
- ESP\_ERR\_INVALID\_ARG Function pointer error.

**Parameters**

- `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. Calling `esp_intr_free` to unregister this ISR service if needed, but only if the handle is not NULL.

*esp\_err\_t* `pcnt_set_pin` (*pcnt\_unit\_t* unit, *pcnt\_channel\_t* channel, int pulse\_io, int ctrl\_io)

Configure PCNT pulse signal input pin and control input pin.

**Note** Set the signal input to `PCNT_PIN_NOT_USED` if unused.

**Return**

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error

**Parameters**

- `unit`: PCNT unit number
- `channel`: PCNT channel number
- `pulse_io`: Pulse signal input GPIO
- `ctrl_io`: Control signal input GPIO

*esp\_err\_t* `pcnt_filter_enable` (*pcnt\_unit\_t* unit)

Enable PCNT input filter.

**Return**

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error

**Parameters**

- `unit`: PCNT unit number

*esp\_err\_t* `pcnt_filter_disable` (*pcnt\_unit\_t* unit)

Disable PCNT input filter.

**Return**

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error

**Parameters**

- `unit`: PCNT unit number

*esp\_err\_t* `pcnt_set_filter_value` (*pcnt\_unit\_t* unit, uint16\_t filter\_val)

Set PCNT filter value.

**Note** `filter_val` is a 10-bit value, so the maximum `filter_val` should be limited to 1023.

**Return**

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error

**Parameters**

- `unit`: PCNT unit number
- `filter_val`: PCNT signal filter value, counter in `APB_CLK` cycles. Any pulses lasting shorter than this will be ignored when the filter is enabled.

*esp\_err\_t* `pcnt_get_filter_value` (*pcnt\_unit\_t* unit, uint16\_t \*filter\_val)

Get PCNT filter value.

**Return**

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error

**Parameters**

- `unit`: PCNT unit number
- `filter_val`: Pointer to accept PCNT filter value.

*esp\_err\_t* `pcnt_set_mode` (*pcnt\_unit\_t* unit, *pcnt\_channel\_t* channel, *pcnt\_count\_mode\_t* pos\_mode, *pcnt\_count\_mode\_t* neg\_mode, *pcnt\_ctrl\_mode\_t* hctrl\_mode, *pcnt\_ctrl\_mode\_t* lctrl\_mode)

Set PCNT counter mode.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- `unit`: PCNT unit number
- `channel`: PCNT channel number
- `pos_mode`: Counter mode when detecting positive edge
- `neg_mode`: Counter mode when detecting negative edge
- `hctrl_mode`: Counter mode when control signal is high level
- `lctrl_mode`: Counter mode when control signal is low level

*esp\_err\_t* **pcnt\_isr\_handler\_add** (*pcnt\_unit\_t* unit, void (\**isr\_handler*)) void \*, void \**args* Add ISR handler for specified unit.

Call this function after using `pcnt_isr_service_install()` to install the PCNT driver's ISR handler service.

The ISR handlers do not need to be declared with `IRAM_ATTR`, unless you pass the `ESP_INTR_FLAG_IRAM` flag when allocating the ISR in `pcnt_isr_service_install()`.

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 PCNT interrupt handler due to the additional level of indirection.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- `unit`: PCNT unit number
- `isr_handler`: Interrupt handler function.
- `args`: Parameter for handler function

*esp\_err\_t* **pcnt\_isr\_service\_install** (int *intr\_alloc\_flags*)

Install PCNT ISR service.

**Note** We can manage different interrupt service for each unit. This function will use the default ISR handle service, Calling `pcnt_isr_service_uninstall` to uninstall the default service if needed. Please do not use `pcnt_isr_register` if this function was called.

**Return**

- ESP\_OK Success
- ESP\_ERR\_NO\_MEM No memory to install this service
- ESP\_ERR\_INVALID\_STATE ISR service already installed

**Parameters**

- `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.

void **pcnt\_isr\_service\_uninstall** (void)

Uninstall PCNT ISR service, freeing related resources.

*esp\_err\_t* **pcnt\_isr\_handler\_remove** (*pcnt\_unit\_t* unit)

Delete ISR handler for specified unit.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- `unit`: PCNT unit number

**Structures**

**struct** **pcnt\_config\_t**

Pulse Counter configuration for a single channel.

**Public Members****int pulse\_gpio\_num**

Pulse input GPIO number, if you want to use GPIO16, enter pulse\_gpio\_num = 16, a negative value will be ignored

**int ctrl\_gpio\_num**

Control signal input GPIO number, a negative value will be ignored

*pcnt\_ctrl\_mode\_t* **lctrl\_mode**

PCNT low control mode

*pcnt\_ctrl\_mode\_t* **hctrl\_mode**

PCNT high control mode

*pcnt\_count\_mode\_t* **pos\_mode**

PCNT positive edge count mode

*pcnt\_count\_mode\_t* **neg\_mode**

PCNT negative edge count mode

**int16\_t counter\_h\_lim**

Maximum counter value

**int16\_t counter\_l\_lim**

Minimum counter value

*pcnt\_unit\_t* **unit**

PCNT unit number

*pcnt\_channel\_t* **channel**

the PCNT channel

**Macros****PCNT\_PIN\_NOT\_USED**

When selected for a pin, this pin will not be used

**Type Definitions****typedef intr\_handle\_t pcnt\_isr\_handle\_t****Enumerations****enum pcnt\_ctrl\_mode\_t**

Selection of available modes that determine the counter's action depending on the state of the control signal's input GPIO.

**Note** Configuration covers two actions, one for high, and one for low level on the control input*Values:***PCNT\_MODE\_KEEP = 0**

Control mode: won't change counter mode

**PCNT\_MODE\_REVERSE = 1**

Control mode: invert counter mode(increase -&gt; decrease, decrease -&gt; increase)

**PCNT\_MODE\_DISABLE = 2**

Control mode: Inhibit counter(counter value will not change in this condition)

**PCNT\_MODE\_MAX****enum pcnt\_count\_mode\_t**

Selection of available modes that determine the counter's action on the edge of the pulse signal's input GPIO.

**Note** Configuration covers two actions, one for positive, and one for negative edge on the pulse input



*Values:*

**PCNT\_COUNT\_DIS = 0**

Counter mode: Inhibit counter(counter value will not change in this condition)

**PCNT\_COUNT\_INC = 1**

Counter mode: Increase counter value

**PCNT\_COUNT\_DEC = 2**

Counter mode: Decrease counter value

**PCNT\_COUNT\_MAX**

**enum pcnt\_unit\_t**

Selection of all available PCNT units.

*Values:*

**PCNT\_UNIT\_0 = 0**

PCNT unit 0

**PCNT\_UNIT\_1 = 1**

PCNT unit 1

**PCNT\_UNIT\_2 = 2**

PCNT unit 2

**PCNT\_UNIT\_3 = 3**

PCNT unit 3

**PCNT\_UNIT\_4 = 4**

PCNT unit 4

**PCNT\_UNIT\_5 = 5**

PCNT unit 5

**PCNT\_UNIT\_6 = 6**

PCNT unit 6

**PCNT\_UNIT\_7 = 7**

PCNT unit 7

**PCNT\_UNIT\_MAX**

**enum pcnt\_channel\_t**

Selection of channels available for a single PCNT unit.

*Values:*

**PCNT\_CHANNEL\_0 = 0x00**

PCNT channel 0

**PCNT\_CHANNEL\_1 = 0x01**

PCNT channel 1

**PCNT\_CHANNEL\_MAX**

**enum pcnt\_evt\_type\_t**

Selection of counter's events that may trigger an interrupt.

*Values:*

**PCNT\_EVT\_L\_LIM = 0**

PCNT watch point event: Minimum counter value

**PCNT\_EVT\_H\_LIM = 1**

PCNT watch point event: Maximum counter value

**PCNT\_EVT\_THRES\_0 = 2**

PCNT watch point event: threshold0 value event

**PCNT\_EVT\_THRES\_1 = 3**

PCNT watch point event: threshold1 value event

**PCNT\_EVT\_ZERO = 4**

PCNT watch point event: counter value zero event

**PCNT\_EVT\_MAX**

### 2.3.10 RMT

The RMT (Remote Control) module driver can be used to send and receive infrared remote control signals. Due to flexibility of RMT module, the driver can also be used to generate or receive many other types of signals.

The signal, which consists of a series of pulses, is generated by RMT's transmitter based on a list of values. The values define the pulse duration and a binary level, see below. The transmitter can also provide a carrier and modulate it with provided pulses.

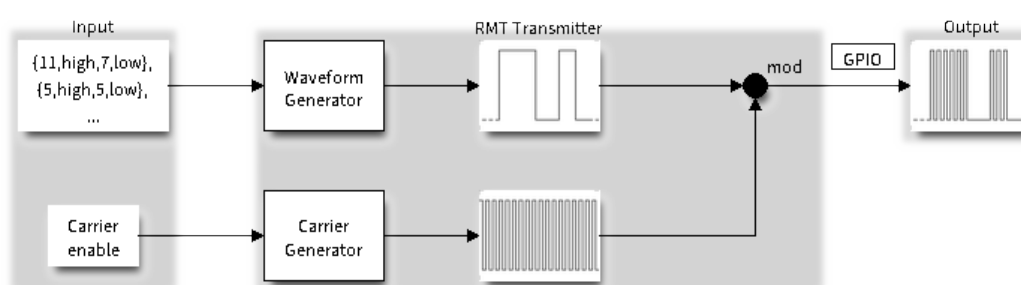


图 19: RMT Transmitter Overview

The reverse operation is performed by the receiver, where a series of pulses is decoded into a list of values containing the pulse duration and binary level. A filter may be applied to remove high frequency noise from the input signal.

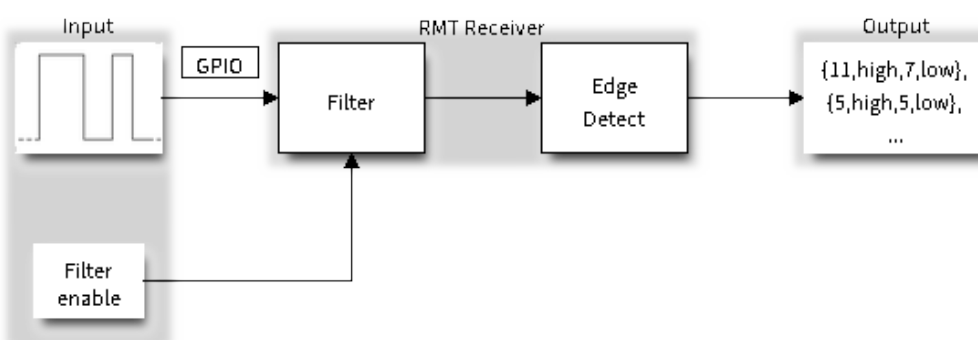


图 20: RMT Receiver Overview

There couple of typical steps to setup and operate the RMT and they are discussed in the following sections:

1. *Configure Driver*
2. *Transmit Data or Receive Data*
3. *Change Operation Parameters*
4. *Use Interrupts*

The RMT has eight channels numbered from zero to seven. Each channel is able to independently transmit or receive data. They are referred to using indexes defined in structure `rmt_channel_t`.

## Configure Driver

There are several parameters that define how particular channel operates. Most of these parameters are configured by setting specific members of `rmt_config_t` structure. Some of the parameters are common to both transmit or receive mode, and some are mode specific. They are all discussed below.

### Common Parameters

- The **channel** to be configured, select one from the `rmt_channel_t` enumerator.
- The RMT **operation mode** - whether this channel is used to transmit or receive data, selected by setting a **rmt\_mode** members to one of the values from `rmt_mode_t`.
- What is the **pin number** to transmit or receive RMT signals, selected by setting **gpio\_num**.
- How many **memory blocks** will be used by the channel, set with **mem\_block\_num**.
- A **clock divider**, that will determine the range of pulse length generated by the RMT transmitter or discriminated by the receiver. Selected by setting **clk\_div** to a value within [1 .. 255] range. The RMT source clock is typically APB CLK, 80Mhz by default.

---

**注解:** The period of a square wave after the clock divider is called a ‘tick’ . The length of the pulses generated by the RMT transmitter or discriminated by the receiver is configured in number of ‘ticks’ .

---

There are also couple of specific parameters that should be set up depending if selected channel is configured in *Transmit Mode* or *Receive Mode*:

**Transmit Mode** When configuring channel in transmit mode, set **tx\_config** and the following members of `rmt_tx_config_t`:

- Transmit the currently configured data items in a loop - **loop\_en**
- Enable the RMT carrier signal - **carrier\_en**
- Frequency of the carrier in Hz - **carrier\_freq\_hz**
- Duty cycle of the carrier signal in percent (%) - **carrier\_duty\_percent**
- Level of the RMT output, when the carrier is applied - **carrier\_level**
- Enable the RMT output if idle - **idle\_output\_en**
- Set the signal level on the RMT output if idle - **idle\_level**

**Receive Mode** In receive mode, set **rx\_config** and the following members of `rmt_rx_config_t`:

- Enable a filter on the input of the RMT receiver - **filter\_en**
- A threshold of the filter, set in the number of ticks - **filter\_ticks\_thresh**. Pulses shorter than this setting will be filtered out. Note, that the range of entered tick values is [0..255].
- A pulse length threshold that will turn the RMT receiver idle, set in number of ticks - **idle\_threshold**. The receiver will ignore pulses longer than this setting.

**Finalize Configuration** Once the `rmt_config_t` structure is populated with parameters, it should be then invoked with `rmt_config()` to make the configuration effective.

The last configuration step is installation of the driver in memory by calling `rmt_driver_install()`. If `rx_buf_size` parameter of this function is > 0, then a ring buffer for incoming data will be allocated. A default ISR handler will be installed, see a note in *Use Interrupts*.

Now, depending on how the channel is configured, we are ready to either *Transmit Data* or *Receive Data*. This is described in next two sections.

### Transmit Data

Before being able to transmit some RMT pulses, we need to define the pulse pattern. The minimum pattern recognized by the RMT controller, later called an ‘item’ , is provided in a structure `rmt_item32_t`, see

[soc/esp32/include/soc/rmt\\_struct.h](#). Each item consists of two pairs of two values. The first value in a pair describes the signal duration in ticks and is 15 bits long, the second provides the signal level (high or low) and is contained in a single bit. A block of couple of items and the structure of an item is presented below.

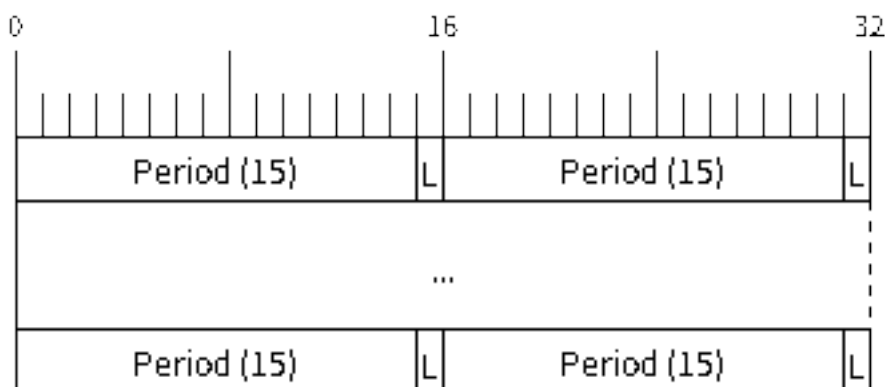


图 21: Structure of RMT items (L - signal level)

For a simple example how to define a block of items see [peripherals/rmt\\_tx](#).

The items are provided to the RMT controller by calling function `rmt_write_items()`. This function also automatically triggers start of transmission. It may be called to wait for transmission completion or exit just after transmission start. In such case you can wait for the transmission end by calling `rmt_wait_tx_done()`. This function does not limit the number of data items to transmit. It is using an interrupt to successively copy the new data chunks to RMT's internal memory as previously provided data are sent out.

Another way to provide data for transmission is by calling `rmt_fill_tx_items()`. In this case transmission is not started automatically. To control the transmission process use `rmt_tx_start()` and `rmt_tx_stop()`. The number of items to sent is restricted by the size of memory blocks allocated in the RMT controller's internal memory, see `rmt_set_mem_block_num()`.

## Receive Data

Before starting the receiver we need some storage for incoming items. The RMT controller has 512 x 32-bits of internal RAM shared between all eight channels. In typical scenarios it is not enough as an ultimate storage for all incoming (and outgoing) items. Therefore this API supports retrieval of incoming items on the fly to save them in a ring buffer of a size defined by the user. The size is provided when calling `rmt_driver_install()` discussed above. To get a handle to this buffer call `rmt_get_ringbuf_handle()`.

With the above steps complete we can start the receiver by calling `rmt_rx_start()` and then move to checking what's inside the buffer. To do so, you can use common FreeRTOS functions that interact with the ring buffer. Please see an example how to do it in [peripherals/rmt\\_nec\\_tx\\_rx](#).

To stop the receiver, call `rmt_rx_stop()`.

## Change Operation Parameters

Previously described function `rmt_config()` provides a convenient way to set several configuration parameters in one shot. This is usually done on application start. Then, when the application is running, the API provides an alternate way to update individual parameters by calling dedicated functions. Each function refers to the specific RMT channel provided as the first input parameter. Most of the functions have `_get_` counterpart to read back the currently configured value.

### Parameters Common to Transmit and Receive Mode

- Selection of a GPIO pin number on the input or output of the RMT - `rmt_set_pin()`
- Number of memory blocks allocated for the incoming or outgoing data - `rmt_set_mem_pd()`
- Setting of the clock divider - `rmt_set_clk_div()`
- Selection of the clock source, note that currently one clock source is supported, the APB clock which is 80Mhz - `rmt_set_source_clk()`

### Transmit Mode Parameters

- Enable or disable the loop back mode for the transmitter - `rmt_set_tx_loop_mode()`
- Binary level on the output to apply the carrier - `rmt_set_tx_carrier()`, selected from `rmt_carrier_level_t`
- Determines the binary level on the output when transmitter is idle - `rmt_set_idle_level()`, selected from `rmt_idle_level_t`

### Receive Mode Parameters

- The filter setting - `rmt_set_rx_filter()`
- The receiver threshold setting - `rmt_set_rx_idle_thresh()`
- Whether the transmitter or receiver is entitled to access RMT's memory - `rmt_set_memory_owner()`, selection is from `rmt_mem_owner_t`.

### Use Interrupts

Registering of an interrupt handler for the RMT controller is done by calling `rmt_isr_register()`.

---

**注解:** When calling `rmt_driver_install()` to use the system RMT driver, a default ISR is being installed. In such a case you cannot register a generic ISR handler with `rmt_isr_register()`.

---

The RMT controller triggers interrupts on four specific events described below. To enable interrupts on these events, the following functions are provided:

- The RMT receiver has finished receiving a signal - `rmt_set_rx_intr_en()`
- The RMT transmitter has finished transmitting the signal - `rmt_set_tx_intr_en()`
- The number of events the transmitter has sent matches a threshold value `rmt_set_tx_thr_intr_en()`
- Ownership to the RMT memory block has been violated - `rmt_set_err_intr_en()`

Setting or clearing an interrupt enable mask for specific channels and events may be also done by calling `rmt_set_intr_enable_mask()` or `rmt_clr_intr_enable_mask()`.

When servicing an interrupt within an ISR, the interrupt need to explicitly cleared. To do so, set specific bits described as `RMT.int_clr.val.chN_event_name` and defined as a volatile struct in `soc/esp32/include/soc/rmt_struct.h`, where N is the RMT channel number [0, 7] and the `event_name` is one of four events described above.

If you do not need an ISR anymore, you can deregister it by calling a function `rmt_isr_deregister()`.

### Uninstall Driver

If the RMT driver has been installed with `rmt_driver_install()` for some specific period of time and then not required, the driver may be removed to free allocated resources by calling `rmt_driver_uninstall()`.

### Application Examples

- A simple RMT TX example: [peripherals/rmt\\_tx](#).
- NEC remote control TX and RX example: [peripherals/rmt\\_nec\\_tx\\_rx](#).

## API Reference

### Header File

- [driver/include/driver/rmt.h](#)

### Functions

*esp\_err\_t* **rmt\_set\_clk\_div** (*rmt\_channel\_t* channel, uint8\_t div\_cnt)

Set RMT clock divider, channel clock is divided from source clock.

#### Return

- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_OK Success

#### Parameters

- channel: RMT channel (0-7)
- div\_cnt: RMT counter clock divider

*esp\_err\_t* **rmt\_get\_clk\_div** (*rmt\_channel\_t* channel, uint8\_t \*div\_cnt)

Get RMT clock divider, channel clock is divided from source clock.

#### Return

- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_OK Success

#### Parameters

- channel: RMT channel (0-7)
- div\_cnt: pointer to accept RMT counter divider

*esp\_err\_t* **rmt\_set\_rx\_idle\_thresh** (*rmt\_channel\_t* channel, uint16\_t thresh)

Set RMT RX idle threshold value.

In receive mode, when no edge is detected on the input signal for longer than idle\_thres channel clock cycles, the receive process is finished.

#### Return

- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_OK Success

#### Parameters

- channel: RMT channel (0-7)
- thresh: RMT RX idle threshold

*esp\_err\_t* **rmt\_get\_rx\_idle\_thresh** (*rmt\_channel\_t* channel, uint16\_t \*thresh)

Get RMT idle threshold value.

In receive mode, when no edge is detected on the input signal for longer than idle\_thres channel clock cycles, the receive process is finished.

#### Return

- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_OK Success

#### Parameters

- channel: RMT channel (0-7)
- thresh: pointer to accept RMT RX idle threshold value

*esp\_err\_t* **rmt\_set\_mem\_block\_num** (*rmt\_channel\_t* channel, uint8\_t rmt\_mem\_num)

Set RMT memory block number for RMT channel.

This function is used to configure the amount of memory blocks allocated to channel n. The 8 channels share a 512x32-bit RAM block which can be read and written by the processor cores over the APB bus, as well as read by the transmitters and written by the receivers.

The RAM address range for channel n is start\_addr\_CHn to end\_addr\_CHn, which are defined by: Memory block start address is RMT\_CHANNEL\_MEM(n) (in soc/rmt\_reg.h), that is, start\_addr\_chn = RMT base address + 0x800 + 64 \* 4 \* n, and end\_addr\_chn = RMT base address + 0x800 + 64 \* 4 \* n + 64 \* 4 \* RMT\_MEM\_SIZE\_CHn mod 512 \* 4

**Note** If memory block number of one channel is set to a value greater than 1, this channel will occupy the memory block of the next channel. Channel 0 can use at most 8 blocks of memory, accordingly channel 7 can only use one memory block.

**Return**

- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_OK Success

**Parameters**

- channel: RMT channel (0-7)
- rmt\_mem\_num: RMT RX memory block number, one block has 64 \* 32 bits.

*esp\_err\_t* **rmt\_get\_mem\_block\_num** (*rmt\_channel\_t* channel, uint8\_t \*rmt\_mem\_num)

Get RMT memory block number.

**Return**

- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_OK Success

**Parameters**

- channel: RMT channel (0-7)
- rmt\_mem\_num: Pointer to accept RMT RX memory block number

*esp\_err\_t* **rmt\_set\_tx\_carrier** (*rmt\_channel\_t* channel, bool carrier\_en, uint16\_t high\_level, uint16\_t low\_level, *rmt\_carrier\_level\_t* carrier\_level)

Configure RMT carrier for TX signal.

Set different values for carrier\_high and carrier\_low to set different frequency of carrier. The unit of carrier\_high/low is the source clock tick, not the divided channel counter clock.

**Return**

- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_OK Success

**Parameters**

- channel: RMT channel (0-7)
- carrier\_en: Whether to enable output carrier.
- high\_level: High level duration of carrier
- low\_level: Low level duration of carrier.
- carrier\_level: Configure the way carrier wave is modulated for channel 0-7.
  - 1' b1:transmit on low output level
  - 1' b0:transmit on high output level

*esp\_err\_t* **rmt\_set\_mem\_pd** (*rmt\_channel\_t* channel, bool pd\_en)

Set RMT memory in low power mode.

Reduce power consumed by memory. 1:memory is in low power state.

**Return**

- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_OK Success

**Parameters**

- channel: RMT channel (0-7)
- pd\_en: RMT memory low power enable.

*esp\_err\_t* **rmt\_get\_mem\_pd** (*rmt\_channel\_t* channel, bool \*pd\_en)

Get RMT memory low power mode.

**Return**

- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_OK Success

**Parameters**

- channel: RMT channel (0-7)
- pd\_en: Pointer to accept RMT memory low power mode.

*esp\_err\_t* **rmt\_tx\_start** (*rmt\_channel\_t* channel, bool tx\_idx\_rst)

Set RMT start sending data from memory.

**Return**

- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_OK Success

**Parameters**

- `channel`: RMT channel (0-7)
- `tx_idx_rst`: Set true to reset memory index for TX. Otherwise, transmitter will continue sending from the last index in memory.

*esp\_err\_t* **rmt\_tx\_stop**(*rmt\_channel\_t* channel)

Set RMT stop sending.

**Return**

- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_OK Success

**Parameters**

- `channel`: RMT channel (0-7)

*esp\_err\_t* **rmt\_rx\_start**(*rmt\_channel\_t* channel, bool *rx\_idx\_rst*)

Set RMT start receiving data.

**Return**

- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_OK Success

**Parameters**

- `channel`: RMT channel (0-7)
- `rx_idx_rst`: Set true to reset memory index for receiver. Otherwise, receiver will continue receiving data to the last index in memory.

*esp\_err\_t* **rmt\_rx\_stop**(*rmt\_channel\_t* channel)

Set RMT stop receiving data.

**Return**

- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_OK Success

**Parameters**

- `channel`: RMT channel (0-7)

*esp\_err\_t* **rmt\_memory\_rw\_rst**(*rmt\_channel\_t* channel)

Reset RMT TX/RX memory index.

**Return**

- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_OK Success

**Parameters**

- `channel`: RMT channel (0-7)

*esp\_err\_t* **rmt\_set\_memory\_owner**(*rmt\_channel\_t* channel, *rmt\_mem\_owner\_t* owner)

Set RMT memory owner.

**Return**

- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_OK Success

**Parameters**

- `channel`: RMT channel (0-7)
- `owner`: To set when the transmitter or receiver can process the memory of channel.

*esp\_err\_t* **rmt\_get\_memory\_owner**(*rmt\_channel\_t* channel, *rmt\_mem\_owner\_t* \*owner)

Get RMT memory owner.

**Return**

- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_OK Success

**Parameters**

- `channel`: RMT channel (0-7)



- `owner`: Pointer to get memory owner.

`esp_err_t rmt_set_tx_loop_mode(rmt_channel_t channel, bool loop_en)`

Set RMT tx loop mode.

#### Return

- `ESP_ERR_INVALID_ARG` Parameter error
- `ESP_OK` Success

#### Parameters

- `channel`: RMT channel (0-7)
- `loop_en`: Enable RMT transmitter loop sending mode. If set true, transmitter will continue sending from the first data to the last data in channel 0-7 over and over again in a loop.

`esp_err_t rmt_get_tx_loop_mode(rmt_channel_t channel, bool *loop_en)`

Get RMT tx loop mode.

#### Return

- `ESP_ERR_INVALID_ARG` Parameter error
- `ESP_OK` Success

#### Parameters

- `channel`: RMT channel (0-7)
- `loop_en`: Pointer to accept RMT transmitter loop sending mode.

`esp_err_t rmt_set_rx_filter(rmt_channel_t channel, bool rx_filter_en, uint8_t thresh)`

Set RMT RX filter.

In receive mode, channel 0-7 will ignore input pulse when the pulse width is smaller than threshold. Counted in source clock, not divided counter clock.

#### Return

- `ESP_ERR_INVALID_ARG` Parameter error
- `ESP_OK` Success

#### Parameters

- `channel`: RMT channel (0-7)
- `rx_filter_en`: To enable RMT receiver filter.
- `thresh`: Threshold of pulse width for receiver.

`esp_err_t rmt_set_source_clk(rmt_channel_t channel, rmt_source_clk_t base_clk)`

Set RMT source clock.

RMT module has two clock sources:

1. APB clock which is 80Mhz
2. REF tick clock, which would be 1Mhz (not supported in this version).

#### Return

- `ESP_ERR_INVALID_ARG` Parameter error
- `ESP_OK` Success

#### Parameters

- `channel`: RMT channel (0-7)
- `base_clk`: To choose source clock for RMT module.

`esp_err_t rmt_get_source_clk(rmt_channel_t channel, rmt_source_clk_t *src_clk)`

Get RMT source clock.

RMT module has two clock sources:

1. APB clock which is 80Mhz
2. REF tick clock, which would be 1Mhz (not supported in this version).

#### Return

- `ESP_ERR_INVALID_ARG` Parameter error
- `ESP_OK` Success

#### Parameters

- `channel`: RMT channel (0-7)
- `src_clk`: Pointer to accept source clock for RMT module.

*esp\_err\_t* **rmt\_set\_idle\_level** (*rmt\_channel\_t* channel, bool *idle\_out\_en*, *rmt\_idle\_level\_t* level)  
Set RMT idle output level for transmitter.

**Return**

- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_OK Success

**Parameters**

- channel: RMT channel (0-7)
- idle\_out\_en: To enable idle level output.
- level: To set the output signal's level for channel 0-7 in idle state.

*esp\_err\_t* **rmt\_get\_idle\_level** (*rmt\_channel\_t* channel, bool \**idle\_out\_en*, *rmt\_idle\_level\_t* \**level*)  
Get RMT idle output level for transmitter.

**Return**

- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_OK Success

**Parameters**

- channel: RMT channel (0-7)
- idle\_out\_en: Pointer to accept value of enable idle.
- level: Pointer to accept value of output signal's level in idle state for specified channel.

*esp\_err\_t* **rmt\_get\_status** (*rmt\_channel\_t* channel, uint32\_t \**status*)  
Get RMT status.

**Return**

- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_OK Success

**Parameters**

- channel: RMT channel (0-7)
- status: Pointer to accept channel status. Please refer to RMT\_CHnSTATUS\_REG(n=0~7) in `rmt_reg.h` for more details of each field.

void **rmt\_set\_intr\_enable\_mask** (uint32\_t *mask*)  
Set mask value to RMT interrupt enable register.

**Parameters**

- mask: Bit mask to set to the register

void **rmt\_clr\_intr\_enable\_mask** (uint32\_t *mask*)  
Clear mask value to RMT interrupt enable register.

**Parameters**

- mask: Bit mask to clear the register

*esp\_err\_t* **rmt\_set\_rx\_intr\_en** (*rmt\_channel\_t* channel, bool *en*)  
Set RMT RX interrupt enable.

**Return**

- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_OK Success

**Parameters**

- channel: RMT channel (0 - 7)
- en: enable or disable RX interrupt.

*esp\_err\_t* **rmt\_set\_err\_intr\_en** (*rmt\_channel\_t* channel, bool *en*)  
Set RMT RX error interrupt enable.

**Return**

- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_OK Success

**Parameters**

- channel: RMT channel (0 - 7)
- en: enable or disable RX err interrupt.

`esp_err_t rmt_set_tx_intr_en(rmt_channel_t channel, bool en)`

Set RMT TX interrupt enable.

**Return**

- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_OK Success

**Parameters**

- channel: RMT channel (0 - 7)
- en: enable or disable TX interrupt.

`esp_err_t rmt_set_tx_thr_intr_en(rmt_channel_t channel, bool en, uint16_t evt_thresh)`

Set RMT TX threshold event interrupt enable.

An interrupt will be triggered when the number of transmitted items reaches the threshold value

**Return**

- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_OK Success

**Parameters**

- channel: RMT channel (0 - 7)
- en: enable or disable TX event interrupt.
- evt\_thresh: RMT event interrupt threshold value

`esp_err_t rmt_set_pin(rmt_channel_t channel, rmt_mode_t mode, gpio_num_t gpio_num)`

Set RMT pin.

**Return**

- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_OK Success

**Parameters**

- channel: RMT channel (0 - 7)
- mode: TX or RX mode for RMT
- gpio\_num: GPIO number to transmit or receive the signal.

`esp_err_t rmt_config(const rmt_config_t *rmt_param)`

Configure RMT parameters.

**Return**

- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_OK Success

**Parameters**

- rmt\_param: RMT parameter struct

`esp_err_t rmt_isr_register(void (*fn)) void *`

`, void *arg, int intr_alloc_flags, rmt_isr_handle_t *handle` Register RMT interrupt handler, the handler is an ISR.

The handler will be attached to the same CPU core that this function is running on.

**Note** If you already called `rmt_driver_install` to use system RMT driver, please do not register ISR handler again.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Function pointer error.
- ESP\_FAIL System driver installed, can not register ISR handler for RMT

**Parameters**

- fn: Interrupt handler function.
- arg: Parameter for the 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: If non-zero, a handle to later clean up the ISR gets stored here.

`esp_err_t rmt_isr_deregister(rmt_isr_handle_t handle)`

Deregister previously registered RMT interrupt handler.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Handle invalid

**Parameters**

- handle: Handle obtained from rmt\_isr\_register

`esp_err_t rmt_fill_tx_items(rmt_channel_t channel, const rmt_item32_t *item, uint16_t item_num, uint16_t mem_offset)`

Fill memory data of channel with given RMT items.

**Return**

- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_OK Success

**Parameters**

- channel: RMT channel (0 - 7)
- item: Pointer of items.
- item\_num: RMT sending items number.
- mem\_offset: Index offset of memory.

`esp_err_t rmt_driver_install(rmt_channel_t channel, size_t rx_buf_size, int intr_alloc_flags)`

Initialize RMT driver.

**Return**

- ESP\_ERR\_INVALID\_STATE Driver is already installed, call rmt\_driver\_uninstall first.
- ESP\_ERR\_NO\_MEM Memory allocation failure
- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_OK Success

**Parameters**

- channel: RMT channel (0 - 7)
- rx\_buf\_size: Size of RMT RX ringbuffer. Can be 0 if the RX ringbuffer is not used.
- intr\_alloc\_flags: Flags for the RMT driver interrupt handler. Pass 0 for default flags. See esp\_intr\_alloc.h for details. If ESP\_INTR\_FLAG\_IRAM is used, please do not use the memory allocated from psram when calling rmt\_write\_items.

`esp_err_t rmt_driver_uninstall(rmt_channel_t channel)`

Uninstall RMT driver.

**Return**

- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_OK Success

**Parameters**

- channel: RMT channel (0 - 7)

`esp_err_t rmt_get_channel_status(rmt_channel_status_result_t *channel_status)`

Get the current status of eight channels.

**Note** Do not call this function if it is possible that rmt\_driver\_uninstall will be called at the same time.

**Return**

- ESP\_ERR\_INVALID\_ARG Parameter is NULL
- ESP\_OK Success

**Parameters**

- [out] channel\_status: store the current status of each channel

`esp_err_t rmt_write_items(rmt_channel_t channel, const rmt_item32_t *rmt_item, int item_num, bool wait_tx_done)`

RMT send waveform from rmt\_item array.

This API allows user to send waveform with any length.

**Note** This function will not copy data, instead, it will point to the original items, and send the waveform items. If wait\_tx\_done is set to true, this function will block and will not return until all items have been sent out. If wait\_tx\_done is set to false, this function will return immediately, and the driver interrupt will

continue sending the items. We must make sure the item data will not be damaged when the driver is still sending items in driver interrupt.

**Return**

- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_OK Success

**Parameters**

- `channel`: RMT channel (0 - 7)
- `rmt_item`: head point of RMT items array. If ESP\_INTR\_FLAG\_IRAM is used, please do not use the memory allocated from psram when calling `rmt_write_items`.
- `item_num`: RMT data item number.
- `wait_tx_done`:
  - If set 1, it will block the task and wait for sending done.
  - If set 0, it will not wait and return immediately.

`esp_err_t rmt_wait_tx_done(rmt_channel_t channel, TickType_t wait_time)`

Wait RMT TX finished.

**Return**

- ESP\_OK RMT Tx done successfully
- ESP\_ERR\_TIMEOUT Exceeded the 'wait\_time' given
- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_FAIL Driver not installed

**Parameters**

- `channel`: RMT channel (0 - 7)
- `wait_time`: Maximum time in ticks to wait for transmission to be complete. If set 0, return immediately with ESP\_ERR\_TIMEOUT if TX is busy (polling).

`esp_err_t rmt_get_ringbuf_handle(rmt_channel_t channel, RingbufHandle_t *buf_handle)`

Get ringbuffer from RMT.

Users can get the RMT RX ringbuffer handle, and process the RX data.

**Return**

- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_OK Success

**Parameters**

- `channel`: RMT channel (0 - 7)
- `buf_handle`: Pointer to buffer handle to accept RX ringbuffer handle.

`esp_err_t rmt_translator_init(rmt_channel_t channel, sample_to_rmt_t fn)`

Init rmt translator and register user callback. The callback will convert the raw data that needs to be sent to rmt format. If a channel is initialized more than once, the user callback will be replaced by the later.

**Return**

- ESP\_FAIL Init fail.
- ESP\_OK Init success.

**Parameters**

- `channel`: RMT channel (0 - 7).
- `fn`: Point to the data conversion function.

`esp_err_t rmt_write_sample(rmt_channel_t channel, const uint8_t *src, size_t src_size, bool wait_tx_done)`

Translate uint8\_t type of data into rmt format and send it out. Requires `rmt_translator_init` to init the translator first.

**Return**

- ESP\_FAIL Send fail
- ESP\_OK Send success

**Parameters**

- `channel`: RMT channel (0 - 7).
- `src`: Pointer to the raw data.
- `src_size`: The size of the raw data.
- `wait_tx_done`: Set true to wait all data send done.

*rmt\_tx\_end\_callback\_t* **rmt\_register\_tx\_end\_callback** (*rmt\_tx\_end\_fn\_t* function, void \*arg)

Registers a callback that will be called when transmission ends.

Called by `rmt_driver_isr_default` in interrupt context.

**Note** Requires `rmt_driver_install` to install the default ISR handler.

**Return** the previous callback settings (members will be set to NULL if there was none)

**Parameters**

- `function`: Function to be called from the default interrupt handler or NULL.
- `arg`: Argument which will be provided to the callback when it is called.

## Structures

**struct** `rmt_channel_status_result_t`

Data struct of RMT channel status.

### Public Members

*rmt\_channel\_status\_t* `status[RMT_CHANNEL_MAX]`

Store the current status of each channel

**struct** `rmt_tx_config_t`

Data struct of RMT TX configure parameters.

### Public Members

bool `loop_en`

Enable sending RMT items in a loop

uint32\_t `carrier_freq_hz`

RMT carrier frequency

uint8\_t `carrier_duty_percent`

RMT carrier duty (%)

*rmt\_carrier\_level\_t* `carrier_level`

Level of the RMT output, when the carrier is applied

bool `carrier_en`

RMT carrier enable

*rmt\_idle\_level\_t* `idle_level`

RMT idle level

bool `idle_output_en`

RMT idle level output enable

**struct** `rmt_rx_config_t`

Data struct of RMT RX configure parameters.

### Public Members

bool `filter_en`

RMT receiver filter enable

uint8\_t `filter_ticks_thresh`

RMT filter tick number

uint16\_t `idle_threshold`

RMT RX idle threshold

**struct** `rmt_config_t`

Data struct of RMT configure parameters.

## Public Members

*rmt\_mode\_t* **rmt\_mode**  
RMT mode: transmitter or receiver

*rmt\_channel\_t* **channel**  
RMT channel

uint8\_t **clk\_div**  
RMT channel counter divider

*gpio\_num\_t* **gpio\_num**  
RMT GPIO number

uint8\_t **mem\_block\_num**  
RMT memory block number

*rmt\_tx\_config\_t* **tx\_config**  
RMT TX parameter

*rmt\_rx\_config\_t* **rx\_config**  
RMT RX parameter

**struct rmt\_tx\_end\_callback\_t**  
Structure encapsulating a RMT TX end callback.

## Public Members

*rmt\_tx\_end\_fn\_t* **function**  
Function which is called on RMT TX end

void \***arg**  
Optional argument passed to function

## Macros

**RMT\_MEM\_BLOCK\_BYTE\_NUM**

**RMT\_MEM\_ITEM\_NUM**

## Type Definitions

**typedef intr\_handle\_t rmt\_isr\_handle\_t**

**typedef void (\*rmt\_tx\_end\_fn\_t) (rmt\_channel\_t channel, void \*arg)**

**typedef void (\*sample\_to\_rmt\_t) (const void \*src, rmt\_item32\_t \*dest, size\_t src\_size, size\_t wanted\_num, size\_t \*translated\_size, size\_t \*item\_num)**

User callback function to convert uint8\_t type data to rmt format(rmt\_item32\_t).

This function may be called from an ISR, so, the code should be short and efficient.

**Note** In fact, item\_num should be a multiple of translated\_size, e.g. : When we convert each byte of uint8\_t type data to rmt format data, the relation between item\_num and translated\_size should be  $item\_num = translated\_size * 8$ .

## Parameters

- **src**: Pointer to the buffer storing the raw data that needs to be converted to rmt format.
- **[out] dest**: Pointer to the buffer storing the rmt format data.
- **src\_size**: The raw data size.
- **wanted\_num**: The number of rmt format data that wanted to get.
- **[out] translated\_size**: The size of the raw data that has been converted to rmt format, it should return 0 if no data is converted in user callback.
- **[out] item\_num**: The number of the rmt format data that actually converted to, it can be less than wanted\_num if there is not enough raw data, but cannot exceed wanted\_num. it should return 0 if no data was converted.

**Enumerations****enum rmt\_channel\_t***Values:***RMT\_CHANNEL\_0** = 0  
RMT Channel 0**RMT\_CHANNEL\_1**  
RMT Channel 1**RMT\_CHANNEL\_2**  
RMT Channel 2**RMT\_CHANNEL\_3**  
RMT Channel 3**RMT\_CHANNEL\_4**  
RMT Channel 4**RMT\_CHANNEL\_5**  
RMT Channel 5**RMT\_CHANNEL\_6**  
RMT Channel 6**RMT\_CHANNEL\_7**  
RMT Channel 7**RMT\_CHANNEL\_MAX****enum rmt\_mem\_owner\_t***Values:***RMT\_MEM\_OWNER\_TX** = 0  
RMT RX mode, RMT transmitter owns the memory block**RMT\_MEM\_OWNER\_RX** = 1  
RMT RX mode, RMT receiver owns the memory block**RMT\_MEM\_OWNER\_MAX****enum rmt\_source\_clk\_t***Values:***RMT\_BASECLK\_REF** = 0  
RMT source clock system reference tick, 1MHz by default (not supported in this version)**RMT\_BASECLK\_APB**  
RMT source clock is APB CLK, 80Mhz by default**RMT\_BASECLK\_MAX****enum rmt\_data\_mode\_t***Values:***RMT\_DATA\_MODE\_FIFO** = 0**RMT\_DATA\_MODE\_MEM** = 1**RMT\_DATA\_MODE\_MAX****enum rmt\_mode\_t***Values:***RMT\_MODE\_TX** = 0  
RMT TX mode**RMT\_MODE\_RX**  
RMT RX mode**RMT\_MODE\_MAX**



```
enum rmt_idle_level_t
```

Values:

```
RMT_IDLE_LEVEL_LOW = 0
    RMT TX idle level: low Level

RMT_IDLE_LEVEL_HIGH
    RMT TX idle level: high Level

RMT_IDLE_LEVEL_MAX
```

```
enum rmt_carrier_level_t
```

Values:

```
RMT_CARRIER_LEVEL_LOW = 0
    RMT carrier wave is modulated for low Level output

RMT_CARRIER_LEVEL_HIGH
    RMT carrier wave is modulated for high Level output

RMT_CARRIER_LEVEL_MAX
```

```
enum rmt_channel_status_t
```

Values:

```
RMT_CHANNEL_UNINIT = 0
    RMT channel uninitialized

RMT_CHANNEL_IDLE = 1
    RMT channel status idle

RMT_CHANNEL_BUSY = 2
    RMT channel status busy
```

### 2.3.11 SDMMC Host Driver

#### Overview

ESP32's SDMMC host peripheral has two slots:

- 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.

Pin mappings of these slots are given in the table below.

Signal	Slot 0	Slot 1
CMD	GPIO11	GPIO15
CLK	GPIO6	GPIO14
D0	GPIO7	GPIO2
D1	GPIO8	GPIO4
D2	GPIO9	GPIO12
D3	GPIO10	GPIO13
D4	GPIO16	
D5	GPIO17	
D6	GPIO5	
D7	GPIO18	
CD	any input via GPIO matrix	
WP	any input via GPIO matrix	

The Card Detect and Write Protect signals can be routed to arbitrary pins using the GPIO matrix. To reserve the pins, set the `gpio_cd` and `gpio_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.

**警告:** 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), 4-line/1-line (with SD cards), and 8-line (with 3.3 V eMMC)
- High Speed (40 MHz), 4-line/1-line (with SD cards), and 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;
```

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;
```

## 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 Pullup Requirements](#) for pullup support and compatibilities of modules and development kits.

## API Reference

### Header File

- [driver/include/driver/sdmmc\\_host.h](#)

### Functions

*esp\_err\_t* **sdmmc\_host\_init** ()

Initialize SDMMC host peripheral.

**Note** This function is not thread safe

#### Return

- ESP\_OK on success
- ESP\_ERR\_INVALID\_STATE if sdmmc\_host\_init was already called
- ESP\_ERR\_NO\_MEM if memory can not 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.

**Note** This function is not thread safe

#### Return

- ESP\_OK on success
- ESP\_ERR\_INVALID\_STATE if host has not been initialized using sdmmc\_host\_init

#### Parameters

- *slot*: slot number (SDMMC\_HOST\_SLOT\_0 or SDMMC\_HOST\_SLOT\_1)
- *slot\_config*: additional configuration for the slot

*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)

**Note** This function is not thread safe

#### Return

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG if slot number or width is not valid

#### Parameters

- *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)

size\_t **sdmmc\_host\_get\_slot\_width** (int *slot*)

Get bus width configured in sdmmc\_host\_init\_slot to be used for data transfer.

**Return** configured bus width of the specified slot.

#### Parameters

- *slot*: slot number (SDMMC\_HOST\_SLOT\_0 or SDMMC\_HOST\_SLOT\_1)

*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.

**Note** This function is not thread safe

#### Return

- ESP\_OK on success
- other error codes may be returned in the future

**Parameters**

- `slot`: slot number (SDMMC\_HOST\_SLOT\_0 or SDMMC\_HOST\_SLOT\_1)
- `freq_khz`: card clock frequency, in kHz

*esp\_err\_t* **sdmmc\_host\_set\_bus\_ddr\_mode** (int *slot*, bool *ddr\_enabled*)

Enable or disable DDR mode of SD interface.

**Return**

- ESP\_OK on success
- ESP\_ERR\_NOT\_SUPPORTED if DDR mode is not supported on this slot

**Parameters**

- `slot`: slot number (SDMMC\_HOST\_SLOT\_0 or SDMMC\_HOST\_SLOT\_1)
- `ddr_enabled`: enable or disable DDR mode

*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.

**Note** 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.

**Attention** Data buffer passed in `cmdinfo->data` must be in DMA capable memory

**Return**

- 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

**Parameters**

- `slot`: slot number (SDMMC\_HOST\_SLOT\_0 or SDMMC\_HOST\_SLOT\_1)
- `cmdinfo`: pointer to structure describing command and data to transfer

*esp\_err\_t* **sdmmc\_host\_io\_int\_enable** (int *slot*)

Enable IO interrupts.

This function configures the host to accept SDIO interrupts.

**Return** returns ESP\_OK, other errors possible in the future

**Parameters**

- `slot`: slot number (SDMMC\_HOST\_SLOT\_0 or SDMMC\_HOST\_SLOT\_1)

*esp\_err\_t* **sdmmc\_host\_io\_int\_wait** (int *slot*, TickType\_t *timeout\_ticks*)

Block until an SDIO interrupt is received, or timeout occurs.

**Return**

- ESP\_OK on success (interrupt received)
- ESP\_ERR\_TIMEOUT if the interrupt did not occur within `timeout_ticks`

**Parameters**

- `slot`: slot number (SDMMC\_HOST\_SLOT\_0 or SDMMC\_HOST\_SLOT\_1)
- `timeout_ticks`: number of RTOS ticks to wait for the interrupt

*esp\_err\_t* **sdmmc\_host\_deinit** ()

Disable SDMMC host and release allocated resources.

**Note** This function is not thread safe

**Return**

- ESP\_OK on success
- ESP\_ERR\_INVALID\_STATE if `sdmmc_host_init` function has not been called

*esp\_err\_t* **sdmmc\_host\_pullup\_en** (int *slot*, int *width*)

Enable the pull-ups of sd pins.

**Note** You should always place actual pullups on the lines instead of using this function. Internal pullup resistance are high and not sufficient, may cause instability in products. This is for debug or examples only.

**Return**

- ESP\_OK: if success
- ESP\_ERR\_INVALID\_ARG: if configured width larger than maximum the slot can support

**Parameters**

- `slot`: Slot to use, normally set it to 1.
- `width`: Bit width of your configuration, 1 or 4.

### 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* **gpio\_wp**

GPIO number of write protect signal.

*uint8\_t* **width**

Bus width used by the slot (might be less than the max width supported)

*uint32\_t* **flags**

Features used by this slot.

### Macros

**SDMMC\_HOST\_SLOT\_0**

SDMMC slot 0.

**SDMMC\_HOST\_SLOT\_1**

SDMMC slot 1.

**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

**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.

**SDMMC\_SLOT\_NO\_CD**

indicates that card detect line is not used

**SDMMC\_SLOT\_NO\_WP**

indicates that write protect line is not used

**SDMMC\_SLOT\_WIDTH\_DEFAULT**

use the default width for the slot (8 for slot 0, 4 for slot 1)

**SDMMC\_SLOT\_CONFIG\_DEFAULT ()**

Macro defining default configuration of SDMMC host slot

## 2.3.12 SD SPI Host Driver

## Overview

SPI controllers accessible via `spi_master` driver (HSPI, VSPI) can be used to work with SD cards. The driver which provides this capability is called “SD SPI Host”, due to its similarity with the *SDMMC Host* driver.

In SPI mode, SD driver has lower throughput than in 1-line SD mode. However SPI mode makes pin selection more flexible, as SPI peripheral can be connected to any ESP32 pins using GPIO Matrix. SD SPI driver uses software controlled CS signal. Currently SD SPI driver assumes that it can use the SPI controller exclusively, so applications which need to share SPI bus between SD cards and other peripherals need to make sure that SD card and other devices are not used at the same time from different tasks.

SD SPI driver is represented using an `sdmmc_host_t` structure initialized using `SDSPI_HOST_DEFAULT` macro. For slot initialization, `SDSPI_SLOT_CONFIG_DEFAULT` can be used to fill in default pin mapping, which is the same as the pin mapping in SD mode.

SD SPI driver APIs are very similar to *SDMMC host APIs*. As with the SDMMC host driver, only `sdspi_host_init()`, `sdspi_host_init_slot()`, and `sdspi_host_deinit()` functions are normally used by the applications. Other functions are called by the protocol level driver via function pointers in `sdmmc_host_t` structure.

See *SD/SDIO/MMC Driver* for the higher level driver which implements the protocol layer.

## API Reference

### Header File

- `driver/include/driver/sdspi_host.h`

### Functions

`esp_err_t sdspi_host_init()`

Initialize SD SPI driver.

**Note** This function is not thread safe

#### Return

- ESP\_OK on success
- other error codes may be returned in future versions

`esp_err_t sdspi_host_init_slot(int slot, const sdspi_slot_config_t *slot_config)`

Initialize SD SPI driver for the specific SPI controller.

**Note** This function is not thread safe

**Note** The SDIO over sdspi needs an extra interrupt line. Call `gpio_install_isr_service()` before this function.

#### Return

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG if `sdspi_init_slot` has invalid arguments
- ESP\_ERR\_NO\_MEM if memory can not be allocated
- other errors from the underlying `spi_master` and `gpio` drivers

#### Parameters

- `slot`: SPI controller to use (HSPI\_HOST or VSPI\_HOST)
- `slot_config`: pointer to slot configuration structure

`esp_err_t sdspi_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.

**Note** 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.

#### Return

- 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

**Parameters**

- `slot`: SPI controller (HSPI\_HOST or VSPI\_HOST)
- `cmdinfo`: pointer to structure describing command and data to transfer

*esp\_err\_t* **sdspi\_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.

**Note** This function is not thread safe

**Return**

- ESP\_OK on success
- other error codes may be returned in the future

**Parameters**

- `slot`: SPI controller (HSPI\_HOST or VSPI\_HOST)
- `freq_khz`: card clock frequency, in kHz

*esp\_err\_t* **sdspi\_host\_deinit** ()

Release resources allocated using `sdspi_host_init`.

**Note** This function is not thread safe

**Return**

- ESP\_OK on success
- ESP\_ERR\_INVALID\_STATE if `sdspi_host_init` function has not been called

*esp\_err\_t* **sdspi\_host\_io\_int\_enable** (int *slot*)

Enable SDIO interrupt.

**Return**

- ESP\_OK on success

**Parameters**

- `slot`: SPI controller to use (HSPI\_HOST or VSPI\_HOST)

*esp\_err\_t* **sdspi\_host\_io\_int\_wait** (int *slot*, TickType\_t *timeout\_ticks*)

Wait for SDIO interrupt until timeout.

**Return**

- ESP\_OK on success

**Parameters**

- `slot`: SPI controller to use (HSPI\_HOST or VSPI\_HOST)
- `timeout_ticks`: Ticks to wait before timeout.

**Structures**

**struct** **sdspi\_slot\_config\_t**

Extra configuration for SPI host

**Public Members**

*gpio\_num\_t* **gpio\_miso**

GPIO number of MISO signal.

*gpio\_num\_t* **gpio\_mosi**

GPIO number of MOSI signal.

*gpio\_num\_t* **gpio\_sck**

GPIO number of SCK signal.

*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.

int `dma_channel`  
DMA channel to be used by SPI driver (1 or 2)

### Macros

`SDSPI_HOST_DEFAULT ()`  
Default `sdmmc_host_t` structure initializer for SD over SPI driver.  
Uses SPI mode and max frequency set to 20MHz  
'slot' can be set to one of HSPI\_HOST, VSPI\_HOST.

`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_SLOT_CONFIG_DEFAULT ()`  
Macro defining default configuration of SPI host

## 2.3.13 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.

Pin Name	Slot1	Slot2
	GPIO Number	
CLK	6	14
CMD	11	15
DAT0	7	2
DAT1	8	4
DAT2	9	12
DAT3	10	13

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. Then the host initialize the slave into SD mode by first sending CMD0 with DAT3 pin high, while initialize the slave into SPI mode by sending CMD0 with CS pin (the same pin as DAT3) low.

---

**注解:** CMD and DATA lines D0-D3 of the card should be pulled up by 50KOhm resistor even in 1-bit mode or SPI mode. Most official devkits don't meet the pullup requirements by default, and there are conflicts on strapping pins as well. Please refer to [SD Pullup Requirements](#) to see how to setup your system correctly.

---



**SD Pullup Requirements** CMD and DATA lines D0-D3 of the slave should be pulled up by 50KOhm resistor even in 1-bit mode or SPI mode. The pullups of the slave cards should be connected even if they're not connected to the host.

The MTDI strapping pin is incompatible with DAT2 line pull-up by default when the code flash is 3.3V. See [MTDI strapping pin](#) below.

**Pullup inside Official Modules** For Espressif official modules, different weak pullups / pulldowns are connected to CMD, and DATA pins as below. To use these modules, these pins are required to be pulled up by 50KOhm resistors, since internal weak pullups are insufficient.

GPIO	15	2	4	12	13
Name	CMD	DAT0	DAT1	DAT2	DAT3
At startup	WPU	WPD	WPD	PU for 1.8v flash; WPD for 3.3v flash	WPU
Strapping requirement		Low to download to flash		High for 1.8v flash; Low for 3.3v flash	

- WPU: Weak pullup
- WPD: Weak pulldown
- PU: Pullup inside the module

For Wrover modules, they use 1.8v flash, and have pullup on GPIO12 inside. For Wroom-32 Series, PICO-D4 modules, they use 3.3v flash, and is weakly pulled down internally. See [MTDI strapping pin](#) below.

**Pullup on Official Devkit (WroverKit)** For official Wrover Kit (till version 3), some of the pullups are provided on the board as the table below. For other devkits that don't have pullups, please connect them yourselves.

GPIO	15	2	4	12	13
Name	CMD	DAT0	DAT1	DAT2	DAT3
Pullup on the Kit	PU	PU	PU		PU & PD

- PU: Pullup
- PD: Pulldown

The DAT3 pullup conflicts with JTAG pulldown in WroverKit v3 and earlier, please either:

1. pull it up by resistor less than 5KOhm (2kOhm suggested) in 4-bit mode.
2. pull it up or drive it high by host or VDD3.3V in 1-bit mode.

**MTDI strapping pin** MTDI (GPIO12) is used as a bootstrapping pin to select output voltage of an internal regulator which powers the flash chip (VDD\_SDIO). This pin has an internal pulldown so if left unconnected it will read low at reset (selecting default 3.3V operation). When adding a pullup to this pin for SD card operation, consider the following:

- For boards which don't use the internal regulator (VDD\_SDIO) to power the flash, GPIO12 can be pulled high.
- For boards which use 1.8V flash chip, GPIO12 needs to be pulled high at reset. This is fully compatible with SD card operation.
- On boards which use the internal regulator and a 3.3V flash chip, GPIO12 must be low at reset. This is incompatible with SD card operation. Please check the table below to see whether your modules/kits use 3.3v flash.

Module	Flash voltage	DAT2 connections
PICO-D4	3.3V	Internal PD, change EFUSE and pullup or disable DAT2 line*
Wroom-32 Series		
Wrover	1.8V	Internal PU, pullup suggested

Official devkits of different types and version mount different types of modules, please refer to the table below to see whether your devkit can support SDIO slave without steps above.

Devkit	Module	Flash voltage
PICO Kit	PICO-D4	3.3V (see steps below)
DevKitC	Wroom-32 Series	
WroverKit v2 and earlier		
WroverKit v3	Wrover	1.8V

If your board requires internal regulator with 3.3v output, to make it compatible with SD pullup, you can either:

- **In the case using ESP32 host only**, external pullup can be omitted and an internal pullup can be enabled using a `gpio_pullup_en(GPIO_NUM_12)`; call. Most SD cards work fine when an internal pullup on GPIO12 line is enabled. Note that if ESP32 experiences a power-on reset while the SD card is sending data, high level on GPIO12 can be latched into the bootstrapping register, and ESP32 will enter a boot loop until external reset with correct GPIO12 level is applied.
- **In the case using ESP32 slave in 1-bit mode**, specify `SDIO_SLAVE_FLAG_DAT2_DISABLED` in the slave to avoid slave detecting on DAT2 line. Note the host will not know 4-bit mode is not supported any more by the standard CCCR register. You have to tell the host use 1-bit only.
- **For ESP32 host or slave**, another option is to burn the flash voltage selection efuses. This will permanently select 3.3V output voltage for the internal regulator, and GPIO12 will not be used as a bootstrapping pin. Then it is safe to connect a pullup resistor to GPIO12. This option is suggested for production use. NOTE this cannot be reverted once the EFUSE is burnt.

The following command can be used to program flash voltage selection efuses **to 3.3V**:

```
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. With all three burned to value 1, the internal VDD\_SDIO flash voltage regulator is permanently enabled at 3.3V. See the technical reference manual for more details.

`espefuse.py` has a `-do-not-confirm` option if running from an automated flashing script.

**GPIO2 Strapping pin** GPIO2 pin is used as a bootstrapping pin, and should be low to enter UART download mode. You may find it unable to enter the UART download mode if you correctly connect the pullup of SD on GPIO2. For WroverKit v3, there are dedicated circuits to pulldown the GPIO2 when downloading. For other boards, one way to do this is to connect GPIO0 and GPIO2 using a jumper, and then the auto-reset circuit on most development boards will pull GPIO2 low along with GPIO0, when entering download mode.

- Some boards have pulldown and/or LED on GPIO2. LED is usually ok, but pulldown will interfere with D0 signals and must be removed. Check the schematic of your development board for anything connected to GPIO2.

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.

---

**注解:** Register names in ESP Rechnical Reference Manual 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.

---

**注解:** 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](#).

However, there' s an ESP32-specific upper-level communication protocol upon the CMD52/CMD53 to Func 1. Please refer to [ESP SDIO slave protocol](#), or example [peripherals/sdio](#) when programming your host.

### Communication with ESP SDIO Slave

**ESP SDIO slave initialization** The host should initialize the ESP32 SDIO slave according to the standard SDIO initialization process (Sector 3.1.2 of [SDIO Simplified Specification](#)). In this specification and below, the SDIO slave is also called an (SD)IO card. All the initialization CMD52 and CMD53 are sent to Func 0 (CIA region). Here is an brief example on how to do this:

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 0xFFFFF00.  
 Keep sending CMD5 with arg=0x00FFFF00 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 protocol is based on Function 1 access by CMD52 and CMD53, offering 3 services: (1) sending and receiving FIFO, (2) 52 8-bit R/W register shared by host and slave, (3) 8 general purpose interrupt sources from host to slave and 8 in the opposite direction.

The host should access the registers below as described to communicate with slave.

### Slave register table

#### 32-bit

- 0x044 (TOKEN\_RDATA): in which bit 27-16 holds the receiving buffer number.
- 0x058 (INT\_ST): holds the interrupt source bits from slave to host.
- 0x060 (PKT\_LEN): holds the accumulated length (by byte) to be sent from slave to host.
- 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/write to the slave in a single transfer:

$$\text{requested length} = 0x1F800 - \text{address}$$

The slave will respond with the length according to the length field in CMD53, with the data longer than *requested length* filled with 0 (sending) or discard (receiving).

---

**注解:** This includes both the block and the byte mode of CMD53.

The function number should be set to 1, OP Code should be set to 1 (for CMD53).

It is allowed to use CMD53 mode combination of block+byte to get higher efficiency when accessing the FIFO by arbitrary length. E.g. The block size is set to 512 by default, you can write/get 1031 bytes of data to/from the FIFO by:

1. Send CMD53 in block mode, block count=2 (1024 bytes) to address 0x1F3F9=0x1F800-1031.
  2. Then send CMD53 in byte mode, byte count=8 (or 7 if your controller supports that) to address 0x1F7F9=0x1F800-7.
- 

**Interrupts** For the host interrupts, the slave raise the interrupt by pulling DAT1 line down at a proper time (level sensitive). The host detect this and read the INT\_ST register to see the source. Then the host can clear it by writing the INT\_CLR register and do something with the interrupt. The host can also mask unneeded sources by clearing the bits in INT\_ENA register corresponding to the sources. If all the sources are cleared (or masked), the DAT1 line goes inactive.

sdio\_slave\_hostint\_t (*SDIO Card Slave Driver*) shows the bit definition corresponding to host interrupt sources.

For the slave interrupts, the host send transfers to write the SLAVE\_INT register. Once a bit is written from 0 to 1, the slave hardware and driver will detect it and inform the app.

**Receiving FIFO** To write the receiving FIFO in the slave, host should work in the following steps:

1. Read the TOKEN1 field (bits 27-16) of TOKEN\_RDATA (0x044) register. The buffer number remaining is TOKEN1 minus the number of buffers used by host.
2. Make sure the buffer number is sufficient ( $buffer\_size * buffer\_num$  is greater than data to write,  $buffer\_size$  is pre-defined between the host and the slave before the communication starts). Or go back to step 1 until the buffer is enough.
3. Write to the FIFO address with CMD53. Note that the *requested length* should not be larger than calculated in step 2, and the FIFO address is related to *requested length*.
4. Calculate used buffers, note that non-full buffer at the tail should be seen as one that is used.

**Sending FIFO** To read the sending FIFO in the slave, host should work in the following steps:

1. Wait for the interrupt line to be active (optional, low by default).
2. Read (poll) the interrupt bits in INT\_ST register to see whether new packets exists.
3. If new packets are ready, reads the PKT\_LEN reg. The data length to read from slave is PKT\_LEN minuses the length that has been read from the host. If the PKT\_LEN is not larger than used, wait and poll until the slave is ready and update the PKT\_LEN.
4. Read from the FIFO with CMD53. Note that the *requested length* should not be larger than calculated in step3, and the FIFO address is related to *requested length*.
5. Recored read length.

**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.

---

**注解:** 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.
2. Load buffers onto the driver by passing the buffer handle to `sdio_slave_recv_load_buf`.
3. Call `sdio_slave_recv` to get the received data. If non-blocking call is needed, set `wait=0`.
4. Pass the handle of processed buffer back to the driver by `sdio_recv_load_buf` again.

**注解:** To avoid overhead from copying data, the driver itself doesn't have any buffer inside, the application is responsible to offer new buffers in time. The DMA will automatically store received data to the buffer.

**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.

**注解:** 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), 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:

```
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);
```

(下页继续)

```
//... 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", "(%d) successfully send %d bytes of %p", arg->id, arg->size,
↳arg->buffer);
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:

```
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

- [driver/include/driver/sdio\\_slave.h](#)

### Functions

*esp\_err\_t* **sdio\_slave\_initialize** (*sdio\_slave\_config\_t* \**config*)

Initialize the sdio slave driver

#### Return

- `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

#### Parameters

- `config`: Configuration of the sdio slave driver.

void **sdio\_slave\_deinit** ()

De-initialize the sdio slave driver to release the resources.

*esp\_err\_t* **sdio\_slave\_start** ()

Start hardware for sending and receiving, as well as set the `IOREADY1` to 1.

**Note** 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 `sdio_slave_reset`.

#### Return

- `ESP_ERR_INVALID_STATE` if already started.
- `ESP_OK` otherwise.

void **sdio\_slave\_stop** ()

Stop hardware from sending and receiving, also set IOREADY1 to 0.

**Note** 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.

*esp\_err\_t* **sdio\_slave\_reset** ()

Clear the data still in the driver, as well as reset the PKT\_LEN and TOKEN1 counting.

**Return** always return ESP\_OK.

*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.

**Note** 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.

**Return** The buffer handle if success, otherwise NULL.

**Parameters**

- `start`: The start address of the buffer.

*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.

**Return** ESP\_OK if success, ESP\_ERR\_INVALID\_ARG if the handle is NULL or the buffer is being used.

**Parameters**

- `handle`: Handle to the buffer to release.

*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.

**Return**

- 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

**Parameters**

- `handle`: Handle to the buffer ready to receive data.

*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.

**Note** 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`.

**Return**

- ESP\_ERR\_INVALID\_ARG if `handle_ret` is NULL
- ESP\_ERR\_TIMEOUT if timeout before receiving new data
- ESP\_OK if success

**Parameters**

- `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] `out_addr`: Output of the start address, set to NULL if not needed.
- [out] `out_len`: Actual length of the data in the buffer, set to NULL if not needed.
- `wait`: Time to wait before data received.

uint8\_t \***sdio\_slave\_recv\_get\_buf** (*sdio\_slave\_buf\_handle\_t* handle, size\_t \*len\_o)

Retrieve the buffer corresponding to a handle.

**Return** buffer address if success, otherwise NULL.

**Parameters**

- `handle`: Handle to get the buffer.
- `len_o`: Output of buffer length



*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.

**Return**

- 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.

**Parameters**

- `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.

*esp\_err\_t* **sdio\_slave\_send\_get\_finished** (void \*\*out\_arg, TickType\_t wait)

Return the ownership of a finished transaction.

**Return** ESP\_ERR\_TIMEOUT if no transaction finished, or ESP\_OK if succeed.

**Parameters**

- `out_arg`: Argument of the finished transaction. Set to NULL if unused.
- `wait`: Time to wait if there's no finished sending transaction.

*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.

**Return**

- 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.

**Parameters**

- `addr`: Start address of the buffer to send
- `len`: Length of buffer to send.

uint8\_t **sdio\_slave\_read\_reg** (int pos)

Read the spi slave register shared with host.

**Note** register 28 to 31 are reserved for interrupt vector.

**Return** value of the register.

**Parameters**

- `pos`: register address, 0-27 or 32-63.

*esp\_err\_t* **sdio\_slave\_write\_reg** (int pos, uint8\_t reg)

Write the spi slave register shared with host.

**Note** register 29 and 31 are used for interrupt vector.

**Return** ESP\_ERR\_INVALID\_ARG if address wrong, otherwise ESP\_OK.

**Parameters**

- `pos`: register address, 0-11, 14-15, 18-19, 24-27 and 32-63, other address are reserved.
- `reg`: the value to write.

*sdio\_slave\_hostint\_t* **sdio\_slave\_get\_host\_intena** ()

Get the interrupt enable for host.

**Return** the interrupt mask.

void **sdio\_slave\_set\_host\_intena** (*sdio\_slave\_hostint\_t* ena)

Set the interrupt enable for host.

**Parameters**

- `ena`: Enable mask for host interrupt.

*esp\_err\_t* **sdio\_slave\_send\_host\_int** (uint8\_t pos)

Interrupt the host by general purpose interrupt.

**Return**

- ESP\_ERR\_INVALID\_ARG if interrupt num error
- ESP\_OK otherwise

**Parameters**

- pos: Interrupt num, 0-7.

void **sdio\_slave\_clear\_host\_int** (uint8\_t mask)

Clear general purpose interrupt to host.

**Parameters**

- 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.

**Note** this clears the interrupt at the same time.

**Return** ESP\_OK if success, ESP\_ERR\_TIMEOUT if timeout.

**Parameters**

- pos: Interrupt source number to wait for. is set.
- wait: Time to wait before interrupt triggered.

**Structures**

**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.

**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.

**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.

**Enumerations**

```
enum sdio_slave_hostint_t
```

Mask of interrupts sending to the host.

*Values:*

```
SDIO_SLAVE_HOSTINT_SEND_NEW_PACKET = HOST_SLC0_RX_NEW_PACKET_INT_ENA  
New packet available.
```

```
SDIO_SLAVE_HOSTINT_RECV_OVF = HOST_SLC0_TX_OVF_INT_ENA  
Slave receive buffer overflow.
```

```
SDIO_SLAVE_HOSTINT_SEND_UDF = HOST_SLC0_RX_UDF_INT_ENA  
Slave sending buffer underflow (this case only happen when the host do not request for packet according to the packet len).
```

```
SDIO_SLAVE_HOSTINT_BIT7 = HOST_SLC0_TOHOST_BIT7_INT_ENA  
General purpose interrupt bits that can be used by the user.
```

```
SDIO_SLAVE_HOSTINT_BIT6 = HOST_SLC0_TOHOST_BIT6_INT_ENA
```

```
SDIO_SLAVE_HOSTINT_BIT5 = HOST_SLC0_TOHOST_BIT5_INT_ENA
```

```
SDIO_SLAVE_HOSTINT_BIT4 = HOST_SLC0_TOHOST_BIT4_INT_ENA
```

```
SDIO_SLAVE_HOSTINT_BIT3 = HOST_SLC0_TOHOST_BIT3_INT_ENA
```

```
SDIO_SLAVE_HOSTINT_BIT2 = HOST_SLC0_TOHOST_BIT2_INT_ENA
```

```
SDIO_SLAVE_HOSTINT_BIT1 = HOST_SLC0_TOHOST_BIT1_INT_ENA
```

```
SDIO_SLAVE_HOSTINT_BIT0 = HOST_SLC0_TOHOST_BIT0_INT_ENA
```

```
enum sdio_slave_timing_t
```

Timing of SDIO slave.

*Values:*

```
SDIO_SLAVE_TIMING_PSEND_PSAMPLE = 0  
Send at posedge, and sample at posedge. Default value for HS mode. Normally there's no problem using this to work in DS mode.
```

```
SDIO_SLAVE_TIMING_NSEND_PSAMPLE  
Send at negedge, and sample at posedge. Default value for DS mode and below.
```

```
SDIO_SLAVE_TIMING_PSEND_NSAMPLE  
Send at posedge, and sample at negedge.
```

**SDIO\_SLAVE\_TIMING\_NSEND\_NSAMPLE**

Send at negedge, and sample at negedge.

**enum sdio\_slave\_sending\_mode\_t**

Configuration of SDIO slave mode.

Values:

**SDIO\_SLAVE\_SEND\_STREAM = 0**

Stream mode, all packets to send will be combined as one if possible.

**SDIO\_SLAVE\_SEND\_PACKET = 1**

Packet mode, one packets will be sent one after another (only increase packet\_len if last packet sent).

## 2.3.14 Sigma-delta Modulation

### Introduction

ESP32 has a second-order sigma-delta modulation module. This driver configures the channels of the sigma-delta module.

### Functionality Overview

There are eight independent sigma-delta modulation channels identified with *sigmadelta\_channel\_t*. Each channel is capable to output the binary, hardware generated signal with the sigma-delta modulation.

Selected channel should be set up by providing configuration parameters in *sigmadelta\_config\_t* and then applying this configuration with *sigmadelta\_config()*.

Another option is to call individual functions, that will configure all required parameters one by one:

- **Prescaler** of the sigma-delta generator - *sigmadelta\_set\_prescale()*
- **Duty** of the output signal - *sigmadelta\_set\_duty()*
- **GPIO pin** to output modulated signal - *sigmadelta\_set\_pin()*

The range of the 'duty' input parameter of *sigmadelta\_set\_duty()* is from -128 to 127 (eight bit signed integer). If zero value is set, then the output signal's duty will be about 50%, see description of *sigmadelta\_set\_duty()*.

### Application Example

Sigma-delta Modulation example: [peripherals/sigmadelta](#).

### API Reference

#### Header File

- [driver/include/driver/sigmadelta.h](#)

#### Functions

*esp\_err\_t sigmadelta\_config(const sigmadelta\_config\_t \*config)*

Configure Sigma-delta channel.

#### Return

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

#### Parameters

- config: Pointer of Sigma-delta channel configuration struct

*esp\_err\_t* **sigmadelta\_set\_duty** (*sigmadelta\_channel\_t* channel, int8\_t duty)

Set Sigma-delta channel duty.

This function is used to set Sigma-delta channel duty, If you add a capacitor between the output pin and ground, the average output voltage will be  $V_{dc} = V_{DDIO} / 256 * duty + V_{DDIO}/2$ , where VDDIO is the power supply voltage.

#### Return

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

#### Parameters

- channel: Sigma-delta channel number
- duty: Sigma-delta duty of one channel, the value ranges from -128 to 127, recommended range is -90 ~ 90. The waveform is more like a random one in this range.

*esp\_err\_t* **sigmadelta\_set\_prescale** (*sigmadelta\_channel\_t* channel, uint8\_t prescale)

Set Sigma-delta channel's clock pre-scale value. The source clock is APP\_CLK, 80MHz. The clock frequency of the sigma-delta channel is APP\_CLK / pre\_scale.

#### Return

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

#### Parameters

- channel: Sigma-delta channel number
- prescale: The divider of source clock, ranges from 0 to 255

*esp\_err\_t* **sigmadelta\_set\_pin** (*sigmadelta\_channel\_t* channel, *gpio\_num\_t* gpio\_num)

Set Sigma-delta signal output pin.

#### Return

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

#### Parameters

- channel: Sigma-delta channel number
- gpio\_num: GPIO number of output pin.

## Structures

**struct sigmadelta\_config\_t**

Sigma-delta configure struct.

## Public Members

*sigmadelta\_channel\_t* **channel**

Sigma-delta channel number

int8\_t **sigmadelta\_duty**

Sigma-delta duty, duty ranges from -128 to 127.

uint8\_t **sigmadelta\_prescale**

Sigma-delta prescale, prescale ranges from 0 to 255.

uint8\_t **sigmadelta\_gpio**

Sigma-delta output io number, refer to gpio.h for more details.

## Enumerations

**enum sigmadelta\_channel\_t**

Sigma-delta channel list.

Values:

**SIGMADELTA\_CHANNEL\_0** = 0  
Sigma-delta channel 0

**SIGMADELTA\_CHANNEL\_1** = 1  
Sigma-delta channel 1

**SIGMADELTA\_CHANNEL\_2** = 2  
Sigma-delta channel 2

**SIGMADELTA\_CHANNEL\_3** = 3  
Sigma-delta channel 3

**SIGMADELTA\_CHANNEL\_4** = 4  
Sigma-delta channel 4

**SIGMADELTA\_CHANNEL\_5** = 5  
Sigma-delta channel 5

**SIGMADELTA\_CHANNEL\_6** = 6  
Sigma-delta channel 6

**SIGMADELTA\_CHANNEL\_7** = 7  
Sigma-delta channel 7

**SIGMADELTA\_CHANNEL\_MAX**

### 2.3.15 SPI Master driver

#### Overview

The ESP32 has four SPI peripheral devices, called SPI0, SPI1, HSPI and VSPI. SPI0 is entirely dedicated to the flash cache the ESP32 uses to map the SPI flash device it is connected to into memory. SPI1 is connected to the same hardware lines as SPI0 and is used to write to the flash chip. HSPI and VSPI are free to use. SPI1, HSPI and VSPI all have three chip select lines, allowing them to drive up to three SPI devices each as a master.

**The spi\_master driver** The spi\_master driver allows easy communicating with SPI slave devices, even in a multi-threaded environment. It fully transparently handles DMA transfers to read and write data and automatically takes care of multiplexing between different SPI slaves on the same master.

---

#### 注解: Notes about thread safety

The SPI driver API is thread safe when multiple SPI devices on the same bus are accessed from different tasks. However, the driver is not thread safe if the same SPI device is accessed from multiple tasks.

In this case, it is recommended to either refactor your application so only a single task accesses each SPI device, or to add mutex locking around access of the shared device.

---

**Terminology** The spi\_master driver uses the following terms:

- Host: The SPI peripheral inside the ESP32 initiating the SPI transmissions. One of SPI, HSPI or VSPI. (For now, only HSPI or VSPI are actually supported in the driver; it will support all 3 peripherals somewhere in the future.)
- Bus: The SPI bus, common to all SPI devices connected to one host. In general the bus consists of the miso, mosi, sclk and optionally quadwp and quadhd signals. The SPI slaves are connected to these signals in parallel.
  - miso - Also known as q, this is the input of the serial stream into the ESP32
  - mosi - Also known as d, this is the output of the serial stream from the ESP32
  - sclk - Clock signal. Each data bit is clocked out or in on the positive or negative edge of this signal
  - quadwp - Write Protect signal. Only used for 4-bit (qio/qout) transactions.
  - quadhd - Hold signal. Only used for 4-bit (qio/qout) transactions.

- Device: A SPI slave. Each SPI slave has its own chip select (CS) line, which is made active when a transmission to/from the SPI slave occurs.
- Transaction: One instance of CS going active, data transfer from and/or to a device happening, and CS going inactive again. Transactions are atomic, as in they will never be interrupted by another transaction.

**SPI transactions** A transaction on the SPI bus consists of five phases, any of which may be skipped:

- The command phase. In this phase, a command (0-16 bit) is clocked out.
- The address phase. In this phase, an address (0-64 bit) is clocked out.
- The write phase. The master sends data to the slave.
- The dummy phase. The phase is configurable, used to meet the timing requirements.
- The read phase. The slave sends data to the master.

In full duplex mode, the read and write phases are combined, and the SPI host reads and writes data simultaneously. The total transaction length is decided by `command_bits + address_bits + trans_conf.length`, while the `trans_conf.rx_length` only determines length of data received into the buffer.

While in half duplex mode, the host has independent write and read phases. The length of write phase and read phase are decided by `trans_conf.length` and `trans_conf.rx_length` respectively.

The command and address phase are optional in that not every SPI device will need to be sent a command and/or address. This is reflected in the device configuration: when the `command_bits` or `address_bits` fields are set to zero, no command or address phase is done.

Something similar is true for the read and write phase: not every transaction needs both data to be written as well as data to be read. When `rx_buffer` is NULL (and `SPI_TRANS_USE_RXDATA`) is not set) the read phase is skipped. When `tx_buffer` is NULL (and `SPI_TRANS_USE_TXDATA`) is not set) the write phase is skipped.

The driver offers two different kinds of transactions: the interrupt transactions and the polling transactions. Each device can choose one kind of transaction to send. See [Notes to send mixed transactions to the same device](#) if your device does require both kinds of transactions.

**Interrupt transactions** The interrupt transactions use an interrupt-driven logic when the transactions are in-flight. The routine will get blocked, allowing the CPU to run other tasks, while it is waiting for a transaction to be finished.

Interrupt transactions can be queued into a device, the driver automatically sends them one-by-one in the ISR. A task can queue several transactions, and then do something else before the transactions are finished.

**Polling transactions** The polling transactions don't rely on the interrupt, the routine keeps polling the status bit of the SPI peripheral until the transaction is done.

All the tasks that do interrupt transactions may get blocked by the queue, at which point they need to wait for the ISR to run twice before the transaction is done. Polling transactions save the time spent on queue handling and context switching, resulting in a smaller transaction interval. The disadvantage is that the CPU is busy while these transactions are in flight.

The `spi_device_polling_end` routine spends at least 1  $\mu$ s overhead to unblock other tasks when the transaction is done. It is strongly recommended to wrap a series of polling transactions inside of `spi_device_acquire_bus` and `spi_device_release_bus` to avoid the overhead. (See [Bus acquiring](#))

**Command and address phases** During the command and address phases, `cmd` and `addr` fields in the `spi_transaction_t` struct are sent to the bus, while nothing is read at the same time. The default length of command and address phase are set in the `spi_device_interface_config_t` and by `spi_bus_add_device`. When the flags `SPI_TRANS_VARIABLE_CMD` and `SPI_TRANS_VARIABLE_ADDR` are not set in the `spi_transaction_t`, the driver automatically sets the length of these phases to the default value as set when the device is initialized respectively.

If the length of command and address phases needs to be variable, declare a `spi_transaction_ext_t` descriptor, set the flag `SPI_TRANS_VARIABLE_CMD` or/and `SPI_TRANS_VARIABLE_ADDR` in the flags of base

member and configure the rest part of `base` as usual. Then the length of each phases will be `command_bits` and `address_bits` set in the `spi_transaction_ext_t`.

**Write and read phases** Normally, data to be transferred to or from a device will be read from or written to a chunk of memory indicated by the `rx_buffer` and `tx_buffer` members of the transaction structure. When DMA is enabled for transfers, these buffers are highly recommended to meet the requirements as below:

1. allocated in DMA-capable memory using `pvPortMallocCaps(size, MALLOC_CAP_DMA)`;
2. 32-bit aligned (start from the boundary and have length of multiples of 4 bytes).

If these requirements are not satisfied, efficiency of the transaction will suffer due to the allocation and memcopy of temporary buffers.

---

**注解:** Half duplex transactions with both read and write phases are not supported when using DMA. See [Known Issues](#) for details and workarounds.

---

**Bus acquiring** Sometimes you may want to send `spi` transactions exclusively, continuously, to make it as fast as possible. You may use `spi_device_acquire_bus` and `spi_device_release_bus` to realize this. When the bus is acquired, transactions to other devices (no matter polling or interrupt) are pending until the bus is released.

### Using the `spi_master` driver

- Initialize a SPI bus by calling `spi_bus_initialize`. Make sure to set the correct IO pins in the `bus_config` struct. Take care to set signals that are not needed to -1.
- Tell the driver about a SPI slave device connected to the bus by calling `spi_bus_add_device`. Make sure to configure any timing requirements the device has in the `dev_config` structure. You should now have a handle for the device, to be used when sending it a transaction.
- To interact with the device, fill one or more `spi_transaction_t` structure with any transaction parameters you need. Then send them either in a polling way or the interrupt way:
  - **Interrupt** Either queue all transactions by calling `spi_device_queue_trans`, and at a later time query the result using `spi_device_get_trans_result`, or handle all requests synchronously by feeding them into `spi_device_transmit`.
  - **Polling** Call the `spi_device_polling_transmit` to send polling transactions. Alternatively, you can send a polling transaction by `spi_device_polling_start` and `spi_device_polling_end` if you want to insert something between them.
- Optional: to do back-to-back transactions to a device, call `spi_device_acquire_bus` before and `spi_device_release_bus` after the transactions.
- Optional: to unload the driver for a 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`.

### Tips

1. **Transactions with small amount of data:** Sometimes, the amount of data is very small making it less than optimal allocating a separate buffer for it. If the data to be transferred is 32 bits or less, it can be stored in the transaction struct itself. For transmitted data, use the `tx_data` member for this and set the `SPI_TRANS_USE_TXDATA` flag on the transmission. For received data, use `rx_data` and set `SPI_TRANS_USE_RXDATA`. In both cases, do not touch the `tx_buffer` or `rx_buffer` members, because they use the same memory locations as `tx_data` and `rx_data`.
2. **Transactions with integers other than `uint8_t`** The SPI peripheral reads and writes the memory byte-by-byte. By default, the SPI works at MSB first mode, each bytes are sent or received from the MSB to the LSB. However, if you want to send data with length which is not multiples of 8 bits, unused bits are sent. E.g. you write `uint8_t data = 0x15 (00010101B)`, and set length to only 5 bits, the sent data is 00010B rather than expected 10101B.



Moreover, ESP32 is a little-endian chip whose lowest byte is stored at the very beginning address for `uint16_t` and `uint32_t` variables. Hence if a `uint16_t` is stored in the memory, its bit 7 is first sent, then bit 6 to 0, then comes its bit 15 to bit 8.

To send data other than `uint8_t` arrays, macros `SPI_SWAP_DATA_TX` is provided to shift your data to the MSB and swap the MSB to the lowest address; while `SPI_SWAP_DATA_RX` can be used to swap received data from the MSB to its correct place.

**GPIO matrix and IOMUX** Most peripheral signals in ESP32 can connect directly to a specific GPIO, which is called its IOMUX pin. When a peripheral signal is routed to a pin other than its IOMUX pin, ESP32 uses the less direct GPIO matrix to make this connection.

If the driver is configured with all SPI signals set to their specific IOMUX pins (or left unconnected), it will bypass the GPIO matrix. If any SPI signal is configured to a pin other than its IOMUX pin, the driver will automatically route all the signals via the GPIO Matrix. The GPIO matrix samples all signals at 80MHz and sends them between the GPIO and the peripheral.

When the GPIO matrix is used, signals faster than 40MHz cannot propagate and the setup time of MISO is more easily violated, since the input delay of MISO signal is increased. The maximum clock frequency with GPIO Matrix is 40MHz or less, whereas using all IOMUX pins allows 80MHz.

---

**注解:** More details about influence of input delay on the maximum clock frequency, see [Timing considerations](#) below.

---

IOMUX pins for SPI controllers are as below:

Pin Name	HSPI	VSPI
	GPIO Number	
CS0*	15	5
SCLK	14	18
MISO	12	19
MOSI	13	23
QUADWP	2	22
QUADHD	4	21

note \* Only the first device attaching to the bus can use CS0 pin.

**Notes to send mixed transactions to the same device** Though we suggest to send only one type (interrupt or polling) of transactions to one device to reduce coding complexity, it is supported to send both interrupt and polling transactions alternately. Notes below is to help you do this.

The polling transactions should be started when all the other transactions are finished, no matter they are polling or interrupt.

An unfinished polling transaction forbid other transactions from being sent. Always call `spi_device_polling_end` after `spi_device_polling_start` to allow other device using the bus, or allow other transactions to be started to the same device. You can use `spi_device_polling_transmit` to simplify this if you don't need to do something during your polling transaction.

An in-flight polling transaction would get disturbed by the ISR operation caused by interrupt transactions. Always make sure all the interrupt transactions sent to the ISR are finished before you call `spi_device_polling_start`. To do that, you can call `spi_device_get_trans_result` until all the transactions are returned.

It is strongly recommended to send mixed transactions to the same device in only one task to control the calling sequence of functions.

## Speed and Timing Considerations

**Transferring speed** There are three factors limiting the transferring speed: (1) The transaction interval, (2) The SPI clock frequency used. (3) The cache miss of SPI functions including callbacks. When large transactions are used, the clock frequency determines the transferring speed; while the interval effects the speed a lot if small transactions are used.

- Transaction interval: It takes time for the software to setup spi peripheral registers as well as copy data to FIFOs, or setup DMA links. When the interrupt transactions are used, an extra overhead is appended, from the cost of FreeRTOS queues and the time 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 interval. 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](#).
  - When the DMA is enabled, it needs about 2us per transaction to setup the linked list. When the master is transferring, it automatically read data from the linked list. If the DMA is not enabled, CPU has to write/read each byte to/from the FIFO by itself. Usually this is faster than 2us, but the transaction length is limited to 64 bytes for both write and read.

Typical transaction interval with one byte data is as below:

	Typical Transaction Time (us)	
	Interrupt	Polling
DMA	24	8
No DMA	22	7

- SPI clock frequency: Each byte transferred takes 8 times of the clock period  $8/f_{spi}$ . If the clock frequency is too high, some functions may be limited to use. See [Timing considerations](#).
- The cache miss: the default config puts only the ISR into the IRAM. Other SPI related functions including the driver itself and the callback may suffer from the cache miss and wait for some time while reading code from the 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 this.

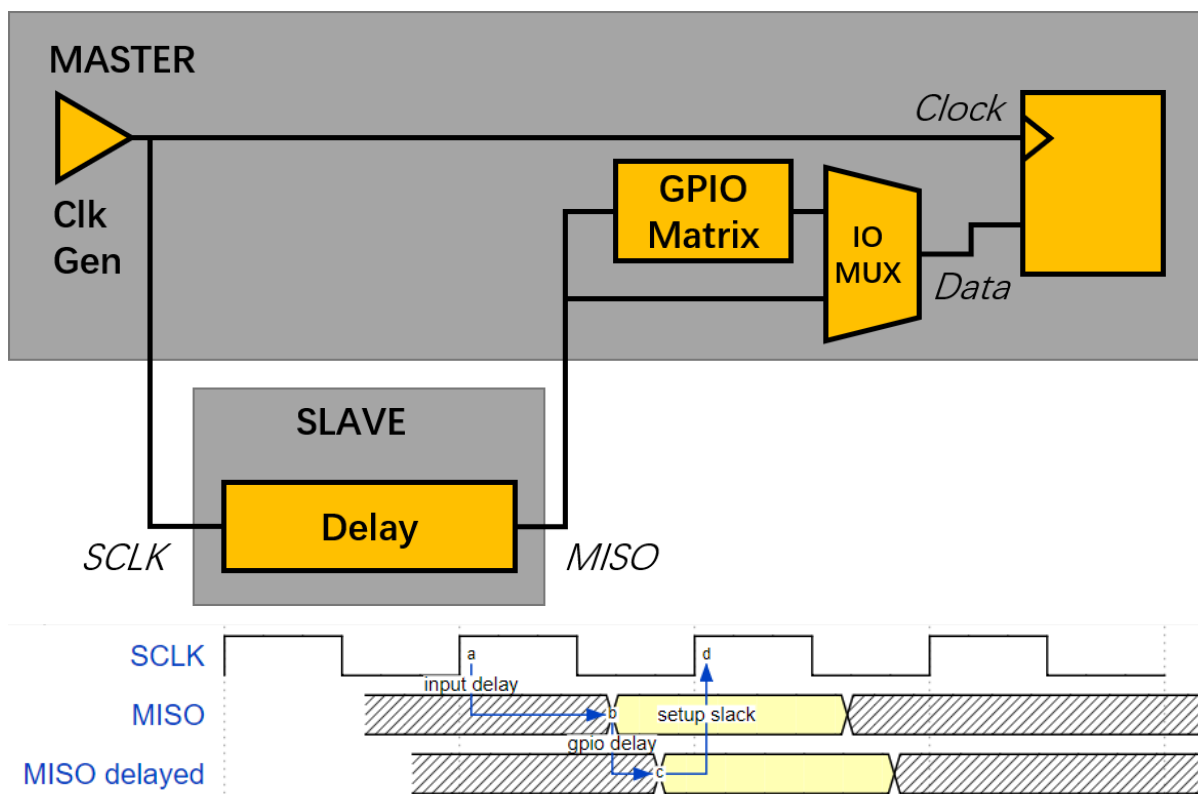
For an interrupt transaction, the overall cost is  $20+8n/F_{spi}[MHz]$  [us] for n bytes transferred in one transaction. Hence the transferring speed is :  $n/(20+8n/F_{spi})$ . Example of transferring speed under 8MHz clock speed:

Frequency (MHz)	Transaction Interval (us)	Transaction Length (bytes)	Total Time (us)	Total Speed (kBps)
8	25	1	26	38.5
8	25	8	33	242.4
8	25	16	41	490.2
8	25	64	89	719.1
8	25	128	153	836.6

When the length of transaction is short, the cost of transaction interval is really high. Please try to squash data into one transaction if possible to get higher transfer speed.

BTW, 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 `intr_flags` member of [spi\\_bus\\_config\\_t](#). Then all the transactions queued before the flash operations will be handled by the ISR continuously during flash operation. Note that the callback of each devices, and their callee functions, should be in the IRAM in this case, or your callback will crash due to cache miss.

**Timing considerations** As shown in the figure below, there is a delay on the MISO signal after SCLK launch edge and before it's latched by the internal register. As a result, the MISO pin setup time is the limiting factor for SPI clock speed. When the delay is too large, setup slack is  $< 0$  and the setup timing requirement is violated, leads to the failure of reading correctly.



The maximum frequency allowed is related to the *input delay* (maximum valid time after SCLK on the MISO bus), as well as the usage of GPIO matrix. The maximum frequency allowed is reduced to about 33~77% (related to existing *input delay*) when the GPIO matrix is used. To work at higher frequency, you have to use the IOMUX pins or the *dummy bit workaround*. You can get the maximum reading frequency of the master by `spi_get_freq_limit`.

**Dummy bit workaround:** We can insert dummy clocks (during which the host does not read data) before the read phase actually begins. The slave still sees the dummy clocks and gives out data, but the host does not read until the read phase. This compensates the lack of setup time of MISO required by the host, allowing the host reading at higher frequency.

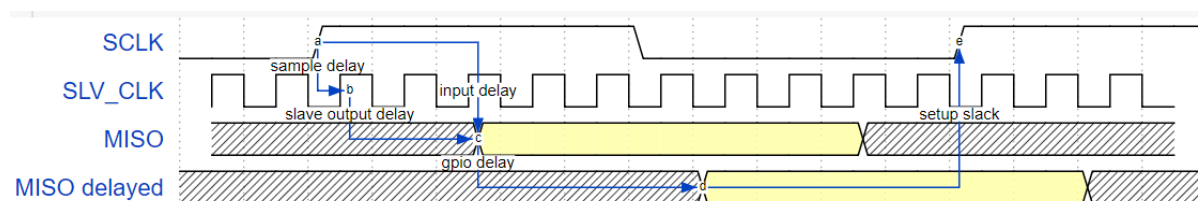
In the ideal case (the slave is so fast that the input delay is shorter than an apb clock, 12.5ns), the maximum frequency host can read (or read and write) under different conditions is as below:

Frequency Limit (MHz)		Dummy Bits Used By Driver	Comments
GPIO matrix	IOMUX pins		
26.6	80	No	
40	–	Yes	Half Duplex, no DMA allowed

And if the host only writes, the *dummy bit workaround* is not used and the frequency limit is as below:

GPIO matrix (MHz)	IOMUX pins (MHz)
40	80

The spi master driver can work even if the *input delay* in the `spi_device_interface_config_t` is set to 0. However, setting a accurate value helps to: (1) calculate the frequency limit in full duplex mode, and (2) compensate the timing correctly by dummy bits in half duplex mode. You may find the maximum data valid time after the launch edge of SPI clocks in the AC characteristics chapter of the device specifications, or measure the time on a oscilloscope or logic analyzer.



As shown in the figure above, the input delay is usually:

$$[\text{input delay}] = [\text{sample delay}] + [\text{slave output delay}]$$

1. The sample delay is the maximum random delay due to the asynchronization of SCLK and peripheral clock of the slave. It's usually 1 slave peripheral clock if the clock is asynchronize with SCLK, or 0 if the slave just use the SCLK to latch the SCLK and launch MISO data. e.g. for ESP32 slaves, the delay is 12.5ns (1 apb clock), while it is reduced to 0 if the slave is in the same chip as the master.
2. The slave output delay is the time for the MOSI to be stable after the launch edge. e.g. for ESP32 slaves, the output delay is 37.5ns (3 apb clocks) when IOMUX pins in the slave is used, or 62.5ns (5 apb clocks) if through the GPIO matrix.

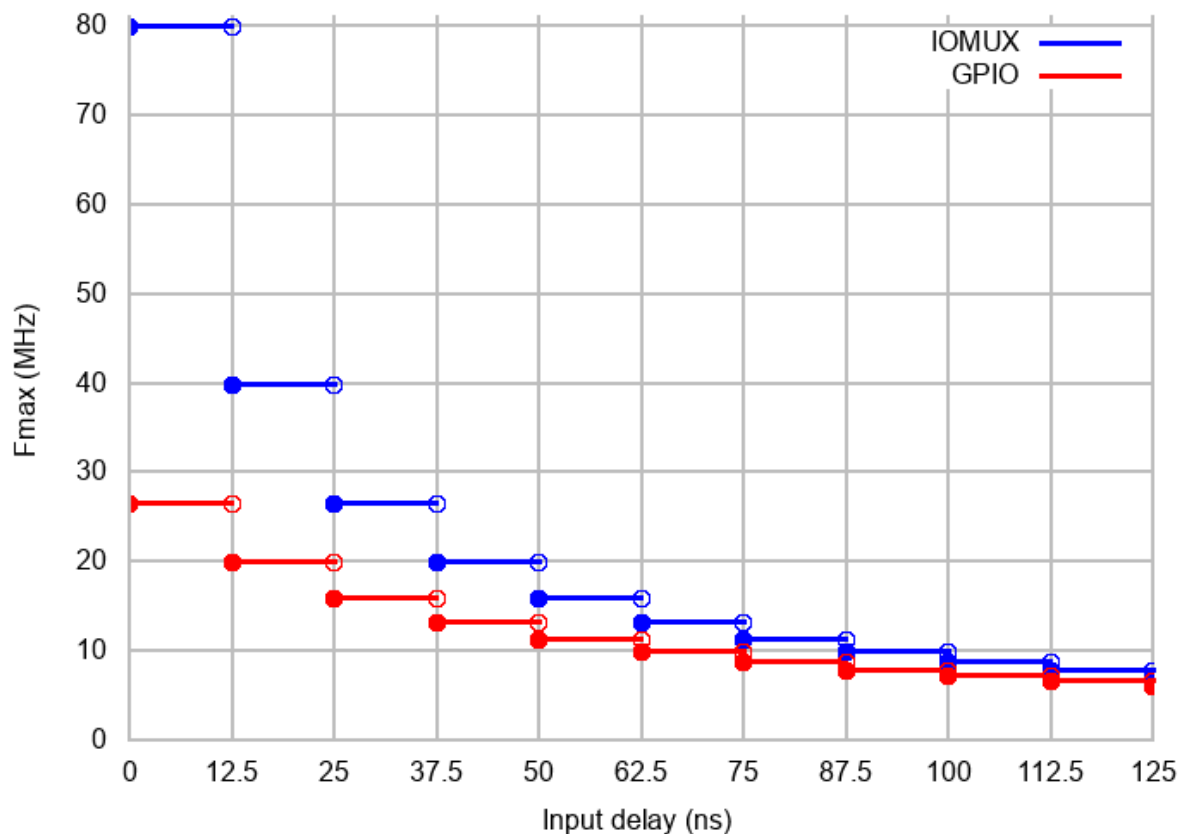
Some typical delays are shown in the following table:

Device	Input delay (ns)
Ideal device	0
ESP32 slave IOMUX*	50
ESP32 slave GPIO*	75
ESP32 slave is on an independent chip, 12.5ns sample delay included.	

The MISO path delay( $t_v$ ), consists of slave *input delay* and master *GPIO matrix delay*, finally determines the frequency limit, above which the full duplex mode will not work, or dummy bits are used in the half duplex mode. The frequency limit is:

$$\text{Freq limit}[\text{MHz}] = 80 / (\text{floor}(\text{MISO delay}[\text{ns}] / 12.5) + 1)$$

The figure below shows the relations of frequency limit against the input delay. 2 extra apb clocks should be counted into the MISO delay if the GPIO matrix in the master is used.



Corresponding frequency limit for different devices with different *input delay* are shown in the following table:

Master	Input delay (ns)	MISO path delay (ns)	Freq. limit (MHz)
IOMUX (0ns)	0	0	80
	50	50	16
	75	75	11.43
GPIO (25ns)	0	25	26.67
	50	75	11.43
	75	100	8.89

### Known Issues

- Half duplex mode is not compatible with DMA when both writing and reading phases exist. If such transactions are required, you have to use one of the alternative solutions:
  - use full-duplex mode instead.
  - disable the DMA by setting the last parameter to 0 in bus initialization function just as below:

```
ret=spi_bus_initialize(VSPI_HOST, &buscfg, 0);
```

this may prohibit you from transmitting and receiving data longer than 64 bytes.
  - try to use command and address field to replace the write phase.
- Full duplex mode is not compatible with the *dummy bit workaround*, hence the frequency is limited. See [dummy bit speed-up workaround](#).
- `cs_ena_pretrans` is not compatible with command, address phases in full duplex mode.

### Application Example

Display graphics on the 320x240 LCD of WROVER-Kits: [peripherals/spi\\_master](#).

## API Reference - SPI Common

### Header File

- [soc/include/hal/spi\\_types.h](#)

### Macros

`SPI_HOST`

`HSPI_HOST`

`VSPI_HOST`

### Enumerations

`enum spi_host_device_t`

Enum with the three SPI peripherals that are software-accessible in it.

Values:

`SPI1_HOST = 0`  
SPI1.

`SPI2_HOST = 1`  
SPI2.

`SPI3_HOST = 2`  
SPI3.

### Header File

- [driver/include/driver/spi\\_common.h](#)

### Functions

`esp_err_t spi_bus_initialize` (*spi\_host\_device\_t* host, **const** *spi\_bus\_config\_t* \*bus\_config, int dma\_chan)

Initialize a SPI bus.

**Warning** For now, only supports HSPI and VSPI.

**Warning** If a DMA channel is selected, any transmit and receive buffer used should be allocated in DMA-capable memory.

**Warning** 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.

#### Return

- `ESP_ERR_INVALID_ARG` if configuration is invalid
- `ESP_ERR_INVALID_STATE` if host already is in use
- `ESP_ERR_NO_MEM` if out of memory
- `ESP_OK` on success

#### Parameters

- `host`: 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`: Either channel 1 or 2, or 0 in the case when no DMA is required. Selecting a DMA channel for a SPI bus allows transfers on the bus to have sizes only limited by the amount of internal memory. Selecting no DMA channel (by passing the value 0) limits the amount of bytes transferred to a maximum of 64. Set to 0 if only the SPI flash uses this bus.

`esp_err_t spi_bus_free` (*spi\_host\_device\_t* host)

Free a SPI bus.

**Warning** In order for this to succeed, all devices have to be removed first.

#### Return

- `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

#### Parameters

- `host`: SPI peripheral to free

### Structures

#### **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.

**Note** Be advised that the slave driver does not use the quadwp/quadhd lines and fields in `spi_bus_config_t` referring to these lines will be ignored and can thus safely be left uninitialized.

### Public Members

#### int `mosi_io_num`

GPIO pin for Master Out Slave In (=spi\_d) signal, 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 `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 which is used as D2 in 4-bit communication modes, or -1 if not used.

#### int `quadhd_io_num`

GPIO pin for HD (HoLD) signal which is used as D3 in 4-bit communication modes, or -1 if not used.

#### int `max_transfer_sz`

Maximum transfer size, in bytes. Defaults to 4094 if 0.

#### 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  $\leq 32$  bits to the format which can be sent by the SPI driver directly.

E.g. to send 9 bits of data, you can:

```
uint16_t data = SPI_SWAP_DATA_TX(0x145, 9);
```

Then points `tx_buffer` to `&data`.

#### Parameters

- `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  $\leq 32$  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:

```
uint16_t data = SPI_SWAP_DATA_RX(*(uint32_t*)t->rx_data, 15);
```

**Parameters**

- 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\_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 CLK 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\_NATIVE\_PINS****API Reference - SPI Master****Header File**

- [driver/include/driver/spi\\_master.h](#)

**Functions**

`esp_err_t spi_bus_add_device` (*spi\_host\_device\_t* host, **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.

**Note** 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.

**Return**

- ESP\_ERR\_INVALID\_ARG if parameter is invalid
- 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

**Parameters**



- `host`: SPI peripheral to allocate device on
- `dev_config`: SPI interface protocol config for the device
- `handle`: Pointer to variable to hold the device handle

*esp\_err\_t* **spi\_bus\_remove\_device** (*spi\_device\_handle\_t* handle)

Remove a device from the SPI bus.

#### Return

- `ESP_ERR_INVALID_ARG` if parameter is invalid
- `ESP_ERR_INVALID_STATE` if device already is freed
- `ESP_OK` on success

#### Parameters

- `handle`: Device handle to free

*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`.

**Note** Normally a device cannot start (queue) polling and interrupt transactions simultaneously.

#### Return

- `ESP_ERR_INVALID_ARG` if parameter is invalid
- `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

#### Parameters

- `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; use `portMAX_DELAY` to never time out.

*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.

#### Return

- `ESP_ERR_INVALID_ARG` if parameter is invalid
- `ESP_ERR_TIMEOUT` if there was no completed transaction before `ticks_to_wait` expired
- `ESP_OK` on success

#### 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.

*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 hasn't been finalized.

**Note** 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.

#### Return

- `ESP_ERR_INVALID_ARG` if parameter is invalid
- `ESP_OK` on success

**Parameters**

- `handle`: Device handle obtained using `spi_host_add_dev`
- `trans_desc`: Description of transaction to execute

`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.

**Note** 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.

**Return**

- `ESP_ERR_INVALID_ARG` if parameter is invalid
- `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

**Parameters**

- `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.

`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.

**Return**

- `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

**Parameters**

- `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.

`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.

**Note** 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.

**Return**

- `ESP_ERR_INVALID_ARG` if parameter is invalid
- `ESP_OK` on success

**Parameters**

- `handle`: Device handle obtained using `spi_host_add_dev`
- `trans_desc`: Description of transaction to execute

`esp_err_t spi_device_acquire_bus` (`spi_device_handle_t device`, `TickType_t wait`)

Occupy 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.

**Note** The function will wait until all the existing transactions have been sent.

**Return**

- `ESP_ERR_INVALID_ARG`: `wait` is not set to `portMAX_DELAY`.
- `ESP_OK`: Success.

**Parameters**

- `device`: The device to occupy the bus.

- `wait`: Time to wait before the the bus is occupied by the device. Currently MUST set to `port-MAX_DELAY`.

void **spi\_device\_release\_bus** (*spi\_device\_handle\_t dev*)

Release the SPI bus occupied by the device. All other devices can start sending transactions.

#### Parameters

- `dev`: The device to release the bus.

int **spi\_cal\_clock** (int *fapb*, int *hz*, int *duty\_cycle*, uint32\_t \**reg\_o*)

Calculate the working frequency that is most close to desired frequency, and also the register value.

#### Parameters

- `fapb`: The frequency of apb clock, should be `APB_CLK_FREQ`.
- `hz`: Desired working frequency
- `duty_cycle`: Duty cycle of the spi clock
- `reg_o`: Output of value to be set in clock register, or NULL if not needed.

**Return** Actual working frequency that most fit.

int **spi\_get\_actual\_clock** (int *fapb*, int *hz*, int *duty\_cycle*)

Calculate the working frequency that is most close to desired frequency.

**Return** Actual working frequency that most fit.

#### Parameters

- `fapb`: The frequency of apb clock, should be `APB_CLK_FREQ`.
- `hz`: Desired working frequency
- `duty_cycle`: Duty cycle of the spi clock

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.

**Note** If `**dummy_o` is not zero, it means dummy bits should be applied in half duplex mode, and full duplex mode may not work.

#### Parameters

- `gpio_is_used`: True if using GPIO matrix, or False if iomux pins are used.
- `input_delay_ns`: Input delay from SCLK launch edge to MISO data valid.
- `eff_clk`: Effective clock frequency (in Hz) from `spi_cal_clock`.
- `dummy_o`: Address of dummy bits used output. Set to NULL if not needed.
- `cycles_remain_o`: Address of cycles remaining (after dummy bits are used) output.
  - -1 If too many cycles remaining, suggest to compensate half a clock.
  - 0 If no remaining cycles or dummy bits are not used.
  - positive value: cycles suggest to compensate.

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.

**Return** Frequency limit of current configurations.

#### Parameters

- `gpio_is_used`: True if using GPIO matrix, or False if native pins are used.
- `input_delay_ns`: Input delay from SCLK launch edge to MISO data valid.

## Structures

**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

uint8\_t **command\_bits**

Default amount of bits in command phase (0-16), used when `SPI_TRANS_VARIABLE_CMD` is not

used, otherwise ignored.

**uint8\_t address\_bits**

Default amount of bits in address phase (0-64), used when `SPI_TRANS_VARIABLE_ADDR` is not used, otherwise ignored.

**uint8\_t dummy\_bits**

Amount of dummy bits to insert between address and data phase.

**uint8\_t mode**

SPI mode (0-3)

**uint8\_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.

**uint8\_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-IDF3.0.**

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 base**

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.

**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\_HALFDUPLEX**  
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\_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.

**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`.

**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.
```

## 2.3.16 SPI Slave driver

**Overview**

The ESP32 has four SPI peripheral devices, called SPI0, SPI1, HSPI and VSPI. SPI0 is entirely dedicated to the flash cache the ESP32 uses to map the SPI flash device it is connected to into memory. SPI1 is connected to the same hardware lines as SPI0 and is used to write to the flash chip. HSPI and VSPI are free to use, and with the `spi_slave` driver, these can be used as a SPI slave, driven from a connected SPI master.

**The `spi_slave` driver** The `spi_slave` driver allows using the HSPI and/or VSPI peripheral as a full-duplex SPI slave. It can send/receive transactions within 64 bytes, or make use of DMA to send/receive transactions longer than that. However, there are some [known issues](#) when the DMA is enabled.

**Terminology** The `spi_slave` driver uses the following terms:

- **Host:** The SPI peripheral inside the ESP32 initiating the SPI transmissions. One of HSPI or VSPI.
- **Bus:** The SPI bus, common to all SPI devices connected to a master. In general the bus consists of the `miso`, `mosi`, `sclk` and optionally `quadwp` and `quadhd` signals. The SPI slaves are connected to these signals in parallel. Each SPI slave is also connected to one CS signal.
  - `miso` - Also known as `q`, this is the output of the serial stream from the ESP32 to the SPI master
  - `mosi` - Also known as `d`, this is the output of the serial stream from the SPI master to the ESP32
  - `sclk` - Clock signal. Each data bit is clocked out or in on the positive or negative edge of this signal
  - `cs` - Chip Select. An active Chip Select delineates a single transaction to/from a slave.
- **Transaction:** One instance of CS going active, data transfer from and to a master happening, and CS going inactive again. Transactions are atomic, as in they will never be interrupted by another transaction.

**SPI transactions** A full-duplex SPI transaction starts with the master pulling CS low. After this happens, the master starts sending out clock pulses on the CLK line: every clock pulse causes a data bit to be shifted from the master to the slave on the MOSI line and vice versa on the MISO line. At the end of the transaction, the master makes CS high again.

---

**注解:** The SPI slave peripheral relies on the control of software very much. The master shouldn't start a transaction when the slave hasn't prepared for it. Using one more GPIO as the handshake signal to sync is a good idea. For more details, see [Transaction interval](#).

---

**GPIO matrix and IOMUX** Most peripheral signals in ESP32 can connect directly to a specific GPIO, which is called its IOMUX pin. When a peripheral signal is routed to a pin other than its IOMUX pin, ESP32 uses the less direct GPIO matrix to make this connection.

If the driver is configured with all SPI signals set to their specific IOMUX pins (or left unconnected), it will bypass the GPIO matrix. If any SPI signal is configured to a pin other than its IOMUX pin, the driver will automatically route all the signals via the GPIO Matrix. The GPIO matrix samples all signals at 80MHz and sends them between the GPIO and the peripheral.

When the GPIO matrix is used, setup time of MISO is more easily violated, since the output delay of MISO signal is increased.

---

**注解:** More details about influence of output delay on the maximum clock frequency, see [Timing considerations](#) below.

---

IOMUX pins for SPI controllers are as below:

Pin Name	HSPI	VSPI
	GPIO Number	
CS0*	15	5
SCLK	14	18
MISO	12	19
MOSI	13	23
QUADWP	2	22
QUADHD	4	21

note \* Only the first device attaching to the bus can use CS0 pin.

### Using the spi\_slave driver

- Initialize a SPI peripheral as a slave by calling `spi_slave_initialize`. Make sure to set the correct IO pins in the `bus_config` struct. Take care to set signals that are not needed to -1. A DMA channel (either 1 or 2) must be given if transactions will be larger than 32 bytes, if not the `dma_chan` parameter may be 0.
- To set up a transaction, fill one or more `spi_transaction_t` structure with any transaction parameters you need. Either queue all transactions by calling `spi_slave_queue_trans`, later quering the result using `spi_slave_get_trans_result`, or handle all requests synchronously by feeding them into `spi_slave_transmit`. The latter two functions will block until the master has initiated and finished a transaction, causing the queued data to be sent and received.
- Optional: to unload the SPI slave driver, call `spi_slave_free`.

**Transaction data and master/slave length mismatches** Normally, data to be transferred to or from a device will be read from or written to a chunk of memory indicated by the `rx_buffer` and `tx_buffer` members of the transaction structure. The SPI driver may decide to use DMA for transfers, so these buffers should be allocated in DMA-capable memory using `pvPortMallocCaps(size, MALLOC_CAP_DMA)`.

The amount of data written to the buffers is limited by the `length` member of the transaction structure: the driver will never read/write more data than indicated there. The `length` cannot define the actual length of the SPI transaction; this is determined by the master as it drives the clock and CS lines. The actual length transferred can be read from the `trans_len` member of the `spi_slave_transaction_t` structure after transaction. In case the length of the transmission is larger than the buffer length, only the start of the transmission will be sent and received, and the `trans_len` is set to `length` instead of the actual length. It's recommended to set `length` longer than the maximum length expected if the `trans_len` is required. In case the transmission length is shorter than the buffer length, only data up to the length of the buffer will be exchanged.

Warning: Due to a design peculiarity in the ESP32, if the amount of bytes sent by the master or the length of the transmission queues in the slave driver, in bytes, is not both larger than eight and dividable by four, the SPI hardware can fail to write the last one to seven bytes to the receive buffer.



## Speed and Timing considerations

**Transaction interval** The SPI slave is designed as a general purpose device controlled by the CPU. Different from dedicated devices, CPU-based SPI slave doesn't have too much pre-defined registers. All transactions should be triggered by the CPU, which means the response speed would not be real-time, and there'll always be noticeable intervals between transfers.

During the transaction intervals, the device is not prepared for transactions, the response is not meaningful at all. It is suggested to use `spi_slave_queue_trans()` with `spi_slave_get_trans_result()` to shorten the interval to half the case when using `spi_slave_transmit()`.

The master should always wait for the slave to be ready to start new transactions. Suggested way is to use a gpio by the slave to indicate whether it's ready. The example is in [peripherals/spi\\_slave](#).

**SCLK frequency requirement** The spi slave is designed to work under 10MHz or lower. The clock and data cannot be recognized or received correctly if the clock is too fast or doesn't have a 50% duty cycle.

Moreover, there are more requirements if the data meets the timing requirement:

- **Read (MOSI):** Given that the MOSI is valid right at the launch edge, the slave can read data correctly. Luckily, it's usually the case for most masters.
- **Write (MISO):** To meet the requirement that MISO is stable before the next latch edge of SPI clock, the output delay of MISO signal should be shorter than half a clock. The output delay and frequency limitation (given that the clock is balanced) of different cases are as below :

	Output delay of MISO (ns)	Freq. limit (MHZ)
IOMUX	43.75	<11.4
GPIO matrix	68.75	<7.2

### Note:

1. Random error will happen if the frequency exactly equals the limitation
2. The clock uncertainty between master and slave (12.5ns) is included.
3. The output delay is measured under ideal case (free of load). When the loading of MISO pin is too heavy, the output delay will be longer, and the maximum allowed frequency will be lower.

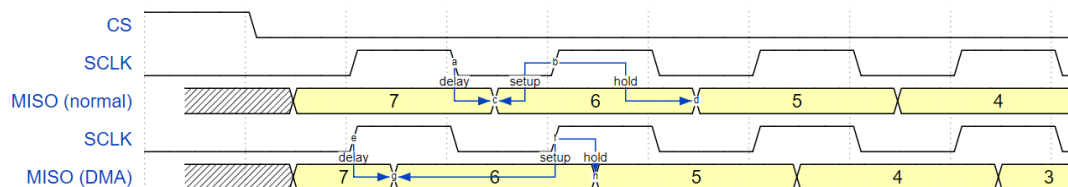
There is an exceptions: The frequency is allowed to be higher if the master has more toleration for the MISO setup time, e.g. latch data at the next edge than expected, or configurable latching time.

## Restrictions and Known issues

1. If the DMA is enabled, the rx buffer should be WORD aligned, i.e. Start from the boundary of 32-bit and have length of multiples of 4 bytes. Or the DMA may write incorrectly or out of the boundary. The driver will check for this.

Also, master should write lengths which are a multiple of 4 bytes. Data longer than that will be discarded.

2. Furthermore, the DMA requires a spi mode 1/3 timing. When using spi mode 0/2, the MISO signal has to output half a clock earlier to meet the timing. The new timing is as below:



The hold time after the latch edge is 68.75ns (when GPIO matrix is bypassed), no longer half a SPI clock. The master should sample immediately at the latch edge, or communicate in mode 1/3. Or just initial the spi slave without DMA.

## Application Example

Slave/master communication: [peripherals/spi\\_slave](#).

## API Reference

### Header File

- [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, int dma\_chan)

Initialize a SPI bus as a slave interface.

**Warning** For now, only supports HSPI and VSPI.

**Warning** If a DMA channel is selected, any transmit and receive buffer used should be allocated in DMA-capable memory.

**Warning** 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.

#### Return

- ESP\_ERR\_INVALID\_ARG if configuration is invalid
- ESP\_ERR\_INVALID\_STATE if host already is in use
- ESP\_ERR\_NO\_MEM if out of memory
- ESP\_OK on success

#### Parameters

- 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: Either 1 or 2. A SPI bus used by this driver must have a DMA channel associated with it. The SPI hardware has two DMA channels to share. This parameter indicates which one to use.

*esp\_err\_t spi\_slave\_free* (*spi\_host\_device\_t* host)

Free a SPI bus claimed as a SPI slave interface.

#### Return

- 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

#### Parameters

- host: SPI peripheral to free

*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.

Queues 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.

#### Return

- ESP\_ERR\_INVALID\_ARG if parameter is invalid
- ESP\_OK on success

#### 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 a room in the queue; use `portMAX_DELAY` to never time out.

`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`.

#### Return

- `ESP_ERR_INVALID_ARG` if parameter is invalid
- `ESP_OK` on success

#### Parameters

- `host`: SPI peripheral to that is acting as a slave
- `[out] trans_desc`: 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 `portMAX_DELAY` to never time out.

`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`.

#### Return

- `ESP_ERR_INVALID_ARG` if parameter is invalid
- `ESP_OK` on success

#### Parameters

- `host`: SPI peripheral to 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 `portMAX_DELAY` to never time out.

## 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 (0-3)

***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.

**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.

**Type Definitions**

```
typedef struct spi_slave_transaction_t spi_slave_transaction_t
```

```
typedef void (*slave_transaction_cb_t) (spi_slave_transaction_t *trans)
```

**2.3.17 Timer****Introduction**

The ESP32 chip contains two hardware timer groups. Each group has two general-purpose hardware timers. They are all 64-bit generic timers based on 16-bit prescalers and 64-bit up / down counters which are capable of being auto-reloaded.

## Functional Overview

The following sections of this document cover the typical steps to configure and operate a timer:

- *Timer Initialization* - covers which parameters should be set up to get the timer working, and also what specific functionality is provided depending on the timer configuration.
- *Timer Control* - describes how to read a timer's value, pause or start a timer, and change how it operates.
- *Alarms* - shows how to set and use alarms.
- *Interrupts* - explains how to enable and use interrupts.

**Timer Initialization** The two ESP32 timer groups, with two timers in each, provide the total of four individual timers for use. An ESP32 timer group should be identified using `timer_group_t`. An individual timer in a group should be identified with `timer_idx_t`.

First of all, the timer should be initialized by calling the function `timer_init()` and passing a structure `timer_config_t` to it to define how the timer should operate. In particular, the following timer parameters can be set:

- **Divider:** Sets how quickly the timer's counter is "ticking". The setting `divider` is used as a divisor of the incoming 80 MHz APB\_CLK clock.
- **Mode:** Sets if the counter should be incrementing or decrementing. It can be defined using `counter_dir` by selecting one of the values from `timer_count_dir_t`.
- **Counter Enable:** If the counter is enabled, it will start incrementing / decrementing immediately after calling `timer_init()`. You can change the behavior with `counter_en` by selecting one of the values from `timer_start_t`.
- **Alarm Enable:** Can be set using `alarm_en`.
- **Auto Reload:** Sets if the counter should `auto_reload` the initial counter value on the timer's alarm or continue incrementing or decrementing.
- **Interrupt Type:** Select which interrupt type should be triggered on the timer's alarm. Set the value defined in `timer_intr_mode_t`.

To get the current values of the timer's settings, use the function `timer_get_config()`.

**Timer Control** Once the timer is enabled, its counter starts running. To enable the timer, call the function `timer_init()` with `counter_en` set to `true`, or call `timer_start()`. You can specify the timer's initial counter value by calling `timer_set_counter_value()`. To check the timer's current value, call `timer_get_counter_value()` or `timer_get_counter_time_sec()`.

To pause the timer at any time, call `timer_pause()`. To resume it, call `timer_start()`.

To reconfigure the timer, you can call `timer_init()`. This function is described in Section *Timer Initialization*.

You can also reconfigure the timer by using dedicated functions to change individual settings:

Setting	Dedicated Function	Description
Divider	<code>timer_set_divider()</code>	Change the rate of ticking. To avoid unpredictable results, the timer should be paused when changing the divider. If the timer is running, <code>timer_set_divider()</code> pauses it, change the setting, and start the timer again.
Mode	<code>timer_set_counter_dir()</code>	Set if the counter should be incrementing or decrementing
Auto Reload	<code>timer_set_auto_reload()</code>	Set if the initial counter value should be reloaded on the timer's alarm

**Alarms** To set an alarm, call the function `timer_set_alarm_value()` and then enable the alarm using `timer_set_alarm()`. The alarm can also be enabled during the timer initialization stage, when `timer_init()` is called.

After the alarm is enabled, and the timer reaches the alarm value, the following two actions can occur depending on the configuration:

- An interrupt will be triggered if previously configured. See Section *Interrupts* on how to configure interrupts.
- When `auto_reload` is enabled, the timer's counter will automatically be reloaded to start counting again from a previously configured value. This value should be set in advance with `timer_set_counter_value()`.

**注解:**

- If an alarm value is set and the timer has already reached this value, the alarm is triggered immediately.
- Once triggered, the alarm is disabled automatically and needs to be re-enabled to trigger again.

To check the specified alarm value, call `timer_get_alarm_value()`.

**Interrupts** Registration of the interrupt handler for a specific timer or a timer group can be done by calling `timer_isr_register()`.

To enable interrupts for a timer group, call `timer_group_intr_enable()`, for a specific timer call `timer_enable_intr()`. To disable interrupts for a timer group, call `timer_group_intr_disable()`, for a specified timer, call `timer_disable_intr()`.

When handling an interrupt within an interrupt service routine (ISR), the interrupt status bit needs to be explicitly cleared. To do that, set the `TIMERGN.int_clr_timers.tM` structure, defined in `soc/esp32/include/soc/timer_group_struct.h`. In this structure, N is the timer group number [0, 1], M is the timer number [0, 1]. For example, to clear an interrupt status bit for the timer 1 in the timer group 0, call the following:

```
TIMERG0.int_clr_timers.t1 = 1
```

For more information on how to use interrupts, please see the application example below.

**Application Example**

The 64-bit hardware timer example: [peripherals/timer\\_group](#).

**API Reference****Header File**

- [driver/include/driver/timer.h](#)

**Functions**

`esp_err_t timer_get_counter_value(timer_group_t group_num, timer_idx_t timer_num, uint64_t *timer_val)`

Read the counter value of hardware timer.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- `group_num`: Timer group, 0 for `TIMERG0` or 1 for `TIMERG1`
- `timer_num`: Timer index, 0 for `hw_timer[0]` & 1 for `hw_timer[1]`
- `timer_val`: Pointer to accept timer counter value.

`esp_err_t timer_get_counter_time_sec(timer_group_t group_num, timer_idx_t timer_num, double *time)`

Read the counter value of hardware timer, in unit of a given scale.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- `group_num`: Timer group, 0 for `TIMERG0` or 1 for `TIMERG1`
- `timer_num`: Timer index, 0 for `hw_timer[0]` & 1 for `hw_timer[1]`
- `time`: Pointer, type of `double*`, to accept timer counter value, in seconds.

`esp_err_t timer_set_counter_value` (`timer_group_t` `group_num`, `timer_idx_t` `timer_num`, `uint64_t` `load_val`)

Set counter value to hardware timer.

#### Return

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error

#### Parameters

- `group_num`: Timer group, 0 for `TIMERG0` or 1 for `TIMERG1`
- `timer_num`: Timer index, 0 for `hw_timer[0]` & 1 for `hw_timer[1]`
- `load_val`: Counter value to write to the hardware timer.

`esp_err_t timer_start` (`timer_group_t` `group_num`, `timer_idx_t` `timer_num`)

Start the counter of hardware timer.

#### Return

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error

#### Parameters

- `group_num`: Timer group number, 0 for `TIMERG0` or 1 for `TIMERG1`
- `timer_num`: Timer index, 0 for `hw_timer[0]` & 1 for `hw_timer[1]`

`esp_err_t timer_pause` (`timer_group_t` `group_num`, `timer_idx_t` `timer_num`)

Pause the counter of hardware timer.

#### Return

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error

#### Parameters

- `group_num`: Timer group number, 0 for `TIMERG0` or 1 for `TIMERG1`
- `timer_num`: Timer index, 0 for `hw_timer[0]` & 1 for `hw_timer[1]`

`esp_err_t timer_set_counter_mode` (`timer_group_t` `group_num`, `timer_idx_t` `timer_num`, `timer_count_dir_t` `counter_dir`)

Set counting mode for hardware timer.

#### Return

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error

#### Parameters

- `group_num`: Timer group number, 0 for `TIMERG0` or 1 for `TIMERG1`
- `timer_num`: Timer index, 0 for `hw_timer[0]` & 1 for `hw_timer[1]`
- `counter_dir`: Counting direction of timer, count-up or count-down

`esp_err_t timer_set_auto_reload` (`timer_group_t` `group_num`, `timer_idx_t` `timer_num`, `timer_autoreload_t` `reload`)

Enable or disable counter reload function when alarm event occurs.

#### Return

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error

#### Parameters

- `group_num`: Timer group number, 0 for `TIMERG0` or 1 for `TIMERG1`
- `timer_num`: Timer index, 0 for `hw_timer[0]` & 1 for `hw_timer[1]`
- `reload`: Counter reload mode.

`esp_err_t timer_set_divider` (`timer_group_t` `group_num`, `timer_idx_t` `timer_num`, `uint32_t` `divider`)

Set hardware timer source clock divider. Timer groups clock are divider from APB clock.

#### Return

- `ESP_OK` Success

- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- `group_num`: Timer group number, 0 for `TIMERG0` or 1 for `TIMERG1`
- `timer_num`: Timer index, 0 for `hw_timer[0]` & 1 for `hw_timer[1]`
- `divider`: Timer clock divider value. The divider's range is from 2 to 65536.

`esp_err_t timer_set_alarm_value(timer_group_t group_num, timer_idx_t timer_num, uint64_t alarm_value)`

Set timer alarm value.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- `group_num`: Timer group, 0 for `TIMERG0` or 1 for `TIMERG1`
- `timer_num`: Timer index, 0 for `hw_timer[0]` & 1 for `hw_timer[1]`
- `alarm_value`: A 64-bit value to set the alarm value.

`esp_err_t timer_get_alarm_value(timer_group_t group_num, timer_idx_t timer_num, uint64_t *alarm_value)`

Get timer alarm value.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- `group_num`: Timer group, 0 for `TIMERG0` or 1 for `TIMERG1`
- `timer_num`: Timer index, 0 for `hw_timer[0]` & 1 for `hw_timer[1]`
- `alarm_value`: Pointer of A 64-bit value to accept the alarm value.

`esp_err_t timer_set_alarm(timer_group_t group_num, timer_idx_t timer_num, timer_alarm_t alarm_en)`

Enable or disable generation of timer alarm events.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- `group_num`: Timer group, 0 for `TIMERG0` or 1 for `TIMERG1`
- `timer_num`: Timer index, 0 for `hw_timer[0]` & 1 for `hw_timer[1]`
- `alarm_en`: To enable or disable timer alarm function.

`esp_err_t timer_isr_register(timer_group_t group_num, timer_idx_t timer_num, void (*fn)) void * , void *arg, int intr_alloc_flags, timer_isr_handle_t *handle` Register Timer interrupt handler, the handler is an ISR. The handler will be attached to the same CPU core that this function is running on.

**Note** If the `intr_alloc_flags` value `ESP_INTR_FLAG_IRAM` is set, the handler function must be declared with `IRAM_ATTR` attribute and can only call functions in IRAM or ROM. It cannot call other timer APIs. Use direct register access to configure timers from inside the ISR in this case.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- `group_num`: Timer group number
- `timer_num`: Timer index of timer group
- `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.

`esp_err_t timer_init(timer_group_t group_num, timer_idx_t timer_num, const timer_config_t *config)`

Initializes and configure the timer.



**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- group\_num: Timer group number, 0 for TIMERG0 or 1 for TIMERG1
- timer\_num: Timer index, 0 for hw\_timer[0] & 1 for hw\_timer[1]
- config: Pointer to timer initialization parameters.

*esp\_err\_t* **timer\_get\_config** (*timer\_group\_t* group\_num, *timer\_idx\_t* timer\_num, *timer\_config\_t* \*config)

Get timer configure value.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- group\_num: Timer group number, 0 for TIMERG0 or 1 for TIMERG1
- timer\_num: Timer index, 0 for hw\_timer[0] & 1 for hw\_timer[1]
- config: Pointer of struct to accept timer parameters.

*esp\_err\_t* **timer\_group\_intr\_enable** (*timer\_group\_t* group\_num, *uint32\_t* en\_mask)

Enable timer group interrupt, by enable mask.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- group\_num: Timer group number, 0 for TIMERG0 or 1 for TIMERG1
- en\_mask: Timer interrupt enable mask. Use TIMG\_T0\_INT\_ENA\_M to enable t0 interrupt Use TIMG\_T1\_INT\_ENA\_M to enable t1 interrupt

*esp\_err\_t* **timer\_group\_intr\_disable** (*timer\_group\_t* group\_num, *uint32\_t* disable\_mask)

Disable timer group interrupt, by disable mask.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- group\_num: Timer group number, 0 for TIMERG0 or 1 for TIMERG1
- disable\_mask: Timer interrupt disable mask. Use TIMG\_T0\_INT\_ENA\_M to disable t0 interrupt Use TIMG\_T1\_INT\_ENA\_M to disable t1 interrupt

*esp\_err\_t* **timer\_enable\_intr** (*timer\_group\_t* group\_num, *timer\_idx\_t* timer\_num)

Enable timer interrupt.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- group\_num: Timer group number, 0 for TIMERG0 or 1 for TIMERG1
- timer\_num: Timer index.

*esp\_err\_t* **timer\_disable\_intr** (*timer\_group\_t* group\_num, *timer\_idx\_t* timer\_num)

Disable timer interrupt.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- group\_num: Timer group number, 0 for TIMERG0 or 1 for TIMERG1
- timer\_num: Timer index.

**Structures**

**struct timer\_config\_t**

Data structure with timer's configuration settings.

**Public Members**

bool **alarm\_en**

Timer alarm enable

bool **counter\_en**

Counter enable

*timer\_intr\_mode\_t* **intr\_type**

Interrupt mode

*timer\_count\_dir\_t* **counter\_dir**

Counter direction

bool **auto\_reload**

Timer auto-reload

uint32\_t **divider**

Counter clock divider. The divider's range is from 2 to 65536.

**Macros**

**TIMER\_BASE\_CLK**

Frequency of the clock on the input of the timer groups

**Type Definitions**

**typedef intr\_handle\_t timer\_isr\_handle\_t**

Interrupt handle, used in order to free the isr after use. Aliases to an int handle for now.

**Enumerations**

**enum timer\_group\_t**

Selects a Timer-Group out of 2 available groups.

*Values:*

**TIMER\_GROUP\_0 = 0**

Hw timer group 0

**TIMER\_GROUP\_1 = 1**

Hw timer group 1

**TIMER\_GROUP\_MAX**

**enum timer\_idx\_t**

Select a hardware timer from timer groups.

*Values:*

**TIMER\_0 = 0**

Select timer0 of GROUPx

**TIMER\_1 = 1**

Select timer1 of GROUPx

**TIMER\_MAX**

**enum timer\_count\_dir\_t**

Decides the direction of counter.

*Values:*

**TIMER\_COUNT\_DOWN = 0**  
Descending Count from cnt.hightcnt.low

**TIMER\_COUNT\_UP = 1**  
Ascending Count from Zero

**TIMER\_COUNT\_MAX**

**enum timer\_start\_t**  
Decides whether timer is on or paused.

*Values:*

**TIMER\_PAUSE = 0**  
Pause timer counter

**TIMER\_START = 1**  
Start timer counter

**enum timer\_alarm\_t**  
Decides whether to enable alarm mode.

*Values:*

**TIMER\_ALARM\_DIS = 0**  
Disable timer alarm

**TIMER\_ALARM\_EN = 1**  
Enable timer alarm

**TIMER\_ALARM\_MAX**

**enum timer\_intr\_mode\_t**  
Select interrupt type if running in alarm mode.

*Values:*

**TIMER\_INTR\_LEVEL = 0**  
Interrupt mode: level mode

**TIMER\_INTR\_MAX**

**enum timer\_autoreload\_t**  
Select if Alarm needs to be loaded by software or automatically reload by hardware.

*Values:*

**TIMER\_AUTORELOAD\_DIS = 0**  
Disable auto-reload: hardware will not load counter value after an alarm event

**TIMER\_AUTORELOAD\_EN = 1**  
Enable auto-reload: hardware will load counter value after an alarm event

**TIMER\_AUTORELOAD\_MAX**

## 2.3.18 Touch Sensor

### Introduction

A touch sensor system is built on a substrate which carries electrodes and relevant connections under a protective flat surface. When a user touches the surface, the capacitance variation is used to evaluate if the touch was valid.

ESP32 can handle up to 10 capacitive touch pads / GPIOs. The sensing pads can be arranged in different combinations (e.g., matrix, slider), so that a larger area or more points can be detected. The touch pad sensing process is under the control of a hardware-implemented finite-state machine (FSM) which is initiated by software or a dedicated hardware timer.

Design, operation, and control registers of a touch sensor are discussed in [ESP32 Technical Reference Manual \(PDF\)](#). Please refer to this manual for additional details on how this subsystem works.

In-depth design details of touch sensors and firmware development guidelines for ESP32 are available in [Touch Sensor Application Note](#). If you want to test touch sensors in various configurations without building them on your own, check the [Guide for ESP32-Sense Development Kit](#).

### Functionality Overview

Description of API is broken down into groups of functions to provide a quick overview of the following features:

- Initialization of touch pad driver
- Configuration of touch pad GPIO pins
- Taking measurements
- Adjusting parameters of measurements
- Filtering measurements
- Touch detection methods
- Setting up interrupts to report touch detection
- Waking up from Sleep mode on interrupt

For detailed description of a particular function, please go to Section [API Reference](#). Practical implementation of this API is covered in Section [Application Examples](#).

**Initialization** Before using a touch pad, you need to initialize the touch pad driver by calling the function `touch_pad_init()`. This function sets several `..._DEFAULT` driver parameters listed in [API Reference](#) under *Macros*. It also removes the information about which pads have been touched before, if any, and disables interrupts.

If the driver is not required anymore, deinitialize it by calling `touch_pad_deinit()`.

**Configuration** Enabling the touch sensor functionality for a particular GPIO is done with `touch_pad_config()`.

Use the function `touch_pad_set_fsm_mode()` to select if touch pad measurement (operated by FSM) should be started automatically by a hardware timer, or by software. If software mode is selected, use `touch_pad_sw_start()` to start the FSM.

**Touch State Measurements** The following two functions come in handy to read raw or filtered measurements from the sensor:

- `touch_pad_read()`
- `touch_pad_read_filtered()`

They can also be used, for example, to evaluate a particular touch pad design by checking the range of sensor readings when a pad is touched or released. This information can be then used to establish a touch threshold.

---

**注解:** Before using `touch_pad_read_filtered()`, you need to initialize and configure the filter by calling specific filter functions described in Section [Filtering of Measurements](#).

---

For the demonstration of how to use both read functions, check the application example [peripherals/touch\\_pad\\_read](#).

**Optimization of Measurements** A touch sensor has several configurable parameters to match the characteristics of a particular touch pad design. For instance, to sense smaller capacity changes, it is possible to narrow down the reference voltage range within which the touch pads are charged / discharged. The high and low reference voltages are set using the function `touch_pad_set_voltage()`.

Besides the ability to discern smaller capacity changes, a positive side effect is reduction of power consumption for low power applications. A likely negative effect is an increase in measurement noise. If the dynamic range of obtained

readings is still satisfactory, then further reduction of power consumption might be done by reducing the measurement time with `touch_pad_set_meas_time()`.

The following list summarizes available measurement parameters and corresponding ‘set’ functions:

- Touch pad charge / discharge parameters:
  - voltage range: `touch_pad_set_voltage()`
  - speed (slope): `touch_pad_set_cnt_mode()`
- Measurement time: `touch_pad_set_meas_time()`

Relationship between the voltage range (high / low reference voltages), speed (slope), and measurement time is shown in the figure below.

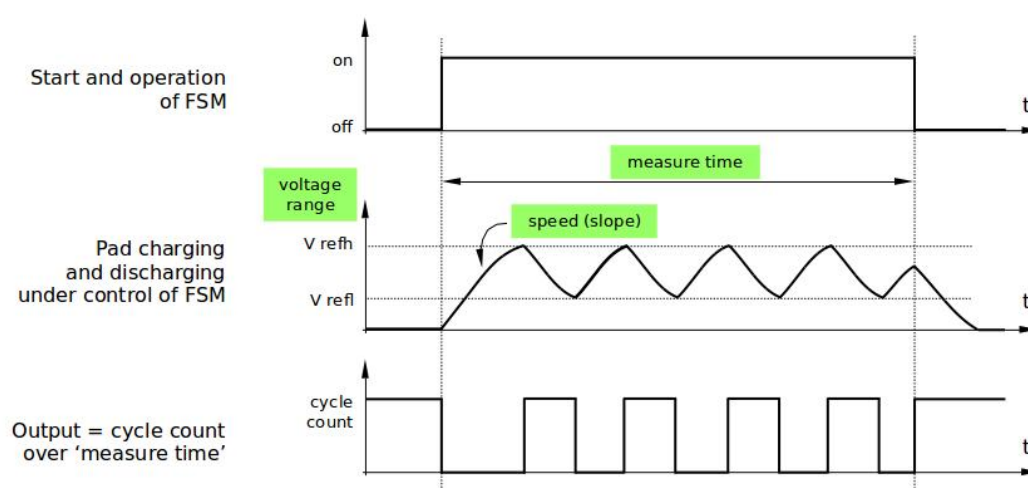


图 22: Touch pad - relationship between measurement parameters

The last chart *Output* represents the touch sensor reading, i.e., the count of pulses collected within the measurement time.

All functions are provided in pairs to *set* a specific parameter and to *get* the current parameter's value, e.g., `touch_pad_set_voltage()` and `touch_pad_get_voltage()`.

**Filtering of Measurements** If measurements are noisy, you can filter them with provided API functions. Before using the filter, please start it by calling `touch_pad_filter_start()`.

The filter type is IIR (infinite impulse response), and it has a configurable period that can be set with the function `touch_pad_set_filter_period()`.

You can stop the filter with `touch_pad_filter_stop()`. If not required anymore, the filter can be deleted by invoking `touch_pad_filter_delete()`.

**Touch Detection** Touch detection is implemented in ESP32's hardware based on the user-configured threshold and raw measurements executed by FSM. Use the functions `touch_pad_get_status()` to check which pads have been touched and `touch_pad_clear_status()` to clear the touch status information.

Hardware touch detection can also be wired to interrupts. This is described in the next section.

If measurements are noisy and capacity changes are small, hardware touch detection might be unreliable. To resolve this issue, instead of using hardware detection / provided interrupts, implement measurement filtering and perform touch detection in your own application. For sample implementation of both methods of touch detection, see [peripherals/touch\\_pad\\_interrupt](#).

**Touch Triggered Interrupts** Before enabling an interrupt on a touch detection, you should establish a touch detection threshold. Use the functions described in *Touch State Measurements* to read and display sensor measurements when a pad is touched and released. Apply a filter if measurements are noisy and relative capacity changes are small. Depending on your application and environment conditions, test the influence of temperature and power supply voltage changes on measured values.

Once a detection threshold is established, it can be set during initialization with `touch_pad_config()` or at the runtime with `touch_pad_set_thresh()`.

In the next step, configure how interrupts are triggered. They can be triggered below or above the threshold, which is set with the function `touch_pad_set_trigger_mode()`.

Finally, configure and manage interrupt calls using the following functions:

- `touch_pad_isr_register()` / `touch_pad_isr_deregister()`
- `touch_pad_intr_enable()` / `touch_pad_intr_disable()`

When interrupts are operational, you can obtain the information from which particular pad an interrupt came by invoking `touch_pad_get_status()` and clear the pad status with `touch_pad_clear_status()`.

---

**注解:** Interrupts on touch detection operate on raw / unfiltered measurements checked against user established threshold and are implemented in hardware. Enabling the software filtering API (see *Filtering of Measurements*) does not affect this process.

---

**Wakeup from Sleep Mode** If touch pad interrupts are used to wake up the chip from a sleep mode, you can select a certain configuration of pads (SET1 or both SET1 and SET2) that should be touched to trigger the interrupt and cause the subsequent wakeup. To do so, use the function `touch_pad_set_trigger_source()`.

Configuration of required bit patterns of pads may be managed for each ‘SET’ with:

- `touch_pad_set_group_mask()` / `touch_pad_get_group_mask()`
- `touch_pad_clear_group_mask()`

## Application Examples

- Touch sensor read example: [peripherals/touch\\_pad\\_read](#).
- Touch sensor interrupt example: [peripherals/touch\\_pad\\_interrupt](#).

## API Reference

### Header File

- `driver/include/driver/touch_pad.h`

### Functions

`esp_err_t touch_pad_init()`

Initialize touch module.

**Note** The default FSM mode is ‘TOUCH\_FSM\_MODE\_SW’. If you want to use interrupt trigger mode, then set it using function ‘touch\_pad\_set\_fsm\_mode’ to ‘TOUCH\_FSM\_MODE\_TIMER’ after calling ‘touch\_pad\_init’.

#### Return

- ESP\_OK Success
- ESP\_FAIL Touch pad init error

`esp_err_t touch_pad_deinit()`

Un-install touch pad driver.

**Note** After this function is called, other touch functions are prohibited from being called.

#### Return

- ESP\_OK Success
- ESP\_FAIL Touch pad driver not initialized

*esp\_err\_t touch\_pad\_config(touch\_pad\_t touch\_num, uint16\_t threshold)*

Configure touch pad interrupt threshold.

**Note** 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.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG if argument wrong
- ESP\_FAIL if touch pad not initialized

**Parameters**

- touch\_num: touch pad index
- threshold: interrupt threshold,

*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.

**Note** This API requests hardware measurement once. If IIR filter mode is enabled, please use ‘touch\_pad\_read\_raw\_data’ interface instead.

**Return**

- 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

**Parameters**

- touch\_num: touch pad index
- touch\_value: pointer to accept touch sensor value

*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.

**Note** touch\_pad\_filter\_start has to be called before calling touch\_pad\_read\_filtered. This function can be called from ISR

**Return**

- 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

**Parameters**

- touch\_num: touch pad index
- touch\_value: pointer to accept touch sensor value

*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.

**Note** touch\_pad\_filter\_start has to be called before calling touch\_pad\_read\_raw\_data. This function can be called from ISR

**Return**

- 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

**Parameters**

- touch\_num: touch pad index
- touch\_value: pointer to accept touch sensor value

*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.

**Note** The ‘read\_cb’ callback is called in timer task in each filtering cycle.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG set error

**Parameters**

- read\_cb: Pointer to filtered callback function. If the argument passed in is NULL, the callback will stop.

*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.

**Return**

- ESP\_OK Success ;
- ESP\_ERR\_INVALID\_ARG GPIO error
- ESP\_ERR\_NO\_MEM No memory

**Parameters**

- fn: Pointer to ISR handler
- arg: Parameter for ISR

*esp\_err\_t* **touch\_pad\_isr\_deregister** (void (\*fn)) void \*

, void \*arg Deregister the handler previously registered using touch\_pad\_isr\_handler\_register.

**Return**

- ESP\_OK on success
- ESP\_ERR\_INVALID\_STATE if a handler matching both fn and arg isn't registered

**Parameters**

- 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)

*esp\_err\_t* **touch\_pad\_set\_meas\_time** (uint16\_t sleep\_cycle, uint16\_t meas\_cycle)

Set touch sensor measurement and sleep time.

**Return**

- ESP\_OK on success

**Parameters**

- sleep\_cycle: The touch sensor will sleep after each measurement. sleep\_cycle decide the interval between each measurement. t\_sleep = sleep\_cycle / (RTC\_SLOW\_CLK frequency). 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 / 8M, the maximum measure time is 0xffff / 8M = 8.19 ms

*esp\_err\_t* **touch\_pad\_get\_meas\_time** (uint16\_t \*sleep\_cycle, uint16\_t \*meas\_cycle)

Get touch sensor measurement and sleep time.

**Return**

- ESP\_OK on success

**Parameters**

- sleep\_cycle: Pointer to accept sleep cycle number
- meas\_cycle: Pointer to accept measurement cycle count.

*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 reference voltage, if the voltage gap between high and low reference voltage get less, the charging and discharging time would be faster, accordingly, the counter value would be larger. In the case of detecting very slight change of capacitance, we can narrow down the gap so as to increase the sensitivity. On the other hand, narrow voltage gap would also introduce more noise, but we can use a software filter to pre-process the counter value.

**Return**

- ESP\_OK on success



- ESP\_ERR\_INVALID\_ARG if argument is wrong

**Parameters**

- refh: the value of DREFH
- refl: the value of DREFL
- atten: the attenuation on DREFH

*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,.

**Return**

- ESP\_OK on success

**Parameters**

- refh: pointer to accept DREFH value
- refl: pointer to accept DREFL value
- atten: pointer to accept the attenuation on DREFH

*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, the counter value would be small. If the slope is set 7, which is the maximum value, the charging and discharging would be fast, accordingly, the counter value would be larger.

**Return**

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG if argument is wrong

**Parameters**

- touch\_num: touch pad index
- slope: touch pad charge/discharge speed
- opt: the initial voltage

*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.

**Return**

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG if argument is wrong

**Parameters**

- touch\_num: touch pad index
- slope: pointer to accept touch pad charge/discharge slope
- opt: pointer to accept the initial voltage

*esp\_err\_t touch\_pad\_io\_init(touch\_pad\_t touch\_num)*

Initialize touch pad GPIO.

**Return**

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG if argument is wrong

**Parameters**

- touch\_num: touch pad index

*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.

**Return**

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG if argument is wrong

**Parameters**

- mode: FSM mode

*esp\_err\_t touch\_pad\_get\_fsm\_mode(touch\_fsm\_mode\_t \*mode)*

Get touch sensor FSM mode.

**Return**

- ESP\_OK on success

**Parameters**

- mode: pointer to accept FSM mode

*esp\_err\_t* **touch\_pad\_sw\_start** ()

Trigger a touch sensor measurement, only support in SW mode of FSM.

**Return**

- ESP\_OK on success

*esp\_err\_t* **touch\_pad\_set\_thresh** (*touch\_pad\_t* touch\_num, uint16\_t threshold)

Set touch sensor interrupt threshold.

**Return**

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG if argument is wrong

**Parameters**

- 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\_err\_t* **touch\_pad\_get\_thresh** (*touch\_pad\_t* touch\_num, uint16\_t \*threshold)

Get touch sensor interrupt threshold.

**Return**

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG if argument is wrong

**Parameters**

- touch\_num: touch pad index
- threshold: pointer to accept threshold

*esp\_err\_t* **touch\_pad\_set\_trigger\_mode** (*touch\_trigger\_mode\_t* mode)

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.

**Return**

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG if argument is wrong

**Parameters**

- mode: touch sensor interrupt trigger mode

*esp\_err\_t* **touch\_pad\_get\_trigger\_mode** (*touch\_trigger\_mode\_t* \*mode)

Get touch sensor interrupt trigger mode.

**Return**

- ESP\_OK on success

**Parameters**

- mode: pointer to accept touch sensor interrupt trigger mode

*esp\_err\_t* **touch\_pad\_set\_trigger\_source** (*touch\_trigger\_src\_t* src)

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.

**Return**

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG if argument is wrong

**Parameters**

- src: touch sensor interrupt trigger source

*esp\_err\_t* **touch\_pad\_get\_trigger\_source** (*touch\_trigger\_src\_t* \*src)

Get touch sensor interrupt trigger source.

**Return**

- ESP\_OK on success

**Parameters**

- `src`: pointer to accept touch sensor interrupt trigger source

*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.

**Return**

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG if argument is wrong

**Parameters**

- *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

*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.

**Return**

- ESP\_OK on success

**Parameters**

- *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

*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.

**Return**

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG if argument is wrong

**Parameters**

- *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

*esp\_err\_t* **touch\_pad\_clear\_status** ()

To clear the touch status register, usually use this function in touch ISR to clear status.

**Return**

- ESP\_OK on success

uint32\_t **touch\_pad\_get\_status** ()

Get the touch sensor status, usually used in ISR to decide which pads are ‘touched’.

**Return**

- touch status

*esp\_err\_t* **touch\_pad\_intr\_enable** ()

To enable touch pad interrupt.

**Return**

- ESP\_OK on success

*esp\_err\_t* **touch\_pad\_intr\_disable** ()

To disable touch pad interrupt.

**Return**

- 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

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_STATE driver state error
- ESP\_ERR\_INVALID\_ARG parameter error

**Parameters**

- *new\_period\_ms*: filter period, in ms

*esp\_err\_t* **touch\_pad\_get\_filter\_period** (uint32\_t \**p\_period\_ms*)

get touch pad filter calibration period, in ms Need to call `touch_pad_filter_start` before all touch filter APIs

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_STATE driver state error
- ESP\_ERR\_INVALID\_ARG parameter error

**Parameters**

- *p\_period\_ms*: pointer to accept period

*esp\_err\_t* **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

**Note** This filter uses FreeRTOS timer, which is dispatched from a task with priority 1 by default on CPU 0. So if some application task with higher priority takes a lot of CPU0 time, then the quality of data obtained from this filter will be affected. You can adjust FreeRTOS timer task priority in `menuconfig`.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG parameter error
- ESP\_ERR\_NO\_MEM No memory for driver
- ESP\_ERR\_INVALID\_STATE driver state error

**Parameters**

- *filter\_period\_ms*: filter calibration period, in ms

*esp\_err\_t* **touch\_pad\_filter\_stop** ()

stop touch pad filter function Need to call `touch_pad_filter_start` before all touch filter APIs

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_STATE driver state error

*esp\_err\_t* **touch\_pad\_filter\_delete** ()

delete touch pad filter driver and release the memory Need to call `touch_pad_filter_start` before all touch filter APIs

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_STATE driver state error

*esp\_err\_t* **touch\_pad\_get\_wakeup\_status** (*touch\_pad\_t* \**pad\_num*)

Get the touch pad which caused wakeup from sleep.

**Return**

- ESP\_OK Success
- ESP\_FAIL get status err

**Parameters**

- *pad\_num*: pointer to touch pad which caused wakeup

## Macros

**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\_PAD\_MEASURE\_WAIT\_DEFAULT**

The timer frequency is 8Mhz, the max value is 0xff

**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

**TOUCH\_PAD\_BIT\_MASK\_MAX****Type Definitions**

```
typedef intr_handle_t touch_isr_handle_t
```

```
typedef void (*filter_cb_t) (uint16_t *raw_value, uint16_t *filtered_value)
```

Callback function that is called after each IIR filter calculation.

**Note** This callback is called in timer task in each filtering cycle.

**Note** This callback should not be blocked.

**Parameters**

- `raw_value`: The latest raw data(touch sensor counter value) that points to all channels(`raw_value[0..TOUCH_PAD_MAX-1]`).
- `filtered_value`: The latest IIR filtered data(calculated from raw data) that points to all channels(`filtered_value[0..TOUCH_PAD_MAX-1]`).

**Enumerations**

```
enum touch_pad_t
```

*Values:*

```
TOUCH_PAD_NUM0 = 0
```

Touch pad channel 0 is GPIO4

```
TOUCH_PAD_NUM1
```

Touch pad channel 1 is GPIO0

```
TOUCH_PAD_NUM2
```

Touch pad channel 2 is GPIO2

```
TOUCH_PAD_NUM3
```

Touch pad channel 3 is GPIO15

```
TOUCH_PAD_NUM4
```

Touch pad channel 4 is GPIO13

```
TOUCH_PAD_NUM5
```

Touch pad channel 5 is GPIO12

```
TOUCH_PAD_NUM6
```

Touch pad channel 6 is GPIO14

```
TOUCH_PAD_NUM7
```

Touch pad channel 7 is GPIO27

```
TOUCH_PAD_NUM8
```

Touch pad channel 8 is GPIO33

```
TOUCH_PAD_NUM9
```

Touch pad channel 9 is GPIO32

```
TOUCH_PAD_MAX
```

**enum touch\_high\_volt\_t***Values:*

- TOUCH\_HVOLT\_KEEP** = -1  
Touch sensor high reference voltage, no change
- TOUCH\_HVOLT\_2V4** = 0  
Touch sensor high reference voltage, 2.4V
- TOUCH\_HVOLT\_2V5**  
Touch sensor high reference voltage, 2.5V
- TOUCH\_HVOLT\_2V6**  
Touch sensor high reference voltage, 2.6V
- TOUCH\_HVOLT\_2V7**  
Touch sensor high reference voltage, 2.7V
- TOUCH\_HVOLT\_MAX**

**enum touch\_low\_volt\_t***Values:*

- TOUCH\_LVOLT\_KEEP** = -1  
Touch sensor low reference voltage, no change
- TOUCH\_LVOLT\_0V5** = 0  
Touch sensor low reference voltage, 0.5V
- TOUCH\_LVOLT\_0V6**  
Touch sensor low reference voltage, 0.6V
- TOUCH\_LVOLT\_0V7**  
Touch sensor low reference voltage, 0.7V
- TOUCH\_LVOLT\_0V8**  
Touch sensor low reference voltage, 0.8V
- TOUCH\_LVOLT\_MAX**

**enum touch\_volt\_atten\_t***Values:*

- TOUCH\_HVOLT\_ATTEN\_KEEP** = -1  
Touch sensor high reference voltage attenuation, no change
- TOUCH\_HVOLT\_ATTEN\_1V5** = 0  
Touch sensor high reference voltage attenuation, 1.5V attenuation
- TOUCH\_HVOLT\_ATTEN\_1V**  
Touch sensor high reference voltage attenuation, 1.0V attenuation
- TOUCH\_HVOLT\_ATTEN\_0V5**  
Touch sensor high reference voltage attenuation, 0.5V attenuation
- TOUCH\_HVOLT\_ATTEN\_0V**  
Touch sensor high reference voltage attenuation, 0V attenuation
- TOUCH\_HVOLT\_ATTEN\_MAX**

**enum touch\_cnt\_slope\_t***Values:*

- TOUCH\_PAD\_SLOPE\_0** = 0  
Touch sensor charge / discharge speed, always zero
- TOUCH\_PAD\_SLOPE\_1** = 1  
Touch sensor charge / discharge speed, slowest

**TOUCH\_PAD\_SLOPE\_2 = 2**  
Touch sensor charge / discharge speed

**TOUCH\_PAD\_SLOPE\_3 = 3**  
Touch sensor charge / discharge speed

**TOUCH\_PAD\_SLOPE\_4 = 4**  
Touch sensor charge / discharge speed

**TOUCH\_PAD\_SLOPE\_5 = 5**  
Touch sensor charge / discharge speed

**TOUCH\_PAD\_SLOPE\_6 = 6**  
Touch sensor charge / discharge speed

**TOUCH\_PAD\_SLOPE\_7 = 7**  
Touch sensor charge / discharge speed, fast

**TOUCH\_PAD\_SLOPE\_MAX**

**enum touch\_trigger\_mode\_t**

*Values:*

**TOUCH\_TRIGGER\_BELOW = 0**  
Touch interrupt will happen if counter value is less than threshold.

**TOUCH\_TRIGGER\_ABOVE = 1**  
Touch interrupt will happen if counter value is larger than threshold.

**TOUCH\_TRIGGER\_MAX**

**enum touch\_trigger\_src\_t**

*Values:*

**TOUCH\_TRIGGER\_SOURCE\_BOTH = 0**  
wakeup interrupt is generated if both SET1 and SET2 are “touched”

**TOUCH\_TRIGGER\_SOURCE\_SET1 = 1**  
wakeup interrupt is generated if SET1 is “touched”

**TOUCH\_TRIGGER\_SOURCE\_MAX**

**enum touch\_tie\_opt\_t**

*Values:*

**TOUCH\_PAD\_TIE\_OPT\_LOW = 0**  
Initial level of charging voltage, low level

**TOUCH\_PAD\_TIE\_OPT\_HIGH = 1**  
Initial level of charging voltage, high level

**TOUCH\_PAD\_TIE\_OPT\_MAX**

**enum touch\_fsm\_mode\_t**

*Values:*

**TOUCH\_FSM\_MODE\_TIMER = 0**  
To start touch FSM by timer

**TOUCH\_FSM\_MODE\_SW**  
To start touch FSM by software trigger

**TOUCH\_FSM\_MODE\_MAX**

**GPIO Lookup Macros** Some useful macros can be used to specified the GPIO number of a touch pad channel, or vice versa. e.g.

1. **TOUCH\_PAD\_NUM5\_GPIO\_NUM** is the GPIO number of channel 5 (12);
2. **TOUCH\_PAD\_GPIO4\_CHANNEL** is the channel number of GPIO 4 (channel 0).

## Header File

- [soc/esp32/include/soc/touch\\_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  
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

## 2.3.19 UART

### Overview

A Universal Asynchronous Receiver/Transmitter (UART) is a component known to handle the timing requirements for a variety of widely-adapted interfaces (RS232, RS485, RS422, ...). A UART provides a widely adopted and cheap method to realize full-duplex or half-duplex data exchange among different devices.

There are three UART controllers available on the ESP32 chip. They are compatible with UART-enabled devices from various manufacturers. All UART controllers integrated in the ESP32 feature an identical set of registers for ease of programming and flexibility. In this documentation, these controllers are referred to as UART0, UART1, and UART2.

### Functional Overview

The following overview describes functions and data types used to establish communication between ESP32 and some other UART device. The overview reflects a typical workflow when programming ESP32's UART driver and is broken down into the following sections:

1. *Setting Communication Parameters* - baud rate, data bits, stop bits, etc,
2. *Setting Communication Pins* - pins the other UART is connected to
3. *Driver Installation* - allocate ESP32's resources for the UART driver



4. *Running UART Communication* - send / receive the data
5. *Using Interrupts* - trigger interrupts on specific communication events
6. *Deleting Driver* - release ESP32's resources, if UART communication is not required anymore

The minimum to make the UART working is to complete the first four steps, the last two steps are optional.

The driver is identified by `uart_port_t`, that corresponds to one of the tree UART controllers. Such identification is present in all the following function calls.

**Setting Communication Parameters** There are two ways to set the communications parameters for UART. One is to do it in one shot by calling `uart_param_config()` provided with configuration parameters in `uart_config_t` structure.

The alternate way is to configure specific parameters individually by calling dedicated functions:

- Baud rate - `uart_set_baudrate()`
- Number of transmitted bits - `uart_set_word_length()` selected out of `uart_word_length_t`
- Parity control - `uart_set_parity()` selected out of `uart_parity_t`
- Number of stop bits - `uart_set_stop_bits()` selected out of `uart_stop_bits_t`
- Hardware flow control mode - `uart_set_hw_flow_ctrl()` selected out of `uart_hw_flowcontrol_t`
- Communication mode - `uart_set_mode()` selected out of `uart_mode_t`

Configuration example:

```
const int 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));
```

All the above functions have a `_get_` equivalent to retrieve the current setting, e.g. `uart_get_baudrate()`.

**Setting Communication Pins** In next step, after configuring communication parameters, we are setting physical GPIO pin numbers the other UART will be connected to. This is done in a single step by calling function `uart_set_pin()` and providing it with GPIO numbers, that driver should use for the Tx, Rx, RTS and CTS signals.

Instead of GPIO pin number we can enter a macro `UART_PIN_NO_CHANGE` and the currently allocated pin will not be changed. The same macro should be entered if certain pin will not be used.

```
// Set UART pins(TX: IO16 (UART2 default), RX: IO17 (UART2 default), RTS: IO18, ↵
↵CTS: IO19)
ESP_ERROR_CHECK(uart_set_pin(UART_NUM_2, UART_PIN_NO_CHANGE, UART_PIN_NO_CHANGE, ↵
↵18, 19));
```

**Driver Installation** Once configuration of driver is complete, we can install it by calling `uart_driver_install()`. As result several resources required by the UART will be allocated. The type / size of resources are specified as function call parameters and concern:

- size of the send buffer
- size of the receive buffer
- the event queue handle and size
- flags to allocate an interrupt

Example:

```
// 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));
```

If all above steps have been complete, we are ready to connect the other UART device and check the communication.

**Running UART Communication** The processes of serial communication are under control of UART's hardware FSM. The data to be sent should be put into Tx FIFO buffer, FSM will serialize them and sent out. A similar process, but in reverse order, is done to receive the data. Incoming serial stream is processed by FSM and moved to the Rx FIFO buffer. Therefore the task of API's communication functions is limited to writing and reading the data to / from the respective buffer. This is reflected in some function names, e.g.: `uart_write_bytes()` to transmit the data out, or `uart_read_bytes()` to read the incoming data.

**Transmitting** The basic API function to write the data to Tx FIFO buffer is `uart_tx_chars()`. If the buffer contains not sent characters, this function will write what fits into the empty space and exit reporting the number of bytes actually written.

There is a 'companion' function `uart_wait_tx_done()` that waits until all the data are transmitted out and the Tx FIFO is empty.

```
// Wait for packet to be sent
const int uart_num = UART_NUM_2;
ESP_ERROR_CHECK(uart_wait_tx_done(uart_num, 100)); // wait timeout is 100 RTOS_
↳ticks (TickType_t)
```

An easier to work with function is `uart_write_bytes()`. It sets up an intermediate ring buffer and exits after copying the data to this buffer. When there is an empty space in the FIFO, the data are moved from the ring buffer to the FIFO in the background by an ISR. The code below demonstrates using of this function.

```
// 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));
```

There is a similar function as above that adds a serial break signal after sending the data - `uart_write_bytes_with_break()`. The 'serial break signal' means holding TX line low for period longer than one data frame

```
// Write data to UART, end with a break signal.
uart_write_bytes_with_break(uart_num, "test break\n", strlen("test break\n"), 100);
```

**Receiving** To retrieve the data received by UART and saved in Rx FIFO, use function `uart_read_bytes()`. You can check in advance what is the number of bytes available in Rx FIFO by calling `uart_get_buffered_data_len()`. Below is the example of using this function:

```
// Read data from UART.
const int 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 Rx FIFO is not required and should be discarded, call `uart_flush()`.

**Software Flow Control** When the hardware flow control is disabled, then use `uart_set_rts()` and `uart_set_dtr()` to manually set the levels of the RTS and DTR signals.

**Communication Mode Selection** The UART controller supports set of communication modes. The selection of mode can be performed using function `uart_set_mode()`. Once the specific mode is selected the UART driver will handle behavior of external peripheral according to mode. As an example it can control RS485 driver chip over 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 nineteen interrupts reported on specific states of UART or on detected errors. The full list of available interrupts is described in [ESP32 Technical Reference Manual \(PDF\)](#). To enable specific interrupts call `uart_enable_intr_mask()`, to disable call `uart_disable_intr_mask()`. The mask of all interrupts is available as `UART_INTR_MASK`. Registration of an handler to service interrupts is done with `uart_isr_register()`, freeing the handler with `uart_isr_free()`. To clear the interrupt status bits once the handler is called use `uart_clear_intr_status()`.

The API provides a convenient way to handle specific interrupts discussed above by wrapping them into dedicated functions:

- **Event detection** - there are several events defined in `uart_event_type_t` that may be reported to user application using FreeRTOS queue functionality. You can enable this functionality when calling `uart_driver_install()` described in [Driver Installation](#). Example how to use it is covered in [peripherals/uart\\_events](#).
- **FIFO space threshold or transmission timeout reached** - the interrupts on TX or Rx FIFO buffer being filled with specific number of characters or on a timeout of sending or receiving data. To use these interrupts, first configure respective threshold values of the buffer length and the timeout by entering them in `uart_intr_config_t` structure and calling `uart_intr_config()`. Then enable interrupts with functions `uart_enable_rx_intr()` and `uart_enable_tx_intr()`. To disable these interrupts there are corresponding functions `uart_disable_rx_intr()` or `uart_disable_tx_intr()`.
- **Pattern detection** - an interrupt triggered on detecting a ‘pattern’ of the same character being sent number of times. The functions that allow to configure, enable and disable this interrupt are `uart_enable_pattern_det_intr()` and `uart_disable_pattern_det_intr()`.

**Macros** The API provides several macros to define configuration parameters, e.g. `UART_FIFO_LEN` to define the length of the hardware FIFO buffers, `UART_BITRATE_MAX` that gives the maximum baud rate supported by UART, etc.

**Deleting Driver** If communication is established with `uart_driver_install()` for some specific period of time and then not required, the driver may be removed to free allocated resources by calling `uart_driver_delete()`.

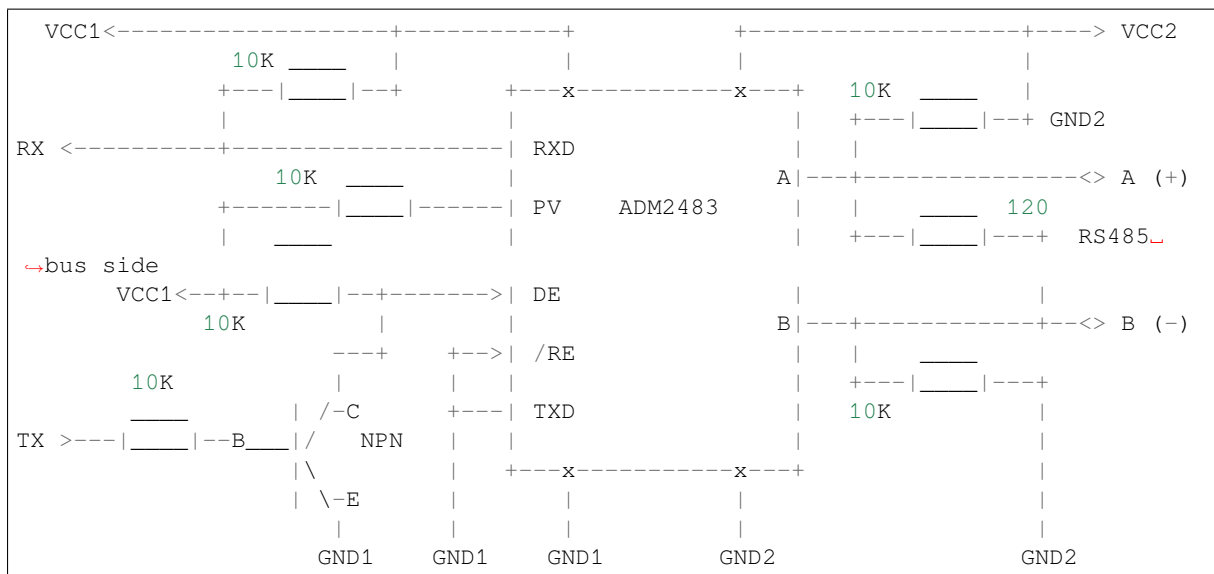
## Overview of RS485 specific communication options

**注解:** Here and below the notation `UART_REGISTER.UART_OPTION_BIT` will be used to describe register options of UART. See the [ESP32 Technical Reference Manual](#) for more information.

- `UART_RS485_CONF_REG.UART_RS485_EN = 1`, enable RS485 communication mode support.
- `UART_RS485_CONF_REG.UART_RS485TX_RX_EN`, transmitter’s output signal loop back to the receiver’s input signal when this bit is set.
- `UART_RS485_CONF_REG.UART_RS485RXBY_TX_EN`, when bit is set the transmitter should send data when its receiver is busy (remove collisions automatically by hardware).

The on chip RS485 UART hardware is able to detect signal collisions during transmission of datagram and generate an interrupt `UART_RS485_CLASH_INT` when it is enabled. The term collision means that during transmission of datagram the received data is different with what has been transmitted out or framing errors exist. Data collisions are usually associated with the presence of other active devices on the bus or due to bus errors. The collision detection feature



**The circuit C: auto switching of transmitter/receiver**

This galvanic isolated circuit does not require RTS pin control by software application or driver because it controls transceiver direction automatically. However it requires removing null bytes during transmission by setting `UART_RS485_CONF_REG.UART_RS485RXBY_TX_EN = 1`, `UART_RS485_CONF_REG.UART_RS485TX_RX_EN = 0`. This variant can work in any RS485 UART mode or even in `UART_MODE_UART`.

**Application Examples**

Configure UART settings and install UART driver to read/write using UART1 interface: [peripherals/uart/uart\\_echo](#).

Demonstration of how to report various communication events and how to use pattern detection interrupts: [peripherals/uart/uart\\_events](#).

Transmitting and receiving with the same UART in two separate FreeRTOS tasks: [peripherals/uart/uart\\_async\\_rxtxtasks](#).

Using synchronous I/O multiplexing for UART file descriptors: [peripherals/uart/uart\\_select](#).

Setup of UART driver to communicate over RS485 interface in half-duplex mode: [peripherals/uart/uart\\_echo\\_rs485](#). This example is similar to `uart_echo` but provide communication through RS485 interface chip connected to ESP32 pins.

Demonstration of how to get GPS information by parsing NMEA0183 statements received from GPS via UART peripheral: [peripherals/uart/nmea0183\\_parser](#).

**API Reference****Header File**

- [driver/include/driver/uart.h](#)

**Functions**

bool `uart_is_driver_installed`(*uart\_port\_t* *uart\_num*)

Checks whether the driver is installed or not.

**Return**

- true driver is installed
- false driver is not installed

**Parameters**

- `uart_num`: UART port number, the max port number is (`UART_NUM_MAX - 1`).

*esp\_err\_t* **uart\_set\_word\_length** (*uart\_port\_t* *uart\_num*, *uart\_word\_length\_t* *data\_bit*)

Set UART data bits.

#### Return

- `ESP_OK` Success
- `ESP_FAIL` Parameter error

#### Parameters

- `uart_num`: `UART_NUM_0`, `UART_NUM_1` or `UART_NUM_2`
- `data_bit`: UART data bits

*esp\_err\_t* **uart\_get\_word\_length** (*uart\_port\_t* *uart\_num*, *uart\_word\_length\_t* \**data\_bit*)

Get UART data bits.

#### Return

- `ESP_FAIL` Parameter error
- `ESP_OK` Success, result will be put in (\**data\_bit*)

#### Parameters

- `uart_num`: `UART_NUM_0`, `UART_NUM_1` or `UART_NUM_2`
- `data_bit`: Pointer to accept value of UART data bits.

*esp\_err\_t* **uart\_set\_stop\_bits** (*uart\_port\_t* *uart\_num*, *uart\_stop\_bits\_t* *stop\_bits*)

Set UART stop bits.

#### Return

- `ESP_OK` Success
- `ESP_FAIL` Fail

#### Parameters

- `uart_num`: `UART_NUM_0`, `UART_NUM_1` or `UART_NUM_2`
- `stop_bits`: UART stop bits

*esp\_err\_t* **uart\_get\_stop\_bits** (*uart\_port\_t* *uart\_num*, *uart\_stop\_bits\_t* \**stop\_bits*)

Get UART stop bits.

#### Return

- `ESP_FAIL` Parameter error
- `ESP_OK` Success, result will be put in (\**stop\_bit*)

#### Parameters

- `uart_num`: `UART_NUM_0`, `UART_NUM_1` or `UART_NUM_2`
- `stop_bits`: Pointer to accept value of UART stop bits.

*esp\_err\_t* **uart\_set\_parity** (*uart\_port\_t* *uart\_num*, *uart\_parity\_t* *parity\_mode*)

Set UART parity mode.

#### Return

- `ESP_FAIL` Parameter error
- `ESP_OK` Success

#### Parameters

- `uart_num`: `UART_NUM_0`, `UART_NUM_1` or `UART_NUM_2`
- `parity_mode`: the enum of uart parity configuration

*esp\_err\_t* **uart\_get\_parity** (*uart\_port\_t* *uart\_num*, *uart\_parity\_t* \**parity\_mode*)

Get UART parity mode.

#### Return

- `ESP_FAIL` Parameter error
- `ESP_OK` Success, result will be put in (\**parity\_mode*)

#### Parameters

- `uart_num`: `UART_NUM_0`, `UART_NUM_1` or `UART_NUM_2`
- `parity_mode`: Pointer to accept value of UART parity mode.

*esp\_err\_t* **uart\_set\_baudrate** (*uart\_port\_t* *uart\_num*, *uint32\_t* *baudrate*)

Set UART baud rate.

#### Return

- ESP\_FAIL Parameter error
- ESP\_OK Success

**Parameters**

- `uart_num`: UART\_NUM\_0, UART\_NUM\_1 or UART\_NUM\_2
- `baudrate`: UART baud rate.

*esp\_err\_t* **uart\_get\_baudrate** (*uart\_port\_t* `uart_num`, *uint32\_t* `*baudrate`)

Get UART baud rate.

**Return**

- ESP\_FAIL Parameter error
- ESP\_OK Success, result will be put in (`*baudrate`)

**Parameters**

- `uart_num`: UART\_NUM\_0, UART\_NUM\_1 or UART\_NUM\_2
- `baudrate`: Pointer to accept value of UART baud rate

*esp\_err\_t* **uart\_set\_line\_inverse** (*uart\_port\_t* `uart_num`, *uint32\_t* `inverse_mask`)

Set UART line inverse mode.

**Return**

- ESP\_OK Success
- ESP\_FAIL Parameter error

**Parameters**

- `uart_num`: UART\_NUM\_0, UART\_NUM\_1 or UART\_NUM\_2
- `inverse_mask`: Choose the wires that need to be inverted. `inverse_mask` should be chosen from UART\_INVERSE\_RXD / UART\_INVERSE\_TXD / UART\_INVERSE\_RTS / UART\_INVERSE\_CTS, combined with OR operation.

*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.

**Return**

- ESP\_OK Success
- ESP\_FAIL Parameter error

**Parameters**

- `uart_num`: UART\_NUM\_0, UART\_NUM\_1 or UART\_NUM\_2
- `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.

*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.

**Return**

- ESP\_OK Success
- ESP\_FAIL Parameter error

**Parameters**

- `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

*esp\_err\_t* **uart\_get\_hw\_flow\_ctrl** (*uart\_port\_t* `uart_num`, *uart\_hw\_flowcontrol\_t* `*flow_ctrl`)

Get hardware flow control mode.

**Return**

- ESP\_FAIL Parameter error
- ESP\_OK Success, result will be put in (`*flow_ctrl`)

**Parameters**

- `uart_num`: UART\_NUM\_0, UART\_NUM\_1 or UART\_NUM\_2
- `flow_ctrl`: Option for different flow control mode.

*esp\_err\_t* **uart\_clear\_intr\_status** (*uart\_port\_t* *uart\_num*, *uint32\_t* *clr\_mask*)

Clear UART interrupt status.

**Return**

- ESP\_OK Success
- ESP\_FAIL Parameter error

**Parameters**

- *uart\_num*: UART\_NUM\_0, UART\_NUM\_1 or UART\_NUM\_2
- *clr\_mask*: Bit mask of the interrupt status to be cleared. The bit mask should be composed from the fields of register UART\_INT\_CLR\_REG.

*esp\_err\_t* **uart\_enable\_intr\_mask** (*uart\_port\_t* *uart\_num*, *uint32\_t* *enable\_mask*)

Set UART interrupt enable.

**Return**

- ESP\_OK Success
- ESP\_FAIL Parameter error

**Parameters**

- *uart\_num*: UART\_NUM\_0, UART\_NUM\_1 or UART\_NUM\_2
- *enable\_mask*: Bit mask of the enable bits. The bit mask should be composed from the fields of register UART\_INT\_ENA\_REG.

*esp\_err\_t* **uart\_disable\_intr\_mask** (*uart\_port\_t* *uart\_num*, *uint32\_t* *disable\_mask*)

Clear UART interrupt enable bits.

**Return**

- ESP\_OK Success
- ESP\_FAIL Parameter error

**Parameters**

- *uart\_num*: UART\_NUM\_0, UART\_NUM\_1 or UART\_NUM\_2
- *disable\_mask*: Bit mask of the disable bits. The bit mask should be composed from the fields of register UART\_INT\_ENA\_REG.

*esp\_err\_t* **uart\_enable\_rx\_intr** (*uart\_port\_t* *uart\_num*)

Enable UART RX interrupt (RX\_FULL & RX\_TIMEOUT INTERRUPT)

**Return**

- ESP\_OK Success
- ESP\_FAIL Parameter error

**Parameters**

- *uart\_num*: UART\_NUM\_0, UART\_NUM\_1 or UART\_NUM\_2

*esp\_err\_t* **uart\_disable\_rx\_intr** (*uart\_port\_t* *uart\_num*)

Disable UART RX interrupt (RX\_FULL & RX\_TIMEOUT INTERRUPT)

**Return**

- ESP\_OK Success
- ESP\_FAIL Parameter error

**Parameters**

- *uart\_num*: UART\_NUM\_0, UART\_NUM\_1 or UART\_NUM\_2

*esp\_err\_t* **uart\_disable\_tx\_intr** (*uart\_port\_t* *uart\_num*)

Disable UART TX interrupt (TX\_FULL & TX\_TIMEOUT INTERRUPT)

**Return**

- ESP\_OK Success
- ESP\_FAIL Parameter error

**Parameters**

- *uart\_num*: UART\_NUM\_0, UART\_NUM\_1 or UART\_NUM\_2

*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)

**Return**

- ESP\_OK Success



- ESP\_FAIL Parameter error

**Parameters**

- `uart_num`: UART\_NUM\_0, UART\_NUM\_1 or UART\_NUM\_2
- `enable`: 1: enable; 0: disable
- `thresh`: Threshold of TX interrupt, 0 ~ UART\_FIFO\_LEN

*esp\_err\_t* **uart\_isr\_register** (*uart\_port\_t* `uart_num`, void (\**fn*)) void \*  
 , void \**arg*, int *intr\_alloc\_flags*, *uart\_isr\_handle\_t* \**handle* Register UART interrupt handler (ISR).

**Note** UART ISR handler will be attached to the same CPU core that this function is running on.

**Return**

- ESP\_OK Success
- ESP\_FAIL Parameter error

**Parameters**

- `uart_num`: UART\_NUM\_0, UART\_NUM\_1 or UART\_NUM\_2
- `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.

*esp\_err\_t* **uart\_isr\_free** (*uart\_port\_t* `uart_num`)

Free UART interrupt handler registered by `uart_isr_register`. Must be called on the same core as `uart_isr_register` was called.

**Return**

- ESP\_OK Success
- ESP\_FAIL Parameter error

**Parameters**

- `uart_num`: UART\_NUM\_0, UART\_NUM\_1 or UART\_NUM\_2

*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`)

Set UART pin number.

**Note** Internal signal can be output to multiple GPIO pads. Only one GPIO pad can connect with input signal.

**Note** Instead of GPIO number a macro 'UART\_PIN\_NO\_CHANGE' may be provided to keep the currently allocated pin.

**Return**

- ESP\_OK Success
- ESP\_FAIL Parameter error

**Parameters**

- `uart_num`: UART\_NUM\_0, UART\_NUM\_1 or UART\_NUM\_2
- `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.

*esp\_err\_t* **uart\_set\_rts** (*uart\_port\_t* `uart_num`, int `level`)

Manually set the UART RTS pin level.

**Note** UART must be configured with hardware flow control disabled.

**Return**

- ESP\_OK Success
- ESP\_FAIL Parameter error

**Parameters**

- `uart_num`: UART\_NUM\_0, UART\_NUM\_1 or UART\_NUM\_2
- `level`: 1: RTS output low (active); 0: RTS output high (block)

*esp\_err\_t* **uart\_set\_dtr** (*uart\_port\_t* `uart_num`, int `level`)

Manually set the UART DTR pin level.

**Return**

- ESP\_OK Success
- ESP\_FAIL Parameter error

**Parameters**

- `uart_num`: UART\_NUM\_0, UART\_NUM\_1 or UART\_NUM\_2
- `level`: 1: DTR output low; 0: DTR output high

*esp\_err\_t* **uart\_set\_tx\_idle\_num** (*uart\_port\_t* `uart_num`, *uint16\_t* `idle_num`)

Set UART idle interval after tx FIFO is empty.

**Return**

- ESP\_OK Success
- ESP\_FAIL Parameter error

**Parameters**

- `uart_num`: UART\_NUM\_0, UART\_NUM\_1 or UART\_NUM\_2
- `idle_num`: idle interval after tx FIFO is empty(unit: the time it takes to send one bit under current baudrate)

*esp\_err\_t* **uart\_param\_config** (*uart\_port\_t* `uart_num`, **const** *uart\_config\_t* \*`uart_config`)

Set UART configuration parameters.

**Return**

- ESP\_OK Success
- ESP\_FAIL Parameter error

**Parameters**

- `uart_num`: UART\_NUM\_0, UART\_NUM\_1 or UART\_NUM\_2
- `uart_config`: UART parameter settings

*esp\_err\_t* **uart\_intr\_config** (*uart\_port\_t* `uart_num`, **const** *uart\_intr\_config\_t* \*`intr_conf`)

Configure UART interrupts.

**Return**

- ESP\_OK Success
- ESP\_FAIL Parameter error

**Parameters**

- `uart_num`: UART\_NUM\_0, UART\_NUM\_1 or UART\_NUM\_2
- `intr_conf`: UART interrupt settings

*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.

UART ISR handler will be attached to the same CPU core that this function is running on.

**Note** `Rx_buffer_size` should be greater than UART\_FIFO\_LEN. `Tx_buffer_size` should be either zero or greater than UART\_FIFO\_LEN.

**Return**

- ESP\_OK Success
- ESP\_FAIL Parameter error

**Parameters**

- `uart_num`: UART\_NUM\_0, UART\_NUM\_1 or UART\_NUM\_2
- `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 (ORred) 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)

*esp\_err\_t* **uart\_driver\_delete** (*uart\_port\_t* `uart_num`)

Uninstall UART driver.

**Return**

- ESP\_OK Success
- ESP\_FAIL Parameter error

**Parameters**

- `uart_num`: UART\_NUM\_0, UART\_NUM\_1 or UART\_NUM\_2

`esp_err_t uart_wait_tx_done (uart_port_t uart_num, TickType_t ticks_to_wait)`

Wait until UART TX FIFO is empty.

**Return**

- ESP\_OK Success
- ESP\_FAIL Parameter error
- ESP\_ERR\_TIMEOUT Timeout

**Parameters**

- `uart_num`: UART\_NUM\_0, UART\_NUM\_1 or UART\_NUM\_2
- `ticks_to_wait`: Timeout, count in RTOS ticks

`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.

**Note** This function should only be used when UART TX buffer is not enabled.

**Return**

- (-1) Parameter error
- OTHERS (>=0) The number of bytes pushed to the TX FIFO

**Parameters**

- `uart_num`: UART\_NUM\_0, UART\_NUM\_1 or UART\_NUM\_2
- `buffer`: data buffer address
- `len`: data length to send

`int uart_write_bytes (uart_port_t uart_num, const char *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.

**Return**

- (-1) Parameter error
- OTHERS (>=0) The number of bytes pushed to the TX FIFO

**Parameters**

- `uart_num`: UART\_NUM\_0, UART\_NUM\_1 or UART\_NUM\_2
- `src`: data buffer address
- `size`: data length to send

`int uart_write_bytes_with_break (uart_port_t uart_num, const char *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.

**Return**

- (-1) Parameter error
- OTHERS (>=0) The number of bytes pushed to the TX FIFO

**Parameters**

- `uart_num`: UART\_NUM\_0, UART\_NUM\_1 or UART\_NUM\_2
- `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)

int `uart_read_bytes` (*uart\_port\_t* `uart_num`, uint8\_t \*`buf`, uint32\_t `length`, TickType\_t `ticks_to_wait`)  
UART read bytes from UART buffer.

**Return**

- (-1) Error
- OTHERS (>=0) The number of bytes read from UART FIFO

**Parameters**

- `uart_num`: UART\_NUM\_0, UART\_NUM\_1 or UART\_NUM\_2
- `buf`: pointer to the buffer.
- `length`: data length
- `ticks_to_wait`: sTimeout, count in RTOS ticks

*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.

**Note** 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.

**Return**

- ESP\_OK Success
- ESP\_FAIL Parameter error

**Parameters**

- `uart_num`: UART\_NUM\_0, UART\_NUM\_1 or UART\_NUM\_2

*esp\_err\_t* `uart_flush_input` (*uart\_port\_t* `uart_num`)

Clear input buffer, discard all the data is in the ring-buffer.

**Note** In order to send all the data in tx FIFO, we can use `uart_wait_tx_done` function.

**Return**

- ESP\_OK Success
- ESP\_FAIL Parameter error

**Parameters**

- `uart_num`: UART\_NUM\_0, UART\_NUM\_1 or UART\_NUM\_2

*esp\_err\_t* `uart_get_buffered_data_len` (*uart\_port\_t* `uart_num`, size\_t \*`size`)

UART get RX ring buffer cached data length.

**Return**

- ESP\_OK Success
- ESP\_FAIL Parameter error

**Parameters**

- `uart_num`: UART port number.
- `size`: Pointer of size\_t to accept cached data length

*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.

**Return**

- ESP\_OK Success
- ESP\_FAIL Parameter error

**Parameters**

- `uart_num`: UART port number.

*esp\_err\_t* `uart_enable_pattern_det_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.

**Return**

- ESP\_OK Success
- ESP\_FAIL Parameter error

**Parameters**

- `uart_num`: UART port number.
- `pattern_chr`: character of the pattern
- `chr_num`: number of the character, 8bit value.
- `chr_tout`: timeout of the interval between each pattern characters, 24bit value, unit is APB (80Mhz) clock cycle. When the duration is less than this value, it will not take this data as `at_cmd` char
- `post_idle`: idle time after the last pattern character, 24bit value, unit is APB (80Mhz) clock cycle. 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, 24bit value, unit is APB (80Mhz) clock cycle. When the duration is less than this value, it will not take this data as the first `at_cmd` char

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.

**Note** 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.

**Return**

- (-1) No pattern found for current index or parameter error
- others the pattern position in rx buffer.

**Parameters**

- `uart_num`: UART port number

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.

**Note** 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.

**Return**

- (-1) No pattern found for current index or parameter error
- others the pattern position in rx buffer.

**Parameters**

- `uart_num`: UART port number

*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.

**Return**

- `ESP_ERR_NO_MEM` No enough memory
- `ESP_ERR_INVALID_STATE` Driver not installed
- `ESP_FAIL` Parameter error
- `ESP_OK` Success

**Parameters**

- `uart_num`: UART port number
- `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.

*esp\_err\_t* `uart_set_mode` (*uart\_port\_t* *uart\_num*, *uart\_mode\_t* *mode*)

UART set communication mode.

**Note** This function must be executed after `uart_driver_install()`, when the driver object is initialized.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- `uart_num`: Uart number to configure
- `mode`: UART UART mode to set

*esp\_err\_t* **uart\_set\_rx\_timeout** (*uart\_port\_t* `uart_num`, **const** *uint8\_t* `tout_thresh`)

UART set threshold timeout for TOUT feature.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_ERR\_INVALID\_STATE Driver is not installed

**Parameters**

- `uart_num`: Uart number to configure
- `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\_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()`).

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Parameter error

**Parameters**

- `uart_num`: Uart number to configure
- `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 REF\_TICK as UART clock source, by setting `use_ref_tick` field in `uart_config_t` to true.

**Note** 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.

**Return**

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG if `uart_num` is incorrect or `wakeup_threshold` is outside of [3, 0x3ff] range.

**Parameters**

- `uart_num`: UART number
- `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.

#### Return

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG if `out_wakeup_threshold` is NULL

#### Parameters

- `uart_num`: UART number
- [`out`] `out_wakeup_threshold`: output, set to the current value of wakeup threshold for the given UART.

### Structures

**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

*uart\_hw\_flowcontrol\_t* `flow_ctrl`

UART HW flow control mode (cts/rts)

uint8\_t `rx_flow_ctrl_thresh`

UART HW RTS threshold

bool `use_ref_tick`

Set to true if UART should be clocked from REF\_TICK

**struct** `uart_intr_config_t`

UART interrupt configuration parameters for `uart_intr_config` function.

#### Public Members

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

uint8\_t `rx_timeout_thresh`

UART timeout interrupt threshold (unit: time of sending one byte)

uint8\_t `txfifo_empty_intr_thresh`

UART TX empty interrupt threshold.

uint8\_t `rxfifo_full_thresh`

UART RX full interrupt threshold.

**struct** `uart_event_t`

Event structure used in UART event queue.

## Public Members

`uart_event_type_t` **type**

UART event type

`size_t` **size**

UART data size for UART\_DATA event

## Macros

**UART\_FIFO\_LEN**

Length of the hardware FIFO buffers

**UART\_INTR\_MASK**

Mask of all UART interrupts

**UART\_LINE\_INV\_MASK**

TBD

**UART\_BITRATE\_MAX**

Max bit rate supported by UART

**UART\_PIN\_NO\_CHANGE**

Constant for `uart_set_pin` function which indicates that UART pin should not be changed

**UART\_INVERSE\_DISABLE**

Disable UART signal inverse

**UART\_INVERSE\_RXD**

UART RXD input inverse

**UART\_INVERSE\_CTS**

UART CTS input inverse

**UART\_INVERSE\_TXD**

UART TXD output inverse

**UART\_INVERSE\_RTS**

UART RTS output inverse

## Type Definitions

`typedef intr_handle_t` **uart\_isr\_handle\_t**

## Enumerations

`enum` **uart\_mode\_t**

UART mode selection.

*Values:*

**UART\_MODE\_UART** = 0x00

mode: regular UART mode

**UART\_MODE\_RS485\_HALF\_DUPLEX** = 0x01

mode: half duplex RS485 UART mode control by RTS pin

**UART\_MODE\_IRDA** = 0x02

mode: IRDA UART mode

**UART\_MODE\_RS485\_COLLISION\_DETECT** = 0x03

mode: RS485 collision detection UART mode (used for test purposes)

**UART\_MODE\_RS485\_APP\_CTRL** = 0x04

mode: application control RS485 UART mode (used for test purposes)



**enum uart\_word\_length\_t**

UART word length constants.

*Values:***UART\_DATA\_5\_BITS** = 0x0

word length: 5bits

**UART\_DATA\_6\_BITS** = 0x1

word length: 6bits

**UART\_DATA\_7\_BITS** = 0x2

word length: 7bits

**UART\_DATA\_8\_BITS** = 0x3

word length: 8bits

**UART\_DATA\_BITS\_MAX** = 0x4**enum uart\_stop\_bits\_t**

UART stop bits number.

*Values:***UART\_STOP\_BITS\_1** = 0x1

stop bit: 1bit

**UART\_STOP\_BITS\_1\_5** = 0x2

stop bit: 1.5bits

**UART\_STOP\_BITS\_2** = 0x3

stop bit: 2bits

**UART\_STOP\_BITS\_MAX** = 0x4**enum uart\_port\_t**

UART peripheral number.

*Values:***UART\_NUM\_0** = 0x0

UART base address 0x3ff40000

**UART\_NUM\_1** = 0x1

UART base address 0x3ff50000

**UART\_NUM\_MAX****enum uart\_parity\_t**

UART parity constants.

*Values:***UART\_PARITY\_DISABLE** = 0x0

Disable UART parity

**UART\_PARITY\_EVEN** = 0x2

Enable UART even parity

**UART\_PARITY\_ODD** = 0x3

Enable UART odd parity

**enum uart\_hw\_flowcontrol\_t**

UART hardware flow control modes.

*Values:***UART\_HW\_FLOWCTRL\_DISABLE** = 0x0

disable hardware flow control

```

UART_HW_FLOWCTRL_RTS = 0x1
    enable RX hardware flow control (rts)

UART_HW_FLOWCTRL_CTS = 0x2
    enable TX hardware flow control (cts)

UART_HW_FLOWCTRL_CTS_RTS = 0x3
    enable hardware flow control

UART_HW_FLOWCTRL_MAX = 0x4

```

```

enum uart_event_type_t
    UART event types used in the ring buffer.

```

*Values:*

```

UART_DATA
    UART data event

UART_BREAK
    UART break event

UART_BUFFER_FULL
    UART RX buffer full event

UART_FIFO_OVF
    UART FIFO overflow event

UART_FRAME_ERR
    UART RX frame error event

UART_PARITY_ERR
    UART RX parity event

UART_DATA_BREAK
    UART TX data and break event

UART_PATTERN_DET
    UART pattern detected

UART_EVENT_MAX
    UART event max index

```

**GPIO Lookup Macros** You can use macros to specify the **direct** GPIO (UART module connected to pads through direct IO mux without the GPIO mux) number of a UART channel, or vice versa. The pin name can be omitted if the channel of a GPIO number is specified, e.g.:

1. `UART_NUM_2_TXD_DIRECT_GPIO_NUM` is the GPIO number of UART channel 2 TXD pin (17);
2. `UART_GPIO19_DIRECT_CHANNEL` is the UART channel number of GPIO 19 (channel 0);
3. `UART_CTS_GPIO19_DIRECT_CHANNEL` is the UART channel number of GPIO 19, and GPIO 19 must be a CTS pin (channel 0).

### Header File

- `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_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
```

本部分的 API 示例代码存放在 ESP-IDF 示例项目的 [peripherals](#) 目录下。

## 2.4 应用层协议

### 2.4.1 mDNS Service

#### Overview

mDNS is a multicast UDP service that is used to provide local network service and host discovery.

mDNS is installed by default on most operating systems or is available as separate package. On Mac OS it is installed by default and is called Bonjour. Apple releases an installer for Windows that can be found on [Apple's support page](#). On Linux, mDNS is provided by [avahi](#) and is usually installed by default.

### mDNS Properties

- `hostname`: the hostname that the device will respond to. If not set, the hostname will be read from the interface. Example: `my-esp32` will resolve to `my-esp32.local`
- `default_instance`: friendly name for your device, like `Jhon's ESP32 Thing`. If not set, hostname will be used.

Example method to start mDNS for the STA interface and set `hostname` and `default_instance`:

```
void start_mdns_service()
{
    //initialize mDNS service
    esp_err_t err = mdns_init();
    if (err) {
        printf("MDNS Init failed: %d\n", err);
        return;
    }

    //set hostname
    mdns_hostname_set("my-esp32");
    //set default instance
    mdns_instance_name_set("Jhon's ESP32 Thing");
}
```

**mDNS Services** mDNS can advertise information about network services that your device offers. Each service is defined by a few properties.

- `instance_name`: friendly name for your service, like `Jhon's ESP32 Web Server`. If not defined, `default_instance` will be used.
- `service_type`: (required) service type, prepended with underscore. Some common types can be found [here](#).
- `proto`: (required) protocol that the service runs on, prepended with underscore. Example: `_tcp` or `_udp`
- `port`: (required) network port that the service runs on
- `txt`: {var, val} array of strings, used to define properties for your service

Example method to add a few services and different properties:

```
void add_mdns_services()
{
    //add our services
    mdns_service_add(NULL, "_http", "_tcp", 80, NULL, 0);
    mdns_service_add(NULL, "_arduino", "_tcp", 3232, NULL, 0);
    mdns_service_add(NULL, "_myservice", "_udp", 1234, NULL, 0);

    //NOTE: services must be added before their properties can be set
    //use custom instance for the web server
    mdns_service_instance_name_set("_http", "_tcp", "Jhon's ESP32 Web Server");

    mdns_txt_item_t serviceTxtData[3] = {
        {"board", "esp32"},
        {"u", "user"},
        {"p", "password"}
    };
    //set txt data for service (will free and replace current data)
    mdns_service_txt_set("_http", "_tcp", serviceTxtData, 3);

    //change service port
```

(下页继续)

```
mdns_service_port_set("_myservice", "udp", 4321);
}
```

**mDNS Query** mDNS provides methods for browsing for services and resolving host's IP/IPv6 addresses.

Results for services are returned as a linked list of `mdns_result_t` objects.

Example method to resolve host IPs:

```
void resolve_mdns_host(const char * host_name)
{
    printf("Query A: %s.local", host_name);

    struct ip4_addr addr;
    addr.addr = 0;

    esp_err_t err = mdns_query_a(host_name, 2000, &addr);
    if(err){
        if(err == ESP_ERR_NOT_FOUND){
            printf("Host was not found!");
            return;
        }
        printf("Query Failed");
        return;
    }

    printf(IPSTR, IP2STR(&addr));
}
```

Example method to resolve local services:

```
static const char * if_str[] = {"STA", "AP", "ETH", "MAX"};
static const char * ip_protocol_str[] = {"V4", "V6", "MAX"};

void mdns_print_results(mdns_result_t * results){
    mdns_result_t * r = results;
    mdns_ip_addr_t * a = NULL;
    int i = 1, t;
    while(r){
        printf("%d: Interface: %s, Type: %s\n", i++, if_str[r->tcpip_if], ip_
        ↪protocol_str[r->ip_protocol]);
        if(r->instance_name){
            printf(" PTR : %s\n", r->instance_name);
        }
        if(r->hostname){
            printf(" SRV : %s.local:%u\n", r->hostname, r->port);
        }
        if(r->txt_count){
            printf(" TXT : [%u] ", r->txt_count);
            for(t=0; t<r->txt_count; t++){
                printf("%s=%s; ", r->txt[t].key, r->txt[t].value);
            }
            printf("\n");
        }
        a = r->addr;
        while(a){
            if(a->addr.type == IPADDR_TYPE_V6){
                printf(" AAAA: " IPV6STR "\n", IPV62STR(a->addr.u_addr.ip6));
            } else {
                printf(" A : " IPSTR "\n", IP2STR(&(a->addr.u_addr.ip4)));
            }
        }
    }
}
```

(下页继续)

```

        }
        a = a->next;
    }
    r = r->next;
}

}

void find_mdns_service(const char * service_name, const char * proto)
{
    ESP_LOGI(TAG, "Query PTR: %s.%s.local", service_name, proto);

    mdns_result_t * results = NULL;
    esp_err_t err = mdns_query_ptr(service_name, proto, 3000, 20, &results);
    if(err){
        ESP_LOGE(TAG, "Query Failed");
        return;
    }
    if(!results){
        ESP_LOGW(TAG, "No results found!");
        return;
    }

    mdns_print_results(results);
    mdns_query_results_free(results);
}

```

Example of using the methods above:

```

void my_app_some_method(){
    //search for esp32-mdns.local
    resolve_mdns_host("esp32-mdns");

    //search for HTTP servers
    find_mdns_service("_http", "_tcp");
    //or file servers
    find_mdns_service("_smb", "_tcp"); //windows sharing
    find_mdns_service("_afpovertcp", "_tcp"); //apple sharing
    find_mdns_service("_nfs", "_tcp"); //NFS server
    find_mdns_service("_ftp", "_tcp"); //FTP server
    //or networked printer
    find_mdns_service("_printer", "_tcp");
    find_mdns_service("_ipp", "_tcp");
}

```

## Application Example

mDNS server/scanner example: [protocols/mdns](#).

## API Reference

### Header File

- [mdns/include/mdns.h](#)

### Functions

`esp_err_t mdns_init()`

Initialize mDNS on given interface.

**Return**

- ESP\_OK on success
- ESP\_ERR\_INVALID\_STATE when failed to register event handler
- ESP\_ERR\_NO\_MEM on memory error
- ESP\_FAIL when failed to start mdns task

void **mdns\_free** ()

Stop and free mDNS server.

*esp\_err\_t* **mdns\_hostname\_set** (const char \*hostname)

Set the hostname for mDNS server required if you want to advertise services.

**Return**

- ESP\_OK success
- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_ERR\_NO\_MEM memory error

**Parameters**

- hostname: Hostname to set

*esp\_err\_t* **mdns\_instance\_name\_set** (const char \*instance\_name)

Set the default instance name for mDNS server.

**Return**

- ESP\_OK success
- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_ERR\_NO\_MEM memory error

**Parameters**

- instance\_name: Instance name to set

*esp\_err\_t* **mdns\_service\_add** (const char \*instance\_name, const char \*service\_type, const char \*proto, uint16\_t port, *mdns\_txt\_item\_t* txt[], size\_t num\_items)

Add service to mDNS server.

**Return**

- ESP\_OK success
- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_ERR\_NO\_MEM memory error
- ESP\_FAIL failed to add service

**Parameters**

- instance\_name: instance name to set. If NULL, global instance name or hostname will be used
- service\_type: service type (\_http, \_ftp, etc)
- proto: service protocol (\_tcp, \_udp)
- port: service port
- txt: string array of TXT data (eg. {{ "var" , " val" }, { "other" , " 2" }})
- num\_items: number of items in TXT data

*esp\_err\_t* **mdns\_service\_remove** (const char \*service\_type, const char \*proto)

Remove service from mDNS server.

**Return**

- ESP\_OK success
- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_ERR\_NOT\_FOUND Service not found
- ESP\_ERR\_NO\_MEM memory error

**Parameters**

- service\_type: service type (\_http, \_ftp, etc)
- proto: service protocol (\_tcp, \_udp)

*esp\_err\_t* **mdns\_service\_instance\_name\_set** (const char \*service\_type, const char \*proto, const char \*instance\_name)

Set instance name for service.

**Return**

- ESP\_OK success

- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_ERR\_NOT\_FOUND Service not found
- ESP\_ERR\_NO\_MEM memory error

**Parameters**

- *service\_type*: service type (`_http`, `_ftp`, etc)
- *proto*: service protocol (`_tcp`, `_udp`)
- *instance\_name*: instance name to set

*esp\_err\_t* `mdns_service_port_set` (`const char *service_type`, `const char *proto`, `uint16_t port`)

Set service port.

**Return**

- ESP\_OK success
- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_ERR\_NOT\_FOUND Service not found
- ESP\_ERR\_NO\_MEM memory error

**Parameters**

- *service\_type*: service type (`_http`, `_ftp`, etc)
- *proto*: service protocol (`_tcp`, `_udp`)
- *port*: service port

*esp\_err\_t* `mdns_service_txt_set` (`const char *service_type`, `const char *proto`, `mdns_txt_item_t txt[]`, `uint8_t num_items`)

Replace all TXT items for service.

**Return**

- ESP\_OK success
- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_ERR\_NOT\_FOUND Service not found
- ESP\_ERR\_NO\_MEM memory error

**Parameters**

- *service\_type*: service type (`_http`, `_ftp`, etc)
- *proto*: service protocol (`_tcp`, `_udp`)
- *txt*: array of TXT data (eg. {{ "var" ," val" },{ "other" ," 2" }})
- *num\_items*: number of items in TXT data

*esp\_err\_t* `mdns_service_txt_item_set` (`const char *service_type`, `const char *proto`, `const char *key`, `const char *value`)

Set/Add TXT item for service TXT record.

**Return**

- ESP\_OK success
- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_ERR\_NOT\_FOUND Service not found
- ESP\_ERR\_NO\_MEM memory error

**Parameters**

- *service\_type*: service type (`_http`, `_ftp`, etc)
- *proto*: service protocol (`_tcp`, `_udp`)
- *key*: the key that you want to add/update
- *value*: the new value of the key

*esp\_err\_t* `mdns_service_txt_item_remove` (`const char *service_type`, `const char *proto`, `const char *key`)

Remove TXT item for service TXT record.

**Return**

- ESP\_OK success
- ESP\_ERR\_INVALID\_ARG Parameter error
- ESP\_ERR\_NOT\_FOUND Service not found
- ESP\_ERR\_NO\_MEM memory error

**Parameters**

- *service\_type*: service type (`_http`, `_ftp`, etc)
- *proto*: service protocol (`_tcp`, `_udp`)



- `key`: the key that you want to remove

`esp_err_t mdns_service_remove_all()`

Remove and free all services from mDNS server.

#### Return

- `ESP_OK` success
- `ESP_ERR_INVALID_ARG` Parameter error

`esp_err_t mdns_query(const char *name, const char *service_type, const char *proto, uint16_t type, uint32_t timeout, size_t max_results, mdns_result_t **results)`

Query mDNS for host or service All following query methods are derived from this one.

#### Return

- `ESP_OK` success
- `ESP_ERR_INVALID_STATE` mDNS is not running
- `ESP_ERR_NO_MEM` memory error
- `ESP_ERR_INVALID_ARG` timeout was not given

#### Parameters

- `name`: service instance or host name (NULL for PTR queries)
- `service_type`: service type (`_http`, `_arduino`, `_ftp` etc.) (NULL for host queries)
- `proto`: service protocol (`_tcp`, `_udp`, etc.) (NULL for host queries)
- `type`: type of query (`MDNS_TYPE_*`)
- `timeout`: time in milliseconds to wait for answers.
- `max_results`: maximum results to be collected
- `results`: pointer to the results of the query results must be freed using `mdns_query_results_free` below

`void mdns_query_results_free(mdns_result_t *results)`

Free query results.

#### Parameters

- `results`: linked list of results to be freed

`esp_err_t mdns_query_ptr(const char *service_type, const char *proto, uint32_t timeout, size_t max_results, mdns_result_t **results)`

Query mDNS for service.

#### Return

- `ESP_OK` success
- `ESP_ERR_INVALID_STATE` mDNS is not running
- `ESP_ERR_NO_MEM` memory error
- `ESP_ERR_INVALID_ARG` parameter error

#### Parameters

- `service_type`: service type (`_http`, `_arduino`, `_ftp` etc.)
- `proto`: service protocol (`_tcp`, `_udp`, etc.)
- `timeout`: time in milliseconds to wait for answer.
- `max_results`: maximum results to be collected
- `results`: pointer to the results of the query

`esp_err_t mdns_query_srv(const char *instance_name, const char *service_type, const char *proto, uint32_t timeout, mdns_result_t **result)`

Query mDNS for SRV record.

#### Return

- `ESP_OK` success
- `ESP_ERR_INVALID_STATE` mDNS is not running
- `ESP_ERR_NO_MEM` memory error
- `ESP_ERR_INVALID_ARG` parameter error

#### Parameters

- `instance_name`: service instance name
- `service_type`: service type (`_http`, `_arduino`, `_ftp` etc.)
- `proto`: service protocol (`_tcp`, `_udp`, etc.)
- `timeout`: time in milliseconds to wait for answer.

- `result`: pointer to the result of the query

```
esp_err_t mdns_query_txt(const char *instance_name, const char *service_type, const char
                        *proto, uint32_t timeout, mdns_result_t **result)
```

Query mDNS for TXT record.

#### Return

- `ESP_OK` success
- `ESP_ERR_INVALID_STATE` mDNS is not running
- `ESP_ERR_NO_MEM` memory error
- `ESP_ERR_INVALID_ARG` parameter error

#### Parameters

- `instance_name`: service instance name
- `service_type`: service type (`_http`, `_arduino`, `_ftp` etc.)
- `proto`: service protocol (`_tcp`, `_udp`, etc.)
- `timeout`: time in milliseconds to wait for answer.
- `result`: pointer to the result of the query

```
esp_err_t mdns_query_a(const char *host_name, uint32_t timeout, ip4_addr_t *addr)
```

Query mDNS for A record.

#### Return

- `ESP_OK` success
- `ESP_ERR_INVALID_STATE` mDNS is not running
- `ESP_ERR_NO_MEM` memory error
- `ESP_ERR_INVALID_ARG` parameter error

#### Parameters

- `host_name`: host name to look for
- `timeout`: time in milliseconds to wait for answer.
- `addr`: pointer to the resulting IP4 address

```
esp_err_t mdns_query_aaaa(const char *host_name, uint32_t timeout, ip6_addr_t *addr)
```

Query mDNS for AAAA record.

#### Return

- `ESP_OK` success
- `ESP_ERR_INVALID_STATE` mDNS is not running
- `ESP_ERR_NO_MEM` memory error
- `ESP_ERR_INVALID_ARG` parameter error

#### Parameters

- `host_name`: host name to look for
- `timeout`: time in milliseconds to wait for answer. If 0, `max_results` needs to be defined
- `addr`: pointer to the resulting IP6 address

```
esp_err_t mdns_handle_system_event(void *ctx, system_event_t *event)
```

System event handler This method controls the service state on all active interfaces and applications are required to call it from the system event handler for normal operation of mDNS service.

#### Parameters

- `ctx`: The system event context
- `event`: The system event

## Structures

```
struct mdns_txt_item_t
```

mDNS basic text item structure Used in `mdns_service_add()`

## Public Members

```
const char *key
    item key name
```

**const char \*value**  
item value string  
**struct mdns\_ip\_addr\_s**  
mDNS query linked list IP item

### Public Members

**ip\_addr\_t addr**  
IP address

**struct mdns\_ip\_addr\_s \*next**  
next IP, or NULL for the last IP in the list

**struct mdns\_result\_s**  
mDNS query result structure

### Public Members

**struct mdns\_result\_s \*next**  
next result, or NULL for the last result in the list

**tcpip\_adapter\_if\_t tcpip\_if**  
interface on which the result came (AP/STA/ETH)

**mdns\_ip\_protocol\_t ip\_protocol**  
ip\_protocol type of the interface (v4/v6)

**char \*instance\_name**  
instance name

**char \*hostname**  
hostname

**uint16\_t port**  
service port

**mdns\_txt\_item\_t \*txt**  
txt record

**size\_t txt\_count**  
number of txt items

**mdns\_ip\_addr\_t \*addr**  
linked list of IP addresses found

### Macros

**MDNS\_TYPE\_A**

**MDNS\_TYPE\_PTR**

**MDNS\_TYPE\_TXT**

**MDNS\_TYPE\_AAAA**

**MDNS\_TYPE\_SRV**

**MDNS\_TYPE\_OPT**

**MDNS\_TYPE\_NSEC**

**MDNS\_TYPE\_ANY**

### Type Definitions

```
typedef struct mdns_ip_addr_s mdns_ip_addr_t
    mDNS query linked list IP item
typedef struct mdns_result_s mdns_result_t
    mDNS query result structure
```

### Enumerations

```
enum mdns_ip_protocol_t
    mDNS enum to specify the ip_protocol type
```

Values:

```
MDNS_IP_PROTOCOL_V4
MDNS_IP_PROTOCOL_V6
MDNS_IP_PROTOCOL_MAX
```

## 2.4.2 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_conn_new()`: for opening a new TLS connection `* esp_tls_conn_read/write()`: for reading/writing from the connection `* esp_tls_conn_delete()`: 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](#).

### API Reference

#### Header File

- [esp-tls/esp\\_tls.h](#)

#### Functions

`esp_tls_t *esp_tls_init ()`

Create TLS connection.

This function allocates and initializes esp-tls structure handle.

**Return** `tls` Pointer to esp-tls as esp-tls handle if successfully initialized, NULL if allocation error

`esp_tls_t *esp_tls_conn_new (const char *hostname, int hostlen, int port, const esp_tls_cfg_t *cfg)`

Create a new blocking TLS/SSL connection.

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

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

**Return** pointer to `esp_tls_t`, or NULL if connection couldn't be opened.

#### Parameters

- [in] `hostname`: Hostname of the host.
- [in] `hostlen`: Length of hostname.
- [in] `port`: Port number of the host.

- [in] `cfg`: 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.

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.

#### Return

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

#### Parameters

- [in] `hostname`: Hostname of the host.
- [in] `hostlen`: Length of `hostname`.
- [in] `port`: Port number of the host.
- [in] `cfg`: 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.
- [in] `tls`: Pointer to esp-tls as esp-tls handle.

`esp_tls_t` \*`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.

The behaviour is same as `esp_tls_conn_new()` API. However this API accepts `host's` url.

**Return** pointer to `esp_tls_t`, or NULL if connection couldn't be opened.

#### Parameters

- [in] `url`: url of host.
- [in] `cfg`: 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.

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.

#### Return

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

#### Parameters

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

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()` API. However this API accepts `host's` url.

#### Return

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

#### Parameters

- [in] url: url of host.
- [in] cfg: TLS configuration as `esp_tls_cfg_t`.
- [in] tls: pointer to esp-tls as esp-tls handle.

**static** ssize\_t **esp\_tls\_conn\_write** (*esp\_tls\_t* \*tls, const void \*data, size\_t datalen)

Write from buffer 'data' into specified tls connection.

#### Return

- >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. The underlying connection was closed.
- <0 if write operation was not successful, because either an error occurred or an action must be taken by the calling process.

#### Parameters

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

**static** ssize\_t **esp\_tls\_conn\_read** (*esp\_tls\_t* \*tls, void \*data, size\_t datalen)

Read from specified tls connection into the buffer 'data' .

#### Return

- >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.

#### Parameters

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

void **esp\_tls\_conn\_delete** (*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()` or `esp_tls_conn_http_new()` APIs.

#### Parameters

- [in] tls: pointer to esp-tls as esp-tls handle.

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 mbedtls's `mbedtls_ssl_get_bytes_avail()` API.

#### Return

- -1 in case of invalid arg
- bytes available in the application data record read buffer

#### Parameters

- [in] tls: pointer to esp-tls as esp-tls handle.

*esp\_err\_t* **esp\_tls\_init\_global\_ca\_store** ()

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\_ERR\_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()`.

#### Return

- ESP\_OK if adding certificates was successful.
- Other if an error occurred or an action must be taken by the calling process.

#### Parameters

- [in] `cacert_pem_buf`: 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.
- [in] `cacert_pem_bytes`: Length of the buffer.

`MBEDTLS_X509_CRT *`**esp\_tls\_get\_global\_ca\_store**()

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*.

**Note** Modifying the pointer might cause a failure in verifying the certificates.

#### Return

- Pointer to the global CA store currently being used if successful.
- NULL if there is no global CA store set.

void **esp\_tls\_free\_global\_ca\_store**()

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 \*mbedtls\_code, int \*mbedtls\_flags)

Returns last error in *esp\_tls* with detailed mbedtls related error codes. The error information is cleared internally upon return.

#### 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

#### Parameters

- [in] `h`: esp-tls error handle.
- [out] `mbedtls_code`: last error code returned from mbedtls api (set to zero if none) This pointer could be NULL if caller does not care about `mbedtls_code`
- [out] `mbedtls_flags`: last certification verification flags (set to zero if none) This pointer could be NULL if caller does not care about `mbedtls_flags`

## Structures

**struct esp\_tls\_last\_error**

Error structure containing relevant errors in case tls error occurred.

## Public Members

*esp\_err\_t* **last\_error**

error code (based on ESP\_ERR\_ESP\_TLS\_BASE) of the last occurred error

int **mbedtls\_error\_code**

mbedtls error code from last mbedtls failed api

int **MBEDTLS\_FLAGS**

last certification verification flags

struct **tls\_keep\_alive\_cfg**

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.

### 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

const unsigned char \***cacert\_pem\_buf**

Certificate Authority's certificate in a buffer. This buffer should be NULL terminated

unsigned int **cacert\_pem\_bytes**

Size of Certificate Authority certificate pointed to by cacert\_pem\_buf

const unsigned char \***clientcert\_pem\_buf**

Client certificate in a buffer This buffer should be NULL terminated

unsigned int **clientcert\_pem\_bytes**

Size of client certificate pointed to by clientcert\_pem\_buf

const unsigned char \***clientkey\_pem\_buf**

Client key in a buffer This buffer should be NULL terminated

unsigned int **clientkey\_pem\_bytes**

Size of client key pointed to by clientkey\_pem\_buf

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

int **timeout\_ms**

Network timeout in milliseconds

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

**struct esp\_tls**

ESP-TLS Connection Handle.

### Public Members

mbedtls\_ssl\_context **ssl**

TLS/SSL context

mbedtls\_entropy\_context **entropy**

mbedTLS entropy context structure

mbedtls\_ctr\_drbg\_context **ctr\_drbg**

mbedTLS ctr drbg context structure. CTR\_DRBG is deterministic random bit generation based on AES-256

mbedtls\_ssl\_config **conf**

TLS/SSL configuration to be shared between mbedtls\_ssl\_context structures

mbedtls\_net\_context **server\_fd**

mbedTLS wrapper type for sockets

mbedtls\_x509\_cert **cacert**

Container for the X.509 CA certificate

mbedtls\_x509\_cert \***cacert\_ptr**

Pointer to the cacert being used.

mbedtls\_x509\_cert **clientcert**

Container for the X.509 client certificate

mbedtls\_pk\_context **clientkey**

Container for the private key of the client certificate

int **sockfd**

Underlying socket file descriptor.

ssize\_t (\***read**) (**struct esp\_tls** \*tls, char \*data, size\_t datalen)

Callback function for reading data from TLS/SSL connection.

ssize\_t (\***write**) (**struct esp\_tls** \*tls, **const** char \*data, size\_t datalen)

Callback function for writing data to TLS/SSL connection.

*esp\_tls\_conn\_state\_t* **conn\_state**

ESP-TLS Connection state

fd\_set **rset**

read file descriptors

fd\_set **wset**

write file descriptors

bool **is\_tls**

indicates connection type (TLS or NON-TLS)

*esp\_tls\_role\_t* **role**

esp-tls role

- ESP\_TLS\_CLIENT
- ESP\_TLS\_SERVER

*esp\_tls\_error\_handle\_t* **error\_handle**  
handle to error descriptor

### Macros

#### **ESP\_ERR\_ESP\_TLS\_BASE**

Starting number of ESP-TLS error codes

#### **ESP\_ERR\_ESP\_TLS\_CANNOT\_RESOLVE\_HOSTNAME**

Error if hostname couldn't be resolved upon tls connection

#### **ESP\_ERR\_ESP\_TLS\_CANNOT\_CREATE\_SOCKET**

Failed to create socket

#### **ESP\_ERR\_ESP\_TLS\_UNSUPPORTED\_PROTOCOL\_FAMILY**

Unsupported protocol family

#### **ESP\_ERR\_ESP\_TLS\_FAILED\_CONNECT\_TO\_HOST**

Failed to connect to host

#### **ESP\_ERR\_ESP\_TLS\_SOCKET\_SETOPT\_FAILED**

failed to set socket option

#### **ESP\_ERR\_MBEDTLS\_CERT\_PARTLY\_OK**

mbedtls parse certificates was partly successful

#### **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\_ESP\_TLS\_CONNECTION\_TIMEOUT**

new connection in esp\_tls\_low\_level\_conn connection timeouted

**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.

```
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 tls_keep_alive_cfg tls_keep_alive_cfg_t
```

Keep alive parameters structure.

```
typedef struct esp_tls_cfg esp_tls_cfg_t
```

ESP-TLS configuration parameters.

```
typedef struct esp_tls esp_tls_t
```

ESP-TLS Connection Handle.

**Enumerations**

```
enum esp_tls_conn_state
```

ESP-TLS Connection State.

*Values:*

```
ESP_TLS_INIT = 0
```

```
ESP_TLS_CONNECTING
```

```
ESP_TLS_HANDSHAKE
```

```
ESP_TLS_FAIL
```

```
ESP_TLS_DONE
```

```
enum esp_tls_role
```

*Values:*

```
ESP_TLS_CLIENT = 0
```

```
ESP_TLS_SERVER
```

**2.4.3 ESP HTTP Client****Overview**

`esp_http_client` provides an API for making HTTP/S requests from ESP-IDF programs. The steps to use this API for an HTTP request are:

- `esp_http_client_init()`: To use the HTTP client, the first thing we must do is create an `esp_http_client` by pass into this function with the `esp_http_client_config_t` configurations. Which configuration values we do not define, the library will use default.
- `esp_http_client_perform()`: The `esp_http_client` argument created from the init function is needed. This function performs all operations of the `esp_http_client`, from opening the connection, sending data, downloading data and closing the connection if necessary. All related events will be invoked in the `event_handle` (defined by `esp_http_client_config_t`). This function performs its job and blocks the current task until it' s done
- `esp_http_client_cleanup()`: After completing our `esp_http_client`' s task, this is the last function to be called. It will close the connection (if any) and free up all the memory allocated to the HTTP client

## Application Example

```

esp_err_t _http_event_handle(esp_http_client_event_t *evt)
{
    switch(evt->event_id) {
        case HTTP_EVENT_ERROR:
            ESP_LOGI(TAG, "HTTP_EVENT_ERROR");
            break;
        case HTTP_EVENT_ON_CONNECTED:
            ESP_LOGI(TAG, "HTTP_EVENT_ON_CONNECTED");
            break;
        case HTTP_EVENT_HEADER_SENT:
            ESP_LOGI(TAG, "HTTP_EVENT_HEADER_SENT");
            break;
        case HTTP_EVENT_ON_HEADER:
            ESP_LOGI(TAG, "HTTP_EVENT_ON_HEADER");
            printf("%.s", evt->data_len, (char*)evt->data);
            break;
        case HTTP_EVENT_ON_DATA:
            ESP_LOGI(TAG, "HTTP_EVENT_ON_DATA, len=%d", evt->data_len);
            if (!esp_http_client_is_chunked_response(evt->client)) {
                printf("%.s", evt->data_len, (char*)evt->data);
            }

            break;
        case HTTP_EVENT_ON_FINISH:
            ESP_LOGI(TAG, "HTTP_EVENT_ON_FINISH");
            break;
        case HTTP_EVENT_DISCONNECTED:
            ESP_LOGI(TAG, "HTTP_EVENT_DISCONNECTED");
            break;
    }
    return ESP_OK;
}

esp_http_client_config_t config = {
    .url = "http://httpbin.org/redirect/2",
    .event_handler = _http_event_handle,
};
esp_http_client_handle_t client = esp_http_client_init(&config);
esp_err_t err = esp_http_client_perform(client);

if (err == ESP_OK) {
    ESP_LOGI(TAG, "Status = %d, content_length = %d",
        esp_http_client_get_status_code(client),
        esp_http_client_get_content_length(client));
}
esp_http_client_cleanup(client);

```

## Persistent Connections

Persistent connections means that the HTTP client can re-use the same connection for several transfers. If the server does not request to close the connection with the `Connection: close` header, the new transfer with sample ip address, port, and protocol.

To allow the HTTP client to take full advantage of persistent connections, you should do as many of your file transfers as possible using the same handle.

## Persistent Connections example

```

esp_err_t err;
esp_http_client_config_t config = {
    .url = "http://httpbin.org/get",
};
esp_http_client_handle_t client = esp_http_client_init(&config);
// first request
err = esp_http_client_perform(client);

// second request
esp_http_client_set_url(client, "http://httpbin.org/anything")
esp_http_client_set_method(client, HTTP_METHOD_DELETE);
esp_http_client_set_header(client, "HeaderKey", "HeaderValue");
err = esp_http_client_perform(client);

esp_http_client_cleanup(client);

```

## HTTPS

The HTTP client supports SSL connections using **mbedtls**, with the **url** configuration starting with `https` scheme (or `transport_type = HTTP_TRANSPORT_OVER_SSL`). HTTPS support can be configured via `:ref:CONFIG_ENABLE_HTTPS` (enabled by default).

**注解:** By providing information using HTTPS, the library will use the SSL transport type to connect to the server. If you want to verify server, then need to provide additional certificate in PEM format, and provide to `cert_pem` in `esp_http_client_config_t`

## HTTPS example

```

static void https ()
{
    esp_http_client_config_t config = {
        .url = "https://www.howsmyssl.com",
        .cert_pem = howsmyssl_com_root_cert_pem_start,
    };
    esp_http_client_handle_t client = esp_http_client_init(&config);
    esp_err_t err = esp_http_client_perform(client);

    if (err == ESP_OK) {
        ESP_LOGI(TAG, "Status = %d, content_length = %d",
            esp_http_client_get_status_code(client),
            esp_http_client_get_content_length(client));
    }
    esp_http_client_cleanup(client);
}

```

## HTTP Stream

Some applications need to open the connection and control the reading of the data in an active manner. the HTTP client supports some functions to make this easier, of course, once you use these functions you should not use the `esp_http_client_perform()` function with that handle, and `esp_http_client_init()` always to called first to get the handle. Perform that functions in the order below:

- `esp_http_client_init()`: to create and handle
- `esp_http_client_set_*` or `esp_http_client_delete_*`: to modify the http connection information (optional)
- `esp_http_client_open()`: Open the http connection with `write_len` parameter, `write_len=0` if we only need read

- `esp_http_client_write()`: Upload data, max length equal to `write_len` of `esp_http_client_open()` function. We may not need to call it if `write_len=0`
- `esp_http_client_fetch_headers()`: After sending the headers and write data (if any) to the server, this function will read the HTTP Server response headers. Calling this function will return the `content-length` from the Server, and we can call `esp_http_client_get_status_code()` for the HTTP status of the connection.
- `esp_http_client_read()`: Now, we can read the HTTP stream by this function.
- `esp_http_client_close()`: We should the connection after finish
- `esp_http_client_cleanup()`: And release the resources

**Perform HTTP request as Stream reader** Check the example function `http_perform_as_stream_reader` at [protocols/esp\\_http\\_client](#).

## HTTP Authentication

The HTTP client supports both **Basic** and **Digest** Authentication. By providing usernames and passwords in `url` or in the `username`, `password` of `config` entry. And with `auth_type = HTTP_AUTH_TYPE_BASIC`, the HTTP client takes only 1 perform to pass the authentication process. If `auth_type = HTTP_AUTH_TYPE_NONE`, but there are `username` and `password` in the configuration, the HTTP client takes 2 performs. The first time it connects to the server and receives the `UNAUTHORIZED` header. Based on this information, it will know which authentication method to choose, and perform it on the second.

### Config authentication example with URI

```
esp_http_client_config_t config = {
    .url = "http://user:passwd@httpbin.org/basic-auth/user/passwd",
    .auth_type = HTTP_AUTH_TYPE_BASIC,
};
```

### Config authentication example with username, password entry

```
esp_http_client_config_t config = {
    .url = "http://httpbin.org/basic-auth/user/passwd",
    .username = "user",
    .password = "passwd",
    .auth_type = HTTP_AUTH_TYPE_BASIC,
};
```

HTTP Client example: [protocols/esp\\_http\\_client](#).

## API Reference

### Header File

- `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.

#### Return

- `esp_http_client_handle_t`
- NULL if any errors

#### Parameters

- [in] `config`: The configurations, see `http_client_config_t`

**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 invokes to set options for the following `esp_http_client_perform`.

**Note** 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`.

**Return**

- ESP\_OK on successful
- ESP\_FAIL on error

**Parameters**

- `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.

**Return**

- ESP\_OK
- ESP\_FAIL

**Parameters**

- [in] `client`: The `esp_http_client` handle
- [in] `url`: The url

**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.

**Return**

- ESP\_OK
- ESP\_FAIL

**Parameters**

- [in] `client`: The `esp_http_client` handle
- [in] `data`: post data pointer
- [in] `len`: post length

**int esp\_http\_client\_get\_post\_field**(*esp\_http\_client\_handle\_t client, char \*\*data*)

Get current post field information.

**Return** Size of post data

**Parameters**

- [in] `client`: The client
- [out] `data`: Point to post data pointer

**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.

**Return**

- ESP\_OK
- ESP\_FAIL

**Parameters**

- [in] client: The esp\_http\_client handle
- [in] key: The header key
- [in] value: The header value

*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.

**Return**

- ESP\_OK
- ESP\_FAIL

**Parameters**

- [in] client: The esp\_http\_client handle
- [in] key: The header key
- [out] value: The header value

*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.

**Return**

- ESP\_OK
- ESP\_ERR\_INVALID\_ARG

**Parameters**

- [in] client: The esp\_http\_client handle
- [out] value: The username value

*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.

**Return**

- ESP\_OK
- ESP\_ERR\_INVALID\_ARG

**Parameters**

- [in] client: The esp\_http\_client handle
- [in] username: The username value

*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.

**Return**

- ESP\_OK
- ESP\_ERR\_INVALID\_ARG

**Parameters**

- [in] client: The esp\_http\_client handle
- [out] value: The password value

*esp\_err\_t* **esp\_http\_client\_set\_password** (*esp\_http\_client\_handle\_t* client, 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.

**Return**

- ESP\_OK
- ESP\_ERR\_INVALID\_ARG

**Parameters**

- [in] client: The esp\_http\_client handle



- [in] password: The password value

`esp_err_t esp_http_client_set_method(esp_http_client_handle_t client, esp_http_client_method_t method)`

Set http request method.

**Return**

- ESP\_OK
- ESP\_ERR\_INVALID\_ARG

**Parameters**

- [in] client: The esp\_http\_client handle
- [in] method: The method

`esp_err_t esp_http_client_delete_header(esp_http_client_handle_t client, const char *key)`

Delete http request header.

**Return**

- ESP\_OK
- ESP\_FAIL

**Parameters**

- [in] client: The esp\_http\_client handle
- [in] key: The key

`esp_err_t 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.

**Return**

- ESP\_OK
- ESP\_FAIL

**Parameters**

- [in] client: The esp\_http\_client handle
- [in] write\_len: HTTP Content length need to write to the server

`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()

**Return**

- (-1) if any errors
- Length of data written

**Parameters**

- [in] client: The esp\_http\_client handle
- buffer: The buffer
- [in] len: This value must not be larger than the write\_len parameter provided to esp\_http\_client\_open()

`int 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.

**Return**

- (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
- Download data length defined by content-length header

**Parameters**

- [in] client: The esp\_http\_client handle

`bool esp_http_client_is_chunked_response(esp_http_client_handle_t client)`

Check response data is chunked.

**Return** true or false

**Parameters**

- [in] client: The esp\_http\_client handle

`int esp_http_client_read(esp_http_client_handle_t client, char *buffer, int len)`

Read data from http stream.

**Return**

- (-1) if any errors
- Length of data was read

**Parameters**

- [in] `client`: The `esp_http_client` handle
- `buffer`: The buffer
- [in] `len`: The length

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`

**Return** Status code

**Parameters**

- [in] `client`: The `esp_http_client` handle

int `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`

**Return**

- (-1) Chunked transfer
- Content-Length value as bytes

**Parameters**

- [in] `client`: The `esp_http_client` handle

*esp\_err\_t* `esp_http_client_close` (*esp\_http\_client\_handle\_t client*)

Close http connection, still kept all http request resources.

**Return**

- ESP\_OK
- ESP\_FAIL

**Parameters**

- [in] `client`: The `esp_http_client` handle

*esp\_err\_t* `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`.

**Return**

- ESP\_OK
- ESP\_FAIL

**Parameters**

- [in] `client`: The `esp_http_client` handle

*esp\_http\_client\_transport\_t* `esp_http_client_get_transport_type` (*esp\_http\_client\_handle\_t client*)

Get transport type.

**Return**

- HTTP\_TRANSPORT\_UNKNOWN
- HTTP\_TRANSPORT\_OVER\_TCP
- HTTP\_TRANSPORT\_OVER\_SSL

**Parameters**

- [in] `client`: The `esp_http_client` handle

*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.

**Return**

- ESP\_OK

- ESP\_FAIL

#### Parameters

- [in] `client`: The `esp_http_client` handle

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.

**Note** There is a possibility of receiving body message with redirection status codes, thus make sure to flush off body data after calling this API.

#### Parameters

- [in] `client`: 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.

#### Return

- true
- false

#### Parameters

- [in] `client`: The `esp_http_client` handle

### 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

**const char \*client\_cert\_pem**  
SSL client certification, PEM format as string, if the server requires to verify client

**const char \*client\_key\_pem**  
SSL client key, PEM format as string, if the server requires to verify client

**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 redirection number, using default value if zero

*http\_event\_handle\_cb* **event\_handler**  
HTTP Event Handle

*esp\_http\_client\_transport\_t* **transport\_type**  
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

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

### 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\_FETCH\_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

### 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 = 0
        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 = HTTP_EVENT_HEADERS_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

**enum esp\_http\_client\_transport\_t**

HTTP Client transport.

*Values:***HTTP\_TRANSPORT\_UNKNOWN = 0x0**

Unknown

**HTTP\_TRANSPORT\_OVER\_TCP**

Transport over tcp

**HTTP\_TRANSPORT\_OVER\_SSL**

Transport over ssl

**enum esp\_http\_client\_method\_t**

HTTP method.

*Values:***HTTP\_METHOD\_GET = 0**

HTTP GET Method

**HTTP\_METHOD\_POST**

HTTP POST Method

**HTTP\_METHOD\_PUT**

HTTP PUT Method

**HTTP\_METHOD\_PATCH**

HTTP PATCH Method

**HTTP\_METHOD\_DELETE**

HTTP DELETE Method

**HTTP\_METHOD\_HEAD**

HTTP HEAD Method

**HTTP\_METHOD\_NOTIFY**

HTTP NOTIFY Method

**HTTP\_METHOD\_SUBSCRIBE**

HTTP SUBSCRIBE Method

**HTTP\_METHOD\_UNSUBSCRIBE**

HTTP UNSUBSCRIBE Method

**HTTP\_METHOD\_OPTIONS**

HTTP OPTIONS Method

**HTTP\_METHOD\_MAX****enum esp\_http\_client\_auth\_type\_t**

HTTP Authentication type.

*Values:***HTTP\_AUTH\_TYPE\_NONE = 0**

No authentication

**HTTP\_AUTH\_TYPE\_BASIC**

HTTP Basic authentication

**HTTP\_AUTH\_TYPE\_DIGEST**

HTTP Digest authentication

**enum HttpStatus\_Code**

Enum for the HTTP status codes.

Values:

**HttpStatus\_Ok** = 200

**HttpStatus\_MultipleChoices** = 300

**HttpStatus\_MovedPermanently** = 301

**HttpStatus\_Found** = 302

**HttpStatus\_Unauthorized** = 401

## 2.4.4 ESP WebSocket Client

### Overview

The ESP WebSocket client is an implementation of [WebSocket protocol client](#) for ESP32

### Features

- supports WebSocket over TCP, SSL with mbedtls
- Easy to setup with URI
- Multiple instances (Multiple clients in one application)

### Configuration

#### URI

- Supports `ws`, `wss` schemes
- WebSocket samples:
  - `ws://websocket.org`: WebSocket over TCP, default port 80
  - `wss://websocket.org`: WebSocket over SSL, default port 443
- Minimal configurations:

```
const esp_websocket_client_config_t ws_cfg = {
    .uri = "ws://websocket.org",
};
```

- If there are any options related to the URI in `esp_websocket_client_config_t`, the option defined by the URI will be overridden. Sample:

```
const esp_websocket_client_config_t ws_cfg = {
    .uri = "ws://websocket.org:123",
    .port = 4567,
};
//WebSocket client will connect to websocket.org using port 4567
```

#### SSL

- Get certificate from server, example: `websocket.org openssl s_client -showcerts -connect websocket.org:443 </dev/null 2>/dev/null|openssl x509 -outform PEM >websocket_org.pem`
- Configuration:

```

const esp_websocket_client_config_t ws_cfg = {
    .uri = "wss://websocket.org",
    .cert_pem = (const char *)websocket_org_pem_start,
};

```

For more options on `esp_websocket_client_config_t`, please refer to API reference below

## Application Example

Simple WebSocket example that uses `esp_websocket_client` to establish a websocket connection and send/receive data with the [websocket.org](https://websocket.org) Server: [protocols/websocket](https://protocols.websocket.org).

## API Reference

### Header File

- `esp_websocket_client/include/esp_websocket_client.h`

### Functions

`esp_err_t esp_websocket_client_init (const esp_websocket_client_config_t *config)`

Start a WebSocket session This function must be the first function to call, and it returns a `esp_websocket_client_handle_t` that you must use as input to other functions in the interface. This call MUST have a corresponding call to `esp_websocket_client_destroy` when the operation is complete.

#### Return

- `esp_websocket_client_handle_t`
- NULL if any errors

#### Parameters

- [in] `config`: The configuration

`esp_err_t esp_websocket_client_set_uri (esp_websocket_client_handle_t client, const char *uri)`

Set URL for client, when performing this behavior, the options in the URL will replace the old ones Must stop the WebSocket client before set URI if the client has been connected.

Return `esp_err_t`

#### Parameters

- [in] `client`: The client
- [in] `uri`: The uri

`esp_err_t esp_websocket_client_start (esp_websocket_client_handle_t client)`

Open the WebSocket connection.

Return `esp_err_t`

#### Parameters

- [in] `client`: The client

`esp_err_t esp_websocket_client_stop (esp_websocket_client_handle_t client)`

Close the WebSocket connection.

Notes:

- Cannot be called from the websocket event handler

Return `esp_err_t`

#### Parameters

- [in] `client`: The client

`esp_err_t esp_websocket_client_destroy (esp_websocket_client_handle_t client)`

Destroy the WebSocket connection and free all resources. This function must be the last function to call for an



session. It is the opposite of the `esp_websocket_client_init` function and must be called with the same handle as input that a `esp_websocket_client_init` call returned. This might close all connections this handle has used.

Notes:

- Cannot be called from the websocket event handler

**Return** `esp_err_t`

**Parameters**

- [in] `client`: The client

int `esp_websocket_client_send` (*`esp_websocket_client_handle_t client`*, **const** char \**`data`*, int *`len`*, `TickType_t timeout`)

Generic write data to the WebSocket connection; defaults to binary send.

**Return**

- Number of data was sent
- (-1) if any errors

**Parameters**

- [in] `client`: The client
- [in] `data`: The data
- [in] `len`: The length
- [in] `timeout`: Write data timeout

int `esp_websocket_client_send_bin` (*`esp_websocket_client_handle_t client`*, **const** char \**`data`*, int *`len`*, `TickType_t timeout`)

Write binary data to the WebSocket connection (data send with WS OPCODE=02, i.e. binary)

**Return**

- Number of data was sent
- (-1) if any errors

**Parameters**

- [in] `client`: The client
- [in] `data`: The data
- [in] `len`: The length
- [in] `timeout`: Write data timeout

int `esp_websocket_client_send_text` (*`esp_websocket_client_handle_t client`*, **const** char \**`data`*, int *`len`*, `TickType_t timeout`)

Write textual data to the WebSocket connection (data send with WS OPCODE=01, i.e. text)

**Return**

- Number of data was sent
- (-1) if any errors

**Parameters**

- [in] `client`: The client
- [in] `data`: The data
- [in] `len`: The length
- [in] `timeout`: Write data timeout

bool `esp_websocket_client_is_connected` (*`esp_websocket_client_handle_t client`*)

Check the WebSocket connection status.

**Return**

- true
- false

**Parameters**

- [in] `client`: The client handle

*`esp_err_t`* `esp_websocket_register_events` (*`esp_websocket_client_handle_t client`*, *`esp_websocket_event_id_t event`*, *`esp_event_handler_t event_handler`*, void \**`event_handler_arg`*)

Register the Websocket Events.

**Return** `esp_err_t`

**Parameters**

- `client`: The client handle
- `event`: The event id
- `event_handler`: The callback function
- `event_handler_arg`: User context

**Structures****struct esp\_websocket\_event\_data\_t**

Websocket event data.

**Public Members****const char \*data\_ptr**

Data pointer

**int data\_len**

Data length

**uint8\_t op\_code**

Received opcode

**esp\_websocket\_client\_handle\_t client**

esp\_websocket\_client\_handle\_t context

**void \*user\_context**user\_data context, from [esp\\_websocket\\_client\\_config\\_t](#) user\_data**int payload\_len**

Total payload length, payloads exceeding buffer will be posted through multiple events

**int payload\_offset**

Actual offset for the data associated with this event

**struct esp\_websocket\_client\_config\_t**

Websocket client setup configuration.

**Public Members****const char \*uri**

Websocket URI, the information on the URI can be overrides the other fields below, if any

**const char \*host**

Domain or IP as string

**int port**

Port to connect, default depend on esp\_websocket\_transport\_t (80 or 443)

**const char \*username**

Using for Http authentication - Not supported for now

**const char \*password**

Using for Http authentication - Not supported for now

**const char \*path**

HTTP Path, if not set, default is /

**bool disable\_auto\_reconnect**

Disable the automatic reconnect function when disconnected

**void \*user\_context**

HTTP user data context

**int task\_prio**

Websocket task priority

int **task\_stack**  
Websocket task stack

int **buffer\_size**  
Websocket buffer size

const char \***cert\_pem**  
SSL Certification, PEM format as string, if the client requires to verify server

*esp\_websocket\_transport\_t* **transport**  
Websocket transport type, see 'esp\_websocket\_transport\_t'

char \***subprotocol**  
Websocket subprotocol

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

### Type Definitions

**typedef struct** esp\_websocket\_client \***esp\_websocket\_client\_handle\_t**

### Enumerations

**enum esp\_websocket\_event\_id\_t**  
Websocket Client events id.

*Values:*

**WEBSOCKET\_EVENT\_ANY** = -1

**WEBSOCKET\_EVENT\_ERROR** = 0

This event occurs when there are any errors during execution

**WEBSOCKET\_EVENT\_CONNECTED**

Once the Websocket has been connected to the server, no data exchange has been performed

**WEBSOCKET\_EVENT\_DISCONNECTED**

The connection has been disconnected

**WEBSOCKET\_EVENT\_DATA**

When receiving data from the server, possibly multiple portions of the packet

**WEBSOCKET\_EVENT\_MAX**

**enum esp\_websocket\_transport\_t**  
Websocket Client transport.

*Values:*

**WEBSOCKET\_TRANSPORT\_UNKNOWN** = 0x0

Transport unknown

**WEBSOCKET\_TRANSPORT\_OVER\_TCP**

Transport over tcp

**WEBSOCKET\_TRANSPORT\_OVER\_SSL**

Transport over ssl

## 2.4.5 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，一个用来监听 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，并释放相关联的内存和资源。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

```

/* 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, strlen(resp));
    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
     * httpd_req_recv() 只接收 char* 数据，但也可以是任意二进制数据（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 */
    char[100] content;

    /* 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, strlen(resp));
    return ESP_OK;
}

```

(下页继续)

```

/* 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)
{
    if (server) {
        /* 停止 httpd server */
        httpd_stop(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好自定义的回调函数, 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

```

/* 自定义函数, 用来释放上下文数据 */
void free_ctx_func(void *ctx)

```

(下页继续)

```

{
    /* 也可以是 free 以外的代码逻辑 */
    free(ctx);
}

esp_err_t adder_post_handler(httpd_req_t *req)
{
    /* 如果会话上下文还不存在则新建一个 */
    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。

## API 参考

### Header File

- [esp\\_http\\_server/include/esp\\_http\\_server.h](#)

### Functions

***esp\_err\_t* httpd\_register\_uri\_handler** (*httpd\_handle\_t* handle, **const** *httpd\_uri\_t* \*uri\_handler)  
Registers a URI handler.

Example usage:

```

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,

```

(下页继续)

```

    .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
    ....
}

```

**Note** URI handlers can be registered in real time as long as the server handle is valid.

#### Return

- `ESP_OK` : On successfully registering the handler
- `ESP_ERR_INVALID_ARG` : Null arguments
- `ESP_ERR_HTTPD_HANDLERS_FULL` : If no slots left for new handler
- `ESP_ERR_HTTPD_HANDLER_EXISTS` : If handler with same URI and method is already registered

#### Parameters

- `[in] handle`: handle to HTTPD server instance
- `[in] uri_handler`: pointer to handler that needs to be registered

*esp\_err\_t* `httpd_unregister_uri_handler` (*httpd\_handle\_t* handle, **const** char \*uri, *httpd\_method\_t* method)

Unregister a URI handler.

#### Return

- `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

#### Parameters

- `[in] handle`: handle to HTTPD server instance
- `[in] uri`: URI string
- `[in] method`: HTTP method

*esp\_err\_t* `httpd_unregister_uri` (*httpd\_handle\_t* handle, **const** char \*uri)

Unregister all URI handlers with the specified uri string.

#### Return

- `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

#### Parameters

- `[in] handle`: handle to HTTPD server instance
- `[in] uri`: uri string specifying all handlers that need to be deregistered

*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.

**Note** 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()`

#### Return

- `ESP_OK` : On successfully registering override
- `ESP_ERR_INVALID_ARG` : Null arguments

#### Parameters

- `[in] hd`: HTTPD instance handle
- `[in] sockfd`: Session socket FD

- [in] `recv_func`: The receive function to be set for this session

*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.

**Note** 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()`

**Return**

- `ESP_OK` : On successfully registering override
- `ESP_ERR_INVALID_ARG` : Null arguments

**Parameters**

- [in] `hd`: HTTPD instance handle
- [in] `sockfd`: Session socket FD
- [in] `send_func`: The send function to be set for this session

*esp\_err\_t* `httpd_sess_set_pending_override` (*httpd\_handle\_t* `hd`, `int` `sockfd`, *httpd\_pending\_func\_t* `pending_func`)

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.

**Note** 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()`

**Return**

- `ESP_OK` : On successfully registering override
- `ESP_ERR_INVALID_ARG` : Null arguments

**Parameters**

- [in] `hd`: HTTPD instance handle
- [in] `sockfd`: Session socket FD
- [in] `pending_func`: The receive function to be set for this session

`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()`.

**Note** This API is supposed to be called only from the context of a URI handler where `httpd_req_t*` request pointer is valid.

**Return**

- Socket descriptor : The socket descriptor for this request
- -1 : Invalid/NULL request pointer

**Parameters**

- [in] `r`: The request whose socket descriptor should be found

`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.

**Note**

- 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

**Return**

- Bytes : Number of bytes read into the buffer 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()

**Parameters**

- [in] r: The request being responded to
- [in] buf: Pointer to a buffer that the data will be read into
- [in] buf\_len: Length of the buffer

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.

**Note**

- 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.

**Return**

- Length : If field is found in the request URL
- Zero : Field not found / Invalid request / Null arguments

**Parameters**

- [in] r: The request being responded to
- [in] field: The header field to be searched in the request

*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.

**Note**

- 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

**Return**

- 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

**Parameters**

- [in] r: The request being responded to
- [in] field: The field to be searched in the header
- [out] val: Pointer to the buffer into which the value will be copied if the field is found
- [in] val\_size: Size of the user buffer "val"

size\_t **httpd\_req\_get\_url\_query\_len** (*httpd\_req\_t* \*r)

Get Query string length from the request URL.

**Note** This API is supposed to be called only from the context of a URI handler where *httpd\_req\_t*\* request pointer is valid

**Return**

- Length : Query is found in the request URL
- Zero : Query not found / Null arguments / Invalid request

**Parameters**

- [in] *r*: The request being responded to

*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.

**Note**

- 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

**Return**

- `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

**Parameters**

- [in] *r*: The request being responded to
- [out] *buf*: Pointer to the buffer into which the query string will be copied (if found)
- [in] *buf\_len*: Length of output buffer

*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`.

**Note**

- The components of URL query string (keys and values) are not URLdecoded. The user must check for ‘Content-Type’ field in the request headers and then depending upon the specified encoding (URLencoded 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.

**Return**

- `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_HTTPD_RESULT_TRUNC` : Value string truncated

**Parameters**

- [in] *qry*: Pointer to query string
- [in] *key*: The key to be searched in the query string
- [out] *val*: Pointer to the buffer into which the value will be copied if the key is found
- [in] *val\_size*: Size of the user buffer “*val*”

bool **httpd\_uri\_match\_wildcard** (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.

**Return** true if a match was found

**Parameters**

- [in] uri\_template: URI template (pattern)
- [in] uri\_to\_match: URI to be matched
- [in] match\_upto: how many characters of the URI buffer to test (there may be trailing query string etc.)

*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

**Note**

- 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.

**Return**

- 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

**Parameters**

- [in] r: The request being responded to
- [in] buf: Buffer from where the content is to be fetched
- [in] buf\_len: Length of the buffer, HTTPD\_RESP\_USE\_STRLEN to use `strlen()`

*esp\_err\_t* **httpd\_resp\_send\_chunk** (*httpd\_req\_t* \*r, const char \*buf, ssize\_t buf\_len)

API to send one HTTP chunk.

This API will send the data as an HTTP response to the request. This API 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.

**Return**

- ESP\_OK : On successfully sending the response packet chunk
- 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 pointer

**Parameters**

- [in] *r*: The request being responded to
- [in] *buf*: Pointer to a buffer that stores the data
- [in] *buf\_len*: Length of the buffer, HTTPD\_RESP\_USE\_STRLEN to use strlen()

**static esp\_err\_t httpd\_resp\_sendstr** (*httpd\_req\_t* \**r*, const char \**str*)

API to send a complete string as HTTP response.

This API simply calls `http_resp_send` with buffer length set to string length assuming the buffer contains a null terminated string

**Return**

- 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

**Parameters**

- [in] *r*: The request being responded to
- [in] *str*: String to be sent as response body

**static 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

**Return**

- 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

**Parameters**

- [in] *r*: The request being responded to
- [in] *str*: String to be sent as response body (NULL to finish response packet)

**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.

**Note**

- 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.

**Return**

- ESP\_OK : On success
- ESP\_ERR\_INVALID\_ARG : Null arguments
- ESP\_ERR\_HTTPD\_INVALID\_REQ : Invalid request pointer

**Parameters**

- [in] *r*: The request being responded to
- [in] *status*: The HTTP status code of this response

**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’.

**Note**

- 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.

**Return**

- `ESP_OK` : On success
- `ESP_ERR_INVALID_ARG` : Null arguments
- `ESP_ERR_HTTPD_INVALID_REQ` : Invalid request pointer

**Parameters**

- `[in] r`: The request being responded to
- `[in] type`: The Content Type of the response

`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.

**Note**

- 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.

**Return**

- `ESP_OK` : On successfully appending new header
- `ESP_ERR_INVALID_ARG` : Null arguments
- `ESP_ERR_HTTPD_RESP_HDR` : Total additional headers exceed max allowed
- `ESP_ERR_HTTPD_INVALID_REQ` : Invalid request pointer

**Parameters**

- `[in] r`: The request being responded to
- `[in] field`: The field name of the HTTP header
- `[in] value`: The value of this HTTP 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.

**Note**

- 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.

**Return**

- `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

**Parameters**

- `[in] req`: Pointer to the HTTP request for which the response needs to be sent
- `[in] error`: Error type to send
- `[in] msg`: Error message string (pass NULL for default message)

`static 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.

**Note**

- 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.

**Return**

- `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

**Parameters**

- `[in]` `r`: The request being responded to

**static** `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.

**Note**

- 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.

**Return**

- `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

**Parameters**

- `[in]` `r`: The request being responded to

**static** `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.

**Note**

- 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.

**Return**

- `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

**Parameters**

- `[in]` `r`: The request being responded to

**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.

**Note**

- 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()`

**Return**

- 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()

**Parameters**

- [in] `r`: The request being responded to
- [in] `buf`: Buffer from where the fully constructed packet is to be read
- [in] `buf_len`: Length of the buffer

`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.

**Return**

- `ESP_OK` : handler registered successfully
- `ESP_ERR_INVALID_ARG` : invalid error code or server handle

**Parameters**

- [in] `handle`: HTTP server handle
- [in] `error`: Error type
- [in] `handler_fn`: User implemented handler function (Pass NULL to unset any previously set handler)

`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:

```
//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;
}
```

**Return**

- `ESP_OK` : Instance created successfully
- `ESP_ERR_INVALID_ARG` : Null argument(s)
- `ESP_ERR_HTTPD_ALLOC_MEM` : Failed to allocate memory for instance
- `ESP_ERR_HTTPD_TASK` : Failed to launch server task

**Parameters**

- [in] `config`: Configuration for new instance of the server

- [out] handle: Handle to newly created instance of the server. NULL on error

*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:

```
// Function for stopping the webserver
void stop_webserver(httpd_handle_t server)
{
    // Ensure handle is non NULL
    if (server != NULL) {
        // Stop the httpd server
        httpd_stop(server);
    }
}
```

#### Return

- ESP\_OK : Server stopped successfully
- ESP\_ERR\_INVALID\_ARG : Handle argument is Null

#### Parameters

- [in] handle: Handle to server returned by httpd\_start

*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

**Note** 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.

#### Return

- ESP\_OK : On successfully queueing the work
- ESP\_FAIL : Failure in ctrl socket
- ESP\_ERR\_INVALID\_ARG : Null arguments

#### Parameters

- [in] handle: Handle to server returned by httpd\_start
- [in] work: Pointer to the function to be executed in the HTTPD' s context
- [in] arg: Pointer to the arguments that should be passed to this function

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.

#### Return

- void\* : Pointer to the context associated with this session
- NULL : Empty context / Invalid handle / Invalid socket fd

#### Parameters

- [in] handle: Handle to server returned by httpd\_start
- [in] sockfd: The socket descriptor for which the context should be extracted.

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.

#### Parameters

- [in] handle: Handle to server returned by httpd\_start
- [in] sockfd: The socket descriptor for which the context should be extracted.
- [in] ctx: Context object to assign to the session
- [in] free\_fn: 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.

See `httpd_sess_get_ctx()`

**Return**

- void\* : Pointer to the transport context associated with this session
- NULL : Empty context / Invalid handle / Invalid socket fd

**Parameters**

- [in] handle: Handle to server returned by `httpd_start`
- [in] sockfd: The socket descriptor for which the context should be extracted.

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.

See `httpd_sess_set_ctx()`

**Parameters**

- [in] handle: Handle to server returned by `httpd_start`
- [in] sockfd: The socket descriptor for which the context should be extracted.
- [in] ctx: Transport context object to assign to the session
- [in] free\_fn: 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)

**Return** global user context

**Parameters**

- [in] handle: Handle to server returned by `httpd_start`

void **httpd\_get\_global\_transport\_ctx** (*httpd\_handle\_t* handle)  
Get HTTPD global transport context (it was set in the server config struct)

**Return** global transport context

**Parameters**

- [in] handle: Handle to server returned by `httpd_start`

*esp\_err\_t* **httpd\_sess\_trigger\_close** (*httpd\_handle\_t* handle, int sockfd)  
Trigger an httpd session close externally.

**Note** Calling this API is only required in special circumstances wherein some application requires to close an httpd client session asynchronously.

**Return**

- 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

**Parameters**

- [in] handle: Handle to server returned by `httpd_start`
- [in] sockfd: The socket descriptor of the session to be closed

*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.

**Note** Calling this API is only necessary if the LRU Purge Enable option is enabled.

**Return**

- `ESP_OK` : Socket found and LRU counter updated
- `ESP_ERR_NOT_FOUND` : Socket not found
- `ESP_ERR_INVALID_ARG` : Null arguments

**Parameters**

- `[in] handle`: Handle to server returned by `httpd_start`
- `[in] sockfd`: The socket descriptor of the session for which LRU counter is to be updated

**Structures****struct httpd\_config**

HTTP Server Configuration Structure.

**Note** 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\_open\_sockets**

Max number of sockets/clients connected at any time

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

[\*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.

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 octext-stream

**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

**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.

**Note** 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

**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()

**Parameters**

- [in] `hd`: server instance
- [in] `sockfd`: session socket file descriptor
- [in] `buf`: buffer with bytes to send
- [in] `buf_len`: data size
- [in] `flags`: flags for the send() function

```
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.

**Note** 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

**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()

**Parameters**

- [in] `hd`: server instance
- [in] `sockfd`: session socket file descriptor
- [in] `buf`: buffer with bytes to send
- [in] `buf_len`: data size
- [in] `flags`: flags for the send() function

```
typedef int (*httpd_pending_func_t) (httpd_handle_t hd, int sockfd)
```

Prototype for HTTPDs low-level “get pending bytes” function.

**Note** 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.

**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()

**Parameters**

- [in] `hd`: server instance
- [in] `sockfd`: session socket file descriptor

```
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.

**Note**

- 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.

**Return**

- `ESP_OK` : error handled successful

- `ESP_FAIL` : failure indicates that the underlying socket needs to be closed

**Parameters**

- `[in] req`: HTTP request for which the error needs to be handled
- `[in] error`: Error type

```
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)

**Parameters**

- `[in] ctx`: 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.

**Return**

- `ESP_OK` : On success
- Any value other than `ESP_OK` will signal the server to close the socket immediately

**Parameters**

- `[in] hd`: server instance
- `[in] sockfd`: session socket file descriptor

```
typedef void (*httpd_close_func_t)(httpd_handle_t hd, int sockfd)
```

Function prototype for closing a session.

**Note** 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.

**Parameters**

- `[in] hd`: server instance
- `[in] sockfd`: 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.

**Return** true on match

**Parameters**

- `[in] reference_uri`: URI/template with respect to which the other URI is matched
- `[in] uri_to_match`: URI/template being matched to the reference URI/template
- `[in] match_upto`: 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`)

```
typedef struct httpd_config httpd_config_t
```

HTTP Server Configuration Structure.

**Note** 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.

**Parameters**

- `[in] arg`: 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:

```
HTTPD_500_INTERNAL_SERVER_ERROR = 0
HTTPD_501_METHOD_NOT_IMPLEMENTED
HTTPD_505_VERSION_NOT_SUPPORTED
HTTPD_400_BAD_REQUEST
HTTPD_404_NOT_FOUND
HTTPD_405_METHOD_NOT_ALLOWED
HTTPD_408_REQ_TIMEOUT
HTTPD_411_LENGTH_REQUIRED
HTTPD_414_URI_TOO_LONG
HTTPD_431_REQ_HDR_FIELDS_TOO_LARGE
HTTPD_ERR_CODE_MAX
```

## 2.4.6 HTTPS server

### Overview

This component is built on top of `esp_http_server`. The HTTPS server takes advantage of hooks and function overrides in the regular HTTP server to provide encryption using OpenSSL.

All documentation for `esp_http_server` applies also to a server you create this way.

### Used APIs

The following API of `esp_http_server` should not be used with `esp_https_server`, as they are used internally to handle secure sessions and to maintain internal state:

- “send”, “receive” and “pending” function overrides - secure socket handling
  - `httpd_sess_set_send_override()`
  - `httpd_sess_set_recv_override()`
  - `httpd_sess_set_pending_override()`
- “transport context” - both global and session
  - `httpd_sess_get_transport_ctx()` - returns SSL used for the session
  - `httpd_sess_set_transport_ctx()`
  - `httpd_get_global_transport_ctx()` - returns the shared SSL context
  - `httpd_config_t.global_transport_ctx`
  - `httpd_config_t.global_transport_ctx_free_fn`
  - `httpd_config_t.open_fn` - used to set up secure sockets

Everything else can be used without limitations.

### Usage

Please see the example [protocols/https\\_server](#) to learn how to set up a secure server.

Basically all you need is to generate a certificate, embed it in the firmware, and provide its pointers and lengths to the start function via the init struct.



The server can be started with or without SSL by changing a flag in the init struct - `httpd_ssl_config.transport_mode`. This could be used e.g. for testing or in trusted environments where you prefer speed over security.

## Performance

The initial session setup can take about two seconds, or more with slower clock speeds or more verbose logging. Subsequent requests through the open secure socket are much faster (down to under 100 ms).

## API Reference

### Header File

- [esp\\_https\\_server/include/esp\\_https\\_server.h](#)

### Functions

`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)

**Return** success

#### Parameters

- [inout] `config`: - server config, must not be const. Does not have to stay valid after calling this function.
- [out] `handle`: - storage for the server handle, must be a valid pointer

void `httpd_ssl_stop (httpd_handle_t handle)`

Stop the server. Blocks until the server is shut down.

#### Parameters

- [in] `handle`:

### Structures

`struct httpd_ssl_config`

HTTPS server config struct

Please use `HTTPD_SSL_CONFIG_DEFAULT()` to initialize it.

### Public Members

`httpd_config_t httpd`

Underlying HTTPD server config

Parameters like task stack size and priority can be adjusted here.

`const uint8_t *cacert_pem`

CA certificate

`size_t cacert_len`

CA certificate byte length

`const uint8_t *prvtkey_pem`

Private key

`size_t prvtkey_len`

Private key byte length

`httpd_ssl_transport_mode_t transport_mode`

Transport Mode (default secure)

`uint16_t port_secure`

Port used when transport mode is secure (default 443)

uint16\_t **port\_insecure**

Port used when transport mode is insecure (default 80)

### 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 ~ 40kB RAM 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 *httpd\_ssl\_config* httpd\_ssl\_config\_t**

### Enumerations

**enum httpd\_ssl\_transport\_mode\_t**

Values:

**HTTPD\_SSL\_TRANSPORT\_SECURE**

**HTTPD\_SSL\_TRANSPORT\_INSECURE**

## 2.4.7 ASIO port

### Overview

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

**ASIO documentation** Please refer to the original asio documentation at <https://think-async.com/Asio/Documentation>. Asio also comes with a number of examples which could be find under Documentation/Examples on that web site.

**Supported features** ESP platform port currently supports only network asynchronous socket operations; does not support serial port and ssl. Internal asio settings for ESP include - EXCEPTIONS: Supported, choice in menuconfig - SIGNAL, SIGACTION: Not supported - EPOLL, EVENTFD: Not supported - TYPEID: Disabled by default, but supported in toolchain and asio (provided stdlib recompiled with -frtti)

### Application Example

ESP examples are based on standard asio examples *examples/protocols/asio*: - udp\_echo\_server - tcp\_echo\_server - chat\_client - chat\_server Please refer to the specific example README.md for details

## 2.4.8 ESP-MQTT

### Overview

ESP-MQTT is an implementation of MQTT protocol client (MQTT is a lightweight publish/subscribe messaging protocol).

## Features

- supports MQTT over TCP, SSL with mbedtls, MQTT over Websocket, MQTT over Websocket Secure.
- Easy to setup with URI
- Multiple instances (Multiple clients in one application)
- Support subscribing, publishing, authentication, will messages, keep alive pings and all 3 QoS levels (it should be a fully functional client).

## Application Example

- [protocols/mqtt/tcp](#): MQTT over tcp, default port 1883
- [protocols/mqtt/ssl](#): MQTT over tcp, default port 8883
- [protocols/mqtt/ws](#): MQTT over Websocket, default port 80
- [protocols/mqtt/wss](#): MQTT over Websocket Secure, default port 443

## Configuration

### URI

- Currently support `mqtt`, `mqttps`, `ws`, `wss` schemes
- MQTT over TCP samples:
  - `mqtt://mqtt.eclipse.org`: MQTT over TCP, default port 1883:
  - `mqtt://mqtt.eclipse.org:1884` MQTT over TCP, port 1884:
  - `mqtt://username:password@mqtt.eclipse.org:1884` MQTT over TCP, port 1884, with username and password
- MQTT over SSL samples:
  - `mqttps://mqtt.eclipse.org`: MQTT over SSL, port 8883
  - `mqttps://mqtt.eclipse.org:8884`: MQTT over SSL, port 8884
- MQTT over Websocket samples:
  - `ws://mqtt.eclipse.org:80/mqtt`
- MQTT over Websocket Secure samples:
  - `wss://mqtt.eclipse.org:443/mqtt`
- Minimal configurations:

```
const esp_mqtt_client_config_t mqtt_cfg = {
    .uri = "mqtt://mqtt.eclipse.org",
    // .user_context = (void *)your_context
};
esp_mqtt_client_handle_t client = esp_mqtt_client_init(&mqtt_cfg);
esp_mqtt_client_register_event(client, ESP_EVENT_ANY_ID, mqtt_event_handler,
↵client);
esp_mqtt_client_start(client);
```

- Note: By default mqtt client uses event loop library to post related mqtt events (connected, subscribed, published, etc.)
- If there are any options related to the URI in `esp_mqtt_client_config_t`, the option defined by the URI will be overridden. Sample:

```
const esp_mqtt_client_config_t mqtt_cfg = {
    .uri = "mqtt://mqtt.eclipse.org:1234",
    .port = 4567,
};
//MQTT client will connect to mqtt.eclipse.org using port 4567
```

### SSL

- Get certificate from server, example: `mqtt.eclipse.org openssl s_client -showcerts -connect mqtt.eclipse.org:8883 </dev/null 2>/dev/null|openssl x509 -outform PEM >mqtt_eclipse_org.pem`
- Check the sample application: `examples/mqtt_ssl`
- Configuration:

```
const esp_mqtt_client_config_t mqtt_cfg = {
    .uri = "mqtts://mqtt.eclipse.org:8883",
    .event_handle = mqtt_event_handler,
    .cert_pem = (const char *)mqtt_eclipse_org_pem_start,
};
```

For more options on `esp_mqtt_client_config_t`, please refer to API reference below

### Change settings in Project Configuration Menu

:: idf.py menuconfig -> Component config -> ESP-MQTT Configuration

- `CONFIG_MQTT_PROTOCOL_311`: Enables 3.1.1 version of MQTT protocol
- `CONFIG_MQTT_TRANSPORT_SSL`, `CONFIG_MQTT_TRANSPORT_WEBSOCKET`: Enables specific MQTT transport layer, such as SSL, WEBSOCKET, WEBSOCKET\_SECURE
- `CONFIG_MQTT_CUSTOM_OUTBOX`: Disables default implementation of `mqtt_outbox`, so a specific implementation can be supplied

## API Reference

### Header File

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

### Functions

`esp_mqtt_client_handle_t esp_mqtt_client_init (const esp_mqtt_client_config_t *config)`

Creates mqtt client handle based on the configuration.

**Return** `mqtt_client_handle` if successfully created, `NULL` on error

#### Parameters

- `config`: mqtt configuration structure

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

Sets mqtt connection URI. This API is usually used to overrides the URI configured in `esp_mqtt_client_init`.

**Return** `ESP_FAIL` if URI parse error, `ESP_OK` on success

#### Parameters

- `client`: mqtt client handle
- `uri`:

`esp_err_t esp_mqtt_client_start (esp_mqtt_client_handle_t client)`

Starts mqtt client with already created client handle.

**Return** `ESP_OK` on success `ESP_ERR_INVALID_ARG` on wrong initialization `ESP_FAIL` on other error

#### Parameters

- `client`: mqtt client handle

`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.

**Return** `ESP_OK` on success `ESP_FAIL` if client is in invalid state

#### Parameters

- `client`: mqtt client handle

`esp_err_t esp_mqtt_client_disconnect (esp_mqtt_client_handle_t client)`

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

**Return** ESP\_OK on success

**Parameters**

- `client`: mqtt client handle

`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

**Return** ESP\_OK on success ESP\_FAIL if client is in invalid state

**Parameters**

- `client`: mqtt client handle

`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:

- Client must be connected to send subscribe message
- This API is could be executed from a user task or from a mqtt event callback i.e. internal mqtt task (API is protected by internal mutex, so it might block if a longer data receive operation is in progress.

**Return** message\_id of the subscribe message on success -1 on failure

**Parameters**

- `client`: mqtt client handle
- `topic`:
- `qos`:

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

Unsubscribe the client from defined topic.

Notes:

- Client must be connected to send unsubscribe message
- It is thread safe, please refer to `esp_mqtt_client_subscribe` for details

**Return** message\_id of the subscribe message on success -1 on failure

**Parameters**

- `client`: mqtt client handle
- `topic`:

`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 to send publish message (although it would drop all qos=0 messages, qos>1 messages would be enqueued)
- It is thread safe, please refer to `esp_mqtt_client_subscribe` for details

**Return** message\_id of the publish message (for QoS 0 message\_id will always be zero) on success. -1 on failure.

**Parameters**

- `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

`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

**Return** ESP\_OK

**Parameters**

- `client`: mqtt client handle

`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.

**Return** ESP\_ERR\_NO\_MEM if failed to allocate ESP\_OK on success

**Parameters**

- `client`: mqtt client handle
- `config`: mqtt configuration structure

`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.

**Return** ESP\_ERR\_NO\_MEM if failed to allocate ESP\_OK on success

**Parameters**

- `client`: mqtt client handle
- `event`: event type
- `event_handler`: handler callback
- `event_handler_arg`: handlers context

`int esp_mqtt_client_get_outbox_size(esp_mqtt_client_handle_t client)`

Get outbox size.

**Return** outbox size

**Parameters**

- `client`: mqtt client handle

## Structures

**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:

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

## Public Members

`esp_err_t esp_tls_last_esp_err`

last esp\_err code reported from esp-tls component

`int esp_tls_stack_err`

tls specific error code reported from underlying tls stack

`int esp_tls_cert_verify_flags`

tls flags reported from underlying tls stack during certificate verification

*esp\_mqtt\_error\_type\_t* **error\_type**

error type referring to the source of the error

*esp\_mqtt\_connect\_return\_code\_t* **connect\_return\_code**

connection refused error code reported from MQTT broker on connection

int **esp\_transport\_sock\_errno**

errno 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

void **\*user\_context**

User context passed from MQTT client config

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

**struct esp\_mqtt\_client\_config\_t**

MQTT client configuration structure

### Public Members

*mqtt\_event\_callback\_t* **event\_handle**

handle for MQTT events as a callback in legacy mode

*esp\_event\_loop\_handle\_t* **event\_loop\_handle**

handle for MQTT event loop library

const char **\*host**

MQTT server domain (ipv4 as string)

const char **\*uri**

Complete MQTT broker URI

**uint32\_t port**  
MQTT server port

**const char \*client\_id**  
default client id is ESP32\_CHIPID% where CHIPID% are last 3 bytes of MAC address in hex format

**const char \*username**  
MQTT username

**const char \*password**  
MQTT password

**const char \*lwt\_topic**  
LWT (Last Will and Testament) message topic (NULL by default)

**const char \*lwt\_msg**  
LWT message (NULL by default)

**int lwt\_qos**  
LWT message qos

**int lwt\_retain**  
LWT retained message flag

**int lwt\_msg\_len**  
LWT message length

**int disable\_clean\_session**  
mqtt clean session, default clean\_session is true

**int keepalive**  
mqtt keepalive, default is 120 seconds

**bool disable\_auto\_reconnect**  
this mqtt client will reconnect to server (when errors/disconnect). Set disable\_auto\_reconnect=true to disable

**void \*user\_context**  
pass user context to this option, then can receive that context in event->user\_context

**int task\_prio**  
MQTT task priority, default is 5, can be changed in make menuconfig

**int task\_stack**  
MQTT task stack size, default is 6144 bytes, can be changed in make menuconfig

**int buffer\_size**  
size of MQTT send/receive buffer, default is 1024 (only receive buffer size if out\_buffer\_size defined)

**const char \*cert\_pem**  
Pointer to certificate data in PEM or DER format for server verify (with SSL), default is NULL, not required to verify the server. PEM-format must have a terminating NULL-character. DER-format requires the length to be passed in cert\_len.

**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**  
Pointer to certificate data in PEM or DER format for SSL mutual authentication, default is NULL, not required if mutual authentication is not needed. If it is not NULL, also client\_key\_pem has to be provided. PEM-format must have a terminating NULL-character. DER-format requires the length to be passed in client\_cert\_len.

**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**  
 Pointer to private key data in PEM or DER format for SSL mutual authentication, default is NULL, not required if mutual authentication is not needed. If it is not NULL, also `client_cert_pem` has to be provided. PEM-format must have a terminating NULL-character. DER-format requires the length to be passed in `client_key_len`

**size\_t client\_key\_len**  
 Length of the buffer pointed to by `client_key_pem`. May be 0 for null-terminated pem

**esp\_mqtt\_transport\_t transport**  
 overrides URI transport

**int refresh\_connection\_after\_ms**  
 Refresh connection after this value (in milliseconds)

**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). If not NULL and server/client certificates are NULL, PSK is enabled

**bool use\_global\_ca\_store**  
 Use a global `ca_store` for all the connections in which this bool is set.

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

**const char \*\*alpn\_protos**  
 NULL-terminated list of supported application protocols to be used for ALPN

**const char \*clientkey\_password**  
 Client key decryption password string

**int clientkey\_password\_len**  
 String length of the password pointed to by `clientkey_password`

**esp\_mqtt\_protocol\_ver\_t protocol\_ver**  
 MQTT protocol version used for connection, defaults to value from `menuconfig`

**int out\_buffer\_size**  
 size of MQTT output buffer. If not defined, both output and input buffers have the same size defined as `buffer_size`

**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

**bool use\_secure\_element**  
 enable secure element for enabling SSL connection

**void \*ds\_data**  
 carrier of handle for digital signature parameters

**int network\_timeout\_ms**  
 Abort network operation if it is not completed after this value, in milliseconds (defaults to 10s)

**bool disable\_keepalive**  
 Set `disable_keepalive=true` to turn off keep-alive mechanism, false by default (keepalive is active by default). Note: setting the config value `keepalive` to 0 doesn't disable keepalive feature, but uses a default keepalive period

## 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 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_esp_err, 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 esp_mqtt_event_t *esp_mqtt_event_handle_t
```

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

**Enumerations**

```
enum esp_mqtt_event_id_t
```

MQTT event types.

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

- `user_context` - user data from `esp_mqtt_client_config_t`
- `client` - mqtt client handle
- various other data depending on event type

Values:

**MQTT\_EVENT\_ANY** = -1

**MQTT\_EVENT\_ERROR** = 0

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

**MQTT\_EVENT\_CONNECTED**

connected event, additional context: `session_present` flag

**MQTT\_EVENT\_DISCONNECTED**

disconnected event

**MQTT\_EVENT\_SUBSCRIBED**

subscribed event, additional context: `msg_id`

**MQTT\_EVENT\_UNSUBSCRIBED**

unsubscribed event

**MQTT\_EVENT\_PUBLISHED**

published event, additional context: `msg_id`

**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 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.

**MQTT\_EVENT\_BEFORE\_CONNECT**

The event occurs before connecting

**enum esp\_mqtt\_connect\_return\_code\_t**

MQTT connection error codes propagated via ERROR event

*Values:***MQTT\_CONNECTION\_ACCEPTED = 0**

Connection accepted

**MQTT\_CONNECTION\_REFUSE\_PROTOCOL**

MQTT connection refused reason: Wrong protocol

**MQTT\_CONNECTION\_REFUSE\_ID\_REJECTED**

MQTT connection refused reason: ID rejected

**MQTT\_CONNECTION\_REFUSE\_SERVER\_UNAVAILABLE**

MQTT connection refused reason: Server unavailable

**MQTT\_CONNECTION\_REFUSE\_BAD\_USERNAME**

MQTT connection refused reason: Wrong user

**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:***MQTT\_ERROR\_TYPE\_NONE = 0****MQTT\_ERROR\_TYPE\_TCP\_TRANSPORT****MQTT\_ERROR\_TYPE\_CONNECTION\_REFUSED****enum esp\_mqtt\_transport\_t***Values:***MQTT\_TRANSPORT\_UNKNOWN = 0x0****MQTT\_TRANSPORT\_OVER\_TCP**

MQTT over TCP, using scheme: mqtt

**MQTT\_TRANSPORT\_OVER\_SSL**

MQTT over SSL, using scheme: mqttssl

**MQTT\_TRANSPORT\_OVER\_WS**

MQTT over Websocket, using scheme: ws

**MQTT\_TRANSPORT\_OVER\_WSS**

MQTT over Websocket Secure, using scheme: wss

**enum esp\_mqtt\_protocol\_ver\_t**

MQTT protocol version used for connection

*Values:***MQTT\_PROTOCOL\_UNDEFINED = 0****MQTT\_PROTOCOL\_V\_3\_1****MQTT\_PROTOCOL\_V\_3\_1\_1**

## 2.4.9 ESP-Modbus

## Overview

The Modbus serial communication protocol is de facto standard protocol widely used to connect industrial electronic devices. Modbus allows communication among many devices connected to the same network, for example, a system that measures temperature and humidity and communicates the results to a computer. The Modbus protocol uses several types of data: Holding Registers, Input Registers, Coils (single bit output), Discrete Inputs. Versions of the Modbus protocol exist for serial port and for Ethernet and other protocols that support the Internet protocol suite. There are many variants of Modbus protocols, some of them are:

- `Modbus RTU` —This is used in serial communication and makes use of a compact, binary representation of the data for protocol communication. The RTU format follows the commands/data with a cyclic redundancy check checksum as an error check mechanism to ensure the reliability of data. Modbus RTU is the most common implementation available for Modbus. A Modbus RTU message must be transmitted continuously without inter-character hesitations. Modbus messages are framed (separated) by idle (silent) periods. The RS-485 interface communication is usually used for this type.
- `Modbus ASCII` —This is used in serial communication and makes use of ASCII characters for protocol communication. The ASCII format uses a longitudinal redundancy check checksum. Modbus ASCII messages are framed by leading colon ( “:” ) and trailing newline (CR/LF).
- `Modbus TCP/IP` or `Modbus TCP` —This is a Modbus variant used for communications over TCP/IP networks, connecting over port 502. It does not require a checksum calculation, as lower layers already provide checksum protection.

**Modbus common interface API overview** The API functions below provide common functionality to setup Modbus stack for slave and master implementation accordingly. ISP-IDF supports Modbus serial slave and master protocol stacks and provides `modbus_controller` interface API to interact with user application.

*esp\_err\_t* `mbc_slave_init` (`mb_port_type_t port_type`, `void **handler`)

Initialize Modbus controller and stack.

### Return

- `ESP_OK` Success
- `ESP_ERR_NO_MEM` Parameter error

### Parameters

- [out] `handler`: handler(pointer) to master data structure
- [in] `port_type`: type of stack

*esp\_err\_t* `mbc_master_init` (`mb_port_type_t port_type`, `void **handler`)

Initialize Modbus controller and stack.

### Return

- `ESP_OK` Success
- `ESP_ERR_NO_MEM` Parameter error

### Parameters

- [out] `handler`: handler(pointer) to master data structure
- [in] `port_type`: the type of port

The function initializes the Modbus controller interface and its active context (tasks, RTOS objects and other resources).

*esp\_err\_t* `mbc_slave_setup` (`void *comm_info`)

Set Modbus communication parameters for the controller.

### Return

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Incorrect parameter data

### Parameters

- `comm_info`: Communication parameters structure.

*esp\_err\_t* `mbc_master_setup` (`void *comm_info`)

Set Modbus communication parameters for the controller.

### Return

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Incorrect parameter data

**Parameters**

- `comm_info`: Communication parameters structure.

The function is used to setup communication parameters of the Modbus stack. See the Modbus controller API documentation for more information.

*mbc\_slave\_set\_descriptor()*: Initialization of slave descriptor.

*mbc\_master\_set\_descriptor()*: Initialization of master descriptor.

The Modbus stack uses parameter description tables (descriptors) for communication. These are different for master and slave implementation of stack and should be assigned by the API call before start of communication.

*esp\_err\_t* **mbc\_slave\_start** (void)

Start Modbus communication stack.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Modbus stack start error

*esp\_err\_t* **mbc\_master\_start** (void)

Start Modbus communication stack.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_ARG Modbus stack start error

Modbus controller start function. Starts stack and interface and allows communication.

*esp\_err\_t* **mbc\_slave\_destroy** (void)

Destroy Modbus controller and stack.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_STATE Parameter error

*esp\_err\_t* **mbc\_master\_destroy** (void)

Destroy Modbus controller and stack.

**Return**

- ESP\_OK Success
- ESP\_ERR\_INVALID\_STATE Parameter error

This function stops Modbus communication stack and destroys controller interface.

There are some configurable parameters of modbus\_controller interface and Modbus stack that can be configured using KConfig values in “Modbus configuration” menu. The most important option in KConfig menu is “Selection of Modbus stack support mode” that allows to select master or slave stack for implementation. See the examples for more information about how to use these API functions.

**Modbus serial slave interface API overview** The slave stack requires the user defined structures which represent Modbus parameters accessed by stack. These structures should be prepared by user and be assigned to the modbus\_controller interface using *mbc\_slave\_set\_descriptor()* API call before start of communication. The interface API functions below are used for Modbus slave application:

*esp\_err\_t* **mbc\_slave\_set\_descriptor** (mb\_register\_area\_descriptor\_t *descr\_data*)

Set Modbus area descriptor.

**Return**

- ESP\_OK: The appropriate descriptor is set
- ESP\_ERR\_INVALID\_ARG: The argument is incorrect

**Parameters**

- `descr_data`: Modbus registers area descriptor structure

The function initializes Modbus communication descriptors for each type of Modbus register area (Holding Registers, Input Registers, Coils (single bit output), Discrete Inputs). Once areas are initialized and the `mbc_slave_start()` API is called the Modbus stack can access the data in user data structures by request from master. See the `mb_register_area_descriptor_t` and example for more information.

`mb_event_group_t mbc_slave_check_event (mb_event_group_t group)`

Wait for specific event on parameter change.

**Return**

- `mb_event_group_t` event bits triggered

**Parameters**

- `group`: Group event bit mask to wait for change

The blocking call to function waits for event specified in the input parameter as event mask. Once master access the parameter and event mask matches the parameter the application task will be unblocked and function will return `ESP_OK`. See the `mb_event_group_t` for more information about Modbus event masks.

`esp_err_t mbc_slave_get_param_info (mb_param_info_t *reg_info, uint32_t timeout)`

Get parameter information.

**Return**

- `ESP_OK` Success
- `ESP_ERR_TIMEOUT` Can not get data from parameter queue or queue overflow

**Parameters**

- `[out] reg_info`: parameter info structure
- `timeout`: Timeout in milliseconds to read information from parameter queue

The function gets information about accessed parameters from modbus controller event queue. The KConfig ‘`CONFIG_FMB_CONTROLLER_NOTIFY_QUEUE_SIZE`’ key can be used to configure the notification queue size. The timeout parameter allows to specify timeout for waiting notification. The `mb_param_info_t` structure contain information about accessed parameter.

**Modbus serial master interface API overview** The Modbus master implementation requires parameter description table be defined before start of stack. This table describes characteristics (physical parameters like temperature, humidity, etc.) and links them to Modbus registers in specific slave device in the Modbus segment. The table has to be assigned to the modbus controller interface using `mbc_master_set_descriptor()` API call before start of communication.

Below are the interface API functions that are used to setup and use Modbus master stack from user application and can be executed in next order:

`esp_err_t mbc_master_set_descriptor (const mb_parameter_descriptor_t *descriptor, const uint16_t num_elements)`

Assign parameter description table for Modbus controller interface.

**Return**

- `esp_err_t ESP_OK` - set descriptor successfully
- `esp_err_t ESP_ERR_INVALID_ARG` - invalid argument in function call

**Parameters**

- `[in] descriptor`: pointer to parameter description table
- `num_elements`: number of elements in the table

Assigns parameter description table for Modbus controller interface. The table has to be prepared by user according to particular

`esp_err_t mbc_master_send_request (mb_param_request_t *request, void *data_ptr)`

Send data request as defined in parameter request, waits response from slave and returns status of command execution. This function provides standard way for read/write access to Modbus devices in the network.

**Return**

- `esp_err_t ESP_OK` - request was successful
- `esp_err_t ESP_ERR_INVALID_ARG` - invalid argument of function
- `esp_err_t ESP_ERR_INVALID_RESPONSE` - an invalid response from slave
- `esp_err_t ESP_ERR_TIMEOUT` - operation timeout or no response from slave

- `esp_err_t ESP_ERR_NOT_SUPPORTED` - the request command is not supported by slave
- `esp_err_t ESP_FAIL` - slave returned an exception or other failure

#### Parameters

- `[in] request`: pointer to request structure of type `mb_param_request_t`
- `[in] data_ptr`: pointer to data buffer to send or received data (dependent of command field in request)

This function sends data request as defined in parameter request, waits response from corresponded slave and returns status of command execution. This function provides a standard way for read/write access to Modbus devices in the network.

```
esp_err_t mbc_master_get_cid_info(uint16_t cid, const mb_parameter_descriptor_t
                                 **param_info)
```

Get information about supported characteristic defined as cid. Uses parameter description table to get this information. The function will check if characteristic defined as a cid parameter is supported and returns its description in param\_info. Returns `ESP_ERR_NOT_FOUND` if characteristic is not supported.

#### Return

- `esp_err_t ESP_OK` - request was successful and buffer contains the supported characteristic name
- `esp_err_t ESP_ERR_INVALID_ARG` - invalid argument of function
- `esp_err_t ESP_ERR_NOT_FOUND` - the characteristic (cid) not found
- `esp_err_t ESP_FAIL` - unknown error during lookup table processing

#### Parameters

- `[in] cid`: characteristic id
- `param_info`: pointer to pointer of characteristic data.

The function gets information about supported characteristic defined as cid. It will check if characteristic is supported and returns its description.

```
esp_err_t mbc_master_get_parameter(uint16_t cid, char *name, uint8_t *value, uint8_t *type)
```

Read parameter from modbus slave device whose name is defined by name and has cid. The additional data for request is taken from parameter description (lookup) table.

#### Return

- `esp_err_t ESP_OK` - request was successful and value buffer contains representation of actual parameter data from slave
- `esp_err_t ESP_ERR_INVALID_ARG` - invalid argument of function
- `esp_err_t ESP_ERR_INVALID_RESPONSE` - an invalid response from slave
- `esp_err_t ESP_ERR_INVALID_STATE` - invalid state during data processing or allocation failure
- `esp_err_t ESP_ERR_TIMEOUT` - operation timed out and no response from slave
- `esp_err_t ESP_ERR_NOT_SUPPORTED` - the request command is not supported by slave
- `esp_err_t ESP_ERR_NOT_FOUND` - the parameter is not found in the parameter description table
- `esp_err_t ESP_FAIL` - slave returned an exception or other failure

#### Parameters

- `[in] cid`: id of the characteristic for parameter
- `[in] name`: pointer into string name (key) of parameter (null terminated)
- `[out] value`: pointer to data buffer of parameter
- `[out] type`: parameter type associated with the name returned from parameter description table.

The function reads data of characteristic defined in parameters from Modbus slave device and returns its data. The additional data for request is taken from parameter description table.

```
esp_err_t mbc_master_set_parameter(uint16_t cid, char *name, uint8_t *value, uint8_t *type)
```

Set characteristic's value defined as a name and cid parameter. The additional data for cid parameter request is taken from master parameter lookup table.

#### Return

- `esp_err_t ESP_OK` - request was successful and value was saved in the slave device registers
- `esp_err_t ESP_ERR_INVALID_ARG` - invalid argument of function
- `esp_err_t ESP_ERR_INVALID_RESPONSE` - an invalid response from slave during processing of parameter
- `esp_err_t ESP_ERR_INVALID_STATE` - invalid state during data processing or allocation failure
- `esp_err_t ESP_ERR_TIMEOUT` - operation timed out and no response from slave

- `esp_err_t ESP_ERR_NOT_SUPPORTED` - the request command is not supported by slave
- `esp_err_t ESP_FAIL` - slave returned an exception or other failure

#### Parameters

- `[in] cid`: id of the characteristic for parameter
- `[in] name`: pointer into string name (key) of parameter (null terminated)
- `[out] value`: pointer to data buffer of parameter (actual representation of json value field in binary form)
- `[out] type`: pointer to parameter type associated with the name returned from parameter lookup table.

The function writes characteristic's value defined as a name and cid parameter in corresponded slave device. The additional data for parameter request is taken from master parameter description table.

### Application Example

The examples below use the FreeModbus library port for slave and master implementation accordingly. The selection of stack is performed through KConfig menu “Selection of Modbus stack support mode” and related configuration keys.

[protocols/modbus\\_slave](#)

[protocols/modbus\\_master](#)

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

### Modbus Readme File

# Modbus assumptions, dependencies and constraints

1. Current implementation of Modbus has limitation: maximum frame size should be less than 120 bytes (default UART receive FIFO full threshold value). This limitation is removed in ESP-IDF v4.2.
2. The only one instance of Modbus port can be initialized at the same time.

## 2.4.10 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:

```
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, 0xeb, 0x31, 0x4a, 0x1e, 0x98, 0x3d
            }
        }
    },
    .handlers = {
        /* User defined handler functions */
        .get_prop_values = get_property_values,
```

(下页继续)



```

        .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:

```

/* Set the configuration */
httpd_ssl_config_t https_conf = HTTPD_SSL_CONFIG_DEFAULT();

/* Load server certificate */
extern const unsigned char cacert_pem_start[] asm("_binary_cacert_pem_
↪start");
extern const unsigned char cacert_pem_end[]   asm("_binary_cacert_pem_end
↪");
https_conf.cacert_pem = cacert_pem_start;
https_conf.cacert_len = cacert_pem_end - cacert_pem_start;

/* Load server private key */
extern const unsigned char prvtkey_pem_start[] asm("_binary_prvtkey_pem_
↪start");
extern const unsigned char prvtkey_pem_end[]   asm("_binary_prvtkey_pem_
↪end");
https_conf.prvtkey_pem = prvtkey_pem_start;
https_conf.prvtkey_len = prvtkey_pem_end - prvtkey_pem_start;

esp_local_ctrl_config_t config = {
    .transport = ESP_LOCAL_CTRL_TRANSPORT_HTTPD,
    .transport_config = {
        .httpd = &https_conf
    },
    .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));

```

### 2.4.11 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 upto the application, hence they may be used as enumerations, bit-fields, 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 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.

```
/* 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.

```
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.

```
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 {
```

(下页继续)

(续上页)

```

        ESP_LOGI(TAG, "Setting %s", props[i].name);

        /* 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](#)

## 2.4.12 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:

表 1: Endpoints provided by ESP Local Control

Endpoint Name (BLE + GATT Server)	URI (HTTPS Server + mDNS)	Description
esp_local_ctrl	<a href="https://&lt;ip&gt;:&lt;port&gt;/.local/esp_local_ctrl/version">https://&lt;ip&gt;:&lt;port&gt;/.local/esp_local_ctrl/version</a>	Endpoint used for retrieving version string
esp_local_ctrl	<a href="https://&lt;ip&gt;:&lt;port&gt;/.local/esp_local_ctrl/control">https://&lt;ip&gt;:&lt;port&gt;/.local/esp_local_ctrl/control</a>	Endpoint used for sending / receiving control messages

## API Reference

### Header File

- [esp\\_local\\_ctrl/include/esp\\_local\\_ctrl.h](#)

### Functions

**const esp\_local\_ctrl\_transport\_t \*esp\_local\_ctrl\_get\_transport\_ble()**

Function for obtaining BLE transport mode.

**const** *esp\_local\_ctrl\_transport\_t* \***esp\_local\_ctrl\_get\_transport\_httpd** ()

Function for obtaining HTTPD transport mode.

*esp\_err\_t* **esp\_local\_ctrl\_start** (const *esp\_local\_ctrl\_config\_t* \**config*)

Start local control service.

**Return**

- ESP\_OK : Success
- ESP\_FAIL : Failure

**Parameters**

- [in] *config*: Pointer to configuration structure

*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*

**Return**

- ESP\_OK : Success
- ESP\_FAIL : Failure

**Parameters**

- [in] *prop*: Property description structure

*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.

**Return**

- ESP\_OK : Success
- ESP\_ERR\_NOT\_FOUND : Failure

**Parameters**

- [in] *name*: Name of the property to remove

**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

**Return**

- Pointer to property
- NULL if not found

**Parameters**

- [in] *name*: Name of the property to find

*esp\_err\_t* **esp\_local\_ctrl\_set\_handler** (const char \**ep\_name*, *protocomm\_req\_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.

**Note** 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* ().

**Return**

- ESP\_OK : Success
- ESP\_FAIL : Failure

**Parameters**

- [in] *ep\_name*: Name of the endpoint
- [in] *handler*: Endpoint handler function

- [in] `user_ctx`: User data

### 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.

**Note** If any of the properties have fixed sizes, the size field of corresponding element in `prop_values` need to be set

**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

**Parameters**

- [in] props\_count: Total elements in the props array
- [in] props: Array of properties, the current values for which have been requested by the client
- [out] prop\_values: Array of empty property values, the elements of which need to be populated with the current values of those properties specified by props argument
- [in] usr\_ctx: This provides value of the `usr_ctx` field of `esp_local_ctrl_handlers_t` structure

**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.

**Note** 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.

**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

**Parameters**

- [in] props\_count: Total elements in the props array
- [in] props: Array of properties, the values for which the client requests to change
- [in] prop\_values: Array of property values, the elements of which need to be used for updating those properties specified by props argument
- [in] usr\_ctx: This provides value of the `usr_ctx` field of `esp_local_ctrl_handlers_t` structure

**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.

`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.

**struct esp\_local\_ctrl\_config**

Configuration structure to pass to `esp_local_ctrl_start()`

### Public Members

**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\_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`.

**typedef struct 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`.

**typedef struct 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 struct esp\_local\_ctrl\_config esp\_local\_ctrl\_config\_t**

Configuration structure to pass to `esp_local_ctrl_start()`

此 API 部分的示例代码在 ESP-IDF 示例工程的 `protocols` 目录下提供。

### 2.4.13 IP 网络层协议

IP 网络层协议（应用层协议之下）的文档位于[连网 API](#)。

## 2.5 配网 API

### 2.5.1 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.



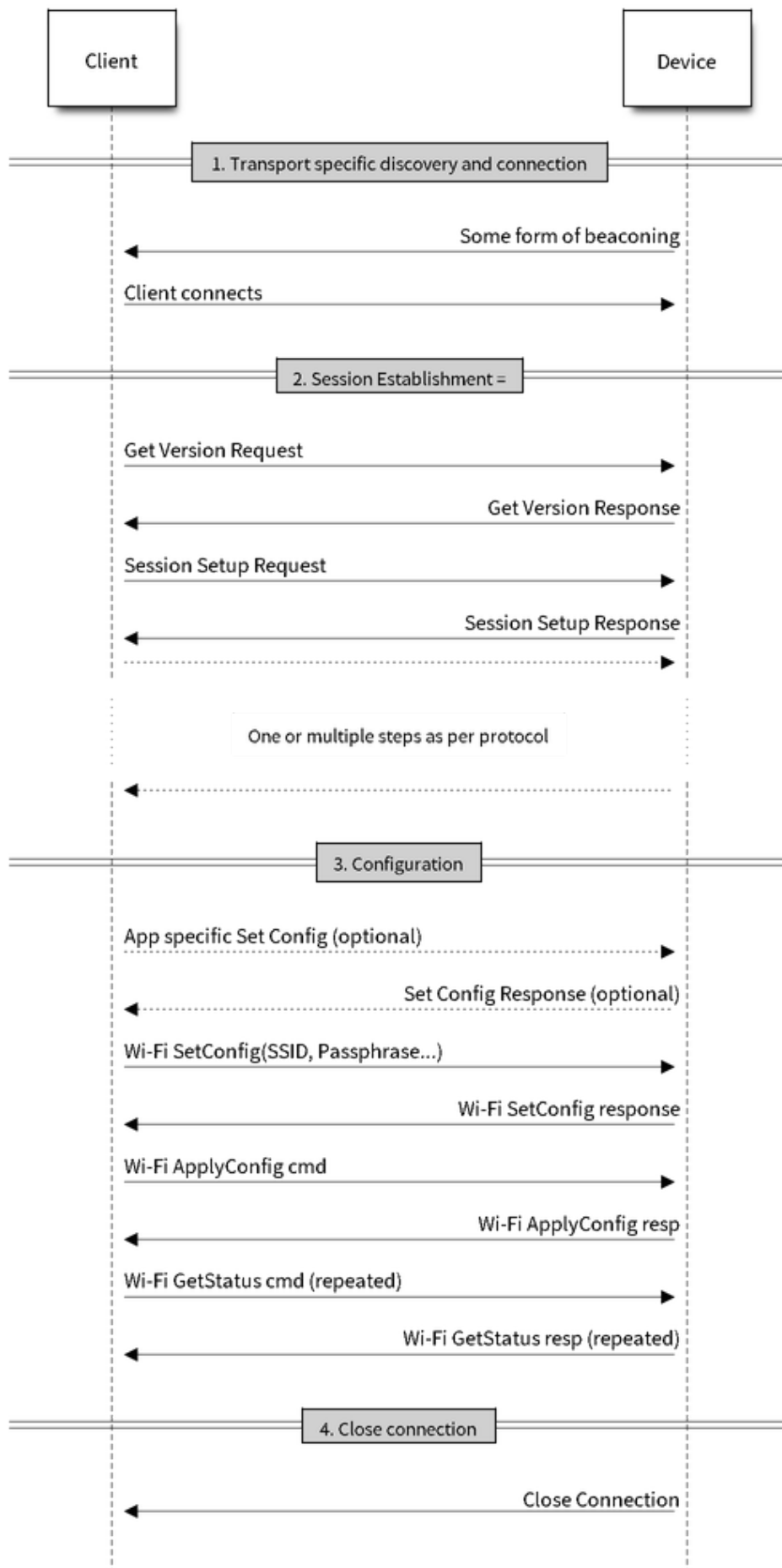


图 23: Typical Provisioning Process

## 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 provision 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.

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 two 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

Security1 scheme details are shown in the below sequence diagram

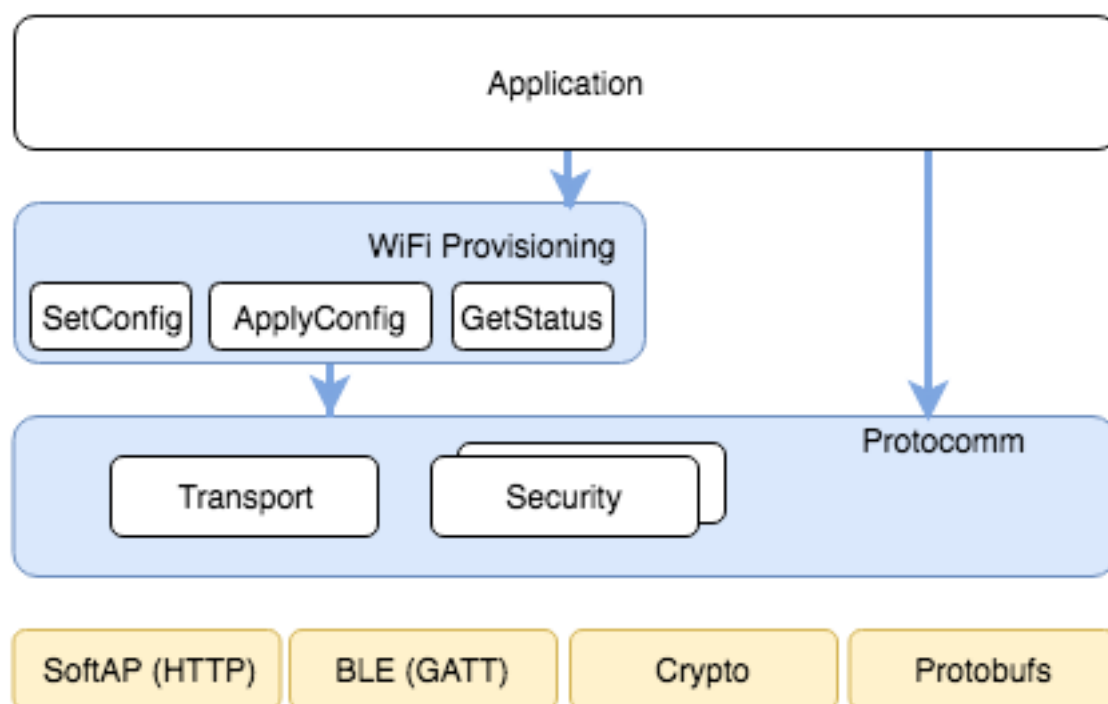


图 24: Unified Provisioning Architecture

### Sample Code

Please refer to *Protocol Communication* and *Wi-Fi Provisioning* for API guides and code snippets on example usage. Various use case implementations can be found as examples under [provisioning](#).

### Provisioning Tools

Provisioning applications are available for various platforms, along with source code:

- Android : [esp-idf-provisioning-android](#)
- iOS : [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.

## 2.5.2 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)
- Proof-of-possession (support with `protocomm_security1` only)

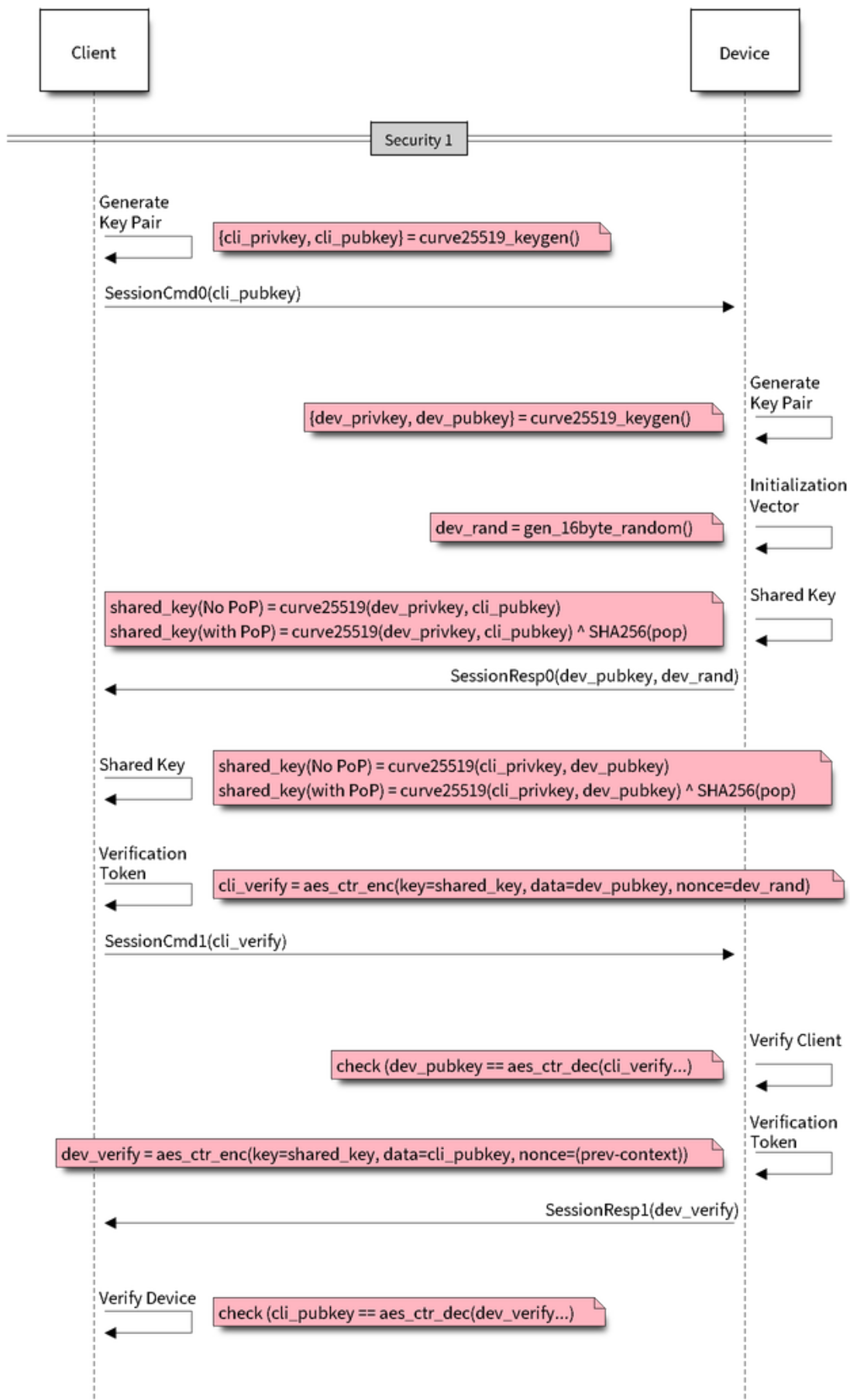


图 25: Security1

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 (only for protocomm\_security1) by performing the two way handshake. See *Unified Provisioning* for more details about the secure handshake logic.

### Transport Example (SoftAP + HTTP) with Security 1

For complete example see [provisioning/softap\\_prov](#)

```

/* 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;
}

/* 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 = PROTOCOMM_HTTPD_DEFAULT_CONFIG()
        }
    };

    /* Start protocomm server on top of HTTP */
    protocomm_httpd_start(pc, &pc_config);

    /* Create Proof of Possession object from pop_string. It must be valid
     * throughout the scope of protocomm endpoint. This need not be
     ↪static,
     * ie. could be dynamically allocated and freed at the time of
     ↪endpoint
     * removal */
    const static protocomm_security_pop_t pop_obj = {
        .data = (const uint8_t *) strdup(pop_string),
        .len = strlen(pop_string)
    }
}

```

(下页继续)

(续上页)

```

};

/* 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,
↳&pop_obj);

/* Private data passed to the endpoint must be valid throughout the
↳scope
 * 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);

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 complete example see [provisioning/ble\\_prov](#)

```

/* 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}
    };
}

```

(下页继续)

```

/* Config for protocomm_ble_start() */
protocomm_ble_config_t config = {
    .service_uuid = {
        /* LSB <-----
        * -----> MSB */
        0xfb, 0x34, 0x9b, 0x5f, 0x80, 0x00, 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);

    protocomm_delete(pc);
}

```

## API Reference

### Header File

- [protocomm/include/common/protocomm.h](#)

### Functions

***protocomm\_t*\*protocomm\_new()**

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.

#### Return

- 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().

#### Parameters

- [in] pc: 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.

#### Note

- 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.

#### Return

- 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

#### Parameters

- [in] pc: Pointer to the protocomm instance
- [in] ep\_name: Endpoint identifier(name) string
- [in] h: Endpoint handler function
- [in] priv\_data: Pointer to private data to be passed as a parameter to the handler function on call. Pass NULL if not needed.

*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.

#### Note

- This function internally calls the registered `remove_endpoint()` function which is a member of the `protocomm_t` instance structure.

#### Return

- ESP\_OK : Success
- ESP\_ERR\_NOT\_FOUND : Endpoint with specified name doesn't exist
- ESP\_ERR\_INVALID\_ARG : Null instance/name arguments

#### Parameters

- [in] pc: Pointer to the protocomm instance
- [in] ep\_name: Endpoint identifier(name) string

*esp\_err\_t* **protocomm\_open\_session** (*protocomm\_t* \*pc, uint32\_t session\_id)

Allocates internal resources for new transport session.

#### Note

- An endpoint must be bound to a valid protocomm instance, created using `protocomm_new()`.

#### Return

- ESP\_OK : Request handled successfully
- ESP\_ERR\_NO\_MEM : Error allocating internal resource
- ESP\_ERR\_INVALID\_ARG : Null instance/name arguments

#### Parameters

- [in] pc: Pointer to the protocomm instance
- [in] session\_id: Unique ID for a communication session

*esp\_err\_t* **protocomm\_close\_session** (*protocomm\_t* \*pc, uint32\_t session\_id)

Frees internal resources used by a transport session.

#### Note

- An endpoint must be bound to a valid protocomm instance, created using `protocomm_new()`.

#### Return

- ESP\_OK : Request handled successfully
- ESP\_ERR\_INVALID\_ARG : Null instance/name arguments

#### Parameters



- [in] `pc`: Pointer to the `protocomm` instance
- [in] `session_id`: Unique ID for a communication session

`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.

#### Note

- An endpoint must be bound to a valid `protocomm` instance, created using `protocomm_new()`.
- Resulting output buffer must be deallocated by the caller.

#### Return

- `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

#### Parameters

- [in] `pc`: Pointer to the `protocomm` instance
- [in] `ep_name`: Endpoint identifier(name) string
- [in] `session_id`: Unique ID for a communication session
- [in] `inbuf`: Input buffer contains input request data which is to be processed by the registered handler
- [in] `inlen`: Length of the input buffer
- [out] `outbuf`: Pointer to internally allocated output buffer, where the resulting response data output from the registered handler is to be stored
- [out] `outlen`: Buffer length of the allocated output buffer

`esp_err_t` **protocomm\_set\_security** (`protocomm_t` \*`pc`, `const` char \*`ep_name`, `const` `protocomm_security_t` \*`sec`, `const` `protocomm_security_pop_t` \*`pop`)

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.

#### Note

- 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` are readily available.

#### Return

- `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

#### Parameters

- [in] `pc`: Pointer to the `protocomm` instance
- [in] `ep_name`: Endpoint identifier(name) string
- [in] `sec`: Pointer to endpoint security instance
- [in] `pop`: Pointer to proof of possession for authenticating a client

`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.

#### Return

- `ESP_OK` : Success
- `ESP_ERR_NOT_FOUND` : Endpoint with specified name doesn't exist
- `ESP_ERR_INVALID_ARG` : Null instance/name arguments

#### Parameters

- [in] pc: Pointer to the protocomm instance
- [in] ep\_name: Endpoint identifier(name) string

**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.

#### Note

- An endpoint must be bound to a valid protocomm instance, created using `protocomm_new()`.

#### Return

- 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

#### Parameters

- [in] pc: Pointer to the protocomm instance
- [in] ep\_name: Endpoint identifier(name) string
- [in] version: Version identifier(name) string

**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.

#### Return

- ESP\_OK : Success
- ESP\_ERR\_NOT\_FOUND : Endpoint with specified name doesn't exist
- ESP\_ERR\_INVALID\_ARG : Null instance/name arguments

#### Parameters

- [in] pc: Pointer to the protocomm instance
- [in] ep\_name: Endpoint identifier(name) string

### 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.

**Note** Structure of the protocomm object is kept private

### Header File

- [protocomm/include/security/protocomm\\_security.h](#)

### Structures

```
struct protocomm_security_pop
```

Proof Of Possession for authenticating a secure session.

### 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\_security**

Protocomm security object structure.

The member functions are used for implementing secure protocomm sessions.

**Note** 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 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 protocomm_security_pop_t *pop, 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_security_pop protocomm_security_pop_t`

Proof Of Possession for authenticating a secure session.

`typedef void *protocomm_security_handle_t`

`typedef struct protocomm_security protocomm_security_t`

Protocomm security object structure.

The member functions are used for implementing secure protocomm sessions.

**Note** This structure should not have any dynamic members to allow re-entrancy

### Header File

- `protocomm/include/security/protocomm_security0.h`

### Header File

- `protocomm/include/security/protocomm_security1.h`

## Header File

- `protocomm/include/transport/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.

**Note** This is a singleton. ie. Protocomm can have multiple instances, but only one instance can be bound to an HTTP transport layer.

### 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

### Parameters

- `[in] pc`: Protocomm instance pointer obtained from `protocomm_new()`
- `[in] config`: Pointer to config structure for initializing HTTP server

`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.

### Return

- `ESP_OK` : Success
- `ESP_ERR_INVALID_ARG` : Null / incorrect protocomm instance pointer

### Parameters

- `[in] pc`: Same protocomm instance that was passed to `protocomm_httpd_start()`

## Unions

`union protocomm_httpd_config_data_t`

`#include <protocomm_httpd.h>` Protocomm HTTPD Configuration Data

## Public Members

`void *handle`

HTTP Server Handle, if `ext_handle_provided` is set to true

`protocomm_http_server_config_t config`

HTTP Server Configuration, if a server is not already active

## Structures

`struct protocomm_http_server_config_t`

Config parameters for protocomm HTTP server.

## Public Members

`uint16_t port`

Port on which the HTTP server will listen

`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 of an external HTTP Server Handle has been provided. In such as 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

**PROTOCOLM\_HTTPD\_DEFAULT\_CONFIG()**

### Header File

- [protocomm/include/transport/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.

#### Return

- 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

#### Parameters

- [in] pc: Protocomm instance pointer obtained from protocomm\_new()
- [in] config: Pointer to config structure for initializing BLE

*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

**Note** You might want to optionally reclaim memory from Bluetooth. Refer to the documentation of `esp_bt_mem_release` in that case.

#### Return

- ESP\_OK : Success
- ESP\_FAIL : Simple BLE stop error
- ESP\_ERR\_INVALID\_ARG : Null / incorrect protocomm instance

#### Parameters

- [in] pc: Same protocomm instance that was passed to protocomm\_ble\_start()

### 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**

**const char \*name**  
Name of the handler, which is passed to protocomm layer

**uint16\_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]**  
BLE device name being broadcast at the time of provisioning

**uint8\_t service\_uuid[BLE\_UUID128\_VAL\_LENGTH]**  
128 bit UUID of the provisioning service

**ssize\_t nu\_lookup\_count**  
Number of entries in the Name-UUID lookup table

*protocomm\_ble\_name\_uuid\_t* \*nu\_lookup  
Pointer to the Name-UUID lookup table

**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

**BLE\_UUID128\_VAL\_LENGTH**

**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.

**2.5.3 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()*.

```
wifi_prov_mgr_config_t config = {
    .scheme = wifi_prov_scheme_ble,
    .scheme_event_handler = WIFI_PROV_SCHEME_BLE_EVENT_HANDLER_FREE_BTDM
};

ESP_ERR_CHECK( wifi_prov_mgr_init(config) );
```

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 reboot 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:

```
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"
                        "\n\tSSID      : %s\n\tPassword : %s",
                        (const char *) wifi_sta_cfg->ssid,
                        (const char *) 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!\n\tReason : %s"
                        "\n\tPlease reset to factory and retry_
↳provisioning",
```

(下页继续)

(续上页)

```

        (*reason == WIFI_PROV_STA_AUTH_ERROR) ?
        "Wi-Fi station authentication failed" : "Wi-Fi_
↪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

```

bool provisioned = false;
ESP_ERR_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.

```

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_ERR_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 successful 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()`.

**注解:** 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()`.

```
// Start provisioning service
ESP_ERR_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:`

```
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 :

表 2: Endpoints provided by Provisioning Service

Endpoint Name (BLE + GATT Server)	URI (SoftAP + HTTP Server + mDNS)	Description
prov-session	<a href="http://&lt;mdns-hostname&gt;.local/prov-session">http://&lt;mdns-hostname&gt;.local/prov-session</a>	Security endpoint used for session establishment
prov-scan	<a href="http://wifi-prov.local/prov-scan">http://wifi-prov.local/prov-scan</a>	Endpoint used for starting Wi-Fi scan and receiving scan results
prov-config	<a href="http://&lt;mdns-hostname&gt;.local/prov-config">http://&lt;mdns-hostname&gt;.local/prov-config</a>	Endpoint used for configuring Wi-Fi credentials on device
proto-ver	<a href="http://&lt;mdns-hostname&gt;.local/proto-ver">http://&lt;mdns-hostname&gt;.local/proto-ver</a>	Endpoint for retrieving version info

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

scanned. 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

**Additional Endpoints** In case users want to have some additional protocomm endpoints customized to their requirements, 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.

```
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 device 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/manager](#)

## API Reference

### Header File

- [wifi\\_provisioning/include/wifi\\_provisioning/manager.h](#)

### Functions

*esp\_err\_t* **wifi\_prov\_mgr\_init** (*wifi\_prov\_mgr\_config\_t* config)

Initialize provisioning manager instance.

Configures the manager and allocates internal resources

Configuration specifies the provisioning scheme (transport) and event handlers

Event WIFI\_PROV\_INIT is emitted right after initialization is complete

#### Return

- ESP\_OK : Success
- ESP\_FAIL : Fail

#### Parameters

- [in] config: Configuration structure

void **wifi\_prov\_mgr\_deinit** (void)

Stop provisioning (if running) and release resource used by the manager.

Event WIFI\_PROV\_DEINIT is emitted right after de-initialization is finished

If provisioning service is still active when this API is called, it first stops the service, hence emitting WIFI\_PROV\_END, and then performs the de-initialization

*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)

**Note** Calling wifi\_prov\_mgr\_start\_provisioning() automatically resets the provision state, irrespective of what the state was prior to making the call.

#### Return

- ESP\_OK : Retrieved provision state successfully
- ESP\_FAIL : Wi-Fi not initialized
- ESP\_ERR\_INVALID\_ARG : Null argument supplied
- ESP\_ERR\_INVALID\_STATE : Manager not initialized

#### Parameters

- [out] provisioned: True if provisioned, else false

*esp\_err\_t* **wifi\_prov\_mgr\_start\_provisioning** (*wifi\_prov\_security\_t* security, const char \*pop, 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

**Note** This API will start provisioning service even if device is found to be already provisioned, i.e. `wifi_prov_mgr_is_provisioned()` yields true

**Return**

- ESP\_OK : Provisioning started successfully
- ESP\_FAIL : Failed to start provisioning service
- ESP\_ERR\_INVALID\_STATE : Provisioning manager not initialized or already started

**Parameters**

- [in] `security`: 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
- [in] `pop`: Pointer to proof of possession string (NULL if not needed). This is relevant only for protocomm security 1, in which case it is used for authenticating secure session
- [in] `service_name`: Unique name of the service. This translates to:
  - Wi-Fi SSID when provisioning mode is softAP
  - Device name when provisioning mode is BLE
- [in] `service_key`: 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

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.

`Cleaup_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.

**Note** This must be called before executing `wifi_prov_mgr_start_provisioning()`

**Return**

- ESP\_OK : Success
- ESP\_ERR\_INVALID\_STATE : Manager not initialized or provisioning service already started

**Parameters**

- [in] `cleanup_delay`: 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.

*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.

**Note** This must be called before executing `wifi_prov_mgr_start_provisioning()`

**Return**

- 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

**Parameters**

- [in] `label`: String indicating the application name.
- [in] `version`: String indicating the application version. There is no constraint on format.
- [in] `capabilities`: Array of strings with capabilities. These could be used by the client side app to know the application registered endpoint capabilities
- [in] `total_capabilities`: Size of capabilities array

*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.

**Note** This API can only be called BEFORE provisioning is started

**Note** 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

**Note** 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

**Return**

- ESP\_OK : Success
- ESP\_FAIL : Failure

**Parameters**

- [in] `ep_name`: unique name of the endpoint

*esp\_err\_t* `wifi_prov_mgr_endpoint_register` (`const char *ep_name`, `proto-comm_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()`.

**Note** This API can only be called AFTER provisioning has started

**Note** 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

**Note** 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

#### Return

- ESP\_OK : Success
- ESP\_FAIL : Failure

#### Parameters

- [in] ep\_name: Name of the endpoint
- [in] handler: Endpoint handler function
- [in] user\_ctx: User data

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.

#### Parameters

- [in] ep\_name: Name of the endpoint

*esp\_err\_t* **wifi\_prov\_mgr\_event\_handler** (void \*ctx, *system\_event\_t* \*event)

Event handler for provisioning manager.

This is called from the main event handler and controls the provisioning manager's internal state machine depending on incoming Wi-Fi events

**Note** : This function is DEPRECATED, because events are now handled internally using the event loop library, esp\_event. Calling this will do nothing and simply return ESP\_OK.

#### Return

- ESP\_OK : Event handled successfully

#### Parameters

- [in] ctx: Event context data
- [in] event: Event info

*esp\_err\_t* **wifi\_prov\_mgr\_get\_wifi\_state** (*wifi\_prov\_sta\_state\_t* \*state)

Get state of Wi-Fi Station during provisioning.

#### Return

- ESP\_OK : Successfully retrieved Wi-Fi state
- ESP\_FAIL : Provisioning app not running

#### Parameters

- [out] state: Pointer to wifi\_prov\_sta\_state\_t variable to be filled

*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.

#### Return

- ESP\_OK : Successfully retrieved Wi-Fi disconnect reason
- ESP\_FAIL : Provisioning app not running

#### Parameters

- [out] reason: Pointer to wifi\_prov\_sta\_fail\_reason\_t variable to be filled

*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

#### Return

- ESP\_OK : Wi-Fi configured and started successfully
- ESP\_FAIL : Failed to set configuration

**Parameters**

- [in] `wifi_cfg`: Pointer to Wi-Fi configuration structure

**Structures****struct wifi\_prov\_event\_handler\_t**

Event handler that is used by the manager while provisioning service is active.

**Public Members***wifi\_prov\_cb\_func\_t* **event\_cb**

Callback function to be executed on provisioning events

void **\*user\_data**

User context data to pass as parameter to callback function

**struct wifi\_prov\_scheme**

Structure for specifying the provisioning scheme to be followed by the manager.

**Note** 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.

**Note** 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)

```
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

## Enumerations

```
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

*Values:*

```
WIFI_PROV_INIT
```

Emitted when the manager is initialized

```
WIFI_PROV_START
```

Indicates that provisioning has started

```
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

**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`

**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`

**WIFI\_PROV\_END**

Signals that provisioning service has stopped

**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:

**WIFI\_PROV\_SECURITY\_0 = 0**

No security (plain-text communication)

**WIFI\_PROV\_SECURITY\_1**

This secure communication mode consists of X25519 key exchange

- proof of possession (pop) based authentication
- AES-CTR encryption

**Header File**

- [wifi\\_provisioning/include/wifi\\_provisioning/scheme\\_ble.h](#)

**Functions**

```
void wifi_prov_scheme_ble_event_cb_free_bt(dm (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 0000ffff-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.

**Note** The data being pointed to by the argument must be valid atleast till provisioning is started. Upon start, the manager will store an internal copy of this UUID, and this data can be freed or invalidated afterwards.

**Return**

- `ESP_OK` : Success
- `ESP_ERR_INVALID_ARG` : Null argument

**Parameters**

- `[in] uuid128`: A custom 128 bit UUID

**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**

- `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.

**Note** This API should be called before `wifi_prov_mgr_start_provisioning()`

**Parameters**

- [in] *handle*: Handle to HTTPD server instance

**Header File**

- `wifi_provisioning/include/wifi_provisioning/scheme_console.h`

**Header File**

- `wifi_provisioning/include/wifi_provisioning/wifi_config.h`

**Functions**

*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**

**struct wifi\_prov\_sta\_conn\_info\_t**

WiFi STA connected status information.

**Public Members**

char **ip\_addr**[IP4ADDR\_STRLEN\_MAX]

IP Address received by station

char **bssid**[6]

BSSID of the AP to which connection was established

char **ssid**[33]

SSID of the to which connection was established

uint8\_t **channel**

Channel of the AP

uint8\_t **auth\_mode**

Authorization mode of the AP

**struct wifi\_prov\_config\_get\_data\_t**

WiFi status data to be sent in response to `get_status` request from master.

**Public Members***wifi\_prov\_sta\_state\_t* **wifi\_state**

WiFi state of the station

*wifi\_prov\_sta\_fail\_reason\_t* **fail\_reason**Reason for disconnection (valid only when *wifi\_state* is `WIFI_STATION_DISCONNECTED`)*wifi\_prov\_sta\_conn\_info\_t* **conn\_info**Connection information (valid only when *wifi\_state* is `WIFI_STATION_CONNECTED`)**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 (eg. 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 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:

`WIFI_PROV_STA_CONNECTING`

`WIFI_PROV_STA_CONNECTED`

`WIFI_PROV_STA_DISCONNECTED`

#### `enum wifi_prov_sta_fail_reason_t`

WiFi STA connection fail reason.

Values:

`WIFI_PROV_STA_AUTH_ERROR`

`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。

## 2.6 Storage API

### 2.6.1 SPI Flash API

#### Overview

The `spi_flash` component contains API functions related to reading, writing, erasing, memory mapping for data in the external flash. The `spi_flash` component also has higher-level API functions which work with partitions defined in the [partition table](#).

Different from the API before IDF v4.0, the functionality is not limited to the “main” SPI flash chip (the same SPI flash chip from which program runs). With different chip pointers, you can access to external flashes chips on not only SPI0/1 but also HSPI/VSPI buses.

---

**注解:** Flash APIs after IDF v4.0 are no longer *atomic*. A writing operation during another on-going read operation, on the overlapped flash address, may cause the return data from the read operation to be partly same as before, and partly updated as new written.

---

Kconfig option `CONFIG_SPI_FLASH_USE_LEGACY_IMPL` can be used to switch `spi_flash_*` functions back to the implementation before IDF v4.0. However, the code size may get bigger if you use the new API and the old API the same time.

Encrypted reads and writes use the old implementation, even if `CONFIG_SPI_FLASH_USE_LEGACY_IMPL` is not enabled. As such, encrypted flash operations are only supported with the main flash chip (and not with other flash chips on SPI1 with different CS).

#### Initializing a flash device

To use `esp_flash_*` APIs, you need to have a chip initialized on a certain SPI bus.

1. Call `spi_bus_initialize()` to properly initialize an SPI bus. This functions initialize the resources (I/O, DMA, interrupts) shared among devices attached to this bus.
2. Call `spi_bus_add_flash_device()` to attach the flash device onto the bus. This allocates memory, and fill the members for the `esp_flash_t` structure. The CS I/O is also initialized here.

3. Call `esp_flash_init()` to actually communicate with the chip. This will also detect the chip type, and influence the following operations.

---

**注解:** Multiple flash chips can be attached to the same bus now. However, using `esp_flash_*` devices and `spi_device_*` devices on the same SPI bus is not supported yet.

---

### SPI flash access API

This is the set of API functions for working with data in flash:

- `esp_flash_read()` reads data from flash to RAM
- `esp_flash_write()` writes data from RAM to flash
- `esp_flash_erase_region()` erases specific region of flash
- `esp_flash_erase_chip()` erases the whole flash
- `esp_flash_get_chip_size()` returns flash chip size, in bytes, as configured in menuconfig

Generally, try to avoid using the raw SPI flash functions to the “main” SPI flash chip in favour of *partition-specific functions*.

### SPI Flash Size

The SPI flash size is configured by writing a field in the software bootloader image header, flashed at offset 0x1000.

By default, the SPI flash size is detected by `esptool.py` when this bootloader is written to flash, and the header is updated with the correct size. Alternatively, it is possible to generate a fixed flash size by setting `CONFIG_ESPTOOLPY_FLASHSIZE` in project configuration.

If it is necessary to override the configured flash size at runtime, it is possible to set the `chip_size` member of the `g_rom_flashchip` structure. This size is used by `esp_flash_*` functions (in both software & ROM) to check the bounds.

### Concurrency Constraints for flash on SPI1

Because the SPI1 flash is also used for firmware execution via the instruction & data caches, these caches must be disabled while reading/writing/erasing. This means that both CPUs must be running code from IRAM and must only be reading data from DRAM while flash write operations occur.

If you use the API functions documented here, then these constraints are applied automatically and transparently. However, note that it will have some performance impact on other tasks in the system.

There are no such constraints and impacts for flash chips on other SPI buses than SPI0/1.

For differences between IRAM, DRAM, and flash cache, please refer to the *application memory layout* documentation.

To avoid reading flash cache accidentally, when one CPU initiates a flash write or erase operation, the other CPU is put into a blocked state, and all non-IRAM-safe interrupts are disabled on both CPUs until the flash operation completes.

If one CPU initiates a flash write or erase operation, the other CPU is put into a blocked state to avoid reading flash cache accidentally. All interrupts not safe for IRAM are disabled on both CPUs until the flash operation completes.

**IRAM-Safe Interrupt Handlers** If you have an interrupt handler that you want to execute while a flash operation is in progress (for example, for low latency operations), set the `ESP_INTR_FLAG_IRAM` flag when the *interrupt handler is registered*.

You must ensure that all data and functions accessed by these interrupt handlers, including the ones that handlers call, are located in IRAM or DRAM.

Use the `IRAM_ATTR` attribute for functions:

```
#include "esp_attr.h"

void IRAM_ATTR gpio_isr_handler(void* arg)
{
    // ...
}
```

Use the `DRAM_ATTR` and `DRAM_STR` attributes for constant data:

```
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");
}
```

Note that knowing which data should be marked with `DRAM_ATTR` can be hard, the compiler will sometimes recognize that a variable or expression is constant (even if it is not marked `const`) and optimize it into flash, unless it is marked with `DRAM_ATTR`.

If a function or symbol is not correctly put into IRAM/DRAM, and the interrupt handler reads from the flash cache during a flash operation, it will cause a crash due to Illegal Instruction exception (for code which should be in IRAM) or garbage data to be read (for constant data which should be in DRAM).

## Partition table API

ESP-IDF projects use a partition table to maintain information about various regions of SPI flash memory (bootloader, various application binaries, data, filesystems). More information on partition tables can be found [here](#).

This component provides API functions to enumerate partitions found in the partition table and perform operations on them. These functions are declared in `esp_partition.h`:

- `esp_partition_find()` checks a partition table for entries with specific type, returns an opaque iterator.
- `esp_partition_get()` returns a structure describing the partition for a given iterator.
- `esp_partition_next()` shifts the iterator to the next found partition.
- `esp_partition_iterator_release()` releases iterator returned by `esp_partition_find`.
- `esp_partition_find_first()` - a convenience function which returns the structure describing the first partition found by `esp_partition_find`.
- `esp_partition_read()`, `esp_partition_write()`, `esp_partition_erase_range()` are equivalent to `spi_flash_read()`, `spi_flash_write()`, `spi_flash_erase_range()`, but operate within partition boundaries.

---

**注解:** Application code should mostly use these `esp_partition_*` API functions instead of lower level `spi_flash_*` API functions. Partition table API functions do bounds checking and calculate correct offsets in flash, based on data stored in a partition table.

---

## SPI Flash Encryption

It is possible to encrypt the contents of SPI flash and have it transparently decrypted by hardware.

Refer to the [Flash Encryption documentation](#) for more details.

## Memory mapping API

ESP32 features memory hardware which allows regions of flash memory to be mapped into instruction and data address spaces. This mapping works only for read operations. It is not possible to modify contents of flash memory by writing to a mapped memory region.

Mapping happens in 64KB pages. Memory mapping hardware can map up to four megabytes of flash into data address space and up to 16 megabytes of flash into instruction address space. See the technical reference manual for more details about memory mapping hardware.

Note that some 64KB pages are used to map the application itself into memory, so the actual number of available 64KB pages may be less.

Reading data from flash using a memory mapped region is the only way to decrypt contents of flash when *flash encryption* is enabled. Decryption is performed at the hardware level.

Memory mapping API are declared in `esp_spi_flash.h` and `esp_partition.h`:

- `spi_flash_mmap()` maps a region of physical flash addresses into instruction space or data space of the CPU.
- `spi_flash_munmap()` unmaps previously mapped region.
- `esp_partition_mmap()` maps part of a partition into the instruction space or data space of the CPU.

Differences between `spi_flash_mmap()` and `esp_partition_mmap()` are as follows:

- `spi_flash_mmap()` must be given a 64KB aligned physical address.
- `esp_partition_mmap()` may be given any arbitrary offset within the partition, it will adjust the returned pointer to mapped memory as necessary

Note that since memory mapping happens in 64KB blocks, it may be possible to read data outside of the partition provided to `esp_partition_mmap`.

## Implementation

The `esp_flash_t` structure holds chip data as well as three important parts of this API:

1. The host driver, which provides the hardware support to access the chip;
2. The chip driver, which provides compatibility service to different chips;
3. The OS functions, provides support of some OS functions (e.g. lock, delay) in different stages (1st/2st boot, or the app).

**Host driver** The host driver relies on an interface (`spi_flash_host_driver_t`) defined in the `spi_flash_host_drv.h` (in the `soc/include/hal` folder). This interface provides some common functions to communicate with the chip.

In other files of the SPI HAL, some of these functions are implemented with existing ESP32 memory-spi functionalities. However due to the speed limitations of ESP32, the HAL layer can't provide high-speed implementations to some reading commands (So we didn't do it at all). The files (`memspi_host_driver.h` and `.c`) implement the high-speed version of these commands with the `common_command` function provided in the HAL, and wrap these functions as `spi_flash_host_driver_t` for upper layer to use.

You can also implement your own host driver, even with the GPIO. As long as all the functions in the `spi_flash_host_driver_t` are implemented, the `esp_flash` API can access to the flash regardless of the low-level hardware.

**Chip driver** The chip driver, defined in `spi_flash_chip_driver.h`, wraps basic functions provided by the host driver for the API layer to use.

Some operations need some commands to be sent first, or read some status after. Some chips need different command or value, or need special communication ways.

There is a type of chip called `generic_chip` which stands for common chips. Other special chip drivers can be developed on the base of the generic chip.

The chip driver relies on the host driver.



**OS functions** Currently the OS function layer provides a lock and a delay entries.

The lock is used to resolve the conflicts between the SPI chip access and other functions. E.g. the cache (used for the code and PSRAM data fetch) should be disabled when the flash chip on the SPI0/1 is being accessed. Also, some devices which don't have CS wire, or the wire is controlled by the software (e.g. SD card via SPI interface), requires the bus to be monopolized during a period.

The delay is used by some long operations which requires the master to wait or polling periodically.

The top API wraps these the chip driver and OS functions into an entire component, and also provides some argument checking.

### See also

- [Partition Table documentation](#)
- [Over The Air Update \(OTA\) API](#) provides high-level API for updating app firmware stored in flash.
- [Non-Volatile Storage \(NVS\) API](#) provides a structured API for storing small pieces of data in SPI flash.

### Implementation details

In order to perform some flash operations, it is necessary to make sure that both CPUs are not running any code from flash for the duration of the flash operation: - In a single-core setup, the SDK does it by disabling interrupts/scheduler before performing the flash operation. - In a dual-core setup, this is slightly more complicated as the SDK needs to make sure that the other CPU is not running any code from flash.

When SPI flash API is called on CPU A (can be PRO or APP), start the `spi_flash_op_block_func` function on CPU B using the `esp_ipc_call` API. This API wakes up a high priority task on CPU B and tells it to execute a given function, in this case, `spi_flash_op_block_func`. This function disables cache on CPU B and signals that the cache is disabled by setting the `s_flash_op_can_start` flag. Then the task on CPU A disables cache as well and proceeds to execute flash operation.

While a flash operation is running, interrupts can still run on CPUs A and B. It is assumed that all interrupt code is placed into RAM. Once the interrupt allocation API is added, a flag should be added to request the interrupt to be disabled for the duration of a flash operations.

Once the flash operation is complete, the function on CPU A sets another flag, `s_flash_op_complete`, to let the task on CPU B know that it can re-enable cache and release the CPU. Then the function on CPU A re-enables the cache on CPU A as well and returns control to the calling code.

Additionally, all API functions are protected with a mutex (`s_flash_op_mutex`).

In a single core environment (`CONFIG_FREERTOS_UNICORE` enabled), you need to disable both caches, so that no inter-CPU communication can take place.

### API Reference - SPI Flash

#### Header File

- `spi_flash/include/esp_flash_spi_init.h`

#### Functions

```
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`.

#### Return

- `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.

**Parameters**

- `out_chip`: Pointer to hold the initialized chip.
- `config`: Configuration of the chips to initialize.

*esp\_err\_t* **spi\_bus\_remove\_flash\_device** (*esp\_flash\_t* \*chip)

Remove a SPI Flash device from the SPI bus.

**Return**

- `ESP_ERR_INVALID_ARG`: The chip is invalid.
- `ESP_OK`: success.

**Parameters**

- `chip`: The flash device to remove.

**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\_id**

CS pin (signal) 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.

*esp\_flash\_speed\_t* **speed**

Speed of the Flash clock.

int **input\_delay\_ns**

Input delay of the data pins, in ns. Set to 0 if unknown.

**Header File**

- `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.

**Note** 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.

**Note** 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.

**Return** `ESP_OK` on success, or a flash error code if initialisation fails.

**Parameters**

- `chip`: Pointer to SPI flash chip to use. If NULL, `esp_flash_default_chip` is substituted.

bool **esp\_flash\_chip\_driver\_initialized** (const *esp\_flash\_t* \*chip)

Check if appropriate chip driver is set.

**Return** true if set, otherwise false.

**Parameters**

- `chip`: Pointer to SPI flash chip to use. If NULL, `esp_flash_default_chip` is substituted.

*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.

**Parameters**

- *chip*: Pointer to identify flash chip. Must have been successfully initialised via *esp\_flash\_init()*
- [out] *out\_id*: Pointer to receive ID value.

**Return** 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.

**Note** 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.

**Return** ESP\_OK on success, or a flash error code if operation failed.

**Parameters**

- *chip*: Pointer to identify flash chip. Must have been successfully initialised via *esp\_flash\_init()*
- [out] *out\_size*: Detected size in bytes.

*esp\_err\_t esp\_flash\_erase\_chip(esp\_flash\_t \*chip)*

Erase flash chip contents.

**Return** ESP\_OK on success, or a flash error code if operation failed.

**Parameters**

- *chip*: Pointer to identify flash chip. Must have been successfully initialised via *esp\_flash\_init()*

*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.

**Parameters**

- *chip*: Pointer to identify flash chip. 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.

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.

**Return** ESP\_OK on success, or a flash error code if operation failed.

*esp\_err\_t esp\_flash\_get\_chip\_write\_protect(esp\_flash\_t \*chip, bool \*write\_protected)*

Read if the entire chip is write protected.

**Note** A correct result for this flag depends on the SPI flash chip model and *chip\_drv* in use (via the ‘*chip->drv*’ field).

**Return** ESP\_OK on success, or a flash error code if operation failed.

**Parameters**

- *chip*: Pointer to identify flash chip. Must have been successfully initialised via *esp\_flash\_init()*
- [out] *write\_protected*: Pointer to boolean, set to the value of the write protect flag.

*esp\_err\_t esp\_flash\_set\_chip\_write\_protect(esp\_flash\_t \*chip, bool write\_protect)*

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. 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.

```
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).

**Return** ESP\_OK on success, or a flash error code if operation failed.

**Parameters**

- chip: Pointer to identify flash chip. Must have been successfully initialised via esp\_flash\_init()
- [out] out\_regions: Pointer to receive a pointer to the array of protectable regions of the chip.
- [out] out\_num\_regions: Pointer to an integer receiving the count of protectable regions in the array returned in 'regions' .

```
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.

**Note** Correct behaviour of this function depends on the SPI flash chip model and chip\_drv in use (via the 'chip->drv' field).

**Return** ESP\_OK on success, or a flash error code if operation failed.

**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] out\_protected: Pointer to a flag which is set based on the protected status for this region.

```
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.

**Note** 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.

**Note** Correct behaviour of this function depends on the SPI flash chip model and chip\_drv in use (via the 'chip->drv' field).

**Return** ESP\_OK on success, or a flash error code if operation failed.

**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(...).
- protect: Write protection flag to set.

```
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.

**Parameters**

- chip: Pointer to identify flash chip. Must have been successfully initialised 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.

**Note** If on-chip flash encryption is used, this function returns raw (ie encrypted) data. Use the flash cache to transparently decrypt data.

**Return**

- 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.

#### Parameters

- `chip`: Pointer to identify flash chip. Must have been successfully initialised 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.

**Return** `ESP_OK` on success, or a flash error code if operation failed.

`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.

**Note** Both address & length must be 16 byte aligned, as this is the encryption block size

#### Return

- `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.
- or other flash error code from `spi_flash_write_encrypted()`.

#### Parameters

- `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.

`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.

#### Return

- `ESP_OK`: on success
- `ESP_ERR_NOT_SUPPORTED`: encrypted read not supported for this chip.
- or other flash error code from `spi_flash_read_encrypted()`.

#### Parameters

- `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.

`static 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.

**Return** true if flash works in quad mode, otherwise false

#### Parameters

- `chip`: Pointer to SPI flash chip to use. If `NULL`, `esp_flash_default_chip` is substituted.

## Structures

`struct esp_flash_region_t`

Structure for describing a region of flash.

### Public Members

**uint32\_t offset**  
Start address of this region.

**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

### Public Members

*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’ ).

*esp\_err\_t* (\***end**) (void \*arg)  
Called after completing any flash operation.

*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

*esp\_err\_t* (\***delay\_us**) (void \*arg, unsigned us)  
Delay for at least ‘us’ microseconds. Called in between ‘start’ and ‘end’ .

*esp\_err\_t* (\***yield**) (void \*arg)  
Yield to other tasks. Called during erase operations.

**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()).

### Public Members

*spi\_flash\_host\_driver\_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.

**uint32\_t chip\_id**  
Detected chip id.

### Type Definitions

```
typedef struct spi_flash_chip_t spi_flash_chip_t
typedef struct esp_flash_t esp_flash_t
```

## Header File

- [soc/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 command**

Command to send, always 8bits.

#### **uint8\_t mosi\_len**

Output data length, in bits.

#### **uint8\_t miso\_len**

Input data length, in bits.

#### **uint32\_t mosi\_data**

Output data to slave.

#### **uint32\_t miso\_data[2]**

[out] Input data from slave, little endian

### **struct spi\_flash\_host\_driver\_t**

Host driver configuration and context structure.

### Public Members

#### **void \*driver\_data**

Configuration and static data used by the specific host driver. The type is determined by the host driver.

#### **esp\_err\_t (\*dev\_config) (spi\_flash\_host\_driver\_t \*driver)**

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\_driver\_t \*driver, spi\_flash\_trans\_t \*t)**

Send an user-defined spi transaction to the device.

#### **esp\_err\_t (\*read\_id) (spi\_flash\_host\_driver\_t \*driver, uint32\_t \*id)**

Read flash ID.

#### **void (\*erase\_chip) (spi\_flash\_host\_driver\_t \*driver)**

Erase whole flash chip.

#### **void (\*erase\_sector) (spi\_flash\_host\_driver\_t \*driver, uint32\_t start\_address)**

Erase a specific sector by its start address.

#### **void (\*erase\_block) (spi\_flash\_host\_driver\_t \*driver, uint32\_t start\_address)**

Erase a specific block by its start address.

#### **esp\_err\_t (\*read\_status) (spi\_flash\_host\_driver\_t \*driver, uint8\_t \*out\_sr)**

Read the status of the flash chip.

#### **esp\_err\_t (\*set\_write\_protect) (spi\_flash\_host\_driver\_t \*driver, bool wp)**

Disable write protection.

#### **void (\*program\_page) (spi\_flash\_host\_driver\_t \*driver, 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\_driver\_t \*driver, const void \*p)**

Check whether need to allocate new buffer to write

bool (**\*supports\_direct\_read**) (*spi\_flash\_host\_driver\_t* \*driver, const void \*p)  
Check whether need to allocate new buffer to read

int **max\_write\_bytes**  
maximum length of program\_page

*esp\_err\_t* (**\*read**) (*spi\_flash\_host\_driver\_t* \*driver, void \*buffer, uint32\_t address, uint32\_t read\_len)  
Read data from the flash. Check `max_read_bytes` for the maximum allowed reading length.

int **max\_read\_bytes**  
maximum length of read

bool (**\*host\_idle**) (*spi\_flash\_host\_driver\_t* \*driver)  
Check whether the host is idle to perform new operations.

*esp\_err\_t* (**\*configure\_host\_io\_mode**) (*spi\_flash\_host\_driver\_t* \*driver, 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.

void (**\*poll\_cmd\_done**) (*spi\_flash\_host\_driver\_t* \*driver)  
Internal use, poll the HW until the last operation is done.

*esp\_err\_t* (**\*flush\_cache**) (*spi\_flash\_host\_driver\_t* \*driver, 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.

## Macros

**ESP\_FLASH\_SPEED\_MIN**  
Lowest speed supported by the driver, currently 5 MHz.

**SPI\_FLASH\_READ\_MODE\_MIN**  
Slowest io mode supported by ESP32, currently SlowRd.

## Type Definitions

**typedef struct *spi\_flash\_host\_driver\_t* spi\_flash\_host\_driver\_t**

## Enumerations

**enum *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.

*Values:*

**ESP\_FLASH\_5MHZ = 0**  
The flash runs under 5MHz.

**ESP\_FLASH\_10MHZ**  
The flash runs under 10MHz.

**ESP\_FLASH\_20MHZ**  
The flash runs under 20MHz.

**ESP\_FLASH\_26MHZ**  
The flash runs under 26MHz.

**ESP\_FLASH\_40MHZ**  
The flash runs under 40MHz.

**ESP\_FLASH\_80MHZ**  
The flash runs under 80MHz.



**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:

**SPI\_FLASH\_SLOWRD = 0**

Data read using single I/O, some limits on speed.

**SPI\_FLASH\_FASTRD**

Data read using single I/O, no limit on speed.

**SPI\_FLASH\_DOUT**

Data read using dual I/O.

**SPI\_FLASH\_DIO**

Both address & data transferred using dual I/O.

**SPI\_FLASH\_QOUT**

Data read using quad I/O.

**SPI\_FLASH\_QIO**

Both address & data transferred using quad I/O.

**SPI\_FLASH\_READ\_MODE\_MAX**

The fastest io mode supported by the host is ESP\_FLASH\_READ\_MODE\_MAX-1.

**API Reference - Partition Table****Header File**

- [spi\\_flash/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.

**Return** 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.

**Parameters**

- type: Partition type, one of `esp_partition_type_t` values
- subtype: Partition subtype, one of `esp_partition_subtype_t` values. 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.

**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.

**Return** pointer to *esp\_partition\_t* structure, or NULL if no partition is found. This pointer is valid for the lifetime of the application.

**Parameters**

- type: Partition type, one of `esp_partition_type_t` values
- subtype: Partition subtype, one of `esp_partition_subtype_t` values. 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.

**const** *esp\_partition\_t* \***esp\_partition\_get** (*esp\_partition\_iterator\_t* iterator)

Get *esp\_partition\_t* structure for given partition.

**Return** pointer to *esp\_partition\_t* structure. This pointer is valid for the lifetime of the application.

**Parameters**

- *iterator*: Iterator obtained using `esp_partition_find`. Must be non-NULL.

*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.

**Return** NULL if no partition was found, valid *esp\_partition\_iterator\_t* otherwise.

**Parameters**

- *iterator*: Iterator obtained using `esp_partition_find`. Must be non-NULL.

void **esp\_partition\_iterator\_release** (*esp\_partition\_iterator\_t* iterator)

Release partition iterator.

**Parameters**

- *iterator*: Iterator obtained using `esp_partition_find`. Must be non-NULL.

**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.

**Return**

- 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.

**Parameters**

- *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.

*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.

**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.

**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.

*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 `spi_flash_write_encrypted()` function. If writing to an encrypted partition, all write offsets and lengths must be multiples of 16 bytes. See the `spi_flash_write_encrypted()` function for more details. Unencrypted partitions do not have this restriction.

**Note** Prior to writing to flash memory, make sure it has been erased with `esp_partition_erase_range` call.

**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.

#### 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.

*esp\_err\_t* `esp_partition_erase_range` (`const esp_partition_t` \*partition, `size_t` offset, `size_t` size)

Erase part of the partition.

**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.

#### 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 4 kilobytes.
- `size`: Size of the range which should be erased, in bytes. Must be divisible by 4 kilobytes.

*esp\_err\_t* `esp_partition_mmap` (`const esp_partition_t` \*partition, `size_t` offset, `size_t` size, `spi_flash_mmap_memory_t` memory, `const void` \*\*out\_ptr, `spi_flash_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 `spi_flash_munmap` function.

**Return** `ESP_OK`, if successful

#### 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 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 `spi_flash_munmap` call

*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.

#### Return

- `ESP_OK`: In case of successful operation.
- `ESP_ERR_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.

#### Parameters

- [in] `partition`: Pointer to info for partition containing app or data. (fields: address, size and type, are required to be filled).
- [out] `sha_256`: Returned SHA-256 digest for a given partition.

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.

#### Return

- True: In case of the two firmware is equal.
- False: Otherwise

#### Parameters

- [in] `partition_1`: Pointer to info for partition 1 containing app or data. (fields: address, size and type, are required to be filled).
- [in] `partition_2`: Pointer to info for partition 2 containing app or data. (fields: address, size and type, are required to be filled).

`esp_err_t esp_partition_register_external` (`esp_flash_t *flash_chip`, `size_t offset`, `size_t size`, `const char *label`, `esp_partition_type_t type`, `esp_partition_subtype_t subtype`, `const esp_partition_t **out_partition`)

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.

#### Return

- ESP\_OK on success
- ESP\_ERR\_NOT\_SUPPORTED if CONFIG\_CONFIG\_SPI\_FLASH\_USE\_LEGACY\_IMPL is enabled
- 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

#### 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\_\*). Note that applications can not be booted from external flash chips, so using ESP\_PARTITION\_TYPE\_APP is not supported.
- `subtype`: One of the partition subtypes (ESP\_PARTITION\_SUBTYPE\_\*)
- [out] `out_partition`: Output, if non-NULL, receives the pointer to the resulting `esp_partition_t` structure

`esp_err_t esp_partition_deregister_external` (`const esp_partition_t *partition`)

Deregister the partition previously registered using `esp_partition_register_external`.

#### Return

- ESP\_OK on success
- ESP\_ERR\_NOT\_FOUND if the partition pointer is not found
- ESP\_ERR\_INVALID\_ARG if the partition comes from the partition table
- ESP\_ERR\_INVALID\_ARG if the partition was not registered using `esp_partition_register_external` function.

#### Parameters

- `partition`: pointer to the partition structure obtained from `esp_partition_register_external`,

## 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

*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

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 struct** *esp\_partition\_iterator\_opaque\_\** **esp\_partition\_iterator\_t**

Opaque partition iterator type.

### Enumerations

**enum** **esp\_partition\_type\_t**

Partition type.

**Note** Keep this enum in sync with PartitionDefinition class *gen\_esp32part.py*

*Values:*

**ESP\_PARTITION\_TYPE\_APP** = 0x00

Application partition type.

**ESP\_PARTITION\_TYPE\_DATA** = 0x01

Data partition type.

**enum** **esp\_partition\_subtype\_t**

Partition subtype.

**Note** Keep this enum in sync with PartitionDefinition class *gen\_esp32part.py*

*Values:*

**ESP\_PARTITION\_SUBTYPE\_APP\_FACTORY** = 0x00

Factory application partition.

**ESP\_PARTITION\_SUBTYPE\_APP\_OTA\_MIN** = 0x10

Base for OTA partition subtypes.

**ESP\_PARTITION\_SUBTYPE\_APP\_OTA\_0** = *ESP\_PARTITION\_SUBTYPE\_APP\_OTA\_MIN* + 0

OTA partition 0.

**ESP\_PARTITION\_SUBTYPE\_APP\_OTA\_1** = *ESP\_PARTITION\_SUBTYPE\_APP\_OTA\_MIN* + 1  
OTA partition 1.

**ESP\_PARTITION\_SUBTYPE\_APP\_OTA\_2** = *ESP\_PARTITION\_SUBTYPE\_APP\_OTA\_MIN* + 2  
OTA partition 2.

**ESP\_PARTITION\_SUBTYPE\_APP\_OTA\_3** = *ESP\_PARTITION\_SUBTYPE\_APP\_OTA\_MIN* + 3  
OTA partition 3.

**ESP\_PARTITION\_SUBTYPE\_APP\_OTA\_4** = *ESP\_PARTITION\_SUBTYPE\_APP\_OTA\_MIN* + 4  
OTA partition 4.

**ESP\_PARTITION\_SUBTYPE\_APP\_OTA\_5** = *ESP\_PARTITION\_SUBTYPE\_APP\_OTA\_MIN* + 5  
OTA partition 5.

**ESP\_PARTITION\_SUBTYPE\_APP\_OTA\_6** = *ESP\_PARTITION\_SUBTYPE\_APP\_OTA\_MIN* + 6  
OTA partition 6.

**ESP\_PARTITION\_SUBTYPE\_APP\_OTA\_7** = *ESP\_PARTITION\_SUBTYPE\_APP\_OTA\_MIN* + 7  
OTA partition 7.

**ESP\_PARTITION\_SUBTYPE\_APP\_OTA\_8** = *ESP\_PARTITION\_SUBTYPE\_APP\_OTA\_MIN* + 8  
OTA partition 8.

**ESP\_PARTITION\_SUBTYPE\_APP\_OTA\_9** = *ESP\_PARTITION\_SUBTYPE\_APP\_OTA\_MIN* + 9  
OTA partition 9.

**ESP\_PARTITION\_SUBTYPE\_APP\_OTA\_10** = *ESP\_PARTITION\_SUBTYPE\_APP\_OTA\_MIN* + 10  
OTA partition 10.

**ESP\_PARTITION\_SUBTYPE\_APP\_OTA\_11** = *ESP\_PARTITION\_SUBTYPE\_APP\_OTA\_MIN* + 11  
OTA partition 11.

**ESP\_PARTITION\_SUBTYPE\_APP\_OTA\_12** = *ESP\_PARTITION\_SUBTYPE\_APP\_OTA\_MIN* + 12  
OTA partition 12.

**ESP\_PARTITION\_SUBTYPE\_APP\_OTA\_13** = *ESP\_PARTITION\_SUBTYPE\_APP\_OTA\_MIN* + 13  
OTA partition 13.

**ESP\_PARTITION\_SUBTYPE\_APP\_OTA\_14** = *ESP\_PARTITION\_SUBTYPE\_APP\_OTA\_MIN* + 14  
OTA partition 14.

**ESP\_PARTITION\_SUBTYPE\_APP\_OTA\_15** = *ESP\_PARTITION\_SUBTYPE\_APP\_OTA\_MIN* + 15  
OTA partition 15.

**ESP\_PARTITION\_SUBTYPE\_APP\_OTA\_MAX** = *ESP\_PARTITION\_SUBTYPE\_APP\_OTA\_MIN* + 16  
Max subtype of OTA partition.

**ESP\_PARTITION\_SUBTYPE\_APP\_TEST** = 0x20  
Test application partition.

**ESP\_PARTITION\_SUBTYPE\_DATA\_OTA** = 0x00  
OTA selection partition.

**ESP\_PARTITION\_SUBTYPE\_DATA\_PHY** = 0x01  
PHY init data partition.

**ESP\_PARTITION\_SUBTYPE\_DATA\_NVS** = 0x02  
NVS partition.

**ESP\_PARTITION\_SUBTYPE\_DATA\_COREDUMP** = 0x03  
COREDUMP partition.

**ESP\_PARTITION\_SUBTYPE\_DATA\_NVS\_KEYS** = 0x04  
Partition for NVS keys.

**ESP\_PARTITION\_SUBTYPE\_DATA\_EFUSE\_EM** = 0x05  
Partition for emulate eFuse bits.

**ESP\_PARTITION\_SUBTYPE\_DATA\_ESPHTTTPD** = 0x80  
ESPHTTTPD partition.

**ESP\_PARTITION\_SUBTYPE\_DATA\_FAT** = 0x81  
FAT partition.

**ESP\_PARTITION\_SUBTYPE\_DATA\_SPIFFS** = 0x82  
SPIFFS partition.

**ESP\_PARTITION\_SUBTYPE\_ANY** = 0xff  
Used to search for partitions with any subtype.

## API Reference - Flash Encrypt

### Header File

- [bootloader\\_support/include/esp\\_flash\\_encrypt.h](#)

### Functions

**static 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.

**Return** true if flash encryption is enabled.

*esp\_err\_t* **esp\_flash\_encrypt\_check\_and\_update** (void)

*esp\_err\_t* **esp\_flash\_encrypt\_region** (uint32\_t *src\_addr*, size\_t *data\_length*)

Encrypt-in-place a block of flash sectors.

**Note** This function resets RTC\_WDT between operations with sectors.

**Return** ESP\_OK if all operations succeeded, ESP\_ERR\_FLASH\_OP\_FAIL if SPI flash fails, ESP\_ERR\_FLASH\_OP\_TIMEOUT if flash times out.

#### Parameters

- *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.

void **esp\_flash\_write\_protect\_crypt\_cnt** ()

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.

*esp\_flash\_enc\_mode\_t* **esp\_get\_flash\_encryption\_mode** ()

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

#### Return

void **esp\_flash\_encryption\_init\_checks** (void)

Check the flash encryption mode during startup.

Verifies the flash encryption config during startup:

**Note** This function is called automatically during app startup, it doesn't need to be called from the app.

- 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

## Enumerations

`enum esp_flash_enc_mode_t`

Values:

`ESP_FLASH_ENC_MODE_DISABLED`

`ESP_FLASH_ENC_MODE_DEVELOPMENT`

`ESP_FLASH_ENC_MODE_RELEASE`

## 2.6.2 SD/SDIO/MMC Driver

### Overview

The SD/SDIO/MMC driver currently supports SD memory, SDIO cards, and eMMC chips. This is a protocol level driver built on top of SDMMC and SD SPI host drivers.

SDMMC and SD SPI host drivers ([driver/include/driver/sdmmc\\_host.h](#)) provide API functions for:

- Sending commands to slave devices
- Sending and receiving data
- Handling error conditions within the bus

For functions used to initialize and configure:

- SDMMC host, see [SDMMC Host API](#)
- SD SPI host, see [SD SPI Host API](#)

The SDMMC protocol layer described in this document handles the specifics of the SD protocol, such as the card initialization and data transfer commands.

The protocol layer works with the host via the `sdmmc_host_t` structure. This structure contains pointers to various functions of the host.

### Application Example

An example which combines the SDMMC driver with the FATFS library is provided in the [storage/sd\\_card](#) directory of ESP-IDF examples. This example initializes the card, then writes and reads data from it using POSIX and C library APIs. See README.md file in the example directory for more information.

### Protocol layer API

The protocol layer is given the `sdmmc_host_t` structure. This structure describes the SD/MMC host driver, lists its capabilities, and provides pointers to functions of the driver. The protocol layer stores card-specific information in the `sdmmc_card_t` structure. When sending commands to the SD/MMC host driver, the protocol layer uses the `sdmmc_command_t` structure to describe the command, arguments, expected return values, and data to transfer if there is any.

### Using API with SD memory cards

1. To initialize the host, call the host driver functions, e.g., `sdmmc_host_init()`, `sdmmc_host_init_slot()`.
2. To initialize the card, call `sdmmc_card_init()` and pass to it the parameters `host` - the host driver information, and `card` - a pointer to the structure `sdmmc_card_t` which will be filled with information about the card when the function completes.
3. To read and write sectors of the card, use `sdmmc_read_sectors()` and `sdmmc_write_sectors()` respectively and pass to it the parameter `card` - a pointer to the card information structure.
4. If the card is not used anymore, call the host driver function - e.g., `sdmmc_host_deinit()` - to disable the host peripheral and free the resources allocated by the driver.



**Using API with eMMC chips** From the protocol layer's perspective, eMMC memory chips behave exactly like SD memory cards. Even though eMMCs are chips and do not have a card form factor, the terminology for SD cards can still be applied to eMMC due to the similarity of the protocol (*sdmmc\_card\_t*, *sdmmc\_card\_init*). Note that eMMC chips cannot be used over SPI, which makes them incompatible with the SD SPI host driver.

To initialize eMMC memory and perform read/write operations, follow the steps listed for SD cards in the previous section.

**Using API with SDIO cards** Initialization and the probing process is the same as with SD memory cards. The only difference is in data transfer commands in SDIO mode.

During the card initialization and probing, performed with *sdmmc\_card\_init()*, the driver only configures the following registers of the IO card:

1. The IO portion of the card is reset by setting RES bit in the I/O Abort (0x06) register.
2. If 4-line mode is enabled in host and slot configuration, the driver attempts to set the Bus width field in the Bus Interface Control (0x07) register. If setting the field is successful, which means that the slave supports 4-line mode, the host is also switched to 4-line mode.
3. If high-speed mode is enabled in the host configuration, the SHS bit is set in the High Speed (0x13) register.

In particular, the driver does not set any bits in (1) I/O Enable and Int Enable registers, (2) I/O block sizes, etc. Applications can set them by calling *sdmmc\_io\_write\_byte()*.

For card configuration and data transfer, choose the pair of functions relevant to your case from the table below.

Action	Read Function	Write Function
Read and write a single byte using IO_RW_DIRECT (CMD52)	<i>sd-mmc_io_read_byte()</i>	<i>sd-mmc_io_write_byte()</i>
Read and write multiple bytes using IO_RW_EXTENDED (CMD53) in byte mode	<i>sd-mmc_io_read_bytes()</i>	<i>sd-mmc_io_write_bytes()</i>
Read and write blocks of data using IO_RW_EXTENDED (CMD53) in block mode	<i>sd-mmc_io_read_blocks()</i>	<i>sd-mmc_io_write_blocks()</i>

SDIO interrupts can be enabled by the application using the function *sdmmc\_io\_enable\_int()*. When using SDIO in 1-line mode, the D1 line also needs to be connected to use SDIO interrupts.

If you want the application to wait until the SDIO interrupt occurs, use *sdmmc\_io\_wait\_int()*.

**Combo (memory + IO) cards** The driver does not support SD combo cards. Combo cards are treated as IO cards.

**Thread safety** Most applications need to use the protocol layer only in one task. For this reason, the protocol layer does not implement any kind of locking on the *sdmmc\_card\_t* structure, or when accessing SDMMC or SD SPI host drivers. Such locking is usually implemented on a higher layer, e.g., in the filesystem driver.

## API Reference

### Header File

- [sdmmc/include/sdmmc\\_cmd.h](#)

### Functions

*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

**Note** Only SD cards (SDSC and SDHC/SDXC) are supported now. Support for MMC/eMMC cards will be added later.

#### Return

- ESP\_OK on success

- One of the error codes from SDMMC host controller

**Parameters**

- `host`: pointer to structure defining host controller
- `out_card`: pointer to structure which will receive information about the card when the function completes

void **sdmmc\_card\_print\_info** (FILE *\*stream*, const *sdmmc\_card\_t* *\*card*)

Print information about the card to a stream.

**Parameters**

- `stream`: stream obtained using `fopen` or `fdopen`
- `card`: card information structure initialized using `sdmmc_card_init`

*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

**Return**

- ESP\_OK on success
- One of the error codes from SDMMC host controller

**Parameters**

- `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

*esp\_err\_t* **sdmmc\_read\_sectors** (*sdmmc\_card\_t* *\*card*, void *\*dst*, size\_t *start\_sector*, size\_t *sector\_count*)

Write given number of sectors to SD/MMC card

**Return**

- ESP\_OK on success
- One of the error codes from SDMMC host controller

**Parameters**

- `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

*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)

**Return**

- ESP\_OK on success
- One of the error codes from SDMMC host controller

**Parameters**

- `card`: pointer to card information structure previously initialized using `sdmmc_card_init`
- `function`: IO function number
- `reg`: byte address within IO function
- [out] `out_byte`: output, receives the value read from the card

*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)

**Return**

- ESP\_OK on success
- One of the error codes from SDMMC host controller

**Parameters**

- `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] out_byte`: if not NULL, receives new byte value read from the card (read-after-write).

*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`.

#### Return

- ESP\_OK on success
- ESP\_ERR\_INVALID\_SIZE if size exceeds 512 bytes
- One of the error codes from SDMMC host controller

#### Parameters

- `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

*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`.

#### Return

- ESP\_OK on success
- ESP\_ERR\_INVALID\_SIZE if size exceeds 512 bytes
- One of the error codes from SDMMC host controller

#### Parameters

- `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

*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`.

#### Return

- 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

#### Parameters

- `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.

*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`.

**Return**

- 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

**Parameters**

- *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.

*esp\_err\_t* `sdmmc_io_enable_int` (*sdmmc\_card\_t* \**card*)

Enable SDIO interrupt in the SDMMC host

**Return**

- ESP\_OK on success
- ESP\_ERR\_NOT\_SUPPORTED if the host controller does not support IO interrupts

**Parameters**

- *card*: pointer to card information structure previously initialized using `sdmmc_card_init`

*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.

**Return**

- 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*

**Parameters**

- *card*: pointer to card information structure previously initialized using `sdmmc_card_init`
- *timeout\_ticks*: time to wait for the interrupt, in RTOS 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 a SDIO card.

You may provide a buffer not sufficient to store all the CIS data. In this case, this functions store as much data into your buffer as possible. Also, this function will try to get and return the size required for you.

**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`

**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.

*esp\_err\_t* `sdmmc_io_print_cis_info` (uint8\_t \**buffer*, size\_t *buffer\_size*, FILE \**fp*)

Parse and print the CIS information of a SDIO card.

**Note** 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.

```
using sdmmc_card_init
```

**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.

**Parameters**

- `buffer`: Buffer to parse
- `buffer_size`: Size of the buffer.
- `fp`: File pointer to print to, set to `NULL` to print to stdout.

**Header File**

- [driver/include/driver/sdmmc\\_types.h](#)

**Structures****struct sdmmc\_csd\_t**

Decoded values from SD card Card Specific Data register

**Public Members**

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

### Public Members

int **sd\_spec**  
SD Physical layer specification version, reported by card

int **bus\_width**  
bus widths supported by card: BIT(0) — 1-bit bus, BIT(2) — 4-bit bus

**struct sdmmc\_ext\_csd\_t**  
Decoded values of Extended Card Specific Data

### Public Members

uint8\_t **power\_class**  
Power class used 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\_t **datalen**  
length of data buffer

size\_t **blklen**  
block length

int **flags**  
see below

*esp\_err\_t* **error**  
error returned from transfer

int **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)

**esp\_err\_t (\*init) (void)**

Host function to initialize the driver

**esp\_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

**esp\_err\_t (\*set\_bus\_ddr\_mode) (int slot, bool ddr\_enable)**

host function to set DDR mode

**esp\_err\_t (\*set\_card\_clk) (int slot, uint32\_t freq\_khz)**

host function to set card clock frequency

**esp\_err\_t (\*do\_transaction) (int slot, sdmmc\_command\_t \*cmdinfo)**

host function to do a transaction

**esp\_err\_t (\*deinit) (void)**

host function to deinitialize the driver

**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.

**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\_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

**uint32\_t is\_mem : 1**

Bit indicates if the card is a memory card

**uint32\_t is\_sdio : 1**

Bit indicates if the card is an IO card

**uint32\_t is\_mmc : 1**

Bit indicates if the card is MMC

**uint32\_t num\_io\_functions : 3**

If is\_sdio is 1, contains the number of IO functions on the card

**uint32\_t log\_bus\_width : 2**log<sub>2</sub>(bus width supported by card)**uint32\_t is\_ddr : 1**

Card supports DDR mode

**uint32\_t reserved : 23**

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\_FREQ\_DEFAULT**

SD/MMC Default speed (limited by clock divider)

**SDMMC\_FREQ\_HIGHSPEED**

SD High speed (limited by clock divider)

**SDMMC\_FREQ\_PROBING**

SD/MMC probing speed

**SDMMC\_FREQ\_52M**

MMC 52MHz speed

**SDMMC\_FREQ\_26M**

MMC 26MHz speed



### Type Definitions

```
typedef uint32_t sdmmc_response_t[4]
    SD/MMC command response buffer
```

## 2.6.3 Non-volatile storage library

### Introduction

Non-volatile storage (NVS) library is designed to store key-value pairs in flash. This section introduces some concepts used by NVS.

**Underlying storage** Currently, NVS uses a portion of main flash memory through `spi_flash_{read|write|erase}` APIs. The library uses all the partitions with data type and `nvs` subtype. The application can choose to use the partition with the label `nvs` through the `nvs_open` API function or any other partition by specifying its name using the `nvs_open_from_part` API function.

Future versions of this library may have other storage backends to keep data in another flash chip (SPI or I2C), RTC, FRAM, etc.

---

**注解:** if an NVS partition is truncated (for example, when the partition table layout is changed), its contents should be erased. ESP-IDF build system provides a `idf.py erase_flash` target to erase all contents of the flash chip.

---

---

**注解:** NVS works best for storing many small values, rather than a few large values of the type 'string' and 'blob'. If you need to store large blobs or strings, consider using the facilities provided by the FAT filesystem on top of the wear levelling library.

---

**Keys and values** NVS operates on key-value pairs. Keys are ASCII strings; the maximum key length is currently 15 characters. Values can have one of the following types:

- integer types: `uint8_t`, `int8_t`, `uint16_t`, `int16_t`, `uint32_t`, `int32_t`, `uint64_t`, `int64_t`
- zero-terminated string
- variable length binary data (blob)

---

**注解:** String values are currently limited to 4000 bytes. This includes the null terminator. Blob values are limited to 508000 bytes or 97.6% of the partition size - 4000 bytes, whichever is lower.

---

Additional types, such as `float` and `double` might be added later.

Keys are required to be unique. Assigning a new value to an existing key works as follows:

- if the new value is of the same type as the old one, value is updated
- if the new value has a different data type, an error is returned

Data type check is also performed when reading a value. An error is returned if the data type of the read operation does not match the data type of the value.

**Namespaces** To mitigate potential conflicts in key names between different components, NVS assigns each key-value pair to one of namespaces. Namespace names follow the same rules as key names, i.e., the maximum length is 15 characters. Namespace name is specified in the `nvs_open` or `nvs_open_from_part` call. This call returns an opaque handle, which is used in subsequent calls to the `nvs_read_*`, `nvs_write_*`, and `nvs_commit` functions. This way, a handle is associated with a namespace, and key names will not collide with same names in other namespaces. Please note that the namespaces with the same name in different NVS partitions are considered as separate namespaces.

**Security, tampering, and robustness** NVS is not directly compatible with the ESP32 flash encryption system. However, data can still be stored in encrypted form if NVS encryption is used together with ESP32 flash encryption. Please refer to *NVS Encryption* for more details.

If NVS encryption is not used, it is possible for anyone with physical access to the flash chip to alter, erase, or add key-value pairs. With NVS encryption enabled, it is not possible to alter or add a key-value pair and get recognized as a valid pair without knowing corresponding NVS encryption keys. However, there is no tamper-resistance against the erase operation.

The library does try to recover from conditions when flash memory is in an inconsistent state. In particular, one should be able to power off the device at any point and time and then power it back on. This should not result in loss of data, except for the new key-value pair if it was being written at the moment of powering off. The library should also be able to initialize properly with any random data present in flash memory.

## Internals

**Log of key-value pairs** NVS stores key-value pairs sequentially, with new key-value pairs being added at the end. When a value of any given key has to be updated, a new key-value pair is added at the end of the log and the old key-value pair is marked as erased.

**Pages and entries** NVS library uses two main entities in its operation: pages and entries. Page is a logical structure which stores a portion of the overall log. Logical page corresponds to one physical sector of flash memory. Pages which are in use have a *sequence number* associated with them. Sequence numbers impose an ordering on pages. Higher sequence numbers correspond to pages which were created later. Each page can be in one of the following states:

**Empty/uninitialized** Flash storage for the page is empty (all bytes are `0xff`). Page is not used to store any data at this point and does not have a sequence number.

**Active** Flash storage is initialized, page header has been written to flash, page has a valid sequence number. Page has some empty entries and data can be written there. No more than one page can be in this state at any given moment.

**Full** Flash storage is in a consistent state and is filled with key-value pairs. Writing new key-value pairs into this page is not possible. It is still possible to mark some key-value pairs as erased.

**Erasing** Non-erased key-value pairs are being moved into another page so that the current page can be erased. This is a transient state, i.e., page should never stay in this state at the time when any API call returns. In case of a sudden power off, the move-and-erase process will be completed upon the next power-on.

**Corrupted** Page header contains invalid data, and further parsing of page data was canceled. Any items previously written into this page will not be accessible. The corresponding flash sector will not be erased immediately and will be kept along with sectors in *uninitialized* state for later use. This may be useful for debugging.

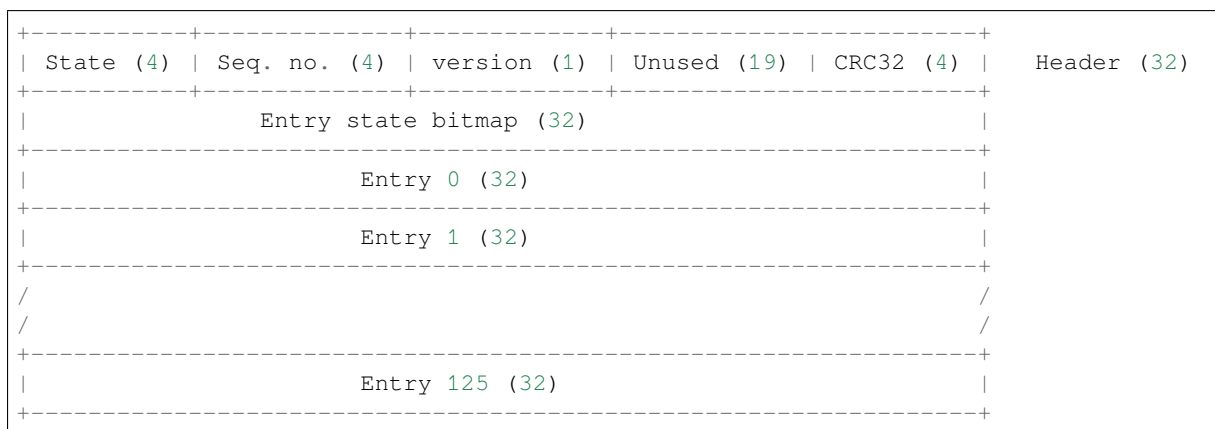
Mapping from flash sectors to logical pages does not have any particular order. The library will inspect sequence numbers of pages found in each flash sector and organize pages in a list based on these numbers.

+-----+	+-----+	+-----+	+-----+	
Page 1	Page 2	Page 3	Page 4	
Full +--->	Full +--->	Active	Empty	<- states
#11 /	#12 /	#14 /	/	<- sequence numbers
+---+---+	+---+---+	+---+---+	+---+---+	
+---v-----+	+---v-----+	+---v-----+	+---v-----+	
Sector 3	Sector 0	Sector 2	Sector 1	<- physical sectors
+-----+	+-----+	+-----+	+-----+	

**Structure of a page** For now, we assume that flash sector size is 4096 bytes and that ESP32 flash encryption hardware operates on 32-byte blocks. It is possible to introduce some settings configurable at compile-time (e.g., via `menuconfig`) to accommodate flash chips with different sector sizes (although it is not clear if other components in the system, e.g., SPI flash driver and SPI flash cache can support these other sizes).

Page consists of three parts: header, entry state bitmap, and entries themselves. To be compatible with ESP32 flash encryption, entry size is 32 bytes. For integer types, entry holds one key-value pair. For strings and blobs, an entry holds part of key-value pair (more on that in the entry structure description).

The following diagram illustrates the page structure. Numbers in parentheses indicate the size of each part in bytes.



Page header and entry state bitmap are always written to flash unencrypted. Entries are encrypted if flash encryption feature of ESP32 is used.

Page state values are defined in such a way that changing state is possible by writing 0 into some of the bits. Therefore it is not necessary to erase the page to change its state unless that is a change to the *erased* state.

The version field in the header reflects the NVS format version used. For backward compatibility reasons, it is decremented for every version upgrade starting at 0xff (i.e., 0xff for version-1, 0xfe for version-2 and so on).

CRC32 value in the header is calculated over the part which does not include a state value (bytes 4 to 28). The unused part is currently filled with 0xff bytes.

The following sections describe the structure of entry state bitmap and entry itself.

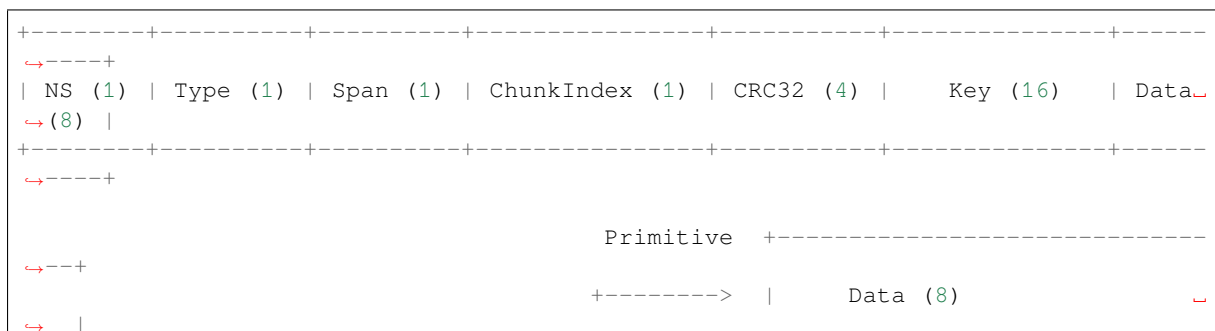
**Entry and entry state bitmap** Each entry can be in one of the following three states represented with two bits in the entry state bitmap. The final four bits in the bitmap (256 - 2 \* 126) are not used.

**Empty (2' b11)** Nothing is written into the specific entry yet. It is in an uninitialized state (all bytes are 0xff).

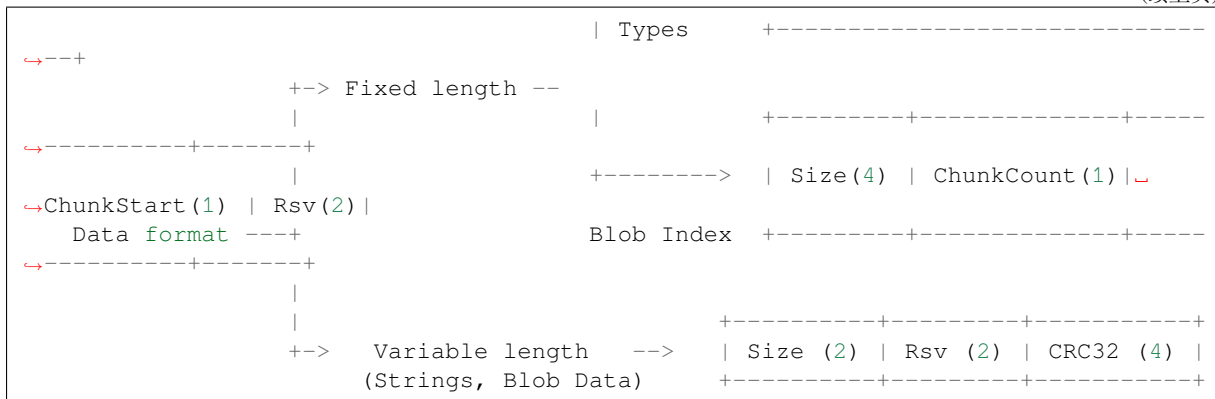
**Written (2' b10)** A key-value pair (or part of key-value pair which spans multiple entries) has been written into the entry.

**Erased (2' b00)** A key-value pair in this entry has been discarded. Contents of this entry will not be parsed anymore.

**Structure of entry** For values of primitive types (currently integers from 1 to 8 bytes long), entry holds one key-value pair. For string and blob types, entry holds part of the whole key-value pair. For strings, in case when a key-value pair spans multiple entries, all entries are stored in the same page. Blobs are allowed to span over multiple pages by dividing them into smaller chunks. For tracking these chunks, an additional fixed length metadata entry is stored called “blob index”. Earlier formats of blobs are still supported (can be read and modified). However, once the blobs are modified, they are stored using the new format.



(下页继续)



Individual fields in entry structure have the following meanings:

**NS** Namespace index for this entry. For more information on this value, see the section on namespaces implementation.

**Type** One byte indicating the value data type. See the `ItemType` enumeration in `nvs_types.h` for possible values.

**Span** Number of entries used by this key-value pair. For integer types, this is equal to 1. For strings and blobs, this depends on value length.

**ChunkIndex** Used to store the index of a blob-data chunk for blob types. For other types, this should be `0xff`.

**CRC32** Checksum calculated over all the bytes in this entry, except for the CRC32 field itself.

**Key** Zero-terminated ASCII string containing a key name. Maximum string length is 15 bytes, excluding a zero terminator.

**Data** For integer types, this field contains the value itself. If the value itself is shorter than 8 bytes, it is padded to the right, with unused bytes filled with `0xff`.

For “blob index” entry, these 8 bytes hold the following information about data-chunks:

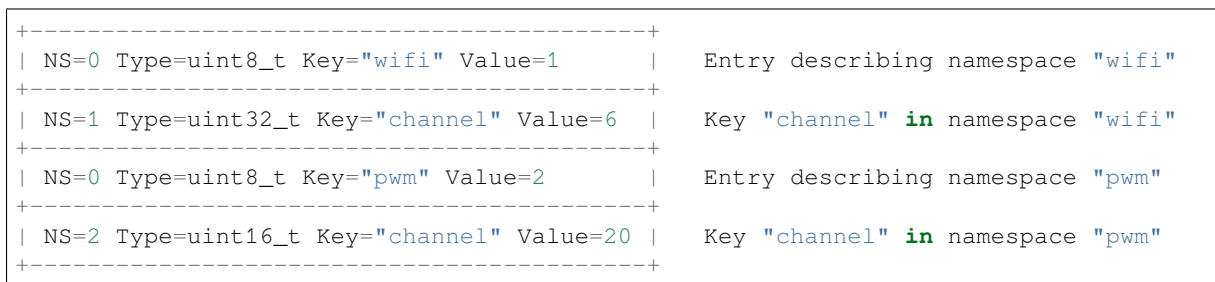
- **Size** (Only for blob index.) Size, in bytes, of complete blob data.
- **ChunkCount** (Only for blob index.) Total number of blob-data chunks into which the blob was divided during storage.
- **ChunkStart** (Only for blob index.) `ChunkIndex` of the first blob-data chunk of this blob. Subsequent chunks have `chunkIndex` incrementally allocated (step of 1).

For string and blob data chunks, these 8 bytes hold additional data about the value, which are described below:

- **Size** (Only for strings and blobs.) Size, in bytes, of actual data. For strings, this includes zero terminators.
- **CRC32** (Only for strings and blobs.) Checksum calculated over all bytes of data.

Variable length values (strings and blobs) are written into subsequent entries, 32 bytes per entry. The *Span* field of the first entry indicates how many entries are used.

**Namespaces** As mentioned above, each key-value pair belongs to one of the namespaces. Namespace identifiers (strings) are stored as keys of key-value pairs in namespace with index 0. Values corresponding to these keys are indexes of these namespaces.



**Item hash list** To reduce the number of reads from flash memory, each member of the `Page` class maintains a list of pairs: item index; item hash. This list makes searches much quicker. Instead of iterating over all entries, reading them from flash one at a time, `Page::findItem` first performs a search for the item hash in the hash list. This

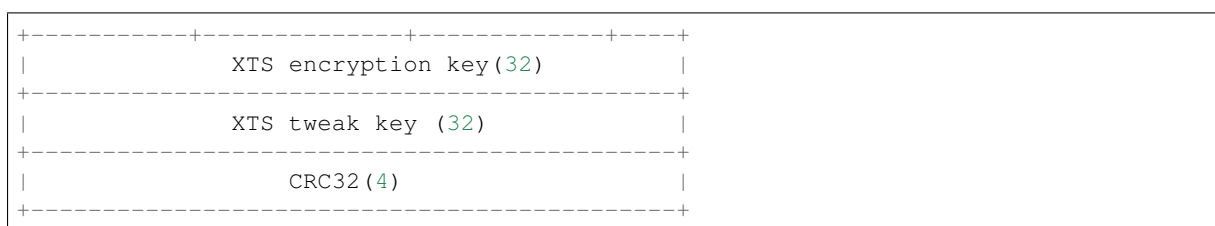
gives the item index within the page if such an item exists. Due to a hash collision, it is possible that a different item will be found. This is handled by falling back to iteration over items in flash.

Each node in the hash list contains a 24-bit hash and 8-bit item index. Hash is calculated based on item namespace, key name, and ChunkIndex. CRC32 is used for calculation; the result is truncated to 24 bits. To reduce the overhead for storing 32-bit entries in a linked list, the list is implemented as a double-linked list of arrays. Each array holds 29 entries, for the total size of 128 bytes, together with linked list pointers and a 32-bit count field. The minimum amount of extra RAM usage per page is therefore 128 bytes; maximum is 640 bytes.

## NVS Encryption

Data stored in NVS partitions can be encrypted using AES-XTS in the manner similar to the one mentioned in disk encryption standard IEEE P1619. For the purpose of encryption, each entry is treated as one *sector* and relative address of the entry (w.r.t. partition-start) is fed to the encryption algorithm as *sector-number*. The keys required for NVS encryption are stored in yet another partition, which is protected using *Flash Encryption*. Therefore, enabling *Flash Encryption* is a prerequisite for NVS encryption.

**NVS key partition** An application requiring NVS encryption support needs to be compiled with a key-partition of the type *data* and subtype *key*. This partition should be marked as *encrypted*. Refer to *Partition Tables* for more details. The size of the partition should be 4096 bytes (minimum partition size). The structure of this partition is depicted below.



This partition can be generated using *nvs partition generator* utility and flashed onto the device. Since the partition is marked *encrypted* and *Flash Encryption* is enabled, bootloader will encrypt this partition using flash encryption key on the first boot. Alternatively, the keys can be generated after startup using the `nvs_flash_generate_keys` API function provided by `nvs_flash.h`, which will then write those keys onto the key-partition in encrypted form.

It is possible for an application to use different keys for different NVS partitions and thereby have multiple key-partitions. However, it is a responsibility of the application to provide correct key-partition/keys for the purpose of encryption/decryption.

**Encrypted Read/Write** The same NVS API functions `nvs_read_*` or `nvs_write_*` can be used for reading of, and writing to an encrypted nvs partition as well. However, the API functions for initialising NVS partitions are different: `nvs_flash_secure_init` and `nvs_flash_secure_init_partition` instead of `nvs_flash_init` and `nvs_flash_init_partition` respectively. The `nvs_sec_cfg_t` structure required for these API functions can be populated using `nvs_flash_generate_keys` or `nvs_flash_read_security_cfg`.

Applications are expected to follow the steps below in order to perform NVS read/write operations with encryption enabled.

1. Find key partition and NVS data partition using `esp_partition_find*` API functions.
2. Populate the `nvs_sec_cfg_t` struct using the `nvs_flash_read_security_cfg` or `nvs_flash_generate_keys` API functions.
3. Initialise NVS flash partition using the `nvs_flash_secure_init` or `nvs_flash_secure_init_partition` API functions.
4. Open a namespace using the `nvs_open` or `nvs_open_from_part` API functions.
5. Perform NVS read/write operations using `nvs_read_*` or `nvs_write_*`.
6. Deinitialise an NVS partition using `nvs_flash_deinit`.

**NVS iterators** Iterators allow to list key-value pairs stored in NVS, based on specified partition name, namespace, and data type.

There are the following functions available:

- `nvs_entry_find` returns an opaque handle, which is used in subsequent calls to the `nvs_entry_next` and `nvs_entry_info` functions.
- `nvs_entry_next` returns iterator to the next key-value pair.
- `nvs_entry_info` returns information about each key-value pair

If none or no other key-value pair was found for given criteria, `nvs_entry_find` and `nvs_entry_next` return NULL. In that case, the iterator does not have to be released. If the iterator is no longer needed, you can release it by using the function `nvs_release_iterator`.

### NVS Partition Generator Utility

This utility helps generate NVS partition binary files which can be flashed separately on a dedicated partition via a flashing utility. Key-value pairs to be flashed onto the partition can be provided via a CSV file. For more details, please refer to [NVS Partition Generator Utility](#).

### Application Example

You can find two code examples in the [storage](#) directory of ESP-IDF examples:

[storage/nvs\\_rw\\_value](#)

Demonstrates how to read a single integer value from, and write it to NVS.

The value checked in this example holds the number of the ESP32 module restarts. The value's function as a counter is only possible due to its storing in NVS.

The example also shows how to check if a read / write operation was successful, or if a certain value has not been initialized in NVS. The diagnostic procedure is provided in plain text to help you track the program flow and capture any issues on the way.

[storage/nvs\\_rw\\_blob](#)

Demonstrates how to read a single integer value and a blob (binary large object), and write them to NVS to preserve this value between ESP32 module restarts.

- `value` - tracks the number of the ESP32 module soft and hard restarts.
- `blob` - contains a table with module run times. The table is read from NVS to dynamically allocated RAM. A new run time is added to the table on each manually triggered soft restart, and then the added run time is written to NVS. Triggering is done by pulling down GPIO0.

The example also shows how to implement the diagnostic procedure to check if the read / write operation was successful.

### API Reference

#### Header File

- [nvs\\_flash/include/nvs\\_flash.h](#)

#### Functions

`esp_err_t nvs_flash_init` (void)

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.

#### Return

- 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
- one of the error codes from the underlying flash storage driver

*esp\_err\_t* **nvs\_flash\_init\_partition** (const char \*partition\_label)

Initialize NVS flash storage for the specified partition.

#### Return

- 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
- one of the error codes from the underlying flash storage driver

#### Parameters

- [in] partition\_label: Label of the partition. Note that internally a reference to passed value is kept and it should be accessible for future operations

*esp\_err\_t* **nvs\_flash\_init\_partition\_ptr** (const *esp\_partition\_t* \*partition)

Initialize NVS flash storage for the partition specified by partition pointer.

#### Return

- 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
- one of the error codes from the underlying flash storage driver

#### Parameters

- [in] partition: pointer to a partition obtained by the ESP partition API.

*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.

#### Return

- 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.

#### Return

- ESP\_OK on success
- ESP\_ERR\_NVS\_NOT\_INITIALIZED if the storage for given partition was not initialized prior to this call

#### Parameters

- [in] partition\_label: Label of the partition

*esp\_err\_t* **nvs\_flash\_erase** (void)

Erase the default NVS partition.

Erases all contents of the default NVS partition (one with label “nvs” ).

**Note** If the partition is initialized, this function first de-initializes it. Afterwards, the partition has to be initialized again to be used.

#### Return

- 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)

*esp\_err\_t* **nvs\_flash\_erase\_partition** (const char \*part\_name)

Erase specified NVS partition.

Erase all content of a specified NVS partition

**Note** If the partition is initialized, this function first de-initializes it. Afterwards, the partition has to be initialized again to be used.

**Return**

- ESP\_OK on success
- ESP\_ERR\_NOT\_FOUND if there is no NVS partition with the specified name in the partition table
- different error in case de-initialization fails (shouldn't happen)

**Parameters**

- [in] `part_name`: Name (label) of the partition which should be erased

`esp_err_t nvs_flash_erase_partition_ptr(const esp_partition_t *partition)`

Erase custom partition.

Erase all content of specified custom partition.

**Note** If the partition is initialized, this function first de-initializes it. Afterwards, the partition has to be initialized again to be used.

**Return**

- 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

**Parameters**

- [in] `partition`: pointer to a partition obtained by the ESP partition API.

`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.

**Return**

- 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
- one of the error codes from the underlying flash storage driver

**Parameters**

- [in] `cfg`: Security configuration (keys) to be used for NVS encryption/decryption. If `cfg` is NULL, no encryption is used.

`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.

**Return**

- 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
- one of the error codes from the underlying flash storage driver

**Parameters**

- [in] `partition_label`: Label of the partition. Note that internally a reference to passed value is kept and it should be accessible for future operations
- [in] `cfg`: Security configuration (keys) to be used for NVS encryption/decryption. If `cfg` is null, no encryption/decryption is used.

`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.

**Return** -ESP\_OK, if `cfg` was read successfully; -or error codes from `esp_partition_write/erase` APIs.

**Parameters**

- [in] `partition`: Pointer to partition structure obtained using `esp_partition_find_first` or `esp_partition_get`. Must be non-NULL.



- [out] `cfg`: Pointer to nvs security configuration structure. Pointer must be non-NULL. Generated keys will be populated in this structure.

`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.

**Note** Provided partition is assumed to be marked 'encrypted'.

**Return** -ESP\_OK, if `cfg` was read successfully; -ESP\_ERR\_NVS\_KEYS\_NOT\_INITIALIZED, if the partition is not yet written with keys. -ESP\_ERR\_NVS\_CORRUPT\_KEY\_PART, if the partition containing keys is found to be corrupt -or error codes from `esp_partition_read` API.

#### Parameters

- [in] `partition`: Pointer to partition structure obtained using `esp_partition_find_first` or `esp_partition_get`. Must be non-NULL.
- [out] `cfg`: Pointer to nvs security configuration structure. Pointer must be non-NULL.

## 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

## Macros

**NVS\_KEY\_SIZE**

## Header File

- [nvs\\_flash/include/nvs.h](#)

## Functions

`esp_err_t nvs_set_i8(nvs_handle_t handle, const char *key, int8_t value)`

set 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.

#### Return

- 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

#### Parameters

- [in] `handle`: Handle obtained from `nvs_open` function. Handles that were opened read only cannot be used.
- [in] `key`: Key name. Maximal length is determined by the underlying implementation, but is guaranteed to be at least 15 characters. Shouldn't be empty.

- [in] value: The value to set. For strings, the maximum length (including null character) is 4000 bytes.

*esp\_err\_t* nvs\_set\_u8(*nvs\_handle\_t* handle, const char \*key, uint8\_t value)

*esp\_err\_t* nvs\_set\_i16(*nvs\_handle\_t* handle, const char \*key, int16\_t value)

*esp\_err\_t* nvs\_set\_u16(*nvs\_handle\_t* handle, const char \*key, uint16\_t value)

*esp\_err\_t* nvs\_set\_i32(*nvs\_handle\_t* handle, const char \*key, int32\_t value)

*esp\_err\_t* nvs\_set\_u32(*nvs\_handle\_t* handle, const char \*key, uint32\_t value)

*esp\_err\_t* nvs\_set\_i64(*nvs\_handle\_t* handle, const char \*key, int64\_t value)

*esp\_err\_t* nvs\_set\_u64(*nvs\_handle\_t* handle, const char \*key, uint64\_t value)

*esp\_err\_t* nvs\_set\_str(*nvs\_handle\_t* handle, const char \*key, const char \*value)

*esp\_err\_t* nvs\_get\_i8(*nvs\_handle\_t* handle, const char \*key, int8\_t \*out\_value)

get 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.

All functions expect out\_value to be a pointer to an already allocated variable of the given type.

```
// 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.
```

### Return

- ESP\_OK if the value was retrieved successfully
- 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

### Parameters

- [in] handle: Handle obtained from nvs\_open function.
- [in] key: Key name. Maximal length is determined by the underlying implementation, but is guaranteed to be at least 15 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.

*esp\_err\_t* nvs\_get\_u8(*nvs\_handle\_t* handle, const char \*key, uint8\_t \*out\_value)

*esp\_err\_t* nvs\_get\_i16(*nvs\_handle\_t* handle, const char \*key, int16\_t \*out\_value)

*esp\_err\_t* nvs\_get\_u16(*nvs\_handle\_t* handle, const char \*key, uint16\_t \*out\_value)

*esp\_err\_t* nvs\_get\_i32(*nvs\_handle\_t* handle, const char \*key, int32\_t \*out\_value)

*esp\_err\_t* nvs\_get\_u32(*nvs\_handle\_t* handle, const char \*key, uint32\_t \*out\_value)

*esp\_err\_t* nvs\_get\_i64(*nvs\_handle\_t* handle, const char \*key, int64\_t \*out\_value)

*esp\_err\_t* nvs\_get\_u64(*nvs\_handle\_t* handle, const char \*key, uint64\_t \*out\_value)

*esp\_err\_t* nvs\_get\_str(*nvs\_handle\_t* handle, const char \*key, char \*out\_value, size\_t \*length)

get 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.

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.

```
// Example (without error checking) of using nvs_get_str to get a string into
↳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:
uint8_t mac_addr[6];
size_t size = sizeof(mac_addr);
nvs_get_blob(my_handle, "dst_mac_addr", mac_addr, &size);
```

### Return

- `ESP_OK` if the value was retrieved successfully
- `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

### Parameters

- `[in]` `handle`: Handle obtained from `nvs_open` function.
- `[in]` `key`: Key name. Maximal length is determined by the underlying implementation, but is guaranteed to be at least 15 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.
- `[inout]` `length`: 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.

`esp_err_t nvs_get_blob(nvs_handle_t handle, const char *key, void *out_value, size_t *length)`

`esp_err_t nvs_open(const char *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.

### Return

- `ESP_OK` if storage handle was opened successfully
- `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` id namespace doesn't exist yet and mode is `NVS_READONLY`
- `ESP_ERR_NVS_INVALID_NAME` if namespace name doesn't satisfy constraints
- other error codes from the underlying storage driver

### Parameters

- `[in]` `name`: Namespace name. Maximal length is determined by the underlying implementation, but is guaranteed to be at least 15 characters. Shouldn't be empty.
- `[in]` `open_mode`: `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]` `out_handle`: If successful (return code is zero), handle will be returned in this argument.

`esp_err_t nvs_open_from_partition(const char *part_name, const char *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.

#### Return

- `ESP_OK` if storage handle was opened successfully
- `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` if namespace doesn't exist yet and mode is `NVS_READONLY`
- `ESP_ERR_NVS_INVALID_NAME` if namespace name doesn't satisfy constraints
- other error codes from the underlying storage driver

#### Parameters

- `[in] part_name`: Label (name) of the partition of interest for object read/write/erase
- `[in] name`: Namespace name. Maximal length is determined by the underlying implementation, but is guaranteed to be at least 15 characters. Shouldn't be empty.
- `[in] open_mode`: `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] out_handle`: If successful (return code is zero), handle will be returned in this argument.

`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.

#### Return

- `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 value is too long

#### Parameters

- `[in] handle`: Handle obtained from `nvs_open` function. Handles that were opened read only cannot be used.
- `[in] key`: Key name. Maximal length is 15 characters. Shouldn't be empty.
- `[in] value`: The value to set.
- `[in] length`: 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.

`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.

#### 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

#### Parameters

- `[in] handle`: Storage handle obtained with `nvs_open`. Handles that were opened read only cannot be used.
- `[in] key`: Key name. Maximal length is determined by the underlying implementation, but is guaranteed to be at least 15 characters. Shouldn't be empty.

*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.

#### 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
- other error codes from the underlying storage driver

#### Parameters

- [in] handle: Storage handle obtained with `nvs_open`. Handles that were opened read only cannot be used.

*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.

#### Return

- `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

#### Parameters

- [in] handle: Storage handle obtained with `nvs_open`. Handles that were opened read only cannot be used.

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.

#### Parameters

- [in] handle: Storage handle to close

*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.

```
// 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);
printf("Count: UsedEntries = (%d), FreeEntries = (%d), AllEntries = (%d)\n",
      nvs_stats.used_entries, nvs_stats.free_entries, nvs_stats.total_
↪entries);
```

#### Return

- `ESP_OK` if the changes have been written successfully. Return param `nvs_stats` will be filled.
- `ESP_ERR_NVS_PART_NOT_FOUND` if the partition with label "name" is not found. Return param `nvs_stats` will be filled 0.
- `ESP_ERR_NVS_NOT_INITIALIZED` if the storage driver is not initialized. Return param `nvs_stats` will be filled 0.
- `ESP_ERR_INVALID_ARG` if `nvs_stats` equal to `NULL`.
- `ESP_ERR_INVALID_STATE` if there is page with the status of `INVALID`. Return param `nvs_stats` will be filled not with correct values because not all pages will be counted. Counting will be interrupted at the first `INVALID` page.

#### Parameters

- [in] `part_name`: Partition name NVS in the partition table. If pass a NULL than will use `NVS_DEFAULT_PART_NAME` ( "nvs" ).
- [out] `nvs_stats`: Returns filled structure `nvs_states_t`. It provides info about used memory the partition.

*esp\_err\_t* `nvs_get_used_entry_count` (*nvs\_handle\_t* `handle`, *size\_t* \*`used_entries`)

Calculate all entries in a namespace.

Note that to find out the total number of records occupied by the namespace, add one to the returned value `used_entries` (if `err` is equal to `ESP_OK`). Because the name space entry takes one entry.

```
// 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 records occupied by the namespace
    total_entries_namespace = used_entries + 1;
}
```

### Return

- `ESP_OK` if the changes have been written successfully. Return param `used_entries` will be filled valid value.
- `ESP_ERR_NVS_NOT_INITIALIZED` if the storage driver is not initialized. Return param `used_entries` will be filled 0.
- `ESP_ERR_NVS_INVALID_HANDLE` if `handle` has been closed or is NULL. Return param `used_entries` will be filled 0.
- `ESP_ERR_INVALID_ARG` if `nvs_stats` equal to NULL.
- Other error codes from the underlying storage driver. Return param `used_entries` will be filled 0.

### Parameters

- [in] `handle`: Handle obtained from `nvs_open` function.
- [out] `used_entries`: Returns amount of used entries from a namespace.

*nvs\_iterator\_t* `nvs_entry_find` (**const** *char* \*`part_name`, **const** *char* \*`namespace_name`, *nvs\_type\_t* `type`)

Create an iterator to enumerate NVS entries based on one or more parameters.

```
// Example of listing all the key-value pairs of any type under specified_
↳partition and namespace
nvs_iterator_t it = nvs_entry_find(partition, namespace, NVS_TYPE_ANY);
while (it != NULL) {
    nvs_entry_info_t info;
    nvs_entry_info(it, &info);
    it = nvs_entry_next(it);
    printf("key '%s', type '%d' \n", info.key, info.type);
};
// Note: no need to release iterator obtained from nvs_entry_find function when
// nvs_entry_find or nvs_entry_next function return NULL, indicating no_
↳other
// element for specified criteria was found.
}
```

**Return** Iterator used to enumerate all the entries found, or NULL if no entry satisfying criteria was found. Iterator obtained through this function has to be released using `nvs_release_iterator` when not used any more.

### Parameters

- [in] `part_name`: Partition name
- [in] `namespace_name`: Set this value if looking for entries with a specific namespace. Pass NULL otherwise.

- [in] type: One of `nvs_type_t` values.

`nvs_iterator_t nvs_entry_next` (`nvs_iterator_t iterator`)

Returns next item matching the iterator criteria, NULL if no such item exists.

Note that any copies of the iterator will be invalid after this call.

**Return** NULL if no entry was found, valid `nvs_iterator_t` otherwise.

**Parameters**

- [in] iterator: Iterator obtained from `nvs_entry_find` function. Must be non-NULL.

void `nvs_entry_info` (`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.

**Parameters**

- [in] iterator: Iterator obtained from `nvs_entry_find` or `nvs_entry_next` function. Must be non-NULL.
- [out] out\_info: Structure to which entry information is copied.

void `nvs_release_iterator` (`nvs_iterator_t iterator`)

Release iterator.

**Parameters**

- [in] iterator: Release iterator obtained from `nvs_entry_find` function. NULL argument is allowed.

### Structures

**struct nvs\_entry\_info\_t**

information about entry obtained from `nvs_entry_info` function

#### Public Members

char **namespace\_name**[16]

Namespace to which key-value belong

char **key**[16]

Key of stored key-value pair

`nvs_type_t` **type**

Type of stored key-value pair

**struct nvs\_stats\_t**

**Note** 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**

Id 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**

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

**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\_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

**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.

*Values:***NVS\_READONLY**

Read only

**NVS\_READWRITE**

Read and write

**enum** **nvs\_type\_t**

Types of variables.

*Values:***NVS\_TYPE\_U8** = 0x01

Type uint8\_t

**NVS\_TYPE\_I8** = 0x11

Type int8\_t

**NVS\_TYPE\_U16** = 0x02

Type uint16\_t

**NVS\_TYPE\_I16** = 0x12

Type int16\_t

**NVS\_TYPE\_U32** = 0x04

Type uint32\_t

**NVS\_TYPE\_I32** = 0x14

Type int32\_t

**NVS\_TYPE\_U64** = 0x08

Type uint64\_t

**NVS\_TYPE\_I64** = 0x18

Type int64\_t

**NVS\_TYPE\_STR** = 0x21

Type string

```

NVS_TYPE_BLOB = 0x42
    Type blob
NVS_TYPE_ANY = 0xff
    Must be last

```

## 2.6.4 NVS Partition Generator Utility

### Introduction

The utility `nvs_flash/nvs_partition_generator/nvs_partition_gen.py` creates a binary file based on key-value pairs provided in a CSV file. The binary file is compatible with NVS architecture defined in *Non-Volatile Storage*. This utility is ideally suited for generating a binary blob, containing data specific to ODM/OEM, which can be flashed externally at the time of device manufacturing. This allows manufacturers to generate many instances of the same application firmware with customized parameters for each device, such as a serial number.

### Prerequisites

To use this utility in encryption mode, install the following packages:

- cryptography package

All the required packages are included in `requirements.txt` in the root of the `esp-idf` directory.

### CSV file format

Each line of a `.csv` file should contain 4 parameters, separated by a comma. The table below provides the description for each of these parameters.

No.	Parameter	Description	Notes
1	Key	Key of the data. The data can be accessed later from an application using this key.	
2	Type	Supported values are <code>file</code> , <code>data</code> and <code>namespace</code> .	
3	Encoding	Supported values are: <code>u8</code> , <code>i8</code> , <code>u16</code> , <code>u32</code> , <code>i32</code> , <code>string</code> , <code>hex2bin</code> , <code>base64</code> and <code>binary</code> . This specifies how actual data values are encoded in the resulting binary file. The difference between the <code>string</code> and <code>binary</code> encoding is that <code>string</code> data is terminated with a NULL character, whereas <code>binary</code> data is not.	As of now, for the <code>file</code> type, only <code>hex2bin</code> , <code>base64</code> , <code>string</code> , and <code>binary</code> encoding is supported.
4	Value	Data value.	Encoding and Value cells for the <code>namespace</code> field type should be empty. Encoding and Value of <code>namespace</code> is fixed and is not configurable. Any values in these cells are ignored.

---

**注解:** The first line of the CSV file should always be the column header and it is not configurable.

---

Below is an example dump of such a CSV file:

```
key, type, encoding, value      <-- column header
namespace_name, namespace,,    <-- First entry should be of type "namespace"
key1, data, u8, 1
key2, file, string, /path/to/file
```

**注解:****Make sure there are no spaces:**

- before and after ‘,’
- at the end of each line in a CSV file

**NVS Entry and Namespace association**

When a namespace entry is encountered in a CSV file, each following entry will be treated as part of that namespace until the next namespace entry is found. At this point, all the following entries will be treated as part of the new namespace.

**注解:** First entry in a CSV file should always be a namespace entry.

**Multipage Blob Support**

By default, binary blobs are allowed to span over multiple pages and are written in the format mentioned in Section [Structure of entry](#). If you intend to use an older format, the utility provides an option to disable this feature.

**Encryption Support**

The NVS Partition Generator utility also allows you to create an encrypted binary file. The utility uses the AES-XTS encryption. Please refer to [NVS Encryption](#) for more details.

**Decryption Support**

This utility allows you to decrypt an encrypted NVS binary file. The utility uses an NVS binary file encrypted using AES-XTS encryption. Please refer to [NVS Encryption](#) for more details.

**Running the utility****Usage:**

```
python nvs_partition_gen.py [-h] {generate,generate-key,encrypt,decrypt} ...
```

## Optional Arguments:

```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
↪-----+
| No. | Parameter | Description |
↪      |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1   | -h, --help | show this help message and exit |
↪      |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
↪-----+
```

(下页继续)

```

Commands:
  Run nvs_partition_gen.py {command} -h for additional help
+-----+
↪-----+
| No. | Parameter | Description |
↪-----+
+-----+
| 1 | generate | Generate NVS partition |
↪-----+
+-----+
↪-----+
| 2 | generate-key | Generate keys for encryption |
↪-----+
+-----+
↪-----+
| 3 | encrypt | Generate NVS encrypted partition |
↪-----+
+-----+
↪-----+
| 4 | decrypt | Decrypt NVS encrypted partition |
↪-----+
+-----+
↪-----+

```

**To generate NVS partition (Default):****Usage:**

```

python nvs_partition_gen.py generate [-h] [--version {1,2}] [--outdir_
↪OUTDIR]
                                input output size
Positional Arguments:
+-----+
↪-----+
| Parameter | Description |
↪-----+
+-----+
| input | Path to CSV file to parse |
↪-----+
+-----+
| output | Path to output NVS binary file |
↪-----+
+-----+
| size | Size of NVS partition in bytes (must be multiple_
↪of 4096) |
+-----+
↪-----+
Optional Arguments:
+-----+
↪-----+
| Parameter | Description |
↪-----+
+-----+
| -h, --help | show this help message and exit |
↪-----+

```

(下页继续)

(续上页)

```

+-----+-----+
| --version {1,2} |      Set multipage blob version.      |
|                 |      Version 1 - Multipage blob support disabled.  |
|                 |      Version 2 - Multipage blob support enabled.  |
|                 |      Default: Version 2                      |
|                 |-----+-----+
| --outdir OUTDIR |      Output directory to store files created      |
|                 |      (Default: current directory)                |
+-----+-----+

```

You can run the utility to generate NVS partition using the command below: A sample CSV file is provided with the utility:

```
python nvs_partition_gen.py generate sample_singlepage_blob.csv sample.bin 0x3000
```

### To generate only encryption keys:

#### Usage:

```
python nvs_partition_gen.py generate-key [-h] [--keyfile KEYFILE]
                                     [--outdir OUTDIR]

Optional Arguments:
+-----+-----+
| Parameter          |      Description      |
+-----+-----+
| -h, --help        |      show this help message and exit      |
+-----+-----+
| --keyfile KEYFILE |      Path to output encryption keys file    |
+-----+-----+
| --outdir OUTDIR   |      Output directory to store files created.  |
|                   |      (Default: current directory)            |
+-----+-----+

```

You can run the utility to generate only encryption keys using the command below:

```
python nvs_partition_gen.py generate-key
```

### To generate encrypted NVS partition:

**Usage:**

```
python nvs_partition_gen.py encrypt [-h] [--version {1,2}] [--keygen]
                                  [--keyfile KEYFILE] [--inputkey 
↳ INPUTKEY]
                                  [--outdir OUTDIR]
                                  input output size
```

Positional Arguments:

Parameter	Description
<code>input</code>	Path to CSV file to parse
<code>output</code>	Path to output NVS binary file
<code>size</code>	Size of NVS partition <b>in bytes</b> (must be multiple <u>↳ of 4096</u> )

Optional Arguments:

Parameter	Description
<code>-h, --help</code>	show this help message <b>and</b> exit
<code>--version {1,2}</code>	Set multipage blob version. Version <b>1</b> - Multipage blob support disabled. Version <b>2</b> - Multipage blob support enabled. Default: Version <b>2</b>
<code>--keygen</code>	Generates key <b>for</b> encrypting NVS partition
<code>--keyfile KEYFILE</code>	Path to output encryption keys file
<code>--inputkey INPUTKEY</code>	File having key <b>for</b> encrypting NVS partition
<code>--outdir OUTDIR</code>	Output directory to store files created

(下页继续)

(续上页)

```

|                                     | (Default: current directory)
↩-----+
+-----+
↩-----+

```

You can run the utility to encrypt NVS partition using the command below: A sample CSV file is provided with the utility:

- Encrypt by allowing the utility to generate encryption keys:

```
python nvs_partition_gen.py encrypt sample_singlepage_blob.csv sample_encr.bin
↩0x3000 --keygen
```

**注解:** Encryption key of the following format <outdir>/keys/keys-<timestamp>.bin is created.

- Encrypt by allowing the utility to generate encryption keys and store it in provided custom filename:

```
python nvs_partition_gen.py encrypt sample_singlepage_blob.csv sample_encr.bin
↩0x3000 --keygen --keyfile sample_keys.bin
```

**注解:** Encryption key of the following format <outdir>/keys/sample\_keys.bin is created.

**注解:** This newly created file having encryption keys in keys/ directory is compatible with NVS key-partition structure. Refer to [NVS key partition](#) for more details.

- Encrypt by providing the encryption keys as input binary file:

```
python nvs_partition_gen.py encrypt sample_singlepage_blob.csv sample_encr.bin
↩0x3000 --inputkey sample_keys.bin
```

### To decrypt encrypted NVS partition:

#### Usage:

```
python nvs_partition_gen.py decrypt [-h] [--outdir OUTDIR] input key_
↩output

Positional Arguments:
+-----+
↩-----+
| Parameter | Description
↩-----+
+-----+
| input     | Path to encrypted NVS partition file to parse
↩-----+
+-----+
| key       | Path to file having keys for decryption
↩-----+
+-----+
| output    | Path to output decrypted binary file
↩-----+
+-----+
↩-----+

```

(下页继续)

(续上页)

```
Optional Arguments:
+-----+-----+
↪-----+
| Parameter          | Description          | ↪
↪
+-----+-----+
| -h, --help        | show this help message and exit | ↪
↪
+-----+-----+
↪-----+
| --outdir OUTDIR   | Output directory to store files created | ↪
↪
|                   | (Default: current directory)           | ↪
↪
+-----+-----+
↪-----+
```

You can run the utility to decrypt encrypted NVS partition using the command below:

```
python nvs_partition_gen.py decrypt sample_encr.bin sample_keys.bin sample_decr.bin
```

**You can also provide the format version number:**

- Multipage Blob Support Disabled (Version 1)
- Multipage Blob Support Enabled (Version 2)

**Multipage Blob Support Disabled (Version 1):** You can run the utility in this format by setting the version parameter to 1, as shown below. A sample CSV file is provided with the utility:

```
python nvs_partition_gen.py generate sample_singlepage_blob.csv sample.bin 0x3000 -
↪-version 1
```

**Multipage Blob Support Enabled (Version 2):** You can run the utility in this format by setting the version parameter to 2, as shown below. A sample CSV file is provided with the utility:

```
python nvs_partition_gen.py generate sample_multipage_blob.csv sample.bin 0x4000 --
↪version 2
```

---

**注解:** *Minimum NVS Partition Size needed is 0x3000 bytes.*

---



---

**注解:** *When flashing the binary onto the device, make sure it is consistent with the application's sdkconfig.*

---

### Caveats

- Utility does not check for duplicate keys and will write data pertaining to both keys. You need to make sure that the keys are distinct.
- Once a new page is created, no data will be written in the space left on the previous page. Fields in the CSV file need to be ordered in such a way as to optimize memory.
- 64-bit datatype is not yet supported.



## 2.6.5 Virtual filesystem component

### Overview

Virtual filesystem (VFS) component provides a unified interface for drivers which can perform operations on file-like objects. These can be real filesystems (FAT, SPIFFS, etc.) or device drivers which provide a file-like interface.

This component allows C library functions, such as `fopen` and `fprintf`, to work with FS drivers. At a high level, each FS driver is associated with some path prefix. When one of C library functions needs to open a file, the VFS component searches for the FS driver associated with the file path and forwards the call to that driver. VFS also forwards read, write, and other calls for the given file to the same FS driver.

For example, one can register a FAT filesystem driver with the `/fat` prefix and call `fopen("/fat/file.txt", "w")`. The VFS component will then call the function `open` of the FAT driver and pass the argument `/file.txt` to it together with appropriate mode flags. All subsequent calls to C library functions for the returned `FILE*` stream will also be forwarded to the FAT driver.

### FS registration

To register an FS driver, an application needs to define an instance of the `esp_vfs_t` structure and populate it with function pointers to FS APIs:

```
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));
```

Depending on the way how the FS driver declares its API functions, either `read`, `write`, etc., or `read_p`, `write_p`, etc., should be used.

Case 1: API functions are declared without an extra context pointer (the FS driver is a singleton):

```
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));
```

Case 2: API functions are declared with an extra context pointer (the FS driver supports multiple instances):

```
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

// When registering FS, pass the FS context pointer into the third argument
// (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));
```

(下页继续)

```
// Can register another instance:
myfs_t* myfs_inst2 = myfs_mount(partition2->offset, partition2->size);
ESP_ERROR_CHECK(esp_vfs_register("/data2", &myfs, myfs_inst2));
```

**Synchronous input/output multiplexing** Synchronous input/output multiplexing by `select()` is supported in the VFS component. The implementation works in the following way.

1. `select()` is called with file descriptors which could belong to various VFS drivers.
2. The file descriptors are divided into groups each belonging to one VFS driver.
3. The file descriptors belonging to non-socket VFS drivers are handed over to the given VFS drivers by `start_select()` described later on this page. This function represents the driver-specific implementation of `select()` for the given driver. This should be a non-blocking call which means the function should immediately return after setting up the environment for checking events related to the given file descriptors.
4. The file descriptors belonging to the socket VFS driver are handed over to the socket driver by `socket_select()` described later on this page. This is a blocking call which means that it will return only if there is an event related to socket file descriptors or a non-socket driver signals `socket_select()` to exit.
5. Results are collected from each VFS driver and all drivers are stopped by deinitialization of the environment for checking events.
6. The `select()` call ends and returns the appropriate results.

**Non-socket VFS drivers** If you want to use `select()` with a file descriptor belonging to a non-socket VFS driver then you need to register the driver with functions `start_select()` and `end_select()` similarly to the following example:

```
// In definition of esp_vfs_t:
    .start_select = &uart_start_select,
    .end_select = &uart_end_select,
// ... other members initialized
```

`start_select()` is called for setting up the environment for detection of read/write/error conditions on file descriptors belonging to the given VFS driver.

`end_select()` is called to stop/deinitialize/free the environment which was setup by `start_select()`.

**注解:** `end_select()` might be called without a previous `start_select()` call in some rare circumstances. `end_select()` should fail gracefully if this is the case.

Please refer to the reference implementation for the UART peripheral in `vfs/vfs_uart.c` and most particularly to the functions `esp_vfs_dev_uart_register()`, `uart_start_select()`, and `uart_end_select()` for more information.

**Please check the following examples that demonstrate the use of `select()` with VFS file descriptors:**

- [peripherals/uart/uart\\_select](#)
- [system/select](#)

**Socket VFS drivers** A socket VFS driver is using its own internal implementation of `select()` and non-socket VFS drivers notify it upon read/write/error conditions.

A socket VFS driver needs to be registered with the following functions defined:

```
// 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
```

`socket_select()` is the internal implementation of `select()` for the socket driver. It works only with file descriptors belonging to the socket VFS.

`get_socket_select_semaphore()` returns the signalization object (semaphore) which will be used in non-socket drivers to stop the waiting in `socket_select()`.

`stop_socket_select()` call is used to stop the waiting in `socket_select()` by passing the object returned by `get_socket_select_semaphore()`.

`stop_socket_select_isr()` has the same functionality as `stop_socket_select()` but it can be used from ISR.

Please see [lwip/port/esp32/vfs\\_lwip.c](#) for a reference socket driver implementation using LWIP.

---

**注解:** If you use `select()` for socket file descriptors only then you can enable the `CONFIG_LWIP_USE_ONLY_LWIP_SELECT` option to reduce the code size and improve performance.

---



---

**注解:** Don't change the socket driver during an active `select()` call or you might experience some undefined behavior.

---

## Paths

Each registered FS has a path prefix associated with it. This prefix can be considered as a “mount point” of this partition.

In case when mount points are nested, the mount point with the longest matching path prefix is used when opening the file. For instance, suppose that the following filesystems are registered in VFS:

- FS 1 on /data
- FS 2 on /data/static

Then:

- FS 1 will be used when opening a file called `/data/log.txt`
- FS 2 will be used when opening a file called `/data/static/index.html`
- Even if `/index.html` does not exist in FS 2, FS 1 will *not* be searched for `/static/index.html`.

As a general rule, mount point names must start with the path separator (/) and must contain at least one character after path separator. However, an empty mount point name is also supported and might be used in cases when an application needs to provide a “fallback” filesystem or to override VFS functionality altogether. Such filesystem will be used if no prefix matches the path given.

VFS does not handle dots (.) in path names in any special way. VFS does not treat `..` as a reference to the parent directory. In the above example, using a path `/data/static/./log.txt` will not result in a call to FS 1 to open `/log.txt`. Specific FS drivers (such as FATFS) might handle dots in file names differently.

When opening files, the FS driver receives only relative paths to files. For example:

1. The `myfs` driver is registered with `/data` as a path prefix.
2. The application calls `fopen("/data/config.json", ...)`.
3. The VFS component calls `myfs_open("/config.json", ...)`.
4. The `myfs` driver opens the `/config.json` file.

VFS does not impose any limit on total file path length, but it does limit the FS path prefix to `ESP_VFS_PATH_MAX` characters. Individual FS drivers may have their own filename length limitations.

## File descriptors

File descriptors are small positive integers from 0 to `FD_SETSIZE - 1`, where `FD_SETSIZE` is defined in newlib's `sys/types.h`. The largest file descriptors (configured by `CONFIG_LWIP_MAX_SOCKETS`) are reserved for sockets. The VFS component contains a lookup-table called `s_fd_table` for mapping global file descriptors to VFS driver indexes registered in the `s_vfs` array.

## Standard IO streams (stdin, stdout, stderr)

If the menuconfig option `UART` for console output is not set to `None`, then `stdin`, `stdout`, and `stderr` are configured to read from, and write to, a UART. It is possible to use `UART0` or `UART1` for standard IO. By default, `UART0` is used with 115200 baud rate; TX pin is `GPIO1`; RX pin is `GPIO3`. These parameters can be changed in menuconfig.

Writing to `stdout` or `stderr` will send characters to the UART transmit FIFO. Reading from `stdin` will retrieve characters from the UART receive FIFO.

By default, VFS uses simple functions for reading from and writing to UART. Writes busy-wait until all data is put into UART FIFO, and reads are non-blocking, returning only the data present in the FIFO. Due to this non-blocking read behavior, higher level C library calls, such as `fscanf("%d\n", &var);`, might not have desired results.

Applications which use the UART driver can instruct VFS to use the driver's interrupt driven, blocking read and write functions instead. This can be done using a call to the `esp_vfs_dev_uart_use_driver` function. It is also possible to revert to the basic non-blocking functions using a call to `esp_vfs_dev_uart_use_nonblocking`.

VFS also provides an optional newline conversion feature for input and output. Internally, most applications send and receive lines terminated by the LF ( `'\n'` ) character. Different terminal programs may require different line termination, such as CR or CRLF. Applications can configure this separately for input and output either via menuconfig, or by calls to the functions `esp_vfs_dev_uart_port_set_rx_line_endings` and `esp_vfs_dev_uart_port_set_tx_line_endings`.

**Standard streams and FreeRTOS tasks** FILE objects for `stdin`, `stdout`, and `stderr` are shared between all FreeRTOS tasks, but the pointers to these objects are stored in per-task `struct _reent`.

The following code is transferred to `fprintf(__getreent()->stderr, "42\n");` by the preprocessor:

```
fprintf(stderr, "42\n");
```

The `__getreent()` function returns a per-task pointer to `struct _reent` ([newlib/include/sys/reent.h#L370-L417](#)). This structure is allocated on the TCB of each task. When a task is initialized, `_stdin`, `_stdout`, and `_stderr` members of `struct _reent` are set to the values of `_stdin`, `_stdout`, and `_stderr` of `_GLOBAL_REENT` (i.e., the structure which is used before FreeRTOS is started).

Such a design has the following consequences:

- It is possible to set `stdin`, `stdout`, and `stderr` for any given task without affecting other tasks, e.g., by doing `stdin = fopen("/dev/uart/1", "r")`.
- Closing default `stdin`, `stdout`, or `stderr` using `fclose` will close the FILE stream object, which will affect all other tasks.
- To change the default `stdin`, `stdout`, `stderr` streams for new tasks, modify `_GLOBAL_REENT->_stdin` (`_stdout`, `_stderr`) before creating the task.

## Application Example

### Instructions

## API Reference

### Header File

- [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.

**Return** ESP\_OK if successful, ESP\_ERR\_NO\_MEM if too many VFSEs are registered.

#### Parameters

- `base_path`: file path prefix associated with the filesystem. Must be a zero-terminated C string, 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” .
- `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.

`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.

**Return** 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.

#### Parameters

- `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).

`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`.

**Return** 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.

#### Parameters

- `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.

`esp_err_t esp_vfs_unregister (const char *base_path)`

Unregister a virtual filesystem for given path prefix

**Return** ESP\_OK if successful, ESP\_ERR\_INVALID\_STATE if VFS for given prefix hasn't been registered

**Parameters**

- `base_path`: file prefix previously used in `esp_vfs_register` call

`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`.

**Return** 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.

**Parameters**

- `vfs_id`: VFS identifier returned by `esp_vfs_register_with_id`.
- `fd`: The registered file descriptor will be written to this address.

`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`.

**Return** ESP\_OK if the registration is successful, ESP\_ERR\_INVALID\_ARG if the arguments are incorrect.

**Parameters**

- `vfs_id`: VFS identifier returned by `esp_vfs_register_with_id`.
- `fd`: File descriptor which should be unregistered.

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.

**Return** The number of descriptors set in the descriptor sets, or -1 when an error (specified by `errno`) have occurred.

**Parameters**

- `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.

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`.

**Parameters**

- `sem`: semaphore structure which was passed to the driver by the `start_select` call

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`.

**Parameters**

- `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

int **esp\_vfs\_poll** (**struct** pollfd \**fds*, nfd\_t *nfds*, int *timeout*)

Implements the VFS layer for synchronous I/O multiplexing by `poll()`

The implementation is based on `esp_vfs_select`. The parameters and return values are compatible with POSIX `poll()`.

**Return** A positive return value indicates the number of file descriptors that have been selected. The 0 return value indicates a timed-out poll. -1 is return on failure and `errno` is set accordingly.

**Parameters**

- `fds`: Pointer to the array containing file descriptors and events `poll()` should consider.
- `nfds`: Number of items in the array `fds`.
- `timeout`: `Poll()` should wait at least `timeout` milliseconds. If the value is 0 then it should return immediately. If the value is -1 then it should wait (block) until the event occurs.

ssize\_t **esp\_vfs\_pread** (int *fd*, void \**dst*, size\_t *size*, off\_t *offset*)

Implements the VFS layer of POSIX `pread()`

**Return** A positive return value indicates the number of bytes read. -1 is return on failure and `errno` is set accordingly.

**Parameters**

- `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

ssize\_t **esp\_vfs\_pwrite** (int *fd*, const void \**src*, size\_t *size*, off\_t *offset*)

Implements the VFS layer of POSIX `pwrite()`

**Return** A positive return value indicates the number of bytes written. -1 is return on failure and `errno` is set accordingly.

**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

**Structures**

**struct esp\_vfs\_select\_sem\_t**

VFS semaphore type for `select()`

**Public Members**

bool **is\_sem\_local**

type of "sem" is `SemaphoreHandle_t` when true, defined by socket driver otherwise

void \***sem**

semaphore instance

**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

#### int flags

`ESP_VFS_FLAG_CONTEXT_PTR` or `ESP_VFS_FLAG_DEFAULT`

*esp\_err\_t* (\***start\_select**) (int nfd, 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 nfd, 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**) ()

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

- `vfs/include/esp_vfs_dev.h`

### Functions

```
void esp_vfs_dev_uart_register ()
```

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

**Note** this function is not thread safe w.r.t. reading from UART

**Parameters**

- 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

**Note** this function is not thread safe w.r.t. writing to UART

**Parameters**

- 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

**Note** this function is not thread safe w.r.t. reading from UART

**Return** 0 if succeeded, or -1 when an error (specified by errno) have occurred.

**Parameters**

- uart\_num: the UART number
- mode: line endings to send to UART

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 ( ‘ ’ , 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

**Return** 0 if succeeded, or -1 when an error (specified by errno) have occurred.

**Parameters**

- uart\_num: the UART number
- mode: line endings to send to UART

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

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.

**Parameters**

- `uart_num`: UART peripheral number

**Enumerations**

**enum** `esp_line_endings_t`

Line ending settings.

Values:

**ESP\_LINE\_ENDINGS\_CRLF**

CR + LF.

**ESP\_LINE\_ENDINGS\_CR**

CR.

**ESP\_LINE\_ENDINGS\_LF**

LF.

## 2.6.6 FAT Filesystem Support

ESP-IDF uses the `FatFs` library to work with FAT filesystems. `FatFs` resides in the `fatfs` component. Although the library can be used directly, many of its features can be accessed via VFS, using the C standard library and POSIX API functions.

Additionally, `FatFs` has been modified to support the runtime pluggable disk I/O layer. This allows mapping of `FatFs` drives to physical disks at runtime.

### Using FatFs with VFS

The header file `fatfs/vfs/esp_vfs_fat.h` defines the functions for connecting `FatFs` and VFS.

The function `esp_vfs_fat_register()` allocates a `FATFS` structure and registers a given path prefix in VFS. Subsequent operations on files starting with this prefix are forwarded to `FatFs` APIs. The function `esp_vfs_fat_unregister_path()` deletes the registration with VFS, and frees the `FATFS` structure.

Most applications use the following workflow when working with `esp_vfs_fat_` functions:

1. Call `esp_vfs_fat_register()` to specify:
  - Path prefix where to mount the filesystem (e.g. `"/sdcard"`, `"/spiflash"`)
  - `FatFs` drive number
  - A variable which will receive the pointer to the `FATFS` structure
2. Call `ff_diskio_register()` to register the disk I/O driver for the drive number used in Step 1.
3. Call the `FatFs` function `f_mount`, and optionally `f_fdisk`, `f_mkfs`, to mount the filesystem using the same drive number which was passed to `esp_vfs_fat_register()`. For more information, see *FatFs documentation* <<http://www.elm-chan.org/ffsw/ff/doc/mount.html>>.
4. Call the C standard library and POSIX API functions to perform such actions on files as open, read, write, erase, copy, etc. Use paths starting with the path prefix passed to `esp_vfs_fat_register()` (for example, `"/sdcard/hello.txt"`).
5. Optionally, call the `FatFs` library functions directly. In this case, use paths without a VFS prefix (for example, `"/hello.txt"`).
6. Close all open files.
7. Call the `FatFs` function `f_mount` for the same drive number, with `NULL FATFS*` argument, to unmount the filesystem.
8. Call the `FatFs` function `ff_diskio_unregister()` with `NULL ff_diskio_impl_t*` argument and the same drive number to unregister the disk I/O driver.
9. Call `esp_vfs_fat_unregister_path()` with the path where the file system is mounted to remove `FatFs` from VFS, and free the `FATFS` structure allocated in Step 1.

The convenience functions `esp_vfs_fat_sdmmc_mount` and `esp_vfs_fat_sdmmc_unmount` wrap the steps described above and also handle SD card initialization. These two functions are described in the next section.

*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.

**Note** 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.

#### Return

- 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

#### Parameters

- *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*] *out\_fs*: pointer to FATFS structure which can be used for FATFS `f_mount` call is returned via this argument.

*esp\_err\_t* **esp\_vfs\_fat\_unregister\_path** (**const** char \**base\_path*)

Un-register FATFS from VFS.

**Note** 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_path`. 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

#### Return

- ESP\_OK on success
- ESP\_ERR\_INVALID\_STATE if FATFS is not registered in VFS

#### Parameters

- *base\_path*: path prefix where FATFS is registered. This is the same used when `esp_vfs_fat_register` was called

## Using FatFs with VFS and SD cards

The header file `fatfs/vfs/esp_vfs_fat.h` defines convenience functions `esp_vfs_fat_sdmmc_mount()` and `esp_vfs_fat_sdmmc_unmount()`. These function perform Steps 1–3 and 7–9 respectively and handle SD card initialization, but provide only limited error handling. Developers are encouraged to check its source code and incorporate more advanced features into production applications.

The convenience function `esp_vfs_fat_sdmmc_unmount()` unmounts the filesystem and releases the resources acquired by `esp_vfs_fat_sdmmc_mount()`.

*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.

**Return**

- ESP\_OK on success
- ESP\_ERR\_INVALID\_STATE if esp\_vfs\_fat\_sdmmc\_mount was already called
- ESP\_ERR\_NO\_MEM if memory can not be allocated
- ESP\_FAIL if partition can not be mounted
- other error codes from SDMMC or SPI drivers, SDMMC protocol, or FATFS drivers

**Parameters**

- `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 SDSPI\_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. For SPI peripheral, pass a pointer to `sdspi_slot_config_t` structure initialized using SDSPI\_SLOT\_CONFIG\_DEFAULT.
- `mount_config`: pointer to structure with extra parameters for mounting FATFS
- `[out] out_card`: if not NULL, pointer to the card information structure will be returned via this argument

**struct esp\_vfs\_fat\_mount\_config\_t**

Configuration arguments for esp\_vfs\_fat\_sdmmc\_mount and esp\_vfs\_fat\_spiflash\_mount 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\_leveling, 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.

**esp\_err\_t esp\_vfs\_fat\_sdmmc\_unmount ()**

Unmount FAT filesystem and release resources acquired using esp\_vfs\_fat\_sdmmc\_mount.

**Return**

- ESP\_OK on success
- ESP\_ERR\_INVALID\_STATE if esp\_vfs\_fat\_sdmmc\_mount hasn't been called

**Using FatFs with VFS in read-only mode**

The header file `fatfs/vfs/esp_vfs_fat.h` also defines the convenience functions `esp_vfs_fat_rawflash_mount ()` and `esp_vfs_fat_rawflash_unmount ()`. These functions perform Steps 1-3 and 7-9 respectively for read-only FAT partitions. These are particularly helpful for data partitions written only once during factory provisioning which will not be changed by production application throughout the lifetime of the hardware.

**esp\_err\_t esp\_vfs\_fat\_rawflash\_mount (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

**Note** Wear levelling is not used when FAT is mounted in read-only mode using this function.

**Return**

- `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_rawflash_mount` was already called for the same partition
- `ESP_ERR_NO_MEM` if memory can not be allocated
- `ESP_FAIL` if partition can not be mounted
- other error codes from SPI flash driver, or FATFS drivers

**Parameters**

- `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

```
esp_err_t esp_vfs_fat_rawflash_unmount (const char *base_path, const char
                                     *partition_label)
```

Unmount FAT filesystem and release resources acquired using `esp_vfs_fat_rawflash_mount`.

**Return**

- `ESP_OK` on success
- `ESP_ERR_INVALID_STATE` if `esp_vfs_fat_spiflash_mount` hasn't been called

**Parameters**

- `base_path`: path where partition should be registered (e.g. `"/spiflash"` )
- `partition_label`: label of partition to be unmounted

## FatFS disk IO layer

FatFs has been extended with API functions that register the disk I/O driver at runtime.

They provide implementation of disk I/O functions for SD/MMC cards and can be registered for the given FatFs drive number using the function `ff_diskio_register_sdmmc()`.

```
void ff_diskio_register (BYTE pdrv, const ff_diskio_impl_t *discio_impl)
```

Register or unregister diskio 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.

**Parameters**

- `pdrv`: drive number
- `discio_impl`: pointer to `ff_diskio_impl_t` structure with diskio functions or `NULL` to unregister and free previously registered drive

```
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

`DSTATUS (*status)` (unsigned char pdrv)  
disk status check function

`DRESULT (*read)` (unsigned char pdrv, unsigned char \*buff, uint32\_t sector, unsigned count)  
sector read function

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 diskio driver

#### Parameters

- 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

#### Parameters

- 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

#### Parameters

- pdrv: drive number
- part\_handle: pointer to raw flash partition.

## 2.6.7 Wear Levelling API

### Overview

Most of flash memory and especially SPI flash that is used in ESP32 has a sector-based organization and also has a limited number of erase/modification cycles per memory sector. The wear levelling component helps to distribute wear and tear among sectors more evenly without requiring any attention from the user.

The wear levelling component provides API functions related to reading, writing, erasing, and memory mapping of data in external SPI flash through the partition component. The component also has higher-level API functions which work with the FAT filesystem defined in *FAT filesystem*.

The wear levelling component, together with the FAT FS component, uses FAT FS sectors of 4096 bytes, which is a standard size for flash memory. With this size, the component shows the best performance but needs additional memory in RAM.

To save internal memory, the component has two additional modes which both use sectors of 512 bytes:

- **Performance mode.** Erase sector operation data is stored in RAM, the sector is erased, and then data is copied back to flash memory. However, if a device is powered off for any reason, all 4096 bytes of data is lost.
- **Safety mode.** The data is first saved to flash memory, and after the sector is erased, the data is saved back. If a device is powered off, the data can be recovered as soon as the device boots up.

The default settings are as follows: - Sector size is 512 bytes - Performance mode

You can change the settings through the configuration menu.

The wear levelling component does not cache data in RAM. The write and erase functions modify flash directly, and flash contents are consistent when the function returns.

### Wear Levelling access API functions

This is the set of API functions for working with data in flash:

- `wl_mount` - initializes the wear levelling module and mounts the specified partition
- `wl_unmount` - unmounts the partition and deinitializes the wear levelling module
- `wl_erase_range` - erases a range of addresses in flash
- `wl_write` - writes data to a partition
- `wl_read` - reads data from a partition
- `wl_size` - returns the size of available memory in bytes
- `wl_sector_size` - returns the size of one sector

As a rule, try to avoid using raw wear levelling functions and use filesystem-specific functions instead.

## Memory Size

The memory size is calculated in the wear levelling module based on partition parameters. The module uses some sectors of flash for internal data.

## See also

- [FAT Filesystem](#)
- [Partition Table documentation](#)

## Application Example

An example which combines the wear levelling driver with the FATFS library is provided in the [storage/wear\\_levelling](#) directory. This example initializes the wear levelling driver, mounts FATFS partition, as well as writes and reads data from it using POSIX and C library APIs. See the [storage/wear\\_levelling/README.md](#) file for more information.

## High level API Reference

### Header Files

- [fatfs/src/esp\\_vfs\\_fat.h](#)

### Functions

```
esp_err_t esp_vfs_fat_spiflash_mount (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.

### Return

- `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` was already called
- `ESP_ERR_NO_MEM` if memory can not be allocated
- `ESP_FAIL` if partition can not be mounted
- other error codes from wear levelling library, SPI flash driver, or FATFS drivers

### Parameters

- `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

- [out] `wl_handle`: wear levelling driver handle

**struct esp\_vfs\_fat\_mount\_config\_t**

Configuration arguments for `esp_vfs_fat_sdmmc_mount` and `esp_vfs_fat_spiflash_mount` 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  $128 * \text{sector size}$ . 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.

*esp\_err\_t* **esp\_vfs\_fat\_spiflash\_unmount** (`const char *base_path`, *wl\_handle\_t* `wl_handle`)

Unmount FAT filesystem and release resources acquired using `esp_vfs_fat_spiflash_mount`.

### Return

- `ESP_OK` on success
- `ESP_ERR_INVALID_STATE` if `esp_vfs_fat_spiflash_mount` hasn't been called

### Parameters

- `base_path`: path where partition should be registered (e.g. `"/spiflash"`)
- `wl_handle`: wear levelling driver handle returned by `esp_vfs_fat_spiflash_mount`

## Mid level API Reference

### Header File

- [wear\\_levelling/include/wear\\_levelling.h](#)

### Functions

*esp\_err\_t* **wl\_mount** (`const esp_partition_t *partition`, *wl\_handle\_t* `*out_handle`)

Mount WL for defined partition.

### Return

- `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;

### Parameters

- `partition`: that will be used for access
- `out_handle`: handle of the WL instance

*esp\_err\_t* **wl\_unmount** (*wl\_handle\_t* `handle`)

Unmount WL for defined partition.

### Return

- `ESP_OK`, if the operation completed successfully;
- or one of error codes from lower-level flash driver.

### Parameters

- `handle`: WL partition handle

*esp\_err\_t* **wl\_erase\_range** (*wl\_handle\_t* `handle`, `size_t start_addr`, `size_t size`)

Erase part of the WL storage.



**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.

**Parameters**

- *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(...)`.

`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.

**Note** Prior to writing to WL storage, make sure it has been erased with `wl_erase_range` call.

**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.

**Parameters**

- *handle*: WL handle that are related to the partition
- *dest\_addr*: 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.

`esp_err_t wl_read(wl_handle_t handle, size_t src_addr, void *dest, size_t size)`

Read data from the WL storage.

**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.

**Parameters**

- *handle*: WL module instance that was initialized before
- *dest*: Pointer to the buffer where data should be stored. Pointer must be non-NULL and buffer must be at least 'size' bytes long.
- *src\_addr*: Address of the data to be read, relative to the beginning of the partition.
- *size*: Size of data to be read, in bytes.

`size_t wl_size(wl_handle_t handle)`

Get size of the WL storage.

**Return** usable size, in bytes

**Parameters**

- *handle*: WL module handle that was initialized before

`size_t wl_sector_size(wl_handle_t handle)`

Get sector size of the WL instance.

**Return** sector size, in bytes

**Parameters**

- *handle*: WL module handle that was initialized before

**Macros**

**WL\_INVALID\_HANDLE**

**Type Definitions**

```
typedef int32_t wl_handle_t
        wear levelling handle
```

**2.6.8 SPIFFS Filesystem****Overview**

SPIFFS is a file system intended for SPI NOR flash devices on embedded targets. It supports wear levelling, file system consistency checks, and more.

**Notes**

- Currently, SPIFFS does not support directories, it produces a flat structure. If SPIFFS is mounted under `/spiffs`, then creating a file with the path `/spiffs/tmp/myfile.txt` will create a file called `/tmp/myfile.txt` in SPIFFS, instead of `myfile.txt` in the directory `/spiffs/tmp`.
- It is not a real-time stack. One write operation might take much longer than another.
- For now, it does not detect or handle bad blocks.

**Tools**

**spiffsgen.py** `spiffsgen.py` is a write-only Python SPIFFS implementation used to create filesystem images from the contents of a host folder. To use `spiffsgen.py`, open Terminal and run:

```
python spiffsgen.py <image_size> <base_dir> <output_file>
```

The required arguments are as follows:

- **image\_size**: size of the partition onto which the created SPIFFS image will be flashed.
- **base\_dir**: directory for which the SPIFFS image needs to be created.
- **output\_file**: SPIFFS image output file.

There are also other arguments that control image generation. Documentation on these arguments can be found in the tool's help:

```
python spiffsgen.py --help
```

These optional arguments correspond to a possible SPIFFS build configuration. To generate the right image, please make sure that you use the same arguments/configuration as were used to build SPIFFS. As a guide, the help output indicates the SPIFFS build configuration to which the argument corresponds. In cases when these arguments are not specified, the default values shown in the help output will be used.

When the image is created, it can be flashed using `esptool.py` or `parttool.py`.

Aside from invoking the `spiffsgen.py` standalone by manually running it from the command line or a script, it is also possible to invoke `spiffsgen.py` directly from the build system by calling `spiffs_create_partition_image`.

Make:

```
SPIFFS_IMAGE_FLASH_IN_PROJECT := ...
SPIFFS_IMAGE_DEPENDS := ...
$(eval $(call spiffs_create_partition_image,<partition>,<base_dir>))
```

CMake:

```
spiffs_create_partition_image(<partition> <base_dir> [FLASH_IN_PROJECT] [DEPENDS_
↳dep dep dep...])
```

This is more convenient as the build configuration is automatically passed to the tool, ensuring that the generated image is valid for that build. An example of this is while the *image\_size* is required for the standalone invocation, only the *partition* name is required when using `spiffs_create_partition_image` –the image size is automatically obtained from the project’s partition table.

Due to the differences in structure between Make and CMake, it is important to note that: - for Make `spiffs_create_partition_image` must be called from the project Makefile - for CMake `spiffs_create_partition_image` must be called from one of the component CMakeLists.txt files

Optionally, user can opt to have the image automatically flashed together with the app binaries, partition tables, etc. on `idf.py flash` or `make flash` by specifying `FLASH_IN_PROJECT`. For example,

in Make:

```
SPIFFS_IMAGE_FLASH_IN_PROJECT := 1
$(eval $(call spiffs_create_partition_image, <partition>, <base_dir>))
```

in CMake:

```
spiffs_create_partition_image(my_spiffs_partition my_folder FLASH_IN_PROJECT)
```

If `FLASH_IN_PROJECT/SPIFFS_IMAGE_FLASH_IN_PROJECT` is not specified, the image will still be generated, but you will have to flash it manually using `esptool.py`, `parttool.py`, or a custom build system target.

There are cases where the contents of the base directory itself is generated at build time. Users can use `DEPENDS/SPIFFS_IMAGE_DEPENDS` to specify targets that should be executed before generating the image.

in Make:

```
dep:
    ...

SPIFFS_IMAGE_DEPENDS := dep
$(eval $(call spiffs_create_partition_image, <partition>, <base_dir>))
```

in CMake:

```
add_custom_target(dep COMMAND ...)

spiffs_create_partition_image(my_spiffs_partition my_folder DEPENDS dep)
```

+For an example, see [examples/storage/spiffsgen](#).

**mkspiffs** Another tool for creating SPIFFS partition images is `mkspiffs`. Similar to `spiffsgen.py`, it can be used to create an image from a given folder and then flash that image using `esptool.py`

For that, you need to obtain the following parameters:

- **Block Size:** 4096 (standard for SPI Flash)
- **Page Size:** 256 (standard for SPI Flash)
- **Image Size:** Size of the partition in bytes (can be obtained from a partition table)
- **Partition Offset:** Starting address of the partition (can be obtained from a partition table)

To pack a folder into a 1-Megabyte image, run:

```
mkspiffs -c [src_folder] -b 4096 -p 256 -s 0x100000 spiffs.bin
```

To flash the image onto ESP32 at offset 0x110000, run:

```
python esptool.py --chip esp32 --port [port] --baud [baud] write_flash -z 0x110000_
↪spiffs.bin
```

**Notes on which SPIFFS tool to use** The two tools presented above offer very similar functionality. However, there are reasons to prefer one over the other, depending on the use case.

Use `spiffsgen.py` in the following cases: 1. If you want to simply generate a SPIFFS image during the build, `spiffsgen.py` makes it very convenient by providing functions/commands from the build system itself. 2. If the host has no C/C++ compiler available, because `spiffsgen.py` does not require compilation.

Use `mkspiffs` in the following cases: 1. If you need to unpack SPIFFS images in addition to image generation. For now, it is not possible with `spiffsgen.py`. 2. If you have an environment where a Python interpreter is not available, but a host compiler is available. Otherwise, a pre-compiled `mkspiffs` binary can do the job. However, there is no build system integration for `mkspiffs` and the user has to do the corresponding work: compiling `mk-spiffs` during build (if a pre-compiled binary is not used), creating build rules/targets for the output files, passing proper parameters to the tool, etc.

### See also

- [Partition Table documentation](#)

### Application Example

An example of using SPIFFS is provided in the [storage/spiffs](#) directory. This example initializes and mounts a SPIFFS partition, then writes and reads data from it using POSIX and C library APIs. See the README.md file in the example directory for more information.

### High-level API Reference

#### Header File

- [spiffs/include/esp\\_spiffs.h](#)

#### Functions

`esp_err_t esp_vfs_spiffs_register(const esp_vfs_spiffs_conf_t *conf)`

Register and mount SPIFFS to VFS with given path prefix.

##### Return

- 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

##### Parameters

- `conf`: Pointer to `esp_vfs_spiffs_conf_t` configuration structure

`esp_err_t esp_vfs_spiffs_unregister(const char *partition_label)`

Unregister and unmount SPIFFS from VFS

##### Return

- ESP\_OK if successful
- ESP\_ERR\_INVALID\_STATE already unregistered

##### Parameters

- `partition_label`: Optional, label of the partition to unregister. If not specified, first partition with subtype=spiffs is used.

bool `esp_spiffs_mounted(const char *partition_label)`

Check if SPIFFS is mounted

##### Return

- true if mounted
- false if not mounted

##### Parameters

- `partition_label`: Optional, label of the partition to check. If not specified, first partition with `subtype=spiffs` is used.

*esp\_err\_t* **esp\_spiffs\_format** (`const char *partition_label`)

Format the SPIFFS partition

#### Return

- `ESP_OK` if successful
- `ESP_FAIL` on error

#### Parameters

- `partition_label`: Optional, label of the partition to format. If not specified, first partition with `subtype=spiffs` is used.

*esp\_err\_t* **esp\_spiffs\_info** (`const char *partition_label`, `size_t *total_bytes`, `size_t *used_bytes`)

Get information for SPIFFS

#### Return

- `ESP_OK` if success
- `ESP_ERR_INVALID_STATE` if not mounted

#### Parameters

- `partition_label`: Optional, label of the partition to get info for. If not specified, first partition with `subtype=spiffs` is used.
- `[out] total_bytes`: Size of the file system
- `[out] used_bytes`: Current used bytes in the file system

## 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.

## 2.6.9 Manufacturing Utility

### Introduction

This utility is designed to create instances of factory NVS partition images on a per-device basis for mass manufacturing purposes. The NVS partition images are created from CSV files containing user-provided configurations and values.

Please note that this utility only creates manufacturing binary images which then need to be flashed onto your devices using:

- `esptool.py`
- Flash Download tool (available on Windows only)
- Direct flash programming

## Prerequisites

This utility is dependent on esp-idf's NVS partition utility.

- **Operating System requirements:**
  - Linux / MacOS / Windows (standard distributions)
- **The following packages are needed to use this utility:**
  - Python: <<https://www.python.org/downloads/>>

注解:

**Before using this utility, please make sure that:**

- The path to Python is added to the PATH environment variable.
- You have installed the packages from *requirement.txt*, the file in the root of the esp-idf directory.

## Workflow



## CSV Configuration File

This file contains the configuration of the device to be flashed.

The data in the configuration file has the following format (the *REPEAT* tag is optional):

```

name1,namespace,    <-- First entry should be of type "namespace"
key1,type1,encoding1
key2,type2,encoding2,REPEAT
name2,namespace,
key3,type3,encoding3
key4,type4,encoding4
  
```

注解: The first line in this file should always be the namespace entry.

Each line should have three parameters: *key*, *type*, *encoding*, separated by a comma. If the *REPEAT* tag is present, the value corresponding to this key in the master value CSV file will be the same for all devices.

Please refer to *README of the NVS Partition Generator utility* for detailed description of each parameter.

Below is a sample example of such a configuration file:

```

app,namespace,
firmware_key,data,hex2bin
serial_no,data,string,REPEAT
device_no,data,i32
  
```

注解:

**Make sure there are no spaces:**

- before and after ‘,’
- at the end of each line in a CSV file

## Master Value CSV File

This file contains details of the devices to be flashed. Each line in this file corresponds to a device instance.

The data in the master value CSV file has the following format:

```
key1, key2, key3, .....
value1, value2, value3, .....
```

**注解:** The first line in the file should always contain the `key` names. All the keys from the configuration file should be present here in the **same order**. This file can have additional columns (keys). The additional keys will be treated as metadata and would not be part of the final binary files.

Each line should contain the `value` of the corresponding keys, separated by a comma. If the key has the `REPEAT` tag, its corresponding value **must** be entered in the second line only. Keep the entry empty for this value in the following lines.

The description of this parameter is as follows:

**value** Data value

Data value is the value of data corresponding to the key.

Below is a sample example of a master value CSV file:

```
id, firmware_key, serial_no, device_no
1, 1a2b3c4d5e6faabb, A1, 101
2, 1a2b3c4d5e6fccdd, , 102
3, 1a2b3c4d5e6feeff, , 103
```

**注解:** If the `'REPEAT'` tag is present, a new master value CSV file will be created in the same folder as the input Master CSV File with the values inserted at each line for the key with the `'REPEAT'` tag.

This utility creates intermediate CSV files which are used as input for the NVS partition utility to generate the binary files.

The format of this intermediate CSV file is as follows:

```
key, type, encoding, value
key, namespace, ,
key1, type1, encoding1, value1
key2, type2, encoding2, value2
```

An instance of an intermediate CSV file will be created for each device on an individual basis.

## Running the utility

**Usage:**

```
python mfg_gen.py [-h] {generate, generate-key} ...
```

Optional Arguments:

```
+-----+-----+-----+-----+
| No. | Parameter | Description |
+-----+-----+-----+
↩
```

(下页继续)

(续上页)

```

=====+=====+=====+=====+=====+=====+=====+=====+=====+
| 1 | -h, --help | show this help message and exit |
↩ |
+-----+
↩-----+
Commands:
    Run mfg_gen.py {command} -h for additional help
+-----+
↩-----+
| No. | Parameter | Description |
↩ |
+-----+=====+=====+=====+=====+=====+=====+=====+=====+=====+
| 1 | generate | Generate NVS partition |
↩ |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
↩-----+
| 2 | generate-key | Generate keys for encryption |
↩ |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
↩-----+

```

**To generate factory images for each device (Default): Usage:**

```
python mfg_gen.py generate [-h] [--fileid FILEID] [--version {1,2}] [--keygen]
                        [--keyfile KEYFILE] [--inputkey INPUTKEY]
                        [--outdir OUTDIR]
                        conf values prefix size
```

## Positional Arguments:

```

+-----+-----+-----+-----+-----+-----+-----+-----+-----+
↩-----+
| Parameter | Description |
↩ |
+-----+=====+=====+=====+=====+=====+=====+=====+=====+=====+
| conf | Path to configuration csv file to parse |
↩ |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
↩-----+
| values | Path to values csv file to parse |
↩ |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
↩-----+
| prefix | Unique name for each output filename prefix |
↩ |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
↩-----+
| size | Size of NVS partition in bytes |
↩ |
| | (must be multiple of 4096) |
↩ |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
↩-----+

```

## Optional Arguments:

```

+-----+-----+-----+-----+-----+-----+-----+-----+-----+
↩-----+
| Parameter | Description |
↩ |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| -h, --help | show this help message and exit |
↩ |

```

(下页继续)



(续上页)

<code>--fileid FILEID</code>	Unique file identifier (any key <b>in</b> values file)	
	<b>for</b> each filename suffix (Default: numeric value (1, 2, 3...))	
<code>--version {1,2}</code>	Set multipage blob version.	
	Version 1 - Multipage blob support disabled.	
	Version 2 - Multipage blob support enabled.	
	Default: Version 2	
<code>--keygen</code>	Generates key <b>for</b> encrypting NVS partition	
<code>--inputkey INPUTKEY</code>	File having key <b>for</b> encrypting NVS partition	
<code>--outdir OUTDIR</code>	Output directory to store files created	
	(Default: current directory)	

You can run the utility to generate factory images for each device using the command below. A sample CSV file is provided with the utility:

```
python mfg_gen.py generate samples/sample_config.csv samples/sample_values_singlepage_
↪blob.csv Sample 0x3000
```

The master value CSV file should have the path in the `file` type relative to the directory from which you are running the utility.

### To generate encrypted factory images for each device:

You can run the utility to encrypt factory images for each device using the command below. A sample CSV file is provided with the utility:

- Encrypt by allowing the utility to generate encryption keys:

```
python mfg_gen.py generate samples/sample_config.csv samples/sample_values_
↪singlepage_blob.csv Sample 0x3000 --keygen
```

**注解:** Encryption key of the following format `<outdir>/keys/keys-<prefix>-<fileid>.bin` is created.

**注解:** This newly created file having encryption keys in `keys/` directory is compatible with NVS key-partition structure. Refer to *NVS key partition* for more details.

- Encrypt by providing the encryption keys as input binary file:

```
python mfg_gen.py generate samples/sample_config.csv samples/sample_values_
↳singlepage_blob.csv Sample 0x3000 --inputkey keys/sample_keys.bin
```

### To generate only encryption keys: Usage:

```
python mfg_gen.py generate-key [-h] [--keyfile KEYFILE] [--outdir OUTDIR]
```

Optional Arguments:

Parameter	Description
-h, --help	show this help message <b>and</b> exit
--keyfile KEYFILE	Path to output encryption keys file
--outdir OUTDIR	Output directory to store files created. (Default: current directory)

You can run the utility to generate only encryption keys using the command below:

```
python mfg_gen.py generate-key
```

**注解:** Encryption key of the following format `<outdir>/keys/keys-<timestamp>.bin` is created. Timestamp format is: `%m-%d_%H-%M`.

**注解:** To provide custom target filename use the `--keyfile` argument.

Generated encryption key binary file can further be used to encrypt factory images created on the per device basis.

The default numeric value: 1,2,3... of the `fileid` argument corresponds to each line bearing device instance values in the master value CSV file.

While running the manufacturing utility, the following folders will be created in the specified `outdir` directory:

- `bin/` for storing the generated binary files
- `csv/` for storing the generated intermediate CSV files
- `keys/` for storing encryption keys (when generating encrypted factory images)

Code examples for this API section are provided in the [storage](#) directory of ESP-IDF examples.

## 2.7 System API

### 2.7.1 FreeRTOS

## Overview

This section contains documentation of FreeRTOS types, functions, and macros. It is automatically generated from FreeRTOS header files.

**注解:** ESP-IDF FreeRTOS is based on the Xtensa port of FreeRTOS v8.2.0, however some functions of FreeRTOS v9.0.0 have been backported. See the [Backported Features](#) for more information.

For more information about FreeRTOS features specific to ESP-IDF, see [ESP-IDF FreeRTOS SMP Changes](#) and [ESP-IDF FreeRTOS Additions](#).

## Task API

### Header File

- [freertos/include/freertos/task.h](#)

### Functions

**BaseType\_t xTaskCreatePinnedToCore** (TaskFunction\_t *pvTaskCode*, **const** char \***const** *pcName*, **const** uint32\_t *usStackDepth*, void \***const** *pvParameters*, UBaseType\_t *uxPriority*, *TaskHandle\_t* \***const** *pvCreatedTask*, **const** BaseType\_t *xCoreID*)

Create a new task with a specified affinity.

This function is similar to xTaskCreate, but allows setting task affinity in SMP system.

**Return** pdPASS if the task was successfully created and added to a ready list, otherwise an error code defined in the file projdefs.h

#### Parameters

- *pvTaskCode*: Pointer to the task entry function. Tasks must be implemented to never return (i.e. continuous loop).
- *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*: 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. Other values indicate the index number of the CPU which the task should be pinned to. Specifying values larger than (portNUM\_PROCESSORS - 1) will cause the function to fail.

**static BaseType\_t xTaskCreate** (TaskFunction\_t *pvTaskCode*, **const** char \***const** *pcName*, **const** uint32\_t *usStackDepth*, void \***const** *pvParameters*, UBaseType\_t *uxPriority*, *TaskHandle\_t* \***const** *pvCreatedTask*)

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 <http://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:

```
// 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, 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 );
    }
}
```

**Return** `pdPASS` if the task was successfully created and added to a ready list, otherwise an error code defined in the file `projdefs.h`

**Note** If program uses thread local variables (ones specified with “`__thread`” keyword) then storage for them will be allocated on the task’s stack.

#### Parameters

- `pvTaskCode`: Pointer to the task entry function. Tasks must be implemented to never return (i.e. continuous loop).
- `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`: Used to pass back a handle by which the created task can be referenced.

*TaskHandle\_t* **xTaskCreateStaticPinnedToCore** (TaskFunction\_t *pvTaskCode*, **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.

This function is similar to `xTaskCreateStatic`, but allows specifying task affinity in an SMP system.

**Return** 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.

#### Parameters

- `pvTaskCode`: Pointer to the task entry function. Tasks must be implemented to never return (i.e. continuous loop).
- `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. Other values indicate the index number of the CPU which the task should be pinned to. Specifying values larger than `(portNUM_PROCESSORS - 1)` will cause the function to fail.

**static TaskHandle\_t** **xTaskCreateStatic** (TaskFunction\_t *pvTaskCode*, **const** char \***const** *pcName*, **const** uint32\_t *ulStackDepth*, void \***const** *pvParameters*, UBaseType\_t *uxPriority*, StackType\_t \***const** *pxStackBuffer*, 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 <http://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:

```
// 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.
#define STACK_SIZE 200

// 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 ];
```

(下页继续)

```

// 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 );

    for( ;; )
    {
        // 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,           // Function that implements the task.
        "NAME",             // Text name for the task.
        STACK_SIZE,        // Stack size in bytes, not words.
        ( void * ) 1,       // Parameter passed into the task.
        tskIDLE_PRIORITY, // Priority at which the task is created.
        xStack,             // Array to use as the task's stack.
        &xTaskBuffer );    // Variable to hold the task's data_
↪structure.

    // 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 );
}

```

**Return** 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.

**Note** If program uses thread local variables (ones specified with “\_\_thread” keyword) then storage for them will be allocated on the task’ s stack.

#### Parameters

- pvTaskCode: Pointer to the task entry function. Tasks must be implemented to never return (i.e. continuous loop).
- 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.

void **vTaskDelete** (*TaskHandle\_t* xTaskToDelete)

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.

INCLUDE\_vTaskDelete must be defined as 1 for this function to be available. See the configuration section

for more information.

See the demo application file `death.c` for sample code that utilises `vTaskDelete ()`.

**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.

Example usage:

```
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 );
}
```

### Parameters

- `xTaskToDelete`: The handle of the task to be deleted. Passing `NULL` will cause the calling task to be deleted.

void **vTaskDelay** (const TickType\_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. See `vTaskDelayUntil()` 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:

```
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 );
    }
}
```

### Parameters

- `xTicksToDelay`: The amount of time, in tick periods, that the calling task should block.

void **vTaskDelayUntil** (TickType\_t \*const *pxPreviousWakeTime*, const TickType\_t *xTimeIncrement*)

Delay a task until a specified time.

INCLUDE\_vTaskDelayUntil must be defined as 1 for this function to be available. See the configuration section for more information.

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 though 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, vTaskDelayUntil () specifies the absolute (exact) time at which it wishes to unblock.

The constant portTICK\_PERIOD\_MS can be used to calculate real time from the tick rate - with the resolution of one tick period.

Example usage:

```
// Perform an action every 10 ticks.
void vTaskFunction( void * pvParameters )
{
    TickType_t xLastWakeTime;
    const TickType_t xFrequency = 10;

    // Initialise the xLastWakeTime variable with the current time.
    xLastWakeTime = xTaskGetTickCount ();
    for ( ;; )
    {
        // Wait for the next cycle.
        vTaskDelayUntil( &xLastWakeTime, xFrequency );

        // Perform action here.
    }
}
```

### Parameters

- 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 vTaskDelayUntil ().
- xTimeIncrement: The cycle time period. The task will be unblocked at time \*pxPreviousWakeTime + xTimeIncrement. Calling vTaskDelayUntil with the same xTimeIncrement parameter value will cause the task to execute with a fixed interface period.

UBaseType\_t uxTaskPriorityGet (TaskHandle\_t xTask)

Obtain the priority of any task.

INCLUDE\_uxTaskPriorityGet must be defined as 1 for this function to be available. See the configuration section for more information.

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 obtain the priority of the created task.
    // It was created with tskIDLE_PRIORITY, but may have changed
```

(下页继续)



```

// it itself.
if( uxTaskPriorityGet( xHandle ) != tskIDLE_PRIORITY )
{
    // The task has changed it's priority.
}

// ...

// Is our priority higher than the created task?
if( uxTaskPriorityGet( xHandle ) < uxTaskPriorityGet( NULL ) )
{
    // Our priority (obtained using NULL handle) is higher.
}
}

```

**Return** The priority of xTask.

**Parameters**

- xTask: Handle of the task to be queried. Passing a NULL handle results in the priority of the calling task being returned.

UBaseType\_t **uxTaskPriorityGetFromISR** (*TaskHandle\_t* xTask)

A version of uxTaskPriorityGet() that can be used from an ISR.

**Return** The priority of xTask.

**Parameters**

- xTask: Handle of the task to be queried. Passing a NULL handle results in the priority of the calling task being returned.

*eTaskState* **eTaskGetState** (*TaskHandle\_t* xTask)

Obtain the state of any task.

States are encoded by the eTaskState enumerated type.

INCLUDE\_eTaskGetState must be defined as 1 for this function to be available. See the configuration section for more information.

**Return** 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.

**Parameters**

- xTask: Handle of the task to be queried.

void **vTaskPrioritySet** (*TaskHandle\_t* xTask, UBaseType\_t uxNewPriority)

Set the priority of any task.

INCLUDE\_vTaskPrioritySet must be defined as 1 for this function to be available. See the configuration section for more information.

A context switch will occur before the function returns if the priority being set is higher than the currently executing task.

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 raise the priority of the created task.

```

(下页继续)

```

vTaskPrioritySet( xHandle, tskIDLE_PRIORITY + 1 );

// ...

// Use a NULL handle to raise our priority to the same value.
vTaskPrioritySet( NULL, tskIDLE_PRIORITY + 1 );
}

```

**Parameters**

- `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.

void **vTaskSuspend** (*TaskHandle\_t* `xTaskToSuspend`)

Suspend a task.

INCLUDE\_vTaskSuspend must be defined as 1 for this function to be available. See the configuration section for more information.

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:

```

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 ).

//...

// Suspend ourselves.
vTaskSuspend( NULL );

// We cannot get here unless another task calls vTaskResume
// with our handle as the parameter.
}

```

**Parameters**

- `xTaskToSuspend`: Handle to the task being suspended. Passing a NULL handle will cause the calling task to be suspended.

void **vTaskResume** (*TaskHandle\_t* `xTaskToResume`)

Resumes a suspended task.

INCLUDE\_vTaskSuspend must be defined as 1 for this function to be available. See the configuration section for more information.

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.
}
```

### Parameters

- `xTaskToResume`: Handle to the task being readied.

`BaseType_t xTaskResumeFromISR ( TaskHandle_t xTaskToResume )`

An implementation of `vTaskResume()` that can be called from within an ISR.

`INCLUDE_xTaskResumeFromISR` must be defined as 1 for this function to be available. See the configuration section for more information.

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.

**Return** `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.

### Parameters

- `xTaskToResume`: Handle to the task being readied.

`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:

```

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 kernel
        // tick count will be maintained.

        // ...

        // The operation is complete. Restart the kernel.
        xTaskResumeAll ();
    }
}

```

**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:

```

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

```

(下页继续)

```

// a context switch - but there is no point if resuming the scheduler
// caused a context switch already.
if( !xTaskResumeAll () )
{
    taskYIELD ();
}
}
}

```

**Return** If resuming the scheduler caused a context switch then pdTRUE is returned, otherwise pdFALSE is returned.

TickType\_t **xTaskGetTickCount** (void)  
Get tick count

**Return** The count of ticks since vTaskStartScheduler was called.

TickType\_t **xTaskGetTickCountFromISR** (void)  
Get tick count from ISR

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.

**Return** The count of ticks since vTaskStartScheduler was called.

UBaseType\_t **uxTaskGetNumberOfTasks** (void)  
Get current number of tasks

**Return** 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.

char \***pcTaskGetTaskName** (*TaskHandle\_t* xTaskToQuery)  
Get task name

**Return** 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. INCLUDE\_pcTaskGetTaskName must be set to 1 in FreeRTOSConfig.h for pcTaskGetTaskName() 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.

High water mark is the minimum free stack space there has been (in bytes rather than words as found in vanilla FreeRTOS) since the task started. The smaller the returned number the closer the task has come to overflowing its stack.

**Return** The smallest amount of free stack space there has been (in bytes rather than words as found in vanilla FreeRTOS) since the task referenced by xTask was created.

#### Parameters

- xTask: Handle of the task associated with the stack to be checked. Set xTask to NULL to check the stack of the calling task.

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 highest stack memory address on architectures where the stack grows down from high memory, and the lowest memory address on architectures where the stack grows up from low memory.

**Return** A pointer to the start of the stack.

**Parameters**

- `xTask`: Handle of the task associated with the stack returned. Set `xTask` to `NULL` to return the stack of the calling task.

void **vTaskSetApplicationTaskTag** (*TaskHandle\_t* `xTask`, *TaskHookFunction\_t* `pxHookFunction`)  
Sets `pxHookFunction` to be the task hook function used by the task `xTask`.

**Parameters**

- `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`)  
Get the hook function assigned to given task.

**Return** The `pxHookFunction` value assigned to the task `xTask`.

**Parameters**

- `xTask`: Handle of the task to get the hook function for Passing `xTask` as `NULL` has the effect of getting the calling tasks hook function.

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.

**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.

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.

**Return** Pointer value

**Parameters**

- `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`.

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.

`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.

*TaskHandle\_t* `xTaskGetIdleTaskHandle` (void)

Get the handle of idle task for the current CPU.

`xTaskGetIdleTaskHandle()` is only available if `INCLUDE_xTaskGetIdleTaskHandle` is set to 1 in `FreeRTOSConfig.h`.

**Return** The handle of the idle task. It is not valid to call `xTaskGetIdleTaskHandle()` before the scheduler has been started.

*TaskHandle\_t* `xTaskGetIdleTaskHandleForCPU` (*UBaseType\_t* `cpuid`)

Get the handle of idle task for the given CPU.

`xTaskGetIdleTaskHandleForCPU()` is only available if `INCLUDE_xTaskGetIdleTaskHandle` is set to 1 in `FreeRTOSConfig.h`.

**Return** Idle task handle of a given cpu. It is not valid to call `xTaskGetIdleTaskHandleForCPU()` before the scheduler has been started.

#### Parameters

- `cpuid`: The CPU to get the handle for

*UBaseType\_t* `uxTaskGetSystemState` (*TaskStatus\_t* *\*const* `pxTaskStatusArray`, *const* *UBaseType\_t* `uxArraySize`, *uint32\_t* *\*const* `pulTotalRunTime`)

Get the state of tasks in the system.

`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:

```
// 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;
    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
```

(下页继续)

```

// 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%lu\t\t%lu%%\r\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%lu\t\t<1%\r\n",↳
↳pxTaskStatusArray[ x ].pcTaskName, pxTaskStatusArray[ x ].ulRunTimeCounter );
            }

            pcWriteBuffer += strlen( ( char * ) pcWriteBuffer );
        }
    }

    // The array is no longer needed, free the memory it consumes.
    vPortFree( pxTaskStatusArray );
}
}

```

**Note** This function is intended for debugging use only as its use results in the scheduler remaining suspended for an extended period.

**Return** 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.

**Parameters**

- `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 <http://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.

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.

Lists all the current tasks, along with their current state and stack usage high water mark.

**Note** This function will disable interrupts for its duration. It is not intended for normal application runtime use but as a debug aid.

Tasks are reported as blocked ( 'B' ), ready ( 'R' ), deleted ( 'D' ) or suspended ( 'S' ).

`vTaskList()` calls `uxTaskGetSystemState()`, then formats part of the `uxTaskGetSystemState()` output into a human readable table that displays task names, states and stack usage.

**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()` 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 `vTaskList()`.

#### Parameters

- `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.

void **vTaskGetRunTimeStats** (char \*pcWriteBuffer)

Get the state of running tasks as a string

`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 timers 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. 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** This function will disable interrupts for its duration. It is not intended for normal application runtime use but as a debug aid.

`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.

**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.

`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()`.

#### Parameters

- `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.

BaseType\_t **xTaskNotify** (*TaskHandle\_t xTaskToNotify*, uint32\_t *ulValue*, *eNotifyAction eAction*)  
Send task notification.

`configUSE_TASK_NOTIFICATIONS` must be undefined or defined as 1 for this function to be available.

When `configUSE_TASK_NOTIFICATIONS` is set to one each task has its own private “notification value”, which is a 32-bit unsigned integer (uint32\_t).

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 the task’s notification value. 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 `xTaskNotifyWait()` or `ulTaskNotifyTake()`. 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 `xTaskNotifyWait()` to [optionally] block to wait for a notification to be pending, or `ulTaskNotifyTake()` to [optionally] block to wait for its notification value to have a non-zero value. The task does not consume any CPU time while it is in the Blocked state.

See <http://www.FreeRTOS.org/RTOS-task-notifications.html> for details.

**Return** Dependent on the value of `eAction`. See the description of the `eAction` parameter.

#### Parameters

- `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`. `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.

BaseType\_t **xTaskNotifyFromISR** (*TaskHandle\_t xTaskToNotify*, uint32\_t *ulValue*, *eNotifyAction eAction*, BaseType\_t *\*pxHigherPriorityTaskWoken*)  
Send task notification from an ISR.

`configUSE_TASK_NOTIFICATIONS` must be undefined or defined as 1 for this function to be available.

When `configUSE_TASK_NOTIFICATIONS` is set to one each task has its own private “notification value”, which is a 32-bit unsigned integer (`uint32_t`).

A version of `xTaskNotify()` that can be used from an interrupt service routine (ISR).

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 the task’s notification value. 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 `xTaskNotifyWait()` or `ulTaskNotifyTake()`. 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 `xTaskNotifyWait()` to [optionally] block to wait for a notification to be pending, or `ulTaskNotifyTake()` to [optionally] block to wait for its notification value to have a non-zero value. The task does not consume any CPU time while it is in the Blocked state.

See <http://www.FreeRTOS.org/RTOS-task-notifications.html> for details.

**Return** Dependent on the value of `eAction`. See the description of the `eAction` parameter.

#### Parameters

- `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`. `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.
- `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.

`BaseType_t xTaskNotifyWait` (`uint32_t ulBitsToClearOnEntry`, `uint32_t ulBitsToClearOnExit`, `uint32_t *pulNotificationValue`, `TickType_t xTicksToWait`)

Wait for task notification

`configUSE_TASK_NOTIFICATIONS` must be undefined or defined as 1 for this function to be available.

When `configUSE_TASK_NOTIFICATIONS` is set to one each task has its own private “notification value”, which is a 32-bit unsigned integer (`uint32_t`).

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 the task's notification value. 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 `xTaskNotifyWait()` or `ulTaskNotifyTake()`. 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 `xTaskNotifyWait()` to [optionally] block to wait for a notification to be pending, or `ulTaskNotifyTake()` to [optionally] block to wait for its notification value to have a non-zero value. The task does not consume any CPU time while it is in the Blocked state.

See <http://www.FreeRTOS.org/RTOS-task-notifications.html> for details.

**Return** If a notification was received (including notifications that were already pending when `xTaskNotifyWait` was called) then `pdPASS` is returned. Otherwise `pdFAIL` is returned.

#### Parameters

- `ulBitsToClearOnEntry`: Bits that are set in `ulBitsToClearOnEntry` value will be cleared in the calling task's notification value before the task checks to see if any notifications are pending, and optionally blocks if no notifications are pending. Setting `ulBitsToClearOnEntry` 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. 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_TICSK( value_in_ms )` can be used to convert a time specified in milliseconds to a time specified in ticks.

```
void vTaskNotifyGiveFromISR (TaskHandle_t xTaskToNotify, BaseType_t
                             *pxHigherPriorityTaskWoken)
```

Simplified macro for sending task notification from ISR.

`configUSE_TASK_NOTIFICATIONS` must be undefined or defined as 1 for this macro to be available.

When `configUSE_TASK_NOTIFICATIONS` is set to one each task has its own private "notification value", which is a 32-bit unsigned integer (`uint32_t`).

A version of `xTaskNotifyGive()` that can be called from an interrupt service routine (ISR).

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 the task's notification value. 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.

`vTaskNotifyGiveFromISR()` 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 `vTaskNotifyGiveFromISR()`.

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 `ulTaskNotificationTake()` API function rather than the `xTaskNotifyWait()` API function.

See <http://www.FreeRTOS.org/RTOS-task-notifications.html> for more details.

#### Parameters

- `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()`.
- `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 ulTaskNotifyTake` (`BaseType_t xClearCountOnExit`, `TickType_t xTicksToWait`)

Simplified macro for receiving task notification.

`configUSE_TASK_NOTIFICATIONS` must be undefined or defined as 1 for this function to be available.

When `configUSE_TASK_NOTIFICATIONS` is set to one each task has its own private “notification value”, which is a 32-bit unsigned integer (`uint32_t`).

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 the task’s notification value. 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.

`ulTaskNotifyTake()` 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 `ulTaskNotifyTake()`.

When a task is using its notification value as a binary or counting semaphore other tasks should send notifications to it using the `xTaskNotifyGive()` macro, or `xTaskNotify()` function with the `eAction` parameter set to `eIncrement`.

`ulTaskNotifyTake()` can either clear the task’s notification value to zero on exit, in which case the notification value acts like a binary semaphore, or decrement the task’s notification value on exit, in which case the notification value acts like a counting semaphore.

A task can use `ulTaskNotifyTake()` to [optionally] block to wait for a 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 `xTaskNotifyWait()` will return when a notification is pending, `ulTaskNotifyTake()` will return when the task’s notification value is not zero.

See <http://www.FreeRTOS.org/RTOS-task-notifications.html> for details.

**Return** The task’s notification count before it is either cleared to zero or decremented (see the `xClearCountOnExit` parameter).

#### Parameters

- `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 `ulTaskNotifyTake()` 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_TICSK( value_in_ms )` can be used to convert a time specified in milliseconds to a time specified in ticks.

## Structures

### **struct xTASK\_STATUS**

Used with the `uxTaskGetSystemState()` function to return the state of each task in the system.

### Public Members

#### *TaskHandle\_t* **xHandle**

The handle of the task to which the rest of the information in the structure relates.

#### **const char \*pcTaskName**

A pointer to the task's name. This value will be invalid if the task was deleted since the structure was populated!

#### UBaseType\_t **xTaskNumber**

A number unique to the task.

#### *eTaskState* **eCurrentState**

The state in which the task existed when the structure was populated.

#### UBaseType\_t **uxCurrentPriority**

The priority at which the task was running (may be inherited) when the structure was populated.

#### UBaseType\_t **uxBasePriority**

The priority to which the task will return if the task's current priority has been inherited to avoid unbounded priority inversion when obtaining a mutex. Only valid if `configUSE_MUTEXES` is defined as 1 in `FreeRTOSConfig.h`.

#### uint32\_t **ulRunTimeCounter**

The total run time allocated to the task so far, as defined by the run time stats clock. See <http://www.freertos.org/rtos-run-time-stats.html>. Only valid when `configGENERATE_RUN_TIME_STATS` is defined as 1 in `FreeRTOSConfig.h`.

#### StackType\_t **\*pxStackBase**

Points to the lowest address of the task's stack area.

#### uint32\_t **usStackHighWaterMark**

The minimum amount of stack space that has remained for the task since the task was created. The closer this value is to zero the closer the task has come to overflowing its stack.

#### BaseType\_t **xCoreID**

Core this task is pinned to. This field is present if `CONFIG_FREERTOS_VTASKLIST_INCLUDE_COREID` is set.

### **struct xTASK\_SNAPSHOT**

Used with the `uxTaskGetSnapshotAll()` function to save memory snapshot of each task in the system. We need this struct because `TCB_t` is defined (hidden) in `tasks.c`.

### Public Members

#### void **\*pxTCB**

Address of task control block.

#### StackType\_t **\*pxTopOfStack**

Points to the location of the last item placed on the tasks stack.

#### StackType\_t **\*pxEndOfStack**

Points to the end of the stack. `pxTopOfStack < pxEndOfStack`, stack grows hi2lo `pxTopOfStack > pxEndOfStack`, stack grows lo2hi

**Macros****tskKERNEL\_VERSION\_NUMBER****tskKERNEL\_VERSION\_MAJOR****tskKERNEL\_VERSION\_MINOR****tskKERNEL\_VERSION\_BUILD****tskNO\_AFFINITY**Argument of `xTaskCreatePinnedToCore` indicating that task has no affinity.**tskIDLE\_PRIORITY**

Defines the priority used by the idle task. This must not be modified.

**taskYIELD ()**

task. h

Macro for forcing a context switch.

**taskENTER\_CRITICAL (mux)**

task. h

Macro to mark the start of a critical code region. Preemptive context switches cannot occur when in a critical region.

**Note** This may alter the stack (depending on the portable implementation) so must be used with care!**taskENTER\_CRITICAL\_ISR (mux)****taskEXIT\_CRITICAL (mux)**

task. h

Macro to mark the end of a critical code region. Preemptive context switches cannot occur when in a critical region.

**Note** This may alter the stack (depending on the portable implementation) so must be used with care!**taskEXIT\_CRITICAL\_ISR (mux)****taskDISABLE\_INTERRUPTS ()**

task. h

Macro to disable all maskable interrupts.

**taskENABLE\_INTERRUPTS ()**

task. h

Macro to enable microcontroller interrupts.

**taskSCHEDULER\_SUSPENDED****taskSCHEDULER\_NOT\_STARTED****taskSCHEDULER\_RUNNING****xTaskNotifyGive (xTaskToNotify)**

Simplified macro for sending task notification.

`configUSE_TASK_NOTIFICATIONS` must be undefined or defined as 1 for this macro to be available.When `configUSE_TASK_NOTIFICATIONS` is set to one each task has its own private “notification value”, which is a 32-bit unsigned integer (`uint32_t`).

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 the task’s notification value. 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.

`xTaskNotifyGive()` 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 `xTaskNotifyGive()`.

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 `ulTaskNotificationTake()` API function rather than the `xTaskNotifyWait()` API function.

See <http://www.FreeRTOS.org/RTOS-task-notifications.html> for more details.

**Return** `xTaskNotifyGive()` is a macro that calls `xTaskNotify()` with the `eAction` parameter set to `eIncrement` - so `pdPASS` is always returned.

#### Parameters

- `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()`.

### Type Definitions

```
typedef void *TaskHandle_t
task. h
```

Type by which tasks are referenced. For example, a call to `xTaskCreate` returns (via a pointer parameter) an `TaskHandle_t` variable that can then be used as a parameter to `vTaskDelete` to delete the task.

```
typedef BaseType_t (*TaskHookFunction_t) (void *)
```

Defines the prototype to which the application task hook function must conform.

```
typedef struct xTASK_STATUS TaskStatus_t
```

Used with the `uxTaskGetSystemState()` function to return the state of each task in the system.

```
typedef struct xTASK_SNAPSHOT TaskSnapshot_t
```

Used with the `uxTaskGetSnapshotAll()` function to save memory snapshot of each task in the system. We need this struct because `TCB_t` is defined (hidden) in `tasks.c`.

```
typedef void (*TlsDeleteCallbackFunction_t) (int, void *)
```

Prototype of local storage pointer deletion callback.

### Enumerations

```
enum eTaskState
```

Task states returned by `eTaskGetState`.

*Values:*

```
eRunning = 0
```

A task is querying the state of itself, so must be running.

```
eReady
```

The task being queried is in a read or pending ready list.

```
eBlocked
```

The task being queried is in the Blocked state.

```
eSuspended
```

The task being queried is in the Suspended state, or is in the Blocked state with an infinite time out.

```
eDeleted
```

The task being queried has been deleted, but its TCB has not yet been freed.

```
enum eNotifyAction
```

Actions that can be performed when `vTaskNotify()` is called.

*Values:*

```
eNoAction = 0
```

Notify the task without updating its notify value.



**eSetBits**

Set bits in the task's notification value.

**eIncrement**

Increment the task's notification value.

**eSetValueWithOverwrite**

Set the task's notification value to a specific value even if the previous value has not yet been read by the task.

**eSetValueWithoutOverwrite**

Set the task's notification value if the previous value has been read by the task.

**enum eSleepModeStatus**

Possible return values for eTaskConfirmSleepModeStatus().

*Values:*

**eAbortSleep = 0**

A task has been made ready or a context switch pended since portSUPPORESS\_TICKS\_AND\_SLEEP() was called - abort entering a sleep mode.

**eStandardSleep**

Enter a sleep mode that will not last any longer than the expected idle time.

**eNoTasksWaitingTimeout**

No tasks are waiting for a timeout so it is safe to enter a sleep mode that can only be exited by an external interrupt.

**Queue API****Header File**

- [freertos/include/freertos/queue.h](#)

**Functions**

BaseType\_t **xQueueGenericSendFromISR** (*QueueHandle\_t xQueue*, **const void \*const pvItemToQueue**, BaseType\_t \***const pxHigherPriorityTaskWoken**, **const BaseType\_t xCopyPosition**)

It is preferred that the macros xQueueSendFromISR(), xQueueSendToFrontFromISR() and xQueueSendToBackFromISR() be used in place of calling this function directly. xQueueGiveFromISR() 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):

```
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 );
```

(下页继续)

```

        // 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();
    }
}

```

**Return** pdTRUE if the data was successfully sent to the queue, otherwise errQUEUE\_FULL.

#### Parameters

- **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.
- **[out] pxHigherPriorityTaskWoken**: **xQueueGenericSendFromISR()** 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 **xQueueGenericSendFromISR()** 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).

BaseType\_t **xQueueGiveFromISR** (*QueueHandle\_t* xQueue, BaseType\_t \*const pxHigherPriorityTaskWoken)

BaseType\_t **xQueueIsQueueEmptyFromISR** (const *QueueHandle\_t* xQueue)

Utilities to query queues that are safe to use from an ISR. These utilities should be used only from within an ISR, or within a critical section.

BaseType\_t **xQueueIsQueueFullFromISR** (const *QueueHandle\_t* xQueue)

UBaseType\_t **uxQueueMessagesWaitingFromISR** (const *QueueHandle\_t* xQueue)

BaseType\_t **xQueueGenericSend** (*QueueHandle\_t* xQueue, const void \*const 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:

```

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.
}

```

**Return** pdTRUE if the item was successfully posted, otherwise errQUEUE\_FULL.

#### Parameters

- **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).

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()**.

**Return** pdTRUE if an item was successfully received from the queue, otherwise pdFALSE.

#### Parameters

- **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.

BaseType\_t **xQueueGenericReceive** (*QueueHandle\_t* xQueue, void \***const** pvBuffer, TickType\_t xTicksToWait, **const** BaseType\_t xJustPeek)

It is preferred that the macro `xQueueReceive()` be used rather than calling this function directly.

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.

This function must not be used in an interrupt service routine. See `xQueueReceiveFromISR` for an alternative that can.

Example usage:

```

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 receive from the queue.
void vADifferentTask( void *pvParameters )
{
    struct AMessage *pxRxdMessage;

    if( xQueue != 0 )
    {
        // Receive a message on the created queue. Block for 10 ticks if a
        // message is not immediately available.
        if( xQueueGenericReceive( xQueue, &( pxRxdMessage ), ( TickType_t ) 10_
↪ ) )
        {
            // pxRxdMessage now points to the struct AMessage variable posted
            // by vATask.
        }
    }

    // ... Rest of task code.
}

```

**Return** pdTRUE if an item was successfully received from the queue, otherwise pdFALSE.

**Parameters**

- `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. `xQueueGenericReceive()` will return immediately if the queue is empty and `xTicksToWait` is 0.
- `xJustPeek`: When set to true, the item received from the queue is not actually removed from the queue - meaning a subsequent call to `xQueueReceive()` will return the same item. When set to false, the item being received from the queue is also removed from the queue.

UBaseType\_t **uxQueueMessagesWaiting** (const *QueueHandle\_t* xQueue)

Return the number of messages stored in a queue.

**Return** The number of messages available in the queue.

**Parameters**

- `xQueue`: A handle to the queue being queried.

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.

**Return** The number of spaces available in the queue.

**Parameters**

- `xQueue`: A handle to the queue being queried.

void **vQueueDelete** (*QueueHandle\_t* xQueue)

Delete a queue - freeing all the memory allocated for storing of items placed on the queue.

**Parameters**

- `xQueue`: A handle to the queue to be deleted.

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:

```
QueueHandle_t xQueue;

// Function to create a queue and post some values.
void vAFunction( void *pvParameters )
{
    char cValueToPost;
    const TickType_t xTicksToWait = ( TickType_t )0xffff;

    // 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.
```

(下页继续)

```

    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 cRxdChar;

    while( xQueueReceiveFromISR( xQueue, ( void * ) &cRxdChar, &
    ↪xTaskWokenByReceive) )
    {
        // A character was received. Output the character now.
        vOutputCharacter( cRxdChar );

        // 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 ();
    }
}

```

**Return** pdTRUE if an item was successfully received from the queue, otherwise pdFALSE.

#### Parameters

- 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.
- [out] pxHigherPriorityTaskWoken: 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.

void **vQueueAddToRegistry** (*QueueHandle\_t* xQueue, const char \*pcName)

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.

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.

#### Parameters

- 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.
- pcName: 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.

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.

#### Parameters

- `xQueue`: The handle of the queue being removed from the registry.

**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.

**Note** This function has been back ported from FreeRTOS v9.0.0

**Return** 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.

**Parameters**

- `xQueue`: The handle of the queue the name of which will be returned.

*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.

*QueueHandle\_t* **xQueueGenericCreateStatic** (**const** *UBaseType\_t* *uxQueueLength*, **const** *UBaseType\_t* *uxItemSize*, *uint8\_t* \**puvQueueStorage*, *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 <http://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.

**Return** If the queue set is created successfully then a handle to the created queue set is returned. Otherwise NULL is returned.

**Parameters**

- `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.

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.

**Return** 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.

#### Parameters

- `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.

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.

**Return** 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.

#### Parameters

- `xQueueOrSemaphore`: The handle of the queue or semaphore being removed from the queue set (cast to an `QueueSetMemberHandle_t` type).
- `xQueueSet`: The handle of the queue set in which the queue or semaphore is included.

*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 <http://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.

**Return** `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 the specified block time expires.

#### Parameters

- `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.



**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. This allocates the storage required by the new queue and returns a handle for the queue.

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.
}

```

**Return** 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.

**Parameters**

- **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.

**xQueueCreateStatic (uxQueueLength, uxItemSize, 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 <http://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.

<http://www.FreeRTOS.org/Embedded-RTOS-Queues.html>

Example usage:

```

struct AMessage
{

```

(下页继续)

```

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 xQueue1;

    // Create a queue capable of containing 10 uint32_t values.
    xQueue1 = xQueueCreate( QUEUE_LENGTH, // The number of items the queue can
    →hold.
                            ITEM_SIZE // The size of each item in the queue
    →hold the items in the queue.
                            &( ucQueueStorage[ 0 ] ), // The buffer that will
    →queue structure.
                            &xQueueBuffer ); // The buffer that will hold the
    →queue structure.

    // The queue is guaranteed to be created successfully as no dynamic memory
    // allocation is used. Therefore xQueue1 is now a handle to a valid queue.

    // ... Rest of task code.
}

```

**Return** If the queue is created then a handle to the created queue is returned. If pxQueueBuffer is NULL then NULL is returned.

#### Parameters

- 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 \* uxItemSize ) 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.

#### **xQueueSendToFront** (xQueue, pvItemToQueue, xTicksToWait)

This is a macro that calls xQueueGenericSend().

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:

```

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.
}

```

**Return** `pdTRUE` if the item was successfully posted, otherwise `errQUEUE_FULL`.

#### Parameters

- `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.

**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:

```

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( 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.
}

```

**Return** `pdTRUE` if the item was successfully posted, otherwise `errQUEUE_FULL`.

#### Parameters

- `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.

#### **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 equivalent 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.
}

```

**Return** pdTRUE if the item was successfully posted, otherwise errQUEUE\_FULL.

#### Parameters

- **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.

#### **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:

```

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!
    }

    // ...
}

```

**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.

**Parameters**

- `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.

**xQueuePeek** (`xQueue`, `pvBuffer`, `xTicksToWait`)

This is a macro that calls the `xQueueGenericReceive()` function.

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:

```

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 *pRxedMessage;

if( xQueue != 0 )
{
// Peek a message on the created queue. Block for 10 ticks if a
// message is not immediately available.
if( xQueuePeek( xQueue, &( pRxedMessage ), ( TickType_t ) 10 ) )
{
// pRxedMessage now points to the struct AMessage variable posted
// by vATask, but the item still remains on the queue.
}
}

// ... Rest of task code.
}

```

**Return** `pdTRUE` if an item was successfully received from the queue, otherwise `pdFALSE`.

#### Parameters

- `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.

#### **xQueueReceive** (xQueue, pvBuffer, xTicksToWait)

queue. h

This is a macro that calls the `xQueueGenericReceive()` function.

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:

```

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 receive from the queue.
void vADifferentTask( void *pvParameters )
{
    struct AMessage *pRxedMessage;

    if( xQueue != 0 )
    {
        // Receive a message on the created queue. Block for 10 ticks if a
        // message is not immediately available.
        if( xQueueReceive( xQueue, &( pRxedMessage ), ( TickType_t ) 10 ) )
        {
            // pRxedMessage now points to the struct AMessage variable posted
            // by vATask.
        }
    }
}

```

(下页继续)



```

}

// ... Rest of task code.
}

```

**Return** pdTRUE if an item was successfully received from the queue, otherwise pdFALSE.

**Parameters**

- **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.

**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):

```

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 ();
    }
}

```

**Return** pdTRUE if the data was successfully sent to the queue, otherwise errQUEUE\_FULL.

**Parameters**

- **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.
- [out] **pxHigherPriorityTaskWoken**: xQueueSendToFrontFromISR() 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 xQueueSendToFromFromISR() sets this value to pdTRUE then a context switch should be requested before the interrupt is exited.

**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):

```
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 ();
}
}
```

**Return** `pdTRUE` if the data was successfully sent to the queue, otherwise `errQUEUE_FULL`.

#### Parameters

- `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.
- [out] `pxHigherPriorityTaskWoken`: `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.

#### **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:

```
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.
```

(下页继续)

```

    xQueue = xQueueCreate( 1, sizeof( uint32_t ) );
}

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
→context
// switch so this interrupt returns directly to the unblocked task.
portYIELD_FROM_ISR(); // or portEND_SWITCHING_ISR() depending on the
→port.
}
}
}

```

**Return** `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.

#### Parameters

- `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.
- `[out] pxHigherPriorityTaskWoken`: `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.

#### **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 `xQueueSendToFrontFromISR()` 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):

```

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 ();
}
}

```

**Return** pdTRUE if the data was successfully sent to the queue, otherwise errQUEUE\_FULL.

#### Parameters

- 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.
- [out] pxHigherPriorityTaskWoken: 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.

#### xQueueReset (xQueue)

Reset a queue back to its original empty state. pdPASS is returned if the queue is successfully reset. pdFAIL is returned if the queue could not be reset because there are tasks blocked on the queue waiting to either receive from the queue or send to the queue.

**Return** always returns pdPASS

#### Parameters

- xQueue: The queue to reset

### Type Definitions

**typedef** void \*QueueHandle\_t

Type by which queues are referenced. For example, a call to xQueueCreate() returns an QueueHandle\_t variable that can then be used as a parameter to xQueueSend(), xQueueReceive(), etc.

**typedef** void \*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** void \*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

- [freertos/include/freertos/semphr.h](#)

### Macros

**semBINARY\_SEMAPHORE\_QUEUE\_LENGTH**

**semSEMAPHORE\_QUEUE\_ITEM\_LENGTH**

**semGIVE\_BLOCK\_TIME**

### **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! <http://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 <http://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’.

Function that creates a semaphore by using the existing queue mechanism. The queue length is 1 as this is a binary semaphore. The data size is 0 as nothing is actually stored - all that is important is whether the queue is empty or full (the binary semaphore is available or not).

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:

```
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.
    xSemaphore = xSemaphoreCreateBinary();

    if( xSemaphore != NULL )
    {
        // The semaphore was created successfully.
        // The semaphore can now be used.
    }
}
```

**Return** Handle to the created semaphore.

### **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! <http://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 <http://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:

```
SemaphoreHandle_t xSemaphore = NULL;
StaticSemaphore_t xSemaphoreBuffer;

void vATask( void * pvParameters )
{
    // Semaphore cannot be used before a call to xSemaphoreCreateBinary().
    // 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 = xSemaphoreCreateBinary( &xSemaphoreBuffer );

    // Rest of task code goes here.
}
```

**Return** If the semaphore is created then a handle to the created semaphore is returned. If `pxSemaphoreBuffer` is NULL then NULL is returned.

#### Parameters

- `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.

#### **xSemaphoreTake** (xSemaphore, xBlockTime)

*Macro* to obtain a semaphore. The semaphore must have previously been created with a call to `vSemaphoreCreateBinary()`, `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.
    vSemaphoreCreateBinary( xSemaphore );
}

// A task that uses the semaphore.
void vAnotherTask( void * pvParameters )
{
    // ... Do other things.

    if( xSemaphore != NULL )
    {
```

(下页继续)

```

// 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.
}
}
}

```

**Return** pdTRUE if the semaphore was obtained. pdFALSE if xBlockTime expired without the semaphore becoming available.

#### Parameters

- 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).

#### **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:

```

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 )
    {

```

(下页继续)

```

// 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.
}
}
}

```

**Return** pdTRUE if the semaphore was obtained. pdFALSE if xBlockTime expired without the semaphore becoming available.

#### Parameters

- xMutex: A handle to the mutex being obtained. This is the handle returned by xSemaphoreCreateRecursiveMutex();
- 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 xBlockTime.

#### **xSemaphoreGive** (xSemaphore)

*Macro* to release a semaphore. The semaphore must have previously been created with a call to vSemaphoreCreateBinary(), xSemaphoreCreateMutex() or xSemaphoreCreateCounting(). and obtained using sSemaphoreTake().

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 )
{

```

(下页继续)



```

// Create the semaphore to guard a shared resource.
vSemaphoreCreateBinary( xSemaphore );

if( xSemaphore != NULL )
{
    if( xSemaphoreGive( xSemaphore ) != pdTRUE )
    {
        // 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.
        }
    }
}
}
}

```

**Return** 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.

#### Parameters

- xSemaphore: A handle to the semaphore being released. This is the handle returned when the semaphore was created.

#### **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.
        }
    }
}

```

**Return** pdTRUE if the semaphore was given.

#### Parameters

- **xMutex**: A handle to the mutex being released, or 'given'. This is the handle returned by xSemaphoreCreateMutex();

#### **xSemaphoreGiveFromISR** (xSemaphore, pxHigherPriorityTaskWoken)

*Macro* to release a semaphore. The semaphore must have previously been created with a call to vSemaphoreCreateBinary() 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.
    }
}
```

**Return** pdTRUE if the semaphore was successfully given, otherwise errQUEUE\_FULL.

#### Parameters

- xSemaphore: A handle to the semaphore being released. This is the handle returned when the semaphore was created.
- [out] 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.

**xSemaphoreTakeFromISR** (xSemaphore, pxHigherPriorityTaskWoken)

*Macro* to take a semaphore from an ISR. The semaphore must have previously been created with a call to vSemaphoreCreateBinary() 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).

**Return** pdTRUE if the semaphore was successfully taken, otherwise pdFALSE

**Parameters**

- xSemaphore: A handle to the semaphore being taken. This is the handle returned when the semaphore was created.
- [out] pxHigherPriorityTaskWoken: xSemaphoreTakeFromISR() will set \*pxHigherPriorityTaskWoken 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.

**xSemaphoreCreateMutex** ()

*Macro* that implements a mutex semaphore by using the existing queue mechanism.

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 <http://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 vSemaphoreCreateBinary() 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:

```
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.
    }
}
```

**Return** 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 <http://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:

```
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.
}
```

**Return** If the mutex was successfully created then a handle to the created mutex is returned. If `pxMutexBuffer` was NULL then NULL is returned.

#### Parameters

- `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.

#### **xSemaphoreCreateRecursiveMutex()**

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 <http://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 `vSemaphoreCreateBinary()` 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:

```
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.
    }
}
```

**Return** xSemaphore Handle to the created mutex semaphore. Should be of type `SemaphoreHandle_t`.

#### **xSemaphoreCreateRecursiveMutexStatic** (pxStaticSemaphore)

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 <http://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:

```
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.
}
```

(下页继续)

(续上页)

```

xSemaphore = xSemaphoreCreateRecursiveMutexStatic( &xMutexBuffer );

// As no dynamic memory allocation was performed, xSemaphore cannot be NULL,
// so there is no need to check it.
}

```

**Return** 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.

**Parameters**

- 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.

**xSemaphoreCreateCounting** (uxMaxCount, uxInitialCount)

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! <http://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 <http://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.
    }
}

```

(下页继续)

```

    // The semaphore can now be used.
}
}

```

**Return** Handle to the created semaphore. Null if the semaphore could not be created.

**Parameters**

- `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.

**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! <http://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 <http://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:

```

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.
}

```



**Return** 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.

**Parameters**

- `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.

**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.

**Parameters**

- `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 determining the identity of the mutex holder as the holder may change between the function exiting and the returned value being tested.

**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

- [freertos/include/freertos/timers.h](#)

## Functions

*TimerHandle\_t* **xTimerCreate** (`const` char \*`const` *pcTimerName*, `const` TickType\_t *xTimerPeriodInTicks*, `const` UBaseType\_t *uxAutoReload*, void \*`const` *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 <http://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:

```

#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,
        ↪not used by the kernel.
        ( 100 * x ), // The timer period in
        ↪ticks.
        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
        {

```

(下页继续)

```

// 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( ;; );
}

```

**Return** If the timer is successfully created then a handle to the newly created timer is returned. If the timer cannot be created (because either there is insufficient FreeRTOS heap remaining to allocate the timer structures, or the timer period was set to 0) then NULL is returned.

#### Parameters

- **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  $(500 / \text{portTICK\_PERIOD\_MS})$  provided `configTICK_RATE_HZ` is less than or equal to 1000.
- **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 );” .

**TimerHandle\_t xTimerCreateStatic**(const char \*const pcTimerName, const TickType\_t xTimerPeriodInTicks, const UBaseType\_t uxAutoReload, void \*const pvTimerID, TimerCallbackFunction\_t pxCallbackFunction, StaticTimer\_t \*pxTimerBuffer)

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 <http://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:

```
// The buffer used to hold the software timer's data structure.
static StaticTimer_t xTimerBuffer;

// A variable that will be incremented by the software timer's callback
// function.
UBaseType_t uxVariableToIncrement = 0;

// A software timer callback function that increments a variable passed to
// it when the software timer was created. After the 5th increment the
// callback function stops the software timer.
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 )++;

    // If this callback has executed the required number of times, stop the
    // timer.
    if( *puxVariableToIncrement == 5 )
    {
        // This is called from a timer callback so must not block.
        xTimerStop( xExpiredTimer, staticDONT_BLOCK );
    }
}

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", // Text name for the task.
↳Helps debugging only. Not used by FreeRTOS.
                                xTimerPeriod, // The period of the timer
↳in ticks.
                                pdTRUE, // This is an auto-reload
↳timer.
                                ( void * ) &uxVariableToIncrement, // A
↳variable incremented by the software timer's callback function
                                prvTimerCallback, // The function to execute
↳when the timer expires.
                                &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.
    // ...
}
```

(下页继续)

```

    // Starting the scheduler will start the timers running as they have
    ↪already
    // been set into the active state.
    vTaskStartScheduler();

    // Should not reach here.
    for( ;; );
}

```

**Return** If the timer is created then a handle to the created timer is returned. If pxTimerBuffer was NULL then NULL is returned.

#### Parameters

- 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.
- 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.

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.

If the same callback function is assigned to multiple timers then the timer ID can be used within the callback function to identify which timer actually expired.

Example usage:

**Return** The ID assigned to the timer being queried.

#### Parameters

- xTimer: The timer being queried.

See the xTimerCreate() API function example usage scenario.

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:

#### Parameters

- xTimer: The timer being updated.

- `pvNewID`: The ID to assign to the timer.

See the `xTimerCreate()` API function example usage scenario.

`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:

```
// 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.
    }
}
```

**Return** `pdFALSE` will be returned if the timer is dormant. A value other than `pdFALSE` will be returned if the timer is active.

#### Parameters

- `xTimer`: The timer being queried.

`TaskHandle_t xTimerGetTimerDaemonTaskHandle (void)`

`xTimerGetTimerDaemonTaskHandle()` is only available if `INCLUDE_xTimerGetTimerDaemonTaskHandle` is set to 1 in `FreeRTOSConfig.h`.

Simply returns the handle of the timer service/daemon task. It is not valid to call `xTimerGetTimerDaemonTaskHandle()` before the scheduler has been started.

`TickType_t xTimerGetPeriod (TimerHandle_t xTimer)`

Returns the period of a timer.

**Return** The period of the timer in ticks.

#### Parameters

- `xTimer`: The handle of the timer being queried.

`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.

**Return** 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.

#### Parameters

- `xTimer`: The handle of the timer being queried.

`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:

```
// 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
    // 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 );
}
```

**Return** `pdPASS` is returned if the message was successfully sent to the timer daemon task, otherwise `pdFALSE` is returned.

#### Parameters

- `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.

```
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’ ).

**Return** `pdPASS` is returned if the message was successfully sent to the timer daemon task, otherwise `pdFALSE` is returned.

#### Parameters

- `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.

```
const char *pcTimerGetTimerName(TimerHandle_t xTimer)
```

Returns the name that was assigned to a timer when the timer was created.

**Return** The name assigned to the timer specified by the `xTimer` parameter.

#### Parameters

- `xTimer`: The handle of the timer being queried.

### 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)
```

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.

`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:

**Return** `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.

#### Parameters

- `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.

See the `xTimerCreate()` API function example usage scenario.

#### **xTimerStop** (`xTimer`, `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 `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:

**Return** `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.

#### Parameters

- `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.

See the `xTimerCreate()` API function example usage scenario.

#### **xTimerChangePeriod** (`xTimer`, `xNewPeriod`, `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:

```
// This function assumes xTimer has already been created.  If the timer
// referenced by xTimer is already active when it is called, then the timer
// is deleted.  If the timer referenced by xTimer is not active when it is
// called, then the period of the timer is set to 500ms and the timer is
// started.
void vAFFunction( TimerHandle_t xTimer )
{
    if( xTimerIsTimerActive( xTimer ) != pdFALSE ) // or more simply and
    equivalently "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.
        }
    }
}
```

**Return** 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.

#### Parameters

- 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.

#### xTimerDelete( xTimer, 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:

**Return** 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.

#### Parameters

- 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.

See the xTimerChangePeriod() API function example usage scenario.

#### xTimerReset (xTimer, 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:

```
// 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 );
}
```

(下页继续)

```

    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.
    xTaskStartScheduler();

    // Should not reach here.
    for( ;; );
}

```

**Return** 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/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.

#### Parameters

- `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.

**xTimerStartFromISR** (`xTimer`, `pxHigherPriorityTaskWoken`)

A version of `xTimerStart()` that can be called from an interrupt service routine.

Example usage:

```
// 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 );
}

// The key press interrupt service routine.
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.
    }

    // Perform the rest of the key processing 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).
    }
}
```

**Return** `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.

#### Parameters

- `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.

#### `xTimerStopFromISR` (`xTimer`, `pxHigherPriorityTaskWoken`)

A version of `xTimerStop()` that can be called from an interrupt service routine.

Example usage:

```
// 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).
    }
}
```

**Return** `pdFAIL` will be returned if the stop 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. The timer service/daemon task priority is set by the `configTIMER_TASK_PRIORITY` configuration constant.

#### Parameters

- `xTimer`: The handle of the timer being stopped.
- `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 `xTimerStopFromISR()` 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 `xTimerStopFromISR()` 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 `xTimerStopFromISR()` function. If `xTimerStopFromISR()` sets this value to `pdTRUE` then a context switch should be performed before the interrupt exits.

#### **xTimerChangePeriodFromISR** (xTimer, xNewPeriod, pxHigherPriorityTaskWoken)

A version of `xTimerChangePeriod()` that can be called from an interrupt service routine.

Example usage:

```
// 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).
    }
}
```

**Return** `pdFAIL` will be returned if the command to change the timers period 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. The timer service/daemon task priority is set by the config-`TIMER_TASK_PRIORITY` configuration constant.

#### **Parameters**

- `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  $( 500 / \text{portTICK\_PERIOD\_MS} )$  provided `configTICK_RATE_HZ` is less than or equal to 1000.
- `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 `xTimerChangePeriodFromISR()` 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 `xTimerChangePeriodFromISR()` 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 `xTimerChangePeriodFromISR()` function. If `xTimerChangePeriodFromISR()` sets this value to `pdTRUE` then a context switch should be performed before the interrupt exits.

#### **xTimerResetFromISR** (xTimer, pxHigherPriorityTaskWoken)

A version of `xTimerReset()` that can be called from an interrupt service routine.

Example usage:

```
// 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 );
}

// The key press interrupt service routine.
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 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).
    }
}
```

**Return** `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.

#### 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.

#### Type Definitions

**typedef** void \***TimerHandle\_t**

Type by which software timers are referenced. For example, a call to `xTimerCreate()` returns an `TimerHandle_t` variable that can then be used to reference the subject timer in calls to other software timer API functions (for example, `xTimerStart()`, `xTimerReset()`, etc.).

**typedef** void (\***TimerCallbackFunction\_t**) (*TimerHandle\_t* xTimer)

Defines the prototype to which timer callback functions must conform.

**typedef** void (\***PendedFunction\_t**) (void \*, uint32\_t)

Defines the prototype to which functions used with the `xTimerPendFunctionCallFromISR()` function must conform.

#### Event Group API

##### Header File

- [freertos/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 groups is created using `xEventGroupCreate()` then the required memory is automatically dynamically allocated inside the `xEventGroupCreate()` function. (see <http://www.freertos.org/a00111.html>). If an event group is created using `xEventGropuCreateStatic()` 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:

```
// 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?
```

(下页继续)

```

if( xCreatedEventGroup == NULL )
{
    // The event group was not created because there was insufficient
    // FreeRTOS heap available.
}
else
{
    // The event group was created.
}

```

**Return** 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 <http://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 `xEventGropuCreate()` then the required memory is automatically dynamically allocated inside the `xEventGroupCreate()` function. (see <http://www.freertos.org/a00111.html>). If an event group is created using `xEventGropuCreateStatic()` 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:

```

// 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.
xEventGroup = xEventGroupCreateStatic( &xEventGroupBuffer );

```

**Return** If the event group was created then a handle to the event group is returned. If `pxEventGroupBuffer` was NULL then NULL is returned.

#### Parameters

- `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.

*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:

```

#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.
    }
}

```

{c}

**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 `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`.

#### 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()`.
- `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.

**`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:

```
#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.
    }
}
```

**Return** The value of the event group before the specified bits were cleared.

#### 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 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`.

**`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:

```
#define BIT_0    ( 1 << 0 )
#define BIT_4    ( 1 << 4 )

void aFunction( EventGroupHandle_t xEventGroup )
```

(下页继续)

```

{
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.
}
}
}

```

{c}

**Return** 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 `xClearBitOnExit` 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.

#### Parameters

- `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`.

*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:

```

// 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

```

(下页继续)

(续上页)

```

// 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.
}
}

```

**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.

#### 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.

#### *EventBits\_t* `xEventGroupGetBitsFromISR` (*EventGroupHandle\_t* `xEventGroup`)

A version of `xEventGroupGetBits()` that can be called from an ISR.

**Return** The event group bits at the time `xEventGroupGetBitsFromISR()` was called.

#### Parameters

- `xEventGroup`: The event group being queried.

#### `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.

#### Parameters

- `xEventGroup`: The event group being deleted.

### Macros

#### `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:

```

#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().

```

(下页继续)

```

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.
    }
}

```

**Return** 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.

#### 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.

**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 xEventGroupSetBitFromISR() 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:

```

#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 );

    // Was the message posted successfully?
    if( xResult == pdPASS )
    {
        // 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 );
    }
}

```



```
}

```

**Return** 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.

**Parameters**

- `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.

**xEventGroupGetBits** (`xEventGroup`)

Returns the current value of the bits in an event group. This function cannot be used from an interrupt.

**Return** The event group bits at the time `xEventGroupGetBits()` was called.

**Parameters**

- `xEventGroup`: The event group being queried.

## Type Definitions

```
typedef void *EventGroupHandle_t
```

An event group is a collection of bits to which an application can assign a meaning. For example, an application may create an event group to convey the status of various CAN bus related events in which bit 0 might mean “A CAN

message has been received and is ready for processing”, bit 1 might mean “The application has queued a message that is ready for sending onto the CAN network”, and bit 2 might mean “It is time to send a SYNC message onto the CAN network” etc. A task can then test the bit values to see which events are active, and optionally enter the Blocked state to wait for a specified bit or a group of specified bits to be active. To continue the CAN bus example, a CAN controlling task can enter the Blocked state (and therefore not consume any processing time) until either bit 0, bit 1 or bit 2 are active, at which time the bit that was actually active would inform the task which action it had to take (process a received message, send a message, or send a SYNC).

The event groups implementation contains intelligence to avoid race conditions that would otherwise occur were an application to use a simple variable for the same purpose. This is particularly important with respect to when a bit within an event group is to be cleared, and when bits have to be set and then tested atomically - as is the case where event groups are used to create a synchronisation point between multiple tasks (a ‘rendezvous’). `event_groups.h`

Type by which event groups are referenced. For example, a call to `xEventGroupCreate()` returns an `EventGroupHandle_t` variable that can then be used as a parameter to other event group functions.

```
typedef TickType_t EventBits_t
```

## 2.7.2 FreeRTOS Additions

### Overview

ESP-IDF FreeRTOS is based on the Xtensa port of FreeRTOS v8.2.0 with significant modifications for SMP compatibility (see [ESP-IDF FreeRTOS SMP Changes](#)). However various features specific to ESP-IDF FreeRTOS have been added. The features are as follows:

**Ring Buffers:** Ring buffers were added to provide a form of buffer that could accept entries of arbitrary lengths.

**Hooks:** ESP-IDF FreeRTOS hooks provides support for registering extra Idle and Tick hooks at run time. Moreover, the hooks can be asymmetric amongst both CPUs.

## 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. You may apply for a piece of memory on the ring buffer to send an item, or just use the API to copy your data and send (according to the send API you call). For efficiency reasons, **items are always retrieved from the ring buffer by reference**. As a result, all retrieved items *must also be returned* in order for them to be removed from the ring buffer completely. The ring buffers are split into the three following types:

**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. *Only this buffer type allows you getting the data item address and writing to the item by yourself.*

**Allow-Split** buffers will allow an item to be split when wrapping around if doing so will allow the item to be stored. Allow-split buffers are more memory efficient than no-split buffers but can return an item in two parts when retrieving.

**Byte buffers** do not store data as separate items. All data is stored as a sequence of bytes, and any number of bytes and be sent or retrieved each time. Use byte buffers when separate items do not need to be maintained (e.g. a byte stream).

---

**注解:** No-split/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.

---



---

**注解:** Each item stored in no-split/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/allow-split buffers will also be rounded up when created.

---

**Usage** The following example demonstrates the usage of `xRingbufferCreate()` and `xRingbufferSend()` to create a ring buffer 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 `xRingbufferSendAcquire()` and `xRingbufferSendComplete()` instead of `xRingbufferSend()` to apply for the memory on the ring buffer (of type `RINGBUF_TYPE_NOSPLIT`) and then send an item to it. This way 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 {
    lldesc_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_
↪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()`

```

...

//Receive an item from allow-split ring buffer
size_t item_size1, item_size2;
char *item1, *item2;
BaseType_t ret = xRingbufferReceiveSplit(buf_handle, (void **)&item1, (void_
↳**)&item2, &item_size1, &item_size2, pdMS_TO_TICKS(1000));

//Check received item
if (ret == pdTRUE && item1 != NULL) {
    for (int i = 0; i < item_size1; i++) {
        printf("%c", item1[i]);
    }
    vRingbufferReturnItem(buf_handle, (void *)item1);
    //Check if item was split
    if (item2 != NULL) {
        for (int i = 0; i < item_size2; i++) {
            printf("%c", item2[i]);
        }
        vRingbufferReturnItem(buf_handle, (void *)item2);
    }
    printf("\n");
} else {
    //Failed to receive item
    printf("Failed to receive item\n");
}

```

The following example demonstrates retrieving and returning an item from a **byte buffer** using *xRingbufferReceiveUpTo()* and *vRingbufferReturnItem()*

```

...

//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");
    //Return Item
    vRingbufferReturnItem(buf_handle, (void *)item);
} else {
    //Failed to receive item
    printf("Failed to receive item\n");
}

```

For ISR safe versions of the functions used above, call *xRingbufferSendFromISR()*, *xRingbufferReceiveFromISR()*, *xRingbufferReceiveSplitFromISR()*, *xRingbufferReceiveUpToFromISR()*, and *vRingbufferReturnItemFromISR()*

**Sending to Ring Buffer** The following diagrams illustrate the differences between no-split/allow-split buffers and byte buffers with regards 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/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

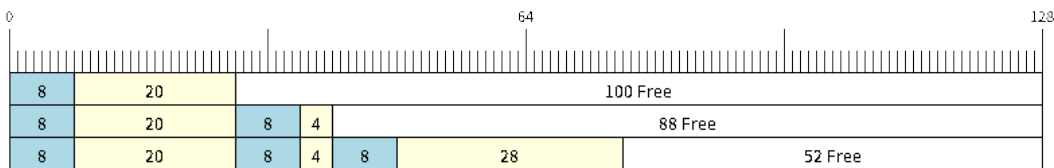


图 26: Sending items to no-split/allow-split ring buffers

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.

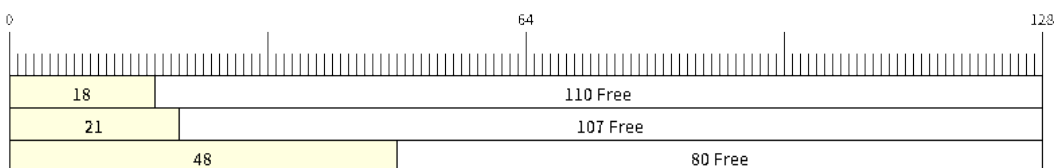


图 27: Sending items to byte buffers

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 the earliest acquired item will prevent the subsequent items from being received.

The following diagrams illustrate what will happen when SendAcquire/SendComplete don't happen in the same order. At the beginning, there is already an 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.

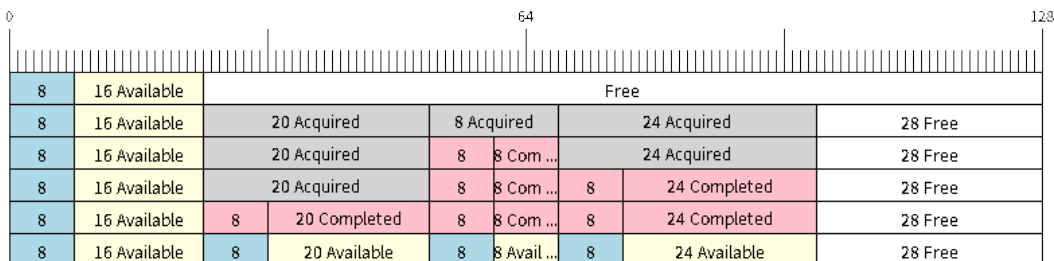


图 28: SendAcquire/SendComplete items in no-split ring buffers

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. Due to the usage if 20 bytes item is not complete, it's not available, nor the following data items.

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/byte buffers do not allow using SendAcquire/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 assumes a buffer of **128 bytes with 56 bytes of free space that wraps around** and a sent item of **28 bytes**.

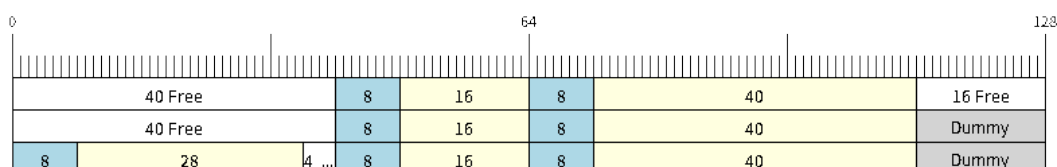


图 29: Wrap around in no-split buffers

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.

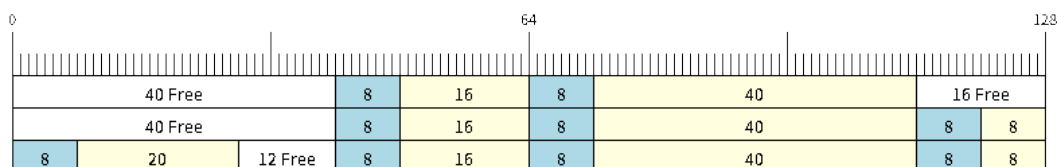


图 30: Wrap around in allow-split buffers

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.

---

**注解:** Allow-split buffers treats the 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.

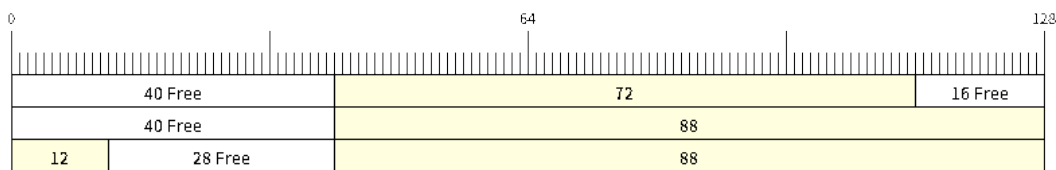


图 31: Wrap around in byte buffers

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 part continuous will be treated as a separate item by the byte buffer.

**Retrieving/Returning** The following diagrams illustrates the differences between no-split/allow-split and byte buffers in retrieving and returning data.

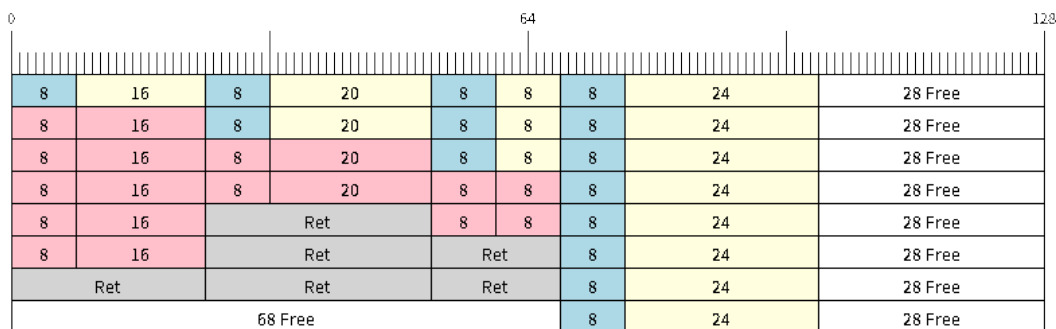


图 32: Retrieving/Returning items in no-split/allow-split ring buffers

Items in no-split/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 they were retrieved (20, 8, 16). As such, the space is not freed until the first item (16 byte) is returned.

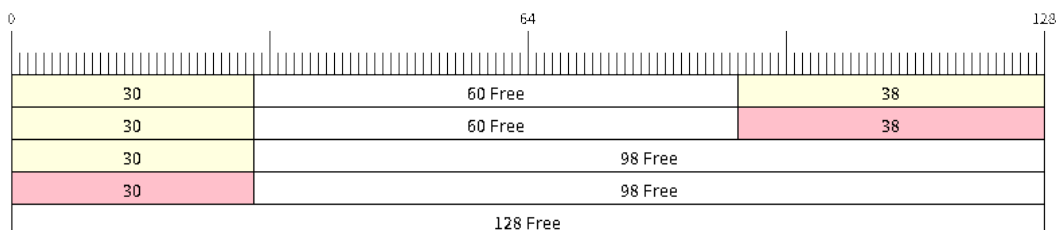


图 33: Retrieving/Returning data in byte buffers

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.

```
#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
xQueueSetMemberHandle 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`
- The ring buffer's storage area of size `xBufferSize`. Note that `xBufferSize` must be 32-bit aligned for no-split/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).



---

**注解:** The `CONFIG_FREERTOS_SUPPORT_STATIC_ALLOCATION` option must be enabled in `menuconfig` for statically allocated ring buffers to be available.

---

**注解:** 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.

```
#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);
```

## Ring Buffer API Reference

---

**注解:** 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 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.

---

**Header File**

- `esp_ringbuf/include/freertos/ringbuf.h`

**Functions**

*RingbufHandle\_t* **xRingbufferCreate** (*size\_t* *xBufferSize*, *RingbufferType\_t* *xBufferType*)

Create a ring buffer.

**Note** *xBufferSize* of no-split/allow-split buffers will be rounded up to the nearest 32-bit aligned size.

**Return** A handle to the created ring buffer, or NULL in case of error.

**Parameters**

- [in] *xBufferSize*: Size of the buffer in bytes. Note that items require space for overhead in no-split/allow-split buffers
- [in] *xBufferType*: Type of ring buffer, see documentation.

*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.

**Return** A *RingbufHandle\_t* handle to the created ring buffer, or NULL in case of error.

**Parameters**

- [in] *xItemSize*: Size of each item to be put into the ring buffer
- [in] *xItemNum*: Maximum number of items the buffer needs to hold simultaneously

*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.

**Note** The `CONFIG_FREERTOS_SUPPORT_STATIC_ALLOCATION` option must be enabled for this to be available

**Note** *xBufferSize* of no-split/allow-split buffers MUST be 32-bit aligned.

**Return** A handle to the created ring buffer

**Parameters**

- [in] *xBufferSize*: Size of the buffer in bytes.
- [in] *xBufferType*: Type of ring buffer, see documentation
- [in] *pucRingbufferStorage*: Pointer to the ring buffer's storage area. Storage area must of the same size as specified by *xBufferSize*
- [in] *pxStaticRingbuffer*: Pointed to a struct of type *StaticRingbuffer\_t* which will be used to hold the ring buffer's data structure

*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.

**Note** 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.

**Return**

- `pdTRUE` if succeeded
- `pdFALSE` on time-out or when the data is larger than the maximum permissible size of the buffer

**Parameters**

- [in] *xRingbuffer*: Ring buffer to insert the item into
- [in] *pvItem*: Pointer to data to insert. NULL is allowed if *xItemSize* is 0.
- [in] *xItemSize*: Size of data to insert.
- [in] *xTicksToWait*: Ticks to wait for room in the ring buffer.

*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.

**Note** 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.

**Return**

- pdTRUE if succeeded
- pdFALSE when the ring buffer does not have space.

**Parameters**

- [in] xRingbuffer: Ring buffer to insert the item into
- [in] pvItem: Pointer to data to insert. NULL is allowed if xItemSize is 0.
- [in] xItemSize: Size of data to insert.
- [out] pxHigherPriorityTaskWoken: Value pointed to will be set to pdTRUE if the function woke up a higher priority task.

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 timesout.

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.

**Note** 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.

**Return**

- pdTRUE if succeeded
- pdFALSE on time-out or when the data is larger than the maximum permissible size of the buffer

**Parameters**

- [in] xRingbuffer: Ring buffer to allocate the memory
- [out] ppvItem: Double pointer to memory acquired (set to NULL if no memory were retrieved)
- [in] xItemSize: Size of item to acquire.
- [in] xTicksToWait: Ticks to wait for room in the ring buffer.

BaseType\_t **xRingbufferSendComplete** (*RingbufHandle\_t* xRingbuffer, void \*pvItem)

Actually send an item into the ring buffer allocated before by `xRingbufferSendAcquire`.

**Note** Only applicable for no-split ring buffers. Only call for items allocated by `xRingbufferSendAcquire`.

**Return**

- pdTRUE if succeeded
- pdFALSE if fail for some reason.

**Parameters**

- [in] xRingbuffer: Ring buffer to insert the item into
- [in] pvItem: Pointer to item in allocated memory to insert.

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.

**Note** A call to `vRingbufferReturnItem()` is required after this to free the item retrieved.

**Return**

- Pointer to the retrieved item on success; \*pxItemSize filled with the length of the item.
- NULL on timeout, \*pxItemSize is untouched in that case.

**Parameters**

- [in] xRingbuffer: Ring buffer to retrieve the item from
- [out] pxItemSize: Pointer to a variable to which the size of the retrieved item will be written.
- [in] xTicksToWait: Ticks to wait for items in the ring buffer.

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

**Note** A call to vRingbufferReturnItemFromISR() is required after this to free the item retrieved.

**Note** Byte buffers do not allow multiple retrievals before returning an item

**Return**

- 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.

**Parameters**

- [in] xRingbuffer: Ring buffer to retrieve the item from
- [out] pxItemSize: Pointer to a variable to which the size of the retrieved item will be written.

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.

**Note** Call(s) to vRingbufferReturnItem() is required after this to free up the item(s) retrieved.

**Note** This function should only be called on allow-split buffers

**Return**

- pdTRUE if an item (split or unsplit) was retrieved
- pdFALSE when no item was retrieved

**Parameters**

- [in] xRingbuffer: Ring buffer to retrieve the item from
- [out] ppvHeadItem: Double pointer to first part (set to NULL if no items were retrieved)
- [out] ppvTailItem: Double pointer to second part (set to NULL if item is not split)
- [out] pxHeadItemSize: Pointer to size of first part (unmodified if no items were retrieved)
- [out] pxTailItemSize: Pointer to size of second part (unmodified if item is not split)
- [in] xTicksToWait: Ticks to wait for items in the ring buffer.

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. 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

**Note** Calls to vRingbufferReturnItemFromISR() is required after this to free up the item(s) retrieved.

**Note** This function should only be called on allow-split buffers

**Return**

- pdTRUE if an item (split or unsplit) was retrieved
- pdFALSE when no item was retrieved

**Parameters**

- [in] xRingbuffer: Ring buffer to retrieve the item from
- [out] ppvHeadItem: Double pointer to first part (set to NULL if no items were retrieved)
- [out] ppvTailItem: Double pointer to second part (set to NULL if item is not split)
- [out] pxHeadItemSize: Pointer to size of first part (unmodified if no items were retrieved)
- [out] pxTailItemSize: 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.

**Note** A call to `vRingbufferReturnItem()` is required after this to free up the data retrieved.

**Note** This function should only be called on byte buffers

**Note** Byte buffers do not allow multiple retrievals before returning an item

**Return**

- Pointer to the retrieved item on success; *\*pxItemSize* filled with the length of the item.
- NULL on timeout, *\*pxItemSize* is untouched in that case.

**Parameters**

- [in] *xRingbuffer*: Ring buffer to retrieve the item from
- [out] *pxItemSize*: Pointer to a variable to which the size of the retrieved item will be written.
- [in] *xTicksToWait*: Ticks to wait for items in the ring buffer.
- [in] *xMaxSize*: Maximum number of bytes to return.

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.

**Note** A call to `vRingbufferReturnItemFromISR()` is required after this to free up the data received.

**Note** This function should only be called on byte buffers

**Note** Byte buffers do not allow multiple retrievals before returning an item

**Return**

- 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.

**Parameters**

- [in] *xRingbuffer*: Ring buffer to retrieve the item from
- [out] *pxItemSize*: Pointer to a variable to which the size of the retrieved item will be written.
- [in] *xMaxSize*: Maximum number of bytes to return.

void **vRingbufferReturnItem** (*RingbufHandle\_t xRingbuffer*, void *\*pvItem*)

Return a previously-retrieved item to the ring buffer.

**Note** If a split item is retrieved, both parts should be returned by calling this function twice

**Parameters**

- [in] *xRingbuffer*: Ring buffer the item was retrieved from
- [in] *pvItem*: Item that was received earlier

void **vRingbufferReturnItemFromISR** (*RingbufHandle\_t xRingbuffer*, void *\*pvItem*, BaseType\_t *\*pxHigherPriorityTaskWoken*)

Return a previously-retrieved item to the ring buffer from an ISR.

**Note** If a split item is retrieved, both parts should be returned by calling this function twice

**Parameters**

- [in] *xRingbuffer*: Ring buffer the item was retrieved from
- [in] *pvItem*: Item that was received earlier
- [out] *pxHigherPriorityTaskWoken*: Value pointed to will be set to `pdTRUE` if the function woke up a higher priority task.

void **vRingbufferDelete** (*RingbufHandle\_t xRingbuffer*)

Delete a ring buffer.

**Note** This function will not deallocate any memory if the ring buffer was created using `xRingbufferCreateStatic()`. Deallocation must be done manually by the user.

**Parameters**

- [in] *xRingbuffer*: Ring buffer to delete

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.

**Note** 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 the an empty no-split buffer regardless of the internal positions of the buffer' s read/write/free pointers.

**Return** Maximum size, in bytes, of an item that can be placed in a ring buffer.

**Parameters**

- [in] xRingbuffer: Ring buffer to query

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.

**Warning** This API is not thread safe. So, if multiple threads are accessing the same ring buffer, it is the application' s responsibility to ensure atomic access to this API and the subsequent Send

**Note** An empty no-split buffer has a max current free size for an item that is limited to ((buffer\_size/2)-header\_size). See API reference for xRingbufferGetMaxItemSize().

**Return** Current free size, in bytes, available for an entry

**Parameters**

- [in] xRingbuffer: Ring buffer to query

BaseType\_t **xRingbufferAddToQueueSetRead** (*RingbufHandle\_t* xRingbuffer, *QueueSetHandle\_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.

**Return**

- pdTRUE on success, pdFALSE otherwise

**Parameters**

- [in] xRingbuffer: Ring buffer to add to the queue set
- [in] xQueueSet: Queue set to add the ring buffer' s read semaphore to

BaseType\_t **xRingbufferCanRead** (*RingbufHandle\_t* xRingbuffer, *QueueSetMemberHandle\_t* xMember)

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.

**Return**

- pdTRUE when semaphore belongs to ring buffer
- pdFALSE otherwise.

**Parameters**

- [in] xRingbuffer: Ring buffer which should be checked
- [in] xMember: Member returned from xQueueSelectFromSet

BaseType\_t **xRingbufferRemoveFromQueueSetRead** (*RingbufHandle\_t* xRingbuffer, *QueueSetHandle\_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

**Return**

- pdTRUE on success
- pdFALSE otherwise

**Parameters**

- [in] xRingbuffer: Ring buffer to remove from the queue set
- [in] xQueueSet: Queue set to remove the ring buffer' s read semaphore from

```
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 the a ring buffer' s current status such as free/read/write pointer positions, and number of items waiting to be retrieved. Arguments can be set to NULL if they are not required.

#### Parameters

- [in] xRingbuffer: Ring buffer to remove from the queue set
- [out] uxFree: Pointer use to store free pointer position
- [out] uxRead: Pointer use to store read pointer position
- [out] uxWrite: Pointer use to store write pointer position
- [out] uxAcquire: Pointer use to store acquire pointer position
- [out] uxItemsWaiting: Pointer use to store number of items (bytes for byte buffer) waiting to be retrieved

```
void xRingbufferPrintInfo (RingbufHandle_t xRingbuffer)
```

Debugging function to print the internal pointers in the ring buffer.

#### Parameters

- xRingbuffer: Ring buffer to show

### 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.

**Note** The CONFIG\_FREERTOS\_SUPPORT\_STATIC\_ALLOCATION option must be enabled for this structure to be available.

### Type Definitions

```
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.

```
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.

**Note** The CONFIG\_FREERTOS\_SUPPORT\_STATIC\_ALLOCATION option must be enabled for this structure to be available.

### Enumerations

```
enum RingbufferType_t
```

Values:

```
RINGBUF_TYPE_NOSPLIT = 0
```

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_ALLOWSPPLIT
```

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.

**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.

**RINGBUF\_TYPE\_MAX**

## Hooks

FreeRTOS consists of Idle Hooks and Tick Hooks which allow for application specific functionality to be added to the Idle Task and Tick Interrupt. ESP-IDF provides its own Idle and Tick Hook API in addition to the hooks provided by Vanilla FreeRTOS. ESP-IDF hooks have the added benefit of being run time configurable and asymmetrical.

**Vanilla FreeRTOS Hooks** Idle and Tick Hooks in vanilla FreeRTOS are implemented by the user defining the functions `vApplicationIdleHook()` and `vApplicationTickHook()` respectively somewhere in the application. Vanilla FreeRTOS will run the user defined Idle Hook and Tick Hook on every iteration of the Idle Task and Tick Interrupt respectively.

Vanilla FreeRTOS hooks are referred to as **Legacy Hooks** in ESP-IDF FreeRTOS. To enable legacy hooks, [\*CONFIG\\_FREERTOS\\_LEGACY\\_HOOKS\*](#) should be enabled in [\*project configuration menu\*](#).

Due to vanilla FreeRTOS being designed for single core, `vApplicationIdleHook()` and `vApplicationTickHook()` can only be defined once. However, the ESP32 is dual core in nature, therefore same Idle Hook and Tick Hook are used for both cores (in other words, the hooks are symmetrical for both cores).

**ESP-IDF Idle and Tick Hooks** Due to the the dual core nature of the ESP32, it may be necessary for some applications to have separate hooks for each core. Furthermore, it may be necessary for the Idle Tasks or Tick Interrupts to execute multiple hooks that are configurable at run time. Therefore the ESP-IDF provides it' s own hooks API in addition to the legacy hooks provided by Vanilla FreeRTOS.

The ESP-IDF tick/idle hooks are registered at run time, and each tick/idle hook must be registered to a specific CPU. When the idle task runs/tick Interrupt occurs on a particular CPU, the CPU will run each of its registered idle/tick hooks in turn.

## Hooks API Reference

### Header File

- [esp\\_common/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.

**Warning** Idle callbacks MUST NOT, UNDER ANY CIRCUMSTANCES, CALL A FUNCTION THAT MIGHT BLOCK.

#### Return

- ESP\_OK: Callback registered to the specified core' s idle hook
- ESP\_ERR\_NO\_MEM: No more space on the specified core' s idle hook to register callback
- ESP\_ERR\_INVALID\_ARG: cpuid is invalid

#### Parameters

- [in] `new_idle_cb`: Callback to be called
- [in] `cpuid`: id of the core



*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.

**Warning** Idle callbacks MUST NOT, UNDER ANY CIRCUMSTANCES, CALL A FUNCTION THAT MIGHT BLOCK.

**Return**

- ESP\_OK: Callback registered to the calling core' s idle hook
- ESP\_ERR\_NO\_MEM: No more space on the calling core' s idle hook to register callback

**Parameters**

- [in] *new\_idle\_cb*: Callback to be called

*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.

**Return**

- 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

**Parameters**

- [in] *new\_tick\_cb*: Callback to be called
- [in] *cpuid*: id of the core

*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.

**Return**

- 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

**Parameters**

- [in] *new\_tick\_cb*: Callback to be called

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.

**Parameters**

- [in] *old\_idle\_cb*: Callback to be unregistered
- [in] *cpuid*: 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.

**Parameters**

- [in] *old\_idle\_cb*: 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.

**Parameters**

- [in] *old\_tick\_cb*: Callback to be unregistered
- [in] *cpuid*: 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.

**Parameters**

- [in] *old\_tick\_cb*: Callback to be unregistered

### Type Definitions

```
typedef bool (*esp_freertos_idle_cb_t)()
typedef void (*esp_freertos_tick_cb_t)()
```

## 2.7.3 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 [应用程序的内存布局](#).

It's also possible to connect external SPI RAM to the ESP32 - *external RAM* 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). When calling `malloc()`, the ESP-IDF `malloc()` implementation internally calls `heap_caps_malloc(size, MALLOC_CAP_8BIT)` in order to allocate DRAM that is byte-addressable. To test the free DRAM heap size at runtime, call `cpp:func:heap_caps_get_free_size(MALLOC_CAP_8BIT)`.

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.

---

**注解:** Due to a technical limitation, the maximum statically allocated DRAM usage is 160KB. The remaining 160KB (for a total of 320KB of DRAM) can only be allocated at runtime as heap.

---

---

**注解:** At runtime, the available heap DRAM may be less than calculated at compile time, because at startup some memory is allocated from the heap before the FreeRTOS scheduler is started (including memory for the stacks of initial FreeRTOS tasks).

---

**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:

```
I (252) heap_init: Initializing. RAM available for dynamic allocation:
I (259) heap_init: At 3FFAE6E0 len 00001920 (6 KiB): DRAM
I (265) heap_init: At 3FFB2EC8 len 0002D138 (180 KiB): DRAM
I (272) heap_init: At 3FFE0440 len 00003AE0 (14 KiB): D/IRAM
I (278) heap_init: At 3FFE4350 len 0001BCB0 (111 KiB): D/IRAM
I (284) heap_init: At 4008944C len 00016BB4 (90 KiB): IRAM
```

**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.

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 [配置片外 RAM](#) for more details.

To use the region above the 4MiB limit, you can use the [himem API](#).

## API Reference - Heap Allocation

### Header File

- [heap/include/esp\\_heap\\_caps.h](#)

**Functions**

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.

In IDF, malloc(*p*) is equivalent to heap\_caps\_malloc(*p*, MALLOC\_CAP\_8BIT).

**Return** A pointer to the memory allocated on success, NULL on failure

**Parameters**

- *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

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*).

**Parameters**

- *ptr*: Pointer to memory previously returned from heap\_caps\_malloc() or heap\_caps\_realloc(). Can be NULL.

void **\*heap\_caps\_realloc** (void \**ptr*, size\_t *size*, int *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).

'caps' parameter can be different to the capabilities that any original 'ptr' was allocated with. In this way, realloc can be used to "move" a buffer if necessary to ensure it meets a new set of capabilities.

**Return** Pointer to a new buffer of size 'size' with capabilities 'caps', or NULL if allocation failed.

**Parameters**

- *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.

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).

**Return** A pointer to the memory allocated on success, NULL on failure

**Parameters**

- *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

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.

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.

**Return** Amount of free bytes in the regions

**Parameters**

- *caps*: Bitwise OR of MALLOC\_CAP\_\* flags indicating the type of memory

`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 water marks of the regions capable of delivering the memory with the given capabilities.

Note the result may be less than the global all-time minimum available heap of this kind, as “low water marks” are tracked per-region. Individual regions’ heaps may have reached their “low water marks” at different points in time. However this result still gives a “worst case” indication for all-time minimum free heap.

**Return** Amount of free bytes in the regions

**Parameters**

- `caps`: Bitwise OR of `MALLOC_CAP_*` flags indicating the type of memory

`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.

**Return** Size of largest free block in bytes.

**Parameters**

- `caps`: Bitwise OR of `MALLOC_CAP_*` flags indicating the type of memory

`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()`.

**Parameters**

- `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.

**Parameters**

- `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`.

**Return** True if all heaps are valid, False if at least one heap is corrupt.

**Parameters**

- `print_errors`: Print specific errors if heap corruption is found.

`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.

**Return** True if all heaps are valid, False if at least one heap is corrupt.

**Parameters**

- `caps`: Bitwise OR of `MALLOC_CAP_*` flags indicating the type of memory
- `print_errors`: Print specific errors if heap corruption is found.

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.

**Note** The entire heap region around the address is checked, not only the adjacent heap blocks.

**Return** 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.

**Parameters**

- *addr*: Address in memory. Check for corruption in region containing this address.
- *print\_errors*: Print specific errors if heap corruption is found.

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.

**Parameters**

- *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.

**Return** A pointer to the memory allocated on success, NULL on failure

**Parameters**

- *size*: Size, in bytes, of the amount of memory to allocate
- *num*: Number of variable paramters

void \***heap\_caps\_realloc\_prefer** (void \**ptr*, size\_t *size*, size\_t *num*, ...)

Allocate a chunk of memory as preference in decreasing order.

**Return** Pointer to a new buffer of size '*size*', or NULL if allocation failed.

**Parameters**

- *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 paramters

void \***heap\_caps\_calloc\_prefer** (size\_t *n*, size\_t *size*, size\_t *num*, ...)

Allocate a chunk of memory as preference in decreasing order.

**Return** A pointer to the memory allocated on success, NULL on failure

**Parameters**

- *n*: Number of continuing chunks of memory to allocate
- *size*: Size, in bytes, of a chunk of memory to allocate
- *num*: Number of variable paramters

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.

**Parameters**

- `caps`: Bitwise OR of `MALLOC_CAP_*` flags indicating the type of memory

void **heap\_caps\_dump\_all** ()

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`.

**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.

**MALLOC\_CAP\_DEFAULT**

Memory can be returned in a non-capability-specific memory allocation (e.g. `malloc()`, `calloc()`) call.

**MALLOC\_CAP\_INVALID**

Memory can't be used / list end marker.

**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. 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.

## 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.)

## API Reference - Initialisation

### Header File

- [heap/include/esp\\_heap\\_caps\\_init.h](#)

### Functions

void **heap\_caps\_init** ()

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** ()

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.

**Return** `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()`.

#### Parameters

- start: Start address of new region.
- end: End address of new region.

*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.

#### Return

- `ESP_OK` on success
- `ESP_ERR_INVALID_ARG` if a parameter is invalid
- `ESP_ERR_NO_MEM` if no memory to register new heap.
- `ESP_ERR_INVALID_SIZE` if the memory region is too small to fit a heap
- `ESP_FAIL` if region overlaps the start and/or end of an existing region

#### Parameters



- `caps`: Ordered array of capability masks for the new region, in order of priority. Must have length `SOC_MEMORY_TYPE_NO_PRIOS`. Does not need to remain valid after the call returns.
- `start`: Start address of new region.
- `end`: End address of new region.

### 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()` or `multi_heap_calloc()` 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 - 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

- `heap/include/multi_heap.h`

#### Functions

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.

**Return** Pointer to new memory, or NULL if allocation fails.

##### Parameters

- `heap`: Handle to a registered heap.
- `size`: Size of desired buffer.

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.

##### Parameters

- `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.

**Return** New buffer of ‘size’ containing contents of ‘p’, or NULL if reallocation failed.

##### Parameters

- `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.
- `size`: Desired new size for buffer.

`size_t multi_heap_get_allocated_size` (*multi\_heap\_handle\_t heap*, void \**p*)

Return the size that a particular pointer was allocated with.

**Return** Size of the memory allocated at this block. May be more than the original size argument, due to padding and minimum block sizes.

**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.

*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.

**Return** Handle of a new heap ready for use, or NULL if the heap region was too small to be initialised.

**Parameters**

- `start`: Start address of the memory to use for a new heap.
- `size`: Size (in bytes) of the new heap.

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.

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.

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_FAIL_PRINTF` in `multi_heap_platform.h`.

**Return** true if heap is valid, false otherwise.

**Parameters**

- `heap`: Handle to a registered heap.
- `print_errors`: If true, errors will be printed to stderr.

`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()`.

**Return** Number of free bytes.

**Parameters**

- `heap`: Handle to a registered heap.

`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 water mark” of possible values returned from `multi_free_heap_size()`, for the specified heap.

**Return** Number of free bytes.

**Parameters**

- `heap`: Handle to a registered heap.

`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.

**Parameters**

- `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 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.

`size_t` **total\_blocks**

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.

## 2.7.4 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 water mark” 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.

**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_set_watchpoint()`. A fatal exception will occur when the watchpoint triggers. For example `esp_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 uninitialized-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 0xCECECECE 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 0xCECECECE 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 0xFEFEFEFE, 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 0xFEFEFEFE 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 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:

```
#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

...

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 ccount 0x2e9b7384 caller↳
↳0x400d276d:0x400d27c1
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 ccount 0x2e9b79c0 caller↳
↳0x400d2776:0x400d27c1
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 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.

- `ccount 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\\_ESP32\\_APPTRACE\\_DESTINATION](#) and select Trace memory.
- In the project configuration menu, navigate to Component settings -> Application Level Tracing -> FreeRTOS SystemView Tracing and enable [CONFIG\\_SYSVIEW\\_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 argument to this function is ignored and 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:

```
#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](#)).



---

**注解:** 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:

```
target remote :3333

mon reset halt
flushregs

tb heap_trace_start
commands
mon esp32 sysview start file:///tmp/heap.svdat
c
end

tb heap_trace_stop
commands
mon esp32 sysview stop
end

c
```

Using this file GDB will connect to the target, reset it, and start tracing when program hits breakpoint at `heap_trace_start()`. Trace data 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 /tmp/heap_log.svdat </path/to/program/elf>`

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/esp-idf/examples/system/sysview_tracing_heap_log/main/
↔sysview_heap_log.c:47
/home/user/projects/esp/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/esp-idf/examples/system/sysview_tracing_heap_log/main/
↔sysview_heap_log.c:48
/home/user/projects/esp/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/esp-idf/examples/system/sysview_tracing_heap_log/main/
↔sysview_heap_log.c:31 (discriminator 9)
/home/user/projects/esp/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/esp-idf/components/freertos/tasks.c:4590
/home/user/projects/esp/esp-idf/components/freertos/tasks.c:4590

[0.002798700] HEAP: Freed bytes @ 0x3ffb50bc from task "main" on core 0 by:
/home/user/projects/esp/esp-idf/components/freertos/tasks.c:4590
/home/user/projects/esp/esp-idf/components/freertos/tasks.c:4590
```

(下页继续)

(续上页)

```
[0.102436025] HEAP: Allocated 2 bytes @ 0x3ffaffe0 from task "alloc" on core 0 by:
/home/user/projects/esp/esp-idf/examples/system/sysview_tracing_heap_log/main/
↪sysview_heap_log.c:47
/home/user/projects/esp/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/esp-idf/examples/system/sysview_tracing_heap_log/main/
↪sysview_heap_log.c:48
/home/user/projects/esp/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/esp-idf/examples/system/sysview_tracing_heap_log/main/
↪sysview_heap_log.c:31 (discriminator 9)
/home/user/projects/esp/esp-idf/components/freertos/port.c:355 (discriminator 1)

[0.202436200] HEAP: Allocated 3 bytes @ 0x3ffaffe8 from task "alloc" on core 0 by:
/home/user/projects/esp/esp-idf/examples/system/sysview_tracing_heap_log/main/
↪sysview_heap_log.c:47
/home/user/projects/esp/esp-idf/components/freertos/port.c:355 (discriminator 1)

[0.202451725] HEAP: Allocated 6 bytes @ 0x3ffaaff0 from task "alloc" on core 0 by:
/home/user/projects/esp/esp-idf/examples/system/sysview_tracing_heap_log/main/
↪sysview_heap_log.c:48
/home/user/projects/esp/esp-idf/components/freertos/port.c:355 (discriminator 1)

[0.202667075] HEAP: Freed bytes @ 0x3ffaaff0 from task "free" on core 0 by:
/home/user/projects/esp/esp-idf/examples/system/sysview_tracing_heap_log/main/
↪sysview_heap_log.c:31 (discriminator 9)
/home/user/projects/esp/esp-idf/components/freertos/port.c:355 (discriminator 1)

[0.302436000] HEAP: Allocated 4 bytes @ 0x3ffaaff0 from task "alloc" on core 0 by:
/home/user/projects/esp/esp-idf/examples/system/sysview_tracing_heap_log/main/
↪sysview_heap_log.c:47
/home/user/projects/esp/esp-idf/components/freertos/port.c:355 (discriminator 1)

[0.302451475] HEAP: Allocated 8 bytes @ 0x3ffb40b8 from task "alloc" on core 0 by:
/home/user/projects/esp/esp-idf/examples/system/sysview_tracing_heap_log/main/
↪sysview_heap_log.c:48
/home/user/projects/esp/esp-idf/components/freertos/port.c:355 (discriminator 1)

[0.302667500] HEAP: Freed bytes @ 0x3ffb40b8 from task "free" on core 0 by:
/home/user/projects/esp/esp-idf/examples/system/sysview_tracing_heap_log/main/
↪sysview_heap_log.c:31 (discriminator 9)
/home/user/projects/esp/esp-idf/components/freertos/port.c:355 (discriminator 1)

Processing completed.
Processed 1019 events
===== HEAP TRACE REPORT =====
Processed 14 heap events.
[0.002244575] HEAP: Allocated 1 bytes @ 0x3ffaafd8 from task "alloc" on core 0 by:
/home/user/projects/esp/esp-idf/examples/system/sysview_tracing_heap_log/main/
↪sysview_heap_log.c:47
/home/user/projects/esp/esp-idf/components/freertos/port.c:355 (discriminator 1)

[0.102436025] HEAP: Allocated 2 bytes @ 0x3ffaffe0 from task "alloc" on core 0 by:
/home/user/projects/esp/esp-idf/examples/system/sysview_tracing_heap_log/main/
↪sysview_heap_log.c:47
/home/user/projects/esp/esp-idf/components/freertos/port.c:355 (discriminator 1)

[0.202436200] HEAP: Allocated 3 bytes @ 0x3ffaffe8 from task "alloc" on core 0 by:
/home/user/projects/esp/esp-idf/examples/system/sysview_tracing_heap_log/main/
↪sysview_heap_log.c:47
```

(下页继续)

(续上页)

```
/home/user/projects/esp/esp-idf/components/freertos/port.c:355 (discriminator 1)
[0.302436000] HEAP: Allocated 4 bytes @ 0x3ffaaff0 from task "alloc" on core 0 by:
/home/user/projects/esp/esp-idf/examples/system/sysview_tracing_heap_log/main/
→sysview_heap_log.c:47
/home/user/projects/esp/esp-idf/components/freertos/port.c:355 (discriminator 1)
Found 10 leaked bytes in 4 blocks.
```

**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 `HEAP_TRACE_ALL` mode allows you to record all of 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.

**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.

**False-Positive Memory Leaks** Not everything printed by `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 `heap_trace_start()` but then freed after `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 `heap_trace_stop()` is called.
- The first time a task uses stdio - for example, when it calls `printf()` - a lock (RTOS mutex semaphore) is allocated by the libc. This allocation lasts until the task is deleted.
- Certain uses of `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, WiFi, 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 `TIME_WAIT` state. After the `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.

## API Reference - Heap Tracing

### Header File

- [heap/include/esp\\_heap\\_trace.h](#)

### Functions

*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)`;

**Return**

- `ESP_ERR_NOT_SUPPORTED` Project was compiled without heap tracing enabled in menuconfig.
- `ESP_ERR_INVALID_STATE` Heap tracing is currently in progress.
- `ESP_OK` Heap tracing initialised successfully.

**Parameters**

- `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.
- `num_records`: Size of the heap trace buffer, as number of record structures.

*esp\_err\_t* **heap\_trace\_init\_tohost** (void)

Initialise heap tracing in host-based mode.

This function must be called before any other heap tracing functions.

**Return**

- `ESP_ERR_INVALID_STATE` Heap tracing is currently in progress.
- `ESP_OK` Heap tracing initialised successfully.

*esp\_err\_t* **heap\_trace\_start** (*heap\_trace\_mode\_t* mode)

Start heap tracing. All heap allocations & frees will be traced, until `heap_trace_stop()` is called.

**Note** `heap_trace_init_standalone()` must be called to provide a valid buffer, before this function is called.

**Note** Calling this function while heap tracing is running will reset the heap trace state and continue tracing.

**Return**

- `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.

**Parameters**

- `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.)

*esp\_err\_t* **heap\_trace\_stop** (void)

Stop heap tracing.

**Return**

- `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..

*esp\_err\_t* **heap\_trace\_resume** (void)

Resume heap tracing which was previously stopped.

Unlike `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 `heap_trace_start()` was last called (or `HEAP_TRACE_ALL` if `heap_trace_start()` was never called).

**Return**

- `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.

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.

**Note** 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.

**Return**

- ESP\_ERR\_NOT\_SUPPORTED Project was compiled without heap tracing enabled in menuconfig.
- ESP\_ERR\_INVALID\_STATE Heap tracing was not initialised.
- ESP\_ERR\_INVALID\_ARG Index is out of bounds for current heap trace record count.
- ESP\_OK Record returned successfully.

**Parameters**

- *index*: Index (zero-based) of the record to return.
- [*out*] *record*: Record where the heap trace record will be copied.

void **heap\_trace\_dump** (void)

Dump heap trace record data to stdout.

**Note** 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

**struct heap\_trace\_record\_t**

Trace record data type. Stores information about an allocated region of memory.

### Public Members

uint32\_t **ccount**

CCOUNT of the CPU when the allocation was made. LSB (bit value 1) is the CPU number (0 or 1).

void \***address**

Address which was allocated.

size\_t **size**

Size of the allocation.

void \***allocated\_by**[CONFIG\_HEAP\_TRACING\_STACK\_DEPTH]

Call stack of the caller which allocated the memory.

void \***freed\_by**[CONFIG\_HEAP\_TRACING\_STACK\_DEPTH]

Call stack of the caller which freed the memory (all zero if not freed.)

### Macros

**CONFIG\_HEAP\_TRACING\_STACK\_DEPTH**

### Enumerations

enum **heap\_trace\_mode\_t**

*Values:*

**HEAP\_TRACE\_ALL**

**HEAP\_TRACE\_LEAKS**

## 2.7.5 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  $\leq 4\text{MiB}$ , 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

- [esp32/include/esp32/himem.h](#)

#### Functions

*esp\_err\_t* **esp\_himem\_alloc** (*size\_t* size, *esp\_himem\_handle\_t* \*handle\_out)

Allocate a block in high memory.

**Return** - ESP\_OK if succesful

- ESP\_ERR\_NO\_MEM if out of memory
- ESP\_ERR\_INVALID\_SIZE if size is not a multiple of 32K

#### Parameters

- 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).
- [out] handle\_out: Handle to be returned

*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.

This allocates a contiguous CPU memory region that can be used to map blocks of physical memory into.

**Return** - ESP\_OK if succesful

- ESP\_ERR\_NO\_MEM if out of memory or address space
- ESP\_ERR\_INVALID\_SIZE if size is not a multiple of 32K

#### Parameters

- size: Size of the range to be allocated. Note this needs to be a multiple of the external RAM mmu block size (32K).
- [out] handle\_out: Handle to be returned

*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.

**Note** The region to be mapped needs to have offsets and sizes that are aligned to the SPI RAM MMU block size (32K)

**Return** - 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.

**Parameters**

- `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] out_ptr`: Pointer to variable to store resulting memory pointer in

`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.

**Return** - `ESP_OK` if the memory is successfully freed

- `ESP_ERR_INVALID_ARG` if the handle still is (partially) mapped

**Parameters**

- `handle`: Handle to the block of memory, as given by `esp_himem_alloc`

`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.

**Return** - `ESP_OK` if the memory is successfully freed

- `ESP_ERR_INVALID_ARG` if the handle still is (partially) mapped to

**Parameters**

- `handle`: Handle to the range block, as given by `esp_himem_alloc_map_range`

`esp_err_t esp_himem_unmap(esp_himem_rangehandle_t range, void *ptr, size_t len)`

Unmap a region.

**Return** - `ESP_OK` if the memory is successfully unmapped,

- `ESP_ERR_INVALID_ARG` if `ptr` or `len` are invalid.

**Parameters**

- `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)

`size_t esp_himem_get_phys_size()`

Get total amount of memory under control of himem API.

**Return** Amount of memory, in bytes

`size_t esp_himem_get_free_size()`

Get free amount of memory under control of himem API.

**Return** Amount of free memory, in bytes

`size_t esp_himem_reserved_area_size()`

Get amount of SPI memory address space needed for bankswitching.

**Note** 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.

**Return** 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
```

## 2.7.6 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_sintrstatus`). It can use the flags passed to this function to set the type of interrupt allocated, specifying a specific 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 has two different types of interrupts it handles differently: Shared interrupts and non-shared interrupts. The simplest of the two 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. Shared interrupts can have multiple peripherals triggering it, 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 see if any action is required.

Non-shared interrupts can be either level- or edge-triggered. Shared interrupts can only be level interrupts (because of the chance of missed interrupts when edge interrupts are used.) (The logic behind this: DevA and DevB share an int. DevB signals an int. Int line goes high. ISR handler calls code for DevA -> does nothing. ISR handler calls code for DevB, but while doing that, DevA signals an int. ISR DevB is done, clears int for DevB, exits interrupt code. Now an interrupt for DevA is still pending, but because the int line never went low (DevA kept it high even when the int 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 e.g. 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 (erases 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'll be all 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 2 ways to stop a interrupt from being triggered: *disable the source* or *mask peripheral interrupt status*. IDF only handles the enabling and disabling of the source itself, leaving status and mask bits to be handled by users. **Status bits should always be masked before the handler responsible for it is disabled, or the status should be handled in other enabled interrupt properly.** You may leave some status bits unhandled if you just disable one of all the handlers without masking the status bits, which causes the interrupt to trigger infinitely resulting in a system crash.

## API Reference

### Header File

- [esp32/include/esp\\_intr\\_alloc.h](#)

### Functions

`esp_err_t 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.

**Return** `ESP_ERR_INVALID_ARG` if `cpu` or `intno` is invalid `ESP_OK` otherwise

**Parameters**

- `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.

`esp_err_t 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.

**Return** `ESP_ERR_INVALID_ARG` if `cpu` or `intno` is invalid `ESP_OK` otherwise

**Parameters**

- `intno`: The number of the interrupt (0-31)
- `cpu`: CPU on which the interrupt should be marked as shared (0 or 1)

`esp_err_t 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.

**Return** `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

**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 ORred 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.

`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

**Return** `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

**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 ORred 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.

*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.

**Note** When the handler shares its source with other handlers, the interrupt status bits it's responsible for should be managed properly before freeing it. see `esp_intr_disable` for more details. Please do not call this function in `esp_ipc_call_blocking`.

**Return** `ESP_ERR_INVALID_ARG` the handle is `NULL` `ESP_FAIL` failed to release this handle `ESP_OK` otherwise

#### Parameters

- `handle`: The handle, as obtained by `esp_intr_alloc` or `esp_intr_alloc_intrstatus`

`int` `esp_intr_get_cpu` (*intr\_handle\_t* handle)

Get CPU number an interrupt is tied to.

**Return** The core number where the interrupt is allocated

#### Parameters

- `handle`: The handle, as obtained by `esp_intr_alloc` or `esp_intr_alloc_intrstatus`

`int` `esp_intr_get_intno` (*intr\_handle\_t* handle)

Get the allocated interrupt for a certain handle.

**Return** The interrupt number

#### Parameters

- `handle`: The handle, as obtained by `esp_intr_alloc` or `esp_intr_alloc_intrstatus`

*esp\_err\_t* `esp_intr_disable` (*intr\_handle\_t* handle)

Disable the interrupt associated with the handle.

#### Note

1. 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.
2. 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.

**Return** `ESP_ERR_INVALID_ARG` if the combination of arguments is invalid. `ESP_OK` otherwise

#### Parameters

- `handle`: The handle, as obtained by `esp_intr_alloc` or `esp_intr_alloc_intrstatus`

*esp\_err\_t* `esp_intr_enable` (*intr\_handle\_t* handle)

Enable the interrupt associated with the handle.

**Note** 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.

**Return** `ESP_ERR_INVALID_ARG` if the combination of arguments is invalid. `ESP_OK` otherwise

#### Parameters

- `handle`: The handle, as obtained by `esp_intr_alloc` or `esp_intr_alloc_intrstatus`

*esp\_err\_t* **esp\_intr\_set\_in\_iram** (*intr\_handle\_t* handle, bool *is\_in\_iram*)

Set the “in IRAM” status of the handler.

**Note** Does not work on shared interrupts.

**Return** ESP\_ERR\_INVALID\_ARG if the combination of arguments is invalid. ESP\_OK otherwise

**Parameters**

- *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.

void **esp\_intr\_noniram\_disable** ()

Disable interrupts that aren't specifically marked as running from IRAM.

void **esp\_intr\_noniram\_enable** ()

Re-enable interrupts disabled by `esp_intr_noniram_disable`.

## 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**

Xtensa 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**

Xtensa timer 1 interrupt source.

**ETS\_INTERNAL\_TIMER2\_INTR\_SOURCE**

Xtensa 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\_INTR\_SOURCE\_OFF**

**ESP\_INTR\_ENABLE** (inum)

**ESP\_INTR\_DISABLE** (inum)

### Type Definitions

```
typedef void (*intr_handler_t) (void *arg)
```

```
typedef struct intr_handle_data_t intr_handle_data_t
```

```
typedef intr_handle_data_t *intr_handle_t
```

## 2.7.7 Watchdogs

### Overview

The ESP-IDF has support for two types of watchdogs: 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.

**Interrupt watchdog** The interrupt watchdog makes sure the FreeRTOS task switching interrupt isn't blocked for a long time. This is bad because no other tasks, including potentially important ones like the WiFi task and the idle task, can't get any CPU runtime. A blocked task switching interrupt can happen because a program runs into an infinite loop with interrupts disabled or hangs in an interrupt.

The default action of the interrupt watchdog is to invoke the panic handler, causing a register dump and an opportunity for the programmer to find out, using either OpenOCD or gdbstub, what bit of code is stuck with interrupts disabled. Depending on the configuration of the panic handler, it can also blindly reset the CPU, which may be preferred in a production environment.

The interrupt watchdog is built around the hardware watchdog in timer group 1. If this watchdog for some reason cannot execute the NMI handler that invokes the panic handler (e.g. because IRAM is overwritten by garbage), it will hard-reset the SOC.

**Task Watchdog Timer** The Task Watchdog Timer (TWDT) is responsible for detecting instances of tasks running for a prolonged period of time without yielding. This is a symptom of CPU starvation and is usually caused by a higher priority task looping without yielding to a lower-priority task thus starving the lower priority task from CPU time. This can be an indicator of poorly written code that spinloops on a peripheral, or a task that is stuck in an infinite loop.

By default the TWDT will watch the Idle Tasks of each CPU, however any task can elect to be watched by the TWDT. Each watched task must ‘reset’ the TWDT periodically to indicate that they have been allocated CPU time. If a task does not reset within the TWDT timeout period, a warning will be printed with information about which tasks failed to reset the TWDT in time and which tasks are currently running on the ESP32 CPUs. And also there is a possibility to redefine the function `esp_task_wdt_isr_user_handler` in the user code to receive this event.

The TWDT is built around the Hardware Watchdog Timer in Timer Group 0. The TWDT can be initialized by calling `esp_task_wdt_init()` which will configure the hardware timer. A task can then subscribe to the TWDT using `esp_task_wdt_add()` in order to be watched. Each subscribed task must periodically call `esp_task_wdt_reset()` to reset the TWDT. Failure by any subscribed tasks to periodically call `esp_task_wdt_reset()` indicates that one or more tasks have been starved of CPU time or are stuck in a loop somewhere.

A watched task can be unsubscribed from the TWDT using `esp_task_wdt_delete()`. A task that has been unsubscribed should no longer call `esp_task_wdt_reset()`. Once all tasks have unsubscribed from the TWDT, the TWDT can be deinitialized by calling `esp_task_wdt_deinit()`.

By default `CONFIG_ESP_TASK_WDT` in *Project Configuration Menu* be enabled causing the TWDT to be initialized automatically during startup. Likewise `CONFIG_ESP_TASK_WDT_CHECK_IDLE_TASK_CPU0` and `CONFIG_ESP_TASK_WDT_CHECK_IDLE_TASK_CPU1` are also enabled by default causing the two Idle Tasks to be subscribed to the TWDT during startup.

**JTAG and 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 reenale 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.

## Interrupt Watchdog API Reference

### Header File

- [esp32/include/esp\\_int\\_wdt.h](#)

### Functions

void `esp_int_wdt_init()`

Initialize the non-CPU-specific parts of interrupt watchdog. This is called in the init code if the interrupt watchdog is enabled in menuconfig.

## Task Watchdog API Reference

A full example using the Task Watchdog is available in esp-idf: [system/task\\_watchdog](#)

### Header File

- [esp\\_common/include/esp\\_task\\_wdt.h](#)

### Functions

`esp_err_t esp_task_wdt_init` (uint32\_t *timeout*, bool *panic*)

Initialize the Task Watchdog Timer (TWDT)

This function configures and initializes the TWDT. If the TWDT is already initialized when this function is called, this function will update the TWDT’s timeout period and panic configurations instead. After initializing the TWDT, any task can elect to be watched by the TWDT by subscribing to it using `esp_task_wdt_add()`.

**Return**

- ESP\_OK: Initialization was successful
- ESP\_ERR\_NO\_MEM: Initialization failed due to lack of memory

**Note** `esp_task_wdt_init()` must only be called after the scheduler started

**Parameters**

- [in] `timeout`: Timeout period of TWDT in seconds
- [in] `panic`: Flag that controls whether the panic handler will be executed when the TWDT times out

**`esp_err_t esp_task_wdt_deinit()`**

Deinitialize the Task Watchdog Timer (TWDT)

This function will deinitialize the TWDT. Calling this function whilst tasks are still subscribed to the TWDT, or when the TWDT is already deinitialized, will result in an error code being returned.

**Return**

- ESP\_OK: TWDT successfully deinitialized
- ESP\_ERR\_INVALID\_STATE: Error, tasks are still subscribed to the TWDT
- ESP\_ERR\_NOT\_FOUND: Error, TWDT has already been deinitialized

**`esp_err_t esp_task_wdt_add(TaskHandle_t 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. If the task being subscribed is one of the Idle Tasks, this function will automatically enable `esp_task_wdt_reset()` to be called from the Idle Hook of the Idle Task. Calling this function whilst the TWDT is uninitialized or attempting to subscribe an already subscribed task will result in an error code being returned.

**Return**

- ESP\_OK: Successfully subscribed the task to the TWDT
- ESP\_ERR\_INVALID\_ARG: Error, the task is already subscribed
- ESP\_ERR\_NO\_MEM: Error, could not subscribe the task due to lack of memory
- ESP\_ERR\_INVALID\_STATE: Error, the TWDT has not been initialized yet

**Parameters**

- [in] `handle`: Handle of the task. Input NULL to subscribe the current running task to the TWDT

**`esp_err_t esp_task_wdt_reset()`**

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. If the IDLE tasks have been subscribed to the TWDT, they will automatically call this function from their idle hooks. Calling this function from a task that has not subscribed to the TWDT, or when the TWDT is uninitialized will result in an error code being returned.

**Return**

- ESP\_OK: Successfully reset the TWDT on behalf of the currently running task
- ESP\_ERR\_NOT\_FOUND: Error, the current running task has not subscribed to the TWDT
- ESP\_ERR\_INVALID\_STATE: Error, the TWDT has not been initialized yet

**`esp_err_t esp_task_wdt_delete(TaskHandle_t handle)`**

Unsubscribes 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()`. If the task is an IDLE task, this function will automatically disable the calling of `esp_task_wdt_reset()` from the Idle Hook. Calling this function whilst the TWDT is uninitialized or attempting to unsubscribe an already unsubscribed task from the TWDT will result in an error code being returned.

**Return**

- ESP\_OK: Successfully unsubscribed the task from the TWDT
- ESP\_ERR\_INVALID\_ARG: Error, the task is already unsubscribed
- ESP\_ERR\_INVALID\_STATE: Error, the TWDT has not been initialized yet

**Parameters**

- [in] `handle`: Handle of the task. Input NULL to unsubscribe the current running task.

`esp_err_t esp_task_wdt_status (TaskHandle_t 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.

**Return :**

- ESP\_OK: The task is currently subscribed to the TWDT
- ESP\_ERR\_NOT\_FOUND: The task is currently not subscribed to the TWDT
- ESP\_ERR\_INVALID\_STATE: The TWDT is not initialized, therefore no tasks can be subscribed

**Parameters**

- [in] `handle`: Handle of the task. Input NULL to query the current running task.

## 2.7.8 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 assigned in eFuse table. This section 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](#) in part 20 eFuse controller. 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).

**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_MAX_BLK_LEN`, 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`:

```
# Factory MAC address #
#####
MAC_FACTORY,          EFUSE_BLK0,    72,    8,    Factory MAC addr [0]
,                    EFUSE_BLK0,    64,    8,    Factory MAC addr [1]
,                    EFUSE_BLK0,    56,    8,    Factory MAC addr [2]
,                    EFUSE_BLK0,    48,    8,    Factory MAC addr [3]
,                    EFUSE_BLK0,    40,    8,    Factory MAC addr [4]
,                    EFUSE_BLK0,    32,    8,    Factory MAC addr [5]
MAC_FACTORY_CRC,     EFUSE_BLK0,    80,    8,    CRC8 for factory MAC address
```

This field will available in code as `ESP_EFUSE_MAC_FACTORY` and `ESP_EFUSE_MAC_FACTORY_CRC`.

## 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:

```
cd $IDF_PATH/components/efuse/
./efuse_table_gen.py esp32/esp_efuse_table.csv
```

After generation in the folder `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 esp32/esp_efuse_table.csv PROJECT_PATH/main/esp_efuse_custom_
↪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"
```

### Support 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. `EFUSE_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 are not used.

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 COM4 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 `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.

After changing the coding scheme, run `efuse_common_table` and `efuse_custom_table` commands to check the tables of the new coding scheme.

### 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.

For frequently used fields, special functions are made, like this `esp_efuse_get_chip_ver()`, `esp_efuse_get_pkg_ver()`.

### How 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
eFuse coding scheme: NONE
#      field_name          efuse_block    bit_start      bit_count
1      WR_DIS_FLASH_CRYPT_CNT EFUSE_BLK0     2              1
2      WR_DIS_BLK1          EFUSE_BLK0     7              1
3      WR_DIS_BLK2          EFUSE_BLK0     8              1
4      WR_DIS_BLK3          EFUSE_BLK0     9              1
5      RD_DIS_BLK1          EFUSE_BLK0     16             1
6      RD_DIS_BLK2          EFUSE_BLK0     17             1
7      RD_DIS_BLK3          EFUSE_BLK0     18             1
8      FLASH_CRYPT_CNT      EFUSE_BLK0     20             7
9      MAC_FACTORY          EFUSE_BLK0     32             8
10     MAC_FACTORY          EFUSE_BLK0     40             8
11     MAC_FACTORY          EFUSE_BLK0     48             8
12     MAC_FACTORY          EFUSE_BLK0     56             8
13     MAC_FACTORY          EFUSE_BLK0     64             8
14     MAC_FACTORY          EFUSE_BLK0     72             8
15     MAC_FACTORY_CRC      EFUSE_BLK0     80             8
16     CHIP_VER_DIS_APP_CPU  EFUSE_BLK0     96             1
17     CHIP_VER_DIS_BT      EFUSE_BLK0     97             1
18     CHIP_VER_PKG          EFUSE_BLK0     105            3
19     CHIP_CPU_FREQ_LOW    EFUSE_BLK0     108            1
20     CHIP_CPU_FREQ_RATED  EFUSE_BLK0     109            1
21     CHIP_VER_REV1        EFUSE_BLK0     111            1
22     ADC_VREF_AND_SDIO_DREF EFUSE_BLK0     136            6
23     XPD_SDIO_REG         EFUSE_BLK0     142            1
24     SDIO_TIEH            EFUSE_BLK0     143            1
25     SDIO_FORCE           EFUSE_BLK0     144            1
26     ENCRYPT_CONFIG        EFUSE_BLK0     188            4
27     CONSOLE_DEBUG_DISABLE EFUSE_BLK0     194            1
28     ABS_DONE_0           EFUSE_BLK0     196            1
29     DISABLE_JTAG         EFUSE_BLK0     198            1
30     DISABLE_DL_ENCRYPT    EFUSE_BLK0     199            1
31     DISABLE_DL_DECRYPT    EFUSE_BLK0     200            1
32     DISABLE_DL_CACHE     EFUSE_BLK0     201            1
33     ENCRYPT_FLASH_KEY     EFUSE_BLK1     0              256
34     SECURE_BOOT_KEY      EFUSE_BLK2     0              256
35     MAC_CUSTOM_CRC       EFUSE_BLK3     0              8
36     MAC_CUSTOM           EFUSE_BLK3     8              48
37     ADC1_TP_LOW          EFUSE_BLK3     96             7
38     ADC1_TP_HIGH         EFUSE_BLK3     103            9
39     ADC2_TP_LOW          EFUSE_BLK3     112            7
40     ADC2_TP_HIGH         EFUSE_BLK3     119            9
41     SECURE_VERSION       EFUSE_BLK3     128            32
```

(下页继续)

```

42      MAC_CUSTOM_VER      EFUSE_BLK3      184      8

Used bits in eFuse table:
EFUSE_BLK0
[2 2] [7 9] [16 18] [20 27] [32 87] [96 97] [105 109] [111 111] [136 144] [188_
↪191] [194 194] [196 196] [198 201]

EFUSE_BLK1
[0 255]

EFUSE_BLK2
[0 255]

EFUSE_BLK3
[0 55] [96 159] [184 191]

Note: Not printed ranges are free for using. (bits in EFUSE_BLK0 are reserved for_
↪Espressif)

Parsing eFuse CSV input file $IDF_PATH/components/efuse/esp32/esp_efuse_table.csv .
↪...
Verifying eFuse table...

```

The number of bits not included in square brackets is free (bits in EFUSE\_BLK0 are reserved for Espressif). All fields are checked for overlapping.

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.

## 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.

**espefuse.py** esptool includes a useful tool for reading/writing ESP32 eFuse bits - [espefuse.py](#).

```

espefuse.py -p COM4 summary

espefuse.py v2.3.1
Connecting....._
Security fuses:
FLASH_CRYPT_CNT      Flash encryption mode counter      = 0 R/W_
↪ (0x0)
FLASH_CRYPT_CONFIG  Flash encryption config (key tweak bits) = 0 R/W_
↪ (0x0)
CONSOLE_DEBUG_DISABLE Disable ROM BASIC interpreter fallback = 1 R/W_
↪ (0x1)
ABS_DONE_0          secure boot enabled for bootloader = 0 R/W_
↪ (0x0)
ABS_DONE_1          secure boot abstract 1 locked      = 0 R/W_
↪ (0x0)
JTAG_DISABLE        Disable JTAG                        = 0 R/W_
↪ (0x0)
DISABLE_DL_ENCRYPT   Disable flash encryption in UART bootloader = 0 R/W_
↪ (0x0)
DISABLE_DL_DECRYPT   Disable flash decryption in UART bootloader = 0 R/W_
↪ (0x0)

```

(下页继续)



```

ADC2_TP_LOW          ADC2 150mV reading          = 449 R/W
↳ (0x7)
ADC2_TP_HIGH        ADC2 850mV reading          = 3362 R/
↳ W (0x1f5)

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.

```

espefuse.py -p COM4 dump

$ espefuse.py -p COM4 dump
espefuse.py v2.3.1
Connecting.....__
EFUSE block 0:
00000000 c403bb68 0082240a 00000000 00000035 00000000 00000000
EFUSE block 1:
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
EFUSE block 2:
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
EFUSE block 3:
00000000 00000000 00000000 00000000 00000000 00000000 00000000

```

## Header File

- [efuse/include/esp\\_efuse.h](#)

## Functions

**`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.

### Return

- ESP\_OK: The operation was successfully completed.
- ESP\_ERR\_INVALID\_ARG: Error in the passed arguments.

### Parameters

- [in] field: A pointer to the structure describing the fields of efuse.
- [out] dst: A pointer to array that will contain the result of reading.
- [in] dst\_size\_bits: 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.

**`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.

### Return

- ESP\_OK: The operation was successfully completed.
- ESP\_ERR\_INVALID\_ARG: Error in the passed arguments.

### Parameters

- [in] field: A pointer to the structure describing the fields of efuse.
- [out] out\_cnt: A pointer that will contain the number of programmed as “1” bits.

**`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.

#### Return

- 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.

#### Parameters

- [in] field: A pointer to the structure describing the fields of efuse.
- [in] src: A pointer to array that contains the data for writing.
- [in] src\_size\_bits: The number of bits required to write.

*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.

#### Return

- 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.

#### Parameters

- [in] field: A pointer to the structure describing the fields of efuse.
- [in] cnt: Required number of programmed as “1” bits.

*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.

#### Return

- 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.

#### Parameters

- [in] blk: Block number of eFuse. (EFUSE\_BLK1, EFUSE\_BLK2 and EFUSE\_BLK3)

*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.

#### Return

- 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.

#### Parameters

- [in] blk: Block number of eFuse. (EFUSE\_BLK1, EFUSE\_BLK2 and EFUSE\_BLK3)

int **esp\_efuse\_get\_field\_size** (const *esp\_efuse\_desc\_t* \*field[])

Returns the number of bits used by field.

**Return** Returns the number of bits used by field.

#### Parameters

- [in] field: A pointer to the structure describing the fields of efuse.

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)

**Return** Value of register

**Parameters**

- [in] blk: Block number of eFuse.
- [in] num\_reg: 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)

**Return**

- ESP\_OK: The operation was successfully completed.
- ESP\_ERR\_EFUSE\_REPEATED\_PROG: Error repeated programming of programmed bits is strictly forbidden.

**Parameters**

- [in] blk: Block number of eFuse.
- [in] num\_reg: The register number in the block.
- [in] val: Value to write.

*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.

**Return** Return efuse coding scheme for blocks

**Parameters**

- [in] blk: Block number of eFuse.

*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.

**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.

**Parameters**

- [in] blk: Block number of eFuse.
- [in] dst\_key: A pointer to array that will contain the result of reading.
- [in] offset\_in\_bits: Start bit in block.
- [in] size\_bits: The number of bits required to read.

*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.

**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

**Parameters**

- [in] blk: Block number of eFuse.
- [in] src\_key: A pointer to array that contains the key for writing.
- [in] offset\_in\_bits: Start bit in block.
- [in] size\_bits: The number of bits required to write.

uint8\_t **esp\_efuse\_get\_chip\_ver** (void)

Returns chip version from efuse.

**Return** chip version



uint32\_t **esp\_efuse\_get\_pkg\_ver** (void)

Returns chip package from efuse.

**Return** chip package

void **esp\_efuse\_burn\_new\_values** (void)

void **esp\_efuse\_reset** (void)

void **esp\_efuse\_disable\_basic\_rom\_console** (void)

*esp\_err\_t* **esp\_efuse\_apply\_34\_encoding** (const uint8\_t \**in\_bytes*, uint32\_t \**out\_words*, size\_t *in\_bytes\_len*)

void **esp\_efuse\_write\_random\_key** (uint32\_t *blk\_wdata0\_reg*)

uint32\_t **esp\_efuse\_read\_secure\_version** ()

bool **esp\_efuse\_check\_secure\_version** (uint32\_t *secure\_version*)

*esp\_err\_t* **esp\_efuse\_update\_secure\_version** (uint32\_t *secure\_version*)

void **esp\_efuse\_init** (uint32\_t *offset*, uint32\_t *size*)

**static** bool **soc\_has\_cache\_lock\_bug** (void)

### Structures

**struct esp\_efuse\_desc\_t**

Structure eFuse field.

### Public Members

*esp\_efuse\_block\_t* **efuse\_block** : 8

Block of eFuse

uint8\_t **bit\_start**

Start bit [0..255]

uint16\_t **bit\_count**

Length of bit field [1..-]

### 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.

### Enumerations

**enum esp\_efuse\_block\_t**

Type of eFuse blocks.

*Values:*

**EFUSE\_BLK0** = 0

Number of eFuse block. Reserved.

**EFUSE\_BLK1 = 1**

Number of eFuse block. Used for Flash Encryption. If not using that Flash Encryption feature, they can be used for another purpose.

**EFUSE\_BLK2 = 2**

Number of eFuse block. Used for Secure Boot. If not using that Secure Boot feature, they can be used for another purpose.

**EFUSE\_BLK3 = 3**

Number of eFuse block. Uses for the purpose of the user.

**enum esp\_efuse\_coding\_scheme\_t**

Type of coding scheme.

*Values:*

**EFUSE\_CODING\_SCHEME\_NONE = 0**

None

**EFUSE\_CODING\_SCHEME\_3\_4 = 1**

3/4 coding

**EFUSE\_CODING\_SCHEME\_REPEAT = 2**

Repeat coding

## 2.7.9 Inter-Processor Call

### Overview

Due to the dual core nature of the ESP32, there are instances where a certain function must be run in the context of a particular core (e.g. allocating ISR to an interrupt source of a particular core). The IPC (Inter-Processor Call) feature allows for the execution of functions on a particular CPU.

A given function can be executed on a particular core by calling `esp_ipc_call()` or `esp_ipc_call_blocking()`. IPC is implemented via two high priority FreeRTOS tasks pinned to each CPU known as the IPC Tasks. The two IPC Tasks remain inactive (blocked) until `esp_ipc_call()` or `esp_ipc_call_blocking()` is called. When an IPC Task of a particular core is unblocked, it will preempt the current running task on that core and execute a given function.

### Usage

`esp_ipc_call()` unblocks the IPC task on a particular core to execute a given function. The task that calls `esp_ipc_call()` will be blocked until the IPC Task begins execution of the given function. `esp_ipc_call_blocking()` is similar but will block the calling task until the IPC Task has completed execution of the given function.

Functions executed by IPCs must be functions of type `void func(void *arg)`. To run more complex functions which require a larger stack, the IPC tasks' stack size can be configured by modifying `CONFIG_ESP_IPC_TASK_STACK_SIZE` in `menuconfig`. The IPC API is protected by a mutex hence simultaneous IPC calls are not possible.

Care should be taken to avoid deadlock when writing functions to be executed by IPC, especially when attempting to take a mutex within the function.

### API Reference

#### Header File

- `esp_common/include/esp_ipc.h`

## Functions

*esp\_err\_t* **esp\_ipc\_call** (uint32\_t *cpu\_id*, esp\_ipc\_func\_t *func*, void \**arg*)

Execute a function on the given CPU.

Run a given function on a particular CPU. The given function must accept a void\* argument and return void. The given function is run in the context of the IPC task of the CPU specified by the *cpu\_id* parameter. The calling task will be blocked until the IPC task begins executing the given function. If another IPC call is ongoing, the calling task will block until the other IPC call completes. The stack size allocated for the IPC task can be configured in the “Inter-Processor Call (IPC) task stack size” setting in menuconfig. Increase this setting if the given function requires more stack than default.

**Note** In single-core mode, returns ESP\_ERR\_INVALID\_ARG for *cpu\_id* 1.

### Return

- ESP\_ERR\_INVALID\_ARG if *cpu\_id* is invalid
- ESP\_ERR\_INVALID\_STATE if the FreeRTOS scheduler is not running
- ESP\_OK otherwise

### Parameters

- [in] *cpu\_id*: CPU where the given function should be executed (0 or 1)
- [in] *func*: Pointer to a function of type void func(void\* arg) to be executed
- [in] *arg*: Arbitrary argument of type void\* to be passed into the function

*esp\_err\_t* **esp\_ipc\_call\_blocking** (uint32\_t *cpu\_id*, esp\_ipc\_func\_t *func*, void \**arg*)

Execute a function on the given CPU and blocks until it completes.

Run a given function on a particular CPU. The given function must accept a void\* argument and return void. The given function is run in the context of the IPC task of the CPU specified by the *cpu\_id* parameter. The calling task will be blocked until the IPC task completes execution of the given function. If another IPC call is ongoing, the calling task will block until the other IPC call completes. The stack size allocated for the IPC task can be configured in the “Inter-Processor Call (IPC) task stack size” setting in menuconfig. Increase this setting if the given function requires more stack than default.

**Note** In single-core mode, returns ESP\_ERR\_INVALID\_ARG for *cpu\_id* 1.

### Return

- ESP\_ERR\_INVALID\_ARG if *cpu\_id* is invalid
- ESP\_ERR\_INVALID\_STATE if the FreeRTOS scheduler is not running
- ESP\_OK otherwise

### Parameters

- [in] *cpu\_id*: CPU where the given function should be executed (0 or 1)
- [in] *func*: Pointer to a function of type void func(void\* arg) to be executed
- [in] *arg*: Arbitrary argument of type void\* to be passed into the function

## 2.7.10 High Resolution 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 can dispatch the callbacks for these events as corresponding hardware interrupts happen.

*esp\_timer* set of APIs provide such facility. Internally, *esp\_timer* uses a 32-bit hardware timer (FRC1, “legacy” timer). *esp\_timer* provides one-shot and periodic timers, microsecond time resolution, and 64-bit range.

Timer callbacks are dispatched from a high-priority *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 `esp_timer` are running, callback dispatching will be delayed until `esp_timer` task has a chance to run. For example, this will happen if a SPI Flash operation is in progress.

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 `esp_timer`. If `esp_timer_start_once()` is called with a timeout value less than 20us, the callback will be dispatched only after approximately 20us.

Periodic `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 `esp_timer` APIs

Single timer is represented by `esp_timer_handle_t` type. Timer has a callback function associated with it. This callback function is called from the `esp_timer` task each time the timer elapses.

- To create a timer, call `esp_timer_create()`.
- 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.

## 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()`:

- 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

- [esp\\_common/include/esp\\_timer.h](#)

### Functions

`esp_err_t esp_timer_init()`  
Initialize `esp_timer` library.

**Note** This function is called from startup code. Applications do not need to call this function before using other `esp_timer` APIs.

#### Return

- `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** ()

De-initialize esp\_timer library.

**Note** Normally this function should not be called from applications

**Return**

- 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.

**Note** When done using the timer, delete it with esp\_timer\_delete function.

**Return**

- 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

**Parameters**

- create\_args: Pointer to a structure with timer creation arguments. Not saved by the library, can be allocated on the stack.
- [out] out\_handle: Output, pointer to esp\_timer\_handle\_t variable which will hold the created timer handle.

*esp\_err\_t* **esp\_timer\_start\_once** (*esp\_timer\_handle\_t* timer, uint64\_t timeout\_us)

Start one-shot timer.

Timer should not be running when this function is called.

**Return**

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG if the handle is invalid
- ESP\_ERR\_INVALID\_STATE if the timer is already running

**Parameters**

- timer: timer handle created using esp\_timer\_create
- timeout\_us: timer timeout, in microseconds relative to the current moment

*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.

**Return**

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG if the handle is invalid
- ESP\_ERR\_INVALID\_STATE if the timer is already running

**Parameters**

- timer: timer handle created using esp\_timer\_create
- period: timer period, in microseconds

*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.

**Return**

- ESP\_OK on success
- ESP\_ERR\_INVALID\_STATE if the timer is not running

**Parameters**

- timer: timer handle created using esp\_timer\_create

*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.

**Return**

- ESP\_OK on success
- ESP\_ERR\_INVALID\_STATE if the timer is not running

**Parameters**

- timer: timer handle allocated using esp\_timer\_create

int64\_t **esp\_timer\_get\_time** ()

Get time in microseconds since boot.

**Return** number of microseconds since esp\_timer\_init was called (this normally happens early during application startup).

int64\_t **esp\_timer\_get\_next\_alarm** ()

Get the timestamp when the next timeout is expected to occur.

**Return** 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\_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 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

**Return**

- ESP\_OK on success
- ESP\_ERR\_NO\_MEM if can not allocate temporary buffer for the output

**Parameters**

- stream: stream (such as stdout) to dump the information to

## Structures

**struct esp\_timer\_create\_args\_t**

Timer configuration passed to esp\_timer\_create.

## Public Members

*esp\_timer\_cb\_t* **callback**

Function to call when timer expires.

void \***arg**

Argument to pass to the callback.

*esp\_timer\_dispatch\_t* **dispatch\_method**

Call the callback from task or from ISR.

**const char \*name**  
Timer name, used in `esp_timer_dump` function.

### Type Definitions

**typedef struct esp\_timer \*esp\_timer\_handle\_t**

Opaque type representing a single `esp_timer`.

**typedef void (\*esp\_timer\_cb\_t)(void \*arg)**

Timer callback function type.

### Parameters

- `arg`: pointer to opaque user-specific data

### Enumerations

**enum esp\_timer\_dispatch\_t**

Method for dispatching timer callback.

*Values:*

**ESP\_TIMER\_TASK**

Callback is called from timer task.

## 2.7.11 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`. All logging statements for verbosity levels higher than `CONFIG_LOG_DEFAULT_LEVEL` will be removed by the preprocessor.
- **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)

---

**注解:** The function `esp_log_level_set()` cannot set logging levels higher than specified by `CONFIG_LOG_DEFAULT_LEVEL`. To increase log level for a specific file at compile time, use the macro `LOG_LOCAL_LEVEL` (see the details below).

---

### How to use this library

In each C file that uses logging functionality, define the TAG variable as shown below:

```
static const char* TAG = "MyModule";
```

Then use one of logging macros to produce output, e.g:

```
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.

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.:

```
#define LOG_LOCAL_LEVEL ESP_LOG_VERBOSE
#include "esp_log.h"
```

At component scope, define it in the component makefile:

```
CFLAGS += -D LOG_LOCAL_LEVEL=ESP_LOG_DEBUG
```

To configure logging output per module at runtime, add calls to the function `esp_log_level_set()` as follows:

```
esp_log_level_set("*", ESP_LOG_ERROR);           // set all components to ERROR level
esp_log_level_set("wifi", ESP_LOG_WARN);        // enable WARN logs from WiFi stack
esp_log_level_set("dhcpc", ESP_LOG_INFO);       // enable INFO logs from DHCP client
```

**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 [记录日志到主机](#).

## 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

- [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.

Note that this function can not raise log level above the level set using `CONFIG_LOG_DEFAULT_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.



**Parameters**

- `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.

*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.

**Return** `func` old Function used for output.

**Parameters**

- `func`: new Function used for output. Must have same signature as `vprintf`.

`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.

**Return** timestamp, in milliseconds

`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.

**Return** timestamp, in milliseconds

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.

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.

See `esp_log_write()`

**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.

**Parameters**

- `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.

**Parameters**

- `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:

```

W (195) log_example: 0x3ffb4280  45 53 50 33 32 20 69 73  20 67 72 65 61 74
↪2c 20 |ESP32 is great, |
W (195) log_example: 0x3ffb4290  77 6f 72 6b 69 6e 67 20  61 6c 6f 6e 67 20
↪77 69 |working along wi|
W (205) log_example: 0x3ffb42a0  74 68 20 74 68 65 20 49  44 46 2e 00
↪      |th the IDF..|

```

It is highly recommend to use terminals with over 102 text width.

#### Parameters

- `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.

See `esp_log_buffer_hex_level`

#### Parameters

- `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.

See `esp_log_buffer_char_level`

#### Parameters

- `tag`: description tag
- `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.

See `printf,ESP_LOGE`

**ESP\_EARLY\_LOGW** (`tag`, `format`, ...)

macro to output logs in startup code at `ESP_LOG_WARN` level.

See `ESP_EARLY_LOGE,ESP_LOGE,printf`

**ESP\_EARLY\_LOGI** (`tag`, `format`, ...)

macro to output logs in startup code at `ESP_LOG_INFO` level.

See `ESP_EARLY_LOGE,ESP_LOGE,printf`

**ESP\_EARLY\_LOGD** (`tag`, `format`, ...)

macro to output logs in startup code at `ESP_LOG_DEBUG` level.

See `ESP_EARLY_LOGE,ESP_LOGE,printf`

**ESP\_EARLY\_LOGV** (`tag`, `format`, ...)

macro to output logs in startup code at `ESP_LOG_VERBOSE` level.

See `ESP_EARLY_LOGE,ESP_LOGE,printf`

**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.

See `printf`

**Parameters**

- `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`.

See `printf`, `ESP_LOG_LEVEL`

### Type Definitions

**typedef** int (**\*vprintf\_like\_t**) (const char \*, va\_list)

### Enumerations

**enum** `esp_log_level_t`

Log level.

*Values:*

**ESP\_LOG\_NONE**

No log output

**ESP\_LOG\_ERROR**

Critical errors, software module can not recover on its own

**ESP\_LOG\_WARN**

Error conditions from which recovery measures have been taken

**ESP\_LOG\_INFO**

Information messages which describe normal flow of events

**ESP\_LOG\_DEBUG**

Extra information which is not necessary for normal use (values, pointers, sizes, etc).

**ESP\_LOG\_VERBOSE**

Bigger chunks of debugging information, or frequent messages which can potentially flood the output.

## 2.7.12 Event Loop Library

### Overview

The event loop library allows components to declare events to which other components can register handlers –code which 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 wifi 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 WiFi, 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:

```
// 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.
```

(下页继续)

```

// 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)
```

**注解:** In IDF, the base identifiers for system events are uppercase and are postfixed with `_EVENT`. For example, the base for wifi 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_EVENT_DEFINE_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 (WiFi 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.

User Event Loops	Default Event Loops
<a href="#"><code>esp_event_loop_create()</code></a>	<a href="#"><code>esp_event_loop_create_default()</code></a>
<a href="#"><code>esp_event_loop_delete()</code></a>	<a href="#"><code>esp_event_loop_delete_default()</code></a>
<a href="#"><code>esp_event_handler_register_with()</code></a>	<a href="#"><code>esp_event_handler_register()</code></a>
<a href="#"><code>esp_event_handler_unregister_with()</code></a>	<a href="#"><code>esp_event_handler_unregister()</code></a>
<a href="#"><code>esp_event_post_to()</code></a>	<a href="#"><code>esp_event_post()</code></a>

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:

```
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 gets 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:

- [NMEA Parser](#) , which will decode the statements received from GPS.

## API Reference

### Header File

- [esp\\_event/include/esp\\_event.h](#)

### Functions

`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.

#### Return

- ESP\_OK: Success
- ESP\_ERR\_NO\_MEM: Cannot allocate memory for event loops list
- ESP\_FAIL: Failed to create task loop
- Others: Fail

#### Parameters

- [in] `event_loop_args`: configuration structure for the event loop to create
- [out] `event_loop`: handle to the created event loop

`esp_err_t esp_event_loop_delete (esp\_event\_loop\_handle\_t event_loop)`

Delete an existing event loop.

#### Return

- ESP\_OK: Success
- Others: Fail

#### Parameters

- [in] `event_loop`: event loop to delete

`esp_err_t esp_event_loop_create_default ()`

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

`esp_err_t esp_event_loop_delete_default ()`

Delete the default event loop.

#### Return

- ESP\_OK: Success
- Others: Fail

*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 of the allotted time waiting on the event queue. Once an event has been unqueued, 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 unqueued event, and (2) during dispatch of the unqueued 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 unqueued event.

In cases where waiting on the queue times out, ESP\_OK is returned and not ESP\_ERR\_TIMEOUT, since it is normal behavior.

**Note** encountering an unknown event that has been posted to the loop will only generate a warning, not an error.

#### Return

- ESP\_OK: Success
- Others: Fail

#### Parameters

- [in] *event\_loop*: event loop to dispatch posted events from
- [in] *ticks\_to\_run*: number of ticks to run the loop

*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.

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.

**Note** 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

#### 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
- Others: Fail

#### Parameters

- [in] *event\_base*: the base id of the event to register the handler for
- [in] *event\_id*: the id of the event to register the handler for
- [in] *event\_handler*: the handler function which gets called when the event is dispatched
- [in] *event\_handler\_arg*: data, aside from event data, that is passed to the handler when it is called

*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.



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.

**Note** 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

**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
- Others: Fail

**Parameters**

- `[in] event_loop`: the event loop to register this handler function to
- `[in] event_base`: the base id of the event to register the handler for
- `[in] event_id`: the id of the event to register the handler for
- `[in] event_handler`: the handler function which gets called when the event is dispatched
- `[in] event_handler_arg`: data, aside from event data, that is passed to the handler when it is called

`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.

This function can be used to unregister a handler so that it no longer gets called during dispatch. Handlers can be unregistered 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`

This function ignores unregistration of handlers that has not been previously registered.

**Return** `ESP_OK` success

**Return** `ESP_ERR_INVALID_ARG` invalid combination of event base and event id

**Return** others fail

**Parameters**

- `[in] event_base`: the base of the event with which to unregister the handler
- `[in] event_id`: the id of the event with which to unregister the handler
- `[in] event_handler`: the handler to unregister

`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 with the system event loop.

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.

**Return**

- `ESP_OK`: Success
- `ESP_ERR_INVALID_ARG`: Invalid combination of event base and event id
- Others: Fail

**Parameters**

- `[in] event_loop`: the event loop with which to unregister this handler function
- `[in] event_base`: the base of the event with which to unregister the handler
- `[in] event_id`: the id of the event with which to unregister the handler
- `[in] event_handler`: the handler to unregister

`esp_err_t esp_event_post` (`esp_event_base_t event_base`, `int32_t event_id`, `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.

**Return**

- 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

**Parameters**

- [in] event\_base: the event base that identifies the event
- [in] event\_id: the event id that identifies the event
- [in] event\_data: the data, specific to the event occurrence, that gets passed to the handler
- [in] event\_data\_size: the size of the event data
- [in] ticks\_to\_wait: number of ticks to block on a full event queue

*esp\_err\_t* **esp\_event\_post\_to**(*esp\_event\_loop\_handle\_t* event\_loop, *esp\_event\_base\_t* event\_base, int32\_t event\_id, 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.

**Return**

- 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

**Parameters**

- [in] event\_loop: the event loop to post to
- [in] event\_base: the event base that identifies the event
- [in] event\_id: the event id that identifies the event
- [in] event\_data: the data, specific to the event occurrence, that gets passed to the handler
- [in] event\_data\_size: the size of the event data
- [in] ticks\_to\_wait: number of ticks to block on a full event queue

*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_recieved dr:total_dropped
  where:
    address - memory address of the event loop
    name - name of the event loop, 'none' if no dedicated task
    total_recieved - number of successfully posted events
    total_dropped - number of events unsuccessfully posted due to queue.
    ↳being full

```

(下页继续)

```

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

```

**Note** this function is a noop when CONFIG\_ESP\_EVENT\_LOOP\_PROFILING is disabled

#### Return

- ESP\_OK: Success
- ESP\_ERR\_NO\_MEM: Cannot allocate memory for event loops list
- Others: Fail

#### Parameters

- [in] file: the file stream to output to

### 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

- [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 const char \*esp\_event\_base\_t**

unique pointer to a subsystem that exposes events

**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
```

## Related Documents

### Legacy event loop

## API Reference

### Header File

- [esp\\_event/include/esp\\_event\\_legacy.h](#)

### Functions

*esp\_err\_t* **esp\_event\_send** (*system\_event\_t* \*event)

Send a event to event task.

Other task/modules, such as the tcpip\_adapter, can call this API to send an event to event task

**Note** This API is part of the legacy event system. New code should use event library API in esp\_event.h

**Return** ESP\_OK : succeed

**Return** others : fail

#### Parameters

- event: Event to send

*esp\_err\_t* **esp\_event\_process\_default** (*system\_event\_t* \*event)

Default event handler for system events.

This function performs default handling of system events. When using esp\_event\_loop APIs, it is called automatically before invoking the user-provided callback function.

**Note** This API is part of the legacy event system. New code should use event library API in esp\_event.h

Applications which implement a custom event loop must call this function as part of event processing.

**Return** ESP\_OK if an event was handled successfully

#### Parameters

- event: pointer to event to be handled

void **esp\_event\_set\_default\_eth\_handlers** ()

Install default event handlers for Ethernet interface.

**Note** This API is part of the legacy event system. New code should use event library API in esp\_event.h

void **esp\_event\_set\_default\_wifi\_handlers** ()

Install default event handlers for Wi-Fi interfaces (station and AP)

**Note** This API is part of the legacy event system. New code should use event library API in esp\_event.h

*esp\_err\_t* **esp\_event\_loop\_init** (*system\_event\_cb\_t* cb, void \*ctx)

Initialize event loop.

Create the event handler and task

**Note** This API is part of the legacy event system. New code should use event library API in esp\_event.h

#### Return

- ESP\_OK: succeed
- others: fail

#### Parameters

- cb: application specified event callback, it can be modified by call esp\_event\_set\_cb
- ctx: reserved for user

*system\_event\_cb\_t* **esp\_event\_loop\_set\_cb** (*system\_event\_cb\_t* cb, void \*ctx)

Set application specified event callback function.

**Note** This API is part of the legacy event system. New code should use event library API in esp\_event.h

**Attention** 1. If cb is NULL, means application don't need to handle. If cb is not NULL, it will be call when an event is received, after the default event callback is completed

**Return** old callback

**Parameters**

- cb: application callback function
- ctx: argument to be passed to callback

## Unions

**union system\_event\_info\_t**

#include <esp\_event\_legacy.h> Union of all possible system\_event argument structures

## Public Members

*system\_event\_sta\_connected\_t* **connected**

ESP32 station connected to AP

*system\_event\_sta\_disconnected\_t* **disconnected**

ESP32 station disconnected to AP

*system\_event\_sta\_scan\_done\_t* **scan\_done**

ESP32 station scan (APs) done

*system\_event\_sta\_authmode\_change\_t* **auth\_change**

the auth mode of AP ESP32 station connected to changed

*system\_event\_sta\_got\_ip\_t* **got\_ip**

ESP32 station got IP, first time got IP or when IP is changed

*system\_event\_sta\_wps\_er\_pin\_t* **sta\_er\_pin**

ESP32 station WPS enrollee mode PIN code received

*system\_event\_sta\_wps\_fail\_reason\_t* **sta\_er\_fail\_reason**

ESP32 station WPS enrollee mode failed reason code received

*system\_event\_sta\_wps\_er\_success\_t* **sta\_er\_success**

ESP32 station WPS enrollee success

*system\_event\_ap\_staconnected\_t* **sta\_connected**

a station connected to ESP32 soft-AP

*system\_event\_ap\_stadisconnected\_t* **sta\_disconnected**

a station disconnected to ESP32 soft-AP

*system\_event\_ap\_probe\_req\_rx\_t* **ap\_probereqrecved**

ESP32 soft-AP receive probe request packet

*system\_event\_ap\_staipassigned\_t* **ap\_staipassigned**

ESP32 soft-AP assign an IP to the station

*system\_event\_got\_ip6\_t* **got\_ip6**

ESP32 station or ap or ethernet ipv6 addr state change to preferred

## Structures

**struct system\_event\_t**

Event, as a tagged enum

## Public Members

*system\_event\_id\_t* **event\_id**  
event ID

*system\_event\_info\_t* **event\_info**  
event information

## Macros

**SYSTEM\_EVENT\_AP\_STA\_GOT\_IP6**

## Type Definitions

**typedef** *wifi\_event\_sta\_wps\_fail\_reason\_t* **system\_event\_sta\_wps\_fail\_reason\_t**

Argument structure of SYSTEM\_EVENT\_STA\_WPS\_ER\_FAILED event

**typedef** *wifi\_event\_sta\_scan\_done\_t* **system\_event\_sta\_scan\_done\_t**

Argument structure of SYSTEM\_EVENT\_SCAN\_DONE event

**typedef** *wifi\_event\_sta\_connected\_t* **system\_event\_sta\_connected\_t**

Argument structure of SYSTEM\_EVENT\_STA\_CONNECTED event

**typedef** *wifi\_event\_sta\_disconnected\_t* **system\_event\_sta\_disconnected\_t**

Argument structure of SYSTEM\_EVENT\_STA\_DISCONNECTED event

**typedef** *wifi\_event\_sta\_authmode\_change\_t* **system\_event\_sta\_authmode\_change\_t**

Argument structure of SYSTEM\_EVENT\_STA\_AUTHMODE\_CHANGE event

**typedef** *wifi\_event\_sta\_wps\_er\_pin\_t* **system\_event\_sta\_wps\_er\_pin\_t**

Argument structure of SYSTEM\_EVENT\_STA\_WPS\_ER\_PIN event

**typedef** *wifi\_event\_sta\_wps\_er\_success\_t* **system\_event\_sta\_wps\_er\_success\_t**

Argument structure of SYSTEM\_EVENT\_STA\_WPS\_ER\_PIN event

**typedef** *wifi\_event\_ap\_staconnected\_t* **system\_event\_ap\_staconnected\_t**

Argument structure of event

**typedef** *wifi\_event\_ap\_stadisconnected\_t* **system\_event\_ap\_stadisconnected\_t**

Argument structure of event

**typedef** *wifi\_event\_ap\_probe\_req\_rx\_t* **system\_event\_ap\_probe\_req\_rx\_t**

Argument structure of event

**typedef** *ip\_event\_ap\_staipassigned\_t* **system\_event\_ap\_staipassigned\_t**

Argument structure of event

**typedef** *ip\_event\_got\_ip\_t* **system\_event\_sta\_got\_ip\_t**

Argument structure of event

**typedef** *ip\_event\_got\_ip6\_t* **system\_event\_got\_ip6\_t**

Argument structure of event

**typedef** *esp\_err\_t* (\***system\_event\_handler\_t**) (*system\_event\_t* \*event)

Event handler function type

**typedef** *esp\_err\_t* (\***system\_event\_cb\_t**) (void \*ctx, *system\_event\_t* \*event)

Application specified event callback function.

**Note** This API is part of the legacy event system. New code should use event library API in esp\_event.h

### Return

- ESP\_OK: succeed
- others: fail

### Parameters

- ctx: reserved for user
- event: event type defined in this file

**Enumerations****enum system\_event\_id\_t**

System event types enumeration

*Values:***SYSTEM\_EVENT\_WIFI\_READY = 0**

ESP32 WiFi ready

**SYSTEM\_EVENT\_SCAN\_DONE**

ESP32 finish scanning AP

**SYSTEM\_EVENT\_STA\_START**

ESP32 station start

**SYSTEM\_EVENT\_STA\_STOP**

ESP32 station stop

**SYSTEM\_EVENT\_STA\_CONNECTED**

ESP32 station connected to AP

**SYSTEM\_EVENT\_STA\_DISCONNECTED**

ESP32 station disconnected from AP

**SYSTEM\_EVENT\_STA\_AUTHMODE\_CHANGE**

the auth mode of AP connected by ESP32 station changed

**SYSTEM\_EVENT\_STA\_GOT\_IP**

ESP32 station got IP from connected AP

**SYSTEM\_EVENT\_STA\_LOST\_IP**

ESP32 station lost IP and the IP is reset to 0

**SYSTEM\_EVENT\_STA\_WPS\_ER\_SUCCESS**

ESP32 station wps succeeds in enrollee mode

**SYSTEM\_EVENT\_STA\_WPS\_ER\_FAILED**

ESP32 station wps fails in enrollee mode

**SYSTEM\_EVENT\_STA\_WPS\_ER\_TIMEOUT**

ESP32 station wps timeout in enrollee mode

**SYSTEM\_EVENT\_STA\_WPS\_ER\_PIN**

ESP32 station wps pin code in enrollee mode

**SYSTEM\_EVENT\_STA\_WPS\_ER\_PBC\_OVERLAP**

ESP32 station wps overlap in enrollee mode

**SYSTEM\_EVENT\_AP\_START**

ESP32 soft-AP start

**SYSTEM\_EVENT\_AP\_STOP**

ESP32 soft-AP stop

**SYSTEM\_EVENT\_AP\_STACONNECTED**

a station connected to ESP32 soft-AP

**SYSTEM\_EVENT\_AP\_STADISCONNECTED**

a station disconnected from ESP32 soft-AP

**SYSTEM\_EVENT\_AP\_STAIPASSIGNED**

ESP32 soft-AP assign an IP to a connected station

**SYSTEM\_EVENT\_AP\_PROBEREQRCVD**

Receive probe request packet in soft-AP interface

**SYSTEM\_EVENT\_STA\_BEACON\_TIMEOUT**

ESP32 station beacon timeout

<b>SYSTEM_EVENT_GOT_IP6</b>	ESP32 station or ap or ethernet interface v6IP addr is preferred
<b>SYSTEM_EVENT_ETH_START</b>	ESP32 ethernet start
<b>SYSTEM_EVENT_ETH_STOP</b>	ESP32 ethernet stop
<b>SYSTEM_EVENT_ETH_CONNECTED</b>	ESP32 ethernet phy link up
<b>SYSTEM_EVENT_ETH_DISCONNECTED</b>	ESP32 ethernet phy link down
<b>SYSTEM_EVENT_ETH_GOT_IP</b>	ESP32 ethernet got IP from connected AP
<b>SYSTEM_EVENT_MAX</b>	Number of members in this enum

## 2.7.13 Application Level Tracing

### Overview

IDF provides useful feature for program behaviour analysis: application level tracing. It is implemented in the corresponding library and can be enabled via menuconfig. This feature allows to transfer arbitrary data between host and ESP32 via JTAG interface with small 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

- `app_trace/include/esp_app_trace.h`

#### Functions

`esp_err_t esp_apptrace_init ()`

Initializes application tracing module.

**Note** Should be called before any `esp_apptrace_xxx` call.

**Return** ESP\_OK on success, otherwise see `esp_err_t`

void `esp_apptrace_down_buffer_config (uint8_t *buf, uint32_t size)`

Configures down buffer.

**Note** Needs to be called before initiating any data transfer using `esp_apptrace_buffer_get` and `esp_apptrace_write`. This function does not protect internal data by lock.

#### Parameters

- `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. After data in buffer are ready to be sent off `esp_apptrace_buffer_put` must be called to indicate it.

**Return** non-NULL on success, otherwise NULL.

#### Parameters



- `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.

*esp\_err\_t* **esp\_appttrace\_buffer\_put** (*esp\_appttrace\_dest\_t* *dest*, *uint8\_t* \**ptr*, *uint32\_t* *tmo*)

Indicates that the data in buffer are ready to be sent off. This function is a counterpart of and must be preceded by `esp_appttrace_buffer_get`.

**Return** `ESP_OK` on success, otherwise see `esp_err_t`

**Parameters**

- `dest`: Indicates HW interface to send data. Should be identical to the same parameter in call to `esp_appttrace_buffer_get`.
- `ptr`: Address of trace buffer to release. Should be the value returned by call to `esp_appttrace_buffer_get`.
- `tmo`: Timeout for operation (in us). Use `ESP_APPTRACE_TMO_INFINITE` to wait indefinitely.

*esp\_err\_t* **esp\_appttrace\_write** (*esp\_appttrace\_dest\_t* *dest*, **const** *void* \**data*, *uint32\_t* *size*, *uint32\_t* *tmo*)

Writes data to trace buffer.

**Return** `ESP_OK` on success, otherwise see `esp_err_t`

**Parameters**

- `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.

**int** **esp\_appttrace\_vprintf\_to** (*esp\_appttrace\_dest\_t* *dest*, *uint32\_t* *tmo*, **const** *char* \**fmt*, *va\_list* *ap*)  
vprintf-like function to sent log messages to host via specified HW interface.

**Return** Number of bytes written.

**Parameters**

- `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.

**int** **esp\_appttrace\_vprintf** (**const** *char* \**fmt*, *va\_list* *ap*)  
vprintf-like function to sent log messages to host.

**Return** Number of bytes written.

**Parameters**

- `fmt`: Address of format string.
- `ap`: List of arguments.

*esp\_err\_t* **esp\_appttrace\_flush** (*esp\_appttrace\_dest\_t* *dest*, *uint32\_t* *tmo*)

Flushes remaining data in trace buffer to host.

**Return** `ESP_OK` on success, otherwise see `esp_err_t`

**Parameters**

- `dest`: Indicates HW interface to flush data on.
- `tmo`: Timeout for operation (in us). Use `ESP_APPTRACE_TMO_INFINITE` to wait indefinitely.

*esp\_err\_t* **esp\_appttrace\_flush\_nolock** (*esp\_appttrace\_dest\_t* *dest*, *uint32\_t* *min\_sz*, *uint32\_t* *tmo*)

Flushes remaining data in trace buffer to host without locking internal data. This is special version of `esp_appttrace_flush` which should be called from panic handler.

**Return** `ESP_OK` on success, otherwise see `esp_err_t`

**Parameters**

- `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.

*esp\_err\_t* **esp\_appttrace\_read** (*esp\_appttrace\_dest\_t* dest, void \*data, uint32\_t \*size, uint32\_t tmo)

Reads host data from trace buffer.

**Return** ESP\_OK on success, otherwise see esp\_err\_t

**Parameters**

- 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\_APPTTRACE\_TMO\_INFINITE to wait indefinitely.

uint8\_t \***esp\_appttrace\_down\_buffer\_get** (*esp\_appttrace\_dest\_t* dest, uint32\_t \*size, uint32\_t tmo)

Retrieves incoming data buffer if any. After data in buffer are processed esp\_appttrace\_down\_buffer\_put must be called to indicate it.

**Return** non-NULL on success, otherwise NULL.

**Parameters**

- 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\_APPTTRACE\_TMO\_INFINITE to wait indefinitely.

*esp\_err\_t* **esp\_appttrace\_down\_buffer\_put** (*esp\_appttrace\_dest\_t* dest, uint8\_t \*ptr, uint32\_t tmo)

Indicates that the data in down buffer are processed. This function is a counterpart of and must be preceded by esp\_appttrace\_down\_buffer\_get.

**Return** ESP\_OK on success, otherwise see esp\_err\_t

**Parameters**

- dest: Indicates HW interface to receive data. Should be identical to the same parameter in call to esp\_appttrace\_down\_buffer\_get.
- ptr: Address of trace buffer to release. Should be the value returned by call to esp\_appttrace\_down\_buffer\_get.
- tmo: Timeout for operation (in us). Use ESP\_APPTTRACE\_TMO\_INFINITE to wait indefinitely.

bool **esp\_appttrace\_host\_is\_connected** (*esp\_appttrace\_dest\_t* dest)

Checks whether host is connected.

**Return** true if host is connected, otherwise false

**Parameters**

- dest: Indicates HW interface to use.

void \***esp\_appttrace\_fopen** (*esp\_appttrace\_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.

**Return** non zero file handle on success, otherwise 0

**Parameters**

- dest: Indicates HW interface to use.
- path: Path to file.
- mode: Mode string. See fopen for details.

int **esp\_appttrace\_fclose** (*esp\_appttrace\_dest\_t* dest, void \*stream)

Closes file on host. This function has the same semantic as 'fclose' except for the first argument.

**Return** Zero on success, otherwise non-zero. See fclose for details.

**Parameters**

- dest: Indicates HW interface to use.
- stream: File handle returned by esp\_appttrace\_fopen.

size\_t **esp\_appttrace\_fwrite** (*esp\_appttrace\_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.

**Return** Number of written items. See fwrite for details.

**Parameters**

- 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`.

size\_t **esp\_apprtrace\_fread** (*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.

**Return** Number of read items. See `fread` for details.

**Parameters**

- *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`.

int **esp\_apprtrace\_fseek** (*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.

**Return** Zero on success, otherwise non-zero. See `fseek` for details.

**Parameters**

- *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.

int **esp\_apprtrace\_ftell** (*esp\_apprtrace\_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.

**Return** Current position in file. See `ftell` for details.

**Parameters**

- *dest*: Indicates HW interface to use.
- *stream*: File handle returned by `esp_apprtrace_fopen`.

int **esp\_apprtrace\_fstop** (*esp\_apprtrace\_dest\_t* *dest*)

Indicates to the host that all file operations are completed. 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.).

**Return** `ESP_OK` on success, otherwise see `esp_err_t`

**Parameters**

- *dest*: Indicates HW interface to use.

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\_apprtrace\_dest\_t**

Application trace data destinations bits.

*Values:*

`ESP_APPTRACE_DEST_TRAX` = 0x1  
JTAG destination.

`ESP_APPTRACE_DEST_UART0` = 0x2  
UART destination.

## Header File

- [app\\_trace/include/esp\\_sysview\\_trace.h](#)

**Functions**

**static** *esp\_err\_t* **esp\_sysview\_flush** (uint32\_t *tmo*)

Flushes remaining data in SystemView trace buffer to host.

**Return** ESP\_OK.

**Parameters**

- *tmo*: Timeout for operation (in us). Use ESP\_APPTRACE\_TMO\_INFINITE to wait indefinitely.

int **esp\_sysview\_vprintf** (const char \**format*, va\_list *args*)

vprintf-like function to sent log messages to the host.

**Return** Number of bytes written.

**Parameters**

- *format*: Address of format string.
- *args*: List of arguments.

*esp\_err\_t* **esp\_sysview\_heap\_trace\_start** (uint32\_t *tmo*)

Starts SystemView heap tracing.

**Return** ESP\_OK on success, ESP\_ERR\_TIMEOUT if operation has been timed out.

**Parameters**

- *tmo*: Timeout (in us) to wait for the host to be connected. Use -1 to wait forever.

*esp\_err\_t* **esp\_sysview\_heap\_trace\_stop** (void)

Stops SystemView heap tracing.

**Return** ESP\_OK.

void **esp\_sysview\_heap\_trace\_alloc** (void \**addr*, uint32\_t *size*, const void \**callers*)

Sends heap allocation event to the host.

**Parameters**

- *addr*: Address of allocated block.
- *size*: Size of allocated block.
- *callers*: Pointer to array with callstack addresses. Array size must be CONFIG\_HEAP\_TRACING\_STACK\_DEPTH.

void **esp\_sysview\_heap\_trace\_free** (void \**addr*, const void \**callers*)

Sends heap de-allocation event to the host.

**Parameters**

- *addr*: Address of de-allocated block.
- *callers*: Pointer to array with callstack addresses. Array size must be CONFIG\_HEAP\_TRACING\_STACK\_DEPTH.

## 2.7.14 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、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。

因为请求较高的 APB 频率或 CPU 频率，以及禁用 Light-sleep 模式会增加功耗，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，且在中断入口将频率由 40 MHz 调节至 80 MHz，则最大额外延迟为 40 us。

应用程序可以通过调用 `esp_pm_configure()` 函数启用动态调频 (DFS) 功能和自动 Light-sleep 模式。此函数的参数为 `esp_pm_config_esp32_t`，定义了频率调节的相关设置。在此参数结构中，需要初始化下面三个字段：

- `max_freq_mhz`：最大 CPU 频率 (MHz)，即获取 `ESP_PM_CPU_FREQ_MAX` 锁后所使用的频率。该字段通常设置为 `CONFIG_ESP32_DEFAULT_CPU_FREQ_MHZ`。
- `min_freq_mhz`：最小 CPU 频率 (MHz)，即仅获取 `ESP_PM_APB_FREQ_MAX` 锁后所使用的频率。该字段可设置为晶振 (XTAL) 频率值，或者 XTAL 频率值除以整数。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)。

或者，如果在 `menuconfig` 中启用了 `CONFIG_PM_DFS_INIT_AUTO` 选项，最大 CPU 频率将由 `CONFIG_ESP32_DEFAULT_CPU_FREQ_MHZ` 设置决定，最小 CPU 频率将锁定为 XTAL 频率。

### 注解：

1. 自动 Light-sleep 模式基于 FreeRTOS Tickless Idle 功能，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。
2. 在 Light-sleep 状态下，外设设有时钟门控，不会产生来自 GPIO 和内部外设的中断。[Sleep Modes](#) 文档中所提到的唤醒源可用于从 Light-sleep 状态触发唤醒。例如，EXT0 和 EXT1 唤醒源就可以通过 GPIO 唤醒芯片。

## 电源管理锁

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 支持下表中所述的三种电源管理锁。

电源管理锁	描述
<code>ESP_PM_CPU_FREQ</code>	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。
<code>ESP_PM_APB_FREQ</code>	请求将 APB 频率设置为最大值，ESP32 支持的最大频率为 80 MHz。
<code>ESP_PM_NO_LIGHT_SLEEP</code>	禁止自动切换至 Light-sleep 模式。

## 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_ESP32_DEFAULT_CPU_FREQ_MHZ` 指定 CPU 最大频率。

CPU 最高频率	电源管理锁获取情况	CPU 频率和 APB 频率
240 MHz	获取 ESP_PM_CPU_FREQ_MAX 或 ESP_PM_APB_FREQ_MAX	CPU: 240 MHz APB: 80 MHz
	无	使用 <code>esp_pm_configure()</code> 为二者设置最小值
160 MHz	获取 ESP_PM_CPU_FREQ_MAX	CPU: 160 MHz APB: 80 MHz
	获取 ESP_PM_CPU_FREQ_MAX, 未获取 ESP_PM_APB_FREQ_MAX	CPU: 80 MHz APB: 80 MHz
	无	使用 <code>esp_pm_configure()</code> 为二者设置最小值
80 MHz	获取 ESP_PM_CPU_FREQ_MAX 或 ESP_PM_APB_FREQ_MAX	CPU: 80 MHz APB: 80 MHz
	无	使用 <code>esp_pm_configure()</code> 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。

### 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題。例如，Timer Group 外设定时会继续计数，但定时器计数的速度将随 APB 频率的变更而变更。

下面的外设不受 APB 频率变更的影响：

- **UART**：如果 REF\_TICK 用作时钟源，则 UART 不受 APB 频率变更影响。请查看 `uart_config_t` 中的 `use_ref_tick`。
- **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 用作时钟源，则 RMT 不受 APB 频率变更影响。此驱动程序尚不支持 REF\_TICK，但可以清除相应通道的 RMT\_REF\_ALWAYS\_ON\_CHx 位来启用该功能。

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 锁：

- 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。
- **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。
- **CAN**：从调用 `can_driver_install()` 至 `can_driver_uninstall()` 期间。

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：

- MCPWM
- PCNT

- Sigma-delta
- Timer Group

## API 参考

### Header File

- [esp\\_common/include/esp\\_pm.h](#)

### Functions

*esp\_err\_t* **esp\_pm\_configure** (**const** void \**config*)

Set implementation-specific power management configuration.

#### Return

- 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

#### Parameters

- *config*: pointer to implementation-specific configuration structure (e.g. `esp_pm_config_esp32`)

*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.

#### Return

- ESP\_OK on success
- ESP\_ERR\_NO\_MEM if the lock structure can not be allocated
- ESP\_ERR\_INVALID\_ARG if *out\_handle* is NULL or type argument is not valid
- ESP\_ERR\_NOT\_SUPPORTED if CONFIG\_PM\_ENABLE is not enabled in sdkconfig

#### Parameters

- *lock\_type*: Power management constraint which the lock should control
- *arg*: argument, value depends on *lock\_type*, see `esp_pm_lock_type_t`
- *name*: arbitrary string identifying the lock (e.g. “wifi” or “spi”). Used by the `esp_pm_dump_locks` function to list existing locks. May be set to NULL. If not set to NULL, must point to a string which is valid for the lifetime of the lock.
- [*out*] *out\_handle*: handle returned from this function. Use this handle when calling `esp_pm_lock_delete`, `esp_pm_lock_acquire`, `esp_pm_lock_release`. Must not be NULL.

*esp\_err\_t* **esp\_pm\_lock\_acquire** (*esp\_pm\_lock\_handle\_t* *handle*)

Take a power management lock.

Once the lock is taken, power management algorithm will not switch to the mode specified in a call to `esp_pm_lock_create`, or any of the lower power modes (higher numeric values of ‘mode’).

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.

#### 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

#### Parameters

- *handle*: handle obtained from `esp_pm_lock_create` function

*esp\_err\_t* **esp\_pm\_lock\_release** (*esp\_pm\_lock\_handle\_t* handle)

Release the lock taken using `esp_pm_lock_acquire`.

Call to this functions removes power management restrictions placed when taking the lock.

Locks are recursive, so 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 actually 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.

#### Return

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG if the handle is invalid
- ESP\_ERR\_INVALID\_STATE if lock is not acquired
- ESP\_ERR\_NOT\_SUPPORTED if CONFIG\_PM\_ENABLE is not enabled in sdkconfig

#### Parameters

- handle: handle obtained from `esp_pm_lock_create` function

*esp\_err\_t* **esp\_pm\_lock\_delete** (*esp\_pm\_lock\_handle\_t* handle)

Delete a lock created using `esp_pm_lock`.

The lock must be released before calling this function.

This function must not be called from an ISR.

#### 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

#### Parameters

- handle: handle obtained from `esp_pm_lock_create` function

*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.

#### Return

- ESP\_OK on success
- ESP\_ERR\_NOT\_SUPPORTED if CONFIG\_PM\_ENABLE is not enabled in sdkconfig

#### Parameters

- 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.

### 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:

```
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.



**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.

**ESP\_PM\_NO\_LIGHT\_SLEEP**

Prevent the system from going into light sleep. Argument is unused and should be set to 0.

**Header File**

- [esp32/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

## 2.7.15 Sleep Modes

**Overview**

ESP32 is capable of light sleep and deep sleep power saving modes.

In light sleep mode, digital peripherals, most of the RAM, and CPUs are clock-gated, and supply voltage is reduced. Upon exit from light sleep, peripherals and CPUs resume operation, their internal state is preserved.

In deep sleep mode, CPUs, most of the RAM, and all the digital peripherals which are clocked from APB\_CLK are powered off. The only parts of the chip which can still be powered on are: RTC controller, RTC peripherals (including ULP coprocessor), and RTC memories (slow and fast).

Wakeup from deep and light sleep modes can be done using several sources. These sources can be combined, in this case the chip will wake up when any one of the sources is triggered. Wakeup sources can be enabled using `esp_sleep_enable_x_wakeup` APIs and can be disabled using `esp_sleep_disable_wakeup_source()` API. Next section describes these APIs in detail. Wakeup sources can be configured at any moment before entering light or deep sleep mode.

Additionally, the application can force specific powerdown modes for the RTC peripherals and RTC memories using `esp_sleep_pd_config()` API.

Once wakeup sources are configured, application can enter sleep mode using `esp_light_sleep_start()` or `esp_deep_sleep_start()` APIs. At this point the hardware will be configured according to the requested wakeup sources, and RTC controller will either power down or power off the CPUs and digital peripherals.

**WiFi/BT and sleep modes**

In deep sleep and light sleep modes, wireless peripherals are powered down. Before entering deep sleep or light sleep modes, applications must disable WiFi and BT using appropriate calls (`esp_bluedroid_disable()`),

`esp_bt_controller_disable()`, `esp_wifi_stop()`). WiFi and BT connections will not be maintained in deep sleep or light sleep, even if these functions are not called.

If WiFi connection needs to be maintained, enable WiFi modem sleep, and enable automatic light sleep feature (see [Power Management APIs](#)). This will allow the system to wake up from sleep automatically when required by WiFi driver, thereby maintaining connection to the AP.

### Wakeup sources

**Timer** RTC controller has a built in timer which can be used to wake up the chip after a predefined amount of time. Time is specified at microsecond precision, but the actual resolution depends on the clock source selected for RTC SLOW\_CLK. See chapter “Reset and Clock” of the ESP32 Technical Reference Manual for details about RTC clock options.

This wakeup mode doesn't require RTC peripherals or RTC memories to be powered on during sleep.

`esp_sleep_enable_timer_wakeup()` function can be used to enable deep sleep wakeup using a timer.

**Touch pad** RTC IO module contains logic to trigger wakeup when a touch sensor interrupt occurs. You need to configure the touch pad interrupt before the chip starts deep sleep.

Revisions 0 and 1 of the ESP32 only support this wakeup mode when RTC peripherals are not forced to be powered on (i.e. `ESP_PD_DOMAIN_RTC_PERIPH` should be set to `ESP_PD_OPTION_AUTO`).

`esp_sleep_enable_touchpad_wakeup()` function can be used to enable this wakeup source.

**External wakeup (ext0)** RTC IO module contains logic to trigger wakeup when one of RTC GPIOs is set to a predefined logic level. RTC IO is part of RTC peripherals power domain, so RTC peripherals will be kept powered on during deep sleep if this wakeup source is requested.

Because RTC IO module is enabled in this mode, internal pullup or pulldown resistors can also be used. They need to be configured by the application using `rtc_gpio_pullup_en()` and `rtc_gpio_pulldown_en()` functions, before calling `esp_sleep_start()`.

In revisions 0 and 1 of the ESP32, this wakeup source is incompatible with ULP and touch wakeup sources.

`esp_sleep_enable_ext0_wakeup()` function can be used to enable this wakeup source.

**警告:** After wake up from sleep, IO pad used for wakeup will be configured as RTC IO. Before using this pad as digital GPIO, reconfigure it using `rtc_gpio_deinit(gpio_num)` function.

**External wakeup (ext1)** RTC controller contains logic to trigger wakeup using multiple RTC GPIOs. One of the two logic functions can be used to trigger wakeup:

- wake up if any of the selected pins is high (`ESP_EXT1_WAKEUP_ANY_HIGH`)
- wake up if all the selected pins are low (`ESP_EXT1_WAKEUP_ALL_LOW`)

This wakeup source is implemented by the RTC controller. As such, RTC peripherals and RTC memories can be powered down in this mode. However, if RTC peripherals are powered down, internal pullup and pulldown resistors will be disabled. To use internal pullup or pulldown resistors, request RTC peripherals power domain to be kept on during sleep, and configure pullup/pulldown resistors using `rtc_gpio_` functions, before entering sleep:

```
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:** After wake up from sleep, IO pad(s) used for wakeup will be configured as RTC IO. Before using these pads as digital GPIOs, reconfigure them using `rtc_gpio_deinit(gpio_num)` function.

`esp_sleep_enable_ext1_wakeup()` function can be used to enable this wakeup source.

**ULP coprocessor wakeup** ULP coprocessor can run while the chip is in sleep mode, and may be used to poll sensors, monitor ADC or touch sensor values, and wake up the chip when a specific event is detected. ULP coprocessor is part of RTC peripherals power domain, and it runs the program stored in RTC slow memory. RTC slow memory will be powered on during sleep if this wakeup mode is requested. RTC peripherals will be automatically powered on before ULP coprocessor starts running the program; once the program stops running, RTC peripherals are automatically powered down again.

Revisions 0 and 1 of the ESP32 only support this wakeup mode when RTC peripherals are not forced to be powered on (i.e. `ESP_PD_DOMAIN_RTC_PERIPH` should be set to `ESP_PD_OPTION_AUTO`).

`esp_sleep_enable_ulp_wakeup()` function can be used to enable this wakeup source.

**GPIO wakeup (light sleep only)** In addition to EXT0 and EXT1 wakeup sources described above, one more method of wakeup from external inputs is available in light sleep mode. With this wakeup source, each pin can be individually configured to trigger wakeup on high or low level using `gpio_wakeup_enable()` function. Unlike EXT0 and EXT1 wakeup sources, which can only be used with RTC IOs, this wakeup source can be used with any IO (RTC or digital).

`esp_sleep_enable_gpio_wakeup()` function can be used to enable this wakeup source.

**UART wakeup (light sleep only)** When ESP32 receives UART input from external devices, it is often required to wake up the chip when input data is available. UART peripheral contains a feature which allows waking up the chip from light sleep when a certain number of positive edges on RX pin are seen. This number of positive edges can be set using `uart_set_wakeup_threshold()` function. Note that the character which triggers wakeup (and any characters before it) will not be received by the UART after wakeup. This means that the external device typically needs to send an extra character to the ESP32 to trigger wakeup, before sending the data.

`esp_sleep_enable_uart_wakeup()` function can be used to enable this wakeup source.

### Power-down of RTC peripherals and memories

By default, `esp_deep_sleep_start()` and `esp_light_sleep_start()` functions will power down all RTC power domains which are not needed by the enabled wakeup sources. To override this behaviour, `esp_sleep_pd_config()` function is provided.

Note: in revision 0 of the ESP32, RTC fast memory will always be kept enabled in deep sleep, so that the deep sleep stub can run after reset. This can be overridden, if the application doesn't need clean reset behaviour after deep sleep.

If some variables in the program are placed into RTC slow memory (for example, using `RTC_DATA_ATTR` attribute), RTC slow memory will be kept powered on by default. This can be overridden using `esp_sleep_pd_config()` function, if desired.

### Entering light sleep

`esp_light_sleep_start()` function can be used to enter light sleep once wakeup sources are configured. It is also possible to go into light sleep with no wakeup sources configured, in this case the chip will be in light sleep mode indefinitely, until external reset is applied.

## Entering deep sleep

`esp_deep_sleep_start()` function can be used to enter deep sleep once wakeup sources are configured. It is also possible to go into deep sleep with no wakeup sources configured, in this case the chip will be in deep sleep mode indefinitely, until external reset is applied.

## Configuring IOs

Some ESP32 IOs have internal pullups or pulldowns, which are enabled by default. If an external circuit drives this pin in deep sleep mode, current consumption may increase due to current flowing through these pullups and pulldowns.

To isolate a pin, preventing extra current draw, call `rtc_gpio_isolate()` function.

For example, on ESP32-WROVER module, GPIO12 is pulled up externally. GPIO12 also has an internal pull-down in the ESP32 chip. This means that in deep sleep, some current will flow through these external and internal resistors, increasing deep sleep current above the minimal possible value. Add the following code before `esp_deep_sleep_start()` to remove this extra current:

```
rtc_gpio_isolate(GPIO_NUM_12);
```

## UART output handling

Before entering sleep mode, `esp_deep_sleep_start()` will flush the contents of UART FIFOs.

When entering light sleep mode using `esp_light_sleep_start()`, UART FIFOs will not be flushed. Instead, UART output will be suspended, and remaining characters in the FIFO will be sent out after wakeup from light sleep.

## Checking sleep wakeup cause

`esp_sleep_get_wakeup_cause()` function can be used to check which wakeup source has triggered wakeup from sleep mode.

For touch pad and ext1 wakeup sources, it is possible to identify pin or touch pad which has caused wakeup using `esp_sleep_get_touchpad_wakeup_status()` and `esp_sleep_get_ext1_wakeup_status()` functions.

## Disable sleep wakeup source

Previously configured wakeup source can be disabled later using `esp_sleep_disable_wakeup_source()` API. This function deactivates trigger for the given wakeup source. Additionally it can disable all triggers if the argument is `ESP_SLEEP_WAKEUP_ALL`.

## Application Example

Implementation of basic functionality of deep sleep is shown in [protocols/sntp](#) example, where ESP module is periodically waken up to retrieve time from NTP server.

More extensive example in [system/deep\\_sleep](#) illustrates usage of various deep sleep wakeup triggers and ULP co-processor programming.

## API Reference

### Header File

- `esp32/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.

**Note** This function does not modify wake up configuration in RTC. It will be performed in `esp_sleep_start` function.

**Return**

- ESP\_OK on success
- ESP\_ERR\_INVALID\_STATE if trigger was not active

**Parameters**

- source: - number of source to disable of type `esp_sleep_source_t`

*esp\_err\_t* **esp\_sleep\_enable\_ulp\_wakeup** ()

Enable wakeup by ULP coprocessor.

**Note** ULP wakeup source cannot be used when RTC\_PERIPH power domain is forced to be powered on (ESP\_PD\_OPTION\_ON) or when ext0 wakeup source is used.

**Return**

- ESP\_OK on success
- ESP\_ERR\_NOT\_SUPPORTED if additional current by touch (CONFIG\_ESP32\_RTC\_EXT\_CRYST\_ADDIT\_CURRENT) is enabled.
- ESP\_ERR\_INVALID\_STATE if ULP co-processor is not enabled or if wakeup triggers conflict

*esp\_err\_t* **esp\_sleep\_enable\_timer\_wakeup** (uint64\_t time\_in\_us)

Enable wakeup by timer.

**Return**

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG if value is out of range (TBD)

**Parameters**

- time\_in\_us: time before wakeup, in microseconds

*esp\_err\_t* **esp\_sleep\_enable\_touchpad\_wakeup** ()

Enable wakeup by touch sensor.

**Note** Touch wakeup source cannot be used when RTC\_PERIPH power domain is forced to be powered on (ESP\_PD\_OPTION\_ON) or when ext0 wakeup source is used.

**Note** The FSM mode of the touch button should be configured as the timer trigger mode.

**Return**

- ESP\_OK on success
- ESP\_ERR\_NOT\_SUPPORTED if additional current by touch (CONFIG\_ESP32\_RTC\_EXT\_CRYST\_ADDIT\_CURRENT) is enabled.
- ESP\_ERR\_INVALID\_STATE if wakeup triggers conflict

*touch\_pad\_t* **esp\_sleep\_get\_touchpad\_wakeup\_status** ()

Get the touch pad which caused wakeup.

If wakeup was caused by another source, this function will return TOUCH\_PAD\_MAX;

**Return** touch pad which caused wakeup

*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_sleep_start`, immediately before entering sleep mode.

**Note** ext0 wakeup source cannot be used together with touch or ULP wakeup sources.

**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

**Parameters**

- `gpio_num`: GPIO number used as wakeup source. Only GPIOs which have RTC functionality can be used: 0,2,4,12-15,25-27,32-39.
- `level`: input level which will trigger wakeup (0=low, 1=high)

*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_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.

**Return**

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG if any of the selected GPIOs is not an RTC GPIO, or mode is invalid

**Parameters**

- `mask`: bit mask of GPIO numbers which will cause wakeup. Only GPIOs which have RTC functionality can be used in this bit map: 0,2,4,12-15,25-27,32-39.
- `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* ()

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.

**Note** GPIO wakeup source cannot be used together with touch or ULP wakeup sources.

**Return**

- ESP\_OK on success
- ESP\_ERR\_INVALID\_STATE if wakeup triggers conflict

*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.

**Note** ESP32 does not support wakeup from UART2.

**Return**

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG if wakeup from given UART is not supported

**Parameters**

- `uart_num`: UART port to wake up from

uint64\_t **esp\_sleep\_get\_ext1\_wakeup\_status** ()

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 n 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 using this API, all power domains default to ESP\_PD\_OPTION\_AUTO.

**Return**

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG if either of the arguments is out of range

**Parameters**

- domain: power domain to configure
- option: power down option (ESP\_PD\_OPTION\_OFF, ESP\_PD\_OPTION\_ON, or ESP\_PD\_OPTION\_AUTO)

void **esp\_deep\_sleep\_start** ()

Enter deep sleep with the configured wakeup options.

This function does not return.

*esp\_err\_t* **esp\_light\_sleep\_start** ()

Enter light sleep with the configured wakeup options.

**Return**

- ESP\_OK on success (returned after wakeup)
- ESP\_ERR\_INVALID\_STATE if WiFi or BT is not stopped

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.

**Parameters**

- *time\_in\_us*: deep-sleep time, unit: microsecond

*esp\_sleep\_wakeup\_cause\_t* **esp\_sleep\_get\_wakeup\_cause** ()

Get the wakeup source which caused wakeup from sleep.

**Return** cause of wake up from last sleep (deep sleep or light sleep)

void **esp\_wake\_deep\_sleep** (void)

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.

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()`.

*esp\_deep\_sleep\_wake\_stub\_fn\_t* **esp\_get\_deep\_sleep\_wake\_stub** (void)  
Get current wake from deep sleep stub.

**Return** Return current wake from deep sleep stub, or NULL if no stub is installed.

void **esp\_default\_wake\_deep\_sleep** (void)  
The default esp-idf-provided `esp_wake_deep_sleep()` stub.

See docs/deep-sleep-stub.rst for details.

void **esp\_deep\_sleep\_disable\_rom\_logging** (void)  
Disable logging from the ROM code after deep sleep.

Using LSB of `RTC_STORE4`.

### 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:*

**ESP\_EXT1\_WAKEUP\_ALL\_LOW** = 0  
Wake the chip when all selected GPIOs go low.

**ESP\_EXT1\_WAKEUP\_ANY\_HIGH** = 1  
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:*

**ESP\_PD\_DOMAIN\_RTC\_PERIPH**  
RTC IO, sensors and ULP co-processor.

**ESP\_PD\_DOMAIN\_RTC\_SLOW\_MEM**  
RTC slow memory.

**ESP\_PD\_DOMAIN\_RTC\_FAST\_MEM**  
RTC fast memory.

**ESP\_PD\_DOMAIN\_XTAL**  
XTAL oscillator.

**ESP\_PD\_DOMAIN\_MAX**  
Number of domains.

**enum** **esp\_sleep\_pd\_option\_t**  
Power down options.

*Values:*

**ESP\_PD\_OPTION\_OFF**  
Power down the power domain in sleep mode.



**ESP\_PD\_OPTION\_ON**

Keep power domain enabled during sleep mode.

**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:*

**ESP\_SLEEP\_WAKEUP\_UNDEFINED**

In case of deep sleep, reset was not caused by exit from deep sleep.

**ESP\_SLEEP\_WAKEUP\_ALL**

Not a wakeup cause, used to disable all wakeup sources with `esp_sleep_disable_wakeup_source`.

**ESP\_SLEEP\_WAKEUP\_EXT0**

Wakeup caused by external signal using `RTC_IO`.

**ESP\_SLEEP\_WAKEUP\_EXT1**

Wakeup caused by external signal using `RTC_CNTL`.

**ESP\_SLEEP\_WAKEUP\_TIMER**

Wakeup caused by timer.

**ESP\_SLEEP\_WAKEUP\_TOUCHPAD**

Wakeup caused by touchpad.

**ESP\_SLEEP\_WAKEUP\_ULP**

Wakeup caused by ULP program.

**ESP\_SLEEP\_WAKEUP\_GPIO**

Wakeup caused by GPIO (light sleep only)

**ESP\_SLEEP\_WAKEUP\_UART**

Wakeup caused by UART (light sleep only)

## 2.7.16 Over The Air Updates (OTA)

### OTA Process Overview

The OTA update mechanism allows a device to update itself based on data received while the normal firmware is running (for example, over WiFi or Bluetooth.)

OTA requires configuring the *Partition Table* of the device with at least two “OTA app slot” partitions (ie `ota_0` and `ota_1`) and an “OTA Data Partition” .

The OTA operation functions write a new app firmware image to whichever OTA app slot is not currently being used for booting. Once the image is verified, the OTA Data partition is updated to specify that this image should be used for the next boot.

### OTA Data Partition

An OTA data partition (type `data`, subtype `ota`) must be included in the *Partition Table* of any project which uses the OTA functions.

For factory boot settings, the OTA data partition should contain no data (all bytes erased to `0xFF`). In this case the `esp-idf` software bootloader will boot the factory app if it is present in the the partition table. If no factory app is included in the partition table, the first available OTA slot (usually `ota_0`) is booted.

After the first OTA update, the OTA data partition is updated to specify which OTA app slot partition should be booted next.

The OTA data partition is two flash sectors (0x2000 bytes) in size, to prevent problems if there is a power failure while it is being written. Sectors are independently erased and written with matching data, and if they disagree a counter field is used to determine which sector was written more recently.

### App rollback

The main purpose of the application rollback is to keep the device working after the update. This feature allows you to roll back to the previous working application in case a new application has critical errors. When the rollback process is enabled and an OTA update provides a new version of the app, one of three things can happen:

- The application works fine, `esp_ota_mark_app_valid_cancel_rollback()` marks the running application with the state `ESP_OTA_IMG_VALID`. There are no restrictions on booting this application.
- The application has critical errors and further work is not possible, a rollback to the previous application is required, `esp_ota_mark_app_invalid_rollback_and_reboot()` marks the running application with the state `ESP_OTA_IMG_INVALID` and reset. This application will not be selected by the bootloader for boot and will boot the previously working application.
- If the `CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE` option is set, and occur a reset without calling either function then happen and is rolled back.

Note: The state is not written to the binary image of the application it is written to the `otadata` partition. The partition contains a `ota_seq` counter which is a pointer to the slot (`ota_0`, `ota_1`, ...) from which the application will be selected for boot.

**App OTA State** States control the process of selecting a boot app:

States	Restriction of selecting a boot app in bootloader
<code>ESP_OTA_IMG_VALID</code>	No restriction. Will be selected.
<code>ESP_OTA_IMG_UNDEFINED</code>	No restriction. Will be selected.
<code>ESP_OTA_IMG_INVALID</code>	Will not be selected.
<code>ESP_OTA_IMG_ABORTED</code>	Will not be selected.
<code>ESP_OTA_IMG_NEW</code>	If <code>CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE</code> option is set it will be selected only once. In bootloader the state immediately changes to <code>ESP_OTA_IMG_PENDING_VERIFY</code> .
<code>ESP_OTA_IMG_PENDING_VERIFY</code>	If <code>CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE</code> option is set it will not be selected and the state will change to <code>ESP_OTA_IMG_ABORTED</code> .

If `CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE` option is not enabled (by default), then the use of the following functions `esp_ota_mark_app_valid_cancel_rollback()` and `esp_ota_mark_app_invalid_rollback_and_reboot()` are optional, and `ESP_OTA_IMG_NEW` and `ESP_OTA_IMG_PENDING_VERIFY` states are not used.

An option in Kconfig `CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE` allows you to track the first boot of a new application. In this case, the application must confirm its operability by calling `esp_ota_mark_app_valid_cancel_rollback()` function, otherwise the application will be rolled back upon reboot. It allows you to control the operability of the application during the boot phase. Thus, a new application has only one attempt to boot successfully.

**Rollback Process** The description of the rollback process when `CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE` option is enabled:

- The new application successfully downloaded and `esp_ota_set_boot_partition()` function makes this partition bootable and sets the state `ESP_OTA_IMG_NEW`. This state means that the application is new and should be monitored for its first boot.
- Reboot `esp_restart()`.
- The bootloader checks for the `ESP_OTA_IMG_PENDING_VERIFY` state if it is set, then it will be written to `ESP_OTA_IMG_ABORTED`.

- The bootloader selects a new application to boot so that the state is not set as `ESP_OTA_IMG_INVALID` or `ESP_OTA_IMG_ABORTED`.
- The bootloader checks the selected application for `ESP_OTA_IMG_NEW` state if it is set, then it will be written to `ESP_OTA_IMG_PENDING_VERIFY`. This state means that the application requires confirmation of its operability, if this does not happen and a reboot occurs, this state will be overwritten to `ESP_OTA_IMG_ABORTED` (see above) and this application will no longer be able to start, i.e. there will be a rollback to the previous work application.
- A new application has started and should make a self-test.
- If the self-test has completed successfully, then you must call the function `esp_ota_mark_app_valid_cancel_rollback()` because the application is awaiting confirmation of operability (`ESP_OTA_IMG_PENDING_VERIFY` state).
- If the self-test fails then call `esp_ota_mark_app_invalid_rollback_and_reboot()` function to roll back to the previous working application, while the invalid application is set `ESP_OTA_IMG_INVALID` state.
- If the application has not been confirmed, the state remains `ESP_OTA_IMG_PENDING_VERIFY`, and the next boot it will be changed to `ESP_OTA_IMG_ABORTED`. That will prevent re-boot of this application. There will be a rollback to the previous working application.

**Unexpected Reset** If a power loss or an unexpected crash occurs at the time of the first boot of a new application, it will roll back the application.

Recommendation: Perform the self-test procedure as quickly as possible, to prevent rollback due to power loss.

Only OTA partitions can be rolled back. Factory partition is not rolled back.

**Booting invalid/aborted apps** Booting an application which was previously set to `ESP_OTA_IMG_INVALID` or `ESP_OTA_IMG_ABORTED` is possible:

- Get the last invalid application partition `esp_ota_get_last_invalid_partition()`.
- Pass the received partition to `esp_ota_set_boot_partition()`, this will update the otadata.
- Restart `esp_restart()`. The bootloader will boot the specified application.

To determine if self-tests should be run during startup of an application, call the `esp_ota_get_state_partition()` function. If result is `ESP_OTA_IMG_PENDING_VERIFY` then self-testing and subsequent confirmation of operability is required.

**Where the states are set** A brief description of where the states are set:

- `ESP_OTA_IMG_VALID` state is set by `esp_ota_mark_app_valid_cancel_rollback()` function.
- `ESP_OTA_IMG_UNDEFINED` state is set by `esp_ota_set_boot_partition()` function if `CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE` option is not enabled.
- `ESP_OTA_IMG_NEW` state is set by `esp_ota_set_boot_partition()` function if `CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE` option is enabled.
- `ESP_OTA_IMG_INVALID` state is set by `esp_ota_mark_app_invalid_rollback_and_reboot()` function.
- `ESP_OTA_IMG_ABORTED` state is set if there was no confirmation of the application operability and occurs reboots (if `CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE` option is enabled).
- `ESP_OTA_IMG_PENDING_VERIFY` state is set in a bootloader if `CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE` option is enabled and selected app has `ESP_OTA_IMG_NEW` state.

### Anti-rollback

Anti-rollback prevents rollback to application with security version lower than one programmed in eFuse of chip.

This function works if set `CONFIG_BOOTLOADER_APP_ANTI_ROLLBACK` option. In the bootloader, when selecting a bootable application, an additional security version check is added which is on the chip and in the application image. The version in the bootable firmware must be greater than or equal to the version in the chip.

`CONFIG_BOOTLOADER_APP_ANTI_ROLLBACK` and `CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE` options are used together. In this case, rollback is possible only on the security version which is equal or higher than the version in the chip.

### A typical anti-rollback scheme is

- New firmware released with the elimination of vulnerabilities with the previous version of security.
- After the developer makes sure that this firmware is working. He can increase the security version and release a new firmware.
- Download new application.
- To make it bootable, run the function `esp_ota_set_boot_partition()`. If the security version of the new application is smaller than the version in the chip, the new application will be erased. Update to new firmware is not possible.
- Reboot.
- In the bootloader, an application with a security version greater than or equal to the version in the chip will be selected. If otadata is in the initial state, and one firmware was loaded via a serial channel, whose secure version is higher than the chip, then the secure version of efuse will be immediately updated in the bootloader.
- New application booted. Then the application should perform diagnostics of the operation and if it is completed successfully, you should call `esp_ota_mark_app_valid_cancel_rollback()` function to mark the running application with the `ESP_OTA_IMG_VALID` state and update the secure version on chip. Note that if was called `esp_ota_mark_app_invalid_rollback_and_reboot()` function a rollback may not happen due to the device may not have any bootable apps then it will return `ESP_ERR_OTA_ROLLBACK_FAILED` error and stay in the `ESP_OTA_IMG_PENDING_VERIFY` state.
- The next update of app is possible if a running app is in the `ESP_OTA_IMG_VALID` state.

Recommendation:

If you want to avoid the download/erase overhead in case of the app from the server has security version lower than running app you have to get `new_app_info.secure_version` from the first package of an image and compare it with the secure version of efuse. Use `esp_efuse_check_secure_version(new_app_info.secure_version)` function if it is true then continue downloading otherwise abort.

```

....
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);
    }
}

```

(下页继续)

```

    }
}
...

```

Restrictions:

- The number of bits in the `secure_version` field is limited to 32 bits. This means that only 32 times you can do an anti-rollback. You can reduce the length of this efuse field use [CONFIG\\_BOOTLOADER\\_APP\\_SEC\\_VER\\_SIZE\\_EFUSE\\_FIELD](#) option.
- Anti-rollback only works if the encoding scheme for efuse is set to NONE.
- The partition table should not have a factory partition, only two of the app.

`security_version`:

- In application image it is stored in `esp_app_desc` structure. The number is set [CONFIG\\_BOOTLOADER\\_APP\\_SECURE\\_VERSION](#).
- In ESP32 it is stored in efuse `EFUSE_BLK3_RDATA4_REG`. (when a eFuse bit is programmed to 1, it can never be reverted to 0). The number of bits set in this register is the `security_version` from app.

### Secure OTA Updates Without Secure boot

The verification of signed OTA updates can be performed even without enabling hardware secure boot. For doing so, refer [Signed App Verification Without Hardware Secure Boot](#)

### OTA Tool (otatool.py)

The component `app_update` provides a tool `otatool.py` for performing OTA partition-related operations on a target device. The following operations can be performed using the tool:

- read contents of otadata partition (`read_otadata`)
- erase otadata partition, effectively resetting device to factory app (`erase_otadata`)
- switch OTA partitions (`switch_ota_partition`)
- erasing OTA partition (`erase_ota_partition`)
- write to OTA partition (`write_ota_partition`)
- read contents of OTA partition (`read_ota_partition`)

The tool can either be imported and used from another Python script or invoked from shell script for users wanting to perform operation programmatically. This is facilitated by the tool's Python API and command-line interface, respectively.

**Python API** Before anything else, make sure that the `otatool` module is imported.

```

import sys
import os

idf_path = os.environ["IDF_PATH"] # get value of IDF_PATH from environment
otatool_dir = os.path.join(idf_path, "components", "app_update") # otatool.py_
↳ lives in $IDF_PATH/components/app_update

sys.path.append(otatool_dir) # this enables Python to find otatool module
from otatool import * # import all names inside otatool module

```

The starting point for using the tool's Python API to do is create a `OtatoolTarget` object:

```

# Create a partool.py target device connected on serial port /dev/ttyUSB1
target = OtatoolTarget("/dev/ttyUSB1")

```

The created object can now be used to perform operations on the target device:

```
# Erase otadata, resetting the device to factory app
target.erase_otadata()

# Erase contents of OTA app slot 0
target.erase_ota_partition(0)

# Switch boot partition to that of app slot 1
target.switch_ota_partition(1)

# Read OTA partition 'ota_3' and save contents to a file named 'ota_3.bin'
target.read_ota_partition("ota_3", "ota_3.bin")
```

The OTA partition to operate on is specified using either the app slot number or the partition name.

More information on the Python API is available in the docstrings for the tool.

**Command-line Interface** The command-line interface of *otatool.py* has the following structure:

```
otatool.py [command-args] [subcommand] [subcommand-args]

- command-args - these are arguments that are needed for executing the main_
↳command (parttool.py), mostly pertaining to the target device
- subcommand - this is the operation to be performed
- subcommand-args - these are arguments that are specific to the chosen operation
```

```
# Erase otadata, resetting the device to factory app
otatool.py --port "/dev/ttyUSB1" erase_otadata

# Erase contents of OTA app slot 0
otatool.py --port "/dev/ttyUSB1" erase_ota_partition --slot 0

# Switch boot partition to that of app slot 1
otatool.py --port "/dev/ttyUSB1" switch_ota_partition --slot 1

# Read OTA partition 'ota_3' and save contents to a file named 'ota_3.bin'
otatool.py --port "/dev/ttyUSB1" read_ota_partition --name=ota_3
```

More information can be obtained by specifying *-help* as argument:

```
# Display possible subcommands and show main command argument descriptions
otatool.py --help

# Show descriptions for specific subcommand arguments
otatool.py [subcommand] --help
```

### See also

- [Partition Table documentation](#)
- [Lower-Level SPI Flash/Partition API](#)
- [ESP HTTPS OTA](#)

### Application Example

End-to-end example of OTA firmware update workflow: [system/ota](#).

### API Reference

### Header File

- [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.

**Return** Pointer to esp\_app\_desc structure.

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.

**Return** Number of bytes written to *dst* (including null terminator)

### Parameters

- *dst*: Destination buffer
- *size*: Size of the buffer

*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).

### 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\_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.

### 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.

*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.

### Return

- 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

**Parameters**

- `handle`: Handle obtained from `esp_ota_begin`
- `data`: Data buffer to write
- `size`: Size of data buffer in bytes.

`esp_err_t esp_ota_end(esp_ota_handle_t handle)`

Finish OTA update and validate newly written app image.

**Note** After calling `esp_ota_end()`, the handle is no longer valid and any memory associated with it is freed (regardless of result).

**Return**

- `ESP_OK`: Newly written OTA app image is valid.
- `ESP_ERR_NOT_FOUND`: OTA handle was not found.
- `ESP_ERR_INVALID_ARG`: Handle was never written to.
- `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.

**Parameters**

- `handle`: Handle obtained from `esp_ota_begin()`.

`esp_err_t esp_ota_set_boot_partition(const esp_partition_t *partition)`

Configure OTA data for a new boot partition.

**Note** If this function returns `ESP_OK`, calling `esp_restart()` will boot the newly configured app partition.

**Return**

- `ESP_OK`: OTA data updated, next reboot will use specified partition.
- `ESP_ERR_INVALID_ARG`: partition argument was NULL or didn't point to a valid OTA partition of type "app".
- `ESP_ERR_OTA_VALIDATE_FAILED`: Partition contained invalid app image. Also returned if secure boot is enabled and signature validation failed.
- `ESP_ERR_NOT_FOUND`: OTA data partition not found.
- `ESP_ERR_FLASH_OP_TIMEOUT` or `ESP_ERR_FLASH_OP_FAIL`: Flash erase or write failed.

**Parameters**

- `partition`: Pointer to info for partition containing app image to boot.

`const esp_partition_t *esp_ota_get_boot_partition(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.

**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.

`const esp_partition_t *esp_ota_get_running_partition(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.

**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.

```
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.

**Return** Pointer to info for partition which should be updated next. NULL result indicates invalid OTA data partition, or that no eligible OTA app slot partition was found.

#### Parameters

- `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.

Returns a description for the requested app partition.

#### Return

- `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.

#### Parameters

- [in] `partition`: Pointer to app partition. (only app partition)
- [out] `app_desc`: Structure of info about app.

```
esp_err_t esp_ota_mark_app_valid_cancel_rollback ()
```

This function is called to indicate that the running app is working well.

#### Return

- `ESP_OK`: if successful.

```
esp_err_t esp_ota_mark_app_invalid_rollback_and_reboot ()
```

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.

#### Return

- `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 ()
```

Returns last partition with invalid state (`ESP_OTA_IMG_INVALID` or `ESP_OTA_IMG_ABORTED`).

**Return** partition.

*esp\_err\_t* **esp\_ota\_get\_state\_partition**(const *esp\_partition\_t* \*partition, *esp\_ota\_img\_states\_t* \*ota\_state)

Returns state for given partition.

#### Return

- 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 otadata or state was not found for given partition.

#### Parameters

- [in] partition: Pointer to partition.
- [out] ota\_state: state of partition (if this partition has a record in otadata).

*esp\_err\_t* **esp\_ota\_erase\_last\_boot\_app\_partition**(void)

Erase previous boot app partition and corresponding otadata select for this partition.

When current app is marked to as valid then you can erase previous app partition.

#### Return

- ESP\_OK: Successful, otherwise ESP\_ERR.

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 otadata 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).

#### Return

- 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

#### 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

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

### 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().

## 2.7.17 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

```
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_err_t ret = esp_https_ota(&config);
    if (ret == ESP_OK) {
        esp_restart();
    } else {
        return ESP_FAIL;
    }
    return ESP_OK;
}
```

### Signature Verification

For additional security, signature of OTA firmware images can be verified. For that, refer [Secure OTA Updates Without Secure boot](#)

### API Reference

#### Header File

- [esp\\_https\\_ota/include/esp\\_https\\_ota.h](#)

#### Functions

`esp_err_t esp_https_ota(const esp_http_client_config_t *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 `config`.

**Note** 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.

#### Return

- `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.

#### Parameters

- [in] config: pointer to *esp\_http\_client\_config\_t* structure.

*esp\_err\_t* **esp\_https\_ota\_begin** (*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.

**Note** This API is blocking, so setting *is\_async* member of *http\_config* structure will result in an error.

#### Return

- 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*.

#### Parameters

- [in] ota\_config: pointer to *esp\_https\_ota\_config\_t* structure
- [out] handle: pointer to an allocated data of type *esp\_https\_ota\_handle\_t* which will be initialised in this function

*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.

#### Return

- 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\_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.

#### Parameters

- [in] https\_ota\_handle: pointer to *esp\_https\_ota\_handle\_t* structure

bool **esp\_https\_ota\_is\_complete\_data\_received** (*esp\_https\_ota\_handle\_t* https\_ota\_handle)

Checks if complete data was received or not.

**Note** This API can be called just before *esp\_https\_ota\_finish*() to validate if the complete image was indeed received.

#### Return

- false
- true

#### Parameters

- [in] https\_ota\_handle: pointer to *esp\_https\_ota\_handle\_t* structure

*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.

**Note** If this API returns successfully, *esp\_restart*() must be called to boot from the new firmware image

#### Return

- ESP\_OK: Clean-up successful
- ESP\_ERR\_INVALID\_STATE
- ESP\_ERR\_INVALID\_ARG: Invalid argument
- ESP\_ERR\_OTA\_VALIDATE\_FAILED: Invalid app image

**Parameters**

- [in] `https_ota_handle`: pointer to `esp_https_ota_handle_t` structure

`esp_err_t esp_https_ota_get_img_desc(esp_https_ota_handle_t https_ota_handle, esp_app_desc_t *new_app_info)`

Reads app description from image header. The app description provides information like the “Firmware version” of the image.

**Note** This API can be called only after `esp_https_ota_begin()` and before `esp_https_ota_perform()`. Calling this API is not mandatory.

**Return**

- ESP\_ERR\_INVALID\_ARG: Invalid arguments
- ESP\_FAIL: Failed to read image descriptor
- ESP\_OK: Successfully read image descriptor

**Parameters**

- [in] `https_ota_handle`: pointer to `esp_https_ota_handle_t` structure
- [out] `new_app_info`: pointer to an allocated `esp_app_desc_t` structure

`int esp_https_ota_get_image_len_read(esp_https_ota_handle_t https_ota_handle)`

**Structures**

`struct esp_https_ota_config_t`  
ESP HTTPS OTA configuration.

**Public Members**

`const esp_http_client_config_t *http_config`  
ESP HTTP client configuration

**Macros**

`ESP_ERR_HTTPS_OTA_BASE`  
`ESP_ERR_HTTPS_OTA_IN_PROGRESS`

**Type Definitions**

`typedef void *esp_https_ota_handle_t`

## 2.7.18 ESP-pthread

**Overview**

This module offers Espressif specific extensions to the pthread library that can be used to influence the behaviour of pthread.

- Stack size of the pthreads
- Priority of the created pthreads
- Inheriting this configuration across threads
- Thread name
- Core affinity / core pinning.

Example to tune the stack size of the pthread:

```

main()
{
    pthread_t t1;

    esp_thread_cfg_t cfg = esp_create_default_thread_config();
    cfg.stack_size = (4 * 1024);
    esp_thread_set_cfg(&cfg);

    pthread_create(&t1, NULL, thread_func);
}

```

The API can also be used for inheriting the settings across threads. For example:

```

void * my_thread2(void * p)
{
    /* This thread will inherit the stack size of 4K */
    printf("In my_thread2\n");
}

void * my_thread1(void * p)
{
    printf("In my_thread1\n");
    pthread_t t2;
    pthread_create(&t2, NULL, my_thread2);
}

main()
{
    pthread_t t1;

    esp_thread_cfg_t cfg = esp_create_default_thread_config();
    cfg.stack_size = (4 * 1024);
    cfg.inherit_cfg = true;
    esp_thread_set_cfg(&cfg);

    pthread_create(&t1, NULL, my_thread1);
}

```

## API Reference

### Header File

- [pthread/include/esp\\_thread.h](#)

### Functions

*esp\_thread\_cfg\_t* **esp\_thread\_get\_default\_config()**

Creates a default pthread configuration based on the values set via menuconfig.

**Return** A default configuration structure.

*esp\_err\_t* **esp\_thread\_set\_cfg(const *esp\_thread\_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 setup 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.

**Note** Passing non-NULL attributes to pthread\_create() will override the stack\_size parameter set using this API

**Return**

- 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`

**Parameters**

- `cfg`: The pthread config parameters

`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.

**Return**

- ESP\_OK if the configuration was available
- ESP\_ERR\_NOT\_FOUND if a configuration wasn't previously set

**Parameters**

- `p`: Pointer to the pthread config structure that will be updated with the currently configured parameters

**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**

## 2.7.19 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](#).

**API Reference****Header File**

- [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.

**Return** string error message

### Parameters

- code: esp\_err\_t error code

**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.

**Return** buf containing the string error message

### Parameters

- code: esp\_err\_t error code
- [out] buf: 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).

## 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



**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\_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 int32_t esp_err_t
```

**2.7.20 App Image Format**

An application image consists of the following structures:

1. The `esp_image_header_t` structure describes the mode of SPI flash and the count of memory segments.
2. The `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 `data_len`. The data offset for each segment in the image is calculated in the following way:
  - offset for 0 Segment = `sizeof(esp_image_header_t) + sizeof(esp_image_segment_header_t)`.
  - offset for 1 Segment = offset for 0 Segment + length of 0 Segment + `sizeof(esp_image_segment_header_t)`.
  - offset for 2 Segment = offset for 1 Segment + length of 1 Segment + `sizeof(esp_image_segment_header_t)`.
  - ...

The count of each segment is defined in the `segment_count` field that is stored in `esp_image_header_t`. The count cannot be more than ESP\_IMAGE\_MAX\_SEGMENTS.

To get the list of your image segments, please run the following command:

```
esptool.py --chip esp32 image_info build/app.bin
```

```
esptool.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 0x3ffb0000 file_offs 0x00013d10 DRAM
Segment 5: len 0x00000 load 0x3ffb28e0 file_offs 0x000165f8 DRAM
Segment 6: len 0x00400 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_IROM
Segment 9: len 0x06cec load 0x40089a00 file_offs 0x00082e64 SOC_IROM
Segment 10: len 0x00000 load 0x400c0000 file_offs 0x00089b58 SOC_RTC_IRAM
Segment 11: len 0x00004 load 0x50000000 file_offs 0x00089b60 SOC_RTC_DATA
```

(下页继续)

(续上页)

```
Segment 12: len 0x00000 load 0x50000004 file_offs 0x00089b6c SOC_RTC_DATA
Segment 13: len 0x00000 load 0x50000004 file_offs 0x00089b74 SOC_RTC_DATA
Checksum: e8 (valid) Validation Hash: ↵
↵407089ca0eae2bbf83b4120979d3354b1c938a49cb7a0c997f240474ef2ec76b (valid)
```

You can also see the information on segments in the 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=0x3ffb8000 size=0x00000 ( 0) ↵
↵load
I (530) esp_image: segment 2: paddr=0x00033d10 vaddr=0x3ffb8000 size=0x00000 ( 0) ↵
↵load
I (571) esp_image: segment 3: paddr=0x00033d18 vaddr=0x3ffb0000 size=0x028e0 ( ↵
↵10464) load
I (612) esp_image: segment 4: paddr=0x00036600 vaddr=0x3ffb28e0 size=0x00000 ( 0) ↵
↵load
I (654) esp_image: segment 5: paddr=0x00036608 vaddr=0x40080000 size=0x00400 ( ↵
↵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=0x000a9b68 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 the ESP32 Technical Reference Manual, Section 1.3.2 *Embedded Memory*.

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 first byte and up to this field. The length of this field is 32 bytes.
5. If the options `CONFIG_SECURE_SIGNED_APPS_NO_SECURE_BOOT` or `CONFIG_SECURE_BOOT_ENABLED` are enabled then the application image will have additional 68 bytes for an ECDSA signature, which includes:
  - version word (4 bytes),
  - signature data (64 bytes).

## Application Description

The DROM segment starts with the `esp_app_desc_t` structure which carries specific fields describing the application:

- `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 elf application file.

\* - The maximum length is 32 characters, including null-termination character. For example, if the length of `PROJECT_NAME` exceeds 32 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.

### Adding a Custom Structure to an Application

Customer also has 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:

```
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:

- For Make: add `COMPONENT_ADD_LDFLAGS += -u custom_app_desc` into `component.mk`
- For Cmake: add `target_link_libraries(${COMPONENT_TARGET} "-u custom_app_desc")` into `CMakeLists.txt`

## API Reference

### Header File

- [bootloader\\_support/include/esp\\_app\\_format.h](#)

### 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 : 4**

flash frequency (`esp_image_spi_freq_t` as `uint8_t`)

**uint8\_t spi\_size : 4**

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**

Minimum chip revision supported by image

uint8\_t **reserved**[8]

Reserved bytes in additional header space, currently unused

uint8\_t **hash\_appended**

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

**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\_IMAGE\_HEADER\_MAGIC**

The magic word for the *esp\_image\_header\_t* structure.

**ESP\_IMAGE\_MAX\_SEGMENTS**

Max count of segments in the image.

**ESP\_APP\_DESC\_MAGIC\_WORD**

The magic word for the *esp\_app\_desc* structure that is in DROM.

**Enumerations****enum esp\_chip\_id\_t**

ESP chip ID.

*Values:*

**ESP\_CHIP\_ID\_ESP32** = 0x0000

chip ID: ESP32

**ESP\_CHIP\_ID\_INVALID** = 0xFFFF

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:*

**ESP\_IMAGE\_SPI\_MODE\_QIO**

SPI mode QIO

**ESP\_IMAGE\_SPI\_MODE\_QOUT**

SPI mode QOUT

**ESP\_IMAGE\_SPI\_MODE\_DIO**

SPI mode DIO

**ESP\_IMAGE\_SPI\_MODE\_DOUT**

SPI mode DOUT

**ESP\_IMAGE\_SPI\_MODE\_FAST\_READ**

SPI mode FAST\_READ

**ESP\_IMAGE\_SPI\_MODE\_SLOW\_READ**

SPI mode SLOW\_READ

**enum esp\_image\_spi\_freq\_t**

SPI flash clock frequency.

*Values:*

**ESP\_IMAGE\_SPI\_SPEED\_40M**

SPI clock frequency 40 MHz

**ESP\_IMAGE\_SPI\_SPEED\_26M**

SPI clock frequency 26 MHz

**ESP\_IMAGE\_SPI\_SPEED\_20M**

SPI clock frequency 20 MHz

**ESP\_IMAGE\_SPI\_SPEED\_80M** = 0xF

SPI clock frequency 80 MHz

**enum esp\_image\_flash\_size\_t**

Supported SPI flash sizes.

*Values:*

**ESP\_IMAGE\_FLASH\_SIZE\_1MB** = 0

SPI flash size 1 MB

**ESP\_IMAGE\_FLASH\_SIZE\_2MB**

SPI flash size 2 MB

**ESP\_IMAGE\_FLASH\_SIZE\_4MB**

SPI flash size 4 MB

**ESP\_IMAGE\_FLASH\_SIZE\_8MB**

SPI flash size 8 MB

**ESP\_IMAGE\_FLASH\_SIZE\_16MB**

SPI flash size 16 MB

**ESP\_IMAGE\_FLASH\_SIZE\_MAX**

SPI flash size MAX

## 2.7.21 Miscellaneous System APIs

### Software reset

To perform software reset of the chip, `esp_restart()` function is provided. When the function is called, execution of the program will stop, both CPUs will be reset, application will be loaded by the bootloader and started again.

Additionally, `esp_register_shutdown_handler()` function is provided to register a routine which needs to be called prior to restart (when done by `esp_restart()`). This is similar to the functionality of `atexit` POSIX function.

### Reset reason

ESP-IDF application can be started or restarted due to a variety of reasons. To get the last reset reason, call `esp_reset_reason()` function. See description of `esp_reset_reason_t` for the list of possible reset reasons.

### Heap memory

Two heap memory related functions are provided:

- `esp_get_free_heap_size()` returns the current size of free heap memory
- `esp_get_minimum_free_heap_size()` returns the minimum size of free heap memory that was available during program execution.

Note that ESP-IDF supports multiple heaps with different capabilities. Functions mentioned in this section return the size of heap memory which can be allocated using `malloc` family of functions. For further information about heap memory see [Heap Memory Allocation](#).

### Random number generation

ESP32 contains a hardware random number generator, values from it can be obtained using `esp_random()`.

When Wi-Fi or Bluetooth are enabled, numbers returned by hardware random number generator (RNG) can be considered true random numbers. Without Wi-Fi or Bluetooth enabled, hardware RNG is a pseudo-random number generator. At startup, ESP-IDF bootloader seeds the hardware RNG with entropy, but care must be taken when reading random values between the start of `app_main` and initialization of Wi-Fi or Bluetooth drivers.

## MAC Address

These APIs allow querying and customizing MAC addresses used by Wi-Fi, Bluetooth, and Ethernet drivers.

ESP32 has up to 4 network interfaces: Wi-Fi station, Wi-Fi AP, Ethernet, and Bluetooth. Each of these interfaces needs to have a MAC address assigned to it. In ESP-IDF these addresses are calculated from *Base MAC address*. Base MAC address can be initialized with factory-programmed value from EFUSE, or with a user-defined value. In addition to setting the base MAC address, applications can specify the way in which MAC addresses are allocated to devices. See [Number of universally administered MAC address](#) section for more details.

Interface	MAC address (4 universally administered)	MAC address (2 universally administered)
Wi-Fi Station	base_mac	base_mac
Wi-Fi SoftAP	base_mac, +1 to the last octet	base_mac, first octet randomized
Bluetooth	base_mac, +2 to the last octet	base_mac, +1 to the last octet
Ethernet	base_mac, +3 to the last octet	base_mac, +1 to the last octet, first octet randomized

**Base MAC address** Wi-Fi, Bluetooth, and Ethernet drivers use `esp_read_mac()` function to get MAC address for a specific interface.

By default, this function will use MAC address factory programmed in BLK0 of EFUSE as the base MAC address. MAC addresses of each interface will be calculated according to the table above.

Applications which don't use MAC address factory programmed into BLK0 of EFUSE can modify base MAC address used by `esp_read_mac()` using a call to `esp_base_mac_addr_set()`. Custom value of MAC address can come from application defined storage, such as Flash, NVS, etc. Note that the call to `esp_base_mac_addr_set()` needs to happen before network protocol stacks are initialized, for example, early in `app_main`.

**Custom MAC address in BLK3 of EFUSE** To facilitate usage of custom MAC addresses, ESP-IDF provides `esp_efuse_mac_get_custom()` function, which loads MAC address from BLK3 of EFUSE. This function assumes that custom MAC address is stored in BLK3 of EFUSE (EFUSE\_BLK3\_RDATA0, EFUSE\_BLK3\_RDATA1, EFUSE\_BLK3\_RDATA2, EFUSE\_BLK3\_RDATA3, EFUSE\_BLK3\_RDATA4, EFUSE\_BLK3\_RDATA5 registers) in the following format:

Field	# of bits	Range of bits	Notes
Version	8	191:184	0: invalid, others — valid
Reserved	128	183:56	
MAC address	48	55:8	
MAC address CRC	8	7:0	CRC-8-CCITT, polynomial 0x07

Once MAC address has been obtained using `esp_efuse_mac_get_custom()`, call `esp_base_mac_addr_set()` to set this MAC address as base MAC address.

**Number of universally administered MAC address** Several MAC addresses (universally administered by IEEE) are uniquely assigned to the networking interfaces (Wi-Fi/BT/Ethernet). The final octet of each universally administered MAC address increases by one. Only the first one of them (which is called base MAC address) is stored in EFUSE or external storage, the others are generated from it. Here, 'generate' means adding 0, 1, 2 and 3 (respectively) to the final octet of the base MAC address.

If the universally administered MAC addresses are not enough for all of the networking interfaces, locally administered MAC addresses which are derived from universally administered MAC addresses are assigned to the rest of networking interfaces.

See [this article](#) for the definition of local and universally administered MAC addresses.

The number of universally administered MAC address can be configured using [CONFIG\\_ESP32\\_UNIVERSAL\\_MAC\\_ADDRESSES](#).

If the number of universal MAC addresses is two, only two interfaces (Wi-Fi 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 (Wi-Fi SoftAP and Ethernet) receive local MAC addresses. These are derived from the universal Wi-Fi station and Bluetooth MAC addresses, respectively.

If the number of universal MAC addresses is four, all four interfaces (Wi-Fi Station, Wi-Fi 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.

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.)

### Chip version

`esp_chip_info()` function fills `esp_chip_info_t` structure with information about the chip. This includes the chip revision, number of CPU cores, and a bit mask of features enabled in the chip.

### SDK version

`esp_get_idf_version()` returns a string describing the IDF version which was used to compile the application. This is the same value as the one available through `IDF_VER` variable of the build system. The version string generally has the format of `git describe output`.

To get the version at build time, additional version macros are provided. They can be used to enable or disable parts of the program depending on IDF version.

- `ESP_IDF_VERSION_MAJOR`, `ESP_IDF_VERSION_MINOR`, `ESP_IDF_VERSION_PATCH` are defined to integers representing major, minor, and patch version.
- `ESP_IDF_VERSION_VAL` and `ESP_IDF_VERSION` can be used when implementing version checks:

```
#include "esp_idf_version.h"

#if ESP_IDF_VERSION >= ESP_IDF_VERSION_VAL(4, 0, 0)
    // enable functionality present in IDF v4.0
#endif
```

### App version

Application version is stored in `esp_app_desc_t` structure. It is located in DROM sector and has a fixed offset from the beginning of the binary file. The structure is located after `esp_image_header_t` and `esp_image_segment_header_t` structures. The field `version` has string type and max length 32 chars.

To set version in your project manually you need to set `PROJECT_VER` variable in your project `CMakeLists.txt/Makefile`:

- In application `CMakeLists.txt` put `set(PROJECT_VER "0.1.0.1")` before including `project.cmake`.

(For legacy GNU Make build system: in application `Makefile` put `PROJECT_VER = "0.1.0.1"` before including `project.mk`.)

If `PROJECT_VER` variable is not set in the project then it will be retrieved from either `$(PROJECT_PATH)/version.txt` file (if present) else using `git` command `git describe`. If neither is available then `PROJECT_VER` will be set to "1". Application can make use of this by calling `esp_ota_get_app_description()` or `esp_ota_get_partition_description()` functions.



## API Reference

### Header File

- [esp\\_common/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.

#### Return

- 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

#### Parameters

- handle: function to execute on restart

*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.

- 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 WiFi, 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.

**Return** 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.

Note that the returned value may be larger than the maximum contiguous block which can be allocated.

**Return** Available heap size, in bytes.

uint32\_t **esp\_get\_free\_internal\_heap\_size** (void)

Get the size of available internal heap.

Note that the returned value may be larger than the maximum contiguous block which can be allocated.

**Return** Available internal heap size, in bytes.

uint32\_t **esp\_get\_minimum\_free\_heap\_size** (void)

Get the minimum heap that has ever been available.

**Return** Minimum free heap ever available

uint32\_t **esp\_random** (void)

Get one random 32-bit word from hardware RNG.

The hardware RNG is fully functional whenever an RF subsystem is running (ie Bluetooth or WiFi is enabled). For random values, call this function after WiFi or Bluetooth are started.

If the RF subsystem is not used by the program, the function `bootloader_random_enable()` can be called to enable an entropy source. `bootloader_random_disable()` must be called before RF subsystem or I2S peripheral are used. See these functions' documentation for more details.

Any time the app is running without an RF subsystem (or `bootloader_random`) enabled, RNG hardware should be considered a PRNG. A very small amount of entropy is available due to pre-seeding while the IDF bootloader is running, but this should not be relied upon for any use.

**Return** Random value between 0 and `UINT32_MAX`

void **esp\_fill\_random** (void \*buf, size\_t len)

Fill a buffer with random bytes from hardware RNG.

**Note** This function has the same restrictions regarding available entropy as `esp_random()`

**Parameters**

- `buf`: Pointer to buffer to fill with random numbers.
- `len`: Length of buffer in bytes

*esp\_err\_t* **esp\_base\_mac\_addr\_set** (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 the networking interfaces. If using base MAC address stored in BLK3 of EFUSE or external storage, call this API to set base MAC address with the MAC address which is stored in BLK3 of EFUSE or external storage before initializing WiFi/BT/Ethernet.

**Return** `ESP_OK` on success

**Parameters**

- `mac`: base MAC address, length: 6 bytes.

*esp\_err\_t* **esp\_base\_mac\_addr\_get** (uint8\_t \*mac)

Return base MAC address which is set using `esp_base_mac_addr_set`.

**Return** `ESP_OK` on success `ESP_ERR_INVALID_MAC` base MAC address has not been set

**Parameters**

- `mac`: base MAC address, length: 6 bytes.

*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 BLK3 of EFUSE. 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.

**Return** `ESP_OK` on success `ESP_ERR_INVALID_VERSION` An invalid MAC version field was read from BLK3 of EFUSE `ESP_ERR_INVALID_CRC` An invalid MAC CRC was read from BLK3 of EFUSE

**Parameters**

- `mac`: base MAC address, length: 6 bytes.

*esp\_err\_t* **esp\_efuse\_mac\_get\_default** (uint8\_t \*mac)

Return base MAC address which is factory-programmed by Espressif in BLK0 of EFUSE.

**Return** `ESP_OK` on success

**Parameters**

- `mac`: base MAC address, length: 6 bytes.

*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` or reads base MAC address from BLK0 of EFUSE. Then set the MAC address of the interface including wifi station, wifi softap, bluetooth and ethernet.

**Return** `ESP_OK` on success

**Parameters**

- `mac`: MAC address of the interface, length: 6 bytes.

- type: type of MAC address, 0:wifi station, 1:wifi softap, 2:bluetooth, 3:ethernet.

*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 derives a local MAC address from an universal MAC address. A definition of local vs universal MAC address can be found on Wikipedia <>. In ESP32, universal MAC address is generated from base MAC address in EFUSE or other external storage. Local MAC address is derived from the universal MAC address.

**Return** ESP\_OK on success

**Parameters**

- local\_mac: Derived local MAC address, length: 6 bytes.
- universal\_mac: Source universal MAC address, length: 6 bytes.

void **esp\_chip\_info** (*esp\_chip\_info\_t* \**out\_info*)

Fill an *esp\_chip\_info\_t* structure with information about the chip.

**Parameters**

- [out] out\_info: structure to be filled

### Structures

**struct esp\_chip\_info\_t**

The structure represents information about the chip.

### Public Members

*esp\_chip\_model\_t* **model**

chip model, one of esp\_chip\_model\_t

uint32\_t **features**

bit mask of CHIP\_FEATURE\_x feature flags

uint8\_t **cores**

number of CPU cores

uint8\_t **revision**

chip revision number

### 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.

### Type Definitions

**typedef** void (\***shutdown\_handler\_t**) (void)

Shutdown handler type

### Enumerations

**enum esp\_mac\_type\_t**

Values:

**ESP\_MAC\_WIFI\_STA**

**ESP\_MAC\_WIFI\_SOFTAP**

**ESP\_MAC\_BT**

**ESP\_MAC\_ETH**

**enum esp\_reset\_reason\_t**

Reset reasons.

*Values:*

**ESP\_RST\_UNKNOWN**

Reset reason can not be determined.

**ESP\_RST\_POWERON**

Reset due to power-on event.

**ESP\_RST\_EXT**

Reset by external pin (not applicable for ESP32)

**ESP\_RST\_SW**

Software reset via esp\_restart.

**ESP\_RST\_PANIC**

Software reset due to exception/panic.

**ESP\_RST\_INT\_WDT**

Reset (software or hardware) due to interrupt watchdog.

**ESP\_RST\_TASK\_WDT**

Reset due to task watchdog.

**ESP\_RST\_WDT**

Reset due to other watchdogs.

**ESP\_RST\_DEEPSLEEP**

Reset after exiting deep sleep mode.

**ESP\_RST\_BROWNOUT**

Brownout reset (software or hardware)

**ESP\_RST\_SDIO**

Reset over SDIO.

**enum esp\_chip\_model\_t**

Chip models.

*Values:*

**CHIP\_ESP32 = 1**

ESP32.

### Header File

- [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.

**Note** 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.

**Return** 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)`

Code examples for this API section are provided in the [system](#) directory of ESP-IDF examples.

## 2.8 Project Configuration

### 2.8.1 Introduction

ESP-IDF uses [Kconfig](#) system to provide a compile-time project configuration mechanism. Kconfig 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.

### 2.8.2 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 `sd-kconfig`, 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.

(For the legacy GNU Make build system, the project configuration menu is opened with `make menuconfig`.)

### 2.8.3 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 must be created manually. It can contain all the options which matter for the given application. The format is the same as that of the `sdkconfig` file. 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](#).

### 2.8.4 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.
- Lines cannot be wrapped by backslash (because there is a bug in earlier versions of `conf-idf` which causes that Windows line endings are not recognized after a backslash).

### 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.

## 2.8.5 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. `confgen.py` 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. `confgen.py` 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.
3. `confgen.py` 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 `confgen.py`.

## 2.8.6 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.

### SDK tool configuration

Contains:

- `CONFIG_SDK_TOOLPREFIX`
- `CONFIG_SDK_PYTHON`
- `CONFIG_SDK_MAKE_WARN_UNDEFINED_VARIABLES`

### CONFIG\_SDK\_TOOLPREFIX

Compiler toolchain path/prefix

*Found in: SDK tool configuration*

The prefix/path that is used to call the toolchain. The default setting assumes a crosstool-ng gcc setup that is in your PATH.

### CONFIG\_SDK\_PYTHON

Python interpreter

*Found in: SDK tool configuration*

The executable name/path that is used to run python.

(Note: This option is used with the legacy GNU Make build system only.)

### CONFIG\_SDK\_MAKE\_WARN\_UNDEFINED\_VARIABLES

‘make’ warns on undefined variables

*Found in: SDK tool configuration*

Adds `-warn-undefined-variables` to MAKEFLAGS. This causes make to print a warning any time an undefined variable is referenced.

This option helps find places where a variable reference is misspelled or otherwise missing, but it can be unwanted if you have Makefiles which depend on undefined variables expanding to an empty string.

(Note: this option is used with the legacy GNU Make build system only.)

### Application manager

Contains:

- `CONFIG_APP_COMPILE_TIME_DATE`
- `CONFIG_APP_EXCLUDE_PROJECT_VER_VAR`
- `CONFIG_APP_EXCLUDE_PROJECT_NAME_VAR`
- `CONFIG_APP_RETRIEVE_LEN_ELF_SHA`

### 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.

### 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.

### 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.

### 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.

### Bootloader config

Contains:

- *CONFIG\_BOOTLOADER\_LOG\_LEVEL*
- *CONFIG\_BOOTLOADER\_SPI\_WP\_PIN*
- *CONFIG\_BOOTLOADER\_VDDSDIO\_BOOST*
- *CONFIG\_BOOTLOADER\_FACTORY\_RESET*
- *CONFIG\_BOOTLOADER\_APP\_TEST*
- *CONFIG\_BOOTLOADER\_HOLD\_TIME\_GPIO*
- *CONFIG\_BOOTLOADER\_WDT\_ENABLE*
- *CONFIG\_BOOTLOADER\_APP\_ROLLBACK\_ENABLE*

### 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\_WP\_PIN

SPI Flash WP Pin when customising pins via eFuse (read help)

*Found in: Bootloader config*

This value is ignored unless flash mode is set to QIO or QOUT \*and\* the SPI flash pins have been overridden by setting the eFuses SPI\_PAD\_CONFIG\_xxx.

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. That pin number is compiled into the bootloader instead.

The default value (GPIO 7) is correct for WP pin on ESP32-D2WD integrated flash.



### 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 MTDI 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 pulled low while device starts up. See settings below.

### 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. To trigger a factory reset, this GPIO must be pulled low on reset. Note that GPIO34-39 do not have an internal pullup and an external one must be provided.

### 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.

### 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.

### **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.

### **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.

### **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 ESP32\_RTC\_CLK\_SRC (INTERNAL\_RC or EXTERNAL\_CRYSTAL).

### **CONFIG\_BOOTLOADER\_WDT\_DISABLE\_IN\_USER\_CODE**

Allows RTC watchdog disable in user code

*Found in: Bootloader config > CONFIG\_BOOTLOADER\_WDT\_ENABLE*

If it is set, the client must itself reset or disable rtc\_wdt in their code (app\_main()). Otherwise rtc\_wdt will be disabled before calling app\_main function. Use function rtc\_wdt\_feed() for resetting counter of rtc\_wdt. Use function rtc\_wdt\_disable() for disabling rtc\_wdt.

### **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.

## 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.

## 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.

## 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 has a scheme with ota\_0 + ota\_1 (without factory).

## 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. This determines how many times the security version can be increased.

## 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 allow emulate read/write operations with eFuse secure version. It allow to test anti-rollback implementation without permanent write eFuse bits. In partition table should be exist this partition *emul\_efuse, data, 5, , 0x2000*.

## Security features

Contains:

- `CONFIG_SECURE_SIGNED_APPS_NO_SECURE_BOOT`
- `CONFIG_SECURE_BOOT_ENABLED`
- `CONFIG_SECURE_BOOT_BUILD_SIGNED_BINARIES`
- `CONFIG_SECURE_BOOT_VERIFICATION_KEY`
- `CONFIG_SECURE_BOOTLOADER_KEY_ENCODING`
- `CONFIG_SECURE_BOOT_INSECURE`
- `CONFIG_SECURE_FLASH_ENC_ENABLED`
- *Potentially insecure options*

### **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\_ON\_BOOT\_NO\_SECURE\_BOOT**

Bootloader verifies app signatures

*Found in: Security features > CONFIG\_SECURE\_SIGNED\_APPS\_NO\_SECURE\_BOOT*

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.

### **CONFIG\_SECURE\_SIGNED\_ON\_UPDATE\_NO\_SECURE\_BOOT**

Verify app signature on update

*Found in: Security features > CONFIG\_SECURE\_SIGNED\_APPS\_NO\_SECURE\_BOOT*

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.

### **CONFIG\_SECURE\_BOOT\_ENABLED**

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.

Refer to *Secure Boot* before enabling.

## CONFIG\_SECURE\_BOOTLOADER\_MODE

Secure bootloader mode

Found in: *Security features* > *CONFIG\_SECURE\_BOOT\_ENABLED*

### 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 in “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` (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.

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 *Secure Boot* for details.

### 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. This key is compiled into the bootloader and/or app, to verify app images.

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 *Secure Boot* before enabling.

### 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 `espefuse.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 *Secure Boot* before enabling.

### 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* before enabling.

### 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 UART bootloader to perform flash encryption operations

Select Release mode only for production or manufacturing. Once enabled you can not reflash using UART bootloader

Refer to [Secure Boot](#) and [Flash Encryption](#) 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\\_ALLOW\\_ROM\\_BASIC](#)
- [CONFIG\\_SECURE\\_BOOT\\_ALLOW\\_JTAG](#)
- [CONFIG\\_SECURE\\_BOOT\\_ALLOW\\_SHORT\\_APP\\_PARTITION](#)
- [CONFIG\\_SECURE\\_FLASH\\_UART\\_BOOTLOADER\\_ALLOW\\_ENC](#)
- [CONFIG\\_SECURE\\_FLASH\\_UART\\_BOOTLOADER\\_ALLOW\\_DEC](#)
- [CONFIG\\_SECURE\\_FLASH\\_UART\\_BOOTLOADER\\_ALLOW\\_CACHE](#)
- [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.

### **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.

### **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\_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.

### **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.

### **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.

### **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 `espefuse.py`.

### **Serial flasher config**

Contains:

- `CONFIG_ESPTOOLPY_PORT`
- `CONFIG_ESPTOOLPY_BAUD`
- `CONFIG_ESPTOOLPY_BAUD_OTHER_VAL`
- `CONFIG_ESPTOOLPY_COMPRESSED`
- `CONFIG_ESPTOOLPY_FLASHMODE`
- `CONFIG_ESPTOOLPY_FLASHFREQ`
- `CONFIG_ESPTOOLPY_FLASHSIZE`
- `CONFIG_ESPTOOLPY_FLASHSIZE_DETECT`
- `CONFIG_ESPTOOLPY_BEFORE`
- `CONFIG_ESPTOOLPY_AFTER`
- `CONFIG_ESPTOOLPY_MONITOR_BAUD`
- `CONFIG_ESPTOOLPY_MONITOR_BAUD_OTHER_VAL`



### CONFIG\_ESPTOOLPY\_PORT

Default serial port

*Found in: Serial flasher config*

The serial port that's connected to the ESP chip. This can be overridden by setting the ESPPORT environment variable.

This value is ignored when using the CMake-based build system or idf.py.

### CONFIG\_ESPTOOLPY\_BAUD

Default baud rate

*Found in: Serial flasher config*

Default baud rate to use while communicating with the ESP chip. Can be overridden by setting the ESPBAUD variable.

This value is ignored when using the CMake-based build system or idf.py.

**Available options:**

- 115200 baud (ESPTOOLPY\_BAUD\_115200B)
- 230400 baud (ESPTOOLPY\_BAUD\_230400B)
- 921600 baud (ESPTOOLPY\_BAUD\_921600B)
- 2Mbaud (ESPTOOLPY\_BAUD\_2MB)
- Other baud rate (ESPTOOLPY\_BAUD\_OTHER)

### CONFIG\_ESPTOOLPY\_BAUD\_OTHER\_VAL

Other baud rate value

*Found in: Serial flasher config*

### CONFIG\_ESPTOOLPY\_COMPRESSED

Use compressed upload

*Found in: Serial flasher config*

The flasher tool can send data compressed using zlib, letting the ROM on the ESP chip decompress it on the fly before flashing it. For most payloads, this should result in a speed increase.

### 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)

### CONFIG\_ESPTOOLPY\_FLASHFREQ

Flash SPI speed

*Found in: Serial flasher config*

The SPI flash frequency to be used.

**Available options:**

- 80 MHz (ESPTOOLPY\_FLASHFREQ\_80M)
- 40 MHz (ESPTOOLPY\_FLASHFREQ\_40M)
- 26 MHz (ESPTOOLPY\_FLASHFREQ\_26M)
- 20 MHz (ESPTOOLPY\_FLASHFREQ\_20M)

**CONFIG\_ESPTOOLPY\_FLASHSIZE**

Flash size

*Found in: [Serial flasher config](#)*

SPI flash size, in megabytes

**Available options:**

- 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)

**CONFIG\_ESPTOOLPY\_FLASHSIZE\_DETECT**

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.

**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)

## CONFIG\_ESPTOOLPY\_MONITOR\_BAUD

‘idf.py monitor’ baud rate

*Found in: Serial flasher config*

Baud rate to use when running ‘idf.py monitor’ or ‘make monitor’ to view serial output from a running chip.

Can override by setting the MONITORBAUD environment variable.

### Available options:

- 9600 bps (ESPTOOLPY\_MONITOR\_BAUD\_9600B)
- 57600 bps (ESPTOOLPY\_MONITOR\_BAUD\_57600B)
- 115200 bps (ESPTOOLPY\_MONITOR\_BAUD\_115200B)
- 230400 bps (ESPTOOLPY\_MONITOR\_BAUD\_230400B)
- 921600 bps (ESPTOOLPY\_MONITOR\_BAUD\_921600B)
- 2 Mbps (ESPTOOLPY\_MONITOR\_BAUD\_2MB)
- Custom baud rate (ESPTOOLPY\_MONITOR\_BAUD\_OTHER)

## CONFIG\_ESPTOOLPY\_MONITOR\_BAUD\_OTHER\_VAL

Custom baud rate value

*Found in: Serial flasher config*

## Partition Table

Contains:

- *CONFIG\_PARTITION\_TABLE\_TYPE*
- *CONFIG\_PARTITION\_TABLE\_CUSTOM\_FILENAME*
- *CONFIG\_PARTITION\_TABLE\_OFFSET*
- *CONFIG\_PARTITION\_TABLE\_MD5*

## 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. Otherwise it’s possible to create a new custom partition CSV for your application.

### Available options:

- Single factory app, no OTA (PARTITION\_TABLE\_SINGLE\_APP)
- Factory app, two OTA definitions (PARTITION\_TABLE\_TWO\_OTA)
- Custom partition table CSV (PARTITION\_TABLE\_CUSTOM)

## CONFIG\_PARTITION\_TABLE\_CUSTOM\_FILENAME

Custom partition CSV file

*Found in: Partition Table*

Name of the custom partition CSV filename. This path is evaluated relative to the project root directory.

## 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.

## 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.

## Compiler options

Contains:

- *CONFIG\_COMPILER\_OPTIMIZATION*
- *CONFIG\_COMPILER\_OPTIMIZATION\_ASSERTION\_LEVEL*
- *CONFIG\_COMPILER\_CXX\_EXCEPTIONS*
- *CONFIG\_COMPILER\_STACK\_CHECK\_MODE*
- *CONFIG\_COMPILER\_WARN\_WRITE\_STRINGS*
- *CONFIG\_COMPILER\_DISABLE\_GCC8\_WARNINGS*

## CONFIG\_COMPILER\_OPTIMIZATION

Optimization Level

*Found in: Compiler options*

This option sets compiler optimization level (gcc -O argument).

- for “Release” setting, -Os flag is added to CFLAGS.
- for “Debug” setting, -Og flag is added to CFLAGS.

“Release” with -Os produces smaller & faster compiled code but it may be harder to correlated code addresses to source files when debugging.

To add custom optimization settings, set CFLAGS and/or CPPFLAGS in project makefile, before including \$(IDF\_PATH)/make/project.mk. Note that custom optimization levels may be unsupported.

**Available options:**

- Debug (-Og) (COMPILER\_OPTIMIZATION\_LEVEL\_DEBUG)
- Release (-Os) (COMPILER\_OPTIMIZATION\_LEVEL\_RELEASE)

## 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\_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.

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.

## 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

**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 *`.

**CONFIG\_COMPILER\_DISABLE\_GCC8\_WARNINGS**

Disable new warnings introduced in GCC 6 - 8

*Found in: [Compiler options](#)*

Enable this option if using GCC 6 or newer, and wanting to disable warnings which don't appear with GCC 5.

**Component config**

Contains:

- [Application Level Tracing](#)
- [Bluetooth](#)
- [CONFIG\\_BLE\\_MESH](#)
- [Driver configurations](#)
- [eFuse Bit Manager](#)
- [ESP-TLS](#)
- [ESP32-specific](#)
- [Power Management](#)
- [ADC-Calibration](#)
- [Common ESP-related](#)
- [Ethernet](#)
- [Event Loop Library](#)
- [GDB Stub](#)
- [ESP HTTP client](#)
- [HTTP Server](#)
- [ESP HTTPS OTA](#)
- [ESP HTTPS server](#)
- [Wi-Fi](#)
- [PHY](#)
- [Core dump](#)
- [FAT Filesystem support](#)
- [Modbus configuration](#)
- [FreeRTOS](#)
- [Heap memory debugging](#)
- [libsodium](#)
- [Log output](#)
- [LWIP](#)

- [mbedTLS](#)
- [mDNS](#)
- [ESP-MQTT Configurations](#)
- [Newlib](#)
- [NVS](#)
- [OpenSSL](#)
- [PThreads](#)
- [SPI Flash driver](#)
- [SPIFFS Configuration](#)
- [TCP/IP Adapter](#)
- [Unity unit testing library](#)
- [Virtual file system](#)
- [Wear Levelling](#)
- [Wi-Fi Provisioning Manager](#)
- [Supplicant](#)

**Application Level Tracing** Contains:

- [CONFIG\\_ESP32\\_APPTRACE\\_DESTINATION](#)
- [CONFIG\\_ESP32\\_APPTRACE\\_ONPANIC\\_HOST\\_FLUSH\\_TMO](#)
- [CONFIG\\_ESP32\\_APPTRACE\\_POSTMORTEM\\_FLUSH\\_THRESH](#)
- [CONFIG\\_ESP32\\_APPTRACE\\_PENDING\\_DATA\\_SIZE\\_MAX](#)
- [FreeRTOS System View Tracing](#)
- [CONFIG\\_ESP32\\_GCOV\\_ENABLE](#)

### **CONFIG\_ESP32\_APPTRACE\_DESTINATION**

Data Destination

*Found in: [Component config](#) > [Application Level Tracing](#)*

Select destination for application trace: trace memory or none (to disable).

**Available options:**

- Trace memory (ESP32\_APPTRACE\_DEST\_TRAX)
- None (ESP32\_APPTRACE\_DEST\_NONE)

### **CONFIG\_ESP32\_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\_ESP32\_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\_ESP32\_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\_SYSVIEW\_ENABLE*

### **CONFIG\_SYSVIEW\_ENABLE**

SystemView Tracing Enable

*Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing*

Enables support for SEGGER SystemView tracing functionality.

### **CONFIG\_SYSVIEW\_TS\_SOURCE**

Timer to use as timestamp source

*Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing > CONFIG\_SYSVIEW\_ENABLE*

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) (SYSVIEW\_TS\_SOURCE\_CCOUNT)
- Timer 0, Group 0 (SYSVIEW\_TS\_SOURCE\_TIMER\_00)
- Timer 1, Group 0 (SYSVIEW\_TS\_SOURCE\_TIMER\_01)
- Timer 0, Group 1 (SYSVIEW\_TS\_SOURCE\_TIMER\_10)
- Timer 1, Group 1 (SYSVIEW\_TS\_SOURCE\_TIMER\_11)
- esp\_timer high resolution timer (SYSVIEW\_TS\_SOURCE\_ESP\_TIMER)

### **CONFIG\_SYSVIEW\_MAX\_TASKS**

Maximum supported tasks

*Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing > CONFIG\_SYSVIEW\_ENABLE*

Configures maximum supported tasks in sysview debug

### **CONFIG\_SYSVIEW\_BUF\_WAIT\_TMO**

Trace buffer wait timeout

*Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing > CONFIG\_SYSVIEW\_ENABLE*

Configures timeout (in us) to wait for free space in trace buffer. Set to -1 to wait forever and avoid lost events.

### **CONFIG\_SYSVIEW\_EVT\_OVERFLOW\_ENABLE**

Trace Buffer Overflow Event

*Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing > CONFIG\_SYSVIEW\_ENABLE*

Enables “Trace Buffer Overflow” event.



### **CONFIG\_SYSVIEW\_EVT\_ISR\_ENTER\_ENABLE**

ISR Enter Event

*Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing > CONFIG\_SYSVIEW\_ENABLE*

Enables “ISR Enter” event.

### **CONFIG\_SYSVIEW\_EVT\_ISR\_EXIT\_ENABLE**

ISR Exit Event

*Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing > CONFIG\_SYSVIEW\_ENABLE*

Enables “ISR Exit” event.

### **CONFIG\_SYSVIEW\_EVT\_ISR\_TO\_SCHEDULER\_ENABLE**

ISR Exit to Scheduler Event

*Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing > CONFIG\_SYSVIEW\_ENABLE*

Enables “ISR to Scheduler” event.

### **CONFIG\_SYSVIEW\_EVT\_TASK\_START\_EXEC\_ENABLE**

Task Start Execution Event

*Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing > CONFIG\_SYSVIEW\_ENABLE*

Enables “Task Start Execution” event.

### **CONFIG\_SYSVIEW\_EVT\_TASK\_STOP\_EXEC\_ENABLE**

Task Stop Execution Event

*Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing > CONFIG\_SYSVIEW\_ENABLE*

Enables “Task Stop Execution” event.

### **CONFIG\_SYSVIEW\_EVT\_TASK\_START\_READY\_ENABLE**

Task Start Ready State Event

*Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing > CONFIG\_SYSVIEW\_ENABLE*

Enables “Task Start Ready State” event.

### **CONFIG\_SYSVIEW\_EVT\_TASK\_STOP\_READY\_ENABLE**

Task Stop Ready State Event

*Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing > CONFIG\_SYSVIEW\_ENABLE*

Enables “Task Stop Ready State” event.

### **CONFIG\_SYSVIEW\_EVT\_TASK\_CREATE\_ENABLE**

Task Create Event

*Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing > CONFIG\_SYSVIEW\_ENABLE*

Enables “Task Create” event.

### **CONFIG\_SYSVIEW\_EVT\_TASK\_TERMINATE\_ENABLE**

Task Terminate Event

*Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing > CONFIG\_SYSVIEW\_ENABLE*

Enables “Task Terminate” event.

### **CONFIG\_SYSVIEW\_EVT\_IDLE\_ENABLE**

System Idle Event

*Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing > CONFIG\_SYSVIEW\_ENABLE*

Enables “System Idle” event.

### **CONFIG\_SYSVIEW\_EVT\_TIMER\_ENTER\_ENABLE**

Timer Enter Event

*Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing > CONFIG\_SYSVIEW\_ENABLE*

Enables “Timer Enter” event.

### **CONFIG\_SYSVIEW\_EVT\_TIMER\_EXIT\_ENABLE**

Timer Exit Event

*Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing > CONFIG\_SYSVIEW\_ENABLE*

Enables “Timer Exit” event.

### **CONFIG\_ESP32\_GCOV\_ENABLE**

GCOV to Host Enable

*Found in: Component config > Application Level Tracing*

Enables support for GCOV data transfer to host.

**Bluetooth** Contains:

- *CONFIG\_BT\_ENABLED*
- *Bluetooth controller*
- *CONFIG\_BT\_HOST*
- *Bluedroid 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.

**Bluetooth controller** Contains:

- `CONFIG_BTDM_CTRL_MODE`
- `CONFIG_BTDM_CTRL_BLE_MAX_CONN`
- `CONFIG_BTDM_CTRL_BR_EDR_MAX_ACL_CONN`
- `CONFIG_BTDM_CTRL_BR_EDR_MAX_SYNC_CONN`
- `CONFIG_BTDM_CTRL_BR_EDR_SCO_DATA_PATH`
- `CONFIG_BTDM_CTRL_PCM_ROLE_EDGE_CONFIG`
- `CONFIG_BTDM_CTRL_AUTO_LATENCY`
- `CONFIG_BTDM_CTRL_LEGACY_AUTH_VENDOR_EVT`
- `CONFIG_BTDM_CTRL_PINNED_TO_CORE_CHOICE`
- `CONFIG_BTDM_CTRL_HCI_MODE_CHOICE`
- *HCI UART(H4) Options*
- *MODEM SLEEP Options*
- `CONFIG_BTDM_BLE_SLEEP_CLOCK_ACCURACY`
- `CONFIG_BTDM_BLE_SCAN_DUPL`
- `CONFIG_BTDM_CTRL_FULL_SCAN_SUPPORTED`
- `CONFIG_BTDM_BLE_ADV_REPORT_FLOW_CTRL_SUPP`
- `CONFIG_BTDM_COEX_BT_OPTIONS`

## CONFIG\_BTDM\_CTRL\_MODE

Bluetooth controller mode (BR/EDR/BLE/DUALMODE)

*Found in: Component config > Bluetooth > Bluetooth controller*

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 > Bluetooth controller*

BLE maximum connections of bluetooth controller. Each connection uses 1KB static DRAM whenever the BT controller is enabled.

## CONFIG\_BTDM\_CTRL\_BR\_EDR\_MAX\_ACL\_CONN

BR/EDR ACL Max Connections

*Found in: Component config > Bluetooth > Bluetooth controller*

BR/EDR ACL maximum connections of bluetooth controller. Each connection uses 1.2KB static DRAM whenever the BT controller is enabled.

### CONFIG\_BTDM\_CTRL\_BR\_EDR\_MAX\_SYNC\_CONN

BR/EDR Sync(SCO/eSCO) Max Connections

*Found in: [Component config](#) > [Bluetooth](#) > [Bluetooth controller](#)*

BR/EDR Synchronize maximum connections of bluetooth controller. Each connection uses 2KB static DRAM whenever the BT controller is enabled.

### CONFIG\_BTDM\_CTRL\_BR\_EDR\_SCO\_DATA\_PATH

BR/EDR Sync(SCO/eSCO) default data path

*Found in: [Component config](#) > [Bluetooth](#) > [Bluetooth controller](#)*

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](#) > [Bluetooth controller](#)*

Contains:

- [CONFIG\\_BTDM\\_CTRL\\_PCM\\_ROLE](#)
- [CONFIG\\_BTDM\\_CTRL\\_PCM\\_POLAR](#)

### CONFIG\_BTDM\_CTRL\_PCM\_ROLE

PCM Role

*Found in: [Component config](#) > [Bluetooth](#) > [Bluetooth controller](#) > [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](#) > [Bluetooth controller](#) > [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 > Bluetooth controller*

BLE auto latency, used to enhance classic BT performance while classic BT and BLE are enabled at the same time.

### CONFIG\_BTDM\_CTRL\_LEGACY\_AUTH\_VENDOR\_EVT

Legacy Authentication Vendor Specific Event Enable

*Found in: Component config > Bluetooth > Bluetooth controller*

To protect from BIAS attack during Legacy authentication, Legacy authentication Vendor specific event should be enabled

### CONFIG\_BTDM\_CTRL\_PINNED\_TO\_CORE\_CHOICE

The cpu core which bluetooth controller run

*Found in: Component config > Bluetooth > Bluetooth controller*

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)

### CONFIG\_BTDM\_CTRL\_HCI\_MODE\_CHOICE

HCI mode

*Found in: Component config > Bluetooth > Bluetooth controller*

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\\_BT\\_HCI\\_UART\\_NO](#)
- [CONFIG\\_BT\\_HCI\\_UART\\_BAUDRATE](#)

### CONFIG\_BT\_HCI\_UART\_NO

UART Number for HCI

*Found in: Component config > Bluetooth > Bluetooth controller > HCI UART(H4) Options*

Uart number for HCI. The available uart is UART1 and UART2.

### CONFIG\_BT\_HCI\_UART\_BAUDRATE

UART Baudrate for HCI

*Found in: Component config > Bluetooth > Bluetooth controller > HCI UART(H4) Options*

UART Baudrate for HCI. Please use standard baudrate.

**MODEM SLEEP Options** Contains:

- [CONFIG\\_BTDM\\_MODEM\\_SLEEP](#)
- [CONFIG\\_BTDM\\_LOW\\_POWER\\_CLOCK](#)

**CONFIG\_BTDM\_MODEM\_SLEEP**

Bluetooth modem sleep

*Found in:* [Component config](#) > [Bluetooth](#) > [Bluetooth controller](#) > [MODEM SLEEP Options](#)

Enable/disable bluetooth controller low power mode.

**CONFIG\_BTDM\_MODEM\_SLEEP\_MODE**

Bluetooth Modem sleep mode

*Found in:* [Component config](#) > [Bluetooth](#) > [Bluetooth controller](#) > [MODEM SLEEP Options](#) > [CONFIG\\_BTDM\\_MODEM\\_SLEEP](#)

To select which strategy to use for modem sleep

**Available options:**

- **ORIG Mode**(sleep with low power clock) ([BTDM\\_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\\_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\_LOW\_POWER\_CLOCK**

Bluetooth low power clock

*Found in:* [Component config](#) > [Bluetooth](#) > [Bluetooth controller](#) > [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\\_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\\_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](#) > [Bluetooth controller](#)

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 oscillator(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](#) > [Bluetooth controller](#)

This select enables parameters setting of BLE scan duplicate.

**CONFIG\_BTDM\_SCAN\_DUPL\_TYPE**

## Scan Duplicate Type

Found in: [Component config](#) > [Bluetooth](#) > [Bluetooth controller](#) > [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](#) > [Bluetooth controller](#) > [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.

### **CONFIG\_BTDM\_BLE\_MESH\_SCAN\_DUPL\_EN**

Special duplicate scan mechanism for BLE Mesh scan

*Found in: Component config > Bluetooth > Bluetooth controller > CONFIG\_BTDM\_BLE\_SCAN\_DUPL*

This enables the BLE scan duplicate for special BLE Mesh scan.

### **CONFIG\_BTDM\_MESH\_DUPL\_SCAN\_CACHE\_SIZE**

Maximum number of Mesh adv packets in scan duplicate filter

*Found in: Component config > Bluetooth > Bluetooth controller > 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.

### **CONFIG\_BTDM\_CTRL\_FULL\_SCAN\_SUPPORTED**

BLE full scan feature supported

*Found in: Component config > Bluetooth > Bluetooth controller*

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.

### **CONFIG\_BTDM\_BLE\_ADV\_REPORT\_FLOW\_CTRL\_SUPP**

BLE adv report flow control supported

*Found in: Component config > Bluetooth > Bluetooth controller*

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.

### **CONFIG\_BTDM\_BLE\_ADV\_REPORT\_FLOW\_CTRL\_NUM**

BLE adv report flow control number

*Found in: Component config > Bluetooth > Bluetooth controller > 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.

### **CONFIG\_BTDM\_BLE\_ADV\_REPORT\_DISCARD\_THRSHOLD**

BLE adv lost event threshold value

*Found in: Component config > Bluetooth > Bluetooth controller > 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\_THRSHOLD* to a small value or printf every adv lost event, it may cause adv packets lost more.



## CONFIG\_BTDM\_COEX\_BT\_OPTIONS

Coexistence Bluetooth Side Options

*Found in: Component config > Bluetooth > Bluetooth controller*

Options of Bluetooth Side of WiFi and bluetooth coexistence.

Contains:

- [CONFIG\\_BTDM\\_COEX\\_BLE\\_ADV\\_HIGH\\_PRIORITY](#)

## CONFIG\_BTDM\_COEX\_BLE\_ADV\_HIGH\_PRIORITY

Improve BLE ADV priority for WiFi & BLE coexistence

*Found in: Component config > Bluetooth > Bluetooth controller > CONFIG\_BTDM\_COEX\_BT\_OPTIONS*

Improve BLE ADV coexistence priority to make it better performance. For example, BLE mesh need to enable this option to improve BLE adv performance.

## CONFIG\_BT\_HOST

Bluetooth Host

*Found in: Component config > Bluetooth*

This helps to choose Bluetooth host stack

### Available options:

- **Bluedroid - Dual-mode (BT\_BLUEDROID\_ENABLED)**  
This option is recommended for classic Bluetooth or for dual-mode usecases
- **NimBLE - BLE only (BT\_NIMBLE\_ENABLED)**  
This option is recommended for BLE only usecases to save on memory
- **Controller Only (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).

### Bluedroid Options

 Contains:

- [CONFIG\\_BT\\_BTC\\_TASK\\_STACK\\_SIZE](#)
- [CONFIG\\_BT\\_BLUEDROID\\_PINNED\\_TO\\_CORE\\_CHOICE](#)
- [CONFIG\\_BT\\_BTU\\_TASK\\_STACK\\_SIZE](#)
- [CONFIG\\_BT\\_BLUEDROID\\_MEM\\_DEBUG](#)
- [CONFIG\\_BT\\_CLASSIC\\_ENABLED](#)
- [CONFIG\\_BT\\_HFP\\_WBS\\_ENABLE](#)
- [CONFIG\\_BT\\_SSP\\_ENABLED](#)
- [CONFIG\\_BT\\_BLE\\_ENABLED](#)
- [CONFIG\\_BT\\_STACK\\_NO\\_LOG](#)
- [BT\\_DEBUG\\_LOG\\_LEVEL](#)
- [CONFIG\\_BT\\_ACL\\_CONNECTIONS](#)
- [CONFIG\\_BT\\_ALLOCATION\\_FROM\\_SPIRAM\\_FIRST](#)
- [CONFIG\\_BT\\_BLE\\_DYNAMIC\\_ENV\\_MEMORY](#)
- [CONFIG\\_BT\\_BLE\\_HOST\\_QUEUE\\_CONG\\_CHECK](#)
- [CONFIG\\_BT\\_BLE\\_ACT\\_SCAN\\_REP\\_ADV\\_SCAN](#)
- [CONFIG\\_BT\\_BLE\\_ESTAB\\_LINK\\_CONN\\_TOUT](#)

## 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

### **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

### **CONFIG\_BT\_BLUEDROID\_MEM\_DEBUG**

Bluedroid memory debug

*Found in: [Component config](#) > [Bluetooth](#) > [Bluedroid Options](#)*

Bluedroid memory debug

### **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

### **CONFIG\_BT\_A2DP\_ENABLE**

A2DP

*Found in: [Component config](#) > [Bluetooth](#) > [Bluedroid Options](#) > [CONFIG\\_BT\\_CLASSIC\\_ENABLED](#)*

Advanced Audio Distribution Profile

### **CONFIG\_BT\_SPP\_ENABLED**

SPP

*Found in: [Component config](#) > [Bluetooth](#) > [Bluedroid Options](#) > [CONFIG\\_BT\\_CLASSIC\\_ENABLED](#)*

This enables the Serial Port Profile

### **CONFIG\_BT\_HFP\_ENABLE**

Hands Free/Handset Profile

*Found in: [Component config](#) > [Bluetooth](#) > [Bluedroid Options](#) > [CONFIG\\_BT\\_CLASSIC\\_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)

### 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.

### 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

### CONFIG\_BT\_BLE\_ENABLED

Bluetooth Low Energy

*Found in: [Component config](#) > [Bluetooth](#) > [Bluedroid Options](#)*

This enables Bluetooth Low Energy

### 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

### 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.

## 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

## 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

## 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.

## CONFIG\_BT\_SMP\_SLAVE\_CON\_PARAMS\_UPD\_ENABLE

Slave enable connection parameters update during pairing

*Found in: [Component config](#) > [Bluetooth](#) > [Bluedroid 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.

## CONFIG\_BT\_STACK\_NO\_LOG

Disable BT debug logs (minimize bin size)

*Found in: [Component config](#) > [Bluetooth](#) > [Bluedroid Options](#)*

This select can save the rodata code size

**BT DEBUG LOG LEVEL** Contains:

- `CONFIG_BT_LOG_HCI_TRACE_LEVEL`
- `CONFIG_BT_LOG_BTM_TRACE_LEVEL`
- `CONFIG_BT_LOG_L2CAP_TRACE_LEVEL`
- `CONFIG_BT_LOG_RFCOMM_TRACE_LEVEL`
- `CONFIG_BT_LOG_SDP_TRACE_LEVEL`
- `CONFIG_BT_LOG_GAP_TRACE_LEVEL`
- `CONFIG_BT_LOG_BNEP_TRACE_LEVEL`
- `CONFIG_BT_LOG_PAN_TRACE_LEVEL`
- `CONFIG_BT_LOG_A2D_TRACE_LEVEL`
- `CONFIG_BT_LOG_AVDT_TRACE_LEVEL`
- `CONFIG_BT_LOG_AVCT_TRACE_LEVEL`
- `CONFIG_BT_LOG_AVRC_TRACE_LEVEL`
- `CONFIG_BT_LOG_MCA_TRACE_LEVEL`
- `CONFIG_BT_LOG_HID_TRACE_LEVEL`
- `CONFIG_BT_LOG_APPL_TRACE_LEVEL`
- `CONFIG_BT_LOG_GATT_TRACE_LEVEL`
- `CONFIG_BT_LOG_SMP_TRACE_LEVEL`
- `CONFIG_BT_LOG_BTIF_TRACE_LEVEL`
- `CONFIG_BT_LOG_BTC_TRACE_LEVEL`
- `CONFIG_BT_LOG_OSI_TRACE_LEVEL`
- `CONFIG_BT_LOG_BLUFI_TRACE_LEVEL`

**CONFIG\_BT\_LOG\_HCI\_TRACE\_LEVEL**

HCI layer

*Found in: Component config > Bluetooth > Blueandroid 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)
- VERBOSE (BT\_LOG\_HCI\_TRACE\_LEVEL\_VERBOSE)

**CONFIG\_BT\_LOG\_BTM\_TRACE\_LEVEL**

BTM layer

*Found in: Component config > Bluetooth > Blueandroid 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\_LEVEL\_VERBOSE)

### 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)
- 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)
- 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](#) > [Bluedroid 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](#) > [Bluedroid 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](#) > [Bluedroid 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

**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

**CONFIG\_BT\_BLE\_DYNAMIC\_ENV\_MEMORY**

Use dynamic memory allocation in BT/BLE stack

*Found in: [Component config](#) > [Bluetooth](#) > [Bluedroid Options](#)*

This select can make the allocation of memory will become more flexible

**CONFIG\_BT\_BLE\_HOST\_QUEUE\_CONG\_CHECK**

BLE queue congestion check

*Found in: [Component config](#) > [Bluetooth](#) > [Bluedroid 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.

**CONFIG\_BT\_BLE\_ACT\_SCAN\_REP\_ADV\_SCAN**

Report adv data and scan response individually when BLE active scan

*Found in: [Component config](#) > [Bluetooth](#) > [Bluedroid Options](#)*

Originally, when doing BLE active scan, Bluedroid will not report adv to application layer until receive scan response. This option is used to disable the behavior. When enable this option, Bluedroid will report adv data or scan response to application layer immediately.

# Memory reserved at start of DRAM for Bluetooth stack

**CONFIG\_BT\_BLE\_ESTAB\_LINK\_CONN\_TOUT**

Timeout of BLE connection establishment

*Found in: [Component config](#) > [Bluetooth](#) > [Bluedroid 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.

**NimBLE Options** Contains:

- `CONFIG_BT_NIMBLE_MEM_ALLOC_MODE`
- `CONFIG_BT_NIMBLE_MAX_CONNECTIONS`
- `CONFIG_BT_NIMBLE_MAX_BONDS`
- `CONFIG_BT_NIMBLE_MAX_CCCDS`
- `CONFIG_BT_NIMBLE_L2CAP_COC_MAX_NUM`
- `CONFIG_BT_NIMBLE_PINNED_TO_CORE_CHOICE`
- `CONFIG_BT_NIMBLE_TASK_STACK_SIZE`
- `CONFIG_BT_NIMBLE_ROLE_CENTRAL`
- `CONFIG_BT_NIMBLE_ROLE_PERIPHERAL`
- `CONFIG_BT_NIMBLE_ROLE_BROADCASTER`
- `CONFIG_BT_NIMBLE_ROLE_OBSERVER`
- `CONFIG_BT_NIMBLE_NVS_PERSIST`
- `CONFIG_BT_NIMBLE_SM_LEGACY`
- `CONFIG_BT_NIMBLE_SM_SC`
- `CONFIG_BT_NIMBLE_DEBUG`
- `CONFIG_BT_NIMBLE_SM_SC_DEBUG_KEYS`
- `CONFIG_BT_NIMBLE_SVC_GAP_DEVICE_NAME`
- `CONFIG_BT_NIMBLE_GAP_DEVICE_NAME_MAX_LEN`
- `CONFIG_BT_NIMBLE_ATT_PREFERRED_MTU`
- `CONFIG_BT_NIMBLE_SVC_GAP_APPEARANCE`
- `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_MSYS1_BLOCK_COUNT`
- `CONFIG_BT_NIMBLE_HS_FLOW_CTRL`
- `CONFIG_BT_NIMBLE_RPA_TIMEOUT`
- `CONFIG_BT_NIMBLE_MESH`
- `CONFIG_BT_NIMBLE_CRYPTO_STACK_MBEDTLS`

**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

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.

**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`)

**CONFIG\_BT\_NIMBLE\_MAX\_CONNECTIONS**

Maximum number of concurrent connections

*Found in: Component config > Bluetooth > NimBLE Options*

Defines maximum number of concurrent BLE connections

### **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

### **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

### **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

### **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\_TASK\_STACK\_SIZE**

NimBLE Host task stack size

*Found in: [Component config](#) > [Bluetooth](#) > [NimBLE Options](#)*

This configures stack size of NimBLE host task

### **CONFIG\_BT\_NIMBLE\_ROLE\_CENTRAL**

Enable BLE Central role

*Found in: [Component config](#) > [Bluetooth](#) > [NimBLE Options](#)*

### **CONFIG\_BT\_NIMBLE\_ROLE\_PERIPHERAL**

Enable BLE Peripheral role

*Found in: [Component config](#) > [Bluetooth](#) > [NimBLE Options](#)*

### **CONFIG\_BT\_NIMBLE\_ROLE\_BROADCASTER**

Enable BLE Broadcaster role

*Found in: [Component config](#) > [Bluetooth](#) > [NimBLE Options](#)*

### **CONFIG\_BT\_NIMBLE\_ROLE\_OBSERVER**

Enable BLE Observer role

*Found in: [Component config](#) > [Bluetooth](#) > [NimBLE Options](#)*

### **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

### **CONFIG\_BT\_NIMBLE\_SM\_LEGACY**

Security manager legacy pairing

*Found in: [Component config](#) > [Bluetooth](#) > [NimBLE Options](#)*

Enable security manager legacy pairing

### **CONFIG\_BT\_NIMBLE\_SM\_SC**

Security manager secure connections (4.2)

*Found in: [Component config](#) > [Bluetooth](#) > [NimBLE Options](#)*

Enable security manager secure connections

### **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

### **CONFIG\_BT\_NIMBLE\_SM\_SC\_DEBUG\_KEYS**

Use predefined public-private key pair

*Found in: [Component config](#) > [Bluetooth](#) > [NimBLE Options](#)*

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.

### **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()`

### **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

### **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()`

### **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

### **CONFIG\_BT\_NIMBLE\_ACL\_BUF\_COUNT**

ACL Buffer count

*Found in: [Component config](#) > [Bluetooth](#) > [NimBLE Options](#)*

The number of ACL data buffers.

### **CONFIG\_BT\_NIMBLE\_ACL\_BUF\_SIZE**

ACL Buffer size

*Found in: [Component config](#) > [Bluetooth](#) > [NimBLE Options](#)*

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)

### **CONFIG\_BT\_NIMBLE\_HCI\_EVT\_BUF\_SIZE**

HCI Event Buffer size

*Found in: [Component config](#) > [Bluetooth](#) > [NimBLE Options](#)*

This is the size of each HCI event buffer in bytes

### **CONFIG\_BT\_NIMBLE\_HCI\_EVT\_HI\_BUF\_COUNT**

High Priority HCI Event Buffer count

*Found in: [Component config](#) > [Bluetooth](#) > [NimBLE Options](#)*

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

### **CONFIG\_BT\_NIMBLE\_HCI\_EVT\_LO\_BUF\_COUNT**

Low Priority HCI Event Buffer count

*Found in: [Component config](#) > [Bluetooth](#) > [NimBLE Options](#)*

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

### **CONFIG\_BT\_NIMBLE\_MSYS1\_BLOCK\_COUNT**

MSYS\_1 Block Count

*Found in: [Component config](#) > [Bluetooth](#) > [NimBLE Options](#)*

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.

### **CONFIG\_BT\_NIMBLE\_HS\_FLOW\_CTRL**

Enable Host Flow control

*Found in: [Component config](#) > [Bluetooth](#) > [NimBLE Options](#)*

Enable Host Flow control

### **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

### **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

### **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

### **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

### **CONFIG\_BT\_NIMBLE\_MESH**

Enable BLE mesh functionality

*Found in: [Component config](#) > [Bluetooth](#) > [NimBLE Options](#)*

Enable BLE Mesh functionality

Contains:

- [CONFIG\\_BT\\_NIMBLE\\_MESH\\_PROXY](#)
- [CONFIG\\_BT\\_NIMBLE\\_MESH\\_PROV](#)
- [CONFIG\\_BT\\_NIMBLE\\_MESH\\_GATT\\_PROXY](#)
- [CONFIG\\_BT\\_NIMBLE\\_MESH\\_RELAY](#)



- [CONFIG\\_BT\\_NIMBLE\\_MESH\\_LOW\\_POWER](#)
- [CONFIG\\_BT\\_NIMBLE\\_MESH\\_FRIEND](#)
- [CONFIG\\_BT\\_NIMBLE\\_MESH\\_DEVICE\\_NAME](#)

### **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 NIMBLE\_MESH\_GATT\_PROXY is set

### **CONFIG\_BT\_NIMBLE\_MESH\_PROV**

Enable BLE mesh provisioning

*Found in: [Component config](#) > [Bluetooth](#) > [NimBLE Options](#) > [CONFIG\\_BT\\_NIMBLE\\_MESH](#)*

Enable mesh provisioning

### **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

### **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

### **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

### **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

### **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

### **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

### **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

### **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.

### **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:

- *CONFIG\_BLE\_MESH\_HCI\_5\_0*
- *CONFIG\_BLE\_MESH\_USE\_DUPLICATE\_SCAN*
- *CONFIG\_BLE\_MESH\_MEM\_ALLOC\_MODE*
- *CONFIG\_BLE\_MESH\_FREERTOS\_STATIC\_ALLOC*
- *CONFIG\_BLE\_MESH\_DEINIT*
- *BLE Mesh and BLE coexistence support*
- *CONFIG\_BLE\_MESH\_FAST\_PROV*
- *CONFIG\_BLE\_MESH\_NODE*
- *CONFIG\_BLE\_MESH\_PROVISIONER*
- *CONFIG\_BLE\_MESH\_PROV*
- *CONFIG\_BLE\_MESH\_PB\_ADV*
- *CONFIG\_BLE\_MESH\_PB\_GATT*
- *CONFIG\_BLE\_MESH\_PROXY*
- *CONFIG\_BLE\_MESH\_GATT\_PROXY\_SERVER*
- *CONFIG\_BLE\_MESH\_GATT\_PROXY\_CLIENT*
- *CONFIG\_BLE\_MESH\_SETTINGS*
- *CONFIG\_BLE\_MESH\_SUBNET\_COUNT*
- *CONFIG\_BLE\_MESH\_APP\_KEY\_COUNT*
- *CONFIG\_BLE\_MESH\_MODEL\_KEY\_COUNT*
- *CONFIG\_BLE\_MESH\_MODEL\_GROUP\_COUNT*
- *CONFIG\_BLE\_MESH\_LABEL\_COUNT*
- *CONFIG\_BLE\_MESH\_CRPL*
- *CONFIG\_BLE\_MESH\_MSG\_CACHE\_SIZE*
- *CONFIG\_BLE\_MESH\_ADV\_BUF\_COUNT*
- *CONFIG\_BLE\_MESH\_IVU\_DIVIDER*
- *CONFIG\_BLE\_MESH\_TX\_SEG\_MSG\_COUNT*
- *CONFIG\_BLE\_MESH\_RX\_SEG\_MSG\_COUNT*

- `CONFIG_BLE_MESH_RX_SDU_MAX`
- `CONFIG_BLE_MESH_TX_SEG_MAX`
- `CONFIG_BLE_MESH_RELAY`
- `CONFIG_BLE_MESH_LOW_POWER`
- `CONFIG_BLE_MESH_FRIEND`
- `CONFIG_BLE_MESH_NO_LOG`
- `BLE Mesh STACK DEBUG LOG LEVEL`
- `BLE Mesh NET BUF DEBUG LOG LEVEL`
- `CONFIG_BLE_MESH_CLIENT_MSG_TIMEOUT`
- *Support for BLE Mesh Foundation models*
- *Support for BLE Mesh Client/Server models*
- `CONFIG_BLE_MESH_IV_UPDATE_TEST`
- *BLE Mesh specific test option*

### **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.

### **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” .

### **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
- External SPIRAM memory only
- Either internal or external memory based on default malloc() behavior in ESP-IDF

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.

#### **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.

### **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) 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.

### 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.

### 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.

### BLE Mesh and BLE coexistence support

 Contains:

- [CONFIG\\_BLE\\_MESH\\_SUPPORT\\_BLE\\_ADV](#)
- [CONFIG\\_BLE\\_MESH\\_SUPPORT\\_BLE\\_SCAN](#)

### 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.

### 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.

### 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.

### 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.

### 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).

### 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.

### CONFIG\_BLE\_MESH\_PBA\_SAME\_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.

### **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.

### **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.

### **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.

### **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.

### **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.

### **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.

### **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.

### **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.

### **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.

### **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.

### **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.

### 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.

### 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.

### 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.

### 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.

### 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.



### 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.

### 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 exist 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.

### 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.

### 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.

### 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.

### **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.

### **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.

### **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.

### **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.

### **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 can be supported by a node.

### **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 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.

### 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.

### 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.

### 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).

### 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.

### 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.

### **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.

### **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.

### **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.

### **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.

### **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.

### 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.

### 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 unprovisioned 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.

### 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.

### 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.

### 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.

### 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.

### 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  $N = 128$ .

### CONFIG\_BLE\_MESH\_LPN\_RECV\_DELAY

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.

### 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.

### 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.

### 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.

### **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.

### **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.

### **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.

### **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.

### **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.

### **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.

### 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.

### 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)

### 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.



**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.

**Support for BLE Mesh Client/Server models** Contains:

- *CONFIG\_BLE\_MESH\_GENERIC\_ONOFF\_CLI*
- *CONFIG\_BLE\_MESH\_GENERIC\_LEVEL\_CLI*
- *CONFIG\_BLE\_MESH\_GENERIC\_DEF\_TRANS\_TIME\_CLI*
- *CONFIG\_BLE\_MESH\_GENERIC\_POWER\_ONOFF\_CLI*
- *CONFIG\_BLE\_MESH\_GENERIC\_POWER\_LEVEL\_CLI*
- *CONFIG\_BLE\_MESH\_GENERIC\_BATTERY\_CLI*
- *CONFIG\_BLE\_MESH\_GENERIC\_LOCATION\_CLI*
- *CONFIG\_BLE\_MESH\_GENERIC\_PROPERTY\_CLI*
- *CONFIG\_BLE\_MESH\_SENSOR\_CLI*
- *CONFIG\_BLE\_MESH\_TIME\_CLI*
- *CONFIG\_BLE\_MESH\_SCENE\_CLI*
- *CONFIG\_BLE\_MESH\_SCHEDULER\_CLI*
- *CONFIG\_BLE\_MESH\_LIGHT\_LIGHTNESS\_CLI*
- *CONFIG\_BLE\_MESH\_LIGHT\_CTL\_CLI*
- *CONFIG\_BLE\_MESH\_LIGHT\_HSL\_CLI*
- *CONFIG\_BLE\_MESH\_LIGHT\_XYL\_CLI*
- *CONFIG\_BLE\_MESH\_LIGHT\_LC\_CLI*
- *CONFIG\_BLE\_MESH\_GENERIC\_SERVER*
- *CONFIG\_BLE\_MESH\_SENSOR\_SERVER*
- *CONFIG\_BLE\_MESH\_TIME\_SCENE\_SERVER*
- *CONFIG\_BLE\_MESH\_LIGHTING\_SERVER*

**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\_MESH\_SENSOR\_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.

### **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.

### **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.

### **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.

### **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.

**BLE Mesh specific test option** Contains:

- *CONFIG\_BLE\_MESH\_SELF\_TEST*
- *CONFIG\_BLE\_MESH\_SHELL*
- *CONFIG\_BLE\_MESH\_DEBUG*

### **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.

### **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 enters 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.

### **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.

### **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.

### **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.

### **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:

- *ADC configuration*
- *SPI configuration*
- *CAN Configuration*
- *UART configuration*

**ADC configuration** Contains:

- *CONFIG\_ADC\_FORCE\_XPD\_FSM*
- *CONFIG\_ADC\_DISABLE\_DAC*

### **CONFIG\_ADC\_FORCE\_XPD\_FSM**

Use the FSM to control ADC power

*Found in: Component config > Driver configurations > ADC configuration*

ADC power can be controlled by the FSM instead of software. This allows the ADC to be shut off when it is not working leading to lower power consumption. However using the FSM control ADC power will increase the noise of ADC.

### **CONFIG\_ADC\_DISABLE\_DAC**

Disable DAC when ADC2 is used on GPIO 25 and 26

*Found in: Component config > Driver configurations > 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.

**SPI configuration** Contains:

- *CONFIG\_SPI\_MASTER\_IN\_IRAM*
- *CONFIG\_SPI\_MASTER\_ISR\_IN\_IRAM*
- *CONFIG\_SPI\_SLAVE\_IN\_IRAM*
- *CONFIG\_SPI\_SLAVE\_ISR\_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.

### **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.

### **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.

### **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.

**CAN Configuration** Contains:

- [CONFIG\\_CAN\\_ISR\\_IN\\_IRAM](#)

### **CONFIG\_CAN\_ISR\_IN\_IRAM**

Place CAN ISR function into IRAM

*Found in: [Component config](#) > [Driver configurations](#) > [CAN Configuration](#)*

Place the CAN 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 `CAN_ALERT_AND_LOG` flag) will have no effect.

**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.

**eFuse Bit Manager** Contains:

- [CONFIG\\_EFUSE\\_CUSTOM\\_TABLE](#)
- [CONFIG\\_EFUSE\\_VIRTUAL](#)
- [CONFIG\\_EFUSE\\_CODE\\_SCHEME\\_SELECTOR](#)



### 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.

### 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.

### CONFIG\_EFUSE\_VIRTUAL

Simulate eFuse operations in RAM

*Found in: Component config > eFuse Bit Manager*

All read and writes operations are redirected to RAM instead of eFuse registers. If this option is set, all permanent changes (via eFuse) are disabled. Log output will state changes which would be applied, but they will not be.

### 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\_SERVER*

### CONFIG\_ESP\_TLS\_SERVER

Enable ESP-TLS Server

*Found in: Component config > ESP-TLS*

Enable support for creating server side SSL/TLS session

### ESP32-specific

 Contains:

- *CONFIG\_ESP32\_REV\_MIN*
- *CONFIG\_ESP32\_DEFAULT\_CPU\_FREQ\_MHZ*
- *CONFIG\_ESP32\_SPIRAM\_SUPPORT*
- *CONFIG\_ESP32\_TRAX*
- *CONFIG\_ESP32\_UNIVERSAL\_MAC\_ADDRESSES*
- *CONFIG\_ESP32\_ULP\_COPROC\_ENABLED*
- *CONFIG\_ESP32\_PANIC*
- *CONFIG\_ESP32\_DEBUG\_OCDAWARE*
- *CONFIG\_ESP32\_DEBUG\_STUBS\_ENABLE*

- `CONFIG_ESP32_BROWNOUT_DET`
- `CONFIG_ESP32_REDUCE_PHY_TX_POWER`
- `CONFIG_ESP32_TIME_SYSCALL`
- `CONFIG_ESP32_RTC_CLK_SRC`
- `CONFIG_ESP32_RTC_EXT_CRYST_ADDIT_CURRENT`
- `CONFIG_ESP32_RTC_CLK_CAL_CYCLES`
- `CONFIG_ESP32_RTC_XTAL_CAL_RETRY`
- `CONFIG_ESP32_RTC_XTAL_BOOTSTRAP_CYCLES`
- `CONFIG_ESP32_DEEP_SLEEP_WAKEUP_DELAY`
- `CONFIG_ESP32_XTAL_FREQ_SEL`
- `CONFIG_ESP32_DISABLE_BASIC_ROM_CONSOLE`
- `CONFIG_ESP32_NO_BLOBS`
- `CONFIG_ESP32_COMPATIBLE_PRE_V2_1_BOOTLOADERS`
- `CONFIG_ESP32_COMPATIBLE_PRE_V3_1_BOOTLOADERS`
- `CONFIG_ESP32_RTCDATA_IN_FAST_MEM`
- `CONFIG_ESP32_USE_FIXED_STATIC_RAM_SIZE`
- `CONFIG_ESP32_DPORT_DIS_INTERRUPT_LVL`

### **CONFIG\_ESP32\_REV\_MIN**

Minimum Supported ESP32 Revision

*Found in: [Component config](#) > [ESP32-specific](#)*

Minimum revision that ESP-IDF would support. ESP-IDF performs different strategy on different esp32 revision.

**Available options:**

- Rev 0 (`ESP32_REV_MIN_0`)
- Rev 1 (`ESP32_REV_MIN_1`)
- Rev 2 (`ESP32_REV_MIN_2`)
- Rev 3 (`ESP32_REV_MIN_3`)

### **CONFIG\_ESP32\_DEFAULT\_CPU\_FREQ\_MHZ**

CPU frequency

*Found in: [Component config](#) > [ESP32-specific](#)*

CPU frequency to be set on application startup.

**Available options:**

- 80 MHz (`ESP32_DEFAULT_CPU_FREQ_80`)
- 160 MHz (`ESP32_DEFAULT_CPU_FREQ_160`)
- 240 MHz (`ESP32_DEFAULT_CPU_FREQ_240`)

### **CONFIG\_ESP32\_SPIRAM\_SUPPORT**

Support for external, SPI-connected RAM

*Found in: [Component config](#) > [ESP32-specific](#)*

This enables support for an external SPI RAM chip, connected in parallel with the main SPI flash chip.

**SPI RAM config** Contains:

- `CONFIG_SPIRAM_BOOT_INIT`
- `CONFIG_SPIRAM_USE`
- `CONFIG_SPIRAM_TYPE`
- `CONFIG_SPIRAM_SPEED`
- `CONFIG_SPIRAM_MEMTEST`

- `CONFIG_SPIRAM_CACHE_WORKAROUND`
- `SPIRAM cache workaround debugging`
- `CONFIG_SPIRAM_BANKSWITCH_ENABLE`
- `CONFIG_SPIRAM_MALLOC_ALWAYSINTERNAL`
- `CONFIG_SPIRAM_TRY_ALLOCATE_WIFI_LWIP`
- `CONFIG_SPIRAM_MALLOC_RESERVE_INTERNAL`
- `CONFIG_SPIRAM_ALLOW_STACK_EXTERNAL_MEMORY`
- `CONFIG_SPIRAM_ALLOW_BSS_SEG_EXTERNAL_MEMORY`
- `CONFIG_SPIRAM_OCCUPY_SPI_HOST`
- `PSRAM clock and cs IO for ESP32-D0WD`
- `PSRAM clock and cs IO for ESP32-D2WD`
- `PSRAM clock and cs IO for ESP32-PICO`
- `CONFIG_SPIRAM_SPIWP_SD3_PIN`
- `CONFIG_SPIRAM_2T_MODE`

### CONFIG\_SPIRAM\_BOOT\_INIT

Initialize SPI RAM when booting the ESP32

*Found in: Component config > ESP32-specific > CONFIG\_ESP32\_SPIRAM\_SUPPORT > 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.

### CONFIG\_SPIRAM\_IGNORE\_NOTFOUND

Ignore PSRAM when not found

*Found in: Component config > ESP32-specific > CONFIG\_ESP32\_SPIRAM\_SUPPORT > 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 ESP32 panic. If this is enabled, the ESP32 will keep on running but will not add the (non-existing) RAM to any allocator.

### CONFIG\_SPIRAM\_USE

SPI RAM access method

*Found in: Component config > ESP32-specific > CONFIG\_ESP32\_SPIRAM\_SUPPORT > SPI RAM config*

The SPI RAM can be accessed in multiple methods: by just having it available as an unmanaged memory region in the ESP32 memory map, by integrating it in the ESP32s 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 ESP32 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\_TYPE

Type of SPI RAM chip in use

*Found in: Component config > ESP32-specific > CONFIG\_ESP32\_SPIRAM\_SUPPORT > SPI RAM config*

#### Available options:

- Auto-detect (`SPIRAM_TYPE_AUTO`)
- ESP-PSRAM32 or IS25WP032 (`SPIRAM_TYPE_ESPPSRAM32`)
- ESP-PSRAM64 or LY68L6400 (`SPIRAM_TYPE_ESPPSRAM64`)

## CONFIG\_SPIRAM\_SPEED

Set RAM clock speed

*Found in: [Component config](#) > [ESP32-specific](#) > [CONFIG\\_ESP32\\_SPIRAM\\_SUPPORT](#) > [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\_MEMTEST

Run memory test on SPI RAM initialization

*Found in: [Component config](#) > [ESP32-specific](#) > [CONFIG\\_ESP32\\_SPIRAM\\_SUPPORT](#) > [SPI RAM config](#)*

Runs a rudimentary memory test on initialization. Aborts when memory test fails. Disable this for slightly faster startup.

## CONFIG\_SPIRAM\_CACHE\_WORKAROUND

Enable workaround for bug in SPI RAM cache for Rev1 ESP32s

*Found in: [Component config](#) > [ESP32-specific](#) > [CONFIG\\_ESP32\\_SPIRAM\\_SUPPORT](#) > [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.

### SPIRAM cache workaround debugging

 Contains:

- [CONFIG\\_SPIRAM\\_CACHE\\_WORKAROUND\\_STRATEGY](#)

## CONFIG\_SPIRAM\_CACHE\_WORKAROUND\_STRATEGY

Workaround strategy

*Found in: [Component config](#) > [ESP32-specific](#) > [CONFIG\\_ESP32\\_SPIRAM\\_SUPPORT](#) > [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) (SPI-RAM\_CACHE\_WORKAROUND\_STRATEGY\_MEMW)
- Duplicate LD/ST for 32-bit, memw for 8/16 bit (SPI-RAM\_CACHE\_WORKAROUND\_STRATEGY\_DUPLDST)
- Insert nops between vulnerable loads/stores (old strategy, obsolete) (SPI-RAM\_CACHE\_WORKAROUND\_STRATEGY\_NOP)

### CONFIG\_SPIRAM\_BANKSWITCH\_ENABLE

Enable bank switching for >4MiB external RAM

*Found in: Component config > ESP32-specific > CONFIG\_ESP32\_SPIRAM\_SUPPORT > 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\_spiram\_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.

### CONFIG\_SPIRAM\_BANKSWITCH\_RESERVE

Amount of 32K pages to reserve for bank switching

*Found in: Component config > ESP32-specific > CONFIG\_ESP32\_SPIRAM\_SUPPORT > 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.

### CONFIG\_SPIRAM\_MALLOC\_ALWAYSINTERNAL

Maximum malloc() size, in bytes, to always put in internal memory

*Found in: Component config > ESP32-specific > CONFIG\_ESP32\_SPIRAM\_SUPPORT > 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.

### 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 > ESP32-specific > CONFIG\_ESP32\_SPIRAM\_SUPPORT > SPI RAM config*

Try to allocate memories of WiFi and LWIP in SPIRAM firstly. If failed, try to allocate internal memory then.

### 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 > ESP32-specific > CONFIG\_ESP32\_SPIRAM\_SUPPORT > 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.

### CONFIG\_SPIRAM\_ALLOW\_STACK\_EXTERNAL\_MEMORY

Allow external memory as an argument to xTaskCreateStatic

*Found in: Component config > ESP32-specific > CONFIG\_ESP32\_SPIRAM\_SUPPORT > 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 friends 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 to disable this and use xTaskCreateStatic to create the tasks stack in external memory.

### CONFIG\_SPIRAM\_ALLOW\_BSS\_SEG\_EXTERNAL\_MEMORY

Allow .bss segment placed in external memory

*Found in: Component config > ESP32-specific > CONFIG\_ESP32\_SPIRAM\_SUPPORT > SPI RAM config*

If enabled the option, and add EXT\_RAM\_ATTR defined your variable, then your variable will be placed in PSRAM instead of internal memory, and placed most of variables of lwip, net802.11, pp, bluebird library to external memory defaultly.

### CONFIG\_SPIRAM\_OCCUPY\_SPI\_HOST

SPI host to use for 32MBit PSRAM

*Found in: Component config > ESP32-specific > CONFIG\_ESP32\_SPIRAM\_SUPPORT > 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-D0WD Contains:

- [CONFIG\\_D0WD\\_PSRAM\\_CLK\\_IO](#)
- [CONFIG\\_D0WD\\_PSRAM\\_CS\\_IO](#)

### CONFIG\_D0WD\_PSRAM\_CLK\_IO

PSRAM CLK IO number

*Found in: Component config > ESP32-specific > CONFIG\_ESP32\_SPIRAM\_SUPPORT > SPI RAM config > PSRAM clock and cs IO for ESP32-D0WD*

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.

### CONFIG\_D0WD\_PSRAM\_CS\_IO

PSRAM CS IO number

*Found in: Component config > ESP32-specific > CONFIG\_ESP32\_SPIRAM\_SUPPORT > 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.

### 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 > ESP32-specific > CONFIG\_ESP32\_SPIRAM\_SUPPORT > 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.

### CONFIG\_D2WD\_PSRAM\_CS\_IO

PSRAM CS IO number

*Found in: Component config > ESP32-specific > CONFIG\_ESP32\_SPIRAM\_SUPPORT > 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.

### 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 > ESP32-specific > CONFIG\_ESP32\_SPIRAM\_SUPPORT > 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](https://www.espressif.com/sites/default/files/documentation/esp32-pico-d4_datasheet_en.pdf)

### CONFIG\_SPIRAM\_SPIWP\_SD3\_PIN

SPI PSRAM WP(SD3) Pin when customising pins via eFuse (read help)

*Found in: Component config > ESP32-specific > CONFIG\_ESP32\_SPIRAM\_SUPPORT > SPI RAM config*

This value is ignored unless flash mode is set to DIO or DOUT and the SPI flash pins have been overridden by setting the eFuses SPI\_PAD\_CONFIG\_XXX.

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. And the psram only has QPI mode, the WP pin is necessary, so we need to configure this value here.

When flash mode is set to QIO or QOUT, the PSRAM WP pin will be set as the value configured in bootloader.

For ESP32-PICO chip, the default value of this config should be 7.

### CONFIG\_SPIRAM\_2T\_MODE

Enable SPI PSRAM 2T mode

*Found in: Component config > ESP32-specific > CONFIG\_ESP32\_SPIRAM\_SUPPORT > 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.

### CONFIG\_ESP32\_TRAX

Use TRAX tracing feature

*Found in: Component config > ESP32-specific*

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.

### CONFIG\_ESP32\_TRAX\_TWOBANKS

Reserve memory for tracing both pro as well as app cpu execution

*Found in: Component config > ESP32-specific > 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 reserve for trace, used in linker script

### CONFIG\_ESP32\_UNIVERSAL\_MAC\_ADDRESSES

Number of universally administered (by IEEE) MAC address

*Found in: Component config > ESP32-specific*

Configure the number of universally administered (by IEEE) MAC addresses. During initialisation, 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)

**CONFIG\_ESP32\_ULP\_COPROC\_ENABLED**

Enable Ultra Low Power (ULP) Coprocessor

*Found in: [Component config](#) > [ESP32-specific](#)*

Set to 'y' if you plan to load a firmware for the coprocessor.

If this option is enabled, further coprocessor configuration will appear in the Components menu.

**CONFIG\_ESP32\_ULP\_COPROC\_RESERVE\_MEM**

RTC slow memory reserved for coprocessor

*Found in: [Component config](#) > [ESP32-specific](#) > [CONFIG\\_ESP32\\_ULP\\_COPROC\\_ENABLED](#)*

Bytes of memory to reserve for ULP coprocessor firmware & data.

Data is reserved at the beginning of RTC slow memory.

**CONFIG\_ESP32\_PANIC**

Panic handler behaviour

*Found in: [Component config](#) > [ESP32-specific](#)*

If FreeRTOS detects unexpected behaviour or an unhandled exception, the panic handler is invoked. Configure the panic handlers action here.

**Available options:**

- Print registers and halt (ESP32\_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 (ESP32\_PANIC\_PRINT\_REBOOT)  
Outputs the relevant registers over the serial port and immediately reset the processor.
- Silent reboot (ESP32\_PANIC\_SILENT\_REBOOT)  
Just resets the processor without outputting anything
- Invoke GDBStub (ESP32\_PANIC\_GDBSTUB)  
Invoke gdbstub on the serial port, allowing for gdb to attach to it to do a postmortem of the crash.

**CONFIG\_ESP32\_DEBUG\_OCDAWARE**

Make exception and panic handlers JTAG/OCD aware

*Found in: [Component config](#) > [ESP32-specific](#)*

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.

### CONFIG\_ESP32\_DEBUG\_STUBS\_ENABLE

OpenOCD debug stubs

*Found in: [Component config](#) > [ESP32-specific](#)*

Debug stubs are used by OpenOCD to execute pre-compiled onboard code which does some useful debugging, e.g. GCOV data dump.

### CONFIG\_ESP32\_BROWNOUT\_DET

Hardware brownout detect & reset

*Found in: [Component config](#) > [ESP32-specific](#)*

The ESP32 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.

### CONFIG\_ESP32\_BROWNOUT\_DET\_LVL\_SEL

Brownout voltage level

*Found in: [Component config](#) > [ESP32-specific](#) > [CONFIG\\_ESP32\\_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 ESP32 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 (ESP32\_BROWNOUT\_DET\_LVL\_SEL\_0)
- 2.48V +/- 0.05 (ESP32\_BROWNOUT\_DET\_LVL\_SEL\_1)
- 2.58V +/- 0.05 (ESP32\_BROWNOUT\_DET\_LVL\_SEL\_2)
- 2.62V +/- 0.05 (ESP32\_BROWNOUT\_DET\_LVL\_SEL\_3)
- 2.67V +/- 0.05 (ESP32\_BROWNOUT\_DET\_LVL\_SEL\_4)
- 2.70V +/- 0.05 (ESP32\_BROWNOUT\_DET\_LVL\_SEL\_5)
- 2.77V +/- 0.05 (ESP32\_BROWNOUT\_DET\_LVL\_SEL\_6)
- 2.80V +/- 0.05 (ESP32\_BROWNOUT\_DET\_LVL\_SEL\_7)

### CONFIG\_ESP32\_REDUCE\_PHY\_TX\_POWER

Reduce PHY TX power when brownout reset

*Found in: [Component config](#) > [ESP32-specific](#)*

When brownout reset occurs, reduce PHY TX power to keep the code running

# Note about the use of “FRC1” name: currently FRC1 timer is not used for # high resolution time-keeping anymore. Instead the esp\_timer API, implemented # using FRC2 timer, is used. # FRC1 name in the option name is kept for compatibility.

### CONFIG\_ESP32\_TIME\_SYSCALL

Timers used for gettimeofday function

*Found in: [Component config](#) > [ESP32-specific](#)*

This setting defines which hardware timers are used to implement ‘gettimeofday’ and ‘time’ functions in C library.

- If both high-resolution 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 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) 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 (`ESP32_TIME_SYSCALL_USE_RTC_FRC1`)
- RTC (`ESP32_TIME_SYSCALL_USE_RTC`)
- High-resolution timer (`ESP32_TIME_SYSCALL_USE_FRC1`)
- None (`ESP32_TIME_SYSCALL_USE_NONE`)

**CONFIG\_ESP32\_RTC\_CLK\_SRC**

RTC clock source

Found in: *Component config* > *ESP32-specific*

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_XP` 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  $0.1 < V_{cm} < 0.5V_{amp}$ , where  $V_{amp}$  is the signal amplitude. Additionally, 1nF capacitor must be connected between `32K_XN` pin and ground. `32K_XN` pin can not 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 (`ESP32_RTC_CLK_SRC_INT_RC`)
- External 32kHz crystal (`ESP32_RTC_CLK_SRC_EXT_CRYST`)
- External 32kHz oscillator at `32K_XP` pin (`ESP32_RTC_CLK_SRC_EXT_OSC`)
- Internal 8.5MHz oscillator, divided by 256 (~33kHz) (`ESP32_RTC_CLK_SRC_INT_8MD256`)

**CONFIG\_ESP32\_RTC\_EXT\_CRYST\_ADDIT\_CURRENT**

Additional current for external 32kHz crystal

Found in: *Component config* > *ESP32-specific*

Choose which additional current is used for rtc external crystal.

- With some 32kHz crystal configurations, the `X32N` and `X32P` pins may not have enough drive strength to keep the crystal oscillating during deep sleep. If this option is enabled, additional current from touchpad 9 is provided internally to drive the 32kHz crystal. If this option is enabled, deep sleep current is slightly higher (4-5uA) and the touchpad and ULP wakeup sources are not available.

**CONFIG\_ESP32\_RTC\_CLK\_CAL\_CYCLES**

Number of cycles for `RTC_SLOW_CLK` calibration

Found in: *Component config* > *ESP32-specific*

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.

### **CONFIG\_ESP32\_RTC\_XTAL\_CAL\_RETRY**

Number of attempts to repeat 32k XTAL calibration

*Found in: [Component config](#) > [ESP32-specific](#)*

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.

### **CONFIG\_ESP32\_RTC\_XTAL\_BOOTSTRAP\_CYCLES**

Bootstrap cycles for external 32kHz crystal

*Found in: [Component config](#) > [ESP32-specific](#)*

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\_ESP32\_DEEP\_SLEEP\_WAKEUP\_DELAY**

Extra delay in deep sleep wake stub (in us)

*Found in: [Component config](#) > [ESP32-specific](#)*

When ESP32 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.

### **CONFIG\_ESP32\_XTAL\_FREQ\_SEL**

Main XTAL frequency

*Found in: [Component config](#) > [ESP32-specific](#)*

ESP32 currently supports the following XTAL frequencies:

- 26 MHz
- 40 MHz

Startup code can automatically estimate XTAL frequency. This feature uses the internal 8MHz oscillator as a reference. Because the internal oscillator frequency is temperature dependent, it is not recommended to use automatic XTAL frequency detection in applications which need to work at high ambient temperatures and use high-temperature qualified chips and modules.

**Available options:**

- 40 MHz (ESP32\_XTAL\_FREQ\_40)
- 26 MHz (ESP32\_XTAL\_FREQ\_26)
- Autodetect (ESP32\_XTAL\_FREQ\_AUTO)

**CONFIG\_ESP32\_DISABLE\_BASIC\_ROM\_CONSOLE**

Permanently disable BASIC ROM Console

*Found in: [Component config](#) > [ESP32-specific](#)*

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.)

**CONFIG\_ESP32\_NO\_BLOBS**

No Binary Blobs

*Found in: [Component config](#) > [ESP32-specific](#)*

If enabled, this disables the linking of binary libraries in the application build. Note that after enabling this Wi-Fi/Bluetooth will not work.

**CONFIG\_ESP32\_COMPATIBLE\_PRE\_V2\_1\_BOOTLOADERS**

App compatible with bootloaders before ESP-IDF v2.1

*Found in: [Component config](#) > [ESP32-specific](#)*

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.

**CONFIG\_ESP32\_COMPATIBLE\_PRE\_V3\_1\_BOOTLOADERS**

App compatible with bootloader and partition table before ESP-IDF v3.1

*Found in: [Component config](#) > [ESP32-specific](#)*

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).

### CONFIG\_ESP32\_RTCDATA\_IN\_FAST\_MEM

Place RTC\_DATA\_ATTR and RTC\_RODATA\_ATTR variables into RTC fast memory segment

*Found in: [Component config](#) > [ESP32-specific](#)*

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.

### CONFIG\_ESP32\_USE\_FIXED\_STATIC\_RAM\_SIZE

Use fixed static RAM size

*Found in: [Component config](#) > [ESP32-specific](#)*

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

### CONFIG\_ESP32\_FIXED\_STATIC\_RAM\_SIZE

Fixed Static RAM size

*Found in: [Component config](#) > [ESP32-specific](#) > [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 BT\_RESERVE\_DRAM if Bluetooth controller is enabled.

### CONFIG\_ESP32\_DPORT\_DIS\_INTERRUPT\_LVL

Disable the interrupt level for the DPORT workarounds

*Found in: [Component config](#) > [ESP32-specific](#)*

To prevent interrupting DPORT workarounds, need to disable interrupt with a maximum used level in the system.

**Power Management** Contains:

- [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.

### 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 CONFIG\_ESP32\_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.

### CONFIG\_PM\_USE\_RTC\_TIMER\_REF

Use RTC timer to prevent time drift (EXPERIMENTAL)

*Found in: [Component config](#) > [Power Management](#) > [CONFIG\\_PM\\_ENABLE](#)*

When APB clock frequency changes, high-resolution timer (esp\_timer) scale and base value need to be adjusted. Each adjustment may cause small error, and over time such small errors may cause time drift. If this option is enabled, RTC timer will be used as a reference to compensate for the drift. It is recommended that this option is only used if 32k XTAL is selected as RTC clock source.

### 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.

### 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.

### ADC-Calibration Contains:

- [CONFIG\\_ADC\\_CAL\\_EFUSE\\_TP\\_ENABLE](#)
- [CONFIG\\_ADC\\_CAL\\_EFUSE\\_VREF\\_ENABLE](#)
- [CONFIG\\_ADC\\_CAL\\_LUT\\_ENABLE](#)

### CONFIG\_ADC\_CAL\_EFUSE\_TP\_ENABLE

Use Two Point Values

*Found in: [Component config](#) > [ADC-Calibration](#)*

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.

### CONFIG\_ADC\_CAL\_EFUSE\_VREF\_ENABLE

Use eFuse Vref

*Found in: [Component config](#) > [ADC-Calibration](#)*

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.

## CONFIG\_ADC\_CAL\_LUT\_ENABLE

Use Lookup Tables

*Found in: [Component config](#) > [ADC-Calibration](#)*

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.

**Common ESP-related** Contains:

- [CONFIG\\_ESP\\_TIMER\\_PROFILING](#)
- [CONFIG\\_ESP\\_ERR\\_TO\\_NAME\\_LOOKUP](#)
- [CONFIG\\_ESP\\_SYSTEM\\_EVENT\\_QUEUE\\_SIZE](#)
- [CONFIG\\_ESP\\_SYSTEM\\_EVENT\\_TASK\\_STACK\\_SIZE](#)
- [CONFIG\\_ESP\\_MAIN\\_TASK\\_STACK\\_SIZE](#)
- [CONFIG\\_ESP\\_IPC\\_TASK\\_STACK\\_SIZE](#)
- [CONFIG\\_ESP\\_TIMER\\_TASK\\_STACK\\_SIZE](#)
- [CONFIG\\_ESP\\_CONSOLE\\_UART](#)
- [CONFIG\\_ESP\\_CONSOLE\\_UART\\_NUM](#)
- [CONFIG\\_ESP\\_CONSOLE\\_UART\\_TX\\_GPIO](#)
- [CONFIG\\_ESP\\_CONSOLE\\_UART\\_RX\\_GPIO](#)
- [CONFIG\\_ESP\\_CONSOLE\\_UART\\_BAUDRATE](#)
- [CONFIG\\_ESP\\_INT\\_WDT](#)
- [CONFIG\\_ESP\\_TASK\\_WDT](#)

## CONFIG\_ESP\_TIMER\_PROFILING

Enable esp\_timer profiling features

*Found in: [Component config](#) > [Common ESP-related](#)*

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.

## 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.

## CONFIG\_ESP\_SYSTEM\_EVENT\_QUEUE\_SIZE

System event queue size

*Found in: [Component config](#) > [Common ESP-related](#)*

Config system event queue size in different application.



### CONFIG\_ESP\_SYSTEM\_EVENT\_TASK\_STACK\_SIZE

Event loop task stack size

*Found in: [Component config](#) > [Common ESP-related](#)*

Config system event task stack size in different application.

### CONFIG\_ESP\_MAIN\_TASK\_STACK\_SIZE

Main task stack size

*Found in: [Component config](#) > [Common ESP-related](#)*

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.

### CONFIG\_ESP\_IPC\_TASK\_STACK\_SIZE

Inter-Processor Call (IPC) task stack size

*Found in: [Component config](#) > [Common ESP-related](#)*

Configure the IPC tasks stack size. One 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 stack size should be enough for most common use cases. It can be shrunk if you are sure that you do not use any custom IPC functionality.

### CONFIG\_ESP\_TIMER\_TASK\_STACK\_SIZE

High-resolution timer task stack size

*Found in: [Component config](#) > [Common ESP-related](#)*

Configure the stack size of `esp_timer/ets_timer` 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.

### CONFIG\_ESP\_CONSOLE\_UART

UART for console output

*Found in: [Component config](#) > [Common ESP-related](#)*

Select whether to use UART for console output (through `stdout` and `stderr`).

- Default is to use UART0 on pins GPIO1(TX) and GPIO3(RX).
- 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 output can be further suppressed by bootstrapping GPIO13 pin to low logic level.

**Available options:**

- Default: UART0, TX=GPIO1, RX=GPIO3 (ESP\_CONSOLE\_UART\_DEFAULT)
- Custom (ESP\_CONSOLE\_UART\_CUSTOM)
- None (ESP\_CONSOLE\_UART\_NONE)

### **CONFIG\_ESP\_CONSOLE\_UART\_NUM**

UART peripheral to use for console output (0-1)

*Found in: [Component config](#) > [Common ESP-related](#)*

Due of a ROM bug, UART2 is not supported for console output via ets\_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](#) > [Common ESP-related](#)*

### **CONFIG\_ESP\_CONSOLE\_UART\_RX\_GPIO**

UART RX on GPIO#

*Found in: [Component config](#) > [Common ESP-related](#)*

### **CONFIG\_ESP\_CONSOLE\_UART\_BAUDRATE**

UART console baud rate

*Found in: [Component config](#) > [Common ESP-related](#)*

### **CONFIG\_ESP\_INT\_WDT**

Interrupt watchdog

*Found in: [Component config](#) > [Common ESP-related](#)*

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.

### **CONFIG\_ESP\_INT\_WDT\_TIMEOUT\_MS**

Interrupt watchdog timeout (ms)

*Found in: [Component config](#) > [Common ESP-related](#) > [CONFIG\\_ESP\\_INT\\_WDT](#)*

The timeout of the watchdog, in milliseconds. Make this higher than the FreeRTOS tick rate.

### **CONFIG\_ESP\_INT\_WDT\_CHECK\_CPU1**

Also watch CPU1 tick interrupt

*Found in: [Component config](#) > [Common ESP-related](#) > [CONFIG\\_ESP\\_INT\\_WDT](#)*

Also detect if interrupts on CPU 1 are disabled for too long.

### CONFIG\_ESP\_TASK\_WDT

Initialize Task Watchdog Timer on startup

*Found in: [Component config](#) > [Common ESP-related](#)*

The Task Watchdog Timer can be used to make sure individual tasks are still running. Enabling this option will cause the Task Watchdog Timer to be initialized automatically at startup. The Task Watchdog timer can be initialized after startup as well (see [Task Watchdog Timer API Reference](#))

### CONFIG\_ESP\_TASK\_WDT\_PANIC

Invoke panic handler on Task Watchdog timeout

*Found in: [Component config](#) > [Common ESP-related](#) > [CONFIG\\_ESP\\_TASK\\_WDT](#)*

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](#))

### CONFIG\_ESP\_TASK\_WDT\_TIMEOUT\_S

Task Watchdog timeout period (seconds)

*Found in: [Component config](#) > [Common ESP-related](#) > [CONFIG\\_ESP\\_TASK\\_WDT](#)*

Timeout period configuration for the Task Watchdog Timer in seconds. This is also configurable at run time (see [Task Watchdog Timer API Reference](#))

### CONFIG\_ESP\_TASK\_WDT\_CHECK\_IDLE\_TASK\_CPU0

Watch CPU0 Idle Task

*Found in: [Component config](#) > [Common ESP-related](#) > [CONFIG\\_ESP\\_TASK\\_WDT](#)*

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.

### CONFIG\_ESP\_TASK\_WDT\_CHECK\_IDLE\_TASK\_CPU1

Watch CPU1 Idle Task

*Found in: [Component config](#) > [Common ESP-related](#) > [CONFIG\\_ESP\\_TASK\\_WDT](#)*

If this option is enabled, the Task Watchdog Timer will watch the CPU1 Idle Task.

**Ethernet** Contains:

- [CONFIG\\_ETH\\_USE\\_ESP32\\_EMAC](#)
- [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.

Contains:

- [CONFIG\\_ETH\\_PHY\\_INTERFACE](#)
- [CONFIG\\_ETH\\_RMII\\_CLK\\_MODE](#)

- `CONFIG_ETH_RMII_CLK_OUTPUT_GPIO0`
- `CONFIG_ETH_RMII_CLK_OUT_GPIO`
- `CONFIG_ETH_DMA_BUFFER_SIZE`
- `CONFIG_ETH_DMA_RX_BUFFER_NUM`
- `CONFIG_ETH_DMA_TX_BUFFER_NUM`

### **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`)
- Media Independent Interface (MII) (`ETH_PHY_INTERFACE_MII`)

### **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.

### **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.

### **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.

**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.

**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.

**CONFIG\_ETH\_USE\_SPI\_ETHERNET**

Support SPI to Ethernet Module

*Found in: [Component config](#) > [Ethernet](#)*

ESP-IDF can also support some SPI-Ethernet module.

Contains:

- [CONFIG\\_ETH\\_SPI\\_ETHERNET\\_DM9051](#)

**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. Set true to enable DM9051 driver.

**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.

**CONFIG\_ESP\_EVENT\_POST\_FROM\_ISR**

Support posting events from ISRs

*Found in: [Component config](#) > [Event Loop Library](#)*

Enable posting events from interrupt handlers.

### 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.

**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.

**ESP HTTP client** Contains:

- [CONFIG\\_ESP\\_HTTP\\_CLIENT\\_ENABLE\\_HTTPS](#)
- [CONFIG\\_ESP\\_HTTP\\_CLIENT\\_ENABLE\\_BASIC\\_AUTH](#)

### CONFIG\_ESP\_HTTP\_CLIENT\_ENABLE\_HTTPS

Enable https

*Found in: [Component config](#) > [ESP HTTP client](#)*

This option will enable https protocol by linking mbedtls library and initializing SSL transport

### 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

**HTTP Server** Contains:

- [CONFIG\\_HTTPD\\_MAX\\_REQ\\_HDR\\_LEN](#)
- [CONFIG\\_HTTPD\\_MAX\\_URI\\_LEN](#)
- [CONFIG\\_HTTPD\\_ERR\\_RESP\\_NO\\_DELAY](#)
- [CONFIG\\_HTTPD\\_PURGE\\_BUF\\_LEN](#)
- [CONFIG\\_HTTPD\\_LOG\\_PURGE\\_DATA](#)

### **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

### **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

### **CONFIG\_HTTPD\_ERR\_RESP\_NO\_DELAY**

Use TCP\_NODELAY socket option when sending HTTP error responses

*Found in: [Component config](#) > [HTTP Server](#)*

Using TCP\_NODELAY 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

### **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.

### **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.

### **ESP HTTPS OTA** Contains:

- [CONFIG\\_OTA\\_ALLOW\\_HTTP](#)

### **CONFIG\_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

### **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

**Wi-Fi** Contains:

- [CONFIG\\_ESP32\\_WIFI\\_SW\\_COEXIST\\_ENABLE](#)
- [CONFIG\\_ESP32\\_WIFI\\_STATIC\\_RX\\_BUFFER\\_NUM](#)
- [CONFIG\\_ESP32\\_WIFI\\_DYNAMIC\\_RX\\_BUFFER\\_NUM](#)
- [CONFIG\\_ESP32\\_WIFI\\_TX\\_BUFFER](#)
- [CONFIG\\_ESP32\\_WIFI\\_STATIC\\_TX\\_BUFFER\\_NUM](#)
- [CONFIG\\_ESP32\\_WIFI\\_CACHE\\_TX\\_BUFFER\\_NUM](#)
- [CONFIG\\_ESP32\\_WIFI\\_DYNAMIC\\_TX\\_BUFFER\\_NUM](#)
- [CONFIG\\_ESP32\\_WIFI\\_CSI\\_ENABLED](#)
- [CONFIG\\_ESP32\\_WIFI\\_AMPDU\\_TX\\_ENABLED](#)
- [CONFIG\\_ESP32\\_WIFI\\_AMPDU\\_RX\\_ENABLED](#)
- [CONFIG\\_ESP32\\_WIFI\\_NVS\\_ENABLED](#)
- [CONFIG\\_ESP32\\_WIFI\\_TASK\\_CORE\\_ID](#)
- [CONFIG\\_ESP32\\_WIFI\\_SOFTAP\\_BEACON\\_MAX\\_LEN](#)
- [CONFIG\\_ESP32\\_WIFI\\_MGMT\\_SBUF\\_NUM](#)
- [CONFIG\\_ESP32\\_WIFI\\_DEBUG\\_LOG\\_ENABLE](#)
- [CONFIG\\_ESP32\\_WIFI\\_IRAM\\_OPT](#)
- [CONFIG\\_ESP32\\_WIFI\\_RX\\_IRAM\\_OPT](#)
- [CONFIG\\_ESP32\\_WIFI\\_ENABLE\\_WPA3\\_SAE](#)

### 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.

### 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.

### 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.

### 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.

### 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.

**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.

**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.

**CONFIG\_ESP32\_WIFI\_AMPDU\_TX\_ENABLED**

WiFi AMPDU TX

*Found in: [Component config](#) > [Wi-Fi](#)*

Select this option to enable AMPDU TX feature

**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.

**CONFIG\_ESP32\_WIFI\_AMPDU\_RX\_ENABLED**

WiFi AMPDU RX

*Found in: [Component config](#) > [Wi-Fi](#)*

Select this option to enable AMPDU RX feature

**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 preferred to allocate in PSRAM first, the default and minimum value should be 16 to achieve better throughput and compatibility with both stations and APs.

**CONFIG\_ESP32\_WIFI\_NVS\_ENABLED**

WiFi NVS flash

*Found in: [Component config](#) > [Wi-Fi](#)*

Select this option to enable WiFi NVS flash

**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.

**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.

**CONFIG\_ESP32\_WIFI\_DEBUG\_LOG\_ENABLE**

Enable WiFi debug log

*Found in: [Component config](#) > [Wi-Fi](#)*

Select this option to enable WiFi debug log

**CONFIG\_ESP32\_WIFI\_DEBUG\_LOG\_LEVEL**

WiFi debug log level

*Found in: [Component config](#) > [Wi-Fi](#) > [CONFIG\\_ESP32\\_WIFI\\_DEBUG\\_LOG\\_ENABLE](#)*

The WiFi log is divided into the following levels: ERROR,WARNING,INFO,DEBUG,VERBOSE. The ERROR,WARNING,INFO levels are enabled by default, and the DEBUG,VERBOSE levels can be enabled here.

**Available options:**

- WiFi Debug Log Debug (ESP32\_WIFI\_DEBUG\_LOG\_DEBUG)
- WiFi Debug Log Verbose (ESP32\_WIFI\_DEBUG\_LOG\_VERBOSE)

**CONFIG\_ESP32\_WIFI\_DEBUG\_LOG\_MODULE**

WiFi debug log module

*Found in: Component config > Wi-Fi > CONFIG\_ESP32\_WIFI\_DEBUG\_LOG\_ENABLE*

The WiFi log module contains three parts: WIFI,COEX,MESH. The WIFI module indicates the logs related to WiFi, the COEX module indicates the logs related to WiFi and BT(or BLE) coexist, the MESH module indicates the logs related to Mesh. When ESP32\_WIFI\_LOG\_MODULE\_ALL is enabled, all modules are selected.

**Available options:**

- WiFi Debug Log Module All (ESP32\_WIFI\_DEBUG\_LOG\_MODULE\_ALL)
- WiFi Debug Log Module WiFi (ESP32\_WIFI\_DEBUG\_LOG\_MODULE\_WIFI)
- WiFi Debug Log Module Coex (ESP32\_WIFI\_DEBUG\_LOG\_MODULE\_COEX)
- WiFi Debug Log Module Mesh (ESP32\_WIFI\_DEBUG\_LOG\_MODULE\_MESH)

**CONFIG\_ESP32\_WIFI\_DEBUG\_LOG\_SUBMODULE**

WiFi debug log submodule

*Found in: Component config > Wi-Fi > CONFIG\_ESP32\_WIFI\_DEBUG\_LOG\_ENABLE*

Enable this option to set the WiFi debug log submodule. Currently the log submodule contains the following parts: INIT,IOCTL,CONN,SCAN. The INIT submodule indicates the initialization process.The IOCTL submodule indicates the API calling process. The CONN submodule indicates the connecting process.The SCAN submodule indicates the scanning process.

**CONFIG\_ESP32\_WIFI\_DEBUG\_LOG\_SUBMODULE\_ALL**

WiFi Debug Log Submodule All

*Found in: Component config > Wi-Fi > CONFIG\_ESP32\_WIFI\_DEBUG\_LOG\_ENABLE > CONFIG\_ESP32\_WIFI\_DEBUG\_LOG\_SUBMODULE*

When this option is enabled, all debug submodules are selected.

**CONFIG\_ESP32\_WIFI\_DEBUG\_LOG\_SUBMODULE\_INIT**

WiFi Debug Log Submodule Init

*Found in: Component config > Wi-Fi > CONFIG\_ESP32\_WIFI\_DEBUG\_LOG\_ENABLE > CONFIG\_ESP32\_WIFI\_DEBUG\_LOG\_SUBMODULE*

**CONFIG\_ESP32\_WIFI\_DEBUG\_LOG\_SUBMODULE\_IOCTL**

WiFi Debug Log Submodule Ioctl

*Found in: Component config > Wi-Fi > CONFIG\_ESP32\_WIFI\_DEBUG\_LOG\_ENABLE > CONFIG\_ESP32\_WIFI\_DEBUG\_LOG\_SUBMODULE*

**CONFIG\_ESP32\_WIFI\_DEBUG\_LOG\_SUBMODULE\_CONN**

WiFi Debug Log Submodule Conn

*Found in: Component config > Wi-Fi > CONFIG\_ESP32\_WIFI\_DEBUG\_LOG\_ENABLE > CONFIG\_ESP32\_WIFI\_DEBUG\_LOG\_SUBMODULE*

**CONFIG\_ESP32\_WIFI\_DEBUG\_LOG\_SUBMODULE\_SCAN**

WiFi Debug Log Submodule Scan

*Found in: Component config > Wi-Fi > CONFIG\_ESP32\_WIFI\_DEBUG\_LOG\_ENABLE > CONFIG\_ESP32\_WIFI\_DEBUG\_LOG\_SUBMODULE*

**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.

**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.

**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.

**PHY** Contains:

- *CONFIG\_ESP32\_PHY\_CALIBRATION\_AND\_DATA\_STORAGE*
- *CONFIG\_ESP32\_PHY\_INIT\_DATA\_IN\_PARTITION*
- *CONFIG\_ESP32\_PHY\_MAX\_WIFI\_TX\_POWER*

**CONFIG\_ESP32\_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'.

## CONFIG\_ESP32\_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'.

## CONFIG\_ESP32\_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.

**Core dump** Contains:

- [CONFIG\\_ESP32\\_COREDUMP\\_TO\\_FLASH\\_OR\\_UART](#)
- [CONFIG\\_ESP32\\_CORE\\_DUMP\\_MAX\\_TASKS\\_NUM](#)
- [CONFIG\\_ESP32\\_CORE\\_DUMP\\_UART\\_DELAY](#)

## CONFIG\_ESP32\_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 (ESP32\_ENABLE\_COREDUMP\_TO\_FLASH)
- UART (ESP32\_ENABLE\_COREDUMP\_TO\_UART)
- None (ESP32\_ENABLE\_COREDUMP\_TO\_NONE)

## CONFIG\_ESP32\_CORE\_DUMP\_MAX\_TASKS\_NUM

Maximum number of tasks

*Found in: [Component config](#) > [Core dump](#)*

Maximum number of tasks snapshots in core dump.

## CONFIG\_ESP32\_CORE\_DUMP\_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.

**FAT Filesystem support** Contains:

- [CONFIG\\_FATFS\\_CHOOSE\\_CODEPAGE](#)
- [CONFIG\\_FATFS\\_LONG\\_FILENAMES](#)
- [CONFIG\\_FATFS\\_MAX\\_LFN](#)
- [CONFIG\\_FATFS\\_API\\_ENCODING](#)
- [CONFIG\\_FATFS\\_FS\\_LOCK](#)
- [CONFIG\\_FATFS\\_TIMEOUT\\_MS](#)
- [CONFIG\\_FATFS\\_PER\\_FILE\\_CACHE](#)
- [CONFIG\\_FATFS\\_ALLOC\\_PREFER\\_EXTRAM](#)

**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)
- Portugese (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\_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\_MAX\_LFN

Max long filename length

*Found in: [Component config](#) > [FAT Filesystem support](#)*

Maximum long filename length. Can be reduced to save RAM.

### 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-16 encoding (FATFS\_API\_ENCODING\_UTF\_16)
- 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.

### 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.

### 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.



### 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.

### Modbus configuration Contains:

- [CONFIG\\_FMB\\_MASTER\\_TIMEOUT\\_MS\\_RESPOND](#)
- [CONFIG\\_FMB\\_MASTER\\_DELAY\\_MS\\_CONVERT](#)
- [CONFIG\\_FMB\\_QUEUE\\_LENGTH](#)
- [CONFIG\\_FMB\\_SERIAL\\_TASK\\_STACK\\_SIZE](#)
- [CONFIG\\_FMB\\_SERIAL\\_BUF\\_SIZE](#)
- [CONFIG\\_FMB\\_SERIAL\\_TASK\\_PRIO](#)
- [CONFIG\\_FMB\\_CONTROLLER\\_SLAVE\\_ID\\_SUPPORT](#)
- [CONFIG\\_FMB\\_CONTROLLER\\_NOTIFY\\_TIMEOUT](#)
- [CONFIG\\_FMB\\_CONTROLLER\\_NOTIFY\\_QUEUE\\_SIZE](#)
- [CONFIG\\_FMB\\_CONTROLLER\\_STACK\\_SIZE](#)
- [CONFIG\\_FMB\\_EVENT\\_QUEUE\\_TIMEOUT](#)
- [CONFIG\\_FMB\\_TIMER\\_PORT\\_ENABLED](#)
- [CONFIG\\_FMB\\_TIMER\\_GROUP](#)
- [CONFIG\\_FMB\\_TIMER\\_INDEX](#)

### CONFIG\_FMB\_MASTER\_TIMEOUT\_MS\_RESPOND

Slave respond timeout (Milliseconds)

*Found in: [Component config > Modbus configuration](#)*

If master sends a frame which is not broadcast, it has to wait sometime for slave response. if slave is not respond in this time, the master will process timeout error.

### CONFIG\_FMB\_MASTER\_DELAY\_MS\_CONVERT

Slave conversion delay (Milliseconds)

*Found in: [Component config > Modbus configuration](#)*

If master sends a broadcast frame, it has to wait conversion time to delay, then master can send next frame.

### CONFIG\_FMB\_QUEUE\_LENGTH

Modbus serial task queue length

*Found in: [Component config > Modbus configuration](#)*

Modbus serial driver queue length. It is used by event queue task. See the serial driver API for more information.

### CONFIG\_FMB\_SERIAL\_TASK\_STACK\_SIZE

Modbus serial task stack size

*Found in: [Component config > Modbus configuration](#)*

Modbus serial task stack size for event queue task. It may be adjusted when debugging is enabled (for example).

### CONFIG\_FMB\_SERIAL\_BUF\_SIZE

Modbus serial task RX/TX buffer size

*Found in: [Component config](#) > [Modbus configuration](#)*

Modbus serial task RX and TX buffer size for UART driver initialization. This buffer is used for modbus frame transfer. The Modbus protocol maximum frame size is 256 bytes. Bigger size can be used for non standard implementations.

### CONFIG\_FMB\_SERIAL\_TASK\_PRIO

Modbus serial task priority

*Found in: [Component config](#) > [Modbus configuration](#)*

Modbus UART driver event task priority. The priority of Modbus controller task is equal to (CONFIG\_FMB\_SERIAL\_TASK\_PRIO - 1).

### CONFIG\_FMB\_CONTROLLER\_SLAVE\_ID\_SUPPORT

Modbus controller slave ID support

*Found in: [Component config](#) > [Modbus configuration](#)*

Modbus slave ID support enable. When enabled the Modbus <Report Slave ID> command is supported by stack.

### CONFIG\_FMB\_CONTROLLER\_SLAVE\_ID

Modbus controller slave ID

*Found in: [Component config](#) > [Modbus configuration](#) > [CONFIG\\_FMB\\_CONTROLLER\\_SLAVE\\_ID\\_SUPPORT](#)*

Modbus slave ID value to identify modbus device in the network using <Report Slave ID> command. Most significant byte of ID is used as short device ID and other three bytes used as long ID.

### CONFIG\_FMB\_CONTROLLER\_NOTIFY\_TIMEOUT

Modbus controller notification timeout (ms)

*Found in: [Component config](#) > [Modbus configuration](#)*

Modbus controller notification timeout in milliseconds. This timeout is used to send notification about accessed parameters.

### CONFIG\_FMB\_CONTROLLER\_NOTIFY\_QUEUE\_SIZE

Modbus controller notification queue size

*Found in: [Component config](#) > [Modbus configuration](#)*

Modbus controller notification queue size. The notification queue is used to get information about accessed parameters.

### CONFIG\_FMB\_CONTROLLER\_STACK\_SIZE

Modbus controller stack size

*Found in: [Component config](#) > [Modbus configuration](#)*

Modbus controller task stack size. The Stack size may be adjusted when debug mode is used which requires more stack size (for example).

### CONFIG\_FMB\_EVENT\_QUEUE\_TIMEOUT

Modbus stack event queue timeout (ms)

*Found in: [Component config](#) > [Modbus configuration](#)*

Modbus stack event queue timeout in milliseconds. This may help to optimize Modbus stack event processing time.

### CONFIG\_FMB\_TIMER\_PORT\_ENABLED

Modbus slave stack use timer for 3.5T symbol time measurement

*Found in: [Component config](#) > [Modbus configuration](#)*

If this option is set the Modbus stack uses timer for T3.5 time measurement. Else the internal UART TOUT timeout is used for 3.5T symbol time measurement.

### CONFIG\_FMB\_TIMER\_GROUP

Modbus Timer group number

*Found in: [Component config](#) > [Modbus configuration](#)*

Modbus Timer group number that is used for timeout measurement.

### CONFIG\_FMB\_TIMER\_INDEX

Modbus Timer index in the group

*Found in: [Component config](#) > [Modbus configuration](#)*

Modbus Timer Index in the group that is used for timeout measurement.

**FreeRTOS** Contains:

- [CONFIG\\_FREERTOS\\_UNICORE](#)
- [CONFIG\\_FREERTOS\\_CORETIMER](#)
- [CONFIG\\_FREERTOS\\_HZ](#)
- [CONFIG\\_FREERTOS\\_ASSERT\\_ON\\_UNTESTED\\_FUNCTION](#)
- [CONFIG\\_FREERTOS\\_CHECK\\_STACKOVERFLOW](#)
- [CONFIG\\_FREERTOS\\_WATCHPOINT\\_END\\_OF\\_STACK](#)
- [CONFIG\\_FREERTOS\\_INTERRUPT\\_BACKTRACE](#)
- [CONFIG\\_FREERTOS\\_THREAD\\_LOCAL\\_STORAGE\\_POINTERS](#)
- [CONFIG\\_FREERTOS\\_ASSERT](#)
- [CONFIG\\_FREERTOS\\_IDLE\\_TASK\\_STACKSIZE](#)
- [CONFIG\\_FREERTOS\\_ISR\\_STACKSIZE](#)
- [CONFIG\\_FREERTOS\\_LEGACY\\_HOOKS](#)
- [CONFIG\\_FREERTOS\\_MAX\\_TASK\\_NAME\\_LEN](#)
- [CONFIG\\_FREERTOS\\_SUPPORT\\_STATIC\\_ALLOCATION](#)
- [CONFIG\\_FREERTOS\\_TIMER\\_TASK\\_PRIORITY](#)
- [CONFIG\\_FREERTOS\\_TIMER\\_TASK\\_STACK\\_DEPTH](#)
- [CONFIG\\_FREERTOS\\_TIMER\\_QUEUE\\_LENGTH](#)
- [CONFIG\\_FREERTOS\\_QUEUE\\_REGISTRY\\_SIZE](#)
- [CONFIG\\_FREERTOS\\_USE\\_TRACE\\_FACILITY](#)
- [CONFIG\\_FREERTOS\\_GENERATE\\_RUN\\_TIME\\_STATS](#)
- [CONFIG\\_FREERTOS\\_USE\\_TICKLESS\\_IDLE](#)
- [CONFIG\\_FREERTOS\\_DEBUG\\_INTERNALS](#)
- [CONFIG\\_FREERTOS\\_TASK\\_FUNCTION\\_WRAPPER](#)
- [CONFIG\\_FREERTOS\\_CHECK\\_MUTEX\\_GIVEN\\_BY\\_OWNER](#)
- [CONFIG\\_FREERTOS\\_CHECK\\_PORT\\_CRITICAL\\_COMPLIANCE](#)

## CONFIG\_FREERTOS\_UNICORE

Run FreeRTOS only on first core

*Found in: [Component config > FreeRTOS](#)*

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.

# This invisible config value sets the value of `tskNO_AFFINITY` in `task.h`. # Intended to be used as a constant from other Kconfig files. # Value is (32-bit) `INT_MAX`.

## CONFIG\_FREERTOS\_CORETIMER

Xtensa timer to use as the FreeRTOS tick source

*Found in: [Component config > FreeRTOS](#)*

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. Check

### 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

## CONFIG\_FREERTOS\_HZ

Tick rate (Hz)

*Found in: [Component config > FreeRTOS](#)*

Select the tick rate at which FreeRTOS does pre-emptive context switching.

## CONFIG\_FREERTOS\_ASSERT\_ON\_UNTESTED\_FUNCTION

Halt when an SMP-untested function is called

*Found in: [Component config > FreeRTOS](#)*

Some functions in FreeRTOS have not been thoroughly tested yet when moving to the SMP implementation of FreeRTOS. When this option is enabled, these functions will throw an `assert()`.

## CONFIG\_FREERTOS\_CHECK\_STACKOVERFLOW

Check for stack overflow

*Found in: [Component config > FreeRTOS](#)*

FreeRTOS can check for stack overflows in threads and trigger a user function called `vApplicationStackOverflowHook` when this happens.

### Available options:

- No checking (`FREERTOS_CHECK_STACKOVERFLOW_NONE`)  
Do not check for stack overflows (`configCHECK_FOR_STACK_OVERFLOW=0`)
- Check by stack pointer value (`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 (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\_WATCHPOINT\_END\_OF\_STACK

Set a debug watchpoint as a stack overflow check

*Found in: [Component config > FreeRTOS](#)*

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 debug memory watchpoint 1 (the second one) to allow breaking into the debugger (or panic'ing) 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 only have one watchpoint; the 2nd one is overwritten as soon as a task switch happens.

This check only triggers if the stack overflow writes within 4 bytes of 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.

### CONFIG\_FREERTOS\_INTERRUPT\_BACKTRACE

Enable backtrace from interrupt to task context

*Found in: [Component config > FreeRTOS](#)*

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.

### CONFIG\_FREERTOS\_THREAD\_LOCAL\_STORAGE\_POINTERS

Number of thread local storage pointers

*Found in: [Component config > FreeRTOS](#)*

FreeRTOS has the ability to store per-thread pointers in the task control block. This controls the number of pointers available.

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.

### CONFIG\_FREERTOS\_ASSERT

FreeRTOS assertions

*Found in: [Component config > FreeRTOS](#)*

Failed FreeRTOS configASSERT() assertions can be configured to behave in different ways.

#### Available options:

- abort() on failed assertions (FREERTOS\_ASSERT\_FAIL\_ABORT)  
If a FreeRTOS configASSERT() fails, FreeRTOS will abort() and halt execution. The panic handler can be configured to handle the outcome of an abort() in different ways.

- Print and continue failed assertions (FREERTOS\_ASSERT\_FAIL\_PRINT\_CONTINUE)  
If a FreeRTOS assertion fails, print it out and continue.
- Disable FreeRTOS assertions (FREERTOS\_ASSERT\_DISABLE)  
FreeRTOS configASSERT() will not be compiled into the binary.

### CONFIG\_FREERTOS\_IDLE\_TASK\_STACKSIZE

Idle Task stack size

*Found in: [Component config](#) > [FreeRTOS](#)*

The idle task has its own stack, sized in bytes. The default size is enough for most uses. Size can be reduced to 768 bytes if no (or simple) FreeRTOS idle hooks are used and pthread local storage or FreeRTOS local storage cleanup callbacks are not used.

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.

### CONFIG\_FREERTOS\_ISR\_STACKSIZE

ISR stack size

*Found in: [Component config](#) > [FreeRTOS](#)*

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.

### CONFIG\_FREERTOS\_LEGACY\_HOOKS

Use FreeRTOS legacy hooks

*Found in: [Component config](#) > [FreeRTOS](#)*

FreeRTOS offers a number of hooks/callback functions that are called when a timer tick happens, the idle thread runs etc. esp-idf replaces these by runtime registerable hooks using the esp\_register\_freertos\_xxx\_hook system, but for legacy reasons the old hooks can also still be enabled. Please enable this only if you have code that for some reason can't be migrated to the esp\_register\_freertos\_xxx\_hook system.

### CONFIG\_FREERTOS\_MAX\_TASK\_NAME\_LEN

Maximum task name length

*Found in: [Component config](#) > [FreeRTOS](#)*

Changes the maximum task name length. Each task allocated will include this many bytes for a task name. Using a shorter value saves a small amount of RAM, a longer value allows more complex names.

For most uses, the default of 16 is OK.

### CONFIG\_FREERTOS\_SUPPORT\_STATIC\_ALLOCATION

Enable FreeRTOS static allocation API

*Found in: [Component config](#) > [FreeRTOS](#)*

FreeRTOS gives the application writer the ability to instead provide the memory themselves, allowing the following objects to optionally be created without any memory being allocated dynamically:

- Tasks
- Software Timers (Daemon task is still dynamic. See documentation)
- Queues
- Event Groups
- Binary Semaphores

- Counting Semaphores
- Recursive Semaphores
- Mutexes

Whether it is preferable to use static or dynamic memory allocation is dependent on the application, and the preference of the application writer. Both methods have pros and cons, and both methods can be used within the same RTOS application.

Creating RTOS objects using statically allocated RAM has the benefit of providing the application writer with more control: RTOS objects can be placed at specific memory locations. The maximum RAM footprint can be determined at link time, rather than run time. The application writer does not need to concern themselves with graceful handling of memory allocation failures. It allows the RTOS to be used in applications that simply don't allow any dynamic memory allocation (although FreeRTOS includes allocation schemes that can overcome most objections).

### **CONFIG\_FREERTOS\_ENABLE\_STATIC\_TASK\_CLEAN\_UP**

Enable static task clean up hook

*Found in: [Component config](#) > [FreeRTOS](#) > [CONFIG\\_FREERTOS\\_SUPPORT\\_STATIC\\_ALLOCATION](#)*

Enable this option to make FreeRTOS call the static task clean up hook when a task is deleted.

Bear in mind that if this option is enabled you will need to implement the following function:

```
void vPortCleanUpTCB ( void *pxTCB ) {  
    // place clean up code here  
}
```

### **CONFIG\_FREERTOS\_TIMER\_TASK\_PRIORITY**

FreeRTOS timer task priority

*Found in: [Component config](#) > [FreeRTOS](#)*

The timer service task (primarily) makes use of existing FreeRTOS features, allowing timer functionality to be added to an application with minimal impact on the size of the application's executable binary.

Use this constant to define the priority that the timer task will run at.

### **CONFIG\_FREERTOS\_TIMER\_TASK\_STACK\_DEPTH**

FreeRTOS timer task stack size

*Found in: [Component config](#) > [FreeRTOS](#)*

The timer service task (primarily) makes use of existing FreeRTOS features, allowing timer functionality to be added to an application with minimal impact on the size of the application's executable binary.

Use this constant to define the size (in bytes) of the stack allocated for the timer task.

### **CONFIG\_FREERTOS\_TIMER\_QUEUE\_LENGTH**

FreeRTOS timer queue length

*Found in: [Component config](#) > [FreeRTOS](#)*

FreeRTOS provides a set of timer related API functions. Many of these functions use a standard FreeRTOS queue to send commands to the timer service task. The queue used for this purpose is called the 'timer command queue'. The 'timer command queue' is private to the FreeRTOS timer implementation, and cannot be accessed directly.

For most uses the default value of 10 is OK.

### CONFIG\_FREERTOS\_QUEUE\_REGISTRY\_SIZE

FreeRTOS queue registry size

*Found in: [Component config](#) > [FreeRTOS](#)*

FreeRTOS uses the queue registry as a means for kernel aware debuggers to locate queues, semaphores, and mutexes. The registry allows for a textual name to be associated with a queue for easy identification within a debugging GUI. A value of 0 will disable queue registry functionality, and a value larger than 0 will specify the number of queues/semaphores/mutexes that the registry can hold.

### CONFIG\_FREERTOS\_USE\_TRACE\_FACILITY

Enable FreeRTOS trace facility

*Found in: [Component config](#) > [FreeRTOS](#)*

If enabled, configUSE\_TRACE\_FACILITY will be defined as 1 in FreeRTOS. This will allow the usage of trace facility functions such as uxTaskGetSystemState().

### CONFIG\_FREERTOS\_USE\_STATS\_FORMATTING\_FUNCTIONS

Enable FreeRTOS stats formatting functions

*Found in: [Component config](#) > [FreeRTOS](#) > [CONFIG\\_FREERTOS\\_USE\\_TRACE\\_FACILITY](#)*

If enabled, configUSE\_STATS\_FORMATTING\_FUNCTIONS will be defined as 1 in FreeRTOS. This will allow the usage of stats formatting functions such as vTaskList().

### CONFIG\_FREERTOS\_VTASKLIST\_INCLUDE\_COREID

Enable display of xCoreID in vTaskList

*Found in: [Component config](#) > [FreeRTOS](#) > [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.

### CONFIG\_FREERTOS\_GENERATE\_RUN\_TIME\_STATS

Enable FreeRTOS to collect run time stats

*Found in: [Component config](#) > [FreeRTOS](#)*

If enabled, configGENERATE\_RUN\_TIME\_STATS will be defined as 1 in FreeRTOS. This will allow FreeRTOS to collect information regarding the usage of processor time amongst FreeRTOS tasks. Run time stats are generated using either the ESP Timer or the CPU Clock as the clock source (Note that run time stats are only valid until the clock source overflows). The function vTaskGetRunTimeStats() will also be available if FREERTOS\_USE\_STATS\_FORMATTING\_FUNCTIONS and FREERTOS\_USE\_TRACE\_FACILITY are enabled. vTaskGetRunTimeStats() will display the run time of each task as a % of the total run time of all CPUs (task run time / no of CPUs) / (total run time / 100)

### CONFIG\_FREERTOS\_RUN\_TIME\_STATS\_CLK

Choose the clock source for run time stats

*Found in: [Component config](#) > [FreeRTOS](#) > [CONFIG\\_FREERTOS\\_GENERATE\\_RUN\\_TIME\\_STATS](#)*

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 ESP32\_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\_USE\_TICKLESS\_IDLE**

Tickless idle support

*Found in: [Component config](#) > [FreeRTOS](#)*

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.

If disabled, automatic light sleep support will be disabled.

**CONFIG\_FREERTOS\_IDLE\_TIME\_BEFORE\_SLEEP**

Minimum number of ticks to enter sleep mode for

*Found in: [Component config](#) > [FreeRTOS](#) > [CONFIG\\_FREERTOS\\_USE\\_TICKLESS\\_IDLE](#)*

FreeRTOS will enter light sleep mode if no tasks need to run for this number of ticks.

**CONFIG\_FREERTOS\_DEBUG\_INTERNALS**

Debug FreeRTOS internals

*Found in: [Component config](#) > [FreeRTOS](#)*

Enable this option to show the menu with internal FreeRTOS debugging features. This option does not change any code by itself, it just shows/hides some options.

Contains:

- [CONFIG\\_FREERTOS\\_PORTMUX\\_DEBUG](#)
- [CONFIG\\_FREERTOS\\_PORTMUX\\_DEBUG\\_RECURSIVE](#)

**CONFIG\_FREERTOS\_PORTMUX\_DEBUG**

Debug portMUX portENTER\_CRITICAL/portEXIT\_CRITICAL

*Found in: [Component config](#) > [FreeRTOS](#) > [CONFIG\\_FREERTOS\\_DEBUG\\_INTERNALS](#)*

If enabled, debug information (including integrity checks) will be printed to UART for the port-specific MUX implementation.

### CONFIG\_FREERTOS\_PORTMUX\_DEBUG\_RECURSIVE

Debug portMUX Recursion

*Found in: [Component config](#) > [FreeRTOS](#) > [CONFIG\\_FREERTOS\\_DEBUG\\_INTERNALS](#)*

If enabled, additional debug information will be printed for recursive portMUX usage.

### CONFIG\_FREERTOS\_TASK\_FUNCTION\_WRAPPER

Enclose all task functions in a wrapper function

*Found in: [Component config](#) > [FreeRTOS](#)*

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.

### CONFIG\_FREERTOS\_CHECK\_MUTEX\_GIVEN\_BY\_OWNER

Check that mutex semaphore is given by owner task

*Found in: [Component config](#) > [FreeRTOS](#)*

If enabled, assert that when a mutex semaphore is given, the task giving the semaphore is the task which is currently holding the mutex.

### CONFIG\_FREERTOS\_CHECK\_PORT\_CRITICAL\_COMPLIANCE

Tests compliance with Vanilla FreeRTOS port\*\_CRITICAL calls

*Found in: [Component config](#) > [FreeRTOS](#)*

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

**Heap memory debugging** Contains:

- [CONFIG\\_HEAP\\_CORRUPTION\\_DETECTION](#)
- [CONFIG\\_HEAP\\_TRACING\\_DEST](#)
- [CONFIG\\_HEAP\\_TRACING\\_STACK\\_DEPTH](#)
- [CONFIG\\_HEAP\\_TASK\\_TRACKING](#)

### 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 size 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 uses 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.

**libsodium** Contains:

- [CONFIG\\_LIBSODIUM\\_USE\\_MBEDTLS\\_SHA](#)

### CONFIG\_LIBSODIUM\_USE\_MBEDTLS\_SHA

Use mbedTLS SHA256 & SHA512 implementations

*Found in: [Component config](#) > [libsodium](#)*

If this option is enabled, libsodium will use thin wrappers around mbedTLS for SHA256 & SHA512 operations.

This saves some code size if mbedTLS is also used. However it is incompatible with hardware SHA acceleration (due to the way libsodium's API manages SHA state).

**Log output** Contains:

- [CONFIG\\_LOG\\_DEFAULT\\_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.

Note that this setting limits which log statements are compiled into the program. So setting this to, say, “Warning” would mean that changing log level to “Debug” at runtime will not be possible.

### 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\_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.

## LWIP

 Contains:

- `CONFIG_LWIP_LOCAL_HOSTNAME`
- `CONFIG_LWIP_L2_TO_L3_COPY`
- `CONFIG_LWIP_IRAM_OPTIMIZATION`
- `CONFIG_LWIP_TIMERS_ONDEMAND`
- `CONFIG_LWIP_MAX_SOCKETS`
- `CONFIG_LWIP_USE_ONLY_LWIP_SELECT`
- `CONFIG_LWIP_SO_LINGER`
- `CONFIG_LWIP_SO_REUSE`
- `CONFIG_LWIP_SO_RCVBUF`
- `CONFIG_LWIP_IP4_FRAG`
- `CONFIG_LWIP_IP6_FRAG`
- `CONFIG_LWIP_IP4_REASSEMBLY`
- `CONFIG_LWIP_IP6_REASSEMBLY`
- `CONFIG_LWIP_STATS`
- `CONFIG_LWIP_ETHARP_TRUST_IP_MAC`
- `CONFIG_LWIP_ESP_GRATUITOUS_ARP`
- `CONFIG_LWIP_TCPIP_RECVMBOX_SIZE`
- `CONFIG_LWIP_DHCP_DOES_ARP_CHECK`
- `CONFIG_LWIP_DHCP_RESTORE_LAST_IP`
- `DHCP server`
- `CONFIG_LWIP_AUTOIP`
- `CONFIG_LWIP_IPV6_AUTOCONFIG`
- `CONFIG_LWIP_NETIF_LOOPBACK`
- `TCP`
- `UDP`
- `CONFIG_LWIP_TCPIP_TASK_STACK_SIZE`
- `CONFIG_LWIP_TCPIP_TASK_AFFINITY`
- `CONFIG_LWIP_PPP_SUPPORT`
- `CONFIG_LWIP_IPV6_MEMP_NUM_ND6_QUEUE`
- `CONFIG_LWIP_IPV6_ND6_NUM_NEIGHBORS`

- [CONFIG\\_LWIP\\_PPP\\_NOTIFY\\_PHASE\\_SUPPORT](#)
- [CONFIG\\_LWIP\\_PPP\\_PAP\\_SUPPORT](#)
- [CONFIG\\_LWIP\\_PPP\\_CHAP\\_SUPPORT](#)
- [CONFIG\\_LWIP\\_PPP\\_MSCHAP\\_SUPPORT](#)
- [CONFIG\\_LWIP\\_PPP\\_MPPE\\_SUPPORT](#)
- [CONFIG\\_LWIP\\_PPP\\_DEBUG\\_ON](#)
- [ICMP](#)
- [LWIP RAW API](#)
- [SNTP](#)

### **CONFIG\_LWIP\_LOCAL\_HOSTNAME**

Local netif hostname

*Found in: [Component config](#) > [LWIP](#)*

The name this device will report to other devices on the network

### **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.

### **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.

### **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.

### **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.

### CONFIG\_LWIP\_USE\_ONLY\_LWIP\_SELECT

Support LWIP socket select() only

*Found in: [Component config](#) > [LWIP](#)*

The virtual filesystem layer of select() redirects sockets to lwip\_select() and non-socket file descriptors to their respective driver implementations. If this option is enabled then all calls of select() will be redirected to lwip\_select(), therefore, select can be used for sockets only.

### 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.

### 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.

### 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.

### CONFIG\_LWIP\_SO\_RCVBUF

Enable SO\_RCVBUF option

*Found in: [Component config](#) > [LWIP](#)*

Enabling this option allows checking for available data on a netconn.

### 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.

### **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.

### **CONFIG\_LWIP\_IP4\_REASSEMBLY**

Enable reassembly incoming fragmented IP4 packets

*Found in: [Component config](#) > [LWIP](#)*

Enabling this option allows reassembling incoming fragmented IP4 packets.

### **CONFIG\_LWIP\_IP6\_REASSEMBLY**

Enable reassembly incoming fragmented IP6 packets

*Found in: [Component config](#) > [LWIP](#)*

Enabling this option allows reassembling incoming fragmented IP6 packets.

### **CONFIG\_LWIP\_STATS**

Enable LWIP statistics

*Found in: [Component config](#) > [LWIP](#)*

Enabling this option allows LWIP statistics

### **CONFIG\_LWIP\_ETHARP\_TRUST\_IP\_MAC**

Enable LWIP ARP trust

*Found in: [Component config](#) > [LWIP](#)*

Enabling this option allows ARP table to be updated.

If this option is enabled, the incoming IP packets cause the ARP table to be updated with the source MAC and IP addresses supplied in the packet. You may want to disable this if you do not trust LAN peers to have the correct addresses, or as a limited approach to attempt to handle spoofing. If disabled, lwIP will need to make a new ARP request if the peer is not already in the ARP table, adding a little latency. The peer \*is\* in the ARP table if it requested our address before. Also notice that this slows down input processing of every IP packet!

There are two known issues in real application if this feature is enabled: - The LAN peer may have bug to update the ARP table after the ARP entry is aged out. If the ARP entry on the LAN peer is aged out but failed to be updated, all IP packets sent from LWIP to the LAN peer will be dropped by LAN peer. - The LAN peer may not be trustful, the LAN peer may send IP packets to LWIP with two different MACs, but the same IP address. If this happens, the LWIP has problem to receive IP packets from LAN peer.

So the recommendation is to disable this option. Here the LAN peer means the other side to which the ESP station or soft-AP is connected.

### **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.

### **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

### **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.

### **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.

### **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.

**DHCP server** Contains:

- *CONFIG\_LWIP\_DHCPS\_LEASE\_UNIT*
- *CONFIG\_LWIP\_DHCPS\_MAX\_STATION\_NUM*

### **CONFIG\_LWIP\_DHCPS\_LEASE\_UNIT**

Multiplier for lease time, in seconds

*Found in: Component config > LWIP > DHCP server*

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.

### **CONFIG\_LWIP\_DHCPS\_MAX\_STATION\_NUM**

Maximum number of stations

*Found in: Component config > LWIP > DHCP server*

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.



## 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.

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.)

## 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.

## 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.

## CONFIG\_LWIP\_IPV6\_AUTOCONFIG

Enable IPV6 stateless address autoconfiguration

*Found in: [Component config](#) > [LWIP](#)*

Enabling this option allows the devices to IPV6 stateless address autoconfiguration.

See RFC 4862.

## 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.

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.

**TCP** Contains:

- [CONFIG\\_LWIP\\_TCP\\_ISN\\_HOOK](#)
- [CONFIG\\_LWIP\\_MAX\\_ACTIVE\\_TCP](#)
- [CONFIG\\_LWIP\\_MAX\\_LISTENING\\_TCP](#)
- [CONFIG\\_LWIP\\_TCP\\_HIGH\\_SPEED\\_RETRANSMISSION](#)
- [CONFIG\\_LWIP\\_TCP\\_MAXRTX](#)
- [CONFIG\\_LWIP\\_TCP\\_SYNMAXRTX](#)
- [CONFIG\\_LWIP\\_TCP\\_MSS](#)
- [CONFIG\\_LWIP\\_TCP\\_MSL](#)
- [CONFIG\\_LWIP\\_TCP\\_SND\\_BUF\\_DEFAULT](#)
- [CONFIG\\_LWIP\\_TCP\\_WND\\_DEFAULT](#)
- [CONFIG\\_LWIP\\_TCP\\_RECVMBOX\\_SIZE](#)
- [CONFIG\\_LWIP\\_TCP\\_QUEUE\\_OOSEQ](#)
- [CONFIG\\_LWIP\\_TCP\\_KEEP\\_CONNECTION\\_WHEN\\_IP\\_CHANGES](#)
- [CONFIG\\_LWIP\\_TCP\\_OVERSIZE](#)
- [CONFIG\\_LWIP\\_WND\\_SCALE](#)
- [CONFIG\\_LWIP\\_TCP\\_RTO\\_TIME](#)

### CONFIG\_LWIP\_TCP\_ISN\_HOOK

Enable TCP ISN Hook

*Found in: Component config > LWIP > TCP*

Enables custom TCP ISN hook to randomize initial sequence number in TCP connection. This is recommended as default lwIP implementation (*tcp\_next\_iss*) is not very strong, as it does not take into consideration any platform specific entropy source.

### 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.

### 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.

### **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.

### **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.

### **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.

### **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

### **CONFIG\_LWIP\_TCP\_MSL**

Maximum segment lifetime (MSL)

*Found in: [Component config](#) > [LWIP](#) > [TCP](#)*

Set maximum segment lifetime in in milliseconds.

### **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.

### 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.

### CONFIG\_LWIP\_TCP\_RECVMBOX\_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:  $LWIP\_TCP\_WND\_DEFAULT/TCP\_MSS + 2$ , e.g. if  $LWIP\_TCP\_WND\_DEFAULT=14360$ ,  $TCP\_MSS=1436$ , then the recommended receive mail box size is  $(14360/1436 + 2) = 12$ .

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 cache maximum `LWIP_TCP_RECVMBOX_SIZE` packets for each TCP socket, so the maximum possible cached TCP packets for all TCP sockets is `LWIP_TCP_RECVMBOX_SIZE` multiples the maximum TCP socket number. In other words, the bigger `LWIP_TCP_RECVMBOX_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.

### 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.

### CONFIG\_LWIP\_TCP\_KEEP\_CONNECTION\_WHEN\_IP\_CHANGES

Keep TCP connections when IP changed

*Found in: [Component config](#) > [LWIP](#) > [TCP](#)*

This option is enabled when the following scenario happen: network dropped and reconnected, IP changes is like: `192.168.0.2->0.0.0.0->192.168.0.2`

Disable this option to keep consistent with the original LWIP code behavior.

### 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.

### 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.

### 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.

### UDP Contains:

- [CONFIG\\_LWIP\\_MAX\\_UDP\\_PCBS](#)
- [CONFIG\\_LWIP\\_UDP\\_RECVMBOX\\_SIZE](#)

### 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.

### 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\_RECVMBOX\_SIZE packets for each UDP socket, so the maximum possible cached UDP packets for all UDP sockets is

UDP\_RECVMBOX\_SIZE multiplies the maximum UDP socket number. In other words, the bigger UDP\_RECVMBOX\_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 UDP receive mail box is big enough to avoid packet drop between LWIP core and application.

### 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.

### 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.

### 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.

### CONFIG\_LWIP\_IPV6\_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

### 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.

### **CONFIG\_LWIP\_PPP\_PAP\_SUPPORT**

Enable PAP support

*Found in: [Component config](#) > [LWIP](#)*

Enable Password Authentication Protocol (PAP) support

### **CONFIG\_LWIP\_PPP\_CHAP\_SUPPORT**

Enable CHAP support

*Found in: [Component config](#) > [LWIP](#)*

Enable Challenge Handshake Authentication Protocol (CHAP) 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

### **CONFIG\_LWIP\_PPP\_MPPE\_SUPPORT**

Enable MPPE support

*Found in: [Component config](#) > [LWIP](#)*

Enable Microsoft Point-to-Point Encryption (MPPE) support

### **CONFIG\_LWIP\_PPP\_DEBUG\_ON**

Enable PPP debug log output

*Found in: [Component config](#) > [LWIP](#)*

Enable PPP debug log output

### **ICMP** Contains:

- [CONFIG\\_LWIP\\_MULTICAST\\_PING](#)
- [CONFIG\\_LWIP\\_BROADCAST\\_PING](#)

### **CONFIG\_LWIP\_MULTICAST\_PING**

Respond to multicast pings

*Found in: [Component config](#) > [LWIP](#) > [ICMP](#)*

### **CONFIG\_LWIP\_BROADCAST\_PING**

Respond to broadcast pings

*Found in: [Component config](#) > [LWIP](#) > [ICMP](#)*

### **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.

**SNTP** Contains:

- `CONFIG_LWIP_DHCP_MAX_NTP_SERVERS`
- `CONFIG_LWIP_SNTP_UPDATE_DELAY`

### CONFIG\_LWIP\_DHCP\_MAX\_NTP\_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 `sntp_setserver/sntp_setservername` functions is limited to this value.

### CONFIG\_LWIP\_SNTP\_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).

**mbedTLS** Contains:

- `CONFIG_MBEDTLS_MEM_ALLOC_MODE`
- `CONFIG_MBEDTLS_SSL_MAX_CONTENT_LEN`
- `CONFIG_MBEDTLS_ASYMMETRIC_CONTENT_LEN`
- `CONFIG_MBEDTLS_DEBUG`
- `CONFIG_MBEDTLS_ECP_RESTARTABLE`
- `CONFIG_MBEDTLS_CMAC_C`
- `CONFIG_MBEDTLS_HARDWARE_AES`
- `CONFIG_MBEDTLS_HARDWARE_MPI`
- `CONFIG_MBEDTLS_HARDWARE_SHA`
- `CONFIG_MBEDTLS_HAVE_TIME`
- `CONFIG_MBEDTLS_TLS_MODE`
- *TLS Key Exchange Methods*
- `CONFIG_MBEDTLS_SSL_RENEGOTIATION`
- `CONFIG_MBEDTLS_SSL_PROTO_SSL3`
- `CONFIG_MBEDTLS_SSL_PROTO_TLS1`
- `CONFIG_MBEDTLS_SSL_PROTO_TLS1_1`
- `CONFIG_MBEDTLS_SSL_PROTO_TLS1_2`
- `CONFIG_MBEDTLS_SSL_PROTO_DTLS`
- `CONFIG_MBEDTLS_SSL_ALPN`
- `CONFIG_MBEDTLS_CLIENT_SSL_SESSION_TICKETS`
- `CONFIG_MBEDTLS_SERVER_SSL_SESSION_TICKETS`
- *Symmetric Ciphers*
- `CONFIG_MBEDTLS_RIPEMD160_C`
- *Certificates*
- `CONFIG_MBEDTLS_ECP_C`



## 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

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.

### 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)

## 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).

## CONFIG\_MBEDTLS\_ASYMMETRIC\_CONTENT\_LEN

Asymmetric in/out fragment length

Found in: *Component config* > *MBEDTLS*

If enabled, this option allows customizing TLS in/out fragment length in asymmetric way. Please note that enabling this with default values saves 12KB of dynamic memory per TLS connection.

## CONFIG\_MBEDTLS\_SSL\_IN\_CONTENT\_LEN

TLS maximum incoming fragment length

Found in: *Component config* > *MBEDTLS* > *CONFIG\_MBEDTLS\_ASYMMETRIC\_CONTENT\_LEN*

This defines maximum incoming fragment length, overriding default maximum content length (MBEDTLS\_SSL\_MAX\_CONTENT\_LEN).

### CONFIG\_MBEDTLS\_SSL\_OUT\_CONTENT\_LEN

TLS maximum outgoing fragment length

*Found in: [Component config](#) > [mbedtls](#) > [CONFIG\\_MBEDTLS\\_ASYMMETRIC\\_CONTENT\\_LEN](#)*

This defines maximum outgoing fragment length, overriding default maximum content length (MBEDTLS\_SSL\_MAX\_CONTENT\_LEN).

### 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.

### 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)

### 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.

### 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.

### 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.

### **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 4096 bit results.

These operations are used by RSA.

### **CONFIG\_MBEDTLS\_MPI\_USE\_INTERRUPT**

Use interrupt for MPI operations

*Found in: [Component config > mbedTLS > CONFIG\\_MBEDTLS\\_HARDWARE\\_MPI](#)*

Use an interrupt to coordinate MPI operations.

This allows other code to run on the CPU while an MPI operation is pending. Otherwise the CPU busy-waits.

### **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, 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.

### **CONFIG\_MBEDTLS\_HAVE\_TIME**

Enable mbedtls time

*Found in: [Component config > mbedTLS](#)*

System has time.h and time(). The time does not need to be correct, only time differences are used.

### **CONFIG\_MBEDTLS\_HAVE\_TIME\_DATE**

Enable mbedtls certificate expiry check

*Found in: [Component config > mbedTLS > CONFIG\\_MBEDTLS\\_HAVE\\_TIME](#)*

System has time.h and time(), gmtime() and the clock is correct. The time needs to be correct (not necessarily very accurate, but at least the date should be correct). This is used to verify the validity period of X.509 certificates.

It is suggested that you should get the real time by “SNTP” .

## 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\\_PSK\\_MODES](#)
- [CONFIG\\_MBEDTLS\\_KEY\\_EXCHANGE\\_RSA](#)
- [CONFIG\\_MBEDTLS\\_KEY\\_EXCHANGE\\_DHE\\_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.

## 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.

## 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.

## 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.

### **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.

### **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-

### **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-

### **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.

### **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-

### **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-

### **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-

### **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-

### **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.

### **CONFIG\_MBEDTLS\_SSL\_PROTO\_SSL3**

Legacy SSL 3.0 support

*Found in:* [Component config > mbedTLS](#)

Support the legacy SSL 3.0 protocol. Most servers will speak a newer TLS protocol these days.

### **CONFIG\_MBEDTLS\_SSL\_PROTO\_TLS1**

Support TLS 1.0 protocol

*Found in:* [Component config > mbedTLS](#)

### **CONFIG\_MBEDTLS\_SSL\_PROTO\_TLS1\_1**

Support TLS 1.1 protocol

*Found in:* [Component config > mbedTLS](#)

### **CONFIG\_MBEDTLS\_SSL\_PROTO\_TLS1\_2**

Support TLS 1.2 protocol

*Found in:* [Component config > mbedTLS](#)

### **CONFIG\_MBEDTLS\_SSL\_PROTO\_DTLS**

Support DTLS protocol (all versions)

*Found in:* [Component config > mbedTLS](#)

Requires TLS 1.1 to be enabled for DTLS 1.0 Requires TLS 1.2 to be enabled for DTLS 1.2

### **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.

### 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.

### 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.

### Symmetric Ciphers Contains:

- [CONFIG\\_MBEDTLS\\_AES\\_C](#)
- [CONFIG\\_MBEDTLS\\_CAMELLIA\\_C](#)
- [CONFIG\\_MBEDTLS\\_DES\\_C](#)
- [CONFIG\\_MBEDTLS\\_RC4\\_MODE](#)
- [CONFIG\\_MBEDTLS\\_BLOWFISH\\_C](#)
- [CONFIG\\_MBEDTLS\\_XTEA\\_C](#)
- [CONFIG\\_MBEDTLS\\_CCM\\_C](#)
- [CONFIG\\_MBEDTLS\\_GCM\\_C](#)

### CONFIG\_MBEDTLS\_AES\_C

AES block cipher

*Found in: [Component config](#) > [mbedtls](#) > [Symmetric Ciphers](#)*

### CONFIG\_MBEDTLS\_CAMELLIA\_C

Camellia block cipher

*Found in: [Component config](#) > [mbedtls](#) > [Symmetric Ciphers](#)*

### 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.

### CONFIG\_MBEDTLS\_RC4\_MODE

RC4 Stream Cipher (legacy, insecure)

*Found in: [Component config](#) > [mbedtls](#) > [Symmetric Ciphers](#)*

ARCFOUR (RC4) stream cipher can be disabled entirely, enabled but not added to default ciphersuites, or enabled completely.

Please consider the security implications before enabling RC4.

#### Available options:

- Disabled (MBEDTLS\_RC4\_DISABLED)

- Enabled, not in default ciphersuites (MBEDTLS\_RC4\_ENABLED\_NO\_DEFAULT)
- Enabled (MBEDTLS\_RC4\_ENABLED)

### 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.

### CONFIG\_MBEDTLS\_XTEA\_C

XTEA block cipher

*Found in: Component config > mbedTLS > Symmetric Ciphers*

Enables the XTEA block cipher.

### 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.

### 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.

### CONFIG\_MBEDTLS\_RIPEMD160\_C

Enable RIPEMD-160 hash algorithm

*Found in: Component config > mbedTLS*

Enable the RIPEMD-160 hash algorithm.

**Certificates** Contains:

- *CONFIG\_MBEDTLS\_PEM\_PARSE\_C*
- *CONFIG\_MBEDTLS\_PEM\_WRITE\_C*
- *CONFIG\_MBEDTLS\_X509\_CRL\_PARSE\_C*
- *CONFIG\_MBEDTLS\_X509\_CSR\_PARSE\_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.

### **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.

### **CONFIG\_MBEDTLS\_X509\_CRL\_PARSE\_C**

X.509 CRL parsing

*Found in: Component config > mbedTLS > Certificates*

Support for parsing X.509 Certificate Revocation Lists.

### **CONFIG\_MBEDTLS\_X509\_CSR\_PARSE\_C**

X.509 CSR parsing

*Found in: Component config > mbedTLS > Certificates*

Support for parsing X.509 Certificate Signing Requests

### **CONFIG\_MBEDTLS\_ECP\_C**

Elliptic Curve Ciphers

*Found in: Component config > mbedTLS*

Contains:

- *CONFIG\_MBEDTLS\_ECDH\_C*
- *CONFIG\_MBEDTLS\_ECP\_DP\_SECP192R1\_ENABLED*
- *CONFIG\_MBEDTLS\_ECP\_DP\_SECP224R1\_ENABLED*
- *CONFIG\_MBEDTLS\_ECP\_DP\_SECP256R1\_ENABLED*
- *CONFIG\_MBEDTLS\_ECP\_DP\_SECP384R1\_ENABLED*
- *CONFIG\_MBEDTLS\_ECP\_DP\_SECP521R1\_ENABLED*
- *CONFIG\_MBEDTLS\_ECP\_DP\_SECP192K1\_ENABLED*
- *CONFIG\_MBEDTLS\_ECP\_DP\_SECP224K1\_ENABLED*
- *CONFIG\_MBEDTLS\_ECP\_DP\_SECP256K1\_ENABLED*
- *CONFIG\_MBEDTLS\_ECP\_DP\_BP256R1\_ENABLED*
- *CONFIG\_MBEDTLS\_ECP\_DP\_BP384R1\_ENABLED*
- *CONFIG\_MBEDTLS\_ECP\_DP\_BP512R1\_ENABLED*
- *CONFIG\_MBEDTLS\_ECP\_DP\_CURVE25519\_ENABLED*
- *CONFIG\_MBEDTLS\_ECP\_NIST\_OPTIM*

### **CONFIG\_MBEDTLS\_ECDH\_C**

Elliptic Curve Diffie-Hellman (ECDH)

*Found in:* [Component config](#) > [mbedtls](#) > [CONFIG\\_MBEDTLS\\_ECP\\_C](#)

Enable ECDH. Needed to use ECDHE-xxx TLS ciphersuites.

### **CONFIG\_MBEDTLS\_ECDSA\_C**

Elliptic Curve DSA

*Found in:* [Component config](#) > [mbedtls](#) > [CONFIG\\_MBEDTLS\\_ECP\\_C](#) > [CONFIG\\_MBEDTLS\\_ECDH\\_C](#)

Enable ECDSA. Needed to use ECDSA-xxx TLS ciphersuites.

### **CONFIG\_MBEDTLS\_ECP\_DP\_SECP192R1\_ENABLED**

Enable SECP192R1 curve

*Found in:* [Component config](#) > [mbedtls](#) > [CONFIG\\_MBEDTLS\\_ECP\\_C](#)

Enable support for SECP192R1 Elliptic Curve.

### **CONFIG\_MBEDTLS\_ECP\_DP\_SECP224R1\_ENABLED**

Enable SECP224R1 curve

*Found in:* [Component config](#) > [mbedtls](#) > [CONFIG\\_MBEDTLS\\_ECP\\_C](#)

Enable support for SECP224R1 Elliptic Curve.

### **CONFIG\_MBEDTLS\_ECP\_DP\_SECP256R1\_ENABLED**

Enable SECP256R1 curve

*Found in:* [Component config](#) > [mbedtls](#) > [CONFIG\\_MBEDTLS\\_ECP\\_C](#)

Enable support for SECP256R1 Elliptic Curve.

### **CONFIG\_MBEDTLS\_ECP\_DP\_SECP384R1\_ENABLED**

Enable SECP384R1 curve

*Found in:* [Component config](#) > [mbedtls](#) > [CONFIG\\_MBEDTLS\\_ECP\\_C](#)

Enable support for SECP384R1 Elliptic Curve.

### **CONFIG\_MBEDTLS\_ECP\_DP\_SECP521R1\_ENABLED**

Enable SECP521R1 curve

*Found in:* [Component config](#) > [mbedtls](#) > [CONFIG\\_MBEDTLS\\_ECP\\_C](#)

Enable support for SECP521R1 Elliptic Curve.

### **CONFIG\_MBEDTLS\_ECP\_DP\_SECP192K1\_ENABLED**

Enable SECP192K1 curve

*Found in:* [Component config](#) > [mbedtls](#) > [CONFIG\\_MBEDTLS\\_ECP\\_C](#)

Enable support for SECP192K1 Elliptic Curve.

### **CONFIG\_MBEDTLS\_ECP\_DP\_SECP224K1\_ENABLED**

Enable SECP224K1 curve

*Found in: [Component config](#) > [mbedtls](#) > [CONFIG\\_MBEDTLS\\_ECP\\_C](#)*

Enable support for SECP224K1 Elliptic Curve.

### **CONFIG\_MBEDTLS\_ECP\_DP\_SECP256K1\_ENABLED**

Enable SECP256K1 curve

*Found in: [Component config](#) > [mbedtls](#) > [CONFIG\\_MBEDTLS\\_ECP\\_C](#)*

Enable support for SECP256K1 Elliptic Curve.

### **CONFIG\_MBEDTLS\_ECP\_DP\_BP256R1\_ENABLED**

Enable BP256R1 curve

*Found in: [Component config](#) > [mbedtls](#) > [CONFIG\\_MBEDTLS\\_ECP\\_C](#)*

support for DP Elliptic Curve.

### **CONFIG\_MBEDTLS\_ECP\_DP\_BP384R1\_ENABLED**

Enable BP384R1 curve

*Found in: [Component config](#) > [mbedtls](#) > [CONFIG\\_MBEDTLS\\_ECP\\_C](#)*

support for DP Elliptic Curve.

### **CONFIG\_MBEDTLS\_ECP\_DP\_BP512R1\_ENABLED**

Enable BP512R1 curve

*Found in: [Component config](#) > [mbedtls](#) > [CONFIG\\_MBEDTLS\\_ECP\\_C](#)*

support for DP Elliptic Curve.

### **CONFIG\_MBEDTLS\_ECP\_DP\_CURVE25519\_ENABLED**

Enable CURVE25519 curve

*Found in: [Component config](#) > [mbedtls](#) > [CONFIG\\_MBEDTLS\\_ECP\\_C](#)*

Enable support for CURVE25519 Elliptic Curve.

### **CONFIG\_MBEDTLS\_ECP\_NIST\_OPTIM**

NIST ‘modulo p’ optimisations

*Found in: [Component config](#) > [mbedtls](#) > [CONFIG\\_MBEDTLS\\_ECP\\_C](#)*

NIST ‘modulo p’ optimisations increase Elliptic Curve operation performance.

Disabling this option saves some code size.

# end of Elliptic Curve options

**mDNS** Contains:

- [CONFIG\\_MDNS\\_MAX\\_SERVICES](#)

### CONFIG\_MDNS\_MAX\_SERVICES

Max number of services

Found in: *Component config > mDNS*

Services take up a certain amount of memory, and allowing fewer services to be open at the same time conserves memory. Specify the maximum amount of services here. The valid value is from 1 to 64.

### ESP-MQTT Configurations Contains:

- *CONFIG\_MQTT\_PROTOCOL\_311*
- *CONFIG\_MQTT\_TRANSPORT\_SSL*
- *CONFIG\_MQTT\_TRANSPORT\_WEBSOCKET*
- *CONFIG\_MQTT\_USE\_CUSTOM\_CONFIG*
- *CONFIG\_MQTT\_TASK\_CORE\_SELECTION\_ENABLED*
- *CONFIG\_MQTT\_CUSTOM\_OUTBOX*

### 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

### CONFIG\_MQTT\_TRANSPORT\_SSL

Enable MQTT over SSL

Found in: *Component config > ESP-MQTT Configurations*

Enable MQTT transport over SSL with mbedtls

### CONFIG\_MQTT\_TRANSPORT\_WEBSOCKET

Enable MQTT over Websocket

Found in: *Component config > ESP-MQTT Configurations*

Enable MQTT transport over Websocket.

### CONFIG\_MQTT\_TRANSPORT\_WEBSOCKET\_SECURE

Enable MQTT over Websocket Secure

Found in: *Component config > ESP-MQTT Configurations > CONFIG\_MQTT\_TRANSPORT\_WEBSOCKET*

Enable MQTT transport over Websocket Secure.

### CONFIG\_MQTT\_USE\_CUSTOM\_CONFIG

MQTT Using custom configurations

Found in: *Component config > ESP-MQTT Configurations*

Custom MQTT configurations.

### **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

### **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

### **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

### **CONFIG\_MQTT\_WSS\_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

### **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

### **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

### **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

### **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.

### 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).

**Newlib** Contains:

- [CONFIG\\_NEWLIB\\_STDOUT\\_LINE\\_ENDING](#)
- [CONFIG\\_NEWLIB\\_STDIN\\_LINE\\_ENDING](#)
- [CONFIG\\_NEWLIB\\_NANO\\_FORMAT](#)

### CONFIG\_NEWLIB\_STDOUT\_LINE\_ENDING

Line ending for UART output

*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).

#### Available options:

- CRLF (NEWLIB\_STDOUT\_LINE\_ENDING\_CRLF)
- LF (NEWLIB\_STDOUT\_LINE\_ENDING\_LF)
- CR (NEWLIB\_STDOUT\_LINE\_ENDING\_CR)

### CONFIG\_NEWLIB\_STDIN\_LINE\_ENDING

Line ending for UART input

*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).

**Available options:**

- CRLF (NEWLIB\_STDIN\_LINE\_ENDING\_CRLF)
- LF (NEWLIB\_STDIN\_LINE\_ENDING\_LF)
- CR (NEWLIB\_STDIN\_LINE\_ENDING\_CR)

### CONFIG\_NEWLIB\_NANO\_FORMAT

Enable 'nano' formatting options for printf/scanf family

*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' : <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.

**NVS** Contains:

- [CONFIG\\_NVS\\_ENCRYPTION](#)

### 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.

**OpenSSL** Contains:

- [CONFIG\\_OPENSSL\\_DEBUG](#)
- [CONFIG\\_OPENSSL\\_ASSERT](#)

### CONFIG\_OPENSSL\_DEBUG

Enable OpenSSL debugging

*Found in: [Component config](#) > [OpenSSL](#)*

Enable OpenSSL debugging function.

If the option is enabled, "SSL\_DEBUG" works.

### CONFIG\_OPENSSL\_DEBUG\_LEVEL

OpenSSL debugging level

Found in: *Component config > OpenSSL > CONFIG\_OPENSSL\_DEBUG*

OpenSSL debugging level.

Only function whose debugging level is higher than “OPENSSL\_DEBUG\_LEVEL” works.

For example: If OPENSSL\_DEBUG\_LEVEL = 2, you use function “SSL\_DEBUG(1, “malloc failed”)” . Because  $1 < 2$ , it will not print.

### CONFIG\_OPENSSL\_LOWLEVEL\_DEBUG

Enable OpenSSL low-level module debugging

Found in: *Component config > OpenSSL > CONFIG\_OPENSSL\_DEBUG*

If the option is enabled, low-level module debugging function of OpenSSL is enabled, e.g. mbedtls internal debugging function.

### CONFIG\_OPENSSL\_ASSERT

Select OpenSSL assert function

Found in: *Component config > OpenSSL*

OpenSSL function needs “assert” function to check if input parameters are valid.

If you want to use assert debugging function, “OPENSSL\_DEBUG” should be enabled.

#### Available options:

- Do nothing (OPENSSL\_ASSERT\_DO\_NOTHING)  
Do nothing and “SSL\_ASSERT” does not work.
- Check and exit (OPENSSL\_ASSERT\_EXIT)  
Enable assert exiting, it will check and return error code.
- Show debugging message (OPENSSL\_ASSERT\_DEBUG)  
Enable assert debugging, it will check and show debugging message.
- Show debugging message and exit (OPENSSL\_ASSERT\_DEBUG\_EXIT)  
Enable assert debugging and exiting, it will check, show debugging message and return error code.
- Show debugging message and block (OPENSSL\_ASSERT\_DEBUG\_BLOCK)  
Enable assert debugging and blocking, it will check, show debugging message and block by “while (1);” .

**PThreads** Contains:

- *CONFIG\_PTHREAD\_TASK\_PRIO\_DEFAULT*
- *CONFIG\_PTHREAD\_TASK\_STACK\_SIZE\_DEFAULT*
- *CONFIG\_PTHREAD\_STACK\_MIN*
- *CONFIG\_PTHREAD\_TASK\_CORE\_DEFAULT*
- *CONFIG\_PTHREAD\_TASK\_NAME\_DEFAULT*

### CONFIG\_PTHREAD\_TASK\_PRIO\_DEFAULT

Default task priority

Found in: *Component config > PThreads*

Priority used to create new tasks with default pthread parameters.



### 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.

### 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

### 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.

### SPI Flash driver

 Contains:

- [CONFIG\\_SPI\\_FLASH\\_VERIFY\\_WRITE](#)
- [CONFIG\\_SPI\\_FLASH\\_ENABLE\\_COUNTERS](#)
- [CONFIG\\_SPI\\_FLASH\\_ROM\\_DRIVER\\_PATCH](#)
- [CONFIG\\_SPI\\_FLASH\\_DANGEROUS\\_WRITE](#)
- [CONFIG\\_SPI\\_FLASH\\_USE\\_LEGACY\\_IMPL](#)
- [CONFIG\\_SPI\\_FLASH\\_BYPASS\\_BLOCK\\_ERASE](#)
- [CONFIG\\_SPI\\_FLASH\\_YIELD\\_DURING\\_ERASE](#)
- [CONFIG\\_SPI\\_FLASH\\_SIZE\\_OVERRIDE](#)
- *Auto-detect flash chips*
- [CONFIG\\_SPI\\_FLASH\\_ENABLE\\_ENCRYPTED\\_READ\\_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.

### 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.

### 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.

### 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.

### 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 is needed to write to flash on ESP32-D2WD, and any configuration where external SPI flash is connected to non-default pins.

### 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\_USE\_LEGACY\_IMPL**

Use the legacy implementation before IDF v4.0

*Found in: [Component config](#) > [SPI Flash driver](#)*

The implementation of SPI flash has been greatly changed in IDF v4.0. Enable this option to use the legacy implementation.

### **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.

### **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.

### **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.

### **CONFIG\_SPI\_FLASH\_ERASE\_YIELD\_TICKS**

CPU release time (tick)

*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.

### **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.

**Auto-detect flash chips** Contains:

- [CONFIG\\_SPI\\_FLASH\\_SUPPORT\\_ISSI\\_CHIP](#)
- [CONFIG\\_SPI\\_FLASH\\_SUPPORT\\_GD\\_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.

## 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.

## 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.

## SPIFFS Configuration

 Contains:

- *CONFIG\_SPIFFS\_MAX\_PARTITIONS*
- *SPIFFS Cache Configuration*
- *CONFIG\_SPIFFS\_PAGE\_CHECK*
- *CONFIG\_SPIFFS\_GC\_MAX\_RUNS*
- *CONFIG\_SPIFFS\_GC\_STATS*
- *CONFIG\_SPIFFS\_PAGE\_SIZE*
- *CONFIG\_SPIFFS\_OBJ\_NAME\_LEN*
- *CONFIG\_SPIFFS\_USE\_MAGIC*
- *CONFIG\_SPIFFS\_META\_LENGTH*
- *CONFIG\_SPIFFS\_USE\_MTIME*
- *Debug Configuration*

## CONFIG\_SPIFFS\_MAX\_PARTITIONS

Maximum Number of Partitions

*Found in: Component config > SPIFFS Configuration*

Define maximum number of partitions that can be mounted.

## SPIFFS Cache Configuration

 Contains:

- *CONFIG\_SPIFFS\_CACHE*

### **CONFIG\_SPIFFS\_CACHE**

Enable SPIFFS Cache

*Found in: [Component config](#) > [SPIFFS Configuration](#) > [SPIFFS Cache Configuration](#)*

Enables/disable memory read caching of nucleus file system operations.

### **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.

### **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.

### **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.

### **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.

### **CONFIG\_SPIFFS\_GC\_STATS**

Enable SPIFFS GC Statistics

*Found in: [Component config](#) > [SPIFFS Configuration](#)*

Enable/disable statistics on gc. Debug/test purpose only.

### **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.

### 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.

### 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.

### 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.

### 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.

### 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.

**Debug Configuration** Contains:

- [CONFIG\\_SPIFFS\\_DBG](#)
- [CONFIG\\_SPIFFS\\_API\\_DBG](#)
- [CONFIG\\_SPIFFS\\_GC\\_DBG](#)
- [CONFIG\\_SPIFFS\\_CACHE\\_DBG](#)
- [CONFIG\\_SPIFFS\\_CHECK\\_DBG](#)
- [CONFIG\\_SPIFFS\\_TEST\\_VISUALISATION](#)

### **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.

### **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.

### **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.

### **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.

### **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.

### **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.

### **TCP/IP Adapter** Contains:

- [CONFIG\\_NETIF\\_IP\\_LOST\\_TIMER\\_INTERVAL](#)
- [CONFIG\\_NETIF\\_USE\\_TCPIP\\_STACK\\_LIB](#)

### **CONFIG\_NETIF\_IP\_LOST\_TIMER\_INTERVAL**

IP Address lost timer interval (seconds)

*Found in: [Component config](#) > [TCP/IP 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.

## CONFIG\_NETIF\_USE\_TCPIP\_STACK\_LIB

TCP/IP Stack Library

*Found in: Component config > TCP/IP Adapter*

Choose the TCP/IP Stack to work, for example, LwIP, uIP, etc.

### Available options:

- LwIP (TCPIP\_LWIP)  
lwIP is a small independent implementation of the TCP/IP protocol suite.

## Unity unit testing library Contains:

- *CONFIG\_UNITY\_ENABLE\_FLOAT*
- *CONFIG\_UNITY\_ENABLE\_DOUBLE*
- *CONFIG\_UNITY\_ENABLE\_COLOR*
- *CONFIG\_UNITY\_ENABLE\_IDF\_TEST\_RUNNER*
- *CONFIG\_UNITY\_ENABLE\_FIXTURE*
- *CONFIG\_UNITY\_ENABLE\_BACKTRACE\_ON\_FAIL*

## 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.

## 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.

## 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.

## 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.



### 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.

### 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.

**Virtual file system** Contains:

- *CONFIG\_VFS\_SUPPRESS\_SELECT\_DEBUG\_OUTPUT*
- *CONFIG\_VFS\_SUPPORT\_TERMIOS*
- *Host File System I/O (Semihosting)*

### CONFIG\_VFS\_SUPPRESS\_SELECT\_DEBUG\_OUTPUT

Suppress select() related debug outputs

*Found in: Component config > Virtual file system*

Select() related functions might produce an inconveniently 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.

### CONFIG\_VFS\_SUPPORT\_TERMIOS

Add support for termios.h

*Found in: Component config > Virtual file system*

Disabling this option can save memory when the support for termios.h is not required.

**Host File System I/O (Semihosting)** Contains:

- *CONFIG\_SEMIHOSTFS\_MAX\_MOUNT\_POINTS*
- *CONFIG\_SEMIHOSTFS\_HOST\_PATH\_MAX\_LEN*

### CONFIG\_SEMIHOSTFS\_MAX\_MOUNT\_POINTS

Maximum number of the host filesystem mount points

*Found in: Component config > Virtual file system > Host File System I/O (Semihosting)*

Define maximum number of host filesystem mount points.

### CONFIG\_SEMIHOSTFS\_HOST\_PATH\_MAX\_LEN

Maximum path length for the host base directory

*Found in: Component config > Virtual file system > Host File System I/O (Semihosting)*

Define maximum path length for the host base directory which is to be mounted. If host path passed to esp\_vfs\_semihost\_register() is longer than this value it will be truncated.

**Wear Levelling** Contains:

- [CONFIG\\_WL\\_SECTOR\\_SIZE](#)
- [CONFIG\\_WL\\_SECTOR\\_MODE](#)

### 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\\_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

## 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.

**Supplicant** Contains:

- [CONFIG\\_WPA\\_MBEDTLS\\_CRYPT0](#)
- [CONFIG\\_WPA\\_WPS\\_WARS](#)
- [CONFIG\\_WPA\\_DEBUG\\_PRINT](#)

## CONFIG\_WPA\_MBEDTLS\_CRYPT0

Use MbedTLS crypto API' s

*Found in: Component config > Supplicant*

Select this option to use MbedTLS crypto API' s which utilize hardware acceleration.

## CONFIG\_WPA\_WPS\_WARS

Add WPS Inter operatability Fixes

*Found in: Component config > Supplicant*

Select this option to enable WPS related IOT fixes with different APs. This option fixes IOT related issues with APs which do not follow some of the standards of WPS-2.0 specification. These do not include any of the security related bypassing, just simple configuration corrections.

Current fixes under this flag. 1. Allow NULL-padded WPS attributes: Some APs keep NULL-padding at the end of some variable length WPS Attributes. This is not as par the WPS2.0 specs, but to avoid interop issues, ignore the padding by reducing the attribute length by 1. 2. Bypass WPS-Config method validation: Some APs set display/pbc button bit without setting virtual/physical display/button bit which will cause M2 validation fail, bypassing WPS-Config method validation.

## 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.

## Compatibility options

Contains:

- [CONFIG\\_LEGACY\\_INCLUDE\\_COMMON\\_HEADERS](#)

## CONFIG\_LEGACY\_INCLUDE\_COMMON\_HEADERS

Include headers across components as before IDF v4.0

Found in: *Compatibility options*

Soc, esp32, and driver components, the most common components. Some header of these components are included implicitly by headers of other components before IDF v4.0. It's not required for high-level components, but still included through long header chain everywhere.

This is harmful to the modularity. So it's changed in IDF v4.0.

You can still include these headers in a legacy way until it is totally deprecated by enable this option.

### Deprecated options and their replacements

- CONFIG\_A2DP\_ENABLE ([CONFIG\\_BT\\_A2DP\\_ENABLE](#))
- CONFIG\_ADC2\_DISABLE\_DAC ([CONFIG\\_ADC\\_DISABLE\\_DAC](#))
- 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\_BLE\_ACTIVE\_SCAN\_REPORT\_ADV\_SCAN\_RSP\_INDIVIDUALLY ([CONFIG\\_BT\\_BLE\\_ACT\\_SCAN\\_REP\\_ADV\\_SCAN](#))
- CONFIG\_BLE\_ADV\_REPORT\_DISCARD\_THRSHOLD ([CONFIG\\_BTDM\\_BLE\\_ADV\\_REPORT\\_DISCARD\\_THRSHOLD](#))
- 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\\_BLE\\_MESH\\_GATT\\_PROXY\\_SERVER](#))
- 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\_BROWNOUT\_DET ([CONFIG\\_ESP32\\_BROWNOUT\\_DET](#))
- CONFIG\_BROWNOUT\_DET\_LVL\_SEL ([CONFIG\\_ESP32\\_BROWNOUT\\_DET\\_LVL\\_SEL](#))
  - CONFIG\_BROWNOUT\_DET\_LVL\_SEL\_0
  - CONFIG\_BROWNOUT\_DET\_LVL\_SEL\_1
  - CONFIG\_BROWNOUT\_DET\_LVL\_SEL\_2
  - CONFIG\_BROWNOUT\_DET\_LVL\_SEL\_3
  - CONFIG\_BROWNOUT\_DET\_LVL\_SEL\_4
  - CONFIG\_BROWNOUT\_DET\_LVL\_SEL\_5
  - CONFIG\_BROWNOUT\_DET\_LVL\_SEL\_6
  - CONFIG\_BROWNOUT\_DET\_LVL\_SEL\_7
- CONFIG\_BTC\_TASK\_STACK\_SIZE ([CONFIG\\_BT\\_BTC\\_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\\_MODEM\\_SLEEP](#))
- CONFIG\_BTDM\_CONTROLLER\_PINNED\_TO\_CORE\_CHOICE ([CON-FIG\\_BTDM\\_CTRL\\_PINNED\\_TO\\_CORE\\_CHOICE](#))
- CONFIG\_BTU\_TASK\_STACK\_SIZE ([CONFIG\\_BT\\_BTU\\_TASK\\_STACK\\_SIZE](#))
- CONFIG\_CLASSIC\_BT\_ENABLED ([CONFIG\\_BT\\_CLASSIC\\_ENABLED](#))
- CONFIG\_COMPATIBLE\_PRE\_V2\_1\_BOOTLOADERS ([CONFIG\\_ESP32\\_COMPATIBLE\\_PRE\\_V2\\_1\\_BOOTLOADERS](#))
- **CONFIG\_CONSOLE\_UART** ([CONFIG\\_ESP\\_CONSOLE\\_UART](#))
  - CONFIG\_CONSOLE\_UART\_DEFAULT
  - CONFIG\_CONSOLE\_UART\_CUSTOM
  - CONFIG\_CONSOLE\_UART\_NONE
- CONFIG\_CONSOLE\_UART\_BAUDRATE ([CONFIG\\_ESP\\_CONSOLE\\_UART\\_BAUDRATE](#))
- 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\_DISABLE\_BASIC\_ROM\_CONSOLE ([CONFIG\\_ESP32\\_DISABLE\\_BASIC\\_ROM\\_CONSOLE](#))
- CONFIG\_DISABLE\_GCC8\_WARNINGS ([CONFIG\\_COMPILER\\_DISABLE\\_GCC8\\_WARNINGS](#))
- 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\_APPTRACE\_POSTMORTEM\_FLUSH\_TRAX\_THRESH ([CON-FIG\\_ESP32\\_APPTRACE\\_POSTMORTEM\\_FLUSH\\_THRESH](#))
- CONFIG\_ESP32\_PTHREAD\_STACK\_MIN ([CONFIG\\_PTHREAD\\_STACK\\_MIN](#))
- 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\_CLOCK\_SOURCE** ([CONFIG\\_ESP32\\_RTC\\_CLK\\_SRC](#))
  - CONFIG\_ESP32\_RTC\_CLOCK\_SOURCE\_INTERNAL\_RC
  - CONFIG\_ESP32\_RTC\_CLOCK\_SOURCE\_EXTERNAL\_CRYSTAL
  - CONFIG\_ESP32\_RTC\_CLOCK\_SOURCE\_EXTERNAL\_OSC
  - CONFIG\_ESP32\_RTC\_CLOCK\_SOURCE\_INTERNAL\_8MD256
- CONFIG\_ESP32\_RTC\_EXTERNAL\_CRYSTAL\_ADDITIONAL\_CURRENT ([CON-FIG\\_ESP32\\_RTC\\_EXT\\_CRYST\\_ADDIT\\_CURRENT](#))
- CONFIG\_ESP\_GRATUITOUS\_ARP ([CONFIG\\_LWIP\\_ESP\\_GRATUITOUS\\_ARP](#))
- CONFIG\_ESP\_TCP\_KEEP\_CONNECTION\_WHEN\_IP\_CHANGES ([CON-FIG\\_LWIP\\_TCP\\_KEEP\\_CONNECTION\\_WHEN\\_IP\\_CHANGES](#))
- 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 ([CON-FIG\\_SECURE\\_FLASH\\_UART\\_BOOTLOADER\\_ALLOW\\_CACHE](#))
- CONFIG\_FLASH\_ENCRYPTION\_UART\_BOOTLOADER\_ALLOW\_DECRYPT ([CON-FIG\\_SECURE\\_FLASH\\_UART\\_BOOTLOADER\\_ALLOW\\_DEC](#))
- CONFIG\_FLASH\_ENCRYPTION\_UART\_BOOTLOADER\_ALLOW\_ENCRYPT ([CON-FIG\\_SECURE\\_FLASH\\_UART\\_BOOTLOADER\\_ALLOW\\_ENC](#))
- CONFIG\_GARP\_TMR\_INTERVAL ([CONFIG\\_LWIP\\_GARP\\_TMR\\_INTERVAL](#))
- CONFIG\_GATTC\_CACHE\_NVS\_FLASH ([CONFIG\\_BT\\_GATTC\\_CACHE\\_NVS\\_FLASH](#))
- CONFIG\_GATTC\_ENABLE ([CONFIG\\_BT\\_GATTC\\_ENABLE](#))
- CONFIG\_GATTS\_ENABLE ([CONFIG\\_BT\\_GATTS\\_ENABLE](#))
- CONFIG\_GDBSTUB\_MAX\_TASKS ([CONFIG\\_ESP\\_GDBSTUB\\_MAX\\_TASKS](#))
- CONFIG\_GDBSTUB\_SUPPORT\_TASKS ([CONFIG\\_ESP\\_GDBSTUB\\_SUPPORT\\_TASKS](#))
- **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\_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\_IP\_LOST\_TIMER\_INTERVAL (*CONFIG\_NETIF\_IP\_LOST\_TIMER\_INTERVAL*)
- CONFIG\_L2\_TO\_L3\_COPY (*CONFIG\_LWIP\_L2\_TO\_L3\_COPY*)
- CONFIG\_MAIN\_TASK\_STACK\_SIZE (*CONFIG\_ESP\_MAIN\_TASK\_STACK\_SIZE*)
- CONFIG\_MAKE\_WARN\_UNDEFINED\_VARIABLES (*CONFIG\_SDK\_MAKE\_WARN\_UNDEFINED\_VARIABLES*)
- CONFIG\_MB\_CONTROLLER\_NOTIFY\_QUEUE\_SIZE (*CONFIG\_FMB\_CONTROLLER\_NOTIFY\_QUEUE\_SIZE*)
- CONFIG\_MB\_CONTROLLER\_NOTIFY\_TIMEOUT (*CONFIG\_FMB\_CONTROLLER\_NOTIFY\_TIMEOUT*)
- CONFIG\_MB\_CONTROLLER\_SLAVE\_ID (*CONFIG\_FMB\_CONTROLLER\_SLAVE\_ID*)
- CONFIG\_MB\_CONTROLLER\_SLAVE\_ID\_SUPPORT (*CONFIG\_FMB\_CONTROLLER\_SLAVE\_ID\_SUPPORT*)
- CONFIG\_MB\_CONTROLLER\_STACK\_SIZE (*CONFIG\_FMB\_CONTROLLER\_STACK\_SIZE*)
- CONFIG\_MB\_EVENT\_QUEUE\_TIMEOUT (*CONFIG\_FMB\_EVENT\_QUEUE\_TIMEOUT*)
- CONFIG\_MB\_MASTER\_DELAY\_MS\_CONVERT (*CONFIG\_FMB\_MASTER\_DELAY\_MS\_CONVERT*)
- CONFIG\_MB\_MASTER\_TIMEOUT\_MS\_RESPOND (*CONFIG\_FMB\_MASTER\_TIMEOUT\_MS\_RESPOND*)
- CONFIG\_MB\_QUEUE\_LENGTH (*CONFIG\_FMB\_QUEUE\_LENGTH*)
- CONFIG\_MB\_SERIAL\_BUF\_SIZE (*CONFIG\_FMB\_SERIAL\_BUF\_SIZE*)
- CONFIG\_MB\_SERIAL\_TASK\_PRIO (*CONFIG\_FMB\_SERIAL\_TASK\_PRIO*)
- CONFIG\_MB\_SERIAL\_TASK\_STACK\_SIZE (*CONFIG\_FMB\_SERIAL\_TASK\_STACK\_SIZE*)
- CONFIG\_MB\_TIMER\_GROUP (*CONFIG\_FMB\_TIMER\_GROUP*)
- CONFIG\_MB\_TIMER\_INDEX (*CONFIG\_FMB\_TIMER\_INDEX*)
- CONFIG\_MB\_TIMER\_PORT\_ENABLED (*CONFIG\_FMB\_TIMER\_PORT\_ENABLED*)
- CONFIG\_MESH\_DUPLICATE\_SCAN\_CACHE\_SIZE (*CONFIG\_BTDM\_MESH\_DUPL\_SCAN\_CACHE\_SIZE*)
- CONFIG\_MONITOR\_BAUD\_OTHER\_VAL (*CONFIG\_ESPTOOLPY\_MONITOR\_BAUD\_OTHER\_VAL*)
- 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\_CRYPTOSTACK\_MBEDTLS (*CONFIG\_BT\_NIMBLE\_CRYPTOSTACK\_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\_ITVL (*CONFIG\_BT\_NIMBLE\_HS\_FLOW\_CTRL\_ITVL*)
- CONFIG\_NIMBLE\_HS\_FLOW\_CTRL\_THRESH (*CONFIG\_BT\_NIMBLE\_HS\_FLOW\_CTRL\_THRESH*)
- 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\_MEM\_ALLOC\_MODE\_INTERNAL
  - CONFIG\_NIMBLE\_MEM\_ALLOC\_MODE\_EXTERNAL
  - CONFIG\_NIMBLE\_MEM\_ALLOC\_MODE\_DEFAULT
- CONFIG\_NIMBLE\_MESH (*CONFIG\_BT\_NIMBLE\_MESH*)
- CONFIG\_NIMBLE\_MESH\_DEVICE\_NAME (*CONFIG\_BT\_NIMBLE\_MESH\_DEVICE\_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\_PROV (*CONFIG\_BT\_NIMBLE\_MESH\_PROV*)
- CONFIG\_NIMBLE\_MESH\_PROXY (*CONFIG\_BT\_NIMBLE\_MESH\_PROXY*)
- CONFIG\_NIMBLE\_MESH\_RELAY (*CONFIG\_BT\_NIMBLE\_MESH\_RELAY*)
- CONFIG\_NIMBLE\_MSYS1\_BLOCK\_COUNT (*CONFIG\_BT\_NIMBLE\_MSYS1\_BLOCK\_COUNT*)
- CONFIG\_NIMBLE\_NVS\_PERSIST (*CONFIG\_BT\_NIMBLE\_NVS\_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\_BT\_NIMBLE\_ROLE\_BROADCASTER*)
- CONFIG\_NIMBLE\_ROLE\_CENTRAL (*CONFIG\_BT\_NIMBLE\_ROLE\_CENTRAL*)
- CONFIG\_NIMBLE\_ROLE\_OBSERVER (*CONFIG\_BT\_NIMBLE\_ROLE\_OBSERVER*)
- CONFIG\_NIMBLE\_ROLE\_PERIPHERAL (*CONFIG\_BT\_NIMBLE\_ROLE\_PERIPHERAL*)
- CONFIG\_NIMBLE\_RPA\_TIMEOUT (*CONFIG\_BT\_NIMBLE\_RPA\_TIMEOUT*)
- CONFIG\_NIMBLE\_SM\_LEGACY (*CONFIG\_BT\_NIMBLE\_SM\_LEGACY*)
- CONFIG\_NIMBLE\_SM\_SC (*CONFIG\_BT\_NIMBLE\_SM\_SC*)
- CONFIG\_NIMBLE\_SM\_SC\_DEBUG\_KEYS (*CONFIG\_BT\_NIMBLE\_SM\_SC\_DEBUG\_KEYS*)
- CONFIG\_NIMBLE\_SVC\_GAP\_APPEARANCE (*CONFIG\_BT\_NIMBLE\_SVC\_GAP\_APPEARANCE*)
- CONFIG\_NIMBLE\_SVC\_GAP\_DEVICE\_NAME (*CONFIG\_BT\_NIMBLE\_SVC\_GAP\_DEVICE\_NAME*)
- CONFIG\_NIMBLE\_TASK\_STACK\_SIZE (*CONFIG\_BT\_NIMBLE\_TASK\_STACK\_SIZE*)
- CONFIG\_NO\_BLOBS (*CONFIG\_ESP32\_NO\_BLOBS*)
- **CONFIG\_OPTIMIZATION\_ASSERTION\_LEVEL** (*CONFIG\_COMPILER\_OPTIMIZATION\_ASSERTION\_LEVEL*)
  - CONFIG\_OPTIMIZATION\_ASSERTIONS\_ENABLED
  - CONFIG\_OPTIMIZATION\_ASSERTIONS\_SILENT
  - CONFIG\_OPTIMIZATION\_ASSERTIONS\_DISABLED
- **CONFIG\_OPTIMIZATION\_COMPILER** (*CONFIG\_COMPILER\_OPTIMIZATION*)
  - CONFIG\_OPTIMIZATION\_LEVEL\_DEBUG
  - CONFIG\_OPTIMIZATION\_LEVEL\_RELEASE
- CONFIG\_POST\_EVENTS\_FROM\_IRAM\_ISR (*CONFIG\_ESP\_EVENT\_POST\_FROM\_IRAM\_ISR*)
- CONFIG\_POST\_EVENTS\_FROM\_ISR (*CONFIG\_ESP\_EVENT\_POST\_FROM\_ISR*)
- CONFIG\_PPP\_CHAP\_SUPPORT (*CONFIG\_LWIP\_PPP\_CHAP\_SUPPORT*)
- CONFIG\_PPP\_DEBUG\_ON (*CONFIG\_LWIP\_PPP\_DEBUG\_ON*)
- CONFIG\_PPP\_MPPE\_SUPPORT (*CONFIG\_LWIP\_PPP\_MPPE\_SUPPORT*)
- CONFIG\_PPP\_MSCHAP\_SUPPORT (*CONFIG\_LWIP\_PPP\_MSCHAP\_SUPPORT*)
- CONFIG\_PPP\_NOTIFY\_PHASE\_SUPPORT (*CONFIG\_LWIP\_PPP\_NOTIFY\_PHASE\_SUPPORT*)
- CONFIG\_PPP\_PAP\_SUPPORT (*CONFIG\_LWIP\_PPP\_PAP\_SUPPORT*)
- CONFIG\_PPP\_SUPPORT (*CONFIG\_LWIP\_PPP\_SUPPORT*)
- CONFIG\_PYTHON (*CONFIG\_SDK\_PYTHON*)
- CONFIG\_REDUCE\_PHY\_TX\_POWER (*CONFIG\_ESP32\_REDUCE\_PHY\_TX\_POWER*)
- CONFIG\_SMP\_SLAVE\_CON\_PARAMS\_UPD\_ENABLE (*CONFIG\_BT\_SMP\_SLAVE\_CON\_PARAMS\_UPD\_ENABLE*)
- CONFIG\_SPIRAM\_SUPPORT (*CONFIG\_ESP32\_SPIRAM\_SUPPORT*)
- **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\_STATIC\_ALLOCATION (*CONFIG\_FREERTOS\_SUPPORT\_STATIC\_ALLOCATION*)
- CONFIG\_SUPPORT\_TERMIOS (*CONFIG\_VFS\_SUPPORT\_TERMIOS*)
- CONFIG\_SUPPRESS\_SELECT\_DEBUG\_OUTPUT (*CONFIG\_VFS\_SUPPRESS\_SELECT\_DEBUG\_OUTPUT*)
- CONFIG\_SW\_COEXIST\_ENABLE (*CONFIG\_ESP32\_WIFI\_SW\_COEXIST\_ENABLE*)
- CONFIG\_SYSTEM\_EVENT\_QUEUE\_SIZE (*CONFIG\_ESP\_SYSTEM\_EVENT\_QUEUE\_SIZE*)
- CONFIG\_SYSTEM\_EVENT\_TASK\_STACK\_SIZE (*CONFIG\_ESP\_SYSTEM\_EVENT\_TASK\_STACK\_SIZE*)

- `CONFIG_TASK_WDT` ([CONFIG\\_ESP\\_TASK\\_WDT](#))
- `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_STACK_SIZE` ([CONFIG\\_LWIP\\_TCPIP\\_TASK\\_STACK\\_SIZE](#))
- `CONFIG_TCP_MAXRTX` ([CONFIG\\_LWIP\\_TCP\\_MAXRTX](#))
- `CONFIG_TCP_MSL` ([CONFIG\\_LWIP\\_TCP\\_MSL](#))
- `CONFIG_TCP_MSS` ([CONFIG\\_LWIP\\_TCP\\_MSS](#))
- **`CONFIG_TCP_OVERSIZE`** ([CONFIG\\_LWIP\\_TCP\\_OVERSIZE](#))
  - `CONFIG_TCP_OVERSIZE_MSS`
  - `CONFIG_TCP_OVERSIZE_QUARTER_MSS`
  - `CONFIG_TCP_OVERSIZE_DISABLE`
- `CONFIG_TCP_QUEUE_OOSEQ` ([CONFIG\\_LWIP\\_TCP\\_QUEUE\\_OOSEQ](#))
- `CONFIG_TCP_RECVMBOX_SIZE` ([CONFIG\\_LWIP\\_TCP\\_RECVMBOX\\_SIZE](#))
- `CONFIG_TCP_SND_BUF_DEFAULT` ([CONFIG\\_LWIP\\_TCP\\_SND\\_BUF\\_DEFAULT](#))
- `CONFIG_TCP_SYNMAXRTX` ([CONFIG\\_LWIP\\_TCP\\_SYNMAXRTX](#))
- `CONFIG_TCP_WND_DEFAULT` ([CONFIG\\_LWIP\\_TCP\\_WND\\_DEFAULT](#))
- `CONFIG_TIMER_QUEUE_LENGTH` ([CONFIG\\_FREERTOS\\_TIMER\\_QUEUE\\_LENGTH](#))
- `CONFIG_TIMER_TASK_PRIORITY` ([CONFIG\\_FREERTOS\\_TIMER\\_TASK\\_PRIORITY](#))
- `CONFIG_TIMER_TASK_STACK_DEPTH` ([CONFIG\\_FREERTOS\\_TIMER\\_TASK\\_STACK\\_DEPTH](#))
- `CONFIG_TIMER_TASK_STACK_SIZE` ([CONFIG\\_ESP\\_TIMER\\_TASK\\_STACK\\_SIZE](#))
- `CONFIG_TOOLPREFIX` ([CONFIG\\_SDK\\_TOOLPREFIX](#))
- `CONFIG_UDP_RECVMBOX_SIZE` ([CONFIG\\_LWIP\\_UDP\\_RECVMBOX\\_SIZE](#))
- `CONFIG_ULP_COPROC_ENABLED` ([CONFIG\\_ESP32\\_ULP\\_COPROC\\_ENABLED](#))
- `CONFIG_ULP_COPROC_RESERVE_MEM` ([CONFIG\\_ESP32\\_ULP\\_COPROC\\_RESERVE\\_MEM](#))
- `CONFIG_USE_ONLY_LWIP_SELECT` ([CONFIG\\_LWIP\\_USE\\_ONLY\\_LWIP\\_SELECT](#))
- `CONFIG_USE_TCPIP_STACK_LIB` ([CONFIG\\_NETIF\\_USE\\_TCPIP\\_STACK\\_LIB](#))
- `CONFIG_WARN_WRITE_STRINGS` ([CONFIG\\_COMPILER\\_WARN\\_WRITE\\_STRINGS](#))
- `CONFIG_WIFI_LWIP_ALLOCATION_FROM_SPIRAM_FIRST` ([CONFIG\\_SPIRAM\\_TRY\\_ALLOCATE\\_WIFI\\_LWIP](#))

## 2.8.7 Customisations

Because IDF builds by default with [警告未定义的变量](#), when the Kconfig tool generates Makefiles (the `auto.conf` file) its behaviour has been customised. In normal Kconfig, a variable which is set to “no” is undefined. In IDF’s version of Kconfig, this variable is defined in the Makefile but has an empty value.

(Note that `ifdef` and `ifndef` can still be used in Makefiles, because they test if a variable is defined *and has a non-empty value*.)

When generating header files for C & C++, the behaviour is not customised - so `ifdef` can be used to test if a boolean config item is set or not.

## 2.9 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](#).

[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\_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**): Id 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**): 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\_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\_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\_NVS (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\_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\_TCPIP\_ADAPTER\_BASE* (0x5000)

*ESP\_ERR\_TCPIP\_ADAPTER\_INVALID\_PARAMS* (0x5001)

*ESP\_ERR\_TCPIP\_ADAPTER\_IF\_NOT\_READY* (0x5002)

*ESP\_ERR\_TCPIP\_ADAPTER\_DHCP\_START\_FAILED* (0x5003)

*ESP\_ERR\_TCPIP\_ADAPTER\_DHCP\_ALREADY\_STARTED* (0x5004)

*ESP\_ERR\_TCPIP\_ADAPTER\_DHCP\_ALREADY\_STOPPED* (0x5005)

*ESP\_ERR\_TCPIP\_ADAPTER\_NO\_MEM* (0x5006)

*ESP\_ERR\_TCPIP\_ADAPTER\_DHCP\_NOT\_STOPPED* (0x5007)

*ESP\_ERR\_FLASH\_BASE* (0x6000): Starting number of flash error codes

*ESP\_ERR\_FLASH\_OP\_FAIL* (0x6001)

*ESP\_ERR\_FLASH\_OP\_TIMEOUT* (0x6002)

*ESP\_ERR\_FLASH\_NOT\_INITIALISED* (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\_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 socket option

*ESP\_ERR\_MBEDTLS\_CERT\_PARTLY\_OK* (0x8006): mbedtls parse certificates was partly successful

*ESP\_ERR\_MBEDTLS\_CTR\_DRBG\_SEED\_FAILED* (0x8007): mbedtls api returned error

*ESP\_ERR\_MBEDTLS\_SSL\_SET\_HOSTNAME\_FAILED* (0x8008): mbedtls api returned error

*ESP\_ERR\_MBEDTLS\_SSL\_CONFIG\_DEFAULTS\_FAILED (0x8009)*: mbedtls api returned error

*ESP\_ERR\_MBEDTLS\_SSL\_CONF\_ALPN\_PROTOCOLS\_FAILED (0x800a)*: mbedtls api returned error

*ESP\_ERR\_MBEDTLS\_X509\_CRT\_PARSE\_FAILED (0x800b)*: mbedtls api returned error

*ESP\_ERR\_MBEDTLS\_SSL\_CONF\_OWN\_CERT\_FAILED (0x800c)*: mbedtls api returned error

*ESP\_ERR\_MBEDTLS\_SSL\_SETUP\_FAILED (0x800d)*: mbedtls api returned error

*ESP\_ERR\_MBEDTLS\_SSL\_WRITE\_FAILED (0x800e)*: mbedtls api returned error

*ESP\_ERR\_MBEDTLS\_PK\_PARSE\_KEY\_FAILED (0x800f)*: mbedtls api returned failed

*ESP\_ERR\_MBEDTLS\_SSL\_HANDSHAKE\_FAILED (0x8010)*: mbedtls api returned failed

*ESP\_ERR\_MBEDTLS\_SSL\_CONF\_PSK\_FAILED (0x8011)*: mbedtls api returned failed

*ESP\_ERR\_ESP\_TLS\_CONNECTION\_TIMEOUT (0x8012)*: new connection in esp\_tls\_low\_level\_conn connection timed out

*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

## Chapter 3

# ESP32 H/W 硬件参考

### 3.1 ESP32 Modules and Boards

Espressif designs and manufactures different modules and development boards to help users evaluate the potential of the ESP32 family of chips.

This document provides description of modules and development boards currently available from Espressif.

---

**注解:** For description of previous versions of modules and development boards as well as for description of discontinued ones, please go to Section [Previous Versions of ESP32 Modules and Boards](#).

---

#### 3.1.1 WROOM, SOLO, WROVER, and PICO Modules

This is a family of ESP32-based modules with some integrated key components, including a crystal oscillator and an antenna matching circuit. The modules constitute ready-made solutions for integration into final products. If combined with a few extra components, such as a programming interface, bootstrapping resistors, and pin headers, these modules can also be used for evaluation of ESP32's functionality.

The key characteristics of these modules are summarized in the table below. Some additional details are covered in the following sections.

Module	Chip	Flash, MB	PSRAM, MB	Ant.	Dimensions, mm
ESP32-WROOM-32	ESP32-D0WDQ6	4	–	MIFA	18 × 25.5 × 3.1
ESP32-WROOM-32D	ESP32-D0WD	4, 8, or 16	–	MIFA	18 × 25.5 × 3.1
ESP32-WROOM-32U	ESP32-D0WD	4, 8, or 16	–	U.FL	18 × 19.2 × 3.1
ESP32-SOLO-1	ESP32-S0WD	4	–	MIFA	18 × 25.5 × 3.1
ESP32-WROVER (PCB)	ESP32-D0WDQ6	4	8	MIFA	18 × 31.4 × 3.3
ESP32-WROVER (IPEX)	ESP32-D0WDQ6	4	8	U.FL	18 × 31.4 × 3.3
ESP32-WROVER-B	ESP32-D0WD	4, 8, or 16	8	MIFA	18 × 31.4 × 3.3
ESP32-WROVER-IB	ESP32-D0WD	4, 8, or 16	8	U.FL	18 × 31.4 × 3.3

- ESP32-**D**.. identifies a dual-core chip, ESP32-**S**.. identifies a single-core chip
- MIFA - Meandered Inverted-F Antenna
- U.FL - U.FL / IPEX antenna connector
- ESP32-WROOM-32x, ESP32-WROVER-B and ESP32-WROVER-IB modules come with 4 MB flash by default but also available with custom flash sizes of 8 MB and 16 MB, see [Espressif Products Ordering Information](#) (PDF)
- [ESP32 Chip Datasheet](#) (PDF)
- Initial release of the ESP32-WROVER module had 4 MB of PSRAM

- *ESP32-WROOM-32* was previously called *ESP-WROOM-32*

### ESP32-WROOM-32

This is a basic and commonly adopted ESP32 module with the ESP32-D0WDQ6 chip on board. It was the first module of the WROOM / WROVER family released to the market.

For key characteristics, see the table in Section [WROOM, SOLO, WROVER, and PICO Modules](#), [Espressif Products Ordering Information](#).

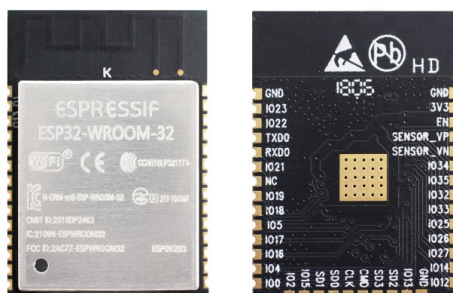


图 1: ESP32-WROOM-32 module (front and back)

### Documentation

- [ESP32-WROOM-32 Datasheet \(PDF\)](#)
- [ESP32-WROOM-32 Reference Design](#) containing OrCAD schematic, PCB layout, gerber and BOM files

### ESP32-WROOM-32D / ESP32-WROOM-32U

Both modules integrate the ESP32-D0WD chip which has a smaller footprint than the chip ESP32-D0WDQ6 installed in [ESP32-WROOM-32](#).

For key characteristics, see the table in Section [WROOM, SOLO, WROVER, and PICO Modules](#) and [Espressif Products Ordering Information](#).

ESP32-WROOM-32U is the smallest representative of the whole WROOM / WROVER family of modules.

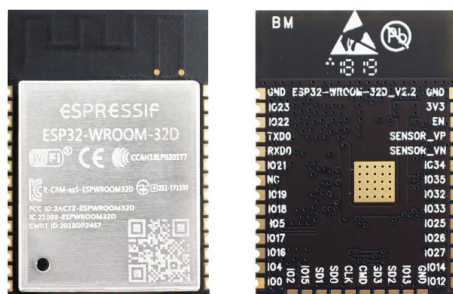


图 2: ESP32-WROOM-32D module (front and back)

### Documentation

- [ESP32-WROOM-32D / ESP32-WROOM-32U Datasheet \(PDF\)](#)

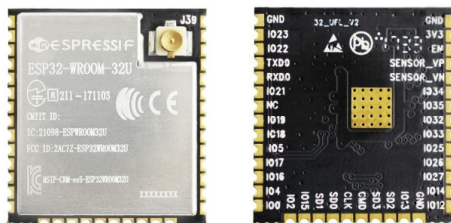


图 3: ESP32-WROOM-32U module (front and back)

### ESP32-SOLO-1

This is a simplified version of the ESP32-WROOM-32D module. It contains a single-core ESP32 chip that supports a clock frequency of up to 160 MHz.

For key characteristics, see the table in Section [WROOM, SOLO, WROVER, and PICO Modules](#) and [Espressif Products Ordering Information](#).

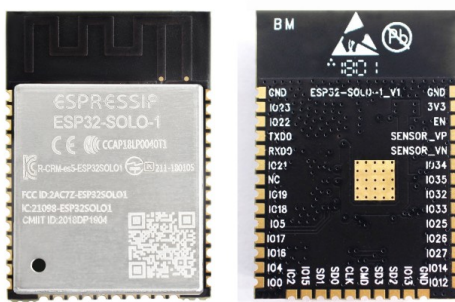


图 4: ESP32-SOLO-1 module (front and back)

### Documentation

- [ESP32-SOLO-1 Datasheet \(PDF\)](#)

### ESP32-WROVER series

This series consists of a few modifications of ESP32-WROOM-32x modules, which among other upgrades include additional 8 MB SPI PSRAM (pseudo static RAM).

For details, see the table in Section [WROOM, SOLO, WROVER, and PICO Modules](#) and [Espressif Products Ordering Information](#).

- **ESP32-WROVER (PCB)** and **ESP32-WROVER (IPEX)** have PSRAM that operates at 1.8 V and supports up to 144 MHz clock rate.
- **ESP32-WROVER-B** and **ESP32-WROVER-IB** have PSRAM that operates at 3.3 V and can supports up to 133 MHz clock rate.

The picture below shows an ESP32-WROVER module with a PCB antenna.

### Documentation

- [ESP32-WROVER Datasheet \(PDF\)](#)
- [ESP32-WROVER-B Datasheet \(PDF\)](#)



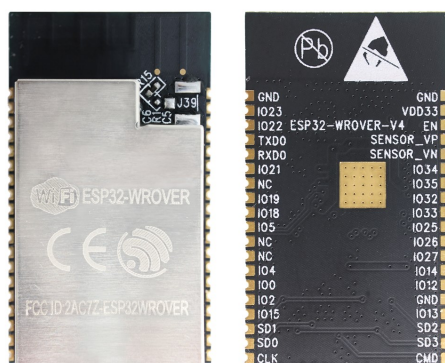


图 5: ESP32-WROVER module (front and back)

- [ESP-PSRAM64 & ESP-PSRAM64H Datasheet \(PDF\)](#)
- [ESP32-WROVER Reference Design](#) containing OrCAD schematic, PCB layout, gerber and BOM files

### ESP32-PICO-D4

ESP32-PICO-D4 is a System-in-Package (SiP) module, integrating all peripheral components seamlessly, including the following:

- 4 MB flash memory
- crystal oscillator
- filter capacitors
- RF matching circuit

For key characteristics, see [Espressif Products Ordering Information](#).

### Documentation

- [ESP32-PICO-D4 Datasheet \(PDF\)](#)

### 3.1.2 Development Boards

Depending on the intended functionality, different development boards feature:

- Access to different ESP32 GPIO pins.
- Different interfaces: USB, JTAG.
- Different peripherals: touchpads, LCD screens, SD card slots, headers for camera modules, etc.

#### ESP32-PICO-KIT V4.1

This is the smallest available ESP32-based development board. It features all the components for direct connection to a computer's USB port as well as pin headers for plugging into a mini breadboard.

The board is equipped with the [ESP32-PICO-D4](#) module. With such a module, the creation of a fully functional development board required only a few external components that fit on a PCB as small as 20 x 52 mm. The external components include antenna, LDO, USB-UART bridge, and two buttons for reset and activation of Firmware Download mode.

Comparing to ESP32-PICO-KIT V4, this version features the CP2102N USB-UART bridge that provides faster transfer rates of up to 3 Mbps.

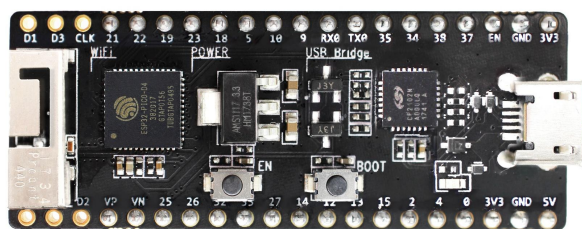


图 6: ESP32-PICO-KIT V4.1 board

### Documentation

- [ESP32-PICO-KIT V4/V4.1 入门指南](#)
- [ESP32-PICO-KIT V4.1 Schematic \(PDF\)](#)
- [ESP32-PICO-KIT Reference Design](#) containing OrCAD schematic, PCB layout, gerber and BOM files
- [ESP32-PICO-D4 Datasheet \(PDF\)](#)

### Previous Versions

- [ESP32-PICO-KIT V4](#)
- [ESP32-PICO-KIT V3](#)

### ESP32 DevKitC V4

This is a small and convenient development board that features:

- [ESP32-WROOM-32](#) module
- USB-to-serial programming interface that also provides power supply for the board
- pin headers
- pushbuttons for reset and activation of Firmware Download mode
- a few other components

Comparing to the previous [ESP32 Core Board V2 / ESP32 DevKitC](#), this version can integrate [ESP32-WROVER series](#) module instead of ESP32-WROOM-32 and has the CP2102N chip that supports faster baud rates.

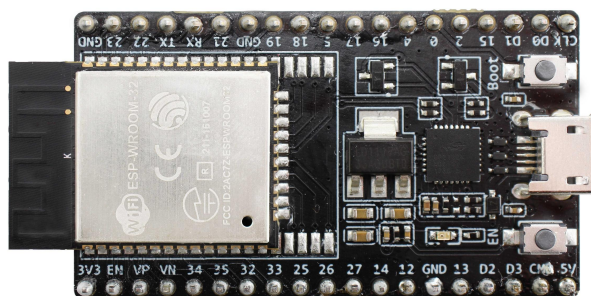


图 7: ESP32 DevKitC V4 board

### Documentation

- [ESP32-DevKitC V4 入门指南](#)
- [ESP32-DevKitC schematic \(PDF\)](#)
- [ESP32-DevKitC Reference Design](#) containing OrCAD schematic, PCB layout, gerber and BOM files
- [CP210x USB to UART Bridge VCP Drivers](#)

### Previous Versions

- [ESP32 Core Board V2 / ESP32 DevKitC](#)

## ESP-WROVER-KIT V4.1

This board features:

- dual port USB-to-serial converter for programming
- JTAG interface for debugging
- MicroSD card slot
- 3.2" SPI LCD screen
- header for a camera module
- RGB diode for diagnostics
- 32.768 kHz XTAL for internal RTC to operate it in low power modes

Power can be supplied either via USB or via a standard 5 mm power supply jack. A power source can be selected with a jumper and can be turned on/off with a separate switch.

This version of the ESP-WROVER-KIT board integrates the ESP-WROVER-B module that has 8 MB PSRAM for flexible extended storage and data processing capabilities. The board can accommodate other versions of ESP modules described in [WROOM](#), [SOLO](#), [WROVER](#), and [PICO Modules](#).

Comparing to [ESP-WROVER-KIT V3](#), this board has the following design changes:

- JP8, JP11, and JP13 have been combined into a single JP2.
- USB connector has been changed to DIP type and moved to the lower right corner of the board.
- R61 has been changed to a Zero-ohm resistor.
- Some components have been replaced with functional equivalents based on test results and sourcing options, e.g., the EN and Boot buttons.

The board in the picture above integrates the ESP32-WROVER-B module.

### Documentation

- [ESP-WROVER-KIT V4.1 入门指南](#)
- [ESP-WROVER-KIT V4.1 Schematic \(PDF\)](#)
- [JTAG 调试](#)
- [FTDI Virtual COM Port Drivers](#)

### Previous Versions

- [ESP-WROVER-KIT V3](#)
- [ESP-WROVER-KIT V2](#)
- [ESP-WROVER-KIT V1 / ESP32 DevKitJ V1](#)

### 3.1.3 Related Documents

- [Previous Versions of ESP32 Modules and Boards](#)

## 3.2 Previous Versions of ESP32 Modules and Boards

This sections contains overview and links to documentation of previous version ESP32 Modules and Boards that have been replaced with newer versions or discontinued. It is maintained for convenience of users as several of these boards are still in use and some may still be available for purchase.

To see the latest development boards, please refer to section [ESP32 Modules and Boards](#).

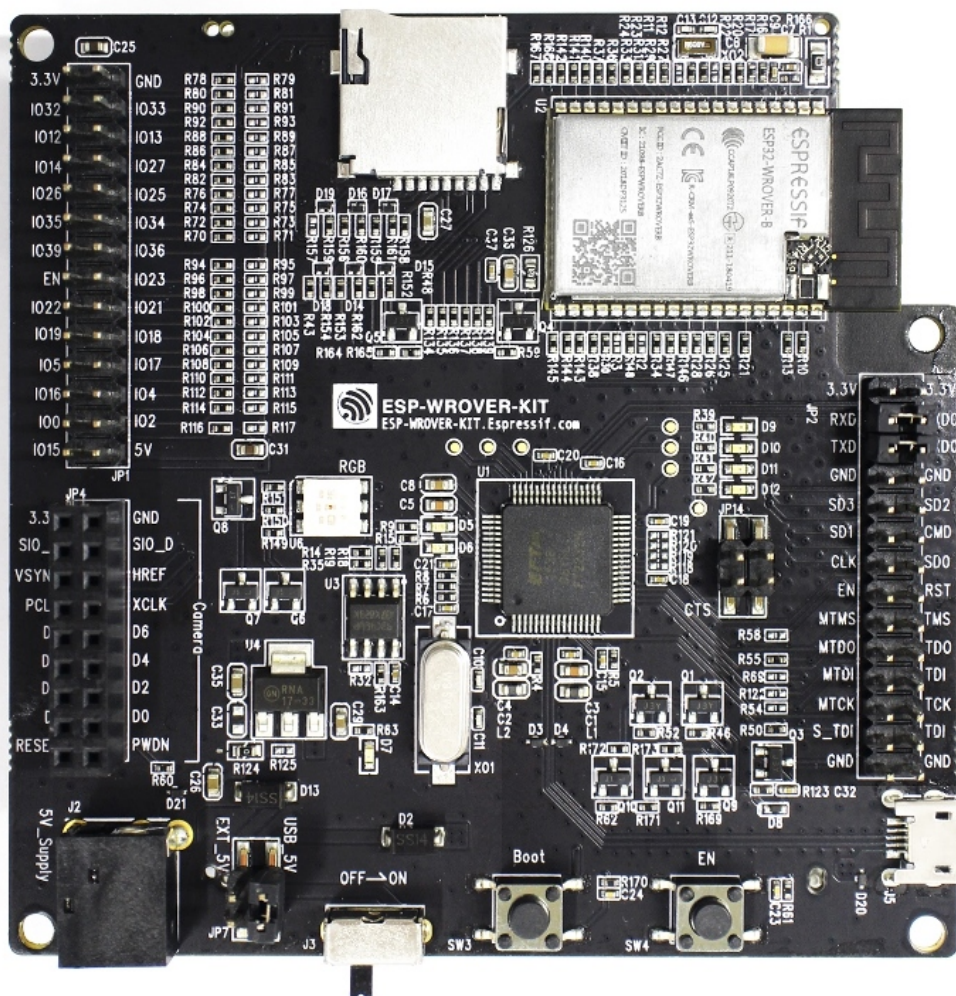


图 8: ESP-WROVER-KIT V4.1 board

### 3.2.1 ESP32-PICO-KIT V4

The smallest ESP32 development board with all the components required to connect it directly to a PC USB port, and pin headers to plug into a mini breadboard. It is equipped with ESP32-PICO-D4 module that integrates 4 MB flash memory, a crystal oscillator, filter capacitors and RF matching circuit in one single package. As result, the fully functional development board requires only a few external components that can easy fit on a 20 x 52 mm PCB including antenna, LDO, USB-UART bridge and two buttons to reset it and put into download mode.

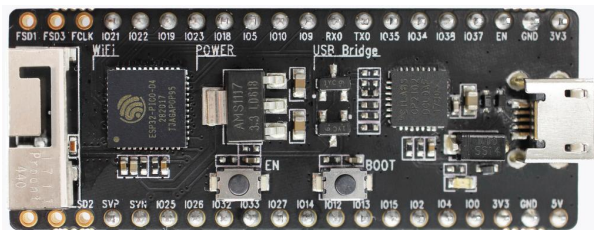


图 9: ESP32-PICO-KIT V4 board

Comparing to ESP32-PICO-KIT V3, this version has revised printout and reduced number of exposed pins. Instead of 20, only 17 header pins are populated, so V4 can fit into a mini breadboard.

#### Documentation

- [ESP32-PICO-KIT V4/V4.1 入门指南](#)
- [ESP32-PICO-KIT V4 Schematic \(PDF\)](#)
- [ESP32-PICO-D4 Datasheet \(PDF\)](#)

### 3.2.2 ESP32-PICO-KIT V3

The first public release of Espressif' s ESP32-PICO-D4 module on a mini development board. The board has a USB port for programming and debugging and two rows of 20 pin headers to plug into a breadboard. The ESP32-PICO-D4 module itself is small and requires only a few external components. Besides two core CPUs it integrates 4MB flash memory, a crystal oscillator and antenna matching components in one single 7 x 7 mm package. As a result the module and all the components making the complete development board fit into 20 x 52 mm PCB.

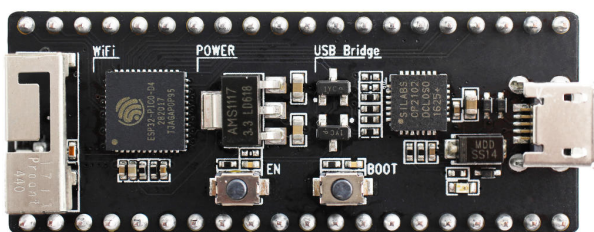


图 10: ESP32-PICO-KIT V3 board

#### Documentation

- [ESP32-PICO-KIT V3 入门指南](#)
- [ESP32-PICO-KIT V3 Schematic \(PDF\)](#)
- [ESP32-PICO-D4 Datasheet \(PDF\)](#)

### 3.2.3 ESP32 Core Board V2 / ESP32 DevKitC

Small and convenient development board with ESP-WROOM-32 module installed, break out pin headers and minimum additional components. Includes USB to serial programming interface, that also provides power supply for the board. Has pushbuttons to reset the board and put it in upload mode.

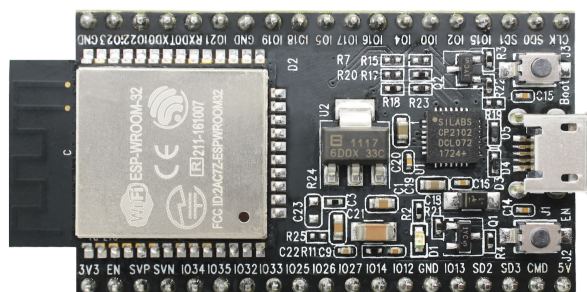


图 11: ESP32 Core Board V2 / ESP32 DevKitC board

#### Documentation

- [ESP32-DevKitC V2 入门指南](#)
- [ESP32 DevKitC V2 Schematic \(PDF\)](#)
- [CP210x USB to UART Bridge VCP Drivers](#)

### 3.2.4 ESP-WROVER-KIT V3

The ESP-WROVER-KIT V3 development board has dual port USB to serial converter for programming and JTAG interface for debugging. Power supply is provided by USB interface or from standard 5 mm power supply jack. Power supply selection is done with a jumper and may be put on/off with a separate switch. This board has MicroSD card slot, 3.2" SPI LCD screen and dedicated header to connect a camera. It provides RGB diode for diagnostics. Includes 32.768 kHz XTAL for internal RTC to operate it in low power modes.

As all previous versions of ESP-WROVER-KIT boards, it is ready to accommodate an [ESP32-WROOM-32](#) or [ESP32-WROVER series](#) module.

This is the first release of ESP-WROVER-KIT shipped with [ESP32-WROVER series](#) module installed by default. This release also introduced several design changes to conditioning and interlocking of signals to the bootstrapping pins. Also, a zero Ohm resistor (R166) has been added between WROVER/WROOM module and VDD33 net, which can be desoldered, or replaced with a shunt resistor, for current measurement. This is intended to facilitate power consumption analysis in various operation modes of ESP32. Refer to schematic - the changes are enclosed in green border.

The camera header has been changed from male back to female. The board soldermask is matte black. The board on picture above has [ESP32-WROVER series](#) is installed.

#### Documentation

- [ESP-WROVER-KIT V3 入门指南](#)
- [ESP-WROVER-KIT V3 Schematic \(PDF\)](#)
- [JTAG 调试](#)
- [FTDI Virtual COM Port Drivers](#)

### 3.2.5 ESP-WROVER-KIT V2

This is updated version of ESP32 DevKitJ V1 described above with design improvements identified when DevKitJ was in use, e.g. improved support for SD card. By default board has ESP-WROOM-32 module installed.

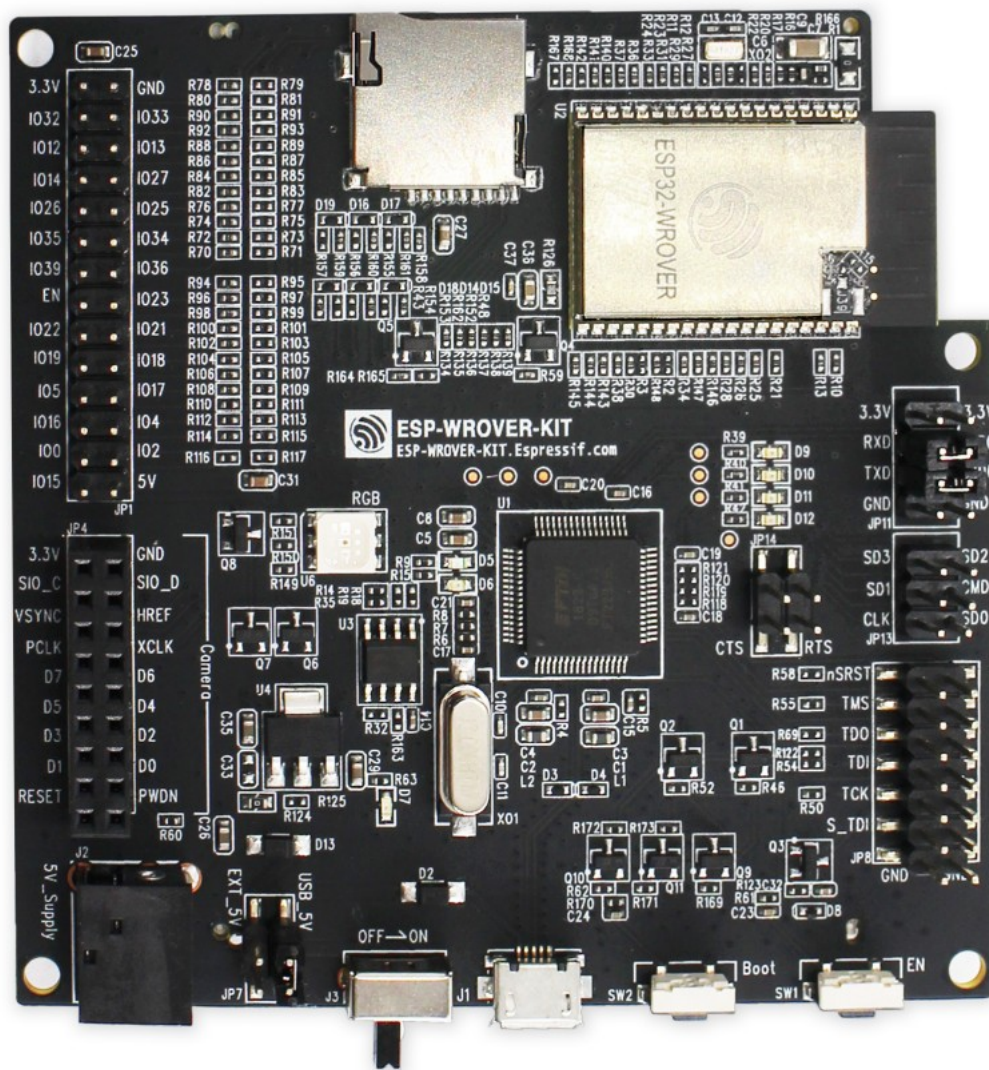


图 12: ESP-WROVER-KIT V3 board

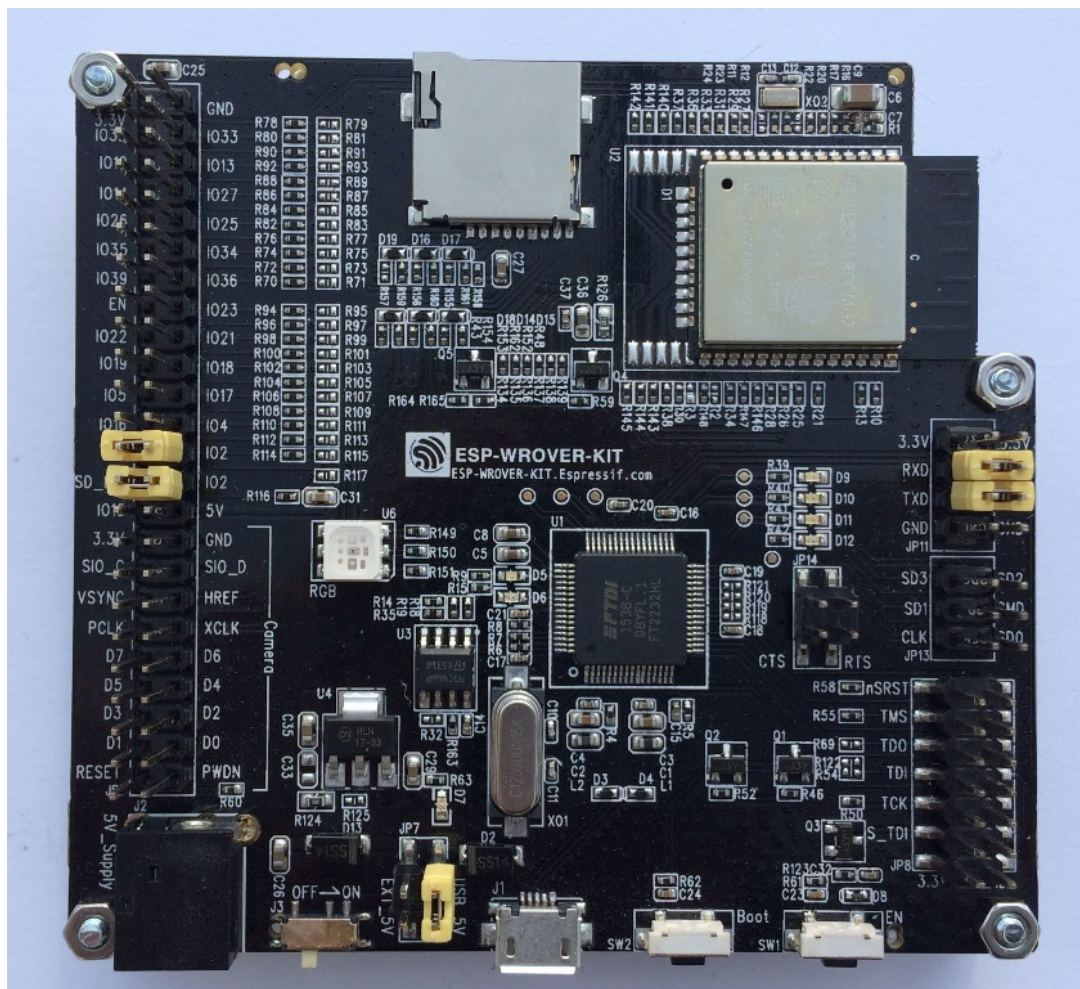


图 13: ESP-WROVER-KIT V2 board



Comparing to previous version, this board has a shiny black finish and a male camera header.

## Documentation

- [ESP-WROVER-KIT V2 入门指南](#)
- [ESP-WROVER-KIT V2 Schematic \(PDF\)](#)
- [JTAG 调试](#)
- [FTDI Virtual COM Port Drivers](#)

### 3.2.6 ESP-WROVER-KIT V1 / ESP32 DevKitJ V1

The first version of ESP-WROVER-KIT development board. Shipped with ESP-WROOM-32 on board.

ESP-WROVER-KIT has dual port USB to serial converter for programming and JTAG interface for debugging. Power supply is provided by USB interface or from standard 5 mm power supply jack. Power supply selection is done with a jumper and may be put on/off with a separate switch. The board has MicroSD card slot, 3.2" SPI LCD screen and dedicated header to connect a camera. It provides RGB diode for diagnostics. Includes 32.768 kHz XTAL for internal RTC to operate it in low power modes.

All versions of ESP-WROVER-KIT are ready to accommodate an ESP-WROOM-32 or ESP32-WROVER module.

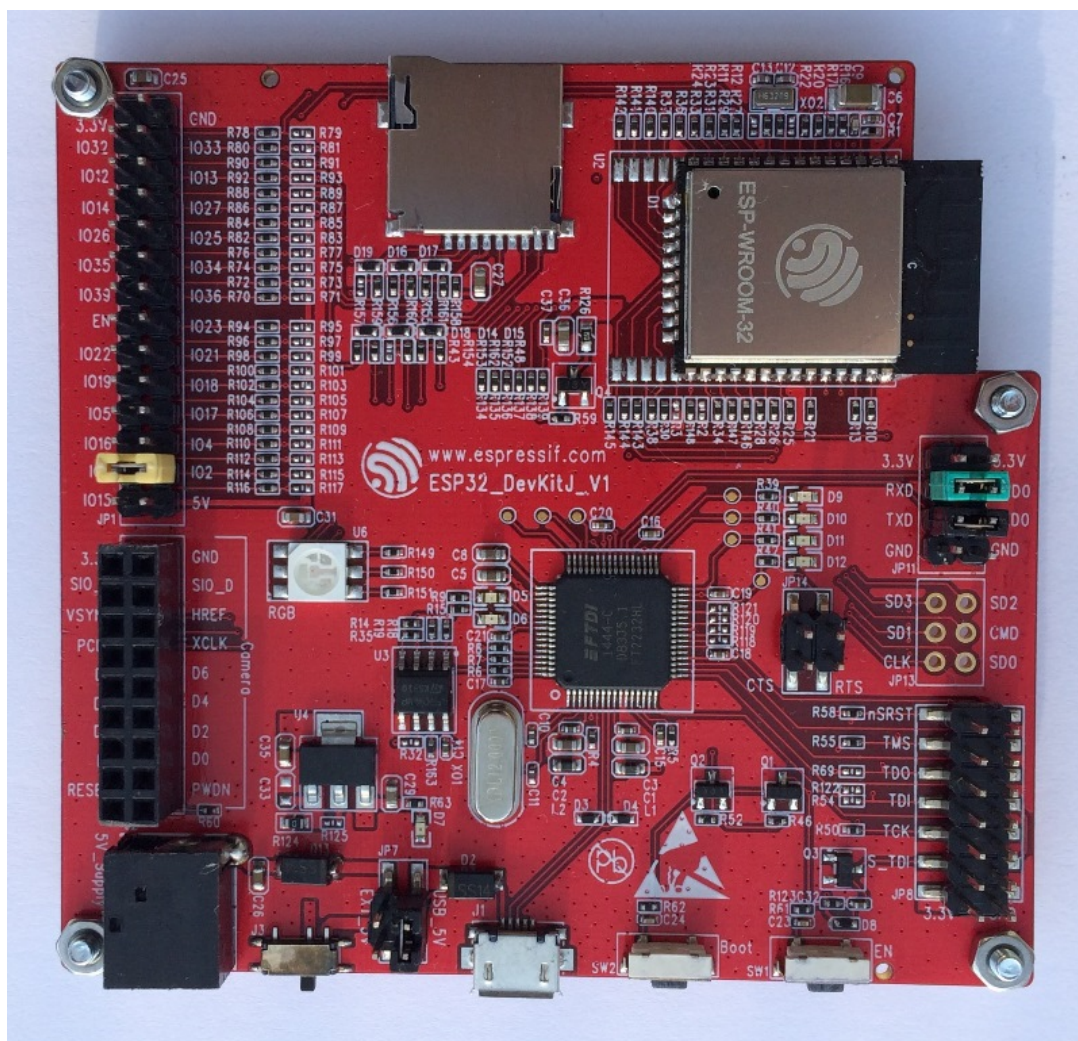


图 14: ESP-WROVER-KIT V1 / ESP32 DevKitJ V1 board

The board has red soldermask.

## Documentation

- [ESP-WROVER-KIT V1 Schematic \(PDF\)](#)
- [JTAG 调试](#)
- [FTDI Virtual COM Port Drivers](#)

### 3.2.7 ESP32 Demo Board V2

One of first feature rich evaluation boards that contains several pin headers, dip switches, USB to serial programming interface, reset and boot mode press buttons, power switch, 10 touch pads and separate header to connect LCD screen.

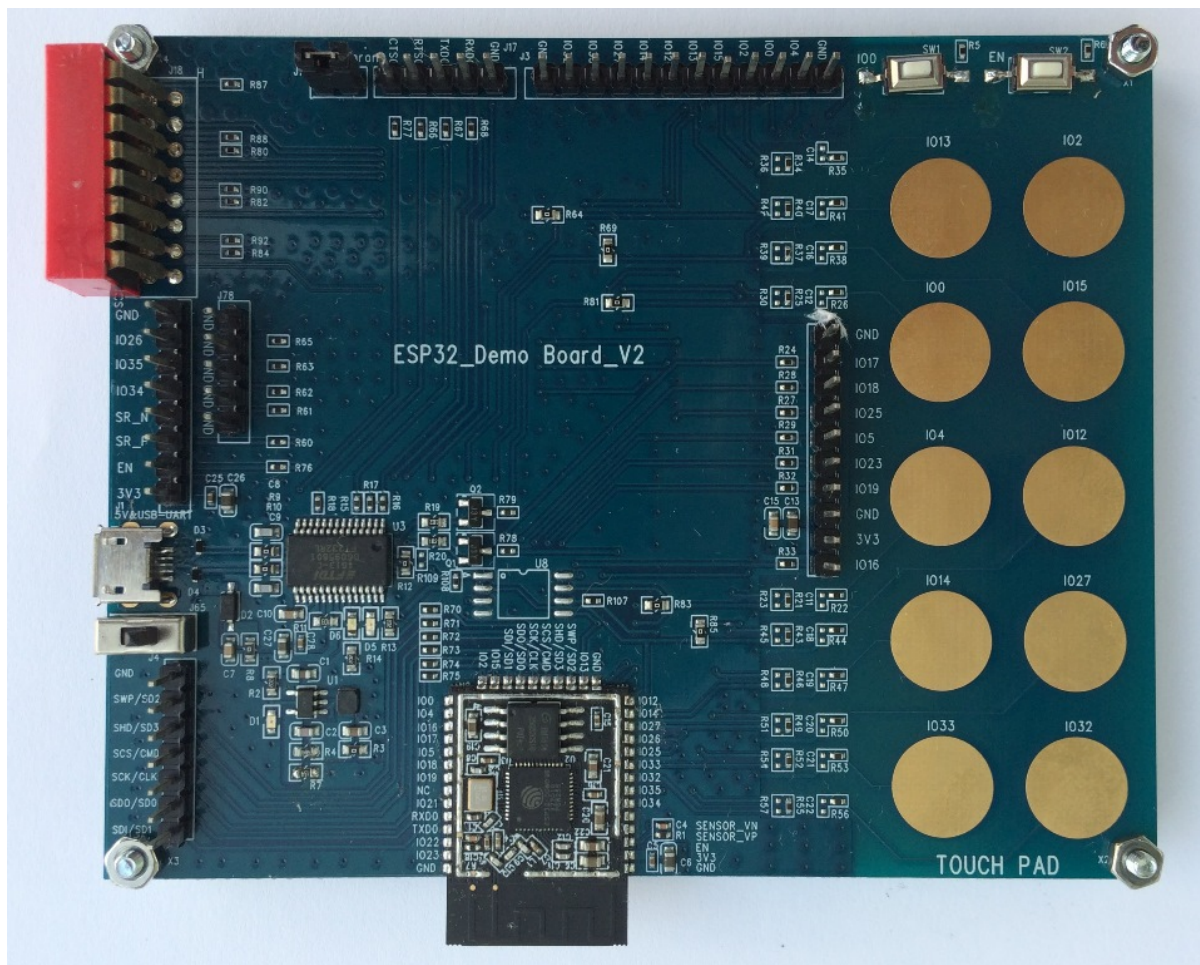


图 15: ESP32 Demo Board V2

Production of this board is discontinued.

## Documentation

- [ESP32 Demo Board V2 Schematic \(PDF\)](#)
- [FTDI Virtual COM Port Drivers](#)

### 3.2.8 Related Documents

- [ESP32 Modules and Boards](#)



## Chapter 4

# API 指南

### 4.1 ESP-IDF 编程注意事项

#### 4.1.1 应用程序的启动流程

本文将介绍 ESP32 从上电到运行 `app_main` 函数中间所经历的步骤（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. 主程序运行，这时第二个 CPU 和 RTOS 的调度器可以开始运行。

下面会对上述过程进行更为详细的阐述。

#### 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 芯片掩膜 ROM 的 `0x40000400` 地址处，该地址不能被修改。

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 的工作模式，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，程序会执行不同的操作：

1. 从深度睡眠模式复位：如果 `RTC_CNTL_STORE6_REG` 寄存器的值非零，并且 `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，或者 `RTC_CNTL_STORE7_REG` 中的 CRC 校验值无效，又或者跳转到 `RTC_CNTL_STORE6_REG` 地址处运行的程序返回，那么将会执行上电复位的相关操作。**注意：**如果想在这里运行自定义的代码，可以参考[深度睡眠](#)文档里面介绍的方法。
2. 上电复位、软件 SoC 复位、看门狗 SoC 复位：检查 `GPIO_STRAP_REG` 寄存器，判断是否 UART 或 SDIO 请求进入下载模式。如果是，则配置好 UART 或者 SDIO，然后等待下载代码。否则程序将会执行软件 CPU 复位的相关操作。
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，这部分的内存将会在后面一小节详细介绍。如果从 Flash 中加载代码失败，就会将 BASIC 解析器加压缩到 RAM 中启动。需要注意的是，此时 RTC 看门狗还在使能状态，如果在几百毫秒内没有任何输入事件，那么看门狗会再次复位 SoC，重复整个过程。如果解析器收到了来自 UART 的输入，程序会关闭看门狗。

应用程序的二进制镜像会从 Flash 的 `0x1000` 地址处加载。Flash 的第一个 4kB 扇区用于存储安全引导程序和应用程序镜像的签名。有关详细信息，请查看[安全启动文档](#)。

## 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。请注意，对于 ESP32 芯片来说，这并不是唯一的安排程序镜像的方式。事实上用户完全可以把一个功能齐全的应用程序烧写到 Flash 的 0x1000 偏移地址处运行，但这超出本文档的范围。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。

二级引导程序从 Flash 的 0x8000 偏移地址处读取分区表。详细信息请参阅分区表文档[分区表](#)。二级引导程序会寻找出厂分区和 OTA 分区，然后根据 OTA 信息分区的数据决定引导哪个分区。

对于选定的分区，二级引导程序将映射到 IRAM 和 DRAM 的数据和代码段复制到它们的加载地址处。对于一些加载地址位于 DROM 和 IROM 区域的段，会通过配置 Flash MMU 为其提供正确的映射。请注意，二级引导程序会为 PRO CPU 和 APP CPU 都配置 Flash MMU，但它只使能了 PRO CPU 的 Flash MMU。这么做的原因在于二级引导程序的代码被加载到了 APP CPU 的高速缓存使用的内存区域，因此使能 APP CPU 高速缓存的任务就交给了应用程序。一旦代码加载完毕并且设置好 Flash MMU，二级引导程序会从应用程序二进制镜像文件的头部寻找入口地址，然后跳转到该地址处运行。

目前还不支持添加钩子函数到二级引导程序中以自定义应用程序分区选择的逻辑，但是可以通过别的途径实现这个需求，比如根据某个 GPIO 的不同状态来引导不同的应用程序镜像。此类自定义的功能将在未来添加到 ESP-IDF 中。目前，可以通过将 `bootloader` 组件复制到应用程序目录并在那里进行必要的更改来自定义引导程序。在这种情况下，ESP-IDF 的编译系统将编译应用程序目录中的组件而不是 ESP-IDF 组件目录。

## 应用程序启动阶段

ESP-IDF 应用程序的入口是 `components/esp32/cpu_start.c` 文件中的 `call_start_cpu0` 函数，该函数主要完成了两件事，一是启用堆分配器，二是使 APP CPU 跳转到其入口点——`call_start_cpu1` 函数。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，PRO CPU 跳转到 `start_cpu0` 函数，APP CPU 跳转到 `start_cpu1` 函数。

`start_cpu0` 和 `start_cpu1` 这两个函数都是弱类型的，这意味着如果某些特定的应用程序需要修改初始化顺序，就可以通过重写这两个函数来实现。`start_cpu0` 默认的实现方式是初始化用户在 `menuconfig` 中选择的组件，具体实现步骤可以阅读 `components/esp32/cpu_start.c` 文件中的源码。请注意，此阶段会调用应用程序中存在的 C++ 全局构造函数。一旦所有必要的组件都初始化好，就会创建 `main task`，并启动 FreeRTOS 的调度器。

当 PRO CPU 在 `start_cpu0` 函数中进行初始化的时候，APP CPU 在 `start_cpu1` 函数中自旋，等待 PRO CPU 上的调度器启动。一旦 PRO CPU 上的调度器启动后，APP CPU 上的代码也会启动调度器。

主任务是指运行 `app_main` 函数的任务，主任务的堆栈大小和优先级可以在 `menuconfig` 中进行配置。应用程序可以用此任务来完成用户程序相关的初始化设置，比如启动其他的任务。应用程序还可以将主任务用于事件循环和其他通用活动。如果 `app_main` 函数返回，那么主任务将会被删除。

### 4.1.2 应用程序的内存布局

ESP32 芯片具有灵活的内存映射功能，本小节将介绍 ESP-IDF 默认使用这些功能的方式。

ESP-IDF 应用程序的代码可以放在以下内存区域之一。

#### IRAM (指令 RAM)

ESP-IDF 将内部 SRAM0 区域（在技术参考手册中有定义）的一部分分配为指令 RAM。除了开始的 64kB 用作 PRO CPU 和 APP CPU 的高速缓存外，剩余内存区域（从 0x40080000 至 0x400A0000）被用来存储应用程序中部分需要在 RAM 中运行的代码。

一些 ESP-IDF 的组件和 WiFi 协议栈的部分代码通过链接脚本文件被存放到了这块内存区域。

如果一些应用程序的代码需要放在 IRAM 中运行，可以使用 IRAM\_ATTR 宏定义进行声明。

```
#include "esp_attr.h"

void IRAM_ATTR gpio_isr_handler(void* arg)
{
    // ...
}
```

下面列举了应用程序中可能或者应该放入 IRAM 中运行例子。

- 当注册中断处理程序的时候设置了 ESP\_INTR\_FLAG\_IRAM，那么中断处理程序就必须放在 IRAM 中运行。这种情况下，ISR 只能调用存放在 IRAM 或者 ROM 中的函数。注意：目前所有 FreeRTOS 的 API 都已经存放到了 IRAM 中，所以在中断中调用 FreeRTOS 的中断专属 API 是安全的。如果将 ISR 放在 IRAM 中运行，那么必须使用宏定义 DRAM\_ATTR 将该 ISR 用到所有常量数据和调用的函数（包括但不限于 const char 数组）放入 DRAM 中。
- 可以将一些时间关键的代码放在 IRAM 中，这样可以缩减从 Flash 加载代码所消耗的时间。ESP32 是通过 32kB 的高速缓存来从外部 Flash 中读取代码和数据的，将函数放在 IRAM 中运行可以减少由高速缓存未命中引起的时间延迟。

### IRAM (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 内存中，则将其置于 Flash 中。Flash 技术参考手册中介绍了 Flash MMU 允许代码从 Flash 执行的机制。ESP-IDF 将从 Flash 中执行的代码放在 0x400D0000—0x40400000 区域的开始，在启动阶段，二级引导程序会初始化 Flash MMU，将代码在 Flash 中的位置映射到这个区域的开头。对这个区域的访问会被透明地缓存到 0x40070000—0x40080000 范围内的两个 32kB 的块中。

请注意，使用 Window ABI CALLx 指令可能无法访问 0x40000000—0x40400000 区域以外的代码，所以要特别留意应用程序是否使用了 0x40400000—0x40800000 或者 0x40800000—0x40C00000 区域，ESP-IDF 默认不会使用这两个区域。

### RTC 快速内存

从深度睡眠模式唤醒后必须要运行的代码要放在 RTC 内存中，更多信息请查阅文档[深度睡眠](#)。

### DRAM (数据 RAM)

链接器将非常量静态数据和零初始化数据放入 0x3FFB0000—0x3FFF0000 这 256kB 的区域。注意，如果使用蓝牙堆栈，此区域会减少 64kB（通过将起始地址移至 0x3FFC0000）。如果使用了内存跟踪的功能，该区域的长度还要减少 16kB 或者 32kB。放置静态数据后，留在此区域中的剩余空间都用作运行时堆。

常量数据也可以放在 DRAM 中，例如，用在 ISR 中的常量数据（参见上面 IRAM 部分的介绍），为此需要使用 DRAM\_ATTR 宏来声明。

```
DRAM_ATTR const char[] format_string = "%p %x";
char buffer[64];
sprintf(buffer, format_string, ptr, val);
```

毋庸置疑，不建议在 ISR 中使用 printf 和其余输出函数。出于调试的目的，可以在 ISR 中使用 ESP\_EARLY\_LOGx 来输出日志，不过要确保将 TAG 和格式字符串都放在了 DRAM 中。

宏 \_\_NOINIT\_ATTR 可以用来声明将数据放在 .noinit 段中，放在此段中的数据不会在启动时被初始化，并且在软件重启后会保留原来的值。

例子：

```
__NOINIT_ATTR uint32_t noinit_data;
```

### 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 4MB 区域 (0x3F400000 — 0x3F800000)，该区域用于通过 Flash MMU 和高速缓存来访问外部 Flash。一种特殊情况是，字面量会被编译器嵌入到应用程序代码中。

### RTC 慢速内存

从 RTC 内存运行的代码（例如深度睡眠模块的代码）使用的全局和静态变量必须要放在 RTC 慢速内存中。更多详细说明请查看文档[深度睡眠](#)。

宏 RTC\_NOINIT\_ATTR 用来声明将数据放入 RTC 慢速内存中，该数据在深度睡眠唤醒后将保持不变。

例子：

```
RTC_NOINIT_ATTR uint32_t rtc_noinit_data;
```

### 4.1.3 DMA 能力要求

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中而不是堆栈中。使用 DMA\_ATTR 宏可以声明该全局/本地的静态变量具备 DMA 能力，例如：

```
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 );
    // 其他程序
}
```

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 );
    // 其他程序
}
```

在堆栈中放置 DMA 缓冲区仍然是允许的，但是你必须记住：

1. 如果堆栈在 pSRAM 中，切勿尝试这么做，因为堆栈在 pSRAM 中的话就要按照[片外 SRAM](#) 文档介绍的步骤来操作（至少要在 menuconfig 中使能 SPIRAM\_ALLOW\_STACK\_EXTERNAL\_MEMORY），所以请确保你的任务不在 pSRAM 中。
2. 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：

```

void app_main()
{
    uint8_t stuff;
    WORD_ALIGNED_ATTR uint8_t buffer[]="I want to send something";    //否则
    buffer 数组会被存储在 stuff 变量的后面
    // 初始化代码...
    spi_transaction_t temp = {
        .tx_buffer = buffer,
        .length = 8*sizeof(buffer),
    };
    spi_device_transmit( spi, &temp );
    // 其他程序
}

```

## 4.2 构建系统 (CMake 版)

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。如需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。

---

**注解：**本文档将介绍基于 CMake 的构建系统，它是 ESP-IDF V4.0 及以上版本的默认系统。此外 ESP-IDF 还支持基于 *GNU Make* 的构建系统，基于 GNU Make 的构建系统是 ESP-IDF V4.0 以下版本的默认系统。

---

### 4.2.1 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：

- ESP32 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。
- 应用程序是由 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。ESP-IDF 当前仅支持 ESP32 这一个硬件目标。

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。

## 4.2.2 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**，配置待构建的系统
- 命令行构建工具 (**Ninja** 或 **GNU Make**)
- **esptool.py**，烧录 ESP32

**入门指南** 简要介绍了如何设置 idf.py 用于配置、构建并烧录项目。

idf.py 应运行在 ESP-IDF 的项目目录下，即包含 CMakeLists.txt 文件的目录。仅包含 Makefile 的老式项目并不支持 idf.py。

运行 idf.py --help 查看完整的命令列表。下面总结了最常用的命令：

- idf.py menuconfig 会运行 menuconfig 工具来配置项目。
- idf.py build 会构建在当前目录下找到的项目，它包括以下步骤：
  - 根据需要创建 build 构建目录，它用于保存构建过程的输出文件，可以使用 -B 选项修改默认的构建目录。
  - 根据需要运行 **CMake** 配置命令，为主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，构建工具会被自动检测，可以使用 -G 选项显式地指定构建工具。

构建过程是增量式的，如果自上次构建以来源文件或项目配置没有发生改变，则不会执行任何操作。

- idf.py clean 会把构建输出的文件从构建目录中删除，从而清理整个项目。下次构建时会强制“重新完整构建”这个项目。清理时，不会删除 **CMake** 配置输出及其他文件。
- idf.py fullclean 会将整个 build 目录下的内容全部删除，包括所有 **CMake** 的配置输出文件。下次构建项目时，**CMake** 会从头开始配置项目。请注意，该命令会递归删除构建目录下的所有文件，请谨慎使用。项目配置文件不会被删除。
- idf.py flash 会在必要时自动构建项目，并将生成的二进制程序烧录进 ESP32 设备中。-p 和 -b 选项可分别设置串口的设备名和烧录时的波特率。
- idf.py monitor 用于显示 ESP32 设备的串口输出。-p 选项可用于设置主机端串口的设备名，按下 Ctrl-J 可退出监视器。更多有关监视器的详情，请参阅 [IDF 监视器](#)。

多个 idf.py 命令可合并成一个，例如，idf.py -p COM4 clean flash monitor 会依次清理源码树，构建项目，烧录进 ESP32 设备，最后运行串口监视器。

---

**注解：** 环境变量 ESPPORT 和 ESPBAUD 可分别用作 -p 和 -b 选项的默认值。在命令行中，重新为这两个选项赋值，会覆盖其默认值。

---

### 高级命令

- idf.py app, idf.py bootloader, idf.py partition\_table 仅可用于从适用的项目中构建应用程序、引导程序或分区表。
- idf.py app-flash 等匹配命令，仅将项目的特定部分烧录至 ESP32。
- idf.py -p PORT erase\_flash 会使用 **esptool.py** 擦除 ESP32 的整个 Flash。
- idf.py size 会打印应用程序相关的大小信息，idf.py size-components 和 idf.py size-files 这两个命令相似，分别用于打印每个组件或源文件的详细信息。如果您在运行 **CMake** (或 idf.py) 时定义了变量 -DOUTPUT\_JSON=1，那么输出的格式会变成 JSON 而不是可读文本。
- idf.py reconfigure 命令会重新运行 **CMake** (即便无需重新运行)。正常使用时，并不需要运行此命令，但当源码树中添加/删除文件或更改 **CMake cache** 变量时，此命令会非常有用，例如，idf.py -DNAME='VALUE' reconfigure 会将 **CMake cache** 中的变量 NAME 的值设置为 VALUE。

- `idf.py python-clean` 会从 IDF 目录中删除生成的 Python 字节码，Python 字节码可能会在切换 IDF 和 Python 版本时引发问题，因此建议在切换 Python 后运行该命令。

同时调用多个 `idf.py` 命令时，命令的输入顺序并不重要，它们会按照正确的顺序依次执行，并保证每一条命令都生效（即先构建后烧录，先擦除后烧录等）。

**idf.py 选项** 运行 `idf.py --help` 命令列出所有根级选项。

- `-C <dir>` 可用来从默认当前工作目录覆盖项目目录。
- `-B <dir>` 可用来从项目目录默认的 `build` 子目录覆盖构建目录。
- `--ccache` 可用来在编译源文件时启用 **CCache**，安装了 **CCache** 工具后可极大缩短编译时间。

请注意，一些旧版本的 **CCache** 在某些平台上可能会出现 **bug**，因此如果文件没有按预期重新构建，请尝试禁用 **CCache** 后再次构建。通过设置环境变量 `IDF_CCACHE_ENABLE` 为非零值，可以默认启用 **CCache**。

- `-v` 可以让 `idf.py` 和编译系统产生详细的编译输出，对于调试编译问题会非常有用。
- `--cmake-warn-uninitialized` ``（或 ``-w`）会让 **CMake** 在项目目录内打印未初始化的变量警告（不包括在项目目录外的目录），这一选项只能控制 **CMake** 内部的 **CMake** 变量警告，不包括其它类型的编译警告。可以通过设置环境变量 `IDF_CMAKE_WARN_UNINITIALIZED` 为非零值，从而永久设置该选项。

## 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。

**注解：**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 [编写纯 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：

```
ninja flash
```

或：

```
make app-flash
```

可用的目标还包括：`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：

```
ESPPORT=/dev/ttyUSB0 ninja flash
```

**注解：** 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。

或：

```
make -j3 app-flash ESPPORT=COM4 ESPBAUD=2000000
```

**注解：** 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 适用于所有支持的 Python 版本。即使您系统中默认的 python 解释器仍是 Python 2.7，ESP-IDF 也可以使用，但建议您升级至 Python 3。

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。

**潜在问题** 使用 idf.py 可能会出现如下 ImportError 错误：

```
Traceback (most recent call last):
  File "/Users/user_name/e/esp-idf/tools/kconfig_new/configgen.py", line 27, in
  →<module>
    import kconfiglib
ImportError: bad magic number in 'kconfiglib': b'\x03\xf3\r\n'
```

该错误通常是由不同 Python 版本生成的 .pyc 文件引起的，可以通过运行以下命令解决该问题：

```
idf.py python-clean
```

### 4.2.3 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/
      - CMakeLists.txt
      - Kconfig
      - src1.c
    - component2/
      - CMakeLists.txt
      - Kconfig
      - src1.c
      - include/
        - component2.h
  - 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 变量。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”项目配置文件，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部分自定义组件，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导入第三方（不属于 ESP-IDF）的组件。或者，您也可以在此顶层 CMakeLists.txt 中设置 `EXTRA_COMPONENT_DIRS` 变量以查找其他指定位置处的组件。有关详细信息，请参阅[重命名 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。
- “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。
- “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，在主构建进程的运行期间，该目录还会保存临时目标文件、库文件以及最终输出的二进制文件。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` 时展示的[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[构建的内部过程](#)。

#### 4.2.4 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：

```

cmake_minimum_required(VERSION 3.5)
include($ENV{IDF_PATH}/tools/cmake/project.cmake)
project(myProject)

```

## 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 3.5)` 必须放在 `CMakeLists.txt` 文件的第一行，它会告诉 CMake 构建该项目所需要的最小版本号。ESP-IDF 支持 CMake 3.5 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、检索组件等任务。
- `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`、`COMPONENTS_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`、`EXTRA_COMPONENTS_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`：要构建进项目中的组件名称列表，默认为 `COMPONENT_DIRS` 目录下检索到的所有组件。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` 组件会减轻上述这些幕后工作量，但要求用户指定重命名后的组件位置，并手动为其添加依赖项。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。

### 4.2.5 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，然后是项目组件，最后是 `EXTRA_COMPONENT_DIRS` 中的组件。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，则使用搜索到的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。

## 最小组件 CMakeLists 文件

最小组件 `CMakeLists.txt` 文件通过使用 `idf_component_register` 将组件添加到构建系统中。

```
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会编译进组件库中。
- `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` 变量一样，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。

以下变量在项目级别中被设置，但可在组件 `CMakeLists` 中使用：

- `CONFIG_*`：项目配置中的每个值在 `cmake` 中都对应一个以 `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。

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 命令来传递编译器选项：

```
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：

```
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，那这么做可能会很有效。

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。

## 4.2.6 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，在此基础上进行修改。

有关示例请参阅[添加条件配置](#)。

## 4.2.7 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。

## 4.2.8 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”。

### 编写组件

```
idf_component_register(...
    REQUIRES mbedtls
    PRIV_REQUIRES console spiffs)
```

- REQUIRES 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 REQUIRES 中）。以及是当前组件正常工作必须要链接的组件。
- REQUIRES 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 REQUIRES 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。

**注解：**在 CMake 中，REQUIRES 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：

```
- autoProject/
  - CMakeLists.txt
  - components/ - car/ - CMakeLists.txt
                    - car.c
                    - car.h
                    - engine/ - CMakeLists.txt
                              - engine.c
                              - include/ - engine.h
  - spark_plug/ - CMakeLists.txt
                - plug.c
                - plug.h
```

**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：

```
/* car.h */
#include "engine.h"

#ifdef ENGINE_IS_HYBRID
#define CAR_MODEL "Hybrid"
#endif
```

同时 `car.c` 也包含了 `car.h`：

```
/* car.c */
#include "car.h"
```

这代表文件 `car/CMakeLists.txt` 需要声明 `car` 需要 `engine`：

```
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:

```
/* engine.h */
#define ENGINE_IS_HYBRID

void engine_start(void);
```

在 `engine.c` 中执行:

```
/* 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`:

```
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于的冗长。

**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 `REQUIRES` 或 `PRIV_REQUIRES`。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：freertos, newlib, heap, log, soc, esp\_rom, esp\_common, xtensa, cxx。

## 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组件列表，可以显著减少项目的构建时间。

## 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` 变量已排好序，依赖组件会排在前面。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。

## 4.2.9 构建的内部过程

关于 CMake 以及 CMake 命令的详细信息，请参阅 [CMake v3.5 官方文档](#)。

### project.cmake 的内容

当项目 `CMakeLists` 文件导入 `project.cmake` 文件时，`project.cmake` 会定义一些实用的模块和全局变量。如果系统环境中没有设置 `IDF_PATH`，那么它还会自动设置 `IDF_PATH` 变量。

`project.cmake` 文件还重写了 CMake 内置的 `project` 函数，以添加所有 ESP-IDF 项目特有的功能。

### project 函数

自定义的 `project()` 函数会执行以下步骤：

- 确定硬件目标（由 `IDF_TARGET` 环境变量设置），并将其保存在 CMake cache 中。如果环境变量中设置的硬件目标与 CMake cache 中的不匹配，则会报错并退出。
- 计算组件依赖，并构造 `BUILD_COMPONENTS` 变量，它是包含所有需要导入到构建系统中的组件列表（详情请见上文）。

- 查找项目中所有的组件（搜索 `COMPONENT_DIRS`，并按 `COMPONENTS` 进行过滤（前提是设置了该变量）。
- 从 `sdkconfig` 文件中加载项目配置信息，生成 `sdkconfig.cmake` 和 `sdkconfig.h` 文件，分别用在 CMake 和 C/C++ 中定义配置项。如果项目配置发生了更改，CMake 会自动重新运行，重新生成上述两个文件，接着重新配置项目。
- 根据硬件目标（`IDF_TARGET`）的值，将 `CMAKE_TOOLCHAIN_FILE` 变量设置为相应的工具链文件。
- 调用 CMake 的 `project` 函数声明实际的 CMake-level 项目。
- 加载 git 版本号。如果在 git 中检出了新的版本，就会使用一些技巧重新运行 CMake。详情请参考[文件通配 & 增量构建](#)。
- 从包含有 `project_include.cmake` 文件的组件中导入该文件。
- 将每个组件都添加到构建系统中。每个组件的 `CMakeLists` 文件都会调用 `register_component` 函数，它会调用 CMake 的 `add_library` 函数来添加一个库，然后添加源文件、编译选项等。
- 将最终的应用程序可执行文件添加到构建系统中。
- 返回并为组件之间指定依赖关系（将每个组件的公共头文件目录添加到其他组件中）。

更多详细信息请参阅 [/tools/cmake/project.cmake](#) 文件和 [/tools/cmake/idf\\_functions.cmake](#) 文件。

## CMake 调试

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，`idf.py` 在调用 CMake 时会给它传递 `--warn-uninitialized` 标志，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。

如果您不想启用此功能，可以给 `idf.py` 传递 `--no-warnings` 标志。

更多信息，请参考文件 [/tools/cmake/project.cmake](#) 以及 [/tools/cmake/idf\\_functions.cmake](#) 中支持的函数。

## 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` 对于大多数常见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。

**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，调用了 `register_config_only_component()` 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。

#### 4.2.10 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:

```
config 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:

```
config ENABLE_LCD_OUTPUT
    bool "Enable LCD output."
    help
        Select this if your board has a LCD.
```

(下页继续)

```

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:

```

if(CONFIG_ENABLE_LCD_OUTPUT)
  set(srcs lcd-real.c lcd-spi.c)
else()
  set(srcs lcd-dummy.c)
endif()

# 如果启用了控制台或绘图功能，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：

```

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)
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，该文件也应该要被清除。因此，要将该文件添加到 `ADDITIONAL_MAKE_CLEAN_FILES` 属性中。

**注解：** 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，但是您又不希望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");
```

符号名会根据文件全名生成，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，请参考 [protocols/https\\_request](#)，证书文件的内容会在编译时从 `.pem` 文件中加载。

### 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：

```

# 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}
    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` 变量设置为空。
  - 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。

**注解：**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 `DEPENDS` 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 `idf.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备增量式构建的能力，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。

### 4.2.11 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，请设置 `SDKCONFIG_DEFAULTS` 环境变量。

#### 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` 覆盖了 `sdkconfig` 默认文件的名称，则硬件目标的 `sdkconfig` 默认文件名也会从 `SDKCONFIG_DEFAULTS` 值中派生。

### 4.2.12 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，构建目录将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：

- `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`：

```
python esptool.py --chip {IDF_TARGET_PATH_NAME} 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。

### 4.2.13 构建 Bootloader

引导程序默认作为 `idf.py build` 的一部分被构建，也可以通过 `idf.py bootloader` 来单独构建。

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）。

### 4.2.14 选择硬件目标

当前 ESP-IDF 仅支持一个硬件目标，即 `esp32`，这也是构建系统默认的硬件目标。开发人员可以按照如下方法来添加对新硬件目标的支持：

```
rm sdkconfig
idf.py -DIDF_TARGET=new_target reconfigure
```



### 4.2.15 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：

```
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。

### 4.2.16 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想库 foo，相应的组件 CMakeLists 文件如下所示：

```
# 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)

# 创建并导入第三方库目标
add_subdirectory(foo)

# 将 `foo` 目标公开链接至 `main` 组件
target_link_libraries(main PUBLIC foo)
```

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。建议仔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 文件以及构建结构也会有所帮助。

用这种方式还可以将第三方库封装成 ESP-IDF 的组件。例如 [mbedtls](#) 组件就是封装了 [mbedtls 项目](#) 得到的。详情请参考 [mbedtls 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。

### 4.2.17 组件中使用预建库

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：

```
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目前的项目相同。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](#)。

## 4.2.18 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_minimum_required(VERSION 3.5)
project(my_custom_app C)

# 导入提供 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到可执行文件是什么，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。

**注解：** 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。

## 4.2.19 ESP-IDF CMake 构建系统 API

### idf 构建命令

```
idf_build_get_property(var property [GENERATOR_EXPRESSION])
```

检索一个构建属性 `property`，并将其存储在当前作用域可访问的 `var` 中。特定 `GENERATOR_EXPRESSION` 将检索该属性的生成器表达式字符串（不是实际值），它可与支持生成器表达式的 CMake 命令一起使用。

```
idf_build_set_property(property val [APPEND])
```

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。

```
idf_build_component(component_dir)
```

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）

```
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 ...])
```

为导入 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** - 默认用于构建的文件，默认为空；
- **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。

```
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。

```
idf_build_get_config(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，即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。

### 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。

```
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。

- `COMPILE_OPTIONS` - 适用于所有组件的源文件（无论是 C 还是 C++）的编译选项。
- `COMPILE_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。
- `IDF_PATH` - ESP-IDF 路径；由 `IDF_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` - 构建过程中组件中的 `Kconfig.projbuild` 文件的列表；由 `idf_build_process` 设置。
- `PROJECT_NAME` - 项目名称；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` - 配置默认文件的完整路径；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。

## idf 组件命令

```
idf_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。

```
idf_component_set_property(component property val [APPEND])
```

设置指定的 *component* 的 *组件属性 property* 的值为 *val*。特定 *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。

```
idf_component_register([[SRCS src1 src2 ...] | [[SRC_DIRS dir1 dir2 ...] [EXCLUDE_
↔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 ...]
                        [EMBED_FILES file1 file2 ...]
                        [EMBED_TXTFILES file1 file2 ...])
```

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。
- **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* 中通配符的限制。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。

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

### 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` 组件的目录。

```
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\_LIB** - 所创建的组件静态/接口库的名称；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。
- **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；由 `idf_component_register` **PRIV\_REQUIRES** 参数设置。
- **REQUIRED\_IDF\_TARGETS** - 组件支持的目标列表；由 `idf_component_register` **EMBED\_TXTFILES** 参数设置。
- **REQUIRES** - 公共组件依赖关系列表；由 `idf_component_register` **REQUIRES** 参数设置。
- **SRCS** - 组件源文件列表；由 `idf_component_register` 的 **SRCS** 或 **SRC\_DIRS/EXCLUDE\_SRCS** 参数设置。

### 4.2.20 文件通配 & 增量构建

在 ESP-IDF 组件中添加源文件的首选方法是在 COMPONENT\_SRCS 中手动列出它们:

```
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:

```
idf_component_register(SRC_DIRS library platform
    ...)
```

后台会使用通配符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, 如果你需要与其他使用 Git 等版本控制工具的开发人员共享项目时, 问题就会变得更加困难, 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。

### 4.2.21 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 在构建的过程中会在 `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 `flash_*_args` 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

`confserver.py` 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 confserver` 从项目中运行 `confserver.py`, 也可以使用不同的构建生成器来触发类似的目标。

配置服务器会向 `stderr` 输出方便阅读的错误和警告信息，向 `stdout` 输出 JSON 文件。启动时，配置服务器将以 JSON 字典的形式输出系统中每个配置项的完整值，以及范围受限的值的可用范围。`sdkconfig.json` 中包含有相同的信息：

```
{ "version": 1, "values": { "ITEM": "value", "ITEM_2": 1024, "ITEM_3": false },
  ↪ "ranges" : { "ITEM_2" : [ 0, 32768 ] } }
```

配置服务器仅发送可见的配置项，其它不可见的或者被禁用的配置项可从 `kconfig_menus.json` 静态文件中解析得到，此文件还包含菜单结构和其它元数据（描述、类型、范围等）。

然后配置服务器将等待客户端的输入，客户端会发起请求，要求更改一个或多个配置项的值，内容的格式是个 JSON 对象，后面跟一个换行符：

```
{ "version": "1", "set": { "SOME_NAME": false, "OTHER_NAME": true } }
```

配置服务器将解析此请求，更新 `sdkconfig` 文件，并返回完整的变更列表：

```
{ "version": 1, "values": { "SOME_NAME": false, "OTHER_NAME": true, "DEPENDS_ON_
  ↪ SOME_NAME": null } }
```

当前不可见或者禁用的配置项会返回 `null`，任何新的可见配置项则会返回其当前新的可见值。

如果配置项的取值范围因另一个值的变化发生了改变，那么配置服务器会发送：

```
{ "version": 1, "values": { "OTHER_NAME": true }, "ranges" : { "HAS_RANGE" : [ 3, 4
  ↪ ] } }
```

如果传递的数据无效，那么 JSON 对象中会有 `error` 字段：

```
{ "version": 1, "values": {}, "error": [ "The following config symbol(s) were not
  ↪ visible so were not updated: NOT_VISIBLE_ITEM" ] }
```

默认情况下，变更后的配置不会被写进 `sdkconfig` 文件。更改的内容在发出 “save” 命令之前会先储存在内存中：

```
{ "version": 1, "save": null }
```

若要从已保存的文件中重新加载配置值，并丢弃内存中的任何更改，可以发送 “load” 命令：

```
{ "version": 1, "load": null }
```

“load” 和 “save” 的值可以是新的路径名，也可以设置为 “null” 用以加载/保存之前使用的路径名。

配置服务器对 “load” 命令的响应始终是完整的配置值和取值范围的集合，这与服务器初始启动阶段的响应相同。

“load”、“set” 和 “save” 的任意组合可以在一条单独的命令中发送出去，服务器按照组合中的顺序执行命令。因此，可以使用一条命令实现从文件中加载配置，更新配置值，然后将其保存到文件中。

---

**注解：** 配置服务器不会自动加载外部对 `sdkconfig` 文件的任何更改。如果文件被外部编辑，则需要发送 “load” 命令或重启服务器。

---



---

**注解：** `sdkconfig` 文件更新后，配置服务器不会重新运行 CMake 来生成其它的构建文件和元数据文件。这些文件会在下一次运行 CMake 或 `idf.py` 时自动生成。

---

## 4.2.22 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

`/tools/cmake/convert_to_cmake.py` 中提供了一个项目自动转换工具。运行此命令时需要加上项目路径，如下所示：

```
$IDF_PATH/tools/cmake/convert_to_cmake.py /path/to/project_dir
```

项目目录必须包含 `Makefile` 文件，并确保主机已安装 GNU Make (`make`) 工具，并且被添加到了 `PATH` 环境变量中。

该工具会将项目 `Makefile` 文件和所有组件的 `component.mk` 文件转换为对应的 `CMakeLists.txt` 文件。

转换过程如下：该工具首先运行 `make` 来展开 ESP-IDF 构建系统设置的变量，然后创建相应的 `CMakeLists` 文件来设置相同的变量。

**重要：**当转换工具转换一个 `component.mk` 文件时，它并不能确定该组件依赖于哪些其他组件。这些信息需要通过编辑新的组件 `CMakeLists.txt` 文件并添加 `REQUIRES` 和/或 `PRIV_REQUIRES` 子句来手动添加。否则，组件中的源文件会因为找不到其他组件的头文件而编译失败。请参考[组件依赖](#)获取更多信息。

转换工具并不能处理复杂的 `Makefile` 逻辑或异常的目标，这些需要手动转换。

### 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_libraries` 命令替代。
- `COMPONENT_ADD_LINKER_DEPS`：链接过程依赖的文件列表。`target_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_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_OWNBUILDTARGET` & `COMPONENT_OWNCLEANTARGET`：已被 CMake [外部项目](#) 替代，详细内容请参阅[完全覆盖组件的构建过程](#)。
- `COMPONENT_CONFIG_ONLY`：已被 `register_config_only_component()` 函数替代，请参阅[仅配置组件](#)。
- `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` 不能再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](#)。

## 4.3 构建系统 (传统 GNU Make)

---

**注解:** Since ESP-IDF V4.0, the default build system is based on CMake. This documentation is for the legacy build system based on GNU Make. Support for this build system may be removed in future major releases.

---

本文将介绍乐鑫物联网开发框架中的 构建系统和 组件的相关概念。

如果您想了解如何构建一个新的 ESP-IDF 项目, 请阅读本文档。

我们建议您使用 [ESP-IDF 模板工程](#) 来开始您的新项目。

### 4.3.1 使用构建系统

ESP-IDF 的 [README.md](#) 文件对如何使用构建系统来构建项目作了简要的说明。

### 4.3.2 概述

一个 ESP-IDF 项目可以看作是许多不同组件的集合, 例如对于一个展示当前湿度的网站服务器来说, 它可能会包含如下一些组件:

- ESP32 基础库 (libc, rom bindings 等)
- WiFi 驱动库
- TCP/IP 协议栈
- FreeRTOS 操作系统
- 网站服务器
- 湿度传感器的驱动
- 将上述组件组织在一起的主代码

ESP-IDF 可以显式地指定和配置每个组件。在构建项目的时候, 构建系统会查找 ESP-IDF 目录、项目目录和用户自定义目录 (可选) 中所有的组件, 然后使用基于文本的菜单系统让用户配置 ESP-IDF 项目中需要的每个组件。在配置结束后, 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文件的所有源文件和配置，还有其他的支持型输出文件，比如分区表、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，它可以通过 `make menuconfig` 进行修改，且一个项目只能包含一个项目配置。
- 应用程序是由 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（主可执行文件，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的引导程序）。
- 组件是模块化的、独立的代码，它们被编译成静态库（.a 文件）后再链接成应用程序，有些组件是 ESP-IDF 官方提供的，有些则可能来自其它项目。

以下内容并不是项目的组成部分：

- ESP-IDF 并不是项目的一部分，相反，它是独立的，并通过 `IDF_PATH` 环境变量链接到项目中，这样做就可以使 IDF 框架与您的项目分离，其中 `IDF_PATH` 变量保存了 ESP-IDF 目录的路径。
- 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，或者可以在项目配置中显式指定工具链的前缀为本地的安装路径。

## 示例项目

示例项目的目录树结构可能如下所示：

```

- myProject/
  - Makefile
  - sdkconfig
  - components/
    - component1/
      - component.mk
      - Kconfig
      - src1.c
    - component2/
      - component.mk
      - Kconfig
      - src1.c
      - include/
        - component2.h
  - main/
    - src1.c
    - src2.c
    - component.mk
  - build/

```

该示例项目 `myProject` 包含以下组成部分：

- 项目顶层 `Makefile`，该 `Makefile` 设置了 `PROJECT_NAME` 变量，还可以定义作用于整个项目的其它 `make` 变量（可选）。顶层 `Makefile` 会导入核心 `Makefile` 文件 `$(IDF_PATH)/make/project.mk`，由它负责实现 ESP-IDF 构建系统的剩余部分。
- 项目配置文件 `sdkconfig`，执行 `make menuconfig` 后会创建或更新此文件，该文件中保存了项目中所有组件的配置信息（包括 ESP-IDF 本身）。`sdkconfig` 文件可能会也可能不会被添加到项目的源代码管理系统中。
- 可选的组件目录中包含了属于项目一部分的自定义组件，不是每一个项目都需要它，但它有助于构建可重用代码或者导入第三方组件。
- `main` 目录是一个特殊的伪组件，它包含项目本身的源代码。`main` 是默认名称，`Makefile` 变量 `COMPONENT_DIRS` 默认会导入此组件，但您也可以修改此变量（或者设置 `EXTRA_COMPONENT_DIRS`）以查找其他位置的组件。
- `build` 目录在项目构建的时候创建或者更新，里面包含有构建生成的临时目标文件和库以及最终输出的二进制文件。此目录通常不会被添加到项目的源代码管理系统中，也不会随着项目源代码被发布。

组件目录中会包含组件自己的 `Makefile` 文件 `component.mk`，里面会定义一些变量来控制该组件的构建过程，以及它与整个项目的集成。更多详细信息请参考[组件 Makefiles](#)。

每个组件还可以包含一个 `Kconfig` 文件，它用于定义 `menuconfig` 时展示的组件配置信息的选项规则。某些组件还可能还会包含 `Kconfig.projbuild` 和 `Makefile.projbuild` 特殊文件，他们可以用来覆盖项目的部分配置。

## 项目 Makefiles

每个项目都有一个 Makefile，它包含整个项目的构建设置。默认情况下，项目 Makefile 可以非常小。

### 最小 Makefile 示例

```
PROJECT_NAME := myProject

include $(IDF_PATH)/make/project.mk
```

### 必须设置的项目变量

- PROJECT\_NAME: 项目名称，最终输出的二进制文件也使用该名称，即 myProject.bin，myProject.elf。

**可选的项目变量** 以下这些变量都有默认值，用户可以重写这些变量以自定义构建行为。查看 make/project.mk 文件可以获得所有的实现细节。

- PROJECT\_PATH: 顶层项目目录，默认是包含 Makefile 文件的目录，许多其他的项目变量都基于此变量。注意，项目路径中不能包含有空格。
- BUILD\_DIR\_BASE: 所有对象、库、二进制文件的输出目录，默认为 \$(PROJECT\_PATH)/build。
- COMPONENT\_DIRS: 组件的搜索目录，默认为 \$(IDF\_PATH)/components, \$(PROJECT\_PATH)/components, \$(PROJECT\_PATH)/main 和 EXTRA\_COMPONENT\_DIRS。如果您不希望从这些目录中搜索组件，请重写此变量。
- EXTRA\_COMPONENT\_DIRS: 组件额外的搜索路径，可选。
- COMPONENTS: 要构建进项目中的组件列表，默认为 COMPONENT\_DIRS 指定目录中所有的组件。
- EXCLUDE\_COMPONENTS: 在构建的过程中需要剔除的组件列表，可选。请注意这只会减少构建的时间，并不会减少最终二进制文件的大小。
- TEST\_EXCLUDE\_COMPONENTS: 在构建单元测试的过程中需要剔除的组件列表，可选。

以上这些 Makefile 变量中的任何路径都要使用绝对路径，您可以使用 \$(PROJECT\_PATH)/xxx, \$(IDF\_PATH)/xxx, 或者使用 Make 内置函数 \$(abspath xxx) 将相对路径转换为绝对路径。

以上这些变量要在 Makefile 中 include \$(IDF\_PATH)/make/project.mk 的前面进行设置。

## 组件 Makefiles

每个项目都包含一个或者多个组件，这些组件可以是 ESP-IDF 的一部分，也可以从其他组件目录添加。

组件是包含 component.mk 文件的任何目录。

### 搜索组件

搜索 COMPONENT\_DIRS 中指定的目录以查找项目会使用的组件，目录可以是组件本身（即他们包含 component.mk 文件），也可以是包含组件的上层目录。

运行 make list-components 命令可以查询这些变量的值，这有助于调试组件的搜索路径是否正确。

**同名组件** ESP-IDF 搜索组件时，会按照 COMPONENT\_DIRS 指定的顺序依次进行，这意味着在默认情况下，首先是 ESP-IDF 组件，然后是项目组件，最后是 EXTRA\_COMPONENT\_DIRS 中的组件。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，则使用搜索到的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来覆盖 ESP-IDF 组件，如果使用这种方式，ESP-IDF 目录本身可以保持不变。

**最小组件 Makefile** 最简单的 `component.mk` 文件可以是一个空文件，如果文件为空，则组件的默认构建行为会被设置为：

- `makefile` 所在目录中的所有源文件 (`*.c`, `*.cpp`, `*.cc`, `*.S`) 将会被编译进组件库中。
- 子目录 `include` 将被添加到其他组件的全局头文件搜索路径中。
- 组件库将会被链接到项目的应用程序中。

更完整的组件 `makefile` 可以查看组件 *Makefile* 示例。

请注意，空的 `component.mk` 文件同没有 `component.mk` 文件之间存在本质差异，前者会调用默认的组件构建行为，后者不会发生默认的组件构建行为。一个组件中如果只包含影响项目配置或构建过程的文件，那么它可以没有 `component.mk` 文件。

**预设的组件变量** 以下特定于组件的变量可以在 `component.mk` 中使用，但不应该被修改。

- `COMPONENT_PATH`: 组件的目录，即包含 `component.mk` 文件的绝对路径，路径中不能包含空格。
- `COMPONENT_NAME`: 组件的名字，默认为组件目录的名称。
- `COMPONENT_BUILD_DIR`: 组件的构建目录，即存放组件构建输出的绝对路径，它是 `$(BUILD_DIR_BASE)` 的子目录。该变量也是构建组件时的当前工作目录，所以 `make` 中的相对路径都以此目录为基础。
- `COMPONENT_LIBRARY`: 组件构建后的静态库文件（相对于组件的构建目录）的名字，默认为 `$(COMPONENT_NAME).a`。

以下变量在项目顶层中设置，并被导出到组件中构建时使用：

- `PROJECT_NAME`: 项目名称，在项目的 `Makefile` 中设置。
- `PROJECT_PATH`: 包含项目 `Makefile` 的目录的绝对路径。
- `COMPONENTS`: 此次构建中包含的所有组件的名字。
- `CONFIG_*`: 项目配置中的每个值在 `make` 中都对应一个以 `CONFIG_` 开头的变量。
- `CC`, `LD`, `AR`, `OBJCOPY`: `gcc xtensa` 交叉编译工具链中每个工具的完整路径。
- `HOSTCC`, `HOSTLD`, `HOSTAR`: 主机本地工具链中每个工具的全名。
- `IDF_VER`: `ESP-IDF` 的版本号，可以通过检索 `$(IDF_PATH)/version.txt` 文件（假如存在的话）或者使用 `git` 命令 `git describe` 来获取。这里推荐的格式是在一行中指定主 `IDF` 的发布版本号，例如标记为 `v2.0` 的发布版本或者是标记任意一次提交记录的 `v2.0-275-g0efaa4f`。应用程序可以通过调用 `esp_get_idf_version()` 函数来使用该变量。
- `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。

如果您在 `component.mk` 文件中修改这些变量，这并不会影响其它组件的构建，但可能会使您的组件变得难以构建或调试。

**可选的项目通用组件变量** 可以在 `component.mk` 中设置以下变量来控制整个项目的构建行为：

- `COMPONENT_ADD_INCLUDEDIRS`: 相对于组件目录的路径，将被添加到项目中所有组件的头文件搜索路径中。如果该变量未被覆盖，则默认为 `include` 目录。如果一个头文件路径仅仅为当前组件所用，那么应该将该路径添加到 `COMPONENT_PRIV_INCLUDEDIRS` 中。
- `COMPONENT_ADD_LDFLAGS`: 添加链接参数到全局 `LDFLAGS` 中用以指导链接最终的可执行文件，默认为 `-l$(COMPONENT_NAME)`。如果将预编译好的库添加到此目录，请使用它们为绝对路径，即 `$(COMPONENT_PATH)/libwhatever.a`。
- `COMPONENT_DEPENDS`: 需要在当前组件之前构建的组件列表，这对于处理链接时的依赖不是必需的，因为所有组件的头文件目录始终可用。如果一个组件会生成一个头文件，然后另外一个组件需要使用它，此时该变量就有必要进行设置。大多数的组件不需要设置此变量。
- `COMPONENT_ADD_LINKER_DEPS`: 保存一些文件的路径，当这些文件发生改变时，会触发 `ELF` 文件重新链接。该变量通常用于链接脚本文件和二进制文件，大多数的组件不需要设置此变量。

以下变量仅适用于属于 `ESP-IDF` 的组件：

- `COMPONENT_SUBMODULES`: 组件使用的 `git` 子模块的路径列表（相对于 `COMPONENT_PATH`）。这些路径会在构建的过程中被检查（并在必要的时候初始化）。如果组件位于 `IDF_PATH` 目录之外，则忽略此变量。

**可选的组件特定变量** 以下变量可以在 `component.mk` 中进行设置，用以控制该组件的构建行为：

- `COMPONENT_PRIV_INCLUDEDIRS`：相对于组件目录的目录路径，该目录仅会被添加到该组件源文件的头文件搜索路径中。
- `COMPONENT_EXTRA_INCLUDES`：编译组件的源文件时需要指定的额外的头文件搜索路径，这些路径将以 `-I` 为前缀传递给编译器。这和 `COMPONENT_PRIV_INCLUDEDIRS` 变量的功能有些类似，但是这些路径不会相对于组件目录进行扩展。
- `COMPONENT_SRCDIRS`：相对于组件目录的目录路径，这些路径用于搜索源文件（`*.cpp`，`*.c`，`*.s`），默认为 `.`，即组件目录本身。重写该变量可以指定包含源文件的不同目录列表。
- `COMPONENT_OBJS`：要编译生成的目标文件，默认是 `COMPONENT_SRCDIRS` 中每个源文件的 `.o` 文件。重写该变量将允许您剔除 `COMPONENT_SRCDIRS` 中的某些源文件，否则他们将会被编译。相关示例请参阅指定需要编译的组件源文件。
- `COMPONENT_EXTRA_CLEAN`：相对于组件构建目录的路径，指向 `component.mk` 文件中自定义 `make` 规则生成的任何文件，它们也是 `make clean` 命令需要删除的文件。相关示例请参阅源代码生成。
- `COMPONENT_OWNBUILDTARGET & COMPONENT_OWNCLEANTARGET`：这些目标允许您完全覆盖组件的默认编译行为。有关详细信息，请参阅完全覆盖组件的 *Makefile*。
- `COMPONENT_CONFIG_ONLY`：如果设置了此标志，则表示组件根本不会产生构建输出（即不会构建得到 `COMPONENT_LIBRARY`），并且会忽略大多数其它组件变量。此标志用于 IDF 内部组件，其中仅包含 `Kconfig.projbuild` 和/或 `Makefile.projbuild` 文件来配置项目，但是没有源文件。
- `CFLAGS`：传递给 C 编译器的标志。根据项目设置已经定义一组默认的 `CFLAGS`，可以通过 `CFLAGS +=` 来为组件添加特定的标志，也可以完全重写该变量（尽管不推荐这么做）。
- `CPPFLAGS`：传递给 C 预处理器的标志（用于 `.c`，`.cpp` 和 `.s` 文件）。根据项目设置已经定义一组默认的 `CPPFLAGS`，可以通过 `CPPFLAGS +=` 来为组件添加特定的标志，也可以完全重写该变量（尽管不推荐这么做）。
- `CXXFLAGS`：传递给 C++ 编译器的标志。根据项目设置已经定义一组默认的 `CXXFLAGS`，可以通过 `CXXFLAGS +=` 来为组件添加特定的标志，也可以完全重写该变量（尽管不推荐这么做）。

如果要将编译标志应用于单个源文件，您可以将该源文件的目标规则覆盖，例如：

```
apps/dhcpserver.o: CFLAGS += -Wno-unused-variable
```

如果上游代码在编译的时候发出了警告，那这么做可能会很有效。

## 配置组件

每个组件都可以包含一个 `Kconfig` 文件，和 `component.mk` 放在同一个目录下。`Kconfig` 中包含此组件在 `make menuconfig` 时要展示的配置规则的设置。

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。

## 预处理器定义

ESP-IDF 构建系统会在命令行中添加以下 C 预处理定义：

- `ESP_PLATFORM` — 可以用来检测在 ESP-IDF 内发生的构建行为。
- `IDF_VER` — ESP-IDF 的版本，请参阅[预设的组件变量](#)。

## 构建的内部过程

### 顶层：项目 Makefile

- `make` 始终从项目目录处运行，并且项目的 `makefile` 名字通常为 `Makefile`。
- 项目的 `makefile` 文件会设置 `PROJECT_NAME`，并且可以自定义其他可选的项目变量。

- 项目 `makefile` 文件会导入 `$(IDF_PATH)/make/project.mk`，该文件中会导入项目级的 `Make` 逻辑。
- `project.mk` 填写默认的项目级 `make` 变量，并导入项目配置中的 `make` 变量。如果生成的包含项目配置的 `makefile` 文件已经过期，那么它将会被重新生成（通过 `project_config.mk` 中的目标规则），然后 `make` 进程从顶层重新开始。
- `project.mk` 根据默认组件目录或者可选项目变量中设置的自定义组件列表来编译组件。
- 每个组件都可以设置一些可选的项目通用组件变量，他们会通过 `component_project_vars.mk` 被导入 `project.mk` 文件中。如果这些文件有缺失或者过期，他们会被重新生成（通过对组件 `makefile` 的递归调用），然后 `make` 进程从顶层重新开始。
- 组件中的 `Makefile.projbuild` 文件被包含在了 `make` 的进程中，以添加额外的目标或者配置。
- 默认情况下，项目 `makefile` 还为每个组件生成顶层的编译和清理目标，并设置 `app` 和 `clean` 目标来调用所有这些子目标。
- 为了编译每个组件，对组件 `makefile` 执行递归构建。

为了更好地理解项目的构建过程，请通读 `project.mk` 文件。

## 第二层：组件 Makefile 文件

- 每次调用组件 `makefile` 文件都是通过 `$(IDF_PATH)/make/component_wrapper.mk` 这个包装器进行的。
- 此组件包装器包含了所有组件的 `Makefile.componentbuild` 文件，使这些文件中的任何配置，变量都可用于每个组件。
- 调用 `component_wrapper.mk` 时将当前目录设置为组件构建目录，并将 `COMPONENT_MAKEFILE` 变量设置为 `component.mk` 的绝对路径。
- `component_wrapper.mk` 为所有组件变量设置默认值，然后导入 `component.mk` 文件来覆盖或修改这些变量。
- 如果未定义 `COMPONENT_OWNBUILDTARGET` 和 `COMPONENT_OWNCLEANTARGET` 文件，则会为组件的源文件和必备组件 `COMPONENT_LIBRARY` 静态库文件创建默认构建和清理目标。
- `component_project_vars.mk` 文件在 `component_wrapper.mk` 中有自己的目标，如果由于组件的 `makefile` 或者项目配置的更改而需要重建此文件，则从 `project.mk` 文件中进行评估。

为了更好地理解组件制作过程，请阅读 `component_wrapper.mk` 文件和 ESP-IDF 中的 `component.mk` 文件。

## 以非交互的方式运行 Make

如果在运行 `make` 的时候不希望出现交互式提示（例如：在 IDE 或自动构建系统中），可以将 `BATCH_BUILD=1` 添加到 `make` 的参数中（或者将其设置为环境变量）。

设置 `BATCH_BUILD` 意味着：

- 详细输出（与 `v=1` 相同，见下文），如果不需要详细输出，就设置 `v=0`。
- 如果项目配置缺少新配置项（来自新组件或者 ESP-IDF 更新），则项目使用默认值，而不是提示用户输入每个项目。
- 如果构建系统需要调用 `menuconfig`，则会打印错误并且构建失败。

## 构建目标的进阶用法

- `make app`, `make bootloader`, `make partition table` 可以根据需要为项目单独构建生成应用程序文件、启动引导文件和分区表文件。
- `make erase_flash` 和 `make erase_otadata` 会调用 `esptool.py` 脚本分别擦除整块闪存芯片或者其中 OTA 分区的内容。
- `make size` 会打印应用程序的大小信息。`make size-components` 和 `make size-files` 两者功能相似，分别打印每个组件或者每个源文件大小的详细信息。

## 调试 Make 的过程

调试 ESP-IDF 构建系统的一些技巧：

- 将 `V=1` 添加到 `make` 的参数中（或将其设置为环境变量）将使 `make` 回显所有已经执行的命令，以及为子 `make` 输入的每个目录。
- 运行 `make -w` 将导致 `make` 在为子 `make` 输入时回显每个目录——与 `V=1` 相同但不回显所有命令。
- 运行 `make --trace`（可能除了上述参数之一）将打印出构建时的每个目标，以及导致它构建的依赖项。
- 运行 `make -p` 会打印每个 `makefile` 中每个生成的目标的（非常详细的）摘要。

更多调试技巧和通用的构建信息，请参阅 [GNU 构建手册](#)。

**警告未定义的变量** 默认情况下，如果引用了未定义的变量（如 `$(DOES_NOT_EXIST)`），构建过程将会打印警告，这对于查找变量名称中的错误非常有用。

如果不要此行为，可以在 `menuconfig` 顶层菜单下的 *SDK tool configuration* 中禁用它。

请注意，如果在 `Makefile` 中使用 `ifdef` 或者 `ifndef`，则此选项不会出发警告。

### 覆盖项目的部分内容

**Makefile.projbuild** 如果一个组件含有必须要在项目构建过程的顶层进行计算的变量，则可以在组件目录下创建名为 `Makefile.projbuild` 的文件，项目在执行 `project.mk` 的时候会导入此 `makefile`。

例如，如果您的组件需要为整个项目添加 `CFLAGS`（不仅仅是为自身的源文件），那么可以在 `Makefile.projbuild` 中设置 `CFLAGS +=`。

`Makefile.projbuild` 文件在 `ESP-IDF` 中大量使用，用于定义项目范围的构建功能，例如 `esptool.py` 命令行参数和 `bootloader` 这个特殊的程序。

请注意，`Makefile.projbuild` 对于最常见的组件不是必需的 - 例如向项目中添加 `include` 目录，或者将 `LDFLAGS` 添加到最终链接步骤，同样可以通过 `component.mk` 文件来自定义这些值。有关详细信息，请参阅 [可选的项目通用组件变量](#)。

**警告：** 在此文件中设置变量或者目标时要小心，由于这些值包含在项目的顶层 `makefile` 中，因此他们可以影响或者破坏所有组件的功能！

**KConfig.projbuild** 这相当于 `Makefile.projbuild` 的组件配置 `KConfig` 文件，如果要在 `menuconfig` 的顶层添加配置选项，而不是在组件配置子菜单中，则可以在 `component.mk` 文件所在目录中的 `KConfig.projbuild` 文件中定义这些选项。

在此文件中添加配置时要小心，因为他们将包含在整个项目配置中，在可能的情况下，通常最好为组件创建和配置 `KConfig` 文件。

**Makefile.componentbuild** 对于一些特殊的组件，比如它们会使用工具从其余文件中生成源文件，这时就有必要将配置、宏或者变量的定义添加到每个组件的构建过程中。这是通过在组件目录中包含 `Makefile.componentbuild` 文件来实现的。此文件在 `component.mk` 文件之前被导入 `component_wrapper.mk` 中。同 `Makefile.projbuild` 文件一样，请注意这些文件，因为他们包含在每个组件的构建中，所有只有在编译完全不同的组件时才会出现 `Makefile.componentbuild` 错误。

**仅配置的组件** 仅配置的组件是一类不包含源文件的特殊组件，只有 `Kconfig.projbuild` 和 `Makefile.projbuild` 文件，可以在 `component.mk` 文件中设置标志 `COMPONENT_CONFIG_ONLY`。如果设置了此标志，则忽略大多数其他组件变量，并且不会为组件执行构建操作。

### 组件 Makefile 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 `component.mk` 可能非常小，甚至是空的，请参考 [最小组件 Makefile](#)。但是某些功能通常需要覆盖组件的变量。

以下是 `component.mk` 的一些更高级的示例：

**增加源文件目录** 默认情况下，将忽略子目录。如果您的项目在子目录中而不是在组件的根目录中有源文件，那么您可以通过设置 `COMPONENT_SRCDIRS` 将其告知构建系统：

```
COMPONENT_SRCDIRS := src1 src2
```

构建系统将会编译 `src1/` 和 `src2/` 子目录中的所有源文件。

**指定源文件** 标准 `component.mk` 逻辑将源目录中的所有 `.S` 和 `.c` 文件添加为无条件编译的源。通过将 `COMPONENT_OBJS` 变量手动设置为需要生成的对象的名称，可以绕过该逻辑并对要编译的对象进行硬编码。

```
COMPONENT_OBJS := file1.o file2.o thing/filea.o thing/fileb.o anotherthing/main.o
COMPONENT_SRCDIRS := . thing anotherthing
```

请注意，还需要另外设置 `COMPONENT_SRCDIRS`。

**添加条件配置** 配置系统可用于有条件地编译某些文件，具体取决于 `make menuconfig` 中选择的选项。为此，ESP-IDF 具有 `compile_only_if` 和 `compile_only_if_not` 的宏：

Kconfig:

```
config FOO_ENABLE_BAR
    bool "Enable the BAR feature."
    help
        This enables the BAR feature of the FOO component.
```

`component.mk`:

```
$(call compile_only_if,$(CONFIG_FOO_ENABLE_BAR),bar.o)
```

从示例中可以看出，`compile_only_if` 宏将条件和目标文件列表作为参数。如果条件为真（在这种情况下：如果在 `menuconfig` 中启用了 `BAR` 功能），将始终编译目标文件（在本例中为 `bar.o`）。相反的情况也是如此，如果条件不成立，`bar.o` 将永远不会被编译。`compile_only_if_not` 执行相反的操作，如果条件为 `false` 则编译，如果条件为 `true` 则不编译。

这也可以用于选择或者删除实现，如下所示：

Kconfig:

```
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
```

`component.mk`:

```
# If LCD is enabled, compile interface to it, otherwise compile dummy interface
$(call compile_only_if,$(CONFIG_ENABLE_LCD_OUTPUT),lcd-real.o lcd-spi.o)
$(call compile_only_if_not,$(CONFIG_ENABLE_LCD_OUTPUT),lcd-dummy.o)
```

(下页继续)



```
#We need font if either console or plot is enabled
$(call compile_only_if,$(or $(CONFIG_ENABLE_LCD_CONSOLE),$(CONFIG_ENABLE_LCD_
↪PLOT)), font.o)
```

请注意使用 **Make** 或者函数来包含字体文件。其他的替换函数，比如 `and` 和 `if` 也适用于此处。也可以使用不在 `menuconfig` 中定义的变量，ESP-IDF 使用默认的 **Make** 策略，将一个空的或者只包含空格的变量视为 `false`，而其中任何非空格的比那辆都为 `true`。

(注意：本文档的历史版本建议将目标文件添加到 `COMPONENT_OBJS` 中，虽然这仍然可行，但是只有当组件中的所有目标文件都明确命名时才会起作用，并且在 `make clean` 后并不会清除 `make` 中取消选择的目标文件)。

**源代码生成** 某些组件会出现源文件未随组件本身提供，而必须从另外一个文件生成的情况。假设我们的组件有一个头文件，该文件由 **BMP** 文件转换后的二进制数据组成，假设使用 `bmp2h` 的工具进行转换，然后将头文件包含在名为 `graphics_lib.c` 的文件中：

```
COMPONENT_EXTRA_CLEAN := logo.h

graphics_lib.o: logo.h

logo.h: $(COMPONENT_PATH)/logo.bmp
    bmp2h -i $^ -o $@
```

这个示例会在当前目录（构建目录）中生成 `graphics_lib.o` 和 `logo.h` 文件，而 `logo.bmp` 随组件一起提供并位于组件路径下。因为 `logo.h` 是一个生成的文件，所以当调用 `make clean` 时需要清理它，这就是为什么要将它添加到 `COMPONENT_EXTRA_CLEAN` 变量中。

**润色与改进** 将 `logo.h` 添加作为 `graphics_lib.o` 的依赖项会导致在编译 `graphics_lib.c` 之前先生成它。

如果另一个组件中的源文件需要使用 `logo.h`，则必须将此组件的名称添加到另一个组件的 `COMPONENT_DEPENDS` 列表中，以确保组件按顺序编译。

**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，但是您又不希望将它重新格式化为 **C** 源文件。

这时，您可以在 `component.mk` 文件中设置变量 `COMPONENT_EMBED_FILES`，以这种方式指定要嵌入的文件名称：

```
COMPONENT_EMBED_FILES := server_root_cert.der
```

或者，如果文件是字符串，则可以使用变量 `COMPONENT_EMBED_TXTFILES`，这将把文本文件的内容当成以 `null` 结尾的字符串嵌入：

```
COMPONENT_EMBED_TXTFILES := server_root_cert.pem
```

文件的内容会被编译进 `flash` 中的 `.rodata` 段，并通过符号名称来访问，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");
```

符号名称是根据文件的全名生成的，如 `COMPONENT_EMBED_FILES` 中的所示，字符 `/`，`.`，等都会被下划线替代。符号名称中的 `_binary` 前缀由 `objcopy` 添加，对于文本和二进制文件都是相同的。

有关使用此技术的示例，请参考 [protocols/https\\_request](#) - 证书文件的内容会在编译时从 `.pem` 文件中加载。

## 完全覆盖组件的 Makefile

显然，在某些情况下，所有这些配置都不足以满足某个组件，例如，当组件基本上是另一个第三方组件的包装器时，该第三方组件最初不打算在 ESP-IDF 构建系统下工作，在这种情况下，可以通过设置 COMPONENT\_OWNBUILDTARGET 和可能的 COMPONENT\_OWNCLEANTARGET，并在 component.mk 中定义名为 build 和 clean 的目标。构建目标可以执行任何操作，只要它为项目生成了 \$(COMPONENT\_LIBRARY)，并最终被链接到应用程序二进制文件中即可。

(实际上，这并不是必须的 - 如果 COMPONENT\_ADD\_LDFLAGS 变量被覆盖，那么组件可以指示链接器链接其他二进制文件。)

## 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文件 sdkconfig.defaults，运行 make defconfig 或从头创建新配置时将会使用此文件。

要想覆盖此文件的名称，请设置 SDKCONFIG\_DEFAULTS 环境变量。

## 保存 flash 参数

在某些情况下，我们希望在没有 IDF 的情况下烧写目标板卡，对于这种情况，我们希望保存构建的二进制文件、esptool.py 和 esptool write\_flash 命令的参数。可以简单编写一段脚本来保存二进制文件和 esptool.py，并且使用命令 make print\_flash\_cmd 来查看烧写 flash 时的参数。

```
--flash_mode dio --flash_freq 40m --flash_size detect 0x1000 bootloader/bootloader.
↪bin 0x10000 example_app.bin 0x8000 partition_table_unit_test_app.bin
```

然后使用这段 flash 参数作为 esptool write\_flash 命令的参数：

```
python esptool.py --chip esp32 --port /dev/ttyUSB0 --baud 921600 --before default_
↪reset --after hard_reset write_flash -z --flash_mode dio --flash_freq 40m --
↪flash_size detect 0x1000 bootloader/bootloader.bin 0x10000 example_app.bin
↪0x8000 partition_table_unit_test_app.bin
```

## 4.3.3 构建 Bootloader

引导程序默认作为 make all 的一部分被构建，或者也可以通过 make bootloader-clean 来单独构建，此外还可以通过 make bootloader-list-components 来查看构建引导程序时包含的组件。

引导程序是一个特殊的组件，因为主项目中的二级引导程序拥有单独的.EFL 和.BIN 文件。但是它与主项目共享配置和构建目录。

这是通过在 components/bootloader/subproject 下添加子项目来完成的。这个子项目有自己的 Makefile，但它希望通过 components/bootloader/Makefile.projectbuild 文件中的一些配置使自己从主项目的 Makefile 被调用。有关详细信息，请参阅这些文件。

## 4.4 错误处理

### 4.4.1 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）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](#)。

## 4.4.2 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。

## 4.4.3 错误码到错误消息

错误代码并不直观，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 **标准 POSIX 错误代码** 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，从而减小应用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 `YYYYY` 分别代表错误代码的十六进制和十进制表示。

## 4.4.4 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：

```
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:0x3ffb4ff0 0x40087328:0x3ffb5010 0x400d1fdf:0x3ffb5030_
↳0x400d0816:0x3ffb5050
```

- 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。
- 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。
- 最后一行打印回溯结果。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。更多有关回溯结果的详细信息，请参阅 [不可恢复错误](#)。

**注解：** 如果使用 `IDF monitor`，则最后一行回溯结果中的地址将会被替换为相应的文件名和行号。

#### 4.4.5 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”；
  - 采用其他带外机制，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：

```
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
}
```

2. 将错误传递回调用程序。在某些中间件组件中，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：

```
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);
    // 将错误码传递给上层（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。对于中间件组件而言，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：

```
ESP_ERROR_CHECK(spi_bus_initialize(host, bus_config, dma_chan));
```

#### 4.4.6 C++ 异常

默认情况下，`ESP-IDF` 会禁用对 C++ 异常的支持，但是可以通过 `CONFIG_COMPILER_CXX_EXCEPTIONS` 选项启用。

通常情况下，启用异常处理会让应用程序的二进制文件增加几 kB。此外，启用该功能时还应为异常事故池预留一定内存。当应用程序无法从堆中分配异常对象时，就可以使用这个池中的内存。该内存池的大

小可以通过 `CONFIG_COMPILER_CXX_EXCEPTIONS_EMG_POOL_SIZE` 来设定。

如果 C++ 程序抛出了异常，但是程序中并没有 `catch` 代码块来捕获该异常，那么程序的运行就会被 `abort` 函数中止，然后打印回溯信息。有关回溯的更多信息，请参阅 [不可恢复错误](#)。

## 4.5 严重错误

### 4.5.1 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
  - Stack 粉碎保护检查
  - Heap 完整性检查
- 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。

### 4.5.2 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)
```

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_ESP32_PANIC` 的设置，支持的选项包括：

- 打印 CPU 寄存器，然后重启 (`CONFIG_ESP32_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_ESP32_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_ESP32_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_ESP32_PANIC_GDBSTUB`)  
启动 GDB 服务器，通过控制台 UART 接口与 GDB 进行通信。详细信息请参阅 [GDB Stub](#)。

紧急处理程序的行为还受到另外两个配置项的影响：

- 如果 `CONFIG_ESP32_DEBUG_OCDAWARE` 被使能了（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 *Core Dump* 功能 (`CONFIG_ESP32_ENABLE_COREDUMP_TO_FLASH` 或者 `CONFIG_ESP32_ENABLE_COREDUMP_TO_UART` 选项)，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。

下图展示了紧急处理程序的行为：

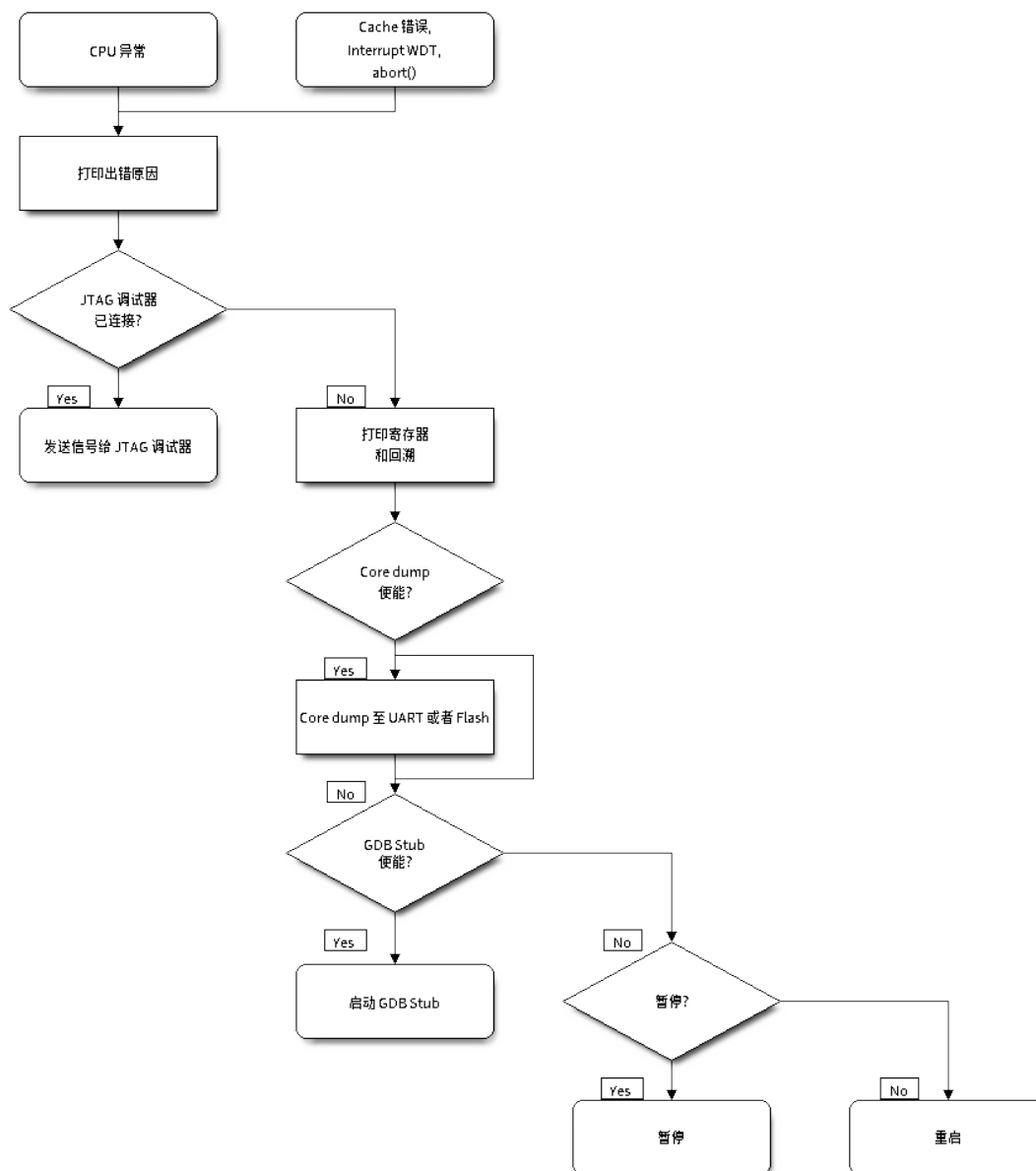


图 1: 紧急处理程序流程图 (点击放大)

### 4.5.3 寄存器转储与回溯

除非启用了 `CONFIG_ESP32_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：

```
Core 0 register dump:
PC      : 0x400e14ed  PS      : 0x00060030  A0      : 0x800d0805  A1      : _
↳0x3ffb5030
A2      : 0x00000000  A3      : 0x00000001  A4      : 0x00000001  A5      : _
↳0x3ffb50dc
A6      : 0x00000000  A7      : 0x00000001  A8      : 0x00000000  A9      : _
↳0x3ffb5000
A10     : 0x00000000  A11     : 0x3ffb2bac  A12     : 0x40082d1c  A13     : _
↳0x06ff1ff8
A14     : 0x3ffb7078  A15     : 0x00000000  SAR     : 0x00000014  EXCCAUSE:_
↳0x0000001d
EXCVADDR: 0x00000000  LBEG    : 0x400c46c  LEND    : 0x400c477  LCOUNT : _
↳0xffffffff

Backtrace: 0x400e14ed:0x3ffb5030 0x400d0802:0x3ffb5050
```

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。

紧急处理程序如果是因 `abort()` 而调用，则不会打印寄存器转储。

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：

```
Core 0 register dump:
PC      : 0x400e14ed  PS      : 0x00060030  A0      : 0x800d0805  A1      : _
↳0x3ffb5030
0x400e14ed: app_main at /Users/user/esp/example/main/main.cpp:36
A2      : 0x00000000  A3      : 0x00000001  A4      : 0x00000001  A5      : _
↳0x3ffb50dc
A6      : 0x00000000  A7      : 0x00000001  A8      : 0x00000000  A9      : _
↳0x3ffb5000
A10     : 0x00000000  A11     : 0x3ffb2bac  A12     : 0x40082d1c  A13     : _
↳0x06ff1ff8
0x40082d1c: _calloc_r at /Users/user/esp/esp-idf/components/newlib/syscalls.c:51
A14     : 0x3ffb7078  A15     : 0x00000000  SAR     : 0x00000014  EXCCAUSE:_
↳0x0000001d
EXCVADDR: 0x00000000  LBEG    : 0x400c46c  LEND    : 0x400c477  LCOUNT : _
↳0xffffffff

Backtrace: 0x400e14ed:0x3ffb5030 0x400d0802:0x3ffb5050
0x400e14ed: app_main at /Users/user/esp/example/main/main.cpp:36
0x400d0802: main_task at /Users/user/esp/esp-idf/components/esp32/cpu_start.c:470
```

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。

## 4.5.4 GDB Stub

如果启用了 `CONFIG_ESP32_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.
$T0b#e6GNU 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
0x400e1b41 in app_main ()
    at /Users/user/esp/example/main/main.cpp:36
36      *((int*) 0) = 0;
(gdb)
```

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。

## 4.5.5 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。

---

**注解：** 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 中加载下一条指令，这通常发生在：
  - 应用程序将 SPI Flash 的引脚重新配置为其它功能（如 GPIO，UART 等等）。有关 SPI Flash 引脚的详细信息，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 的引脚上，干扰了 ESP32 和 SPI Flash 之间的通信。

### InstrFetchProhibited

此 CPU 异常表示 CPU 无法加载指令，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指针。这种情况下，可以查看 PC（程序计数器）寄存器的值并做进一步判断：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正尝试解引用一个 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 - 0x6xxxxxx` 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 位加载指令只能访问 4 字节对齐的内存地址，而 16 位加载指令只能访问 2 字节对齐的内存地址。

### LoadStoreError

应用程序尝试从仅支持 32 位加载/存储的内存区域执行 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的 `char*` 指针就会导致这样的错误。

### 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金丝雀（stack canary）的位置访问堆栈，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](#)。

## 4.5.6 其它严重错误

### 欠压

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_ESP32_BROWNOUT_DET` 和 `CONFIG_ESP32_BROWNOUT_DET_LVL_SEL` 这两个选项进行设置。当掉电检测器被触发时，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。

### Heap 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。

### Stack 粉碎

Stack 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 Stack 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
0
```

回溯信息会指明发生 Stack 粉碎的函数，建议检查函数中是否有代码访问本地数组时发生了越界。

## 4.6 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.

### 4.6.1 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:

```
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

Event ID (legacy event ID)	Event data structure
<b>Wi-Fi</b>	
WIFI_EVENT_WIFI_READY (SYSTEM_EVENT_WIFI_READY)	n/a
WIFI_EVENT_SCAN_DONE (SYSTEM_EVENT_SCAN_DONE)	<a href="#">wifi_event_sta_scan_done_t</a>
WIFI_EVENT_STA_START (SYSTEM_EVENT_STA_START)	n/a
WIFI_EVENT_STA_STOP (SYSTEM_EVENT_STA_STOP)	n/a
WIFI_EVENT_STA_CONNECTED (SYSTEM_EVENT_STA_CONNECTED)	<a href="#">wifi_event_sta_connected_t</a>
WIFI_EVENT_STA_DISCONNECTED (SYSTEM_EVENT_STA_DISCONNECTED)	<a href="#">wifi_event_sta_disconnected_t</a>
WIFI_EVENT_STA_AUTHMODE_CHANGE (SYSTEM_EVENT_STA_AUTHMODE_CHANGE)	<a href="#">wifi_event_sta_authmode_change_t</a>
WIFI_EVENT_STA_WPS_ER_SUCCESS (SYSTEM_EVENT_STA_WPS_ER_SUCCESS)	n/a
WIFI_EVENT_STA_WPS_ER_FAILED (SYSTEM_EVENT_STA_WPS_ER_FAILED)	<a href="#">wifi_event_sta_wps_fail_reason_t</a>
WIFI_EVENT_STA_WPS_ER_TIMEOUT (SYSTEM_EVENT_STA_WPS_ER_TIMEOUT)	n/a
WIFI_EVENT_STA_WPS_ER_PIN (SYSTEM_EVENT_STA_WPS_ER_PIN)	<a href="#">wifi_event_sta_wps_er_pin_t</a>
WIFI_EVENT_AP_START (SYSTEM_EVENT_AP_START)	n/a
WIFI_EVENT_AP_STOP (SYSTEM_EVENT_AP_STOP)	n/a
WIFI_EVENT_AP_STACONNECTED (SYSTEM_EVENT_AP_STACONNECTED)	<a href="#">wifi_event_ap_staconnected_t</a>
WIFI_EVENT_AP_STADISCONNECTED (SYSTEM_EVENT_AP_STADISCONNECTED)	<a href="#">wifi_event_ap_stadisconnected_t</a>
WIFI_EVENT_AP_PROBEREQRCVD (SYSTEM_EVENT_AP_PROBEREQRCVD)	<a href="#">wifi_event_ap_probe_req_rx_t</a>
<b>Ethernet</b>	
ETHERNET_EVENT_START (SYSTEM_EVENT_ETH_START)	n/a
ETHERNET_EVENT_STOP (SYSTEM_EVENT_ETH_STOP)	n/a
ETHERNET_EVENT_CONNECTED (SYSTEM_EVENT_ETH_CONNECTED)	n/a
ETHERNET_EVENT_DISCONNECTED (SYSTEM_EVENT_ETH_DISCONNECTED)	n/a
<b>IP</b>	
IP_EVENT_STA_GOT_IP (SYSTEM_EVENT_STA_GOT_IP)	<a href="#">ip_event_got_ip_t</a>
IP_EVENT_STA_LOST_IP (SYSTEM_EVENT_STA_LOST_IP)	n/a
IP_EVENT_AP_STAIPASSIGNED (SYSTEM_EVENT_AP_STAIPASSIGNED)	n/a
IP_EVENT_GOT_IP6 (SYSTEM_EVENT_GOT_IP6)	<a href="#">ip_event_got_ip6_t</a>
IP_EVENT_ETH_GOT_IP (SYSTEM_EVENT_ETH_GOT_IP)	<a href="#">ip_event_got_ip_t</a>

### 4.6.2 Mesh Events

ESP-MESH uses a system similar to the [Legacy Event Loop](#) to deliver events to the application. See [System Events](#) for details.

### 4.6.3 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 parameters 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`.
- GATTC: `esp_ble_gattc_register_callback()`, `esp_bt_gattc_cb_event_t`,  
`esp_bt_gattc_cb_param_t`.
- GATTS: `esp_ble_gatts_register_callback()`, `esp_bt_gatts_cb_event_t`,  
`esp_bt_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_ct_cb_param_t`.
- HFP Client: `esp_hf_client_register_callback()`, `esp_hf_client_cb_event_t`,  
`esp_hf_client_cb_param_t`.

## 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_deeptime.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:

```
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:

```
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";
    ets_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 similar 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`:

```
int wake_count;

void RTC_IRAM_ATTR esp_wake_deep_sleep(void) {
    esp_default_wake_deep_sleep();
    ets_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.

## 4.8 ESP32 Core Dump

### 4.8.1 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. ESP-IDF provides special script `espcoredump.py` to help users to retrieve and analyse core dumps. This tool provides two commands for core dumps analysis:

- `info_corefile` - 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)
- `dbg_corefile` - 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

### 4.8.2 Configuration

There are a number of core dump related configuration options which user can choose in project configuration menu (`idf.py menuconfig`).

1. Core dump data destination (*Components -> ESP32-specific config -> Core dump -> Data destination*):
  - Disable core dump generation
  - Save core dump to flash
  - Print core dump to UART
2. Maximum number of tasks snapshots in core dump (*Components -> ESP32-specific config -> Core dump -> Maximum number of tasks*).
3. Delay before core dump is printed to UART (*Components -> ESP32-specific config -> Core dump -> Delay before print to UART*). Value is in ms.

### 4.8.3 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 user wants 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: `espcoredump.py -p </path/to/serial/port> info_corefile </path/to/program/elf/file>` or `espcoredump.py -p </path/to/serial/port> dbg_corefile </path/to/program/elf/file>`

## 4.8.4 Print core dump to UART

When this option is selected base64-encoded core dumps are printed on UART upon system panic. In this case user should save core dump text body to some file manually and then run the following command: `espcoredump.py info_corefile -t b64 -c </path/to/saved/base64/text> </path/to/program/elf/file>` or `espcoredump.py dbg_corefile -t b64 -c </path/to/saved/base64/text> </path/to/program/elf/file>`

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.

## 4.8.5 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 you can use ROM ELF provided by Espressif ([https://dl.espressif.com/dl/esp32\\_rom.elf](https://dl.espressif.com/dl/esp32_rom.elf)) and pass it to 'espcoredump.py'.

## 4.8.6 Running 'espcoredump.py'

Generic command syntax:

```
espcoredump.py [options] command [args]
```

### Script Options

- `-chip,-c {auto,esp32}`. Target chip type. Supported values are *auto* and *esp32*.
- `-port,-p PORT`. Serial port device.
- `-baud,-b BAUD`. Serial port baud rate used when flashing/reading.

### Commands

- `info_corefile`. Retrieve core dump and print useful info.
- `dbg_corefile`. Retrieve core dump and start GDB session with it.

### Command Arguments

- `-debug,-d DEBUG`. Log level (0..3).
- `-gdb,-g GDB`. Path to gdb to use for data retrieval.
- `-core,-c CORE`. Path to core dump file to use (if skipped core dump will be read from flash).
- `-core-format,-t CORE_FORMAT`. Specifies that file passed with `"-c"` is an ELF (`"elf"`), dumped raw binary (`"raw"`) or base64-encoded (`"b64"`) format.
- `-off,-o OFF`. Offset of coredump partition in flash (type *idf.py partition\_table* to see it).
- `-save-core,-s SAVE_CORE`. Save core to file. Otherwise temporary core file will be deleted. Ignored with `"-c"`.
- `-rom-elf,-r ROM_ELF`. Path to ROM ELF file to use (if skipped `"esp32_rom.elf"` is used).
- `-print-mem,-m` Print memory dump. Used only with `"info_corefile"`.

## 4.9 Flash Encryption

This document provides introduction to Flash encryption concept on ESP32 and demonstrates how this feature can be used during development as well as production by the user using a sample example. The primary intention of the document is to act as a quick start guide to test and verify flash encryption operations. The details of the flash encryption block can be found in the [ESP32 Technical reference manual](#).



## 4.9.1 Introduction

Flash encryption is a feature for encrypting the contents of the ESP32's attached SPI flash. When flash encryption is enabled, physical readout of the SPI flash is not sufficient to recover most flash contents. Encryption is applied by flashing the ESP32 with plaintext data, and (if encryption is enabled) the bootloader encrypts the data in place on first boot.

With flash encryption enabled, following kinds of flash data are encrypted by default:

- Bootloader
- Partition Table
- All “app” type partitions

Other type of flash data are encrypted conditionally:

- Secure boot bootloader digest (if secure boot is enabled)
- Any partition marked with the “encrypted” flag in the partition table

Flash encryption is separate from the *Secure Boot* feature, and you can use flash encryption without enabling secure boot. However, for a secure environment both should be used simultaneously.

---

**重要:** For production use, flash encryption should be enabled in the “Release” mode only.

---



---

**重要:** Enabling flash encryption limits the options for further updates of the ESP32. Make sure to read this document (including *Limitations of Flash Encryption*) and understand the implications of enabling flash encryption.

---

## 4.9.2 eFuse Used During Flash Encryption Process

The flash encryption operation is controlled by various eFuses available on ESP32. Below is the list of eFuse and their description:

eFuse	Description	Can be
locked for writing	Default Value	reading/
Coding scheme	This 2 bit wide eFuse controls the actual number of bits to be used from BLOCK1 to derive final 256 bit AES key. The coding scheme value is decoded as below: 0: 256 bits 1: 192 bits 2: 128 bits Final AES key is derived based on the FLASH_CRYPT_CONFIG value	Yes
BLOCK1	256 bit wide eFuse block for storing AES key	Yes
FLASH_CRYPT_CONFIG	4 bit wide eFuse which controls the AES encryption process	Yes
download_dis_encrypt	When set, disables the flash encryption	Yes

(下页继续)

	operation while running in UART download mode		
download_dis_decrypt ↔ 0	When set, disables the flash decryption	Yes	↔
	operation while running in UART download mode		
FLASH_CRYPT_CNT ↔ 0	7 bit eFuse which enables/disables encryption at boot time	Yes	↔
	Even number of bits set (0, 2, 4, 6): encrypt flash at boot time Odd number of bits set (1, 3, 5, 7): do not encrypt flash at boot time		

Read and write access to above bits is controlled by appropriate bits in `efuse_wr_disable` and `efuse_rd_disable` registers. More information about ESP32 eFuse can be found at [eFuse manager](#).

### 4.9.3 Flash Encryption Process

Assuming the eFuse values are in default state and second stage bootloader is compiled to support flash encryption, the flash encryption process executes as below:

- On first power-on reset, all data in flash is un-encrypted (plaintext). First stage loader (ROM) will load the second stage loader in IRAM.
- Second stage bootloader will read the `flash_crypt_cnt` (=00000000b) eFuse value and since the value is 0 (even number of bits set) it will configure and enable the flash encryption block. It will also program `FLASH_CRYPT_CFG` eFuse to value 0xF.
- The flash encryption block will generate AES-256 bit key and store into `BLOCK1` eFuse. This operation is performed in hardware and the key can not be accessed by software.
- Next the flash encryption block will encrypt the flash contents (based on partition table flag value). Encrypting in-place can take some time (up to a minute for large partitions).
- Second stage bootloader then sets the first available bit in `flash_crypt_cnt` (=00000001b) to mark the flash contents as encrypted (odd number of bits set).
- For *Release Mode* second stage bootloader will program `download_dis_encrypt`, `download_dis_decrypt` & `download_dis_cache` eFuse bits to 1 to prevent UART bootloader from decrypting the flash contents. It will also write protect the `FLASH_CRYPT_CNT` eFuse bits.
- For *Development Mode* second stage bootloader will program only `download_dis_decrypt` & `download_dis_cache` eFuse bits to allow UART bootloader reflashing of encrypted binaries. Also `FLASH_CRYPT_CNT` eFuse bits will NOT be write protected.
- The second stage bootloader then reboots the device to start executing encrypted image. It will transparently decrypt the flash contents and load into IRAM.

During development stage there is a frequent need to program different plaintext flash images and test the flash encryption process. This requires UART download mode to be able to load new plaintext images as many number of times as required. However during manufacturing or production UART download mode should not be allowed to access flash contents due to security reason. Hence this requires two different ESP32 configurations: one for development and other for production. Following section describes *Development Mode* and *Release Mode* for flash encryption and a step by step process to use them.

---

**重要:** Development mode as the name suggests should be used ONLY DURING DEVELOPMENT as it does not prevent modification and possible read back of encrypted flash contents.

---

## 4.9.4 Steps to Setup Flash Encryption

### Development Mode

It is possible to run flash encryption process for development using either ESP32 internally generated key or external host generated keys.

### Using ESP32 Generated Flash Encryption Key

As mentioned above *Development Mode* allows user to download as many plaintext images using UART download mode. Following steps needs to be done to test flash encryption process:

- Ensure you have a ESP32 device with default flash encryption eFuse settings as shown in *eFuse Used During Flash Encryption Process*.
- Navigate to flash encryption sample application in \$IDF\_PATH/examples/security/flash\_encryption folder. This sample application will print the status of flash encryption: enabled or disabled. It will print the FLASH\_CRYPT\_CNT eFuse value.
- Enable flash encryption support in second stage bootloader. In *Project Configuration Menu*, navigate to “Security Features” .
- Select *Enable flash encryption on boot*.
- By default the mode is set for **Development**.
- Select appropriate Bootloader log verbosity under Bootloader config.
- Update to the partition table offset may be required since after enabling flash encryption the size of bootloader is increased. See *Bootloader Size*
- Save the configuration and exit.

Build and flash the complete image including: bootloader, partition table and app. These partitions are initially written to the flash unencrypted.

```
idf.py flash monitor
```

Once the flashing is complete device will reset and on next boot second stage bootloader will encrypt the flash app partition and then reset. Now the sample application would get decrypted at runtime and executed. Below is a sample output when ESP32 boots after flash encryption is enabled for the first time.

```
--- 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)
configsip: 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:13608
load:0x40080400,len:6664
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: ## Label                Usage              Type ST Offset   Length
I (58) boot:  0 nvs                    WiFi data          01 02 0000a000 00006000
I (66) boot:  1 phy_init                 RF data            01 01 00010000 00001000
I (73) boot:  2 factory                 factory app        00 00 00020000 00100000
I (81) boot: End of partition table
```

(下页继续)

```
I (85) esp_image: segment 0: paddr=0x00020020 vaddr=0x3f400020_
↳size=0x0808c ( 32908) map
I (105) esp_image: segment 1: paddr=0x000280b4 vaddr=0x3ffb0000_
↳size=0x01ea4 ( 7844) load
I (109) 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 (114) esp_image: segment 3: paddr=0x0002a368 vaddr=0x40080400_
↳size=0x05ca8 ( 23720) load
I (132) esp_image: segment 4: paddr=0x00030018 vaddr=0x400d0018_
↳size=0x126a8 ( 75432) map
0x400d0018: _flash_cache_start at ???

I (159) esp_image: segment 5: paddr=0x000426c8 vaddr=0x400860a8_
↳size=0x01f4c ( 8012) load
0x400860a8: prvAddNewTaskToReadyList at esp-idf/esp-idf/components/
↳freertos/tasks.c:4561

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 (187) 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 ( 4)
I (227) esp_image: segment 1: paddr=0x0000102c vaddr=0x3fff001c_
↳size=0x02104 ( 8452)
I (239) esp_image: segment 2: paddr=0x00003138 vaddr=0x40078000_
↳size=0x03528 ( 13608)
I (249) esp_image: segment 3: paddr=0x00006668 vaddr=0x40080400_
↳size=0x01a08 ( 6664)
I (657) esp_image: segment 0: paddr=0x00020020 vaddr=0x3f400020_
↳size=0x0808c ( 32908) map
I (669) esp_image: segment 1: paddr=0x000280b4 vaddr=0x3ffb0000_
↳size=0x01ea4 ( 7844)
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...
```

Once the flash encryption is enabled, on subsequent boot the output would mention that flash encryption is already enabled.

```

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 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
ho 0 tail 12 room 4
load:0x40080400,len:6664
entry 0x40080764
I (30) boot: ESP-IDF v4.0-dev-850-gc4447462d-dirty 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 00001000
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=0x3ffb0000
↳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
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.

```

(下页继续)

(续上页)

```

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_crypt_cnt = 1
Halting...

```

At this stage if user wants to update modified plaintext application image to flash in encrypted format it can be done using following command:

```
idf.py encrypted-app-flash monitor
```

### Encrypt Multiple Partitions

If all partitions needs to be updated in encrypted format, it can be done as

```
idf.py encrypted-flash monitor
```

### Using Host Generated Flash Encryption Key

It is possible to pregenerate the flash encryption key on the host computer and burn it into the ESP32's eFuse key block. This allows data to be pre-encrypted on the host and flashed to the ESP32 without needing a plaintext flash update. This feature allows encrypted flashing in both *Development Mode* and *Release Mode* modes.

**注解:** This option is not recommended for production unless a separate key is generated for each individual device.

- Ensure you have a ESP32 device with default flash encryption eFuse settings as shown in *eFuse Used During Flash Encryption Process*.
- Generate a random key with espsecure.py:

```
espsecure.py generate_flash_encryption_key my_flash_encryption_key.bin
```

- Burn the key to the device (one time only). **This must be done before first encrypted boot**, otherwise the ESP32 will generate a random key that software can't access or modify:

```
espefuse.py --port PORT burn_key flash_encryption my_flash_encryption_key.bin
```

- Enable flash encryption support in second stage bootloader. In *Project Configuration Menu*, navigate to “Security Features” .
- Select *Enable flash encryption on boot*.
- By default the mode is set for **Development**.
- Select appropriate Bootloader log verbosity under Bootloader config.
- Update to the partition table offset may be required since after enabling flash encryption the size of bootloader is increased. See *Bootloader Size*
- Save the configuration and exit.

Build and flash the complete image including: bootloader, partition table and app. These partitions are initially written to the flash unencrypted

```
idf.py flash monitor
```

On next boot second stage bootloader will encrypt the flash app partition and then reset. Now the sample application would get decrypted at runtime and executed.

At this stage if user wants to update new plaintext application image to flash they should issue following command

```
idf.py encrypted-app-flash monitor
```

For reprogramming all partitions in encrypted format follow [Encrypt Multiple Partitions](#).

## Release Mode

In Release mode UART bootloader can not perform flash encryption operations and new plaintext images can be downloaded ONLY using OTA scheme which will encrypt the plaintext image before writing to flash.

- Ensure you have a ESP32 device with default flash encryption eFuse settings as shown in [eFuse Used During Flash Encryption Process](#).
- Enable flash encryption support in second stage bootloader. In [Project Configuration Menu](#), navigate to “Security Features” .
- Select [Enable flash encryption on boot](#).
- Select **Release Mode**, by default the mode is set for **Development**. Please note **once the Release mode is selected the “download\_dis\_encrypt“ and “download\_dis\_decrypt“ eFuse bits will be programmed to disable UART bootloader access to flash contents.**
- Select appropriate Bootloader log verbosity under Bootloader config.
- Update to the partition table offset may be required since after enabling flash encryption the size of bootloader is increased. See [Bootloader Size](#)
- Save the configuration and exit.

Build and flash the complete image including: bootloader, partition table and app. These partitions are initially written to the flash unencrypted

```
idf.py flash monitor
```

On next boot second stage bootloader will encrypt the flash app partition and then reset. Now the sample application should execute correctly.

Once the flash encryption is enabled in Release mode the bootloader will write protect the FLASH\_CRYPT\_CNT eFuse.

For subsequent plaintext update in field OTA scheme should be used. Please refer [OTA](#) for further details.

## Possible Failures

Once flash encryption is enabled and if the FLASH\_CRYPT\_CNT eFuse value has an odd number of bits set then all the partitions (which are marked with encryption flag) are expected to contain encrypted ciphertext. Below are three typical failure cases if the ESP32 is loaded with plaintext data:

1. In case the bootloader partition is re-updated with plaintext bootloader image the ROM loader will fail to load the bootloader and following type of failure will be displayed:

```
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 (TGWDT_SYS_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
```

(下页继续)

(续上页)

```

flash read err, 1000
ets_main.c 371
ets Jun  8 2016 00:22:57

rst:0x7 (TGOWDT_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 (TGOWDT_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 (TGOWDT_SYS_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
flash read err, 1000
ets_main.c 371
ets Jun  8 2016 00:22:57

```

**注解:** This error also appears in the flash contents is erased or corrupted.

2. In case the bootloader is encrypted but partition table is re-updated with plaintext partition table image the bootloader will fail to read the partition table and following type of failure will be displayed:

```

rst:0x3 (SW_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 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:10464
ho 0 tail 12 room 4
load:0x40078000,len:19168
load:0x40080400,len:6664
entry 0x40080764
I (60) boot: ESP-IDF v4.0-dev-763-g2c55fae6c-dirty 2nd stage bootloader
I (60) boot: compile time 19:15:54
I (62) boot: Enabling RNG early entropy source...
I (67) boot: SPI Speed      : 40MHz
I (72) boot: SPI Mode      : DIO
I (76) boot: SPI Flash Size : 4MB
E (80) flash_parts: partition 0 invalid magic number 0x94f6
E (86) boot: Failed to verify partition table
E (91) boot: load partition table error!

```

3. In case the bootloader & partition table are encrypted but application is re-updated with plaintext application image the bootloader will fail load the new application and following type of failure will be displayed:

```

rst:0x3 (SW_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 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:13616
load:0x40080400,len:6664
entry 0x40080764
I (56) boot: ESP-IDF v4.0-dev-850-gc4447462d-dirty 2nd stage bootloader
I (56) boot: compile time 15:37:14
I (58) boot: Enabling RNG early entropy source...

```

(下页继续)



(续上页)

```

I (64) boot: SPI Speed      : 40MHz
I (68) boot: SPI Mode      : DIO
I (72) boot: SPI Flash Size : 4MB
I (76) boot: Partition Table:
I (79) boot: ## Label           Usage             Type ST Offset   Length
I (87) boot:  0 nvs             WiFi data        01 02 0000a000 00006000
I (94) boot:  1 phy_init        RF data          01 01 00010000 00001000
I (102) boot:  2 factory        factory app      00 00 00020000 00100000
I (109) boot: End of partition table
E (113) esp_image: image at 0x20000 has invalid magic byte
W (120) esp_image: image at 0x20000 has invalid SPI mode 108
W (126) esp_image: image at 0x20000 has invalid SPI size 11
E (132) boot: Factory app partition is not bootable
E (138) boot: No bootable app partitions in the partition table

```

## 4.9.5 Key Points About Flash Encryption

- The contents of the flash are encrypted using AES-256. The flash encryption key is stored in eFuse internal to the chip, and is (by default) protected from software access.
- The *flash encryption algorithm* is AES-256, where the key is “tweaked” with the offset address of each 32 byte block of flash. This means every 32 byte block (two consecutive 16 byte AES blocks) is encrypted with a unique key derived from the flash encryption key.
- Flash access is transparent via the flash cache mapping feature of ESP32 - any flash regions which are mapped to the address space will be transparently decrypted when read.
  - It may be desirable for some data partitions to remain unencrypted for ease of access, or to use flash-friendly update algorithms that are ineffective if the data is encrypted. NVS partitions for non-volatile storage cannot be encrypted since NVS library is not directly compatible with flash encryption. Refer to [NVS Encryption](#) for more details.
- If flash encryption may be enabled, the programmer must take certain precautions when writing code that *uses encrypted flash*.
- If secure boot is enabled, reflashing the bootloader of an encrypted device requires a “Reflashable” secure boot digest (see [Flash Encryption and Secure Boot](#)).

---

**注解:** The bootloader app binary `bootloader.bin` may become too large when both secure boot and flash encryption are enabled. See [Bootloader Size](#).

---

**重要:** Do not interrupt power to the ESP32 while the first boot encryption pass is running. If power is interrupted, the flash contents will be corrupted and require flashing with unencrypted data again. A reflash like this will not count towards the flashing limit.

---

## 4.9.6 Using Encrypted Flash

ESP32 app code can check if flash encryption is currently enabled by calling `esp_flash_encryption_enabled()`. Also, device can identify the flash encryption mode by calling `esp_get_flash_encryption_mode()`.

Once flash encryption is enabled, some care needs to be taken when accessing flash contents from code.

### Scope of Flash Encryption

Whenever the `FLASH_CRYPT_CNT` eFuse is set to a value with an odd number of bits set, all flash content which is accessed via the MMU’s flash cache is transparently decrypted. This includes:

- Executable application code in flash (IROM).
- All read-only data stored in flash (DROM).
- Any data accessed via `spi_flash_mmap()`.
- The software bootloader image when it is read by the ROM bootloader.

---

**重要:** The MMU flash cache unconditionally decrypts all data. Data which is stored unencrypted in the flash will be “transparently decrypted” via the flash cache and appear to software like random garbage.

---

### Reading Encrypted Flash

To read data without using a flash cache MMU mapping, we recommend using the partition read function `esp_partition_read()`. When using this function, data will only be decrypted when it is read from an encrypted partition. Other partitions will be read unencrypted. In this way, software can access encrypted and non-encrypted flash in the same way.

Data which is read via other SPI read APIs are not decrypted:

- Data read via `spi_flash_read()` is not decrypted.
- Data read via ROM function `SPIRead()` is not decrypted (this function is not supported in esp-idf apps).
- Data stored using the Non-Volatile Storage (NVS) API is always stored and read decrypted from the perspective of flash encryption. It is up to the library to provide encryption feature if required. Refer to [NVS Encryption](#) for more details.

### Writing Encrypted Flash

Where possible, we recommend using the partition write function `esp_partition_write`. When using this function, data will only be encrypted when writing to encrypted partitions. Data will be written to other partitions unencrypted. In this way, software can access encrypted and non-encrypted flash in the same way.

The `esp_spi_flash_write` function will write data when the `write_encrypted` parameter is set to true. Otherwise, data will be written unencrypted.

The ROM function `esp_rom_spiflash_write_encrypted` will write encrypted data to flash, the ROM function `SPIWrite` will write unencrypted to flash. (these function are not supported in esp-idf apps).

Because data is encrypted in blocks, the minimum write size for encrypted data is 16 bytes (and the alignment is 16 bytes).

## 4.9.7 Updating Encrypted Flash

### OTA Updates

OTA updates to encrypted partitions will automatically write encrypted, as long as the `esp_partition_write` function is used.

## 4.9.8 Disabling Flash Encryption

If you’ve accidentally enabled flash encryption for some reason, the next flash of plaintext data will soft-brick the ESP32 (the device will reboot continuously, printing the error `flash read err, 1000`).

If flash encryption is enabled in Development mode, you can disable flash encryption again by writing `FLASH_CRYPT_CNT` eFuse. This can only be done three times per chip.

- First, open [Project Configuration Menu](#) and disable [Enable flash encryption boot](#) under “Security Features” .
- Exit `menuconfig` and save the new configuration.
- Run `idf.py menuconfig` again and double-check you really disabled this option! *If this option is left enabled, the bootloader will immediately re-enable encryption when it boots.*

- Run `idf.py flash` to build and flash a new bootloader and app, without flash encryption enabled.
- Run `espefuse.py` (in `components/esptool_py/esptool`) to disable the `FLASH_CRYPT_CNT::espefuse.py burn_efuse FLASH_CRYPT_CNT`

Reset the ESP32 and flash encryption should be disabled, the bootloader will boot as normal.

### 4.9.9 Limitations of Flash Encryption

Flash encryption prevents plaintext readout of the encrypted flash, to protect firmware against unauthorised readout and modification. It is important to understand the limitations of the flash encryption system:

- Flash encryption is only as strong as the key. For this reason, we recommend keys are generated on the device during first boot (default behaviour). If generating keys off-device, ensure proper procedure is followed and don't share the same key between all production devices.
- Not all data is stored encrypted. If storing data on flash, check if the method you are using (library, API, etc.) supports flash encryption.
- Flash encryption does not prevent an attacker from understanding the high-level layout of the flash. This is because the same AES key is used for every pair of adjacent 16 byte AES blocks. When these adjacent 16 byte blocks contain identical content (such as empty or padding areas), these blocks will encrypt to produce matching pairs of encrypted blocks. This may allow an attacker to make high-level comparisons between encrypted devices (i.e. to tell if two devices are probably running the same firmware version).
- For the same reason, an attacker can always tell when a pair of adjacent 16 byte blocks (32 byte aligned) contain two identical 16 byte sequences. Keep this in mind if storing sensitive data on the flash, design your flash storage so this doesn't happen (using a counter byte or some other non-identical value every 16 bytes is sufficient). *NVS Encryption* deals with this and is suitable for many uses.
- Flash encryption alone may not prevent an attacker from modifying the firmware of the device. To prevent unauthorised firmware from running on the device, use flash encryption in combination with *Secure Boot*.

### 4.9.10 Flash Encryption and Secure Boot

It is recommended to use flash encryption and secure boot together. However, if Secure Boot is enabled then additional restrictions apply to reflashing the device:

- *OTA Updates* are not restricted (provided the new app is signed correctly with the Secure Boot signing key).
- *Plaintext serial flash updates* are only possible if the *Reflashable* Secure Boot mode is selected and a Secure Boot key was pre-generated and burned to the ESP32 (refer to *Secure Boot* docs.). In this configuration, `idf.py bootloader` will produce a pre-digested bootloader and secure boot digest file for flashing at offset 0x0. When following the plaintext serial reflashing steps it is necessary to re-flash this file before flashing other plaintext data.
- *Reflashing via Pregenerated Flash Encryption Key* is still possible, provided the bootloader is not reflashed. Reflashing the bootloader requires the same *Reflashable* option to be enabled in the Secure Boot config.

### 4.9.11 Advanced Features

The following information is useful for advanced use of flash encryption:

#### Encrypted Partition Flag

Some partitions are encrypted by default. Otherwise, it is possible to mark any partition as requiring encryption:

In the *partition table* description CSV files, there is a field for flags.

Usually left blank, if you write “encrypted” in this field then the partition will be marked as encrypted in the partition table, and data written here will be treated as encrypted (same as an app partition):

```
# Name, Type, SubType, Offset, Size, Flags
nvs, data, nvs, 0x9000, 0x6000
phy_init, data, phy, 0xf000, 0x1000
factory, app, factory, 0x10000, 1M
secret_data, 0x40, 0x01, 0x20000, 256K, encrypted
```

- None of the default partition tables include any encrypted data partitions.
- It is not necessary to mark “app” partitions as encrypted, they are always treated as encrypted.
- The “encrypted” flag does nothing if flash encryption is not enabled.
- It is possible to mark the optional phy partition with phy\_init data as encrypted, if you wish to protect this data from physical access readout or modification.
- It is not possible to mark the nvs partition as encrypted.

### Enabling UART Bootloader Encryption/Decryption

By default, on first boot the flash encryption process will burn eFuses `DISABLE_DL_ENCRYPT`, `DISABLE_DL_DECRYPT` and `DISABLE_DL_CACHE`:

- `DISABLE_DL_ENCRYPT` disables the flash encryption operations when running in UART bootloader boot mode.
- `DISABLE_DL_DECRYPT` disables transparent flash decryption when running in UART bootloader mode, even if `FLASH_CRYPT_CNT` is set to enable it in normal operation.
- `DISABLE_DL_CACHE` disables the entire MMU flash cache when running in UART bootloader mode.

It is possible to burn only some of these eFuses, and write-protect the rest (with unset value 0) before the first boot, in order to preserve them. For example:

```
espefuse.py --port PORT burn_efuse DISABLE_DL_DECRYPT
espefuse.py --port PORT write_protect_efuse DISABLE_DL_ENCRYPT
```

(Note that all 3 of these eFuses are disabled via one write protect bit, so write protecting one will write protect all of them. For this reason, it’s necessary to set any bits before write-protecting.)

---

**重要:** Write protecting these eFuses to keep them unset is not currently very useful, as `esptool.py` does not support reading encrypted flash.

---



---

**重要:** If `DISABLE_DL_DECRYPT` is left unset (0) this effectively makes flash encryption useless, as an attacker with physical access can use UART bootloader mode (with custom stub code) to read out the flash contents.

---

### Setting FLASH\_CRYPT\_CONFIG

The `FLASH_CRYPT_CONFIG` eFuse determines the number of bits in the flash encryption key which are “tweaked” with the block offset. See [Flash Encryption Algorithm](#) for details.

First boot of the bootloader always sets this value to the maximum `0xF`.

It is possible to write these eFuse manually, and write protect it before first boot in order to select different tweak values. This is not recommended.

It is strongly recommended to never write protect `FLASH_CRYPT_CONFIG` when it the value is zero. If this eFuse is set to zero, no bits in the flash encryption key are tweaked and the flash encryption algorithm is equivalent to AES ECB mode.

## JTAG Debugging

By default, when Flash Encryption is enabled (in either Development or Release mode) then JTAG debugging is disabled via eFuse. The bootloader does this on first boot, at the same time it enables flash encryption.

See [JTAG with Flash Encryption or Secure Boot](#) for more information about using JTAG Debugging with Flash Encryption.

### 4.9.12 Technical Details

The following sections provide some reference information about the operation of flash encryption.

#### Flash Encryption Algorithm

- AES-256 operates on 16 byte blocks of data. The flash encryption engine encrypts and decrypts data in 32 byte blocks, two AES blocks in series.
- The main flash encryption key is stored in eFuse (BLOCK1) and by default is protected from further writes or software readout.
- AES-256 key size is 256 bits (32 bytes), read from eFuse block 1. The hardware AES engine uses the key in reversed byte order to the order stored in the eFuse block. - If CODING\_SCHEME eFuse is set to 0 (default “None” Coding Scheme) then the eFuse key block is 256 bits and the key is stored as-is (in reversed byte order). - If CODING\_SCHEME eFuse is set to 1 (3/4 Encoding) then the eFuse key block is 192 bits (in reversed byte order), so overall entropy is reduced. The hardware flash encryption still operates on a 256-bit key, after being read (and un-reversed), the key is extended by as  $key = key[0:255] + key[64:127]$ .
- AES algorithm is used inverted in flash encryption, so the flash encryption “encrypt” operation is AES decrypt and the “decrypt” operation is AES encrypt. This is for performance reasons and does not alter the effectiveness of the algorithm.
- Each 32 byte block (two adjacent 16 byte AES blocks) is encrypted with a unique key. The key is derived from the main flash encryption key in eFuse, XORed with the offset of this block in the flash (a “key tweak” ).
- The specific tweak depends on the setting of FLASH\_CRYPT\_CONFIG eFuse. This is a 4 bit eFuse, where each bit enables XORing of a particular range of the key bits:
  - Bit 1, bits 0-66 of the key are XORed.
  - Bit 2, bits 67-131 of the key are XORed.
  - Bit 3, bits 132-194 of the key are XORed.
  - Bit 4, bits 195-256 of the key are XORed.

It is recommended that FLASH\_CRYPT\_CONFIG is always left to set the default value *0xF*, so that all key bits are XORed with the block offset. See [Setting FLASH\\_CRYPT\\_CONFIG](#) for details.

- The high 19 bits of the block offset (bit 5 to bit 23) are XORed with the main flash encryption key. This range is chosen for two reasons: the maximum flash size is 16MB (24 bits), and each block is 32 bytes so the least significant 5 bits are always zero.
- There is a particular mapping from each of the 19 block offset bits to the 256 bits of the flash encryption key, to determine which bit is XORed with which. See the variable `_FLASH_ENCRYPTION_TWEAK_PATTERN` in the `espsecure.py` source code for the complete mapping.
- To see the full flash encryption algorithm implemented in Python, refer to the `_flash_encryption_operation()` function in the `espsecure.py` source code.

## 4.10 ESP-IDF FreeRTOS SMP Changes

### 4.10.1 Overview

The vanilla FreeRTOS is designed to run on a single core. However the ESP32 is dual core containing a Protocol CPU (known as **CPU 0** or **PRO\_CPU**) and an Application CPU (known as **CPU 1** or **APP\_CPU**). The two cores are identical in practice and share the same memory. This allows the two cores to run tasks interchangeably between them.

The ESP-IDF FreeRTOS is a modified version of vanilla FreeRTOS which supports symmetric multiprocessing (SMP). ESP-IDF FreeRTOS is based on the Xtensa port of FreeRTOS v8.2.0. This guide outlines the major differences between vanilla FreeRTOS and ESP-IDF FreeRTOS. The API reference for vanilla FreeRTOS can be found via <http://www.freertos.org/a00106.html>

For information regarding features that are exclusive to ESP-IDF FreeRTOS, see *ESP-IDF FreeRTOS Additions*.

*Backported Features:* Although ESP-IDF FreeRTOS is based on the Xtensa port of FreeRTOS v8.2.0, a number of FreeRTOS v9.0.0 features have been backported to ESP-IDF.

*Tasks and Task Creation:* Use `xTaskCreatePinnedToCore()` or `xTaskCreateStaticPinnedToCore()` to create tasks in ESP-IDF FreeRTOS. The last parameter of the two functions is `xCoreID`. This parameter specifies which core the task is pinned to. Acceptable values are 0 for **PRO\_CPU**, 1 for **APP\_CPU**, or `tskNO_AFFINITY` which allows the task to run on both.

*Round Robin Scheduling:* The ESP-IDF FreeRTOS scheduler will skip tasks when implementing Round-Robin scheduling between multiple tasks in the Ready state that are of the same priority. To avoid this behavior, ensure that those tasks either enter a blocked state, or are distributed across a wider range of priorities.

*Scheduler Suspension:* Suspending the scheduler in ESP-IDF FreeRTOS will only affect the scheduler on the the calling core. In other words, calling `vTaskSuspendAll()` on **PRO\_CPU** will not prevent **APP\_CPU** from scheduling, and vice versa. Use critical sections or semaphores instead for simultaneous access protection.

*Tick Interrupt Synchronicity:* Tick interrupts of **PRO\_CPU** and **APP\_CPU** are not synchronized. Do not expect to use `vTaskDelay()` or `vTaskDelayUntil()` as an accurate method of synchronizing task execution between the two cores. Use a counting semaphore instead as their context switches are not tied to tick interrupts due to preemption.

*Critical Sections & Disabling Interrupts:* In ESP-IDF FreeRTOS, critical sections are implemented using mutexes. Entering critical sections involve taking a mutex, then disabling the scheduler and interrupts of the calling core. However the other core is left unaffected. If the other core attempts to take same mutex, it will spin until the calling core has released the mutex by exiting the critical section.

*Floating Point Arithmetic:* The ESP32 supports hardware acceleration of single precision floating point arithmetic (`float`). However the use of hardware acceleration leads to some behavioral restrictions in ESP-IDF FreeRTOS. Therefore, tasks that utilize `float` will automatically be pinned to a core if not done so already. Furthermore, `float` cannot be used in interrupt service routines.

*Task Deletion:* Task deletion behavior has been backported from FreeRTOS v9.0.0 and modified to be SMP compatible. Task memory will be freed immediately when `vTaskDelete()` is called to delete a task that is not currently running and not pinned to the other core. Otherwise, freeing of task memory will still be delegated to the Idle Task.

*Thread Local Storage Pointers & Deletion Callbacks:* ESP-IDF FreeRTOS has backported the Thread Local Storage Pointers (TLSP) feature. However the extra feature of Deletion Callbacks has been added. Deletion callbacks are called automatically during task deletion and are used to free memory pointed to by TLSP. Call `vTaskSetThreadLocalStoragePointerAndDelCallback()` to set TLSP and Deletion Callbacks.

*Configuring ESP-IDF FreeRTOS:* Several aspects of ESP-IDF FreeRTOS can be set in the project configuration (`idf.py menuconfig`) such as running ESP-IDF in Unicore (single core) Mode, or configuring the number of Thread Local Storage Pointers each task will have.

## 4.10.2 Backported Features

The following features have been backported from FreeRTOS v9.0.0 to ESP-IDF.

### Static Allocation

This feature has been backported from FreeRTOS v9.0.0 to ESP-IDF. The `CONFIG_FREERTOS_SUPPORT_STATIC_ALLOCATION` option must be enabled in `menuconfig` in order for static allocation functions to be available. Once enabled, the following functions can be called...

- `xTaskCreateStatic()` (see *Backporting Notes* below)

- `xQueueCreateStatic`
- `xSemaphoreCreateBinaryStatic`
- `xSemaphoreCreateCountingStatic`
- `xSemaphoreCreateMutexStatic`
- `xSemaphoreCreateRecursiveMutexStatic`
- `xTimerCreateStatic()` (see *Backporting Notes* below)
- `xEventGroupCreateStatic()`

### Other Features

- `vTaskSetThreadLocalStoragePointer()` (see *Backporting Notes* below)
- `pvTaskGetThreadLocalStoragePointer()` (see *Backporting Notes* below)
- `vTimerSetTimerID()`
- `xTimerGetPeriod()`
- `xTimerGetExpiryTime()`
- `pcQueueGetName()`
- `uxSemaphoreGetCount`

### Backporting Notes

- 1) `xTaskCreateStatic()` has been made SMP compatible in a similar fashion to `xTaskCreate()` (see *Tasks and Task Creation*). Therefore `xTaskCreateStaticPinnedToCore()` can also be called.
- 2) Although vanilla FreeRTOS allows the Timer feature's daemon task to be statically allocated, the daemon task is always dynamically allocated in ESP-IDF. Therefore `vApplicationGetTimerTaskMemory` **does not** need to be defined when using statically allocated timers in ESP-IDF FreeRTOS.
- 3) The Thread Local Storage Pointer feature has been modified in ESP-IDF FreeRTOS to include Deletion Callbacks (see *Thread Local Storage Pointers & Deletion Callbacks*). Therefore the function `vTaskSetThreadLocalStoragePointerAndDelCallback()` can also be called.

## 4.10.3 Tasks and Task Creation

Tasks in ESP-IDF FreeRTOS are designed to run on a particular core, therefore two new task creation functions have been added to ESP-IDF FreeRTOS by appending `PinnedToCore` to the names of the task creation functions in vanilla FreeRTOS. The vanilla FreeRTOS functions of `xTaskCreate()` and `xTaskCreateStatic()` have led to the addition of `xTaskCreatePinnedToCore()` and `xTaskCreateStaticPinnedToCore()` in ESP-IDF FreeRTOS (see *Backported Features*).

For more details see [freertos/task.c](#)

The ESP-IDF FreeRTOS task creation functions are nearly identical to their vanilla counterparts with the exception of the extra parameter known as `xCoreID`. This parameter specifies the core on which the task should run on and can be one of the following values.

- 0 pins the task to **PRO\_CPU**
- 1 pins the task to **APP\_CPU**
- `tskNO_AFFINITY` allows the task to be run on both CPUs

For example `xTaskCreatePinnedToCore(tsk_callback, "APP_CPU Task", 1000, NULL, 10, NULL, 1)` creates a task of priority 10 that is pinned to **APP\_CPU** with a stack size of 1000 bytes. It should be noted that the `uxStackDepth` parameter in vanilla FreeRTOS specifies a task's stack depth in terms of the number of words, whereas ESP-IDF FreeRTOS specifies the stack depth in terms of bytes.

Note that the vanilla FreeRTOS functions `xTaskCreate()` and `xTaskCreateStatic()` have been defined in ESP-IDF FreeRTOS as inline functions which call `xTaskCreatePinnedToCore()` and `xTaskCreateStaticPinnedToCore()` respectively with `tskNO_AFFINITY` as the `xCoreID` value.

Each Task Control Block (TCB) in ESP-IDF stores the `xCoreID` as a member. Hence when each core calls the scheduler to select a task to run, the `xCoreID` member will allow the scheduler to determine if a given task is permitted to run on the core that called it.

#### 4.10.4 Scheduling

The vanilla FreeRTOS implements scheduling in the `vTaskSwitchContext()` function. This function is responsible for selecting the highest priority task to run from a list of tasks in the Ready state known as the Ready Tasks List (described in the next section). In ESP-IDF FreeRTOS, each core will call `vTaskSwitchContext()` independently to select a task to run from the Ready Tasks List which is shared between both cores. There are several differences in scheduling behavior between vanilla and ESP-IDF FreeRTOS such as differences in Round Robin scheduling, scheduler suspension, and tick interrupt synchronicity.

##### Round Robin Scheduling

Given multiple tasks in the Ready state and of the same priority, vanilla FreeRTOS implements Round Robin scheduling between each task. This will result in running those tasks in turn each time the scheduler is called (e.g. every tick interrupt). On the other hand, the ESP-IDF FreeRTOS scheduler may skip tasks when Round Robin scheduling multiple Ready state tasks of the same priority.

The issue of skipping tasks during Round Robin scheduling arises from the way the Ready Tasks List is implemented in FreeRTOS. In vanilla FreeRTOS, `pxReadyTasksList` is used to store a list of tasks that are in the Ready state. The list is implemented as an array of length `configMAX_PRIORITIES` where each element of the array is a linked list. Each linked list is of type `List_t` and contains TCBs of tasks of the same priority that are in the Ready state. The following diagram illustrates the `pxReadyTasksList` structure.

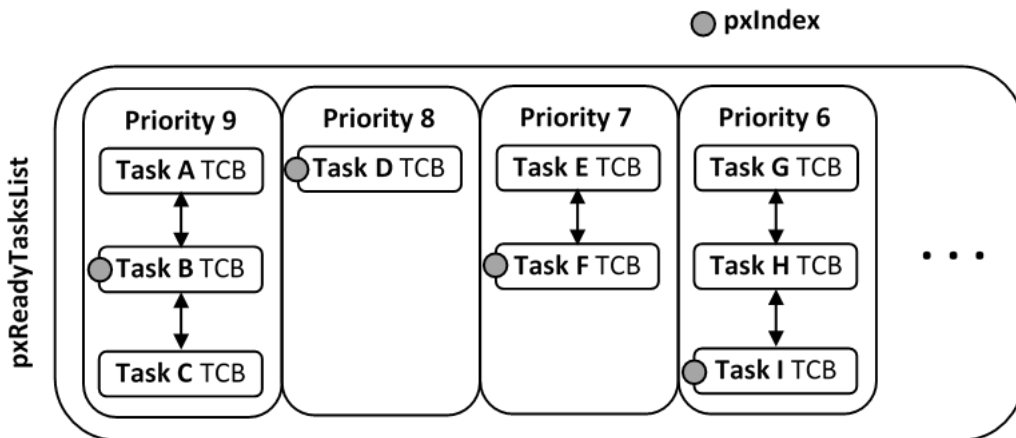


图 2: Illustration of FreeRTOS Ready Task List Data Structure

Each linked list also contains a `pxIndex` which points to the last TCB returned when the list was queried. This index allows the `vTaskSwitchContext()` to start traversing the list at the TCB immediately after `pxIndex` hence implementing Round Robin Scheduling between tasks of the same priority.

In ESP-IDF FreeRTOS, the Ready Tasks List is shared between cores hence `pxReadyTasksList` will contain tasks pinned to different cores. When a core calls the scheduler, it is able to look at the `xCoreID` member of each TCB in the list to determine if a task is allowed to run on calling the core. The ESP-IDF FreeRTOS `pxReadyTasksList` is illustrated below.

Therefore when **PRO\_CPU** calls the scheduler, it will only consider the tasks in blue or purple. Whereas when **APP\_CPU** calls the scheduler, it will only consider the tasks in orange or purple.

Although each TCB has an `xCoreID` in ESP-IDF FreeRTOS, the linked list of each priority only has a single `pxIndex`. Therefore when the scheduler is called from a particular core and traverses the linked list, it will skip all TCBs pinned to the other core and point the `pxIndex` at the selected task. If the other core then calls the scheduler,



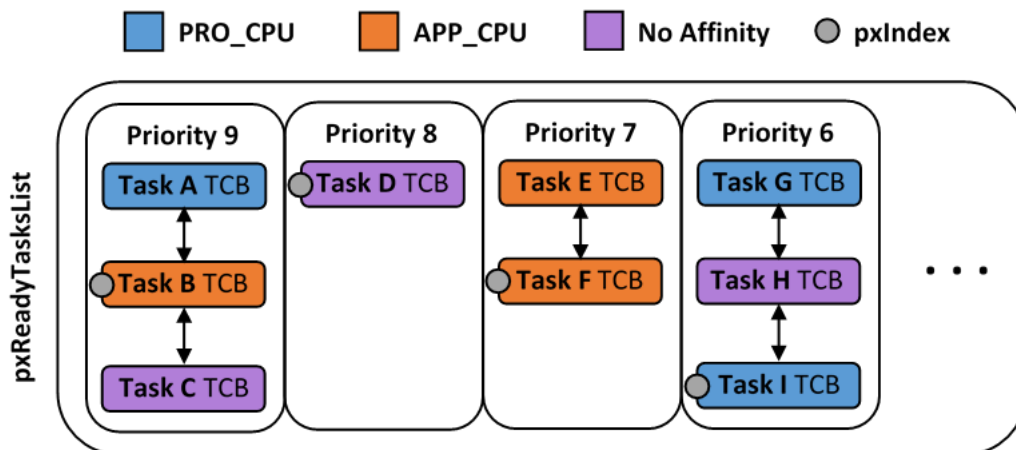
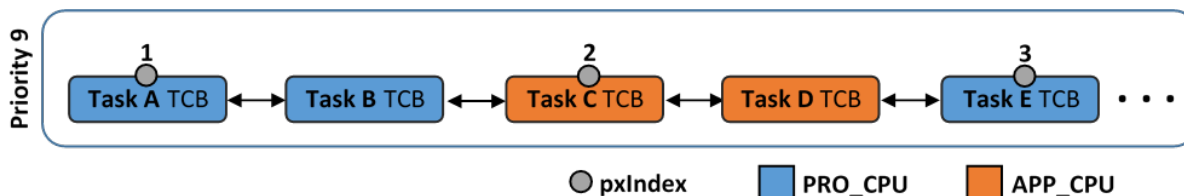


图 3: Illustration of FreeRTOS Ready Task List Data Structure in ESP-IDF

it will traverse the linked list starting at the TCB immediately after `pxIndex`. Therefore, TCBs skipped on the previous scheduler call from the other core would not be considered on the current scheduler call. This issue is demonstrated in the following illustration.

图 4: Illustration of `pxIndex` behavior in ESP-IDF FreeRTOS

Referring to the illustration above, assume that priority 9 is the highest priority, and none of the tasks in priority 9 will block hence will always be either in the running or Ready state.

- 1) **PRO\_CPU** calls the scheduler and selects Task A to run, hence moves `pxIndex` to point to Task A
- 2) **APP\_CPU** calls the scheduler and starts traversing from the task after `pxIndex` which is Task B. However Task B is not selected to run as it is not pinned to **APP\_CPU** hence it is skipped and Task C is selected instead. `pxIndex` now points to Task C
- 3) **PRO\_CPU** calls the scheduler and starts traversing from Task D. It skips Task D and selects Task E to run and points `pxIndex` to Task E. Notice that Task B isn't traversed because it was skipped the last time **APP\_CPU** called the scheduler to traverse the list.
- 4) The same situation with Task D will occur if **APP\_CPU** calls the scheduler again as `pxIndex` now points to Task E

One solution to the issue of task skipping is to ensure that every task will enter a blocked state so that they are removed from the Ready Task List. Another solution is to distribute tasks across multiple priorities such that a given priority will not be assigned multiple tasks that are pinned to different cores.

### Scheduler Suspension

In vanilla FreeRTOS, suspending the scheduler via `vTaskSuspendAll()` will prevent calls of `vTaskSwitchContext` from context switching until the scheduler has been resumed with `xTaskResumeAll()`. However servicing ISRs are still permitted. Therefore any changes in task states as a result from the current running task or ISRs will not be executed until the scheduler is resumed. Scheduler suspension in vanilla FreeRTOS is a common protection method against simultaneous access of data shared between tasks, whilst still allowing ISRs to be serviced.

In ESP-IDF FreeRTOS, `xTaskSuspendAll()` will only prevent calls of `vTaskSwitchContext()` from switching contexts on the core that called for the suspension. Hence if **PRO\_CPU** calls `vTaskSuspendAll()`, **APP\_CPU** will still be able to switch contexts. If data is shared between tasks that are pinned to different cores, scheduler suspension is **NOT** a valid method of protection against simultaneous access. Consider using critical sections (disables interrupts) or semaphores (does not disable interrupts) instead when protecting shared resources in ESP-IDF FreeRTOS.

In general, it's better to use other RTOS primitives like mutex semaphores to protect against data shared between tasks, rather than `vTaskSuspendAll()`.

### Tick Interrupt Synchronicity

In ESP-IDF FreeRTOS, tasks on different cores that unblock on the same tick count might not run at exactly the same time due to the scheduler calls from each core being independent, and the tick interrupts to each core being unsynchronized.

In vanilla FreeRTOS the tick interrupt triggers a call to `xTaskIncrementTick()` which is responsible for incrementing the tick counter, checking if tasks which have called `vTaskDelay()` have fulfilled their delay period, and moving those tasks from the Delayed Task List to the Ready Task List. The tick interrupt will then call the scheduler if a context switch is necessary.

In ESP-IDF FreeRTOS, delayed tasks are unblocked with reference to the tick interrupt on **PRO\_CPU** as **PRO\_CPU** is responsible for incrementing the shared tick count. However tick interrupts to each core might not be synchronized (same frequency but out of phase) hence when **PRO\_CPU** receives a tick interrupt, **APP\_CPU** might not have received it yet. Therefore if multiple tasks of the same priority are unblocked on the same tick count, the task pinned to **PRO\_CPU** will run immediately whereas the task pinned to **APP\_CPU** must wait until **APP\_CPU** receives its out of sync tick interrupt. Upon receiving the tick interrupt, **APP\_CPU** will then call for a context switch and finally switches contexts to the newly unblocked task.

Therefore, task delays should **NOT** be used as a method of synchronization between tasks in ESP-IDF FreeRTOS. Instead, consider using a counting semaphore to unblock multiple tasks at the same time.

## 4.10.5 Critical Sections & Disabling Interrupts

Vanilla FreeRTOS implements critical sections in `vTaskEnterCritical` which disables the scheduler and calls `portDISABLE_INTERRUPTS`. This prevents context switches and servicing of ISRs during a critical section. Therefore, critical sections are used as a valid protection method against simultaneous access in vanilla FreeRTOS.

On the other hand, the ESP32 has no hardware method for cores to disable each other's interrupts. Calling `portDISABLE_INTERRUPTS()` will have no effect on the interrupts of the other core. Therefore, disabling interrupts is **NOT** a valid protection method against simultaneous access to shared data as it leaves the other core free to access the data even if the current core has disabled its own interrupts.

For this reason, ESP-IDF FreeRTOS implements critical sections using mutexes, and calls to enter or exit a critical must provide a mutex that is associated with a shared resource requiring access protection. When entering a critical section in ESP-IDF FreeRTOS, the calling core will disable its scheduler and interrupts similar to the vanilla FreeRTOS implementation. However, the calling core will also take the mutex whilst the other core is left unaffected during the critical section. If the other core attempts to take the same mutex, it will spin until the mutex is released. Therefore, the ESP-IDF FreeRTOS implementation of critical sections allows a core to have protected access to a shared resource without disabling the other core. The other core will only be affected if it tries to concurrently access the same resource.

The ESP-IDF FreeRTOS critical section functions have been modified as follows...

- `taskENTER_CRITICAL(mux)`, `taskENTER_CRITICAL_ISR(mux)`, `portENTER_CRITICAL(mux)`, `portENTER_CRITICAL_ISR(mux)` are all macro defined to call `vTaskEnterCritical()`
- `taskEXIT_CRITICAL(mux)`, `taskEXIT_CRITICAL_ISR(mux)`, `portEXIT_CRITICAL(mux)`, `portEXIT_CRITICAL_ISR(mux)` are all macro defined to call `vTaskExitCritical()`

- `portENTER_CRITICAL_SAFE(mux)`, `portEXIT_CRITICAL_SAFE(mux)` macro identifies the context of execution, i.e. ISR or Non-ISR, and calls appropriate critical section functions (`port*_CRITICAL` in Non-ISR and `port*_CRITICAL_ISR` in ISR) in order to be in compliance with Vanilla FreeRTOS.

For more details see [freertos/include/freertos/portmacro.h](#) and [freertos/task.c](#)

It should be noted that when modifying vanilla FreeRTOS code to be ESP-IDF FreeRTOS compatible, it is trivial to modify the type of critical section called as they are all defined to call the same function. As long as the same mutex is provided upon entering and exiting, the type of call should not matter.

### 4.10.6 Floating Point Arithmetic

The ESP32 supports hardware acceleration of single precision floating point arithmetic (`float`) via Floating Point Units (FPU, also known as coprocessors) attached to each core. The use of the FPUs imposes some behavioral restrictions on ESP-IDF FreeRTOS.

ESP-IDF FreeRTOS implements Lazy Context Switching for FPUs. In other words, the state of a core's FPU registers are not immediately saved when a context switch occurs. Therefore, tasks that utilize `float` must be pinned to a particular core upon creation. If not, ESP-IDF FreeRTOS will automatically pin the task in question to whichever core the task was running on upon the task's first use of `float`. Likewise due to Lazy Context Switching, interrupt service routines must also not use `float`.

ESP32 does not support hardware acceleration for double precision floating point arithmetic (`double`). Instead `double` is implemented via software hence the behavioral restrictions with regards to `float` do not apply to `double`. Note that due to the lack of hardware acceleration, `double` operations may consume significantly larger amount of CPU time in comparison to `float`.

### 4.10.7 Task Deletion

FreeRTOS task deletion prior to v9.0.0 delegated the freeing of task memory entirely to the Idle Task. Currently, the freeing of task memory will occur immediately (within `vTaskDelete()`) if the task being deleted is not currently running or is not pinned to the other core (with respect to the core `vTaskDelete()` is called on). TLSP deletion callbacks will also run immediately if the same conditions are met.

However, calling `vTaskDelete()` to delete a task that is either currently running or pinned to the other core will still result in the freeing of memory being delegated to the Idle Task.

### 4.10.8 Thread Local Storage Pointers & Deletion Callbacks

Thread Local Storage Pointers (TLSP) are pointers stored directly in the TCB. TLSP allow each task to have its own unique set of pointers to data structures. However task deletion behavior in vanilla FreeRTOS does not automatically free the memory pointed to by TLSP. Therefore if the memory pointed to by TLSP is not explicitly freed by the user before task deletion, memory leak will occur.

ESP-IDF FreeRTOS provides the added feature of Deletion Callbacks. Deletion Callbacks are called automatically during task deletion to free memory pointed to by TLSP. Each TLSP can have its own Deletion Callback. Note that due to the `Task Deletion` behavior, there can be instances where Deletion Callbacks are called in the context of the Idle Tasks. Therefore Deletion Callbacks **should never attempt to block** and critical sections should be kept as short as possible to minimize priority inversion.

Deletion callbacks are of type `void (*TlsDeleteCallbackFunction_t)(int, void *)` where the first parameter is the index number of the associated TLSP, and the second parameter is the TLSP itself.

Deletion callbacks are set alongside TLSP by calling `vTaskSetThreadLocalStoragePointerAndDeleteCallback()`. 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. If a deletion callback is `NULL`, users should manually free the memory pointed to by the associated TLSP before task deletion in order to avoid memory leak.

`CONFIG_FREERTOS_THREAD_LOCAL_STORAGE_POINTERS` in menuconfig can be used to configure the number of TLSP and Deletion Callbacks a TCB will have.

For more details see [FreeRTOS API reference](#).

## 4.10.9 Configuring ESP-IDF FreeRTOS

The 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 [FreeRTOS](#)

`CONFIG_FREERTOS_UNICORE` will run ESP-IDF FreeRTOS only on **PRO\_CPU**. Note that this is **not equivalent to running vanilla FreeRTOS**. Behaviors of multiple components in ESP-IDF will be modified such as `esp32/cpu_start.c`. For more details regarding the effects of running ESP-IDF FreeRTOS on a single core, search for occurrences of `CONFIG_FREERTOS_UNICORE` in the ESP-IDF components.

`CONFIG_FREERTOS_THREAD_LOCAL_STORAGE_POINTERS` will define the number of Thread Local Storage Pointers each task will have in ESP-IDF FreeRTOS.

`CONFIG_FREERTOS_SUPPORT_STATIC_ALLOCATION` will enable the backported functionality of `xTaskCreateStaticPinnedToCore()` in ESP-IDF FreeRTOS

`CONFIG_FREERTOS_ASSERT_ON_UNTESTED_FUNCTION` will trigger a halt in particular functions in ESP-IDF FreeRTOS which have not been fully tested in an SMP context.

`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
```

## 4.11 Thread Local Storage

### 4.11.1 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.

### 4.11.2 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 `configNUM_THREAD_LOCAL_STORAGE_POINTERS` macro. 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 than 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.

### 4.11.3 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.

### 4.11.4 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>. 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 just a few Xtensa 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.

## 4.12 High-Level Interrupts

The Xtensa architecture has support for 32 interrupts, divided over 8 levels, 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, allowing for very low interrupt latencies.

### 4.12.1 Interrupt Levels

Level	Symbol	Remark
1	N/A	Exception and level 0 interrupts. Handled by ESP-IDF
2-3	N/A	Medium level interrupts. Handled by ESP-IDF
4	<code>xt_highint4</code>	Normally used by ESP-IDF debug logic
5	<code>xt_highint5</code>	Free to use
NMI	<code>xt_nmi</code>	Free to use
dbg	<code>xt_debugexception</code>	Debug exception. Called on e.g. a BREAK instruction.

Using these symbols is done by creating an assembly file (suffix `.S`) and defining the named symbols, like this:

```
.section .iram1,"ax"
.global xt_highint5
.type xt_highint5,@function
.align 4
xt_highint5:
... your code here
rsr a0, EXCSAVE_5
rfi 5
```

For a real-life example, see the `esp32/dport_panic_highint_hdl.S` file; the panic handler interrupt is implemented there.

### 4.12.2 Notes

- Do not call C code from a high-level interrupt; because these interrupts still run in critical sections, this can cause crashes. (The panic handler interrupt does call normal C code, but this is OK because there is no intention of returning to the normal code flow afterwards.)
- Make sure your assembly code gets linked in. If the interrupt handler symbol is the only symbol the rest of the code uses from this file, the linker will take the default ISR instead and not link the assembly file into the final project. To get around this, in the assembly file, define a symbol, like this:

```
.global ld_include_my_isr_file
ld_include_my_isr_file:
```

(The symbol is called `ld_include_my_isr_file` here but can have any arbitrary name not defined anywhere else.) Then, in the component.mk, add this file as an unresolved symbol to the ld command line arguments:

```
COMPONENT_ADD_LDFLAGS := -u ld_include_my_isr_file
```

This should cause the linker to always include a 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. For now, ESP-IDF does not support this.

## 4.13 JTAG 调试

本文将指导安装 ESP32 的 OpenOCD 调试环境，并介绍如何使用 GDB 来调试 ESP32 的应用程序。本文的组织结构如下：

**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 是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 硬件适配器, 这两者共同组成最终的调试目标。

**启动调试器** 介绍如何从 *Eclipse* 集成开发环境和命令行终端启动 GDB 调试会话。

**调试范例** 如果你对 GDB 不太熟悉, 本小节会分别针对 *Eclipse* 集成开发环境和命令行终端来讲解调试的范例。

**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。

### 4.13.1 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 操作系统具有多核抢占式多线程的功能, 它允许用户以更加直观的方式编写软件。

与此相对地, 简便的编程方式会给程序的调试带来困难 (如果没有合适的工具), 比如找出由两个线程引起的错误, 并且这两个线程在单独的 CPU 核上同时运行, 仅凭 `printf` 语句会花费很长的时间来定位到该错误。在大多数情况下, 调试此类问题更快的方法是使用调试器, 连接到处理器的调试端口。

乐鑫已经为 ESP32 处理器和多核 FreeRTOS 架构移植好了 OpenOCD, 它将成为大多数 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来进一步丰富调试的功能。

本文将指导如何在 Linux, Windows 和 MacOS 环境下为 ESP32 安装 OpenOCD, 并使用 GDB 进行软件调试。除了个别操作系统的安装过程有所差别以外, 软件用户界面和使用流程都是一样的。

**注解：** 本文使用的图片素材来自于 Ubuntu 16.04 LTE 上 Eclipse Neon 3 软件的截图，不同的操作系统（Windows，MacOS 或者 Linux）和 Eclipse 软件版本在用户界面上可能会有细微的差别。

### 4.13.2 工作原理

通过 JTAG（Joint Test Action Group）接口使用 OpenOCD 调试 ESP32 时所需要的一些关键的软件和硬件包括 **xtensa-esp32-elf-gdb 调试器**，**OpenOCD 片上调试器**和连接到 **ESP32 目标**的 **JTAG 适配器**。

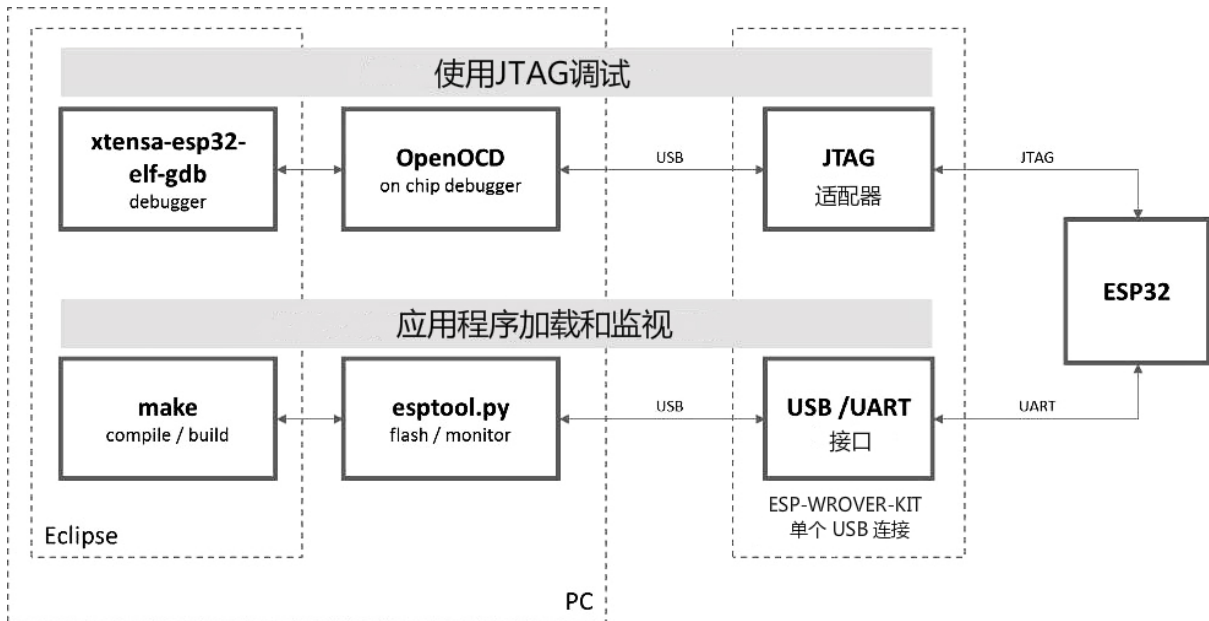


图 5: JTAG 调试 - 概述图

在“Application Loading and Monitoring”下还有另外一组软件和硬件，它们用来编译、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，它使得软件从编写、编译、加载到调试的迭代过程变得更加快速而简单。所有的软件均适用于 Windows，Linux 和 MacOS 平台。

如果你使用的是 **ESP-WROVER-KIT 开发板**，得益于板载的 FT232H 芯片，PC 和 ESP32 的连接仅仅需要一根 USB 线即可完成。FT232H 提供了两路 USB 通道，一路连接到 JTAG，另一路连接到 UART。

根据用户的喜好，除了使用 Eclipse 集成开发环境，上述的调试工具和构建工具还可以直接在命令行终端运行。

### 4.13.3 选择 JTAG 适配器

上手 JTAG 最快速便捷的方式是使用 **ESP-WROVER-KIT 开发板**，因为它板载了 JTAG 调试接口，无需使用外部的 JTAG 硬件适配器和额外的线缆来连接 JTAG 与 ESP32。ESP-WROVER-KIT 采用 FT232H 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。

如果你想使用单独的 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，它省略了（实际上也并不需要）TRST 信号脚。JTAG 使用的 IO 引脚由 VDD\_3P3\_RTC 电源引脚供电（通常连接到外部 3.3 V 的电源轨），因此 JTAG 硬件适配器的引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，可以参阅 [OpenOCD 支持的适配器列表](#)（尽管上面显示的器件不太完整），这个页面还列出了兼容 SWD 接口的适配器，但是请注意，ESP32 目前并不

支持 SWD。此外那些被硬编码为只支持特定产品线的 JTAG 适配器也不能在 ESP32 上工作，比如用于 STM32 产品家族的 ST-LINK 适配器。

JTAG 正常工作至少需要连接的信号线有：TDI，TDO，TCK，TMS 和 GND。某些 JTAG 适配器还需要 ESP32 提供一路电源到适配器的某个引脚上（比如 Vtar）用以设置适配器的工作电压。SRST 信号线是可选的，它可以连接到 ESP32 的 CH\_PD 引脚上，尽管目前 OpenOCD 对该信号线的支持还非常有限。

### 4.13.4 安装 OpenOCD

如果你已经按照[快速入门](#)一文中的介绍安装好了 ESP-IDF 及其 CMake 构建系统，那么 OpenOCD 已经被默认安装到了你的开发系统中。在[设置开发环境](#)结束后，你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（实际版本号可能会比这里列出的更新）：

```
Open On-Chip Debugger v0.10.0-esp32-20190708 (2019-07-08-11:04)
Licensed under GNU GPL v2
For bug reports, read
  http://openocd.org/doc/doxygen/bugs.html
```

你还可以检查 OPENOCD\_SCRIPTS 环境变量的值来确认 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%`。如果终端打印了有效的路径，则表明 OpenOCD 已经被正确安装。

如果上述步骤没有成功执行，请返回[快速入门手册](#)，参考其中[设置安装工具](#)章节的说明。

---

**注解：**另外，我们还可以从源代码编译 OpenOCD 工具，相关详细信息请参阅[从源码构建 OpenOCD](#)章节。

---

### 4.13.5 配置 ESP32 目标板

安装好 OpenOCD 之后就可以配置 ESP32 目标（即带 JTAG 接口的 ESP32 板），具体可以通过以下三个步骤进行：

- 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

此步骤取决于您使用的 JTAG 和 ESP32 板，请参考以下两种情况。

**配置 WROVER 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，还需要设置相关跳帽来启用 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

1. 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。
2. 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。



	ESP32 引脚	JTAG 信号
1	CHIP_PU	TRST_N
2	MTDO / GPIO15	TDO
3	MTDI / GPIO12	TDI
4	MTCK / GPIO13	TCK
5	MTMS / GPIO14	TMS

**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。

**注解：** ESP-WROVER-KIT 使用了 FT232 芯片实现了 JTAG 适配器，所以以下说明同样适用于其他基于 FT232 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：0403 6010。

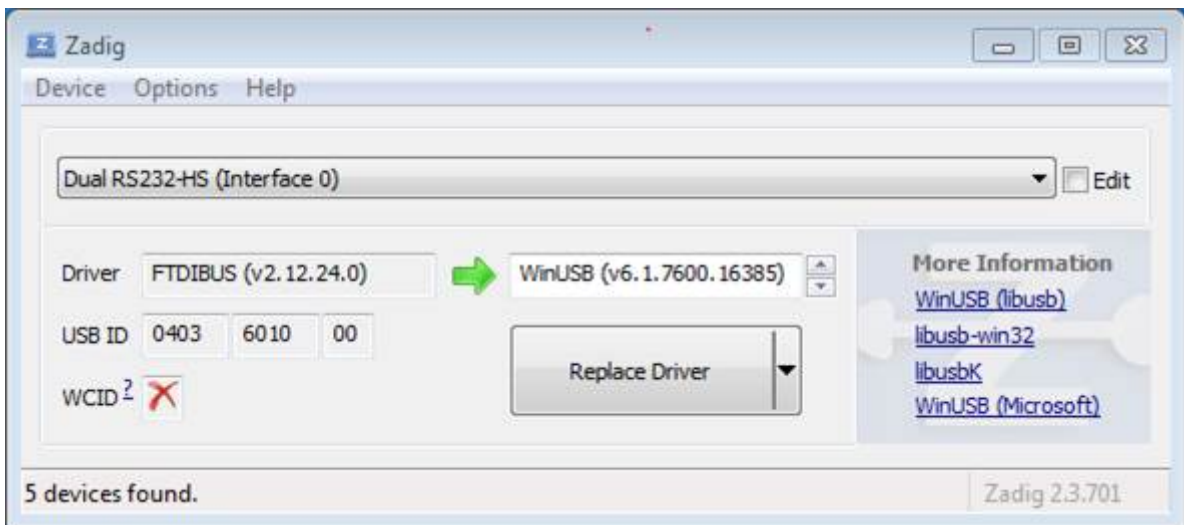


图 6: 在 Zadig 工具中配置 JTAG USB 驱动

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 (v6xxxx)”。为此，请选择“Dual RS232-HS (Interface 0)”并将驱动重新安装为“WinUSB (v6xxxx)”，具体可以参考上图。

**注解：** 请勿更改第二个设备“Dual RS232-HS (Interface 1)”的驱动，它被连接到 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。

**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，然后重新插拔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者是否已经从 dialout 更改为 plugdev:

```
user-name@computer-name:~/esp$ ls -l /dev/ttyUSB*
crw-rw-r-- 1 root plugdev 188, 0 Jul 10 19:07 /dev/ttyUSB0
crw-rw-r-- 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. 修改 FTDI 驱动程序的配置，使其不会为 FT2232 芯片的通道 B 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：

```
sudo kextunload -b com.apple.driver.AppleUSBFTDI
```

4. 运行 OpenOCD:

```
openocd -f interface/ftdi/esp32_devkitj_v1.cfg -f board/esp-wroom-32.cfg
```

5. 在另一个终端窗口，再一次加载 FTDI 串口驱动:

```
sudo kextload -b com.FTDI.driver.FTDIUSBSerialDriver
```

**注解：**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。

**注解：** 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 驱动器的配置文件（注意 sudo）：

```
sudo nano /Library/Extensions/FTDIUSBSerialDriver.kext/Contents/Info.plist
```

2. 找到并删除以下几行：

```
<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 组合键
  3. 进入恢复模式后，打开终端
  4. 运行命令：

```
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。

**配置其它 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引脚/信号。

	ESP32 引脚	JTAG 信号
1	CHIP_PU	TRST_N
2	MTDO / GPIO15	TDO
3	MTDI / GPIO12	TDI
4	MTCK / GPIO13	TCK
5	MTMS / GPIO14	TMS
6	GND	GND

2. 检查 ESP32 上用于 JTAG 通信的引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 接口上的引脚/信号。

**配置驱动** 你可能还需要安装软件驱动，才能使 JTAG 在计算机上正常工作，请参阅你所使用的 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。

要继续设置调试环境，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，按照快速入门中的指南设置好开发环境，然后运行如下命令，启动 OpenOCD（该命令在 Windows，Linux，和 macOS 中通用）：

```
openocd -f interface/ftdi/esp32_devkitj_v1.cfg -f board/esp-wroom-32.cfg
```

**注解：** 上述命令中 `-f` 选项后跟的配置文件专用于板载 *ESP-WROOM-32* 模组的 *ESP-WROVER-KIT* 开发板。您可能需要根据具体使用的硬件而选择或修改不同的配置文件，相关指导请参阅[针对特定目标的 OpenOCD 配置](#)。

现在应该可以看到如下输入（此日志来自 *ESP-WROVER-KIT*）：

```
user-name@computer-name:~/esp/esp-idf$ openocd -f interface/ftdi/esp32_devkitj_v1.
↪cfg -f board/esp-wroom-32.cfg
Open On-Chip Debugger v0.10.0-esp32-20190708 (2019-07-08-11:04)
Licensed under GNU GPL v2
For bug reports, read
    http://openocd.org/doc/doxygen/bugs.html
none separate
adapter speed: 20000 kHz
force hard breakpoints
Info : ftdi: if you experience problems at higher adapter clocks, try the command
↪"ftdi_tdo_sample_edge falling"
Info : clock speed 20000 kHz
Info : JTAG tap: esp32.cpu0 tap/device found: 0x120034e5 (mfg: 0x272 (Tensilica),
↪part: 0x2003, ver: 0x1)
Info : JTAG tap: esp32.cpu1 tap/device found: 0x120034e5 (mfg: 0x272 (Tensilica),
↪part: 0x2003, ver: 0x1)
Info : esp32: Debug controller was reset (pwrstat=0x5F, after clear 0x0F).
Info : esp32: Core was reset (pwrstat=0x5F, after clear 0x0F).
```

- 如果出现指示权限问题的错误，请参阅 `~/esp/openocd-esp32` 目录下 *OpenOCD README* 文件中关于“Permissions delegation”的说明。

- 如果发现配置文件有错误，例如 `Can't find interface/ftdi/esp32_devkitj_v1.cfg`，请检查 `-s` 后面的路径，OpenOCD 会根据此路径来查找 `-f` 指定的文件。此外，还需要检查配置文件是否确实位于该路径下。
- 如果看到 JTAG 错误（输出全是 1 或者全是 0），请检查硬件连接，除了 ESP32 的引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。

### 上传待调试的应用程序

您可以像往常一样构建并上传 ESP32 应用程序，具体请参阅[第八步：编译工程](#) 章节。

除此以外，还支持使用 OpenOCD 通过 JTAG 接口将应用程序镜像烧写到闪存中，命令如下：

```
openocd -f interface/ftdi/esp32_devkitj_v1.cfg -f board/esp-wroom-32.cfg -c
↪ "program_esp32 filename.bin 0x10000 verify exit"
```

其中 OpenOCD 的烧写命令 `program_esp32` 具有以下格式：

```
program_esp32 <image_file> <offset> [verify] [reset] [exit]
```

- `image_file` - 程序镜像文件存放的路径
- `offset` - 镜像烧写到闪存中的偏移地址
- `verify` - 烧写完成后校验闪存中的内容（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）

现在可以进行应用程序的调试了，请按照以下章节中讲解的步骤进行操作。

### 4.13.6 启动调试器

ESP32 的工具链中带有 GNU 调试器（简称 GDB）`xtensa-esp32-elf-gdb`，它和其它工具链软件存放在同一个 `bin` 目录下。除了直接在命令行终端中调用并操作 GDB 外，还可以在 IDE（例如 Eclipse，Visual Studio Code 等）中调用它，在图形用户界面的帮助下间接操作 GDB，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。

- [在 Eclipse 中使用 GDB](#)
- [在命令行中使用 GDB](#)

建议首先检查调试器是否能在[命令行终端](#) 下正常工作，然后再转到使用 Eclipse 等[集成开发环境](#) 下进行调试工作。

### 4.13.7 调试范例

本节适用于不熟悉 GDB 的用户，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[命令行终端进行调试](#) 的案例。

在演示之前，请设置好 ESP32 目标板并加载 [get-started/blink](#) 至 ESP32 中。

### 4.13.8 从源码构建 OpenOCD

请参阅以下文档，它们分别介绍了在各大操作系统平台上从源码构建 OpenOCD 的流程。

#### Windows 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。

**注解：**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。

```
pacman -S libtool
pacman -S autoconf
pacman -S automake
pacman -S texinfo
pacman -S mingw-w64-i686-libusb-compat-git
pacman -S pkg-config
```

**注解：**安装 pkg-config 会破坏 esp-idf 的工具链，因而在 OpenOCD 构建完成后，应将其卸载。详见文末进一步说明。如果想要再次构建 OpenOCD，你需要再次运行 `pacman -S pkg-config`。此步骤安装的其他软件包（在 pkg-config 之前）并不会出现这一问题。

**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。

**注解：**

- 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.Windows。

一旦 make 过程成功完成，OpenOCD 的可执行文件会被保存到 `~/esp/openocd-esp32/src/openocd` 目录中。

如安装依赖步骤所述，最后还需要移除 `pkg-config` 软件包：

```
pacman -Rs pkg-config
```

**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。

**注解：**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
```

**注解：**

- `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。
- 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。

**注解:**

- 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** 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。

**注解:**

- 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 `~/esp/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。

本文档演示所使用的 OpenOCD 是[安装 OpenOCD](#) 章节中介绍的预编译好的二进制发行版。

如果要使用本地从源代码编译的 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，这样 OpenOCD 才能找到配置文件。Linux 和 macOS 用户可以执行：

```
cd ~/esp/openocd-esp32
export OPENOCD_SCRIPTS=$PWD/tcl
```

Windows 用户可以执行：

```
cd %USERPROFILE%\esp\openocd-esp32
set "OPENOCD_SCRIPTS=%CD%\tcl"
```

运行本地编译的 OpenOCD 的示例如下（Linux 和 macOS 用户）：

```
src/openocd -f interface/ftdi/esp32_devkitj_v1.cfg -f board/esp-wroom-32.cfg
```

Windows 用户：

```
src\openocd -f interface\ftdi\esp32_devkitj_v1.cfg -f board\esp-wroom-32.cfg
```

### 4.13.9 注意事项和补充内容

本节列出了本指南中提到的所有注意事项和补充内容的链接。

- [可用的断点和观察点](#)
- [关于断点的补充知识](#)
- [闪存映射 vs 软件闪存断点](#)
- [“next”命令无法跳过子程序的原因](#)
- [OpenOCD 支持的编译时的选项](#)
- [支持 FreeRTOS](#)
- [在 OpenOCD 的配置文件中设置 SPI 闪存的工作电压](#)
- [优化 JTAG 的速度](#)
- [调试器的启动命令的含义](#)
- [针对特定目标的 OpenOCD 配置](#)
- [复位 ESP32](#)
- [不要将 JTAG 引脚用于其他功能](#)
- [报告 OpenOCD / GDB 的问题](#)

### 4.13.10 相关文档

#### 使用调试器

本节会在[Eclipse](#) 和 [命令行](#) 中分别介绍配置和运行调试器的方法。我们建议你首先通过[命令行](#) 检查调试器是否正常工作，然后再转到使用[Eclipse](#) 平台。

**在 Eclipse 中使用 GDB** 标准的 Eclipse 安装流程默认安装调试功能，另外我们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，通过打开 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，请先按照[使用 Eclipse IDE 编译和烧写](#)文章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，那么很有可能该项目还没有编译，请参考[使用 Eclipse IDE 编辑和烧写](#)指南中的介绍。
  5. 最后，在“Build (if required) before launching”下面点击“Disable auto build”。
- 上述步骤 1 - 5 的示例输入如下图所示。

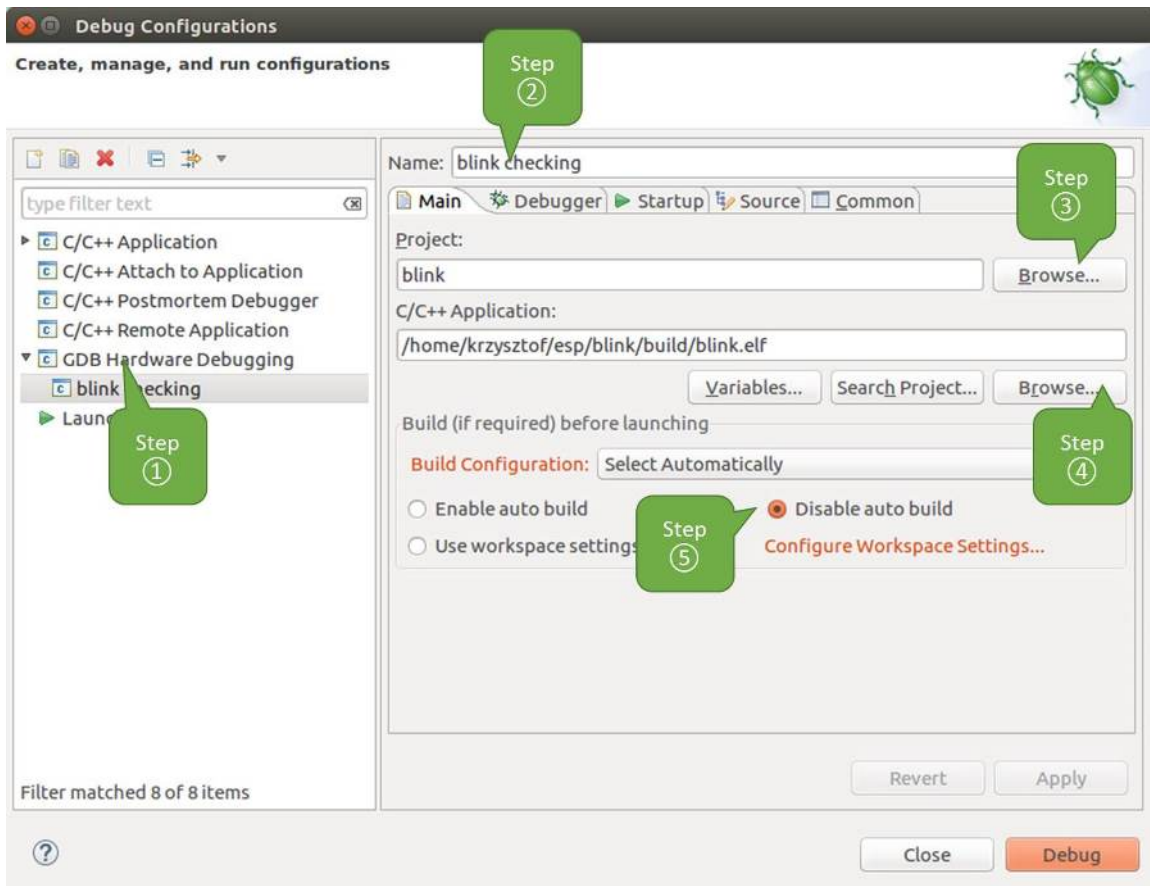


图 7: GDB 硬件调试的配置 - Main 选项卡

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卡是“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
```

**注解：** 如果你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：

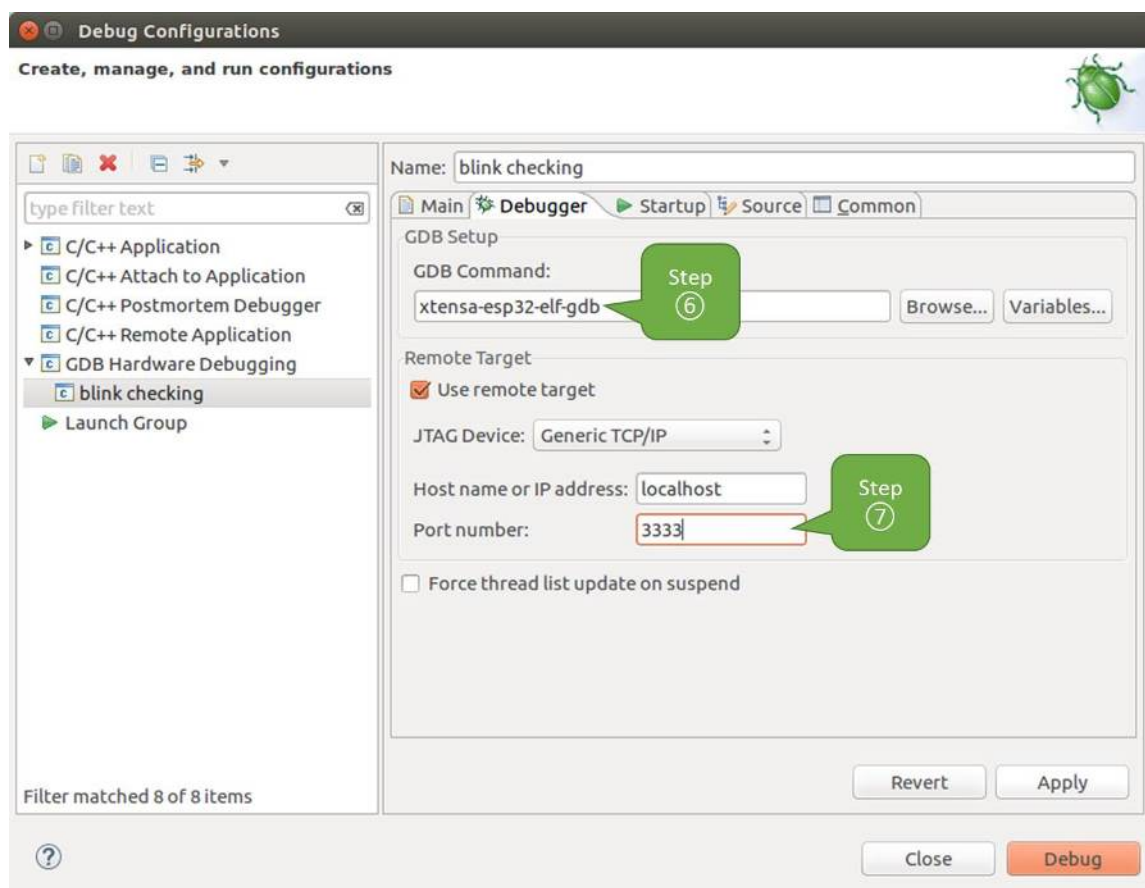


图 8: GDB 硬件调试的配置 - Debugger 选项卡

```
mon reset halt
mon program_esp32 ${workspace_loc:blink/build/blink.bin} 0x10000 verify
```

有关 `program_esp32` 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` 之后恢复，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。

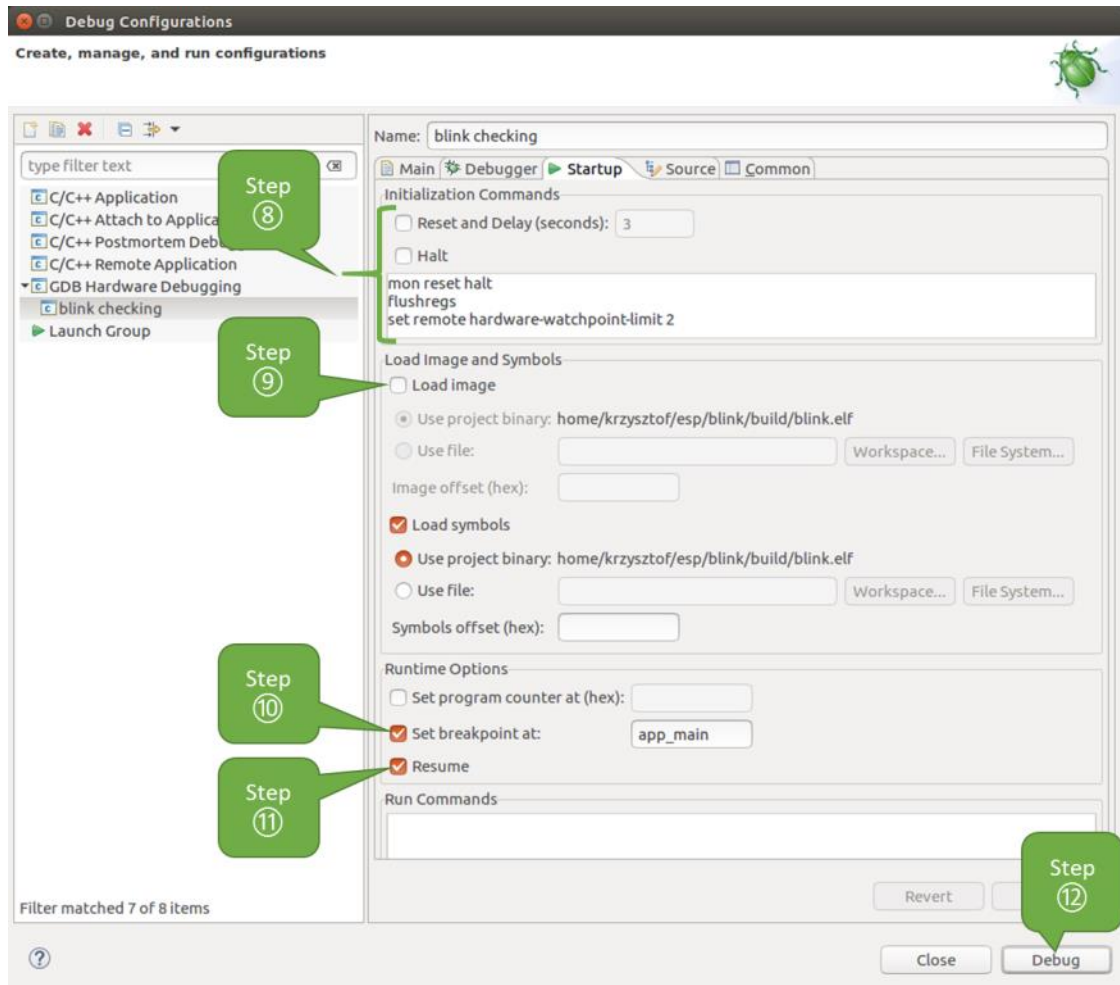


图 9: GDB 硬件调试的配置 - Startup 选项卡

上面的启动序列看起来有些复杂，如果你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. 如果你前面已经完成[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。

如果你不太了解 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](#)。

## 在命令行中使用 GDB



图 10: Eclipse 中的调试视图

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：

```
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，我们可以新建一个配置文件，并将其命名为 gdbinit：

```
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：

```
xtensa-esp32-elf-gdb -x gdbinit build/blink.elf
```

5. 如果前面的步骤已经正确完成，你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：

```
user-name@computer-name:~/esp/blink$ xtensa-esp32-elf-gdb -x gdbinit build/
↳blink.elf
GNU 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:
<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 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).
Target halted. PRO_CPU: PC=0x5000004B (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
```

(下页继续)

```
[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。如果这也是你看到的现象，你可以开始调试了。

如果你不太了解 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](#)。

## 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，然后按照在 *Eclipse* 中使用 GDB 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）为止的相关函数的调用关系。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()` 中被调用的。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。

回到 1 号线程中的 `app_main()` 函数所在的 `blink.c` 源码文件，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。

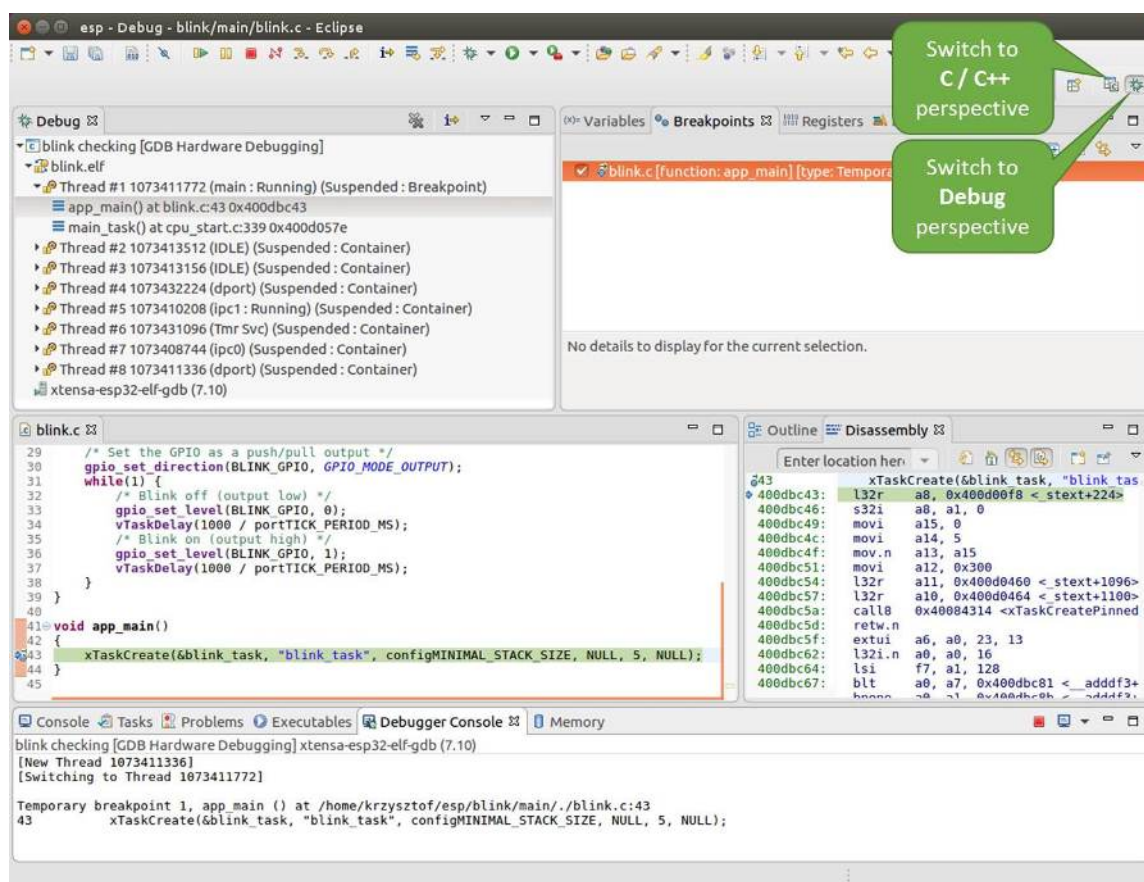


图 11: Eclipse 中的 Debug 视图



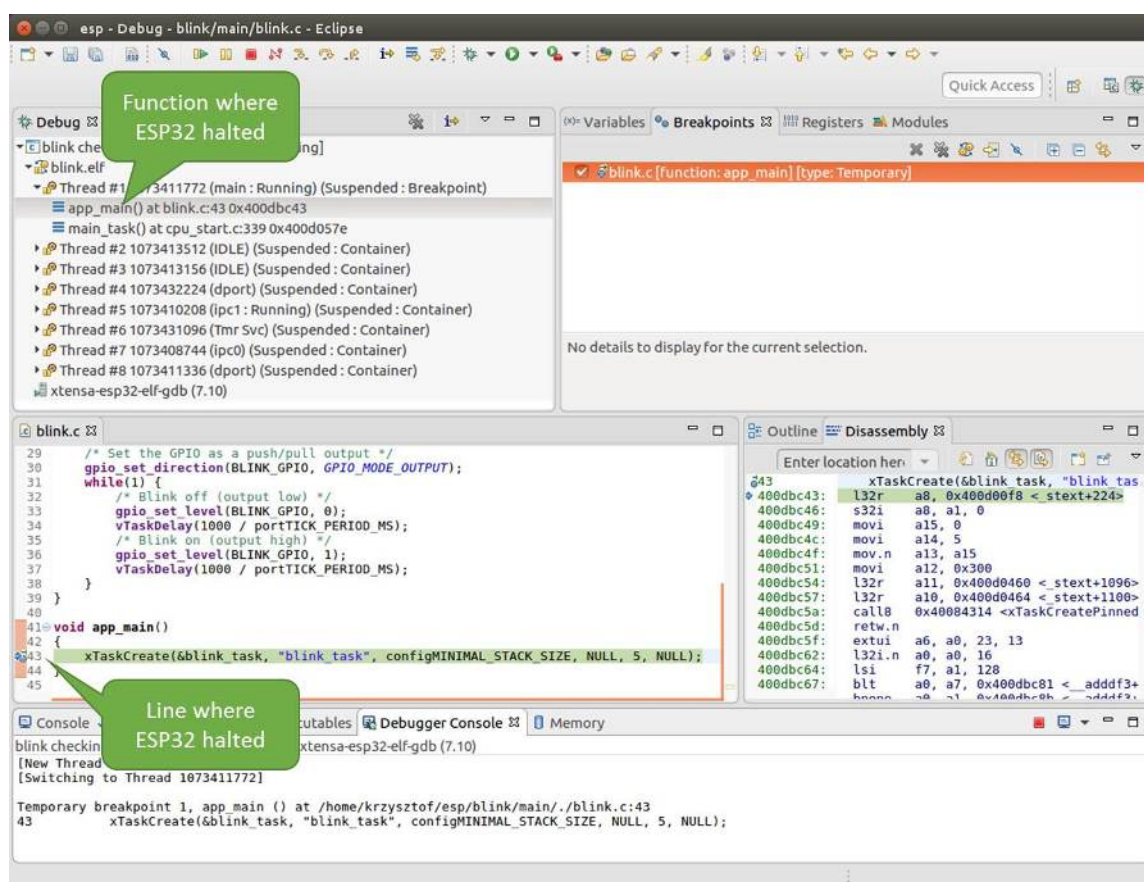


图 12: 调试时目标停止

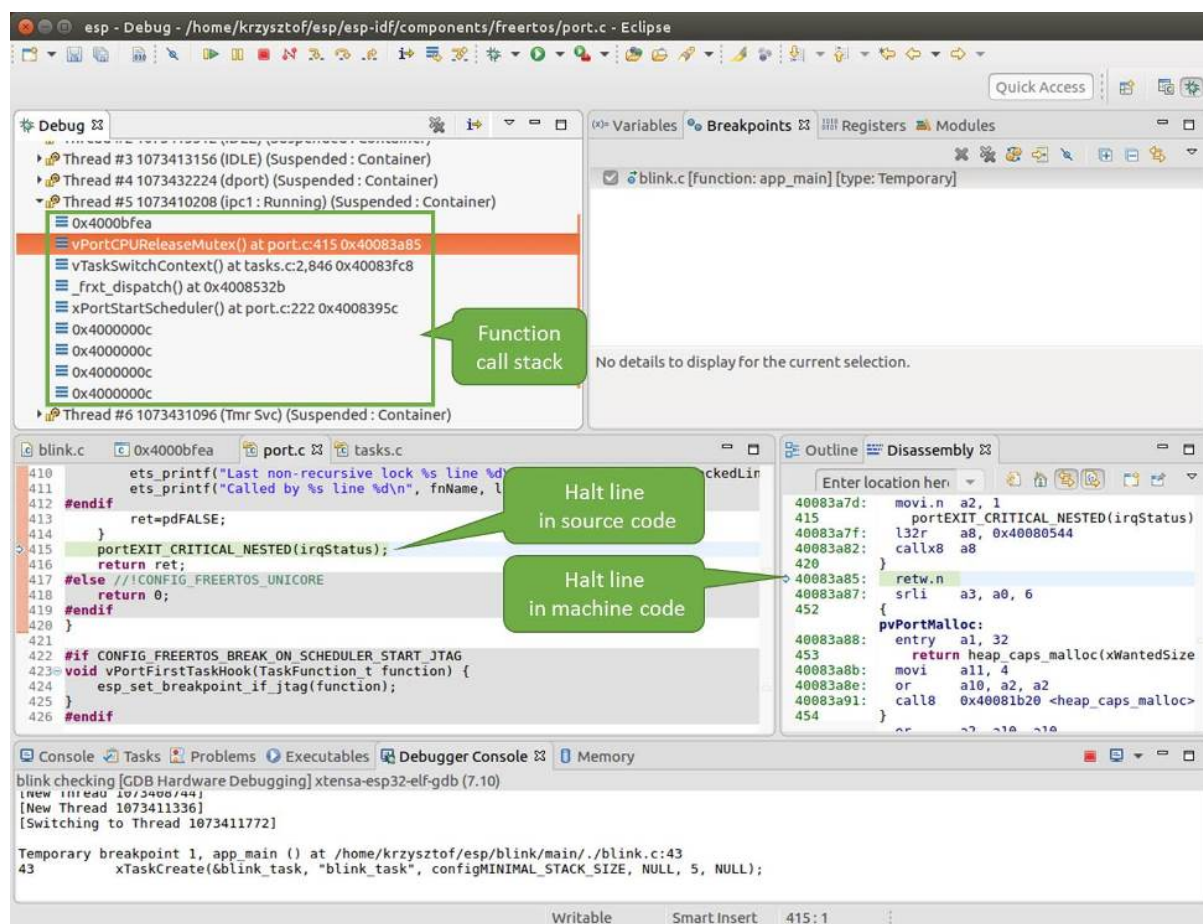


图 13: 浏览函数调用堆栈

**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，以便在特定某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。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。如果你不想看到此对话框，双击行号即可。执行同样操作，在第 36 行设置另外一个断点。

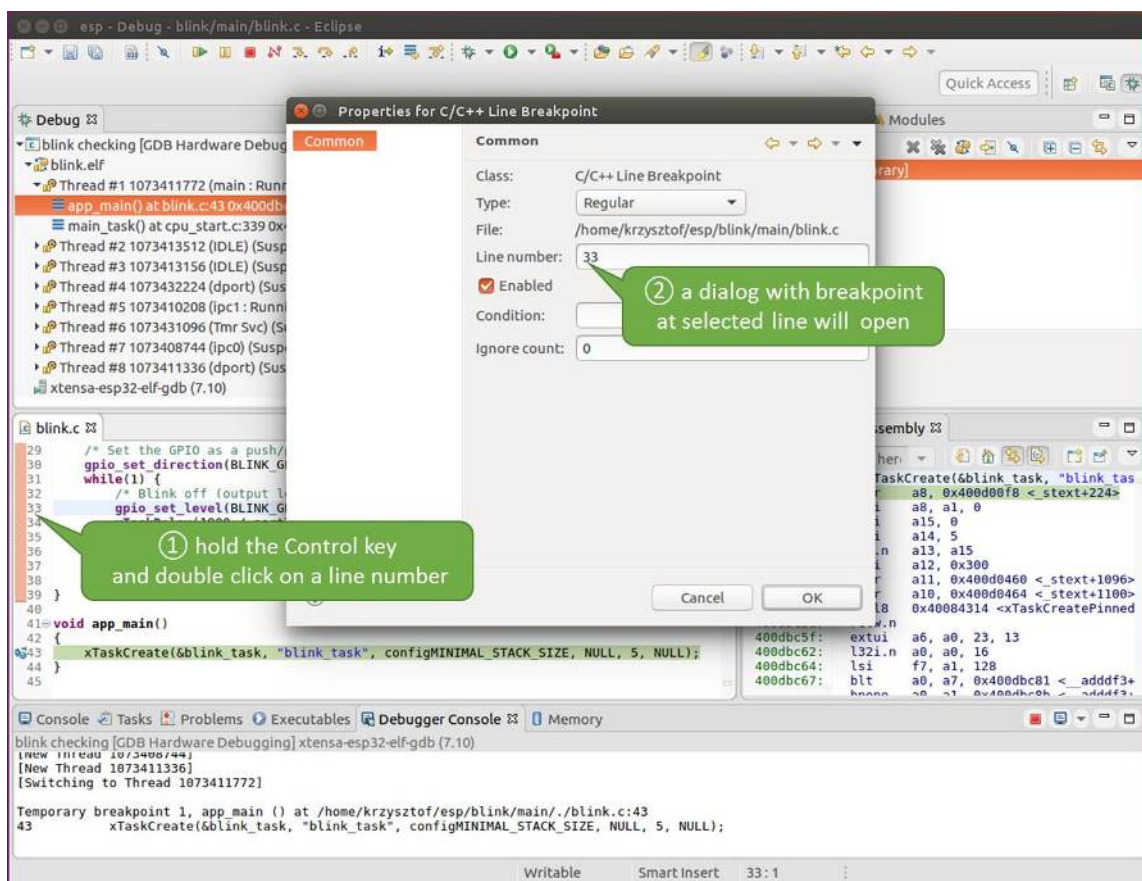


图 14: 设置断点

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。再一次单击“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。

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。

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。

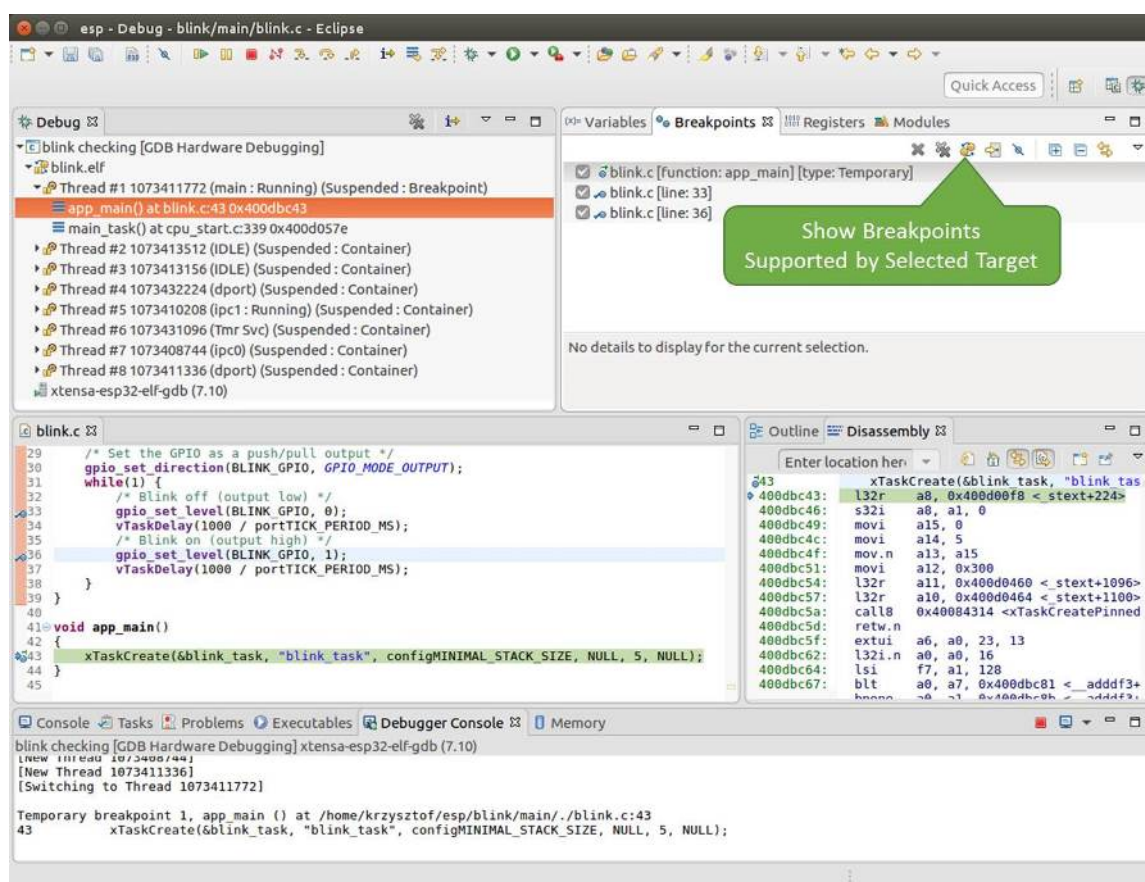


图 15: 设置了三个断点 / 最多允许两个断点

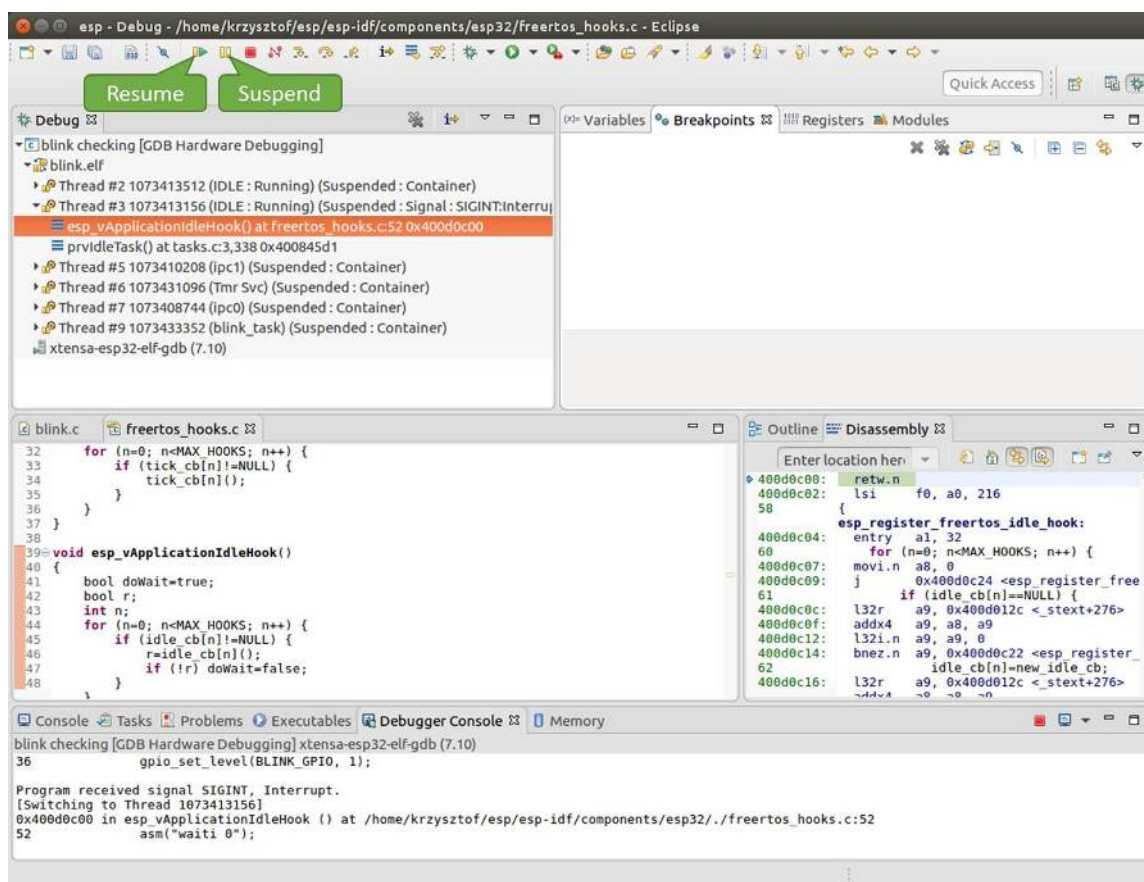


图 16: 手动暂停目标

**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。

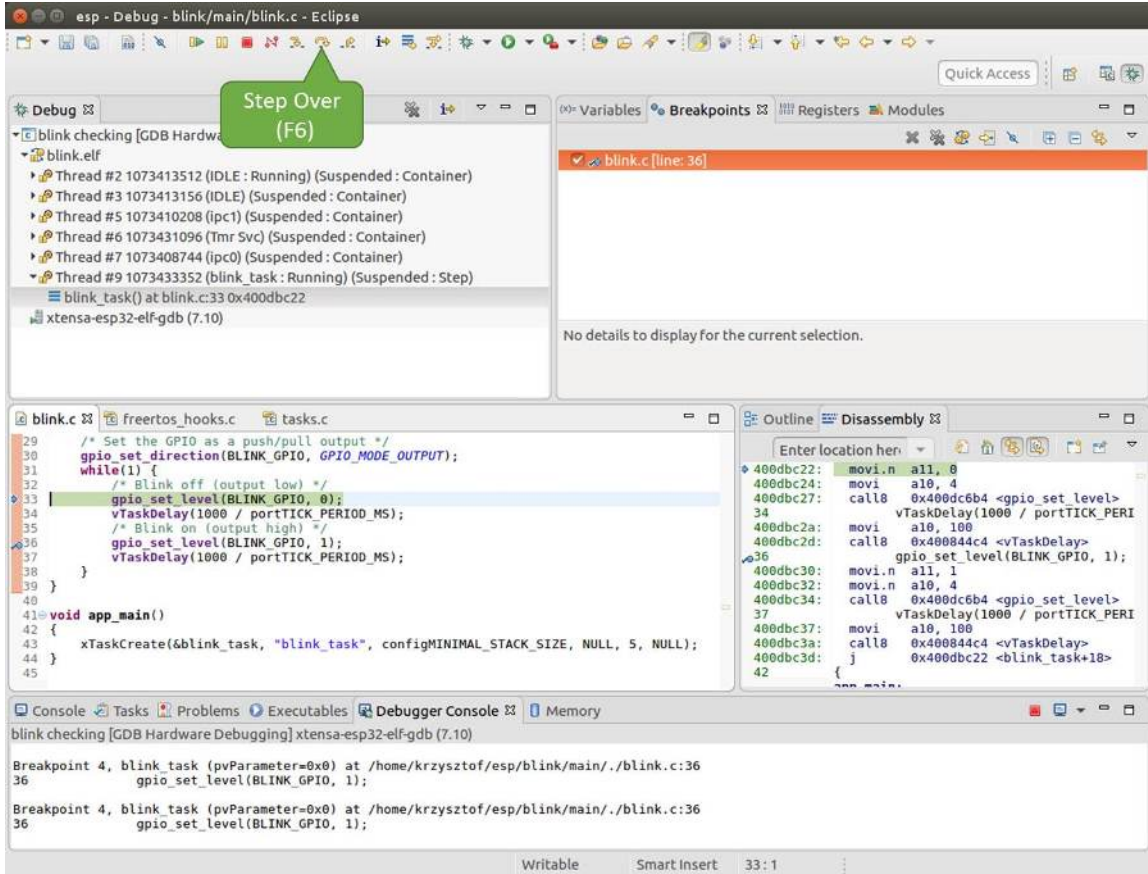


图 17: 使用“Step Over (F6)”单步执行代码

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](#) 中的 [IO\\_MUX](#) 和 [GPIO Matrix](#) 章节。

同样在 `blink.c` 项目文件中，在两个 `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。

每按一下 F8，你就会看到在内存 `0x3FF44004` 地址处的一个比特位被翻转（并且 LED 会改变状态）。

要修改内存的数值，请在“Monitor”选项卡中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，将立即看到 LED 的状态发生了改变。

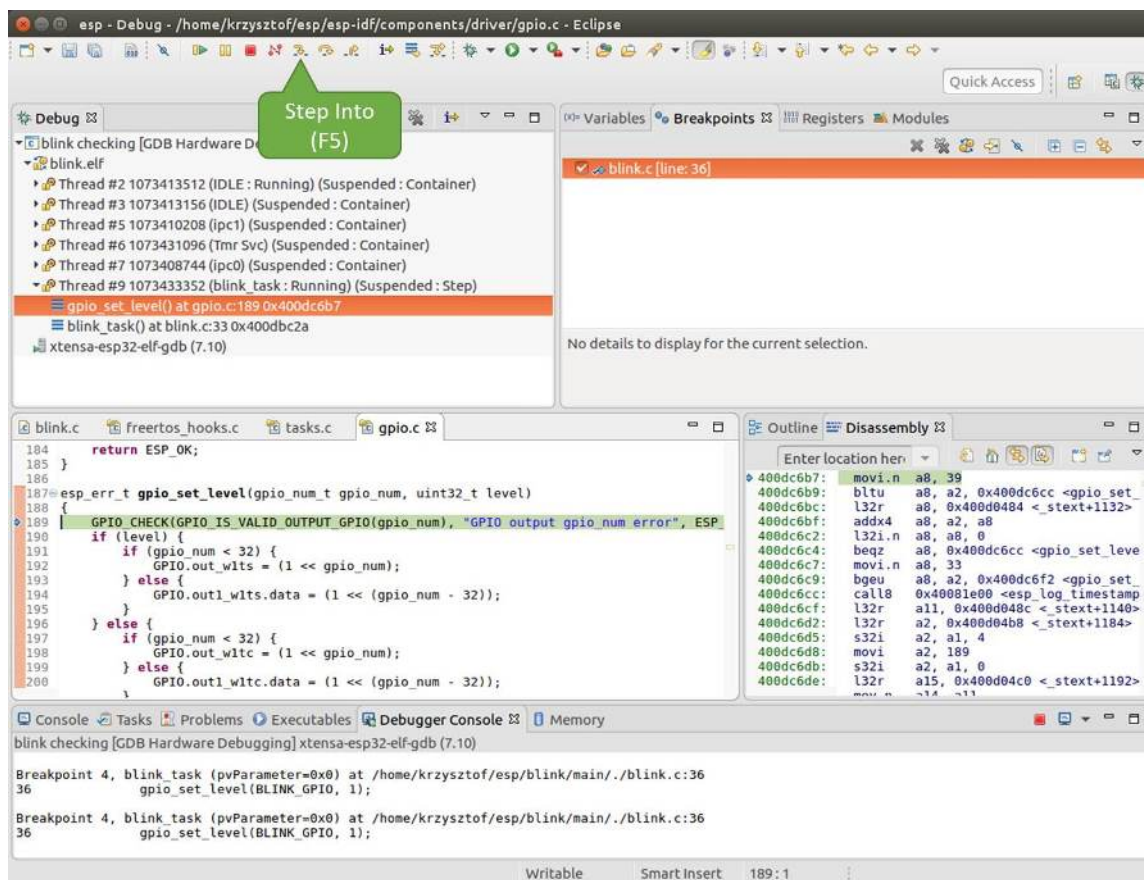


图 18: 使用“Step Into (F5)”单步执行代码

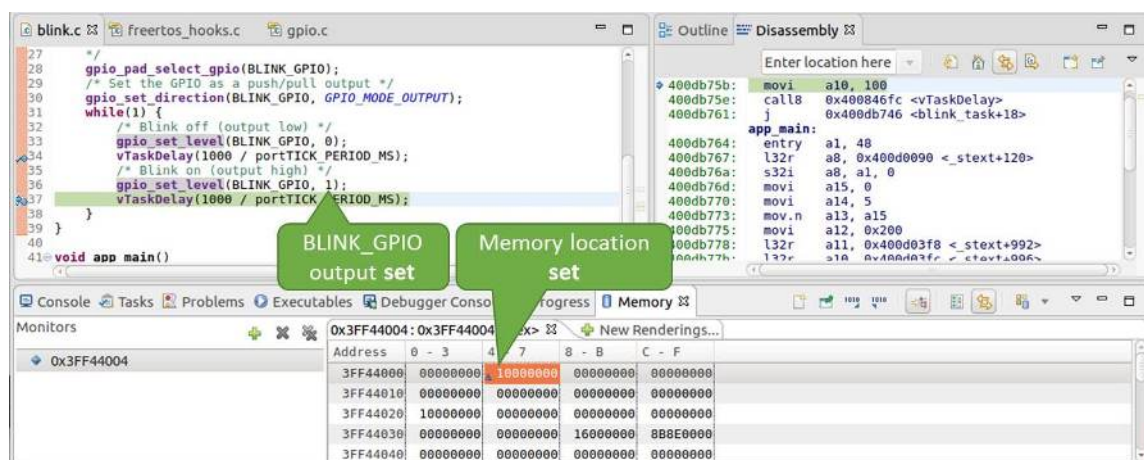


图 19: 观察内存地址 0x3FF44004 处的某个比特被置高

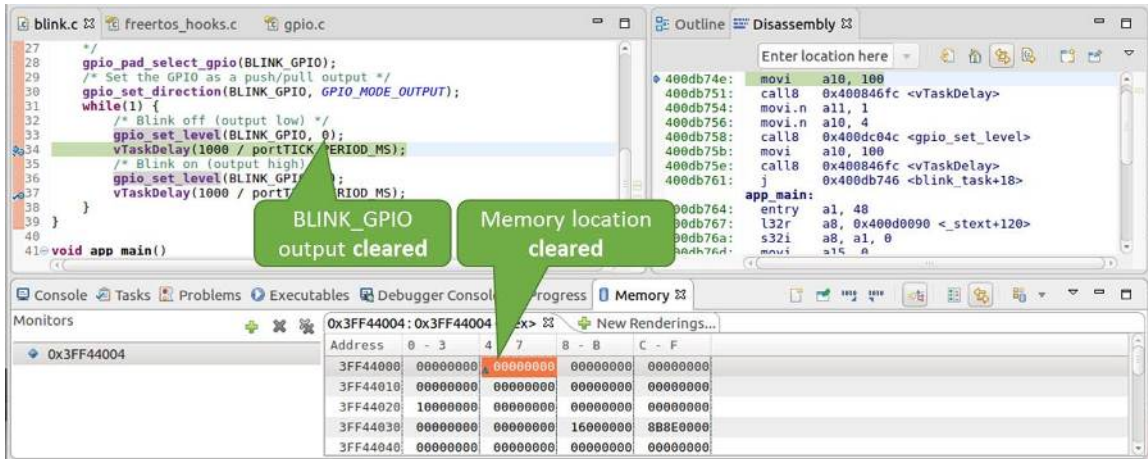


图 20: 观察内存地址 0x3FF44004 处的某个比特被置低

**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。

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。

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，然后按照在命令行中使用 GDB 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);
(gdb)
```

### 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



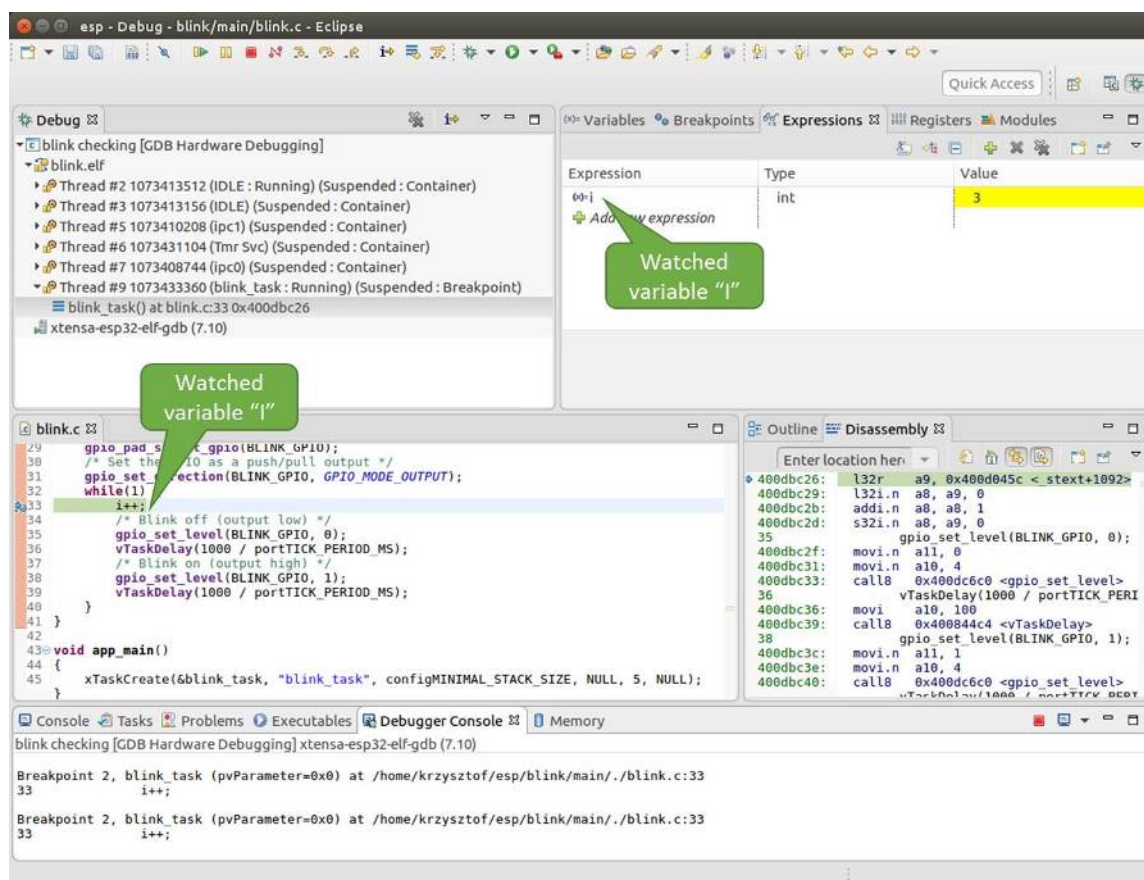


图 21: 观察程序变量 “i”

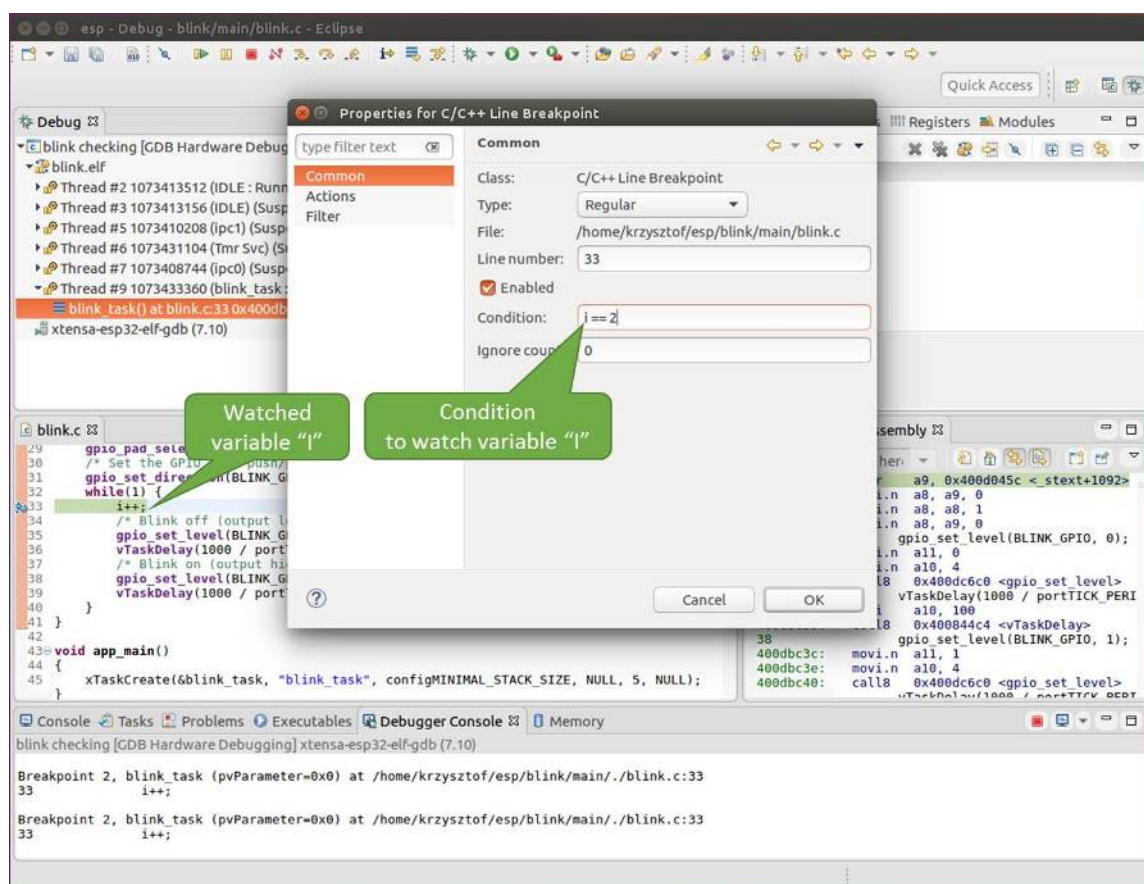


图 22: 设置条件断点

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，调试器会打印出停止点 (`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, ↵
↵5, NULL);
44     }
(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
329         REG_CLR_BIT(TIMG_WDTCONFIG0_REG(0), TIMG_WDT_FLASHBOOT_MOD_EN_S);
```

(下页继续)

(续上页)

```

330     REG_CLR_BIT(RTC_CNTL_WDTCONFIG0_REG, RTC_CNTL_WDT_FLASHBOOT_MOD_EN);
331     #if !CONFIG_FREERTOS_UNICORE
332     // Wait for FreeRTOS initialization to finish on APP CPU, before
↳replacing its startup stack
333     while (port_xSchedulerRunning[1] == 0) {
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 }
(gdb)

```

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 1073413156 (IDLE) prvIdleTask (pvParameters=0x0)
   at /home/user-name/esp/esp-idf/components/freertos/./tasks.c:3282
   2  Thread 1073413512 (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一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  0x40083a85 in vPortCPUReleaseMutex (mux=<optimized out>) at /home/user-name/
↳esp/esp-idf/components/freertos/./port.c:415
#2  0x40083fc8 in vTaskSwitchContext () at /home/user-name/esp/esp-idf/components/
↳freertos/./tasks.c:2846
#3  0x4008532b in _frxt_dispatch ()
#4  0x4008395c in xPortStartScheduler () at /home/user-name/esp/esp-idf/components/
↳freertos/./port.c:222
#5  0x4000000c in ?? ()

```

(下页继续)

```
#6 0x4000000c in ?? ()
#7 0x4000000c in ?? ()
#8 0x4000000c in ?? ()
(gdb)
```

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，以便在特定某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：

```
(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：

```
(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=0x400DB6F8 (active) APP_CPU: PC=0x400D10D8
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`：

```
(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，这是因为在调试器启动时执行 `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：

```
(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。

在此之前，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：

```
(gdb) c
Continuing.
^CTarget halted. PRO_CPU: PC=0x400D0C00          APP_CPU: PC=0x400D0C00 (active)
[New Thread 1073433352]

Program received signal SIGINT, Interrupt.
[Switching to Thread 1073413512]
0x400d0c00 in esp_vApplicationIdleHook () at /home/user-name/esp/esp-idf/
↳components/esp32/./freertos_hooks.c:52
52             asm("waiti 0");
(gdb)
```

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。

**注解：**在 `MSYS2` 的 `shell` 中输入 **Ctrl+C** 并不会暂停目标的运行，而是会退出调试器。解决这个问题的方法可以通过[使用 Eclipse 来调试](#)或者参考[http://www.mingw.org/wiki/Workaround\\_for\\_GDB\\_Ctrl\\_C\\_Interrupt](http://www.mingw.org/wiki/Workaround_for_GDB_Ctrl_C_Interrupt)里的解决方案。

**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`”命令会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：

```
(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：

```
(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：

```
(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=0x400DC04C (active) APP_CPU: PC=0x400D1128
Target halted. PRO_CPU: PC=0x400DB75B (active) APP_CPU: PC=0x400D1128
37         vTaskDelay(1000 / portTICK_PERIOD_MS);
(gdb) n
Target halted. PRO_CPU: PC=0x400DB75E (active) APP_CPU: PC=0x400D1128
Target halted. PRO_CPU: PC=0x400846FC (active) APP_CPU: PC=0x400D1128
Target halted. PRO_CPU: PC=0x400DB761 (active) APP_CPU: PC=0x400D1128
Target halted. PRO_CPU: PC=0x400DB746 (active) APP_CPU: PC=0x400D1128
33         gpio_set_level(BLINK_GPIO, 0);
(gdb)
```

如果你输入 `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/
↳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 `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”命令无法跳过去子程序的原因](#) 文档以了解 `next` 命令的潜在局限。

**查看并设置内存** 使用命令 `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 `help x` 可以查看更多相关细节。与 `x` 命令配合使用的命令是 `set`，它允许你将值写入内存。

为了演示 `x` 和 `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](#) 中的 `IO_MUX` 和 `GPIO Matrix` 章节。

同样在 `blink.c` 项目文件中，在两个 `gpio_set_level` 语句的后面各设置一个断点。输入两次 `c` 命令后停止在断点处，然后输入 `x /1wx 0x3FF44004` 来显示 `GPIO_OUT_REG` 寄存器的值：

```
(gdb) c
Continuing.
Target halted. PRO_CPU: PC=0x400DB75E (active) APP_CPU: PC=0x400D1128
Target halted. PRO_CPU: PC=0x400DB74E (active) APP_CPU: PC=0x400D1128

Breakpoint 2, blink_task (pvParameter=0x0) at /home/user-name/esp/blink/main/./
↳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/./
↳blink.c:37
37         vTaskDelay(1000 / portTICK_PERIOD_MS);
(gdb) x /1wx 0x3FF44004
```

(下页继续)

(续上页)

```
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：

```
0x3ff44004: 0x00000000
...
0x3ff44004: 0x00000010
```

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
(gdb) set {unsigned int}0x3FF44004=0x000010
```

在输入 set {unsigned int}0x3FF44004=0x000010 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
(gdb)
```

这会在所有变量 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++;
(gdb)
```

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
(gdb)
```

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看是否确已修改）：

```
(gdb) set var i = 0
(gdb) p i
$3 = 0
(gdb)
```

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) 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=0x400DB753 (active) APP_CPU: PC=0x400D112C
Target halted. PRO_CPU: PC=0x400DB755 (active) APP_CPU: PC=0x400D112C
Target halted. PRO_CPU: PC=0x400DB753 (active) APP_CPU: PC=0x400D112C

Breakpoint 3, blink_task (pvParameter=0x0) at /home/user-name/esp/blink/main/./
↪blink.c:34
34      gpio_set_level(BLINK_GPIO, 0);
(gdb)
```

**获得命令的帮助信息** 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 <http://darkdust.net/files/GDB%20Cheat%20Sheet.pdf>。虽然不是所有命令都适用于嵌入式环境，但还是会有所裨益。

**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$
```

## 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。

**可用的断点和观察点** ESP32 调试器支持 2 个硬件断点和 64 个软件断点。硬件断点是由 ESP32 芯片内部的逻辑电路实现的，能够设置在代码的任何位置：闪存或者 IRAM 的代码区域。除此以外，OpenOCD 实

现了两种软件断点：闪存断点（最多 32 个）和 IRAM 断点（最多 32 个）。目前 GDB 无法在闪存中设置软件断点，因此除非解决此限制，否则这些断点只能由 OpenOCD 模拟为硬件断点。（详细信息可以参阅[下面](#)）。ESP32 还支持 2 个观察点，所以可以观察两个变量的变化或者通过 GDB 命令 `watch myVariable` 来读取变量的值。请注意 `menuconfig` 中的 `CONFIG_FREERTOS_WATCHPOINT_END_OF_STACK` 选项会使用第二个观察点，如果你想在 OpenOCD 或者 GDB 中再次尝试使用这个观察点，可能不会得到预期的结果。详情请查看 `menuconfig` 中的帮助文档。

**关于断点的补充知识** 使用软件闪存模拟部分硬件断点的意思就是当使用 GDB 命令 `hb myFunction` 给某个函数设置硬件断点时，如果该函数位于闪存中，并且此时还有可用的硬件断点，那调试器就会使用硬件断点，否则就使用 32 个软件闪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，在这种情况下，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闪存断点。

**闪存映射 vs 软件闪存断点** 为了在闪存中设置或者清除软件断点，OpenOCD 需要知道它们在闪存中的地址。为了完成从 ESP32 的地址空间到闪存地址的转换，OpenOCD 使用闪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，并且特定于写入闪存的每一个应用程序的映像。因此，为了支持软件闪存断点，OpenOCD 需要知道待调试的应用程序映像在闪存中的位置。默认情况下，OpenOCD 会在 `0x8000` 处读取分区表并使用第一个找到的应用程序映像的映射，但是也可能存在无法工作的情况，比如分区表不在标准的闪存位置，甚至可能有多个映像：一个出厂映像和两个 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闪存中的具体位置。该命令具有以下格式：

```
esp32 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`。

**注解：**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：

```
openocd -f interface/ftdi/esp32_devkitj_v1.cfg -f board/esp-wroom-32.cfg -c
"init; halt; esp32 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。当两个断点都已经被使用时，`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_ESP32_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_FREERTOS_WATCHPOINT_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 `idf.py menuconfig`。

**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[针对特定目标的 OpenOCD 配置](#)。

**在 OpenOCD 的配置文件中设置 SPI 闪存的工作电压** ESP32 的 MTDI 引脚是用于 JTAG 通信的四个引脚之一，同时也是 ESP32 的 bootstrapping 引脚。上电时，ESP32 会在 MTDI 引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 MTDI 引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 MTDI 引脚为高电平，则稳压器会被设置为 1.8 V。MTDI 引脚通常需要一个上拉电阻或者直接使能内部的弱下拉电阻（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 ESP32-WROOM-32 模组的 `boards\esp-wroom-32.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 模组的规格书，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`。大多数的 WROOM 模组使用 3.3 V 的闪存芯片，但是 WROVER 模组使用 1.8 V 的闪存芯片。

**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 仅使用 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` — `monitor (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 `flushregs` 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。

**针对特定目标的 OpenOCD 配置** OpenOCD 需要知道当前使用的 JTAG 适配器的类型，以及其连接的目标板和处理器的类型。为此，请使用位于 OpenOCD 安装目录下 `share/openocd/scripts/interface` 和 `share/openocd/scripts/board` 文件夹中现有的配置文件。

例如，如果使用板载 ESP-WROOM-32 模组的 ESP-WROVER-KIT 开发板（详见 [ESP-WROVER-KIT V1 / ESP32 DevKitJ V1](#)），请使用以下配置文件：

- `interface/ftdi/esp32_devkitj_v1.cfg`
- `board/esp-wroom-32.cfg`

当然也可以使用自定义的配置文件，建议在已有配置文件的基础上进行修改，以匹配你的硬件。下面列举一些常用的板级配置参数。

### 适配器的时钟速度

```
adapter_khz 20000
```

请参阅 [优化 JTAG 的速度](#) 以获取有关如何设置此值的指导。

### 单核调试

```
set ESP32_ONLYCPU 1
```

如果是双核调试，请注释掉这一行。

### 禁用 RTOS 支持

```
set ESP32_RTOS none
```

如果要支持 RTOS，请注释掉这一行。

### ESP32 的 SPI 闪存芯片的电源电压

```
set ESP32_FLASH_VOLTAGE 1.8
```

如果 SPI 闪存芯片的电源电压为 3.3 V，请注释掉这一行，更多信息请参阅：[在 OpenOCD 的配置文件中设置 SPI 闪存的工作电压](#)。

### ESP32 的目标配置文件

```
source [find target/esp32.cfg]
```

**注解：**除非你熟悉 OpenOCD 内部的工作原理，否则请不要更改 `source [find target/esp32.cfg]` 这一行。

目前 `target/esp32.cfg` 仍然是 ESP32 目标（`esp108` 和 `esp32`）的唯一配置文件。支持的配置矩阵如下所示：

Dual/single	RTOS	Target used
dual	FreeRTOS	esp32
single	FreeRTOS	esp108 (*)
dual	none	esp108
single	none	esp108

(\*)—我们计划修复此问题，并在后续提交中添加对 `esp32` 目标的单核调试的支持。

更多信息，请查看 `board/esp-wroom-32.cfg` 配置文件的注释部分。

**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 引脚用于其他功能** 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 引脚，那么 JTAG 的操作可能会受到干扰。ESP32 JTAG 使用以下引脚：

	ESP32 JTAG Pin	JTAG Signal
1	MTDO / GPIO15	TDO
2	MTDI / GPIO12	TDI
3	MTCK / GPIO13	TCK
4	MTMS / GPIO14	TMS

如果用户应用程序更改了 JTAG 引脚的配置，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 引脚重新配置为其他功能或者用户忘记将 `Vtar` 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录：

```

cpu0: xtensa_resume (line 431): DSR (FFFFFFFF) indicates target still busy!
cpu0: xtensa_resume (line 431): DSR (FFFFFFFF) indicates DIR instruction generated.
↳an exception!
cpu0: xtensa_resume (line 431): DSR (FFFFFFFF) indicates DIR instruction generated.
↳an overrun!
cpu1: xtensa_resume (line 431): DSR (FFFFFFFF) indicates target still busy!
cpu1: xtensa_resume (line 431): DSR (FFFFFFFF) indicates DIR instruction generated.
↳an exception!
cpu1: xtensa_resume (line 431): DSR (FFFFFFFF) indicates DIR instruction generated.
↳an overrun!

```

**JTAG with Flash Encryption or Secure Boot** By default, enabling Flash Encryption and/or Secure Boot will disable JTAG debugging. On first boot, the bootloader will burn an eFuse bit to permanently disable JTAG at the same time it enables the other features.

The project configuration option `CONFIG_SECURE_BOOT_ALLOW_JTAG` will keep JTAG enabled at this time, removing all physical security but allowing debugging. (Although the name suggests Secure Boot, this option can be applied even when only Flash Encryption is enabled).

However, OpenOCD may attempt to automatically read and write the flash in order to set *software breakpoints*. This has two problems:

- Software breakpoints are incompatible with Flash Encryption, OpenOCD currently has no support for encrypting or decrypting flash contents.
- If Secure Boot is enabled, setting a software breakpoint will change the digest of a signed app and make the signature invalid. This means if a software breakpoint is set and then a reset occurs, the signature verification will fail on boot.

To disable software breakpoints while using JTAG, add an extra argument `-c 'set ESP_FLASH_SIZE 0'` to the start of the OpenOCD command line. For example:

```
openocd -c 'set ESP_FLASH_SIZE 0' -f board/esp32-wrover-kit-3.3v.cfg
```

**注解:** For the same reason, the ESP-IDF app may fail bootloader verification of app signatures, when this option is enabled and a software breakpoint is set.

**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相关信息。
  - d. 操作系统是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简单示例工程，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，因而再次遇到同样问题时，更容易复现。
3. 在启动命令中添加额外的参数来输出调试日志。

OpenOCD 端：

```
openocd -l openocd_log.txt -d 3 -f interface/ftdi/esp32_devkitj_v1.cfg
↳-f board/esp-wroom-32.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 `-d 3`）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 -d 3 -f interface/ftdi/esp32_devkitj_v1.cfg -f board/esp-wroom-
↳32.cfg 2>&1 | tee openocd.log
```

Debugger 端：

```
xtensa-esp32-elf-gdb -ex "set remotelogfile gdb_log.txt" <all other_
↳options>
```

也可以将命令 `remlotlogfile 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。

## 应用层跟踪库

**概述** 为了分析应用程序的行为，IDF 提供了一个有用的功能：应用层跟踪。这个功能以库的形式提供，可以通过 `menuconfig` 开启。此功能使得用户可以在程序运行开销很小的前提下，通过 JTAG 接口在主机和 ESP32 之间传输任意数据。

开发人员可以使用这个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. 收集应用程序特定的数据，具体请参阅[特定应用程序的跟踪](#)
2. 轻量级的日志记录，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](#)

使用 JTAG 接口的跟踪组件工作示意图：

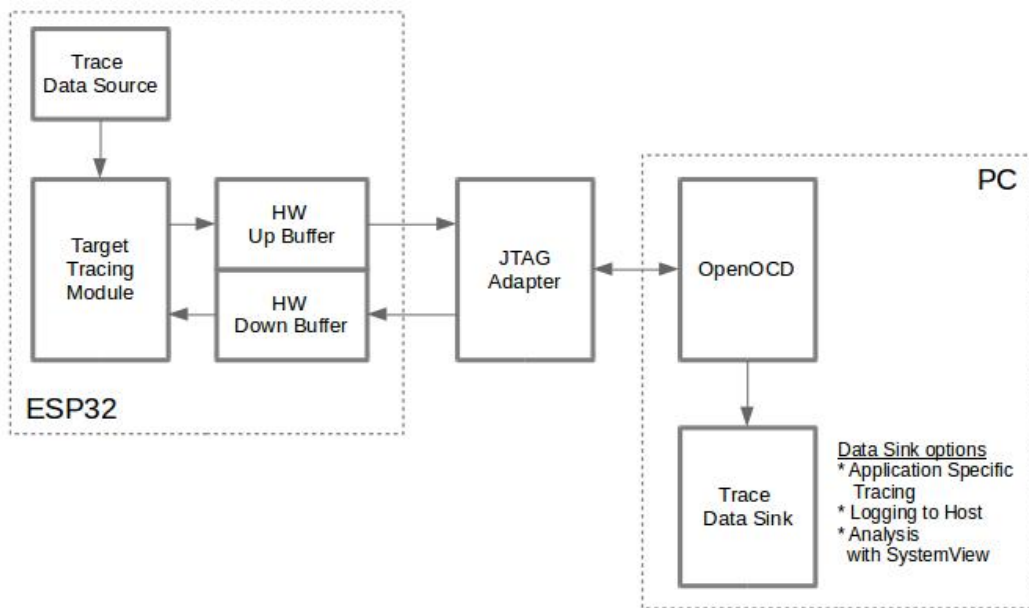


图 23: 使用 JTAG 接口的跟踪组件

**运行模式** 该库支持两种操作模式：

**后验模式：** 这是默认的模式，该模式不需要和主机进行交互。在这种模式下，跟踪模块不会检查主机是否已经从 `HW UP BUFFER` 缓冲区读走所有数据，而是直接使用新数据覆盖旧数据。该模式在用户仅对最新的跟踪数据感兴趣时会很有用，例如分析程序在崩溃之前的行为。主机可以稍后根据用户的请求来读取数据，例如通过特殊的 OpenOCD 命令（假如使用了 JTAG 接口）。

**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` 之前会检查其中是否有足够的空间，并在必要的时候等待主机读取数据并释放足够的内存。用户会将最长的等待时间作为超时时间参数传递给相应的 API 函数，如果超时时间是个有限值，那么应

用程序有可能会因为超时而将待写的的数据丢弃。尤其需要注意，如果在讲究时效的代码中（如中断处理函数，操作系统调度等）指定了无限的超时时间，那么系统会产生故障。为了避免丢失此类关键数据，开发人员可以通过在 `menuconfig` 中开启 `CONFIG_ESP32_APPTRACE_PENDING_DATA_SIZE_MAX` 选项来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过硬件接口的能力时，它也无能为力。

**配置选项与依赖项** 使用此功能需要在主机端和目标端做相应的配置：

1. **主机端：** 应用程序跟踪是通过 JTAG 来完成的，因此需要在主机上安装并运行 OpenOCD。相关信息请参阅 [JTAG Debugging](#)。
2. **目标端：** 在 `menuconfig` 中开启应用程序跟踪功能。 `Component config > Application Level Tracing` 菜单允许选择跟踪数据的传输目标（具体用于传输的硬件接口），选择任一非 None 的目标都会自动开启 `CONFIG_ESP32_APPTRACE_ENABLE` 这个选项。

**注解：** 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_ESP32_APPTRACE_POSTMORTEM_FLUSH_THRESH`)。由于在 JTAG 上工作的性质，此选项是必选项。在该模式下，跟踪数据以 16 KB 数据块的形式曝露给主机。在后验模式中，当一个块被填充时，它会曝露给主机，而之前的块会变得不可用。换句话说，跟踪数据以 16 KB 的粒度进行覆盖。在发生 panic 的时候，当前输入块的最新数据将会被曝露给主机，主机可以读取它们以进行后续分析。如果系统发生 panic 的时候仍有少量数据还没来得及曝光给主机，那么之前收集的 16 KB 的数据将丢失，主机只能看到非常少的最新的跟踪部分，它可能不足以用来诊断问题所在。此 `menuconfig` 选项允许避免此类情况，它可以控制在发生 panic 时刷新数据的阈值，例如用户可以确定它需要不少于 512 字节的最新跟踪数据，所以如果在发生 panic 时待处理的数据少于 512 字节，它们不会被刷新，也不会覆盖之前的 16 KB。该选项仅在后验模式和 JTAG 工作时有意义。
2. *Timeout for flushing last trace data to host on panic* (`CONFIG_ESP32_APPTRACE_ONPANIC_HOST_FLUSH_TMO`)。该选项仅在流模式下起作用，它控制跟踪模块在发生 panic 时等待主机读取最新数据的最长时间。

**如何使用这个库** 该库提供了用于在主机和 ESP32 之间传输任意数据的 API。当在 `menuconfig` 中启用时，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待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，用户必须要执行以下几个步骤。

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。

```
#include "esp_app_trace.h"
...
char buf[] = "Hello World!";
esp_err_t res = esp_apptrace_write(ESP_APPTRACE_DEST_TRAX, buf, strlen(buf),
↳ESP_APPTRACE_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。

```

#include "esp_app_trace.h"
...
int number = 10;
char *ptr = (char *)esp_appttrace_buffer_get(ESP_APPTRACE_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_appttrace_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;
}

```

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。

```

#include "esp_app_trace.h"
...
char buf[32];
char down_buf[32];
size_t sz = sizeof(buf);

/* config down buffer */
esp_appttrace_down_buffer_config(down_buf, sizeof(down_buf));
/* check for incoming data and read them if any */
esp_err_t res = esp_appttrace_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_appttrace_read()` 函数使用 `memcpy` 来把主机端的数据复制到用户缓存区。在某些情况下，使用 `esp_appttrace_down_buffer_get()` 和 `esp_appttrace_down_buffer_put()` 函数可能更为理想。它们允许开发人员占用一块读缓冲区并就地有关处理操作。下面的代码片段展示了如何执行此操作。

```

#include "esp_app_trace.h"
...
char down_buf[32];
uint32_t *number;
size_t sz = 32;

/* config down buffer */
esp_appttrace_down_buffer_config(down_buf, sizeof(down_buf));
char *ptr = (char *)esp_appttrace_down_buffer_get(ESP_APPTRACE_DEST_TRAX, &sz, _
↳100/*tmo in us*/);
if (ptr == NULL) {
    ESP_LOGE(TAG, "Failed to get buffer!");
    return ESP_FAIL;
}
if (sz > 4) {
    number = (uint32_t *)ptr;
}

```

(下页继续)



```

    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，在终端执行如下命令 `telnet <occd_host> 4444`。如果在运行 OpenOCD 的同一台机器上打开 telnet 会话，您可以使用 `localhost` 替换上面命令中的 `<occd_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. 最后一步是处理接收到的数据，由于数据格式由用户定义，因此处理阶段超出了本文档的范围。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，并且会替 API 的调用者（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，有可能发生主机读取还未准备好的用户数据的情况。为了处理这样的情况，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长 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。

下面是 OpenOCD 应用程序跟踪命令的使用说明。

---

**注解：**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：

```
esp32 apptrace [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：

```
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。默认情况下是 -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)。还可以将其设置为比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, 否则命令等待目标停止 (复位, 打断点等), 然后自动恢复它并开始跟踪。默认值为 0。

**skip\_size** 开始时要跳过的字节数, 默认为 0。

**注解:** 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 (不是 `telnet` 会话窗口) 按下 `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。

```
esp32 appttrace start file://trace.log 1 2048 5 0 0
```

跟踪数据会被检索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 秒内都没有新的数据, 那么该过程就会停止。

**注解:** 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, 则目标可能在超时之前没有发送足够的给 OpenOCD 来清空缓冲区。增加超时时间或者使用函数 `esp_appttrace_flush()` 以特定间隔刷新数据都可以解决这个问题。

2. 在非阻塞模式下无限地检索跟踪数据。

```
esp32 appttrace start file://trace.log 1 -1 -1 0 0
```

对收集数据的大小没有限制, 并且没有设置任何超时时间。可以通过在 OpenOCD 的 `telnet` 会话窗口中发送 `esp32 appttrace stop` 命令, 或者在 OpenOCD 窗口中使用快捷键 `Ctrl+C` 来停止此过程。

3. 检索跟踪数据并无限期保存。

```
esp32 appttrace start file://trace.log 0 -1 -1 0 0
```

在跟踪停止之前, OpenOCD 的 `telnet` 会话窗口将不可用。要停止跟踪, 请在 OpenOCD 的窗口中使用快捷键 `Ctrl+C`。

4. 等待目标停止, 然后恢复目标的操作并开始检索数据。当收集满 2048 字节的数据后就停止:

```
esp32 appttrace start file://trace.log 0 2048 -1 1 0
```

想要复位后立即开始跟踪, 请使用 OpenOCD 的 `reset halt` 命令。

**记录日志到主机** 记录日志到主机是 IDF 的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, 这个功能的优势在于它减少了本地的工作量, 而将大部分工作转移到了主机端。

IDF 的日志库会默认使用类 `vprintf` 的函数将格式化的字符串输出到专用的 UART。一般来说, 它涉及到以下几个步骤:

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。

虽然可以将类 `vprintf` 函数优化到一定程度, 但是上述步骤在任何情况下都是必须要执行的, 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。在最差的情况下, 程序会无法正常工作, 最终导致报错甚至挂起。

解决此类问题的可能方法是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_apptrace_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和参数的完全解析，而仅仅计算传递的参数数量，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. 不支持大小超过 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. `printf` 参数最多 256 个。

**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，正如前面提到过的，这个函数是 `esp_apptrace_vprintf`，它会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](#) 章节中第 2-5 步骤中的说明进行操作。
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** 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 是一种实时记录和可视化工具，用来分析应用程序运行时的行为。

---

**注解：**目前，基于 IDF 的应用程序能够以文件的形式生成与 SystemView 格式兼容的跟踪信息，并可以使用 SystemView 工具软件打开。但是还无法使用该工具控制跟踪的过程。

---

**如何使用** 若需使用这个功能，需要在 `menuconfig` 中开启 `CONFIG_SYSVIEW_ENABLE` 选项，具体路径为：`Component config > Application Level Tracing > FreeRTOS SystemView Tracing`。在同一个菜单栏下还开启了其他几个选项：

1. *ESP32 timer to use as SystemView timestamp source* (`CONFIG_SYSVIEW_TS_SOURCE`) 选择 SystemView 事件使用的时间戳来源。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。
2. 可以单独启用或禁用的 SystemView 事件集合 (`CONFIG_SYSVIEW_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

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。

### OpenOCD SystemView 跟踪命令选项 命令用法：

```
esp32 sysview [start <options>] | [stop] | [status]
```

自命令：

**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 <outfile1> [outfile2] [poll_period [trace_size [stop_tmo]]]
```

**outfile1** 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。

**outfile2** 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）。

**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）。

---

**注解：**如果 poll\_period 为 0，则在跟踪停止之前，OpenOCD 的 telnet 命令行将不可用。你需要通过复位板卡或者在 OpenOCD 的窗口（不是 telnet 会话窗口）输入 Ctrl+C 命令来手动停止它。另一个办法是设置 trace\_size 然后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“app-cpu.SVdat”中。这些文件会被保存在“openocd-esp32”目录中。

```
esp32 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 esp32 apptrace stop 命令，或者也可以在 OpenOCD 窗口中按下 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 窗口按下 Ctrl+C。

**数据可视化** 收集到跟踪数据后，用户可以使用特殊的工具来可视化结果并分析程序的行为。遗憾的是，SystemView 不支持从多个核心进行跟踪。所以当追踪双核模式下的 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。

在工具中单独分析每个核的跟踪数据是比较棘手的，幸运的是，Eclipse 中有一款 *Impulse* 的插件可以加载多个跟踪文件，并且可以在同一个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 并使用它可视化来自单个核心的跟踪数据，请参阅 [官方教程](#)。

**注解:** 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 IDF 特定文件的内容。

---

**配置 Impulse 实现双核跟踪** 在安装好 Impulse 插件后, 先确保它能够在单独的选项卡中成功加载每个核心的跟踪文件, 然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, 然后双击它。
2. 在“Signal Ports”视图中, 右键单击“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, 按下确定保存设置。
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, 数据将会被加载。
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](#)。

**注解:** 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很奇怪), 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。

---

## 4.14 Bootloader

Bootloader performs the following functions:

1. Minimal initial configuration of internal modules;
2. Select the application partition to boot, based on the partition table and ota\_data (if any);
3. Load this image to RAM (IRAM & DRAM) and transfer management to it.

Bootloader is located at the address `0x1000` in the flash.

### 4.14.1 Bootloader compatibility

It is recommended to update to newer *versions of ESP-IDF*: when they are released. The OTA (over the air) update process can flash new apps in the field but cannot flash a new bootloader. For this reason, the bootloader supports booting apps built from newer versions of ESP-IDF.

The bootloader does not support booting apps from older versions of ESP-IDF. When updating ESP-IDF manually on an existing product that might need to downgrade the app to an older version, keep using the older ESP-IDF bootloader binary as well.

**注解:** If testing an OTA update for an existing product in production, always test it using the same ESP-IDF bootloader binary that is deployed in production.

---

#### Before ESP-IDF V2.1

Bootloaders built from very old versions of ESP-IDF (before ESP-IDF V2.1) perform less hardware configuration than newer versions. When using a bootloader from these early ESP-IDF versions and building a new app, enable the

config option `CONFIG_ESP32_COMPATIBLE_PRE_V2_1_BOOTLOADERS`.

### Before ESP-IDF V3.1

Bootloaders built from versions of ESP-IDF before V3.1 do not support MD5 checksums in the partition table binary. When using a bootloader from these ESP-IDF versions and building a new app, enable the config option `CONFIG_ESP32_COMPATIBLE_PRE_V3_1_BOOTLOADERS`.

### SPI Flash Configuration

Each ESP-IDF application or bootloader .bin file contains a header with `CONFIG_ESPTOOLPY_FLASHMODE`, `CONFIG_ESPTOOLPY_FLASHFREQ`, `CONFIG_ESPTOOLPY_FLASHSIZE` embedded in it. These are used to configure the SPI flash during boot.

The :first stage bootloader in ROM reads the second stage bootloader header from flash and uses these settings to load it. However, at this time the system clock speed is lower than configured and not all flash modes are supported. When the second stage bootloader then runs and re-configures the flash, it reads values from the currently selected app binary header not the bootloader header. This allows an OTA update to change the SPI flash settings in use.

Bootloaders prior to ESP-IDF V4.0 used the bootloader's own header to configure the SPI flash, meaning these values could not be changed in an update. To maintain compatibility with older bootloaders, the app re-initializes the flash settings during app startup using the configuration found in the app header.

## 4.14.2 FACTORY reset

The user can write a basic working firmware and load it into the factory partition. Next, update the firmware via OTA (over the air). The updated firmware will be loaded into an OTA app partition slot and the OTA data partition is updated to boot from this partition. If you want to be able to roll back to the factory firmware and clear the settings, then you need to set `CONFIG_BOOTLOADER_FACTORY_RESET`. The factory reset mechanism allows to reset the device to factory settings:

- Clear one or more data partitions.
- Boot from “factory” partition.

`CONFIG_BOOTLOADER_DATA_FACTORY_RESET` allows customers to select which data partitions will be erased when the factory reset is executed. Can specify the names of partitions through comma-delimited with optional spaces for readability. (Like this: “nvs, phy\_init, nvs\_custom, …” ). Make sure that the name specified in the partition table and here are the same. Partitions of type “app” cannot be specified here.

`CONFIG_BOOTLOADER_OTA_DATA_ERASE` - the device will boot from “factory” partition after a factory reset. The OTA data partition will be cleared.

`CONFIG_BOOTLOADER_NUM_PIN_FACTORY_RESET` - number of the GPIO input for factory reset uses to trigger a factory reset, this GPIO must be pulled low on reset to trigger this.

`CONFIG_BOOTLOADER_HOLD_TIME_GPIO` - this is hold time of GPIO for reset/test mode (by default 5 seconds). 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.

Partition table.:

```
# Name, Type, SubType, Offset, Size, Flags
# Note: if you have increased the bootloader size, make sure to update the offsets.
↳to avoid overlap
nvs, data, nvs, 0x9000, 0x4000
otadata, data, ota, 0xd000, 0x2000
phy_init, data, phy, 0xf000, 0x1000
factory, 0, 0, 0x10000, 1M
test, 0, test, , 512K
```

(下页继续)

ota_0,	0,	ota_0,	,	512K
ota_1,	0,	ota_1,	,	512K

### 4.14.3 Boot from TEST firmware

The user can write a special firmware for testing in production, and run it as needed. The partition table also needs a dedicated partition for this testing firmware (See *partition table*). To trigger a test app you need to set `CONFIG_BOOTLOADER_APP_TEST`.

`CONFIG_BOOTLOADER_NUM_PIN_APP_TEST` - GPIO number to boot TEST partition. 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 normally configured application will boot (factory or any OTA slot).

`CONFIG_BOOTLOADER_HOLD_TIME_GPIO` - this is hold time of GPIO for reset/test mode (by default 5 seconds). 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.

### 4.14.4 Customer bootloader

The current bootloader implementation allows the customer to override it. To do this, you must copy the folder `/esp-idf/components/bootloader` and then edit `/your_project/components/bootloader/subproject/main/bootloader_main.c`. In the bootloader space, you can not use the drivers and functions from other components. If necessary, then the required functionality should be placed in the folder `bootloader` (note that this will increase its size). It is necessary to monitor its size because there can be overlays in memory with a partition table leading to damage. At the moment the bootloader is limited to the partition table from the address `0x8000`.

## 4.15 分区表

### 4.15.1 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据（例如校准数据、文件系统数据、参数存储器数据等）。因此，我们需要引入分区表的概念。

具体来说，ESP32 在 flash 的默认偏移地址 `0x8000` 处烧写一张分区表。该分区表的长度为 `0xC00` 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，最简单的方法就是用 `idf.py menuconfig` 选择一张预定义的分区表：

- “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`，即可以打印当前使用分区表的信息摘要。

### 4.15.2 内置分区表

以下是 “Single factory app, no OTA” 选项的分区表信息摘要：

```
# Espressif ESP32 Partition Table # Name, Type, SubType, Offset, Size, Flags nvs, data, nvs, 0x9000,
0x6000, phy_init, data, phy, 0xf000, 0x1000, factory, app, factory, 0x10000, 1M,
```

- flash 的 0x10000 (64KB) 偏移地址处存放一个标记为 “factory” 的二进制应用程序，且 Bootloader 将默认加载这个应用程序。
- 分区表中还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：

```
# Espressif ESP32 Partition Table # Name, Type, SubType, Offset, Size, Flags nvs, data, nvs, 0x9000,
0x4000, otadata, data, ota, 0xd000, 0x2000, phy_init, data, phy, 0xf000, 0x1000, factory, app, factory,
0x10000, 1M, ota_0, app, ota_0, 0x110000, 1M, ota_1, app, ota_1, 0x210000, 1M,
```

- 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 升级时候需要的数据。Bootloader 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。

### 4.15.3 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，但是在 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：

```
# Name, Type, SubType, Offset, Size, Flags nvs, data, nvs, 0x9000, 0x4000 otadata, data, ota, 0xd000,
0x2000 phy_init, data, phy, 0xf000, 0x1000 factory, app, factory, 0x10000, 1M ota_0, app, ota_0, , 1M
ota_1, app, ota_1, , 1M nvs_key, data, nvs_keys, , 0x1000
```

- 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 个字符（之后的内容将被截断）。该字段对 ESP32 并不是特别重要。

#### Type 字段

Type 字段可以指定为 app (0) 或者 data (1)，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（预留给 esp-idf 的核心功能）。

如果您的应用程序需要保存数据，请在 0x40-0xFE 内添加一个自定义分区类型。

注意，bootloader 将忽略 app (0) 和 data (1) 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 Type 有关。目前，esp-idf 仅仅规定了 “app” 和 “data” 两种子类型。

- 当 Type 定义为 app 时，SubType 字段可以指定为 factory (0)，ota\_0 (0x10) …ota\_15 (0x1F) 或者 test (0x20)。
  - factory (0) 是默认的 app 分区。Bootloader 将默认加在该应用程序。但如果存在类型为 data/ota 分区，则 Bootloader 将加载 data/ota 分区中的数据，进而判断启动哪个 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，Bootloader 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 (0x2) 为预留 app 子类型，用于工厂测试过程。注意，目前，esp-idf 并不支持这种子类型。
- 当 Type 定义为 data 时，SubType 字段可以指定为 ota (0)，phy (1)，nvs (2) 或者 nvs\_keys (4)。
  - 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（直接将此分区删掉）。
    - \* 如果需从此分区加载 phy 初始化数据，请运行 `idf.py menuconfig`，并且使能 [CONFIG\\_ESP32\\_PHY\\_INIT\\_DATA\\_IN\\_PARTITION](#) 选项。此时，您还需要手动将 phy 初始化数据烧至设备 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 密钥分区。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 4096 字节。

其它数据子类型已预留给 esp-idf 的未来使用。

## Offset 和 Size 字段

分区若为指定偏移地址，则会紧跟着前一个分区之后开始。若此分区为首个分区，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，则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 Encryption](#) 功能，则该分区将会被加密。

---

**注解：** app 分区始终会被加密，不管 Flags 字段是否设置。

---

### 4.15.4 生成二进制分区表

烧写到 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 `idf.py menuconfig` 指定了分区表 CSV 文件的名称，然后执行 `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：

```
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（在运行 `idf.py partition_table` 时，我们正是这样打印上文展示的信息摘要的）：

```
python gen_esp32part.py binary_partitions.bin
```

## 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。

注意，一些版本较老的 bootloader 无法支持 MD5 校验，如果发现 MD5 校验和则将报错 `invalid magic number 0xebeb`。此时，用户可通过 `gen_esp32part.py` 的 `--disable-md5sum` 选项或者 `menuconfig` 的 `CONFIG_PARTITION_TABLE_MD5` 选项关闭 MD5 校验。

### 4.15.5 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。

---

**注解：**分区表的更新并不会擦除根据之前分区表存储的数据。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。

---

## 4.16 Secure Boot

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.

---

**重要：**Enabling secure boot limits your options for further updates of your ESP32. Make sure to read this document thoroughly and understand the implications of enabling secure boot.

---

### 4.16.1 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 about efuse, see Chapter 11 “eFuse Controller” in the Technical Reference Manual.
- 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.

### 4.16.2 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](#).
6. When running in secure boot mode, the software bootloader uses the secure boot signing key (the public key of which is embedded in the bootloader itself, and therefore validated as part of the bootloader) to verify the signature appended to all subsequent partition tables and app images before they are booted.

### 4.16.3 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. See ESP32 Technical Reference Manual section 20.3.1.3 *System Parameter coding\_scheme* for more details. 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.

### 4.16.4 Bootloader Size

When secure boot is enabled the bootloader app binary `bootloader.bin` may exceed the default bootloader size limit. This is especially likely if flash encryption is enabled as well. The default size limit is 0x7000 (28672) bytes (partition table offset 0x8000 - bootloader offset 0x1000).

If the bootloader becomes too large, the ESP32 will fail to boot - errors will be logged about either invalid partition table or invalid bootloader checksum.

Options to work around this are:

- Reduce *bootloader log level*. Setting log level to Warning, Error or None all significantly reduce the final binary size (but may make it harder to debug).
- Set *partition table offset* to a higher value than 0x8000, to place the partition table later in the flash. This increases the space available for the bootloader. If the *partition table* CSV file contains explicit partition offsets, they will need changing so no partition has an offset lower than `CONFIG_PARTITION_TABLE_OFFSET + 0x1000`. (This includes the default partition CSV files supplied with ESP-IDF.)

### 4.16.5 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`.

---

**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 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.

---

**注解:** `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.

---

**注解:** 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.

---

**注解:** 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).

### 4.16.6 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.

**注解:** 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_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.
2. Follow the steps shown above to choose a signing key file, and generate the key file.
3. 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` 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).
4. 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.

### 4.16.7 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_boot_signing_key.pem `
```

Remember that the strength of the secure boot system depends on keeping the signing key private.

### 4.16.8 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`:

```
espsecure.py sign_data --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 --keyfile PRIVATE_SIGNING_KEY --output SIGNED_BINARY_FILE_  
↳BINARY_FILE
```

### 4.16.9 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.

#### 4.16.10 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 Encryption Algorithm*.
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

Deterministic ECDSA as specified by [RFC 6979](#).

- 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
↪image-unsigned.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
```

### 4.16.11 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.

### 4.16.12 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 image should be generated by following instructions in *Remote Signing of Images*.

### 4.16.13 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 with Flash Encryption or Secure Boot* for more information about using JTAG Debugging with either Secure Boot or signed app verification enabled.

## 4.17 ULP 协处理器编程



### 4.17.1 ULP coprocessor instruction set

This document provides details about the instructions used by ESP32 ULP coprocessor assembler.

ULP 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 coprocessor's `JUMP`, `ST`, `LD` instructions which take register as an argument (jump address, store/load base address) expect the argument to be expressed in 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 to be expressed in 32-bit words. To account for this common use case, assembler will convert the address of label `loop` from bytes to words, when generating `MOVE` instruction, so 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`.

Similar considerations apply to `LD` and `ST` instructions. Consider the following code:

```
.global array
array: .long 0
       .long 0
       .long 0
       .long 0

       MOVE R1, array
```

(下页继续)

```

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 R1, R1, 2   // this increments address by 2 words (8 bytes)
ST R2, R1, 0    // write value of R2 into the third array element,
                // i.e. array[2]

```

### 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:

```

#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.

### NOP - no operation

#### Syntax NOP

**Operands** None

**Cycles** 2 cycle to execute, 4 cycles to fetch next instruction

**Description** No operation is performed. Only the PC is incremented.

**Example:**

```
1:    NOP
```

### ADD - Add to register

**Syntax** ADD *Rdst*, *Rsrc1*, *Rsrc2*

ADD *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 adds source register to another source register or to a 16-bit signed value and stores result to the destination register.

**Examples:**

```

1:   ADD R1, R2, R3           //R1 = R2 + R3
2:   Add R1, R2, 0x1234      //R1 = R2 + 0x1234
3:   .set value1, 0x03       //constant value1=0x03
     Add 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

```

### 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 16-bit signed value from a source register, and stores 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
     SUB R1, R2, label       //R1 = R2 - label
     ...
     label: nop              //definition of variable label

```

### AND - 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 logical AND of a source register and another source register or 16-bit signed value and stores 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
      AND R1, R2, label       //R1 = R2 & label
      ...
label:  nop                    //definition of variable label

```

### OR - 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 logical OR of a source register and another source register or 16-bit signed value and stores 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 logical shift to left of source register to number of bits from another source register or 16-bit signed value and store result to the destination register.

#### Examples:

```

1:      LSH R1, R2, R3        //R1 = R2 << R3
2:      LSH R1, R2, 0x03     //R1 = R2 << 0x03

```

(下页继续)

```

3:      .set value1, 0x03          //constant value1=0x03
      LSH R1, R2, value1          //R1 = R2 << value1

4:      .global label             //declaration of variable label
      LSH R1, R2, label           //R1 = R2 << label
      ...
label:  nop                       //definition of variable 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 logical shift to right of source register to number of bits from another source register or 16-bit signed value and store 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 move to destination register value from source register or 16-bit signed value.

Note that when a label is used as an immediate, the address of the label will be converted from bytes to words.

This is because LD, ST, and JUMP instructions expect the address register value to be expressed in words rather than bytes. To avoid using an extra instruction

**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
      ...
label:  nop                       //definition of label

```

**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* –10-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:

```
Mem[Rdst + offset / 4]{31:0} = {PC[10:0], 5'b0, Rsrc[15:0]}
```

The application can use higher 16 bits to determine which instruction in the ULP program has written any particular word into memory.

**Examples:**

```
1:      ST   R1, R2, 0x12           //MEM[R2+0x12] = R1
2:      .data                               //Data section definition
Addr1:  .word 123                       // Define label Addr1 16 bit
        .set  offs, 0x00                // Define constant offs
        .text                               //Text section definition
        MOVE R1, 1                       // R1 = 1
        MOVE R2, Addr1                   // R2 = Addr1
        ST   R1, R2, offs                // MEM[R2 + 0] = 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* –10-bit signed value, offset in bytes**Cycles** 4 cycles to execute, 4 cycles to fetch next instruction**Description** The instruction loads lower 16-bit half-word from memory with address *Rsrc*+*offset* into the destination register *Rdst*:

```
Rdst[15:0] = Mem[Rsrc + offset / 4][15:0]
```

**Examples:**

```
1:      LD   R1, R2, 0x12           //R1 = MEM[R2+0x12]
2:      .data                               //Data section definition
Addr1:  .word 123                       // Define label Addr1 16 bit
        .set  offs, 0x00                // Define constant offs
        .text                               //Text section definition
        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
      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 *LT*, *GE*, *LE* and *GT*: 2 cycles to execute, 2 cycles to fetch next instructionConditions *LE* and *GT* are implemented in the assembler using one **JUMPR** instructions:

```

// JUMPR target, threshold, GT is implemented as:

      JUMPR target, threshold+1, GE

// JUMPR target, threshold, LE is implemented as:

      JUMPR target, threshold + 1, LT

```

Conditions *EQ* is implemented in the assembler using two **JUMPR** instructions:

```

// JUMPR target, threshold, EQ is implemented as:

      JUMPR next, threshold + 1, GE

```

(下页继续)

```

        JUMPR target, threshold, GE
next:

```

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:

```

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** Conditions *LE, LT, GE*: 2 cycles to execute, 2 cycles to fetch next instruction

Conditions *EQ, GT* are implemented in the assembler using two **JUMPS** instructions:

```

// JUMPS target, threshold, EQ is implemented as:

        JUMPS next, threshold, LT
        JUMPS target, threshold, LE
next:

// JUMPS target, threshold, GT is implemented as:

        JUMPS next, threshold, LE
        JUMPS target, threshold, GE
next:

```

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++

```

(下页继续)



```
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 stage count register by 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 stage count register by given value.**Examples:**

```
1:    STAGE_DEC    10    // stage_cnt -= 10;

2:    // Down counting loop exaple
      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
```

**HALT –End the program****Syntax HALT****Operands** No operands

**Cycles** 2 cycles to execute

**Description** The instruction halts the ULP coprocessor and restarts ULP wakeup 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 ULP to RTC controller.

- If the SoC is in deep sleep mode, and ULP wakeup 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.

Note that before using WAKE instruction, ULP program may needs to wait until RTC controller is ready to wake up the main CPU. This is indicated using RTC\_CNTL\_RDY\_FOR\_WAKEUP bit of RTC\_CNTL\_LOW\_POWER\_ST\_REG register. If WAKE instruction is executed while RTC\_CNTL\_RDY\_FOR\_WAKEUP is zero, it has no effect (wake up does not occur).

**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.
```

## SLEEP –set ULP wakeup 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* 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

```

**TSENS –do measurement with temperature sensor****Syntax**

- **TSENS** *Rdst, Wait\_Delay*

**Operands**

- *Rdst* –Destination Register R[0..3], result will be stored to this register
- *Wait\_Delay* –number of cycles used to perform the measurement

**Cycles** 2 + *Wait\_Delay* + 3 \* TSENS\_CLK to execute, 4 cycles to fetch next instruction

**Description** The instruction performs measurement using TSENS and stores the result into a general purpose register.

**Examples:**

```

1:      TSENS     R1, 1000    // Measure temperature sensor for 1000 cycles,
                               // and store result to R1

```

**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* - selected PAD, SARADC Pad[Mux+1] is enabled

**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:**

```

1:      ADC       R1, 0, 1    // Measure value using ADC1 pad 2 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:**

```
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:  $R0 = \text{REG}[\text{Addr}][\text{High}:\text{Low}]$ .

This instruction can access registers in RTC\_CNTL, RTC\_IO, SENS, and RTC\_I2C peripherals. Address of the the register, as seen from the ULP, can be calculated from the address of the same register on the DPORT bus as follows:

$$\text{addr\_ulp} = (\text{addr\_dport} - \text{DR\_REG\_RTCCNTL\_BASE}) / 4$$

#### 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:  $\text{REG}[\text{Addr}][\text{High}:\text{Low}] = \text{data}$ .

This instruction can access registers in RTC\_CNTL, RTC\_IO, SENS, and RTC\_I2C peripherals. Address of the 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_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:

```
#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:

```
#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:

```
#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:

```
#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)
```

**4.17.2 Programming ULP coprocessor using C macros**

In addition to the existing binutils port for the ESP32 ULP coprocessor, it is possible to generate programs for the ULP by embedding assembly-like macros into an ESP32 application. Here is an example how this can be done:

```
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]
```

(下页继续)

```

    I_ADDR(R2, R0, R1),      // R2 <- R0 + R1
    I_ST(R2, R3, 2),        // R2 -> RTC_SLOW_MEM[R2 + 2]
    I_HALT()
};
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);

```

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 [ULP coprocessor instruction defines](#) section for descriptions of instructions and arguments they take.

**注解:** 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.

Load and store instructions use addresses expressed in 32-bit words. Address 0 corresponds to the first word of `RTC_SLOW_MEM` (which is address `0x50000000` as seen by the main CPUs).

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 the extra `ulp_insn_t` token which contains the label number.

Here is an example of using labels and branches:

```

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);

```

## 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.

### Return

- ESP\_OK on success
- ESP\_ERR\_NO\_MEM if auxiliary temporary structure can not be allocated
- one of ESP\_ERR\_ULP\_XXX if program is not valid or can not be loaded

**Parameters**

- 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

*esp\_err\_t* **ulp\_run** (uint32\_t entry\_point)

Run the program loaded into RTC memory.

**Return** ESP\_OK on success

**Parameters**

- entry\_point: entry point, expressed in 32-bit words

**Error codes****ESP\_ERR\_ULP\_BASE**

Offset for ULP-related error codes

**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)

**ULP coprocessor registers**

ULP co-processor has 4 16-bit general purpose registers. All registers have same functionality, with one exception. R0 register is used by some of the compare-and-branch instructions as a source register.

These definitions can be used for all instructions which require a register.

**R0**

general purpose register 0

**R1**

general purpose register 1

**R2**

general purpose register 2

**R3**

general purpose register 3

**ULP coprocessor instruction defines****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\_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\_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 [20:16] = 5' b1
- bits [15:0] are assigned the contents of `reg_val`

$$\text{RTC\_SLOW\_MEM}[\text{addr} + \text{offset\_}] = \{ 5' \text{ b0}, \text{insn\_PC}[10:0], \text{val}[15:0] \}$$
**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\_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\_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\_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.

**Defines****RTC\_SLOW\_MEM**

RTC slow memory, 8k size

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。

**4.17.3 安装工具链**

ULP 协处理器代码是用汇编语言编写的，并使用 [binutils-esp32ulp 工具链](#) 进行编译。

如果你已经按照[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你的开发环境中。

如果你的 ESP-IDF 仍在使用传统的基于 GNU Make 的构建系统，请参考[ULP Coprocessor Programming](#) 一文中的说明，完成工具链的安装。

**4.17.4 编译 ULP 代码**

若需要将 ULP 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 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：

```
...
idf_component_register()

set(ulp_app_name ulp_${COMPONENT_NAME})
```

(下页继续)

(续上页)

```

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 二进制文件命名。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 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 应用程序生成的头文件等相关概念，请参考下文。

### 3. 使用常规方法（例如 `idf.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 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，不用于编译进程的后续步骤。
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。

## 4.17.5 访问 ULP 程序变量

在 ULP 程序中定义的全局符号也可以在主程序中使用。

例如，ULP 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：

```

.global measurement_count
measurement_count:
    .long 0

/* 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，先包含生成的头文件，并使用上述变量，操作如下：

```
#include "ulp_app_name.h"
```

(下页继续)

```
// later
void init_ulp_vars() {
    ulp_measurement_count = 64;
}
```

注意，ULP 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 位写入的值取决于储存指令的地址，因此在读取 ULP 写的变量时，主应用程序需要屏蔽高 16 位，例如：

```
printf("Last measurement value: %d\n", ulp_last_measurement & UINT16_MAX);
```

### 4.17.6 启动 ULP 程序

要运行 ULP 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) Coprocessor”选项，以便为 ULP 预留内存。”RTC slow memory reserved for coprocessor”选项设置的值必须足够储存 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 程序均以二进制 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`）：

```
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) );
}
```

**`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):

1. MAGIC, (value 0x00706c75, 4 bytes)
2. TEXT\_OFFSET, offset of .text section from binary start (2 bytes)
3. TEXT\_SIZE, size of .text section (2 bytes)
4. DATA\_SIZE, size of .data section (2 bytes)
5. BSS\_SIZE, size of .bss section (2 bytes)
6. (TEXT\_OFFSET - 12) bytes of arbitrary data (will not be loaded into RTC memory)
7. .text section
8. .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`.

#### Return

- `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

#### Parameters

- `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

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：

```
ESP_ERROR_CHECK( ulp_run(&ulp_entry - RTC_SLOW_MEM) );
```

*esp\_err\_t* `ulp_run` (*uint32\_t* `entry_point`)

Run the program loaded into RTC memory.

**Return** `ESP_OK` on success

**Parameters**

- `entry_point`: entry point, expressed in 32-bit words

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`：

```
.global entry
entry:
    /* code starts here */
```

### 4.17.7 ULP 程序流

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_CYCx_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_CYCx_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_CYCx_REG`,  $x = 0..4$ )。

*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 wakeup 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,  $x = 0..4$ . By default, wakeup 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.

**Note** The ULP FSM requires two clock cycles to wakeup before being able to run the program. Then additional 16 cycles are reserved after wakeup waiting until the 8M 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.

**Return**

- `ESP_OK` on success
- `ESP_ERR_INVALID_ARG` if `period_index` is out of range

**Parameters**

- `period_index`: wakeup period setting number (0 - 4)
- `period_us`: wakeup period, us

一旦定时器计数到 `SENS_ULP_CP_SLEEP_CYCx_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。

## 4.18 ULP Coprocessor Programming

The ULP (Ultra Low Power) coprocessor is a simple FSM (Finite State Machine) which is designed to perform measurements using the ADC, temperature sensor, and external I2C sensors, while the main processors are in deep sleep mode. The ULP coprocessor can access the `RTC_SLOW_MEM` memory region, and registers in `RTC_CNTL`, `RTC_IO`, and `SARADC` peripherals. The ULP coprocessor uses fixed-width 32-bit instructions, 32-bit memory addressing, and has 4 general-purpose 16-bit registers.

### 4.18.1 Installing the Toolchain

The ULP coprocessor code is written in assembly and compiled using the `binutils-esp32ulp` toolchain.

If you have already set up ESP-IDF with CMake build system according to the *Getting Started Guide*, then the ULP toolchain will already be installed.

If you are using ESP-IDF with the legacy GNU Make based build system, refer to the instructions on this page: *ULP Coprocessor Programming*.

### 4.18.2 Compiling the ULP Code

To compile the ULP code as part of the component, the following steps must be taken:

1. The ULP code, written in assembly, must be added to one or more files with `.S` extension. These files must be placed into a separate directory inside the component directory, for instance `ulp/`.
2. Call `ulp_embed_binary` from the component `CMakeLists.txt` after registration. For example:

```
...
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}")
```

The first argument to `ulp_embed_binary` specifies the ULP binary name. The name specified here will also be used by other generated artifacts such as the ELF file, map file, header file and linker export file. The second argument specifies the ULP assembly source files. Finally, the third argument specifies the list of component source files which include the header file to be generated. This list is needed to build the dependencies correctly and ensure that the generated header file will be created before any of these files are compiled. See section below for the concept of generated header files for ULP applications.

3. Build the application as usual (e.g. `idf.py app`)  
Inside, the build system will take the following steps to build ULP program:
  1. **Run each assembly file (foo.S) through the C preprocessor.** This step generates the preprocessed assembly files (`foo.ulp.S`) in the component build directory. This step also generates dependency files (`foo.ulp.d`).
  2. **Run preprocessed assembly sources through the assembler.** This produces object (`foo.ulp.o`) and listing (`foo.ulp.lst`) files. Listing files are generated for debugging purposes and are not used at later stages of the build process.
  3. **Run the linker script template through the C preprocessor.** The template is located in `components/ulp/ld` directory.
  4. **Link the object files into an output ELF file (`ulp_app_name.elf`).** The Map file (`ulp_app_name.map`) generated at this stage may be useful for debugging purposes.

5. **Dump the contents of the ELF file into a binary** (`ulp_app_name.bin`) which can then be embedded into the application.
6. **Generate a list of global symbols** (`ulp_app_name.sym`) in the ELF file using `esp32ulp-elf-nm`.
7. **Create an LD export script and header file** (`ulp_app_name.ld` and `ulp_app_name.h`) containing the symbols from `ulp_app_name.sym`. This is done using the `esp32ulp_mapgen.py` utility.
8. **Add the generated binary to the list of binary files** to be embedded into the application.

### 4.18.3 Accessing the ULP Program Variables

Global symbols defined in the ULP program may be used inside the main program.

For example, the ULP program may define a variable `measurement_count` which will define the number of ADC measurements the program needs to make before waking up the chip from deep sleep:

```

                                .global measurement_count
measurement_count:              .long 0

                                /* later, use measurement_count */
                                move r3, measurement_count
                                ld r3, r3, 0

```

The main program needs to initialize this variable before the ULP program is started. The build system makes this possible by generating the `_${ULP_APP_NAME}.h` and `_${ULP_APP_NAME}.ld` files which define the global symbols present in the ULP program. These files include each global symbol defined in the ULP program, prefixed with `ulp_`.

The header file contains the declaration of the symbol:

```
extern uint32_t ulp_measurement_count;
```

Note that all symbols (variables, arrays, functions) are declared as `uint32_t`. For functions and arrays, take the address of the symbol and cast it to the appropriate type.

The generated linker script file defines the locations of symbols in `RTC_SLOW_MEM`:

```
PROVIDE ( ulp_measurement_count = 0x50000060 );
```

To access the ULP program variables from the main program, include the generated header file and use variables as one normally would:

```

#include "ulp_app_name.h"

// later
void init_ulp_vars() {
    ulp_measurement_count = 64;
}

```

Note that the ULP program can only use lower 16 bits of each 32-bit word in RTC memory, because the registers are 16-bit, and there is no instruction to load from the high part of the word.

Likewise, the ULP store instruction writes register value into the lower 16 bits part of the 32-bit word. The upper 16 bits are written with a value which depends on the address of the store instruction, thus when reading variables written by the ULP, the main application needs to mask the upper 16 bits, e.g.:

```
printf("Last measurement value: %d\n", ulp_last_measurement & UINT16_MAX);
```

### 4.18.4 Starting the ULP Program

To run a ULP program, the main application needs to load the ULP program into RTC memory using the `ulp_load_binary` function, and then start it using the `ulp_run` function.

Note that “Enable Ultra Low Power (ULP) Coprocessor” option must be enabled in menuconfig to reserve memory for the ULP. “RTC slow memory reserved for coprocessor” option must be set to a value sufficient to store ULP code and data. If the application components contain multiple ULP programs, then the size of the RTC memory must be sufficient to hold the largest one.

Each ULP program is embedded into the ESP-IDF application as a binary blob. The application can reference this blob and load it in the following way (suppose `ULP_APP_NAME` was defined to `ulp_app_name`):

```
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) );
}
```

**`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):

1. MAGIC, (value 0x00706c75, 4 bytes)
2. TEXT\_OFFSET, offset of .text section from binary start (2 bytes)
3. TEXT\_SIZE, size of .text section (2 bytes)
4. DATA\_SIZE, size of .data section (2 bytes)
5. BSS\_SIZE, size of .bss section (2 bytes)
6. (TEXT\_OFFSET - 12) bytes of arbitrary data (will not be loaded into RTC memory)
7. .text section
8. .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`.

#### Return

- `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

#### Parameters

- `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

Once the program is loaded into RTC memory, the application can start it, passing the address of the entry point to the `ulp_run` function:

```
ESP_ERROR_CHECK(ulp_run(&ulp_entry - RTC_SLOW_MEM) );
```

**`esp_err_t ulp_run`** (`uint32_t entry_point`)

Run the program loaded into RTC memory.

**Return** `ESP_OK` on success

#### Parameters

- `entry_point`: entry point, expressed in 32-bit words



Declaration of the entry point symbol comes from the generated header file mentioned above, `_${ULP_APP_NAME}.h`. In the assembly source of the ULP application, this symbol must be marked as `.global`:

```
.global entry
entry:
    /* code starts here */
```

### 4.18.5 ULP program flow

ULP coprocessor is started by a timer. The timer is started once `ulp_run` is called. The timer counts a number of `RTC_SLOW_CLK` ticks (by default, produced by an internal 150 kHz RC oscillator). The number of ticks is set using `SENS_ULP_CP_SLEEP_CYCx_REG` registers ( $x = 0..4$ ). When starting the ULP for the first time, `SENS_ULP_CP_SLEEP_CYC0_REG` will be used to set the number of timer ticks. Later the ULP program can select another `SENS_ULP_CP_SLEEP_CYCx_REG` register using `sleep` instruction.

The application can set ULP timer period values (`SENS_ULP_CP_SLEEP_CYCx_REG`,  $x = 0..4$ ) using `ulp_set_wakeup_period` function.

*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 wakeup 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,  $x = 0..4$ . By default, wakeup 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.

**Note** The ULP FSM requires two clock cycles to wakeup before being able to run the program. Then additional 16 cycles are reserved after wakeup waiting until the 8M 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.

#### Return

- `ESP_OK` on success
- `ESP_ERR_INVALID_ARG` if *period\_index* is out of range

#### Parameters

- *period\_index*: wakeup period setting number (0 - 4)
- *period\_us*: wakeup period, us

Once the timer counts the number of ticks set in the selected `SENS_ULP_CP_SLEEP_CYCx_REG` register, ULP coprocessor powers up and starts running the program from the entry point set in the call to `ulp_run`.

The program runs until it encounters a `halt` instruction or an illegal instruction. Once the program halts, ULP coprocessor powers down, and the timer is started again.

To disable the timer (effectively preventing the ULP program from running again), clear the `RTC_CNTL_ULP_CP_SLP_TIMER_EN` bit in the `RTC_CNTL_STATE0_REG` register. This can be done both from ULP code and from the main program.

## 4.19 ESP32 中的单元测试

ESP-IDF 中附带了一个基于 `Unity` 的单元测试应用程序框架，且所有的单元测试用例分别保存在 ESP-IDF 仓库中每个组件的 `test` 子目录中。

### 4.19.1 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：

```
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。
- 第二个参数用方括号中的标识符来表示，标识符用来对相关测试或具有特定属性的测试进行分组。

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 `:ref: 组件 CMakeLists.txt <component-directories>`，因为他们本身就是一种组件。ESP-IDF 使用了 `unity` 测试框架，需要将其指定为组件的依赖项。通常，组件 `:ref:` 需要手动指定待编译的源文件 `<cmake-file-globbing>`；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：

```
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>。

### 4.19.2 添加多设备测试用例

常规测试用例会在一个 DUT（Device Under Test，在试设备）上执行，那些需要互相通信的组件（比如 GPIO，SPI...）不能使用常规测试用例进行测试。多设备测试用例支持使用多个 DUT 进行写入和运行测试。

以下是一个多设备测试用例：

```
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");
}
```

(下页继续)

(续上页)

```
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，通常会要求它们之间进行同步。我们提供 `unity_wait_for_signal` 和 `unity_send_signal` 这两个函数来使用 UART 去支持同步操作。如上例中的场景，`slave` 应该在 `master` 设置好 GPIO 电平后再去读取 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。

### 4.19.3 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，我们想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，我们可以定义多阶段测试用例来将这些测试函数组合在一起：

```
static void trigger_deepsleep(void)
{
    esp_sleep_enable_timer_wakeup(2000);
    esp_deep_sleep_start();
}

void check_deepsleep_reset_reason()
{
    RESET_REASON reason = rtc_get_reset_reason(0);
    TEST_ASSERT(reason == DEEPSLEEP_RESET);
}

TEST_CASE_MULTIPLE_STAGES("reset reason check for deepsleep", "[esp32]", trigger_
↪deepsleep, check_deepsleep_reset_reason);
```

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。

### 4.19.4 编译单元测试程序

按照 `esp-idf` 顶层目录的 README 文件中的说明进行操作，请确保 `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，测试指定的组件。(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。)

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。

### 4.19.5 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]
(15)     "SPI Master test" [spi][ignore]
(16)     "SPI Master test, interaction of multiple devs" [spi][ignore]
(17)     "SPI Master no response when switch from host1 (HSPI) to host2 (VSPI)"
→[spi]
(18)     "SPI Master DMA test, TX and RX in different regions" [spi]
(19)     "SPI Master DMA test: length, start, not aligned" [spi]
(20)     "reset reason check for deepsleep" [esp32][test_env=UT_T2_1][multi_stage]
    (1)      "trigger_deepsleep"
    (2)      "check_deepsleep_reset_reason"
```

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_STAGES` 和 `TEST_CASE_MULTIPLE_DEVICES` 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：

```
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：

```
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 来运行第一阶段（触发深度睡眠）。在重启 DUT 并再次选择运行此用例后，输入 2 来运行第二阶段。只有在最后一个阶段通过并且之前所有的阶段都成功触发了复位的情况下，该测试才算通过。

## 4.20 ESP32 中的单元测试

**注解：** Since ESP-IDF V4.0, the default build system is based on CMake. This documentation is for the legacy build system based on GNU Make. Support for this build system may be removed in future major releases.

ESP-IDF 中附带了一个基于 Unity 的单元测试应用程序框架，且所有的单元测试用例分别保存在 ESP-IDF 仓库中每个组件的 `test` 子目录中。

### 4.20.1 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：

```
TEST_CASE("test name", "[module name]"
{
    // 在这里添加测试用例
})
```

- 第一个参数是字符串，用来描述当前测试。
- 第二个参数是字符串，用方括号中的标识符来表示，标识符用来对相关测试或具有特定属性的测试进行分组。

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()`。

每一个测试子目录下都应该包含一个 `component.mk`，并且里面至少要包含如下的一行内容：

```
COMPONENT_ADD_LDFLAGS = -Wl,--whole-archive -l$(COMPONENT_NAME) -Wl,--no-whole-
↪archive
```

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>。

## 4.20.2 添加多设备测试用例

常规测试用例会在一个 DUT (Device Under Test, 在试设备) 上执行, 那些需要互相通信的组件 (比如 GPIO, SPI...) 不能使用常规测试用例进行测试。多设备测试用例支持使用多个 DUT 进行写入和运行测试。

以下是一个多设备测试用例:

```
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, 通常会要求它们之间进行同步。我们提供 `unity_wait_for_signal` 和 `unity_send_signal` 这两个函数来使用 UART 去支持同步操作。如上例中的场景, `slave` 应该在在 `master` 设置好 GPIO 电平后再去读取 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 DUT2 的终端输入回车, 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。

信号也可以用来在不同 DUT 之间传递参数。例如, DUT1 希望能够拿到 DUT2 的 MAC 地址, 来进行蓝牙连接。这时, 我们可以使用 `unity_wait_for_signal_param` 以及 `unity_send_signal_param`。

DUT1 终端:

```
Waiting for signal: [dut2 mac address]!
Please input parameter value from any board send this signal and press "Enter" key.
```

DUT2 终端:

```
Send signal: [dut2 mac address][10:20:30:40:50:60]!
```

一旦 DUT2 发送信号，您需要在 DUT1 输入 10:20:30:40:50:60 及回车，然后 DUT1 会从 `unity_wait_for_signal_param` 中获取到蓝牙地址的字符串，并解除阻塞开始蓝牙连接。

### 4.20.3 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，我们想在深度睡眠唤醒后检查复位的原因是否正确。首先我们需要出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，我们可以定义多阶段测试用例来将这些测试函数组合在一起。

```
static void trigger_deepsleep(void)
{
    esp_sleep_enable_timer_wakeup(2000);
    esp_deep_sleep_start();
}

void check_deepsleep_reset_reason()
{
    RESET_REASON reason = rtc_get_reset_reason(0);
    TEST_ASSERT(reason == DEEPSLEEP_RESET);
}

TEST_CASE_MULTIPLE_STAGES("reset reason check for deepsleep", "[esp32]", trigger_
↪deepsleep, check_deepsleep_reset_reason);
```

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。

### 4.20.4 编译单元测试程序

按照 `esp-idf` 顶层目录的 `README` 文件中的说明进行操作，请确保 `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：

- `make menuconfig` - 配置单元测试程序。
- `make TESTS_ALL=1` - 编译单元测试程序，测试每个组件 `test` 子目录下的用例。
- `make TEST_COMPONENTS='xxx'` - 编译单元测试程序，测试指定的组件。
- `make TESTS_ALL=1 TEST_EXCLUDE_COMPONENTS='xxx'` - 编译单元测试程序，测试所有（除开指定）的组件。例如 `make TESTS_ALL=1 TEST_EXCLUDE_COMPONENTS='ulp mbedtls'` - 编译所有的单元测试，不包括 `ulp` 和 `mbedtls` 组件。

当编译完成时，它会打印出烧写芯片的指令。您只需要运行 `make flash` 即可烧写所有编译输出的文件。

您还可以运行 `make flash TESTS_ALL=1` 或者 `make TEST_COMPONENTS='xxx'` 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。

### 4.20.5 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]
(15)  "SPI Master test" [spi][ignore]
(16)  "SPI Master test, interaction of multiple devs" [spi][ignore]
(17)  "SPI Master no response when switch from host1 (HSPI) to host2 (VSPI)" ↪
↪[spi]
(18)  "SPI Master DMA test, TX and RX in different regions" [spi]
(19)  "SPI Master DMA test: length, start, not aligned" [spi]
(20)  "reset reason check for deepsleep" [esp32][test_env=UT_T2_1][multi_stage]
      (1)   "trigger_deepsleep"
      (2)   "check_deepsleep_reset_reason"

```

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：

- 引号中的测试用例的名字（例如 "esp\_ota\_begin() verifies arguments"），运行单个测试用例。
- 测试用例的序号（例如 1），运行单个测试用例。
- 方括号中的模块名字（例如 [cxx]），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\_STAGES 和 TEST\_CASE\_MULTIPLE\_DEVICES 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：

```

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：

```

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 来运行第一阶段（触发深度睡眠）。在重启 DUT 并再次选择运行此用例后，输入 2 来运行第二阶段。只有在最后一个阶段通过并且之前所有的阶段都成功触发了复位的情况下，该测试才算通过。



## 4.21 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：

- 行编辑，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，它支持解析 GNU 样式的命令行参数。
- 用于注册和调度命令的函数。

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。

### 4.21.1 行编辑

行编辑功能允许用户通过按键输入来编辑命令，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。

---

**注解：** 此功能依赖于终端应用程序对 ANSI 转移符的支持，显示原始 UART 数据的串口监视器不能与行编辑库一同使用。如果运行 `get_started/console` 示例程序的时候观察到的输出结果是 `[6n` 或者类似的转义字符而不是命令行提示符 `[esp32]>` 时，就表明当前的串口监视器不支持 ANSI 转移字符。已知可用的串口监视程序有 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
```

使用转移字符清除终端屏幕，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。

#### 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一行内容。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。

## 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`（见后文）。

### 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在。当使用完 `linenoise` 库后需要调用此函数。

### 4.21.2 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移。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" ]`

### 4.21.3 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](#)。

### 4.21.4 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 `argc/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 `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。

### 4.21.5 示例

`examples/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。

## 4.22 ESP32 ROM console

When an ESP32 is unable to boot from flash ROM (and the fuse disabling it hasn't been blown), it boots into a rom console. The console is based on TinyBasic, and statements entered should be in the form of BASIC statements. As

is common in the BASIC language, without a preceding line number, commands entered are executed immediately; lines with a prefixed line number are stored as part of a program.

### 4.22.1 Full list of supported statements and functions

#### System

- **BYE** - exits Basic, reboots ESP32, retries booting from flash
- **END** - stops execution from the program, also “STOP”
- **MEM** - displays memory usage statistics
- **NEW** - clears the current program
- **RUN** - executes the current program

#### IO, Documentation

- **PEEK( address )** - get a 32-bit value from a memory address
- **POKE** - write a 32-bit value to memory
- **USR(addr, arg1, ..)** - Execute a machine language function
- **PRINT expression** - print out the expression, also “?”
- **PHEX expression** - print expression as a hex number
- **REM stuff** - remark/comment, also “ ”

#### Expressions, Math

- **A=V, LET A=V** - assign value to a variable
- **+, -, \*, /** - Math
- **<, <=, =, >, !=, >=, >** - Comparisons
- **ABS( expression )** - returns the absolute value of the expression
- **RSEED( v )** - sets the random seed to v
- **RND( m )** - returns a random number from 0 to m
- **A=1234** - \* Assign a decimal value\*
- **A=&h1A2** - \* Assign a hex value\*
- **A=&b1001** - Assign a binary value

#### Control

- **IF expression statement** - perform statement if expression is true
- **FOR variable = start TO end** - start for block
- **FOR variable = start TO end STEP value** - start for block with step
- **NEXT** - end of for block
- **GOTO linenumber** - continue execution at this line number
- **GOSUB linenumber** - call a subroutine at this line number
- **RETURN** - return from a subroutine
- **DELAY** - Delay a given number of milliseconds

#### Pin IO

- **IODIR** - Set a GPIO-pin as an output (1) or input (0)
- **IOSET** - Set a GPIO-pin, configured as output, to high (1) or low (0)
- **IOGET** - Get the value of a GPIO-pin

### 4.22.2 Example programs

Here are a few example commands and programs to get you started...

#### Read UART\_DATE register of uart0

```
> PHEX PEEK(&h3FF40078)
15122500
```

#### Set GPIO2 using memory writes to GPIO\_OUT\_REG

Note: you can do this easier with the IOSET command

```
> POKE &h3FF44004, PEEK(&h3FF44004) OR &b100
```

#### Get value of GPIO0

```
> IODIR 0,0
> PRINT IOGET(0)
0
```

#### Blink LED

Hook up an LED between GPIO2 and ground. When running the program, the LED should blink 10 times.

```
10 IODIR 2,1
20 FOR A=1 TO 10
30 IOSET 2,1
40 DELAY 250
50 IOSET 2,0
60 DELAY 250
70 NEXT A
RUN
```

### 4.22.3 Credits

The ROM console is based on “TinyBasicPlus” by Mike Field and Scott Lawrence, which is based on “68000 TinyBasic” by Gordon Brandly

## 4.23 RF calibration

ESP32 supports three RF calibration methods during RF initialization:

1. Partial calibration
2. Full calibration
3. No calibration

### 4.23.1 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_ESP32_PHY_CALIBRATION_AND_DATA_STORAGE`.

### 4.23.2 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_ESP32_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.

### 4.23.3 No calibration

No calibration method is only used when ESP32 wakes up from deep sleep.

### 4.23.4 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_wifi/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_ESP32_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_ESP32_PHY_INIT_DATA_IN_PARTITION`.

## 4.24 Wi-Fi Driver

### 4.24.1 ESP32 Wi-Fi Feature List

- Support Station-only mode, AP-only mode, Station/AP-coexistence mode
- Support IEEE-802.11B, IEEE-802.11G, IEEE802.11N and APIs to configure the protocol mode
- Support WPA/WPA2/WPA2-Enterprise and WPS
- Support AMPDU, HT40, QoS and other key features
- Support Modem-sleep
- Support an Espressif-specific protocol which, in turn, supports up to **1 km** of data traffic
- Up to 20 MBit/sec TCP throughput and 30 MBit/sec UDP throughput over the air
- Support Sniffer
- Support set `fast_crypto` algorithm and normal algorithm switch which used in wifi connect
- Support both fast scan and all channel scan feature
- Support multiple antennas
- Support channel state information

## 4.24.2 How To Write a Wi-Fi Application

### Preparation

Generally, the most effective way to begin your own Wi-Fi application is to select an example which is similar to your own application, and port the useful part into your project. It is not a MUST but it is strongly recommended that you take some time to read this article first, especially if you want to program a robust Wi-Fi application. This article is supplementary to the Wi-Fi APIs/Examples. It describes the principles of using the Wi-Fi APIs, the limitations of the current Wi-Fi API implementation, and the most common pitfalls in using Wi-Fi. This article also reveals some design details of the Wi-Fi driver. We recommend that you become familiar at least with the following sections: [<ESP32 Wi-Fi API Error Code>](#), [<ESP32 Wi-Fi Programming Model>](#), and [<ESP32 Wi-Fi Event Description>](#).

### Setting Wi-Fi Compile-time Options

Refer to [<Wi-Fi Menuconfig>](#)

### Init Wi-Fi

Refer to [<ESP32 Wi-Fi Station General Scenario>](#), [<ESP32 Wi-Fi AP General Scenario>](#).

### Start/Connect Wi-Fi

Refer to [<ESP32 Wi-Fi Station General Scenario>](#), [<ESP32 Wi-Fi AP General Scenario>](#).

### Event-Handling

Generally, it is easy to write code in “sunny-day” scenarios, such as [<WIFI\\_EVENT\\_STA\\_START>](#), [<WIFI\\_EVENT\\_STA\\_CONNECTED>](#) etc. The hard part is to write routines in “rainy-day” scenarios, such as [<WIFI\\_EVENT\\_STA\\_DISCONNECTED>](#) etc. Good handling of “rainy-day” scenarios is fundamental to robust Wi-Fi applications. Refer to [<ESP32 Wi-Fi Event Description>](#), [<ESP32 Wi-Fi Station General Scenario>](#), [<ESP32 Wi-Fi AP General Scenario>](#). See also *an overview of event handling in ESP-IDF*.

### Write Error-Recovery Routines Correctly at All Times

Just like the handling of “rainy-day” scenarios, a good error-recovery routine is also fundamental to robust Wi-Fi applications. Refer to [<ESP32 Wi-Fi API Error Code>](#)

## 4.24.3 ESP32 Wi-Fi API Error Code

All of the ESP32 Wi-Fi APIs have well-defined return values, namely, the error code. The error code can be categorized into:

- No errors, e.g. ESP\_OK means that the API returns successfully
- Recoverable errors, such as ESP\_ERR\_NO\_MEM, etc.
- Non-recoverable, non-critical errors
- Non-recoverable, critical errors

Whether the error is critical or not depends on the API and the application scenario, and it is defined by the API user.

**The primary principle to write a robust application with Wi-Fi API is to always check the error code and write the error-handling code.** Generally, the error-handling code can be used:

- for recoverable errors, in which case you can write a recoverable-error code. For example, when esp\_wifi\_start returns ESP\_ERR\_NO\_MEM, the recoverable-error code vTaskDelay can be called, in order to get a microsecond's delay for another try.

- for non-recoverable, yet non-critical, errors, in which case printing the error code is a good method for error handling.
- for non-recoverable, critical errors, in which case “assert” may be a good method for error handling. For example, if `esp_wifi_set_mode` returns `ESP_ERR_WIFI_NOT_INIT`, it means that the Wi-Fi driver is not initialized by `esp_wifi_init` successfully. You can detect this kind of error very quickly in the application development phase.

In `esp_err.h`, `ESP_ERROR_CHECK` checks the return values. It is a rather commonplace error-handling code and can be used as the default error-handling code in the application development phase. However, we strongly recommend that API users write their own error-handling code.

#### 4.24.4 ESP32 Wi-Fi API Parameter Initialization

When initializing struct parameters for the API, one of two approaches should be followed: - explicitly set all fields of the parameter or - use `get API` to get current configuration first, then set application specific fields

Initializing or getting the entire structure is very important because most of the time the value 0 indicates the default value is used. More fields may be added to the struct in the future and initializing these to zero ensures the application will still work correctly after IDF is updated to a new release.

#### 4.24.5 ESP32 Wi-Fi Programming Model

The ESP32 Wi-Fi programming model is depicted as follows:

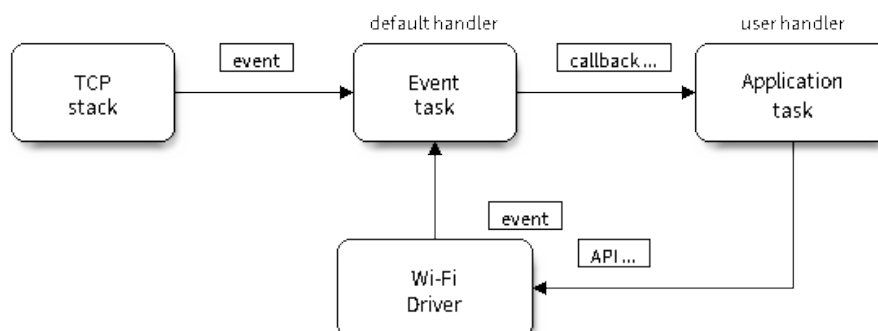


图 24: Wi-Fi Programming Model

The Wi-Fi driver can be considered a black box that knows nothing about high-layer code, such as the TCP/IP stack, application task, event task, etc. The application task (code) generally calls *Wi-Fi driver APIs* to initialize Wi-Fi and handles Wi-Fi events when necessary. Wi-Fi driver receives API calls, handles them, and post events to the application.

Wi-Fi event handling is based on the *esp\_event library*. Events are sent by the Wi-Fi driver to the *default event loop*. Application may handle these events in callbacks registered using `esp_event_handler_register()`. Wi-Fi events are also handled by *tcpip\_adapter component* to provide a set of default behaviors. For example, when Wi-Fi station connects to an AP, `tcpip_adapter` will automatically start the DHCP client.

#### 4.24.6 ESP32 Wi-Fi Event Description



### WIFI\_EVENT\_WIFI\_READY

The Wi-Fi driver will never generate this event, which, as a result, can be ignored by the application event callback. This event may be removed in future releases.

### WIFI\_EVENT\_SCAN\_DONE

The scan-done event is triggered by `esp_wifi_scan_start()` and will arise in the following scenarios:

- The scan is completed, e.g., the target AP is found successfully, or all channels have been scanned.
- The scan is stopped by `esp_wifi_scan_stop()`.
- The `esp_wifi_scan_start()` is called before the scan is completed. A new scan will override the current scan and a scan-done event will be generated.

The scan-done event will not arise in the following scenarios:

- It is a blocked scan.
- The scan is caused by `esp_wifi_connect()`.

Upon receiving this event, the event task does nothing. The application event callback needs to call `esp_wifi_scan_get_ap_num()` and `esp_wifi_scan_get_ap_records()` to fetch the scanned AP list and trigger the Wi-Fi driver to free the internal memory which is allocated during the scan (**do not forget to do this!**)! Refer to ‘ESP32 Wi-Fi Scan’ for a more detailed description.

### WIFI\_EVENT\_STA\_START

If `esp_wifi_start()` returns `ESP_OK` and the current Wi-Fi mode is Station or AP+Station, then this event will arise. Upon receiving this event, the event task will initialize the LwIP network interface (`netif`). Generally, the application event callback needs to call `esp_wifi_connect()` to connect to the configured AP.

### WIFI\_EVENT\_STA\_STOP

If `esp_wifi_stop()` returns `ESP_OK` and the current Wi-Fi mode is Station or AP+Station, then this event will arise. Upon receiving this event, the event task will release the station’s IP address, stop the DHCP client, remove TCP/UDP-related connections and clear the LwIP station `netif`, etc. The application event callback generally does not need to do anything.

### WIFI\_EVENT\_STA\_CONNECTED

If `esp_wifi_connect()` returns `ESP_OK` and the station successfully connects to the target AP, the connection event will arise. Upon receiving this event, the event task starts the DHCP client and begins the DHCP process of getting the IP address. Then, the Wi-Fi driver is ready for sending and receiving data. This moment is good for beginning the application work, provided that the application does not depend on LwIP, namely the IP address. However, if the application is LwIP-based, then you need to wait until the *got ip* event comes in.

### WIFI\_EVENT\_STA\_DISCONNECTED

This event can be generated in the following scenarios:

- When `esp_wifi_disconnect()`, or `esp_wifi_stop()`, or `esp_wifi_deinit()` is called and the station is already connected to the AP.
- When `esp_wifi_connect()` is called, but the Wi-Fi driver fails to set up a connection with the AP due to certain reasons, e.g. the scan fails to find the target AP, authentication times out, etc. If there are more than one AP with the same SSID, the disconnected event is raised after the station fails to connect all of the found APs.
- When the Wi-Fi connection is disrupted because of specific reasons, e.g., the station continuously loses N beacons, the AP kicks off the station, the AP’s authentication mode is changed, etc.

Upon receiving this event, the default behavior of the event task is: - Shuts down the station's LwIP netif. - Notifies the LwIP task to clear the UDP/TCP connections which cause the wrong status to all sockets. For socket-based applications, the application callback can choose to close all sockets and re-create them, if necessary, upon receiving this event.

The most common event handle code for this event in application is to call `esp_wifi_connect()` to reconnect the Wi-Fi. However, if the event is raised because `esp_wifi_disconnect()` is called, the application should not call `esp_wifi_connect()` to reconnect. It's application's responsibility to distinguish whether the event is caused by `esp_wifi_disconnect()` or other reasons. Sometimes a better reconnect strategy is required, refer to [<Wi-Fi Reconnect>](#) and [<Scan When Wi-Fi Is Connecting>](#).

Another thing deserves our attention is that the default behavior of LwIP is to abort all TCP socket connections on receiving the disconnect. Most of time it is not a problem. However, for some special application, this may not be what they want, consider following scenarios:

- The application creates a TCP connection to maintain the application-level keep-alive data that is sent out every 60 seconds.
- Due to certain reasons, the Wi-Fi connection is cut off, and the `<WIFI_EVENT_STA_DISCONNECTED>` is raised. According to the current implementation, all TCP connections will be removed and the keep-alive socket will be in a wrong status. However, since the application designer believes that the network layer should NOT care about this error at the Wi-Fi layer, the application does not close the socket.
- Five seconds later, the Wi-Fi connection is restored because `esp_wifi_connect()` is called in the application event callback function. **Moreover, the station connects to the same AP and gets the same IPV4 address as before.**
- Sixty seconds later, when the application sends out data with the keep-alive socket, the socket returns an error and the application closes the socket and re-creates it when necessary.

In above scenario, ideally, the application sockets and the network layer should not be affected, since the Wi-Fi connection only fails temporarily and recovers very quickly. The application can enable “Keep TCP connections when IP changed” via LwIP menuconfig.

### IP\_EVENT\_STA\_GOT\_IP

This event arises when the DHCP client successfully gets the IPV4 address from the DHCP server, or when the IPV4 address is changed. The event means that everything is ready and the application can begin its tasks (e.g., creating sockets).

The IPV4 may be changed because of the following reasons:

- The DHCP client fails to renew/rebind the IPV4 address, and the station's IPV4 is reset to 0.
- The DHCP client rebinds to a different address.
- The static-configured IPV4 address is changed.

Whether the IPV4 address is changed or NOT is indicated by field `ip_change` of `ip_event_got_ip_t`.

The socket is based on the IPV4 address, which means that, if the IPV4 changes, all sockets relating to this IPV4 will become abnormal. Upon receiving this event, the application needs to close all sockets and recreate the application when the IPV4 changes to a valid one.

### IP\_EVENT\_GOT\_IP6

This event arises when the IPV6 SLAAC support auto-configures an address for the ESP32, or when this address changes. The event means that everything is ready and the application can begin its tasks (e.g., creating sockets).

### IP\_STA\_LOST\_IP

This event arises when the IPV4 address become invalid.

IP\_STA\_LOST\_IP doesn't arise immediately after the WiFi disconnects, instead it starts an IPV4 address lost timer, if the IPV4 address is got before ip lost timer expires, IP\_EVENT\_STA\_LOST\_IP doesn't happen. Otherwise, the event arises when IPV4 address lost timer expires.

Generally the application don't need to care about this event, it is just a debug event to let the application know that the IPV4 address is lost.

### **WIFI\_EVENT\_AP\_START**

Similar to [<WIFI\\_EVENT\\_STA\\_START>](#).

### **WIFI\_EVENT\_AP\_STOP**

Similar to [<WIFI\\_EVENT\\_STA\\_STOP>](#).

### **WIFI\_EVENT\_AP\_STACONNECTED**

Every time a station is connected to ESP32 AP, the [<WIFI\\_EVENT\\_AP\\_STACONNECTED>](#) will arise. Upon receiving this event, the event task will do nothing, and the application callback can also ignore it. However, you may want to do something, for example, to get the info of the connected STA, etc.

### **WIFI\_EVENT\_AP\_STADISCONNECTED**

This event can happen in the following scenarios:

- The application calls `esp_wifi_disconnect()`, or `esp_wifi_deinit_sta()`, to manually disconnect the station.
- The Wi-Fi driver kicks off the station, e.g. because the AP has not received any packets in the past five minutes, etc. The time can be modified by `esp_wifi_set_inactive_time()`.
- The station kicks off the AP.

When this event happens, the event task will do nothing, but the application event callback needs to do something, e.g., close the socket which is related to this station, etc.

### **WIFI\_EVENT\_AP\_PROBEREQRCVD**

This event is disabled by default. The application can enable it via API `esp_wifi_set_event_mask()`. When this event is enabled, it will be raised each time the AP receives a probe request.

## **4.24.7 ESP32 Wi-Fi Station General Scenario**

Below is a “big scenario” which describes some small scenarios in Station mode:

### **1. Wi-Fi/LwIP Init Phase**

- s1.1: The main task calls `tcpip_adapter_init()` to create an LwIP core task and initialize LwIP-related work.
- s1.2: The main task calls `esp_event_loop_init()` to create a system Event task and initialize an application event's callback function. In the scenario above, the application event's callback function does nothing but relaying the event to the application task.
- s1.3: The main task calls `esp_wifi_init()` to create the Wi-Fi driver task and initialize the Wi-Fi driver.
- s1.4: The main task calls OS API to create the application task.

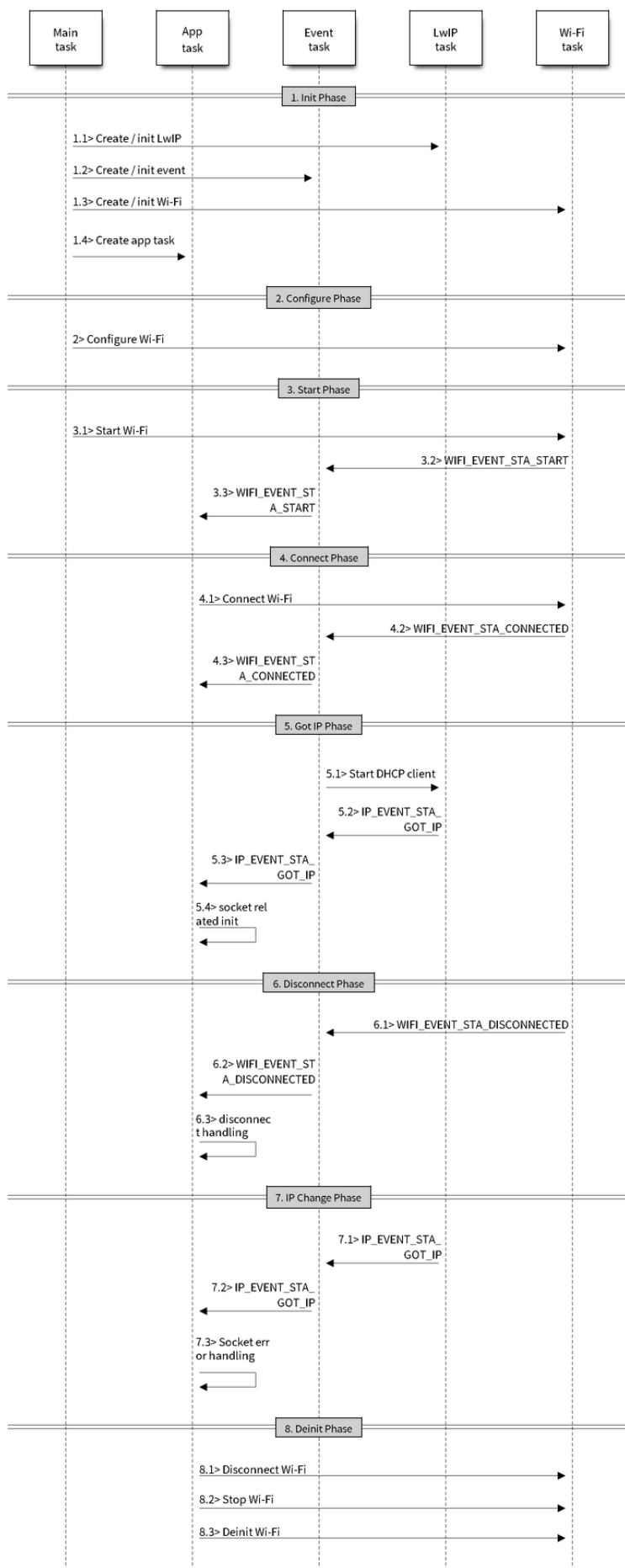


图 25: Sample Wi-Fi Event Scenarios in Station Mode

Step 1.1~1.4 is a recommended sequence that initializes a Wi-Fi/LwIP-based application. However, it is **NOT** a must-follow sequence, which means that you can create the application task in step 1.1 and put all other initializations in the application task. Moreover, you may not want to create the application task in the initialization phase if the application task depends on the sockets. Rather, you can defer the task creation until the IP is obtained.

## 2. Wi-Fi Configuration Phase

Once the Wi-Fi driver is initialized, you can start configuring the Wi-Fi driver. In this scenario, the mode is Station, so you may need to call `esp_wifi_set_mode(WIFI_MODE_STA)` to configure the Wi-Fi mode as Station. You can call other `esp_wifi_set_XXX` APIs to configure more settings, such as the protocol mode, country code, bandwidth, etc. Refer to [<ESP32 Wi-Fi Configuration>](#).

Generally, we configure the Wi-Fi driver before setting up the Wi-Fi connection, but this is **NOT** mandatory, which means that you can configure the Wi-Fi connection anytime, provided that the Wi-Fi driver is initialized successfully. However, if the configuration does not need to change after the Wi-Fi connection is set up, you should configure the Wi-Fi driver at this stage, because the configuration APIs (such as `esp_wifi_set_protocol`) will cause the Wi-Fi to reconnect, which may not be desirable.

If the Wi-Fi NVS flash is enabled by menuconfig, all Wi-Fi configuration in this phase, or later phases, will be stored into flash. When the board powers on/reboots, you do not need to configure the Wi-Fi driver from scratch. You only need to call `esp_wifi_get_XXX` APIs to fetch the configuration stored in flash previously. You can also configure the Wi-Fi driver if the previous configuration is not what you want.

## 3. Wi-Fi Start Phase

- s3.1: Call `esp_wifi_start` to start the Wi-Fi driver.
- s3.2: The Wi-Fi driver posts [<WIFI\\_EVENT\\_STA\\_START>](#) to the event task; then, the event task will do some common things and will call the application event callback function.
- s3.3: The application event callback function relays the [<WIFI\\_EVENT\\_STA\\_START>](#) to the application task. We recommend that you call `esp_wifi_connect()`. However, you can also call `esp_wifi_connect()` in other phrases after the [<WIFI\\_EVENT\\_STA\\_START>](#) arises.

## 4. Wi-Fi Connect Phase

- s4.1: Once `esp_wifi_connect()` is called, the Wi-Fi driver will start the internal scan/connection process.
- s4.2: If the internal scan/connection process is successful, the [<WIFI\\_EVENT\\_STA\\_CONNECTED>](#) will be generated. In the event task, it starts the DHCP client, which will finally trigger the DHCP process.
- s4.3: In the above-mentioned scenario, the application event callback will relay the event to the application task. Generally, the application needs to do nothing, and you can do whatever you want, e.g., print a log, etc.

In step 4.2, the Wi-Fi connection may fail because, for example, the password is wrong, the AP is not found, etc. In a case like this, [<WIFI\\_EVENT\\_STA\\_DISCONNECTED>](#) will arise and the reason for such a failure will be provided. For handling events that disrupt Wi-Fi connection, please refer to phase 6.

## 5. Wi-Fi ‘Got IP’ Phase

- s5.1: Once the DHCP client is initialized in step 4.2, the *got IP* phase will begin.
- s5.2: If the IP address is successfully received from the DHCP server, then [<IP\\_EVENT\\_STA\\_GOT\\_IP>](#) will arise and the event task will perform common handling.
- s5.3: In the application event callback, [<IP\\_EVENT\\_STA\\_GOT\\_IP>](#) is relayed to the application task. For LwIP-based applications, this event is very special and means that everything is ready for the application to begin its tasks, e.g. creating the TCP/UDP socket, etc. A very common mistake is to initialize the socket before [<IP\\_EVENT\\_STA\\_GOT\\_IP>](#) is received. **DO NOT start the socket-related work before the IP is received.**

## 6. Wi-Fi Disconnect Phase

- s6.1: When the Wi-Fi connection is disrupted, e.g. because the AP is powered off, the RSSI is poor, etc., `<WIFI_EVENT_STA_DISCONNECTED>` will arise. This event may also arise in phase 3. Here, the event task will notify the LwIP task to clear/remove all UDP/TCP connections. Then, all application sockets will be in a wrong status. In other words, no socket can work properly when this event happens.
- s6.2: In the scenario described above, the application event callback function relays `<WIFI_EVENT_STA_DISCONNECTED>` to the application task. We recommend that `esp_wifi_connect()` be called to reconnect the Wi-Fi, close all sockets and re-create them if necessary. Refer to `<WIFI_EVENT_STA_DISCONNECTED>`.

## 7. Wi-Fi IP Change Phase

- s7.1: If the IP address is changed, the `<IP_EVENT_STA_GOT_IP>` will arise with “ip\_change” set to true.
- s7.2: **This event is important to the application. When it occurs, the timing is good for closing all created sockets and recreating them.**

## 8. Wi-Fi Deinit Phase

- s8.1: Call `esp_wifi_disconnect()` to disconnect the Wi-Fi connectivity.
- s8.2: Call `esp_wifi_stop()` to stop the Wi-Fi driver.
- s8.3: Call `esp_wifi_deinit()` to unload the Wi-Fi driver.

### 4.24.8 ESP32 Wi-Fi AP General Scenario

Below is a “big scenario” which describes some small scenarios in AP mode:

### 4.24.9 ESP32 Wi-Fi Scan

Currently, the `esp_wifi_scan_start()` API is supported only in Station or Station+AP mode.

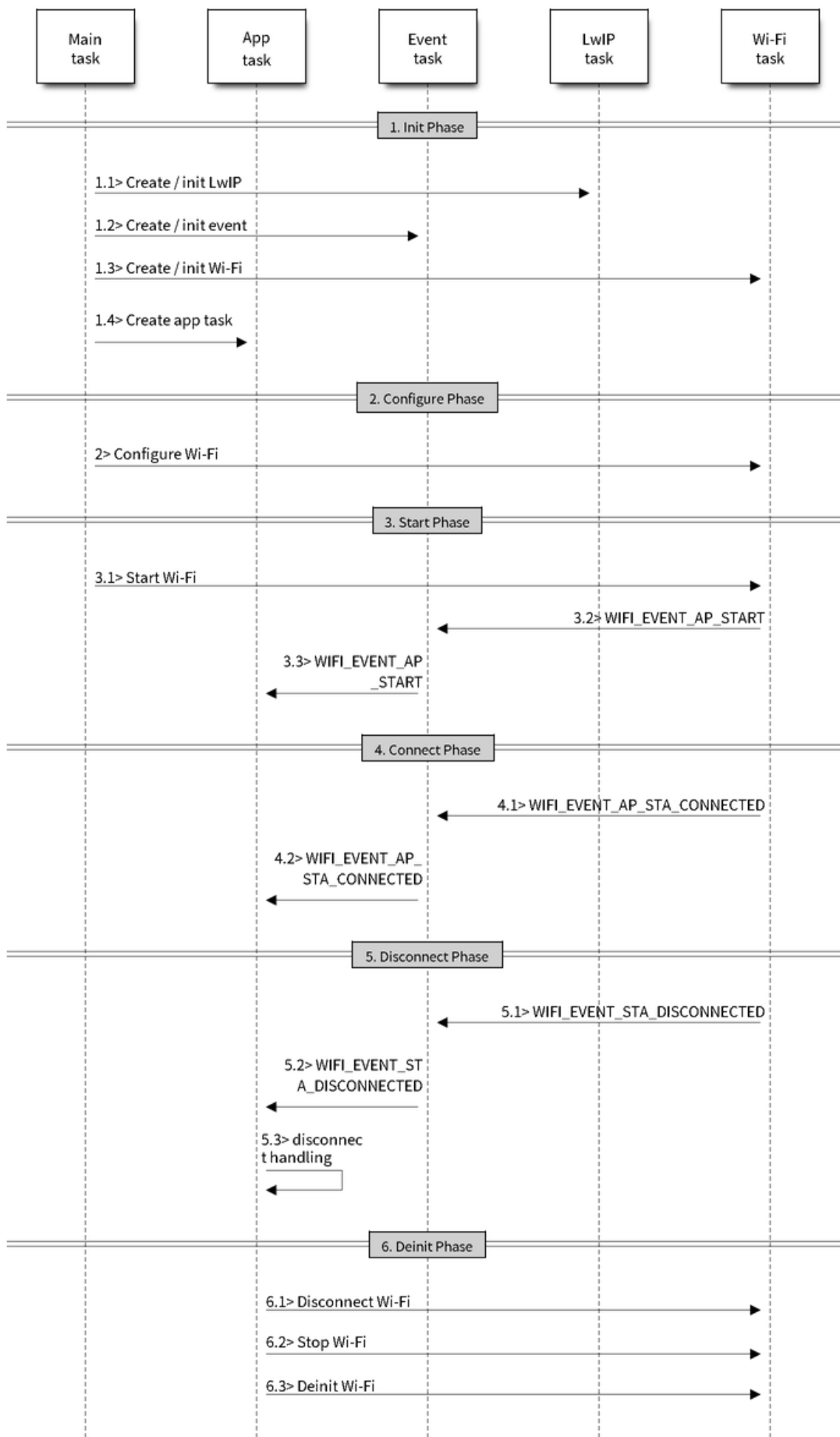


图 26: Sample Wi-Fi Event Scenarios in AP Mode

## Scan Type

Mode	Description
Active Scan	Scan by sending a probe request. The default scan is an active scan.
Passive Scan	No probe request is sent out. Just switch to the specific channel and wait for a beacon. Application can enable it via the <code>scan_type</code> field of <code>wifi_scan_config_t</code> .
Foreground Scan	This scan is applicable when there is no Wi-Fi connection in Station mode. Foreground or background scanning is controlled by the Wi-Fi driver and cannot be configured by the application.
Background Scan	This scan is applicable when there is a Wi-Fi connection in Station mode or in Station+AP mode. Whether it is a foreground scan or background scan depends on the Wi-Fi driver and cannot be configured by the application.
All-Channel Scan	It scans all of the channels. If the <code>channel</code> field of <code>wifi_scan_config_t</code> is set to 0, it is an all-channel scan.
<b>Specific Channel Scan</b>	It scans specific channels only. If the <code>channel</code> field of <code>wifi_scan_config_t</code> set to 1, it is a specific-channel scan.

The scan modes in above table can be combined arbitrarily, so we totally have 8 different scans:

- All-Channel Background Active Scan
- All-Channel Background Passive Scan
- All-Channel Foreground Active Scan
- All-Channel Foreground Passive Scan
- Specific-Channel Background Active Scan
- Specific-Channel Background Passive Scan
- Specific-Channel Foreground Active Scan
- Specific-Channel Foreground Passive Scan

## Scan Configuration

The scan type and other per-scan attributes are configured by `esp_wifi_scan_start`. The table below provides a detailed description of `wifi_scan_config_t`.



Field	Description
ssid	If the SSID is not NULL, it is only the AP with the same SSID that can be scanned.
bssid	If the BSSID is not NULL, it is only the AP with the same BSSID that can be scanned.
channel	If “channel” is 0, there will be an all-channel scan; otherwise, there will be a specific-channel scan.
show_hidden	If “show_hidden” is 0, the scan ignores the AP with a hidden SSID; otherwise, the scan considers the hidden AP a normal one.
scan_type	If “scan_type” is WIFI_SCAN_TYPE_ACTIVE, the scan is “active” ; otherwise, it is a “passive” one.
scan_time	<p>This field is used to control how long the scan dwells on each channel.</p> <p>For passive scans, scan_time.passive designates the dwell time for each channel.</p> <p>For active scans, dwell times for each channel are listed in the table below. Here, min is short for scan_time.active.min and max is short for scan_time.active.max.</p> <ul style="list-style-type: none"> <li>• min=0, max=0: scan dwells on each channel for 120 ms.</li> <li>• min&gt;0, max=0: scan dwells on each channel for 120 ms.</li> <li>• min=0, max&gt;0: scan dwells on each channel for max ms.</li> <li>• min&gt;0, max&gt;0: the minimum time the scan dwells on each channel is min ms. If no AP is found during this time frame, the scan switches to the next channel. Otherwise, the scan dwells on the channel for max ms.</li> </ul> <p>If you want to improve the performance of the the scan, you can try to modify these two parameters.</p>

There also some global scan attributes which is configured by API esp\_wifi\_set\_config, refer to [Station Basic Configuration](#)

### Scan All APs In All Channels(foreground)

Scenario:

The scenario above describes an all-channel, foreground scan. The foreground scan can only occur in Station mode where the station does not connect to any AP. Whether it is a foreground or background scan is totally determined by the Wi-Fi driver, and cannot be configured by the application.

Detailed scenario description:

#### Scan Configuration Phase

- s1.1: Call esp\_wifi\_set\_country() to set the country info if the default country info is not what you want, refer to [<Wi-Fi Country Code>](#).
- s1.2: Call esp\_wifi\_scan\_start() to configure the scan. To do so, you can refer to [<Scan Configuration>](#). Since this is an all-channel scan, just set the SSID/BSSID/channel to 0.

#### Wi-Fi Driver' s Internal Scan Phase

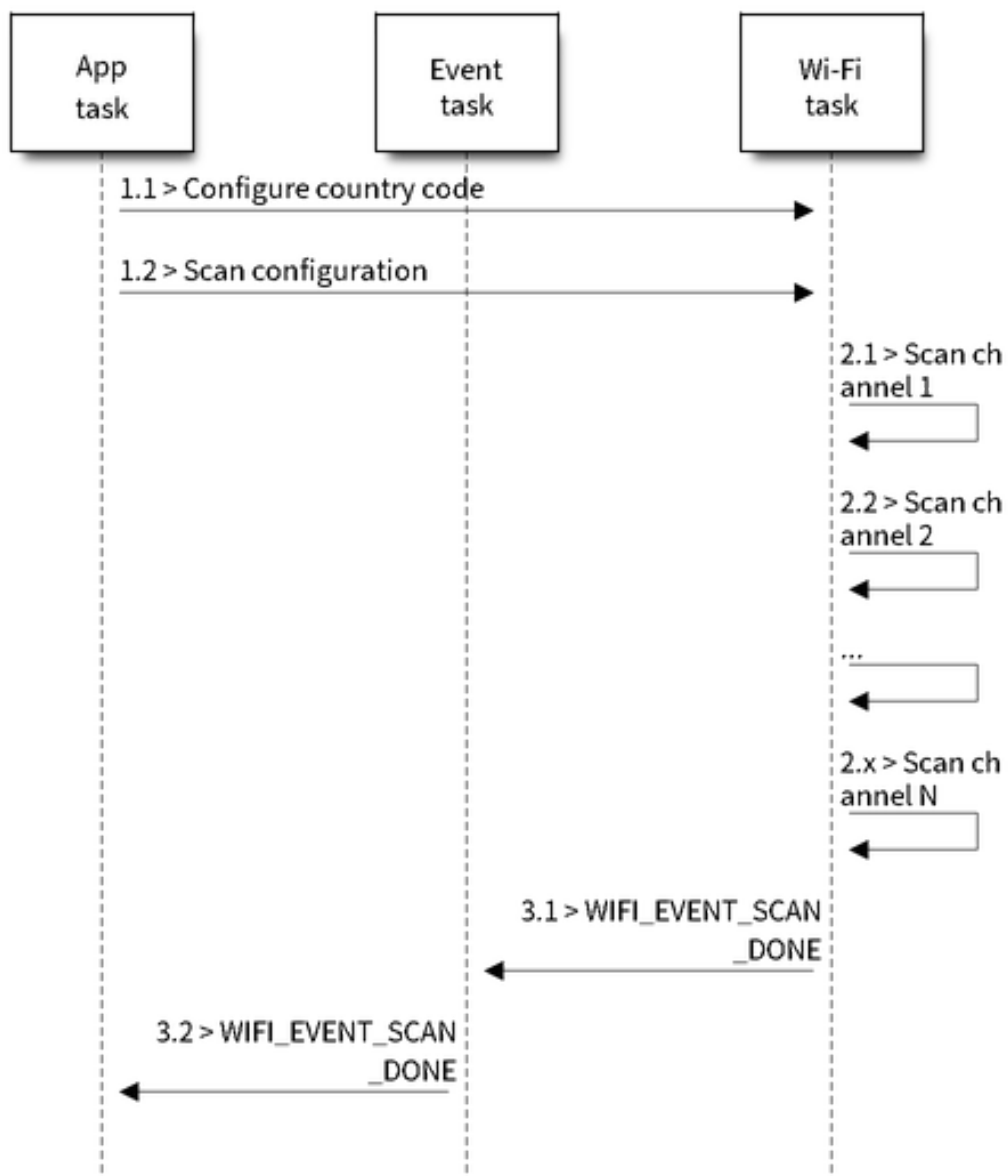


图 27: Foreground Scan of all Wi-Fi Channels

- s2.1: The Wi-Fi driver switches to channel 1, in case the scan type is `WIFI_SCAN_TYPE_ACTIVE`, and broadcasts a probe request. Otherwise, the Wi-Fi will wait for a beacon from the APs. The Wi-Fi driver will stay in channel 1 for some time. The dwell time is configured in min/max time, with default value being 120 ms.
- s2.2: The Wi-Fi driver switches to channel 2 and performs the same operation as in step 2.1.
- s2.3: The Wi-Fi driver scans the last channel N, where N is determined by the country code which is configured in step 1.1.

### Scan-Done Event Handling Phase

- s3.1: When all channels are scanned, `<WIFI_EVENT_SCAN_DONE>` will arise.
- s3.2: The application's event callback function notifies the application task that `<WIFI_EVENT_SCAN_DONE>` is received. `esp_wifi_scan_get_ap_num()` is called to get the number of APs that have been found in this scan. Then, it allocates enough entries and calls `esp_wifi_scan_get_ap_records()` to get the AP records. Please note that the AP records in the Wi-Fi driver will be freed, once `esp_wifi_scan_get_ap_records()` is called. Do not call `esp_wifi_scan_get_ap_records()` twice for a single scan-done event. If `esp_wifi_scan_get_ap_records()` is not called when the scan-done event occurs, the AP records allocated by the Wi-Fi driver will not be freed. So, make sure you call `esp_wifi_scan_get_ap_records()`, yet only once.

### Scan All APs on All Channels(background)

Scenario:

The scenario above is an all-channel background scan. Compared to *Scan All APs In All Channels(foreground)*, the difference in the all-channel background scan is that the Wi-Fi driver will scan the back-to-home channel for 30 ms before it switches to the next channel to give the Wi-Fi connection a chance to transmit/receive data.

### Scan for a Specific AP in All Channels

Scenario:

This scan is similar to *Scan All APs In All Channels(foreground)*. The differences are:

- s1.1: In step 1.2, the target AP will be configured to SSID/BSSID.
- s2.1~s2.N: Each time the Wi-Fi driver scans an AP, it will check whether it is a target AP or not. If the scan is `WIFI_FAST_SCAN` scan and the target AP is found, then the scan-done event will arise and scanning will end; otherwise, the scan will continue. Please note that the first scanned channel may not be channel 1, because the Wi-Fi driver optimizes the scanning sequence.

If there are multiple APs which match the target AP info, for example, if we happen to scan two APs whose SSID is "ap". If the scan is `WIFI_FAST_SCAN`, then only the first scanned "ap" will be found, if the scan is `WIFI_ALL_CHANNEL_SCAN`, both "ap" will be found and the station will connect the "ap" according to the configured strategy, refer to *Station Basic Configuration*.

You can scan a specific AP, or all of them, in any given channel. These two scenarios are very similar.

### Scan in Wi-Fi Connect

When `esp_wifi_connect()` is called, then the Wi-Fi driver will try to scan the configured AP first. The scan in "Wi-Fi Connect" is the same as *Scan for a Specific AP In All Channels*, except that no scan-done event will be generated when the scan is completed. If the target AP is found, then the Wi-Fi driver will start the Wi-Fi connection; otherwise, `<WIFI_EVENT_STA_DISCONNECTED>` will be generated. Refer to *Scan for a Specific AP in All Channels*

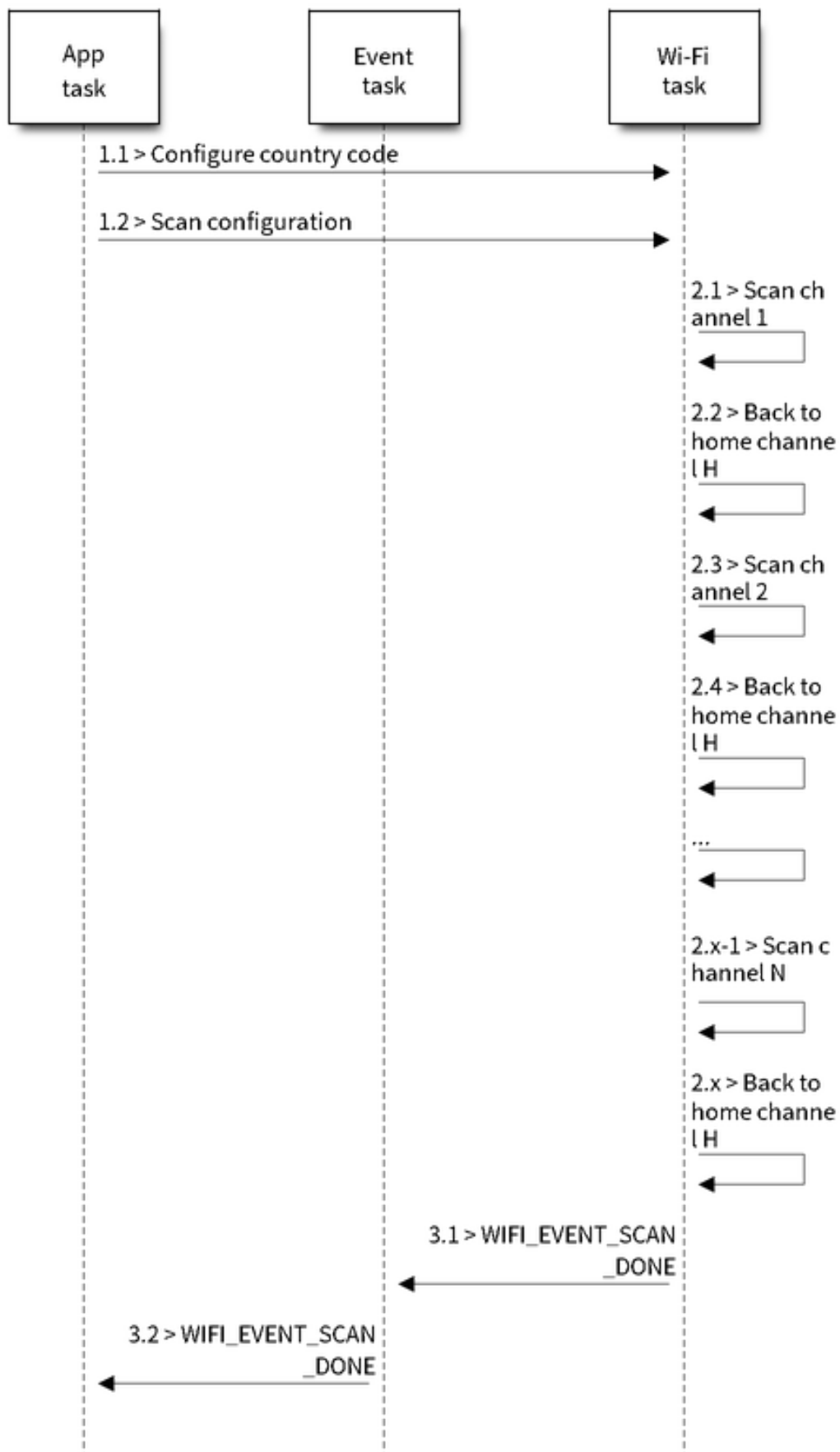


图 28: Background Scan of all Wi-Fi Channels

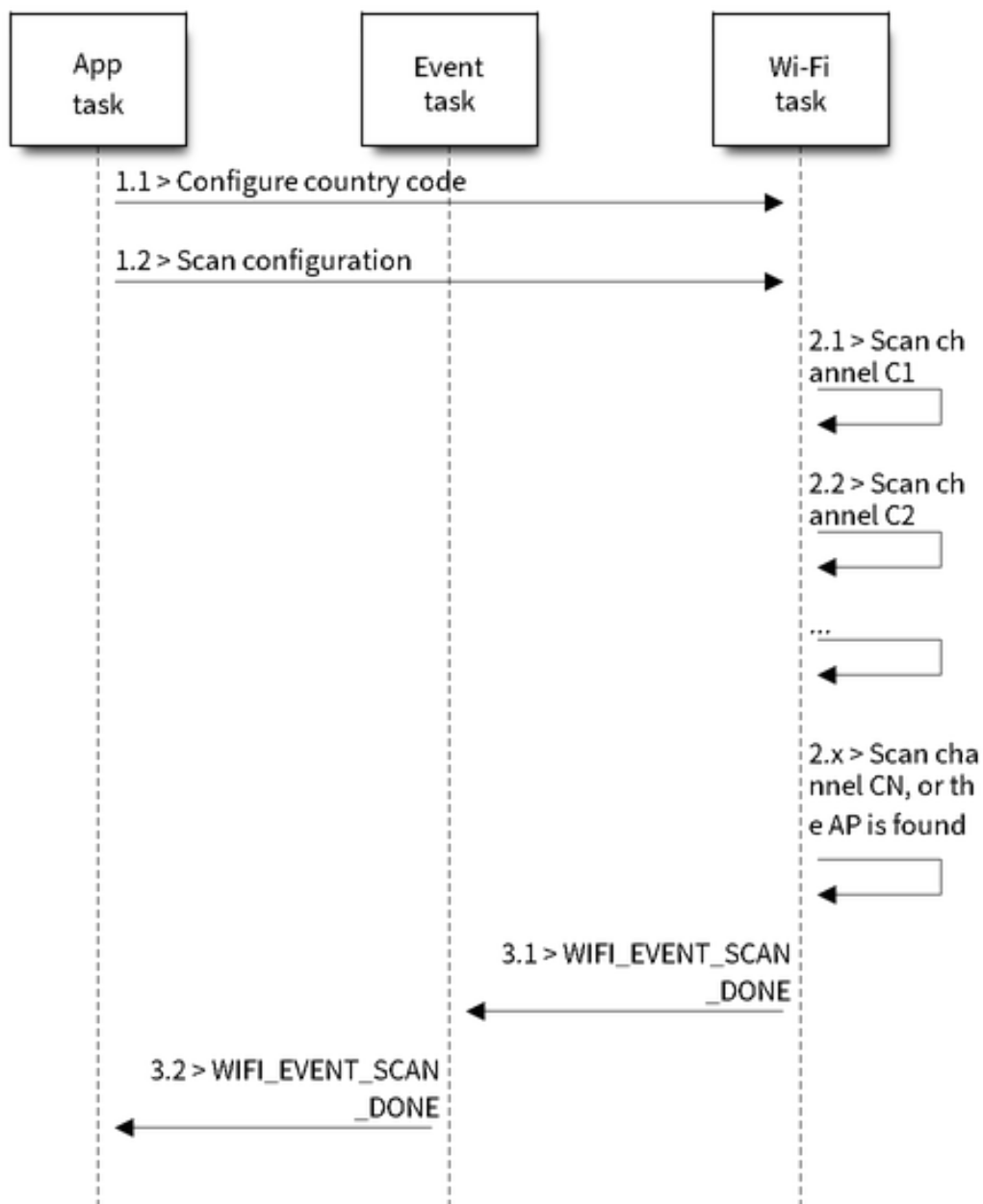


图 29: Scan of specific Wi-Fi Channels

### Scan In Blocked Mode

If the block parameter of `esp_wifi_scan_start()` is true, then the scan is a blocked one, and the application task will be blocked until the scan is done. The blocked scan is similar to an unblocked one, except that no scan-done event will arise when the blocked scan is completed.

### Parallel Scan

Two application tasks may call `esp_wifi_scan_start()` at the same time, or the same application task calls `esp_wifi_scan_start()` before it gets a scan-done event. Both scenarios can happen. **However, the Wi-Fi driver does not support multiple concurrent scans adequately. As a result, concurrent scans should be avoided.** Support for concurrent scan will be enhanced in future releases, as the ESP32's Wi-Fi functionality improves continuously.

### Scan When Wi-Fi Is Connecting

The `esp_wifi_scan_start()` fails immediately if the Wi-Fi is in connecting process because the connecting has higher priority than the scan. If scan fails because of connecting, the recommended strategy is to delay sometime and retry scan again, the scan will succeed once the connecting is completed.

However, the retry/delay strategy may not work all the time. Considering following scenario: - The station is connecting a non-existed AP or if the station connects the existed AP with a wrong password, it always raises the event `<WIFI_EVENT_STA_DISCONNECTED>`. - The application call `esp_wifi_connect()` to do reconnection on receiving the disconnect event. - Another application task, e.g. the console task, call `esp_wifi_scan_start()` to do scan, the scan always fails immediately because the station is keeping connecting. - When scan fails, the application simply delay sometime and retry the scan.

In above scenario the scan will never succeed because the connecting is in process. So if the application supports similar scenario, it needs to implement a better reconnect strategy. E.g. - The application can choose to define a maximum continuous reconnect counter, stop reconnect once the reconnect reaches the max counter. - The application can choose to do reconnect immediately in the first N continuous reconnect, then give a delay sometime and reconnect again.

The application can define its own reconnect strategy to avoid the scan starve to death. Refer to `<Wi-Fi Reconnect>`.

## 4.24.10 ESP32 Wi-Fi Station Connecting Scenario

This scenario only depicts the case when there is only one target AP are found in scan phase, for the scenario that more than one AP with the same SSID are found, refer to `<ESP32 Wi-Fi Station Connecting When Multiple APs Are Found>`.

Generally, the application does not need to care about the connecting process. Below is a brief introduction to the process for those who are really interested.

Scenario:

### Scan Phase

- s1.1, The Wi-Fi driver begins scanning in “Wi-Fi Connect” . Refer to `<Scan in Wi-Fi Connect>` for more details.
- s1.2, If the scan fails to find the target AP, `<WIFI_EVENT_STA_DISCONNECTED>` will arise and the reason-code will be `WIFI_REASON_NO_AP_FOUND`. Refer to `<Wi-Fi Reason Code>`.

### Auth Phase

- s2.1, The authentication request packet is sent and the auth timer is enabled.

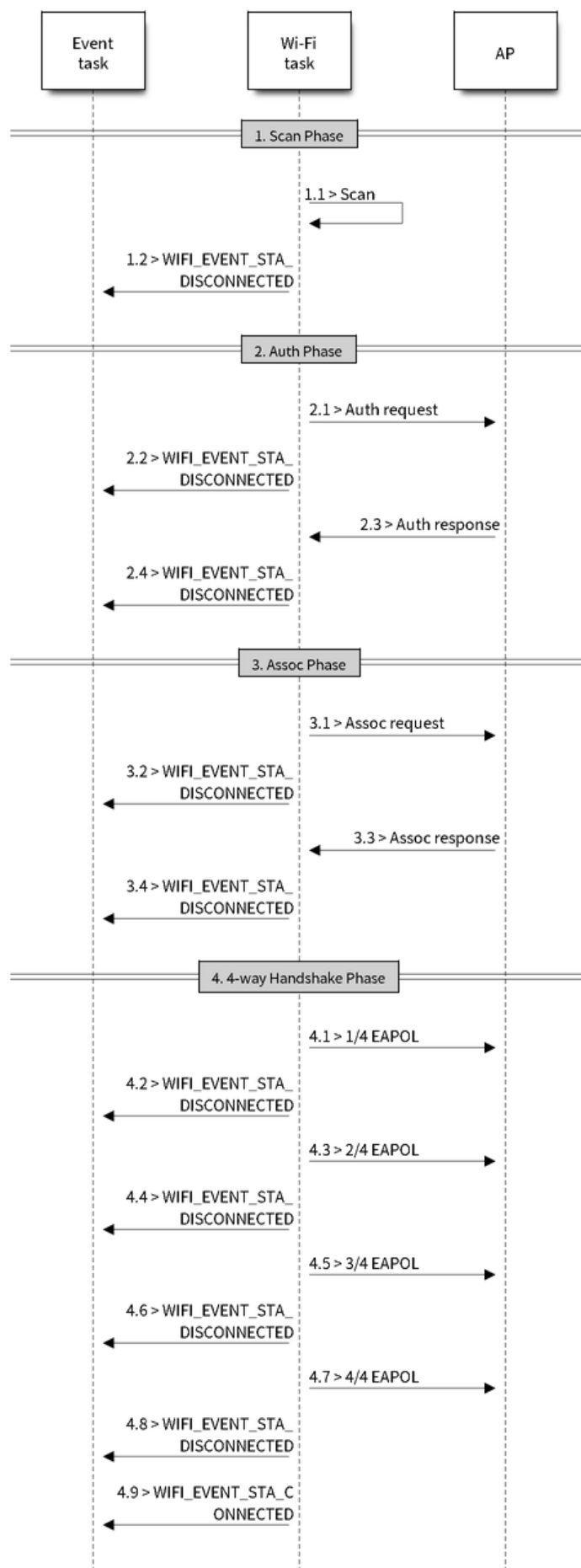


图 30: Wi-Fi Station Connecting Process

- s2.2, If the authentication response packet is not received before the authentication timer times out, `<WIFI_EVENT_STA_DISCONNECTED>` will arise and the reason-code will be `WIFI_REASON_AUTH_EXPIRE`. Refer to [<Wi-Fi Reason Code>](#).
- s2.3, The auth-response packet is received and the auth-timer is stopped.
- s2.4, The AP rejects authentication in the response and `<WIFI_EVENT_STA_DISCONNECTED>` arises, while the reason-code is `WIFI_REASON_AUTH_FAIL` or the reasons specified by the AP. Refer to [<Wi-Fi Reason Code>](#).

### Association Phase

- s3.1, The association request is sent and the association timer is enabled.
- s3.2, If the association response is not received before the association timer times out, `<WIFI_EVENT_STA_DISCONNECTED>` will arise and the reason-code will be `WIFI_REASON_ASSOC_EXPIRE`. Refer to [<Wi-Fi Reason Code>](#).
- s3.3, The association response is received and the association timer is stopped.
- s3.4, The AP rejects the association in the response and `<WIFI_EVENT_STA_DISCONNECTED>` arises, while the reason-code is the one specified in the association response. Refer to [<Wi-Fi Reason Code>](#).

### Four-way Handshake Phase

- s4.1, The four-way handshake is sent out and the association timer is enabled.
- s4.2, If the association response is not received before the association timer times out, `<WIFI_EVENT_STA_DISCONNECTED>` will arise and the reason-code will be `WIFI_REASON_ASSOC_EXPIRE`. Refer to [<Wi-Fi Reason Code>](#).
- s4.3, The association response is received and the association timer is stopped.
- s4.4, The AP rejects the association in the response and `<WIFI_EVENT_STA_DISCONNECTED>` arises and the reason-code will be the one specified in the association response. Refer to [<Wi-Fi Reason Code>](#).

### Wi-Fi Reason Code

The table below shows the reason-code defined in ESP32. The first column is the macro name defined in `esp_wifi_types.h`. The common prefix `WIFI_REASON` is removed, which means that `UNSPECIFIED` actually stands for `WIFI_REASON_UNSPECIFIED` and so on. The second column is the value of the reason. The third column is the standard value to which this reason is mapped in section 8.4.1.7 of `ieee802.11-2012`. (For more information, refer to the standard mentioned above.) The last column is a description of the reason.



Reason code	Value	Mapped To	Description
UNSPECIFIED	1	1	Generally, it means an internal failure, e.g., the memory runs out, the internal TX fails, or the reason is received from the remote side, etc.
AUTH_EXPIRE	2	2	<p>The previous authentication is no longer valid.</p> <p>For the ESP32 Station, this reason is reported when:</p> <ul style="list-style-type: none"> <li>• auth is timed out</li> <li>• the reason is received from the AP.</li> </ul> <p>For the ESP32 AP, this reason is reported when:</p> <ul style="list-style-type: none"> <li>• the AP has not received any packets from the station in the past five minutes.</li> <li>• the AP is stopped by calling <code>esp_wifi_stop()</code>.</li> <li>• the station is deauthenticated by calling <code>esp_wifi_deauth_sta()</code></li> </ul>
AUTH_LEAVE	3	3	<p>De-authenticated, because the sending STA is leaving (or has left).</p> <p>For the ESP32 Station, this reason is reported when:</p> <ul style="list-style-type: none"> <li>• it is received from the AP.</li> </ul>
ASSOC_EXPIRE	4	4	<p>Disassociated due to inactivity.</p> <p>For the ESP32 Station, this reason is reported when:</p> <ul style="list-style-type: none"> <li>• it is received from the AP.</li> </ul> <p>For the ESP32 AP, this reason is reported when:</p> <ul style="list-style-type: none"> <li>• the AP has not received any packets from the station in the past five minutes.</li> <li>• the AP is stopped by calling <code>esp_wifi_stop()</code>.</li> <li>• the station is deauthenticated by calling <code>esp_wifi_deauth_sta()</code></li> </ul>
Espressif Systems		<b>1504</b>	Release v4.0.3
ASSOC_TOOMANY	5	<a href="#">Submit Document Feedback</a>	Disassociated, because the AP is unable to handle all currently associated STAs at the same time.

### 4.24.11 ESP32 Wi-Fi Station Connecting When Multiple APs Are Found

This scenario is similar as [<ESP32 Wi-Fi Station Connecting Scenario>](#), the difference is the station will not raise the event [<WIFI\\_EVENT\\_STA\\_DISCONNECTED>](#) unless it fails to connect all of the found APs.

### 4.24.12 Wi-Fi Reconnect

The station may disconnect due to many reasons, e.g. the connected AP is restarted etc. It's the application's responsibility to do the reconnect. The recommended reconnect strategy is to call `esp_wifi_connect()` on receiving event [<WIFI\\_EVENT\\_STA\\_DISCONNECTED>](#).

Sometimes the application needs more complex reconnect strategy: - If the disconnect event is raised because the `esp_wifi_disconnect()` is called, the application may not want to do reconnect. - If the `esp_wifi_scan_start()` may be called at anytime, a better reconnect strategy is necessary, refer to [<Scan When Wi-Fi Is Connecting>](#).

Another thing we need to consider is the reconnect may not connect the same AP if there are more than one APs with the same SSID. The reconnect always select current best APs to connect.

### 4.24.13 Wi-Fi Beacon Timeout

The beacon timeout mechanism is used by ESP32 station to detect whether the AP is alive or not. If the station continuously loses 60 beacons of the connected AP, the beacon timeout happens.

After the beacon timeout happens, the station sends 5 probe requests to AP, it disconnects the AP and raises the event [<WIFI\\_EVENT\\_STA\\_DISCONNECTED>](#) if still no probe response or beacon is received from AP.

### 4.24.14 ESP32 Wi-Fi Configuration

All configurations will be stored into flash when the Wi-Fi NVS is enabled; otherwise, refer to [<Wi-Fi NVS Flash>](#).

#### Wi-Fi Mode

Call `esp_wifi_set_mode()` to set the Wi-Fi mode.

Mode	Description
WIFI_MODE_NULL	Idle: in this mode, the internal data struct is not allocated to the station and the AP, while both the station and AP interfaces are not initialized for RX/TX Wi-Fi data. Generally, this mode is used for Sniffer, or when you only want to stop both the STA and the AP without calling <code>esp_wifi_deinit()</code> to unload the whole Wi-Fi driver.
WIFI_MODE_STA	Station mode: in this mode, <code>esp_wifi_start()</code> will init the internal station data, while the station's interface is ready for the RX and TX Wi-Fi data. After <code>esp_wifi_connect()</code> is called, the STA will connect to the target AP.
WIFI_MODE_AP	AP mode: in this mode, <code>esp_wifi_start()</code> will init the internal AP data, while the AP's interface is ready for RX/TX Wi-Fi data. Then, the Wi-Fi driver starts broadcasting beacons, and the AP is ready to get connected to other stations.
WIFI_MODE_AP_STA	Station-AP coexistence mode: in this mode, <code>esp_wifi_start()</code> will simultaneously init both the station and the AP. This is done in station mode and AP mode. Please note that the channel of the external AP, which the ESP32 Station is connected to, has higher priority over the ESP32 AP channel.

#### Station Basic Configuration

API `esp_wifi_set_config()` can be used to configure the station. The table below describes the fields in detail.

Field	Description
ssid	This is the SSID of the target AP, to which the station wants to connect to.
password	Password of the target AP
scan_method	If the scan_method is WIFI_FAST_SCAN, the scan ends when the first matched AP is found, for WIFI_ALL_CHANNEL_SCAN, the scan finds all matched APs in all channels. The default scan is WIFI_FAST_SCAN.
bssid	If bssid_set is 0, the station connects to the AP whose SSID is the same as the field “ssid”, while the field “bssid” is ignored. In all other cases, the station connects to the AP whose SSID is the same as the “ssid” field, while its BSSID is the same as the “bssid” field.
bssid	This is valid only when bssid_set is 1; see field “bssid_set”.
channel	If the channel is 0, the station scans the channel 1~N to search for the target AP; otherwise, the station starts by scanning the channel whose value is the same as that of the “channel” field, and then scans others to find the target AP. If you do not know which channel the target AP is running on, set it to 0.
sort_method	This field is only for WIFI_ALL_CHANNEL_SCAN If the sort_method is WIFI_CONNECT_AP_BY_SIGNAL, all matched APs are sorted by signal, for AP with best signal will be connected firstly. E.g. if the station want to connect AP whose ssid is “apxx”, the scan finds two AP whose ssid equals to “apxx”, the first AP’s signal is -90dBm, the second AP’s signal is -30dBm, the station connects the second AP firstly, it doesn’t connect the first one unless it fails to connect the second one. If the sort_method is WIFI_CONNECT_AP_BY_SECURITY, all matched APs are sorted by security. E.g. if the station wants to connect AP whose ssid is “apxx”, the scan finds two AP whose ssid is “apxx”, the security of the first found AP is open while the second one is WPA2, the stations connects to the second AP firstly, it doesn’t connect the second one unless it fails to connect the first one.
threshold	The threshold is used to filter the found AP, if the RSSI or security mode is less than the configured threshold, the AP will be discard. If the RSSI set to 0, it means default threshold, the default RSSI threshold is -127dBm. If the authmode threshold is set to 0, it means default threshold, the default authmode threshold is open.

**注意：** WEP/WPA security modes are deprecated in IEEE802.11-2016 specifications and are recommended not to be used. These modes can be rejected using authmode threshold by setting threshold as WPA2 by threshold.authmode as WIFI\_AUTH\_WPA2\_PSK.

## AP Basic Configuration

API `esp_wifi_set_config()` can be used to configure the AP. The table below describes the fields in detail.

Field	Description
ssid	SSID of AP; if the ssid[0] is 0xFF and ssid[1] is 0xFF, the AP defaults the SSID to ESP_aabcc, where “aabcc” is the last three bytes of the AP MAC.
password	Password of AP; if the auth mode is WIFI_AUTH_OPEN, this field will be ignored.
ssid_len	Length of SSID; if ssid_len is 0, check the SSID until there is a termination character. If ssid_len > 32, change it to 32; otherwise, set the SSID length according to ssid_len.
channel	Channel of AP; if the channel is out of range, the Wi-Fi driver defaults the channel to channel 1. So, please make sure the channel is within the required range. For more details, refer to <Wi-Fi Country Code>.
auth-mode	Auth mode of ESP32 AP; currently, ESP32 Wi-Fi does not support AUTH_WEP. If the authmode is an invalid value, AP defaults the value to WIFI_AUTH_OPEN.
ssid_hidden	If ssid_hidden is 1, AP does not broadcast the SSID; otherwise, it does broadcast the SSID.
max_connection	Currently, ESP32 Wi-Fi supports up to 10 Wi-Fi connections. If max_connection > 10, AP defaults the value to 10.
beacon_interval	Beacon interval; the value is 100 ~ 60000 ms, with default value being 100 ms. If the value is out of range, AP defaults it to 100 ms.

## Wi-Fi Protocol Mode

Currently, the IDF supports the following protocol modes:

Protocol Mode	Description
802.11B	Call <code>esp_wifi_set_protocol(ifx, WIFI_PROTOCOL_11B)</code> to set the station/AP to 802.11B-only mode.
802.11BG	Call <code>esp_wifi_set_protocol(ifx, WIFI_PROTOCOL_11B WIFI_PROTOCOL_11G)</code> to set the station/AP to 802.11BG mode.
802.11BGN	Call <code>esp_wifi_set_protocol(ifx, WIFI_PROTOCOL_11B WIFI_PROTOCOL_11G WIFI_PROTOCOL_11N)</code> to set the station/AP to BGN mode.
802.11BGNLR	Call <code>esp_wifi_set_protocol(ifx, WIFI_PROTOCOL_11B WIFI_PROTOCOL_11G WIFI_PROTOCOL_11N WIFI_PROTOCOL_LR)</code> to set the station/AP to BGN and the Espressif-specific mode.
802.11LR	Call <code>esp_wifi_set_protocol(ifx, WIFI_PROTOCOL_LR)</code> to set the station/AP only to the Espressif-specific mode. <b>This mode is an Espressif-patented mode which can achieve a one-kilometer line of sight range. Please, make sure both the station and the AP are connected to an ESP32 device</b>

## Long Range (LR)

Long Range (LR) mode is an Espressif-patented Wi-Fi mode which can achieve a one-kilometer line of sight range. It has better reception sensitivity, stronger anti-interference ability and longer transmission distance than the traditional 802.11B mode.

**LR Compatibility** Since LR is Espressif unique Wi-Fi mode, only ESP32 devices can transmit and receive the LR data. In other words, the ESP32 device should NOT transmit the data in LR data rate if the connected device doesn't support LR. The application can achieve this by configuring suitable Wi-Fi mode. If the negotiated mode supports LR, the ESP32 may transmit data in LR rate, otherwise, ESP32 will transmit all data in traditional Wi-Fi data rate.

Following table depicts the Wi-Fi mode negotiation:

AP\STA	BGN	BG	B	BGNLR	BGLR	BLR	LR
BGN	BGN	BG	B	BGN	BG	B	•
BG	BG	BG	B	BG	BG	B	•
B	B	B	B	B	B	B	•
BGNLR	•	•	•	BGNLR	BGLR	BLR	LR
BGLR	•	•	•	BGLR	BGLR	BLR	LR
BLR	•	•	•	BLR	BLR	BLR	LR
LR	•	•	•	LR	LR	LR	LR

In above table, the row is the Wi-Fi mode of AP and the column is the Wi-Fi mode of station. The “-” indicates Wi-Fi mode of the AP and station are not compatible.

**According to the table, we can conclude that:**

- For LR enabled in ESP32 AP, it's incompatible with traditional 802.11 mode because the beacon is sent in LR mode.
- For LR enabled in ESP32 station and the mode is NOT LR only mode, it's compatible with traditional 802.11 mode.
- If both station and AP are ESP32 devices and both of them enable LR mode, the negotiated mode supports LR.

If the negotiated Wi-Fi mode supports both traditional 802.11 mode and LR mode, it's the WiFi driver's responsibility to automatically select the best data rate in different Wi-Fi mode and the application don't need to care about it.

### LR Impacts to Traditional Wi-Fi device

**The data transmission in LR rate has no impacts on the traditional Wi-Fi device because:**

- The CCA and backoff process in LR mode are consistent with 802.11 specification.
- The traditional Wi-Fi device can detect the LR signal via CCA and do backoff.

In other words, the impact transmission in LR mode is similar as the impact in 802.11B mode.

**LR Transmission Distance** The reception sensitivity of LR has about 4 dB gain than the traditional 802.11 B mode, theoretically the transmission distance is about 2 to 2.5 times the distance of 11B.

**LR Throughput** The LR rate has very limited throughput because the raw PHY data rate LR is 1/2 Mbits and 1/4 Mbits.

### When to Use LR

**The general conditions for using LR are:**

- Both the AP and station are ESP32 devices.
- Long distance WiFi connection and data transmission is required.
- Data throughput requirements are very small, such as remote device control, etc.

### Wi-Fi Country Code

Call `esp_wifi_set_country()` to set the country info. The table below describes the fields in detail, please consult local 2.4GHz RF operating regulations before configuring these fields.

Field	Description
cc[3]	<p>Country code string, this attributes identify the country or noncountry entity in which the station/AP is operating. If it's a country, the first two octets of this string is the two character country info as described in document ISO/IEC3166-1. The third octect is one of the following:</p> <ul style="list-style-type: none"> <li>• an ASCII space character, if the regulations under which the station/AP is operating encompass all environments for the current frequency band in the country</li> <li>• an ASCII 'O' character if the regulations under which the station/AP is operating are for an outdoor environment only, or</li> <li>• an ASCII 'I' character if the regulations under which the station/AP is operating are for an indoor environment only.</li> <li>• an ASCII 'X' character if the station/AP is operating under a noncountry entity. The first two octets of the noncountry entity is two ASSCII 'XX' characters.</li> <li>• the binary representation of the Operating Class table number currently in use. Refer 802.11-2012 Annex E.</li> </ul>
schan	Start channel, it's the minimum channel number of the regulations under which the station/AP can operate.
snum	Total channel number of the regulations, e.g. if the schan=1, nchan=13, it means the station/AP can send data from channel 1 to 13.
policy	Country policy, this field control which country info will be used if the configured country info is conflict with the connected AP's. More description about policy is provided in following section.

The default country info is {.cc=" CN" , .schan=1, .nchan=13, policy=WIFI\_COUNTRY\_POLICY\_AUTO}, if the WiFi Mode is station/AP coexist mode, they share the same configured country info. Sometimes, the country info of AP, to which the station is connected, is different from the country info of configured. For example, the configured station has country info {.cc=" JP" , .schan=1, .nchan=14, policy=WIFI\_COUNTRY\_POLICY\_AUTO}, but the connected AP has country info {.cc=" CN" , .schan=1, .nchan=13}, then country info of connected AP's is used. Following table depicts which country info is used in different WiFi Mode and different country policy, also describe the impact to active scan.

WiFi Mode	Policy	Description
Station	WIFI_COUNTRY_POLICY_AUTO	<p>If the connected AP has country IE in its beacon, the country info equals to the country info in beacon, otherwise, use default country info.</p> <p>For scan:</p> <ul style="list-style-type: none"> <li>before the station connects to the AP, scans channel “schan” to “min(11, schan+nchan-1)” with active scan and channel min(12, schan+nchan)” to 14 with passive scan. E.g. if the used country info is {.cc=” CN” , .schan=1, .nchan=6} then 1 to 6 is active scan and 7 to 14 is passive scan If the used country info is {.cc=” CN” , .schan=1, .nchan=12} then 1 to 11 is active scan and 12 to 14 is passive scan</li> <li>after the station connects to the AP, scans channel “schan” to “schan+nchan-1” with active scan and channel “schan+nchan” to 14 with passive scan</li> </ul> <p>Always keep in mind that if a AP with with hidden SSID is set to a passive scan channel, the passive scan will not find it. In other words, if the application hopes to find the AP with hidden SSID in every channel, the policy of country info should be configured to WIFI_COUNTRY_POLICY_MANUAL.</p>
Station	WIFI_COUNTRY_POLICY_MANUAL	<p>Always use the configured country info</p> <p>For scan, scans channel “schan” to “schan+nchan-1” with active scan</p>
AP	WIFI_COUNTRY_POLICY_AUTO	Always use the configured country info
AP	WIFI_COUNTRY_POLICY_MANUAL	Always use the configured country info
Station/AP coexit	WIFI_COUNTRY_POLICY_AUTO	<p>If the station doesn't connects to any AP, the AP use the configured country info. If the station connects to an AP, the AP has the same country info as the station.</p> <p>Same as station mode with policy WIFI_COUNTRY_POLICY_AUTO</p>

**Home Channel** In AP mode, the home channel is defined as that of the AP channel. In Station mode, the home channel is defined as the channel of the AP to which the station is connected. In Station+AP mode, the home channel of AP and station must be the same. If the home channels of Station and AP are different, the station's home channel is always in priority. Take the following as an example: at the beginning, the AP is on channel 6, then the

station connects to an AP whose channel is 9. Since the station's home channel has a higher priority, the AP needs to switch its channel from 6 to 9 to make sure that both station and AP have the same home channel.

### Wi-Fi Vendor IE Configuration

By default, all Wi-Fi management frames are processed by the Wi-Fi driver, and the application does not need to care about them. Some applications, however, may have to handle the beacon, probe request, probe response and other management frames. For example, if you insert some vendor-specific IE into the management frames, it is only the management frames which contain this vendor-specific IE that will be processed. In ESP32, `esp_wifi_set_vendor_ie()` and `esp_wifi_set_vendor_ie_cb()` are responsible for this kind of tasks.

### 4.24.15 Wi-Fi Security

In addition to traditional security methods (WEP/WPA-TKIP/WPA2-CCMP), ESP32 Wi-Fi now 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 in traditional modes.

#### Protected Management Frames (PMF)

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 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.

ESP32 supports the following three modes of operation with respect to PMF.

- PMF not supported: In this mode, ESP32 indicates to AP that it is not capable of supporting management protection during association. In effect, security in this mode will be equivalent to that in traditional mode.
- PMF capable, but not required: In this mode, ESP32 indicates to AP that it is capable of supporting PMF. The management protection will be used if AP mandates PMF or is at least capable of supporting PMF.
- PMF capable and required: In this mode, ESP32 will only connect to AP, if AP supports PMF. If not, ESP32 will refuse to connect to the AP.

`esp_wifi_set_config()` can be used to configure PMF mode by setting appropriate flags in `pmf_cfg` parameter. Currently, PMF is supported only in Station mode.

#### WPA3-Personal

Wi-Fi Protected Access-3 (WPA3) is a set of enhancements to Wi-Fi access security intended to replace the current WPA2 standard. In order to provide more robust authentication, 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. WPA3 also 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.

In order to enable WPA3-Personal, "Enable WPA3-Personal" should be selected in menuconfig. If enabled, ESP32 uses SAE for authentication if supported by the AP. Since PMF is a mandatory requirement for WPA3, PMF capability should be at least set to "PMF capable, but not required" for ESP32 to use WPA3 mode. Application developers need not worry about the underlying security mode as highest available is chosen from security standpoint. Note that Wi-Fi stack size requirement will increase approximately by 3k when WPA3 is used. Currently, WPA3 is supported only in Station mode.

#### 4.24.16 ESP32 Wi-Fi Power-saving Mode

##### Station Sleep

Currently, ESP32 Wi-Fi supports the Modem-sleep mode which refers to the legacy power-saving mode in the IEEE 802.11 protocol. Modem-sleep mode works in Station-only mode and the station must connect to the AP first. If the Modem-sleep mode is enabled, station will switch between active and sleep state periodically. In sleep state, RF, PHY and BB are turned off in order to reduce power consumption. Station can keep connection with AP in modem-sleep mode.

Modem-sleep mode includes minimum and maximum power save modes. In minimum power save mode, station wakes up every DTIM to receive beacon. Broadcast data will not be lost because it is transmitted after DTIM. However, it can not save much more power if DTIM is short for DTIM is determined by AP.

In maximum power save mode, station wakes up every listen interval to receive beacon. This listen interval can be set longer than the AP DTIM period. Broadcast data may be lost because station may be in sleep state at DTIM time. If listen interval is longer, more power is saved but broadcast data is more easy to lose. Listen interval can be configured by calling API `esp_wifi_set_config()` before connecting to AP.

Call `esp_wifi_set_ps(WIFI_PS_MIN_MODEM)` to enable Modem-sleep minimum power save mode or `esp_wifi_set_ps(WIFI_PS_MAX_MODEM)` to enable Modem-sleep maximum power save mode after calling `esp_wifi_init()`. When station connects to AP, Modem-sleep will start. When station disconnects from AP, Modem-sleep will stop.

Call `esp_wifi_set_ps(WIFI_PS_NONE)` to disable modem sleep entirely. This has much higher power consumption, but provides minimum latency for receiving Wi-Fi data in real time. When modem sleep is enabled, received Wi-Fi data can be delayed for as long as the DTIM period (minimum power save mode) or the listen interval (maximum power save mode). Disabling modem sleep entirely is not possible for Wi-Fi and Bluetooth coexist mode.

The default Modem-sleep mode is `WIFI_PS_MIN_MODEM`.

##### AP Sleep

Currently ESP32 AP doesn't support all of the power save feature defined in Wi-Fi specification. To be specific, the AP only caches unicast data for the stations connect to this AP, but doesn't cache the multicast data for the stations. If stations connected to the ESP32 AP are power save enabled, they may experience multicast packet loss.

In future, all power save features will be supported on ESP32 AP.

#### 4.24.17 ESP32 Wi-Fi Connect Crypto

Now ESP32 have two group crypto functions can be used when do wifi connect, one is the original functions, the other is optimized by ESP hardware: 1. Original functions which is the source code used in the folder `components/wpa_supplicant/src/crypto` function; 2. The optimized functions is in the folder `components/wpa_supplicant/src/fast_crypto`, these function used the hardware crypto to make it faster than origin one, the type of function's name add `fast_` to distinguish with the original one. For example, the API `aes_wrap()` is used to encrypt frame information when do 4 way handshake, the `fast_aes_wrap()` has the same result but can be faster.

Two groups of crypto function can be used when register in the `wpa_crypto_funcs_t`, `wpa2_crypto_funcs_t` and `wps_crypto_funcs_t` structure, also we have given the recommend functions to register in the `fast_crypto_ops.c`, you can register the function as the way you need, however what should make action is that the `crypto_hash_xxx` function and `crypto_cipher_xxx` function need to register with the same function to operation. For example, if you register `crypto_hash_init()` function to initialize the `esp_crypto_hash` structure, you need use the `crypto_hash_update()` and `crypto_hash_finish()` function to finish the operation, rather than `fast_crypto_hash_update()` or `fast_crypto_hash_finish()`.

#### 4.24.18 ESP32 Wi-Fi Throughput

The table below shows the best throughput results we got in Espressif's lab and in a shield box.

Type/Throughput	Air In Lab	Shield-box	Test Tool	IDF Version (commit ID)
Raw 802.11 Packet RX	N/A	<b>130 MBit/sec</b>	Internal tool	NA
Raw 802.11 Packet TX	N/A	<b>130 MBit/sec</b>	Internal tool	NA
UDP RX	30 MBit/sec	90 MBit/sec	iperf example	05838641
UDP TX	30 MBit/sec	60 MBit/sec	iperf example	05838641
TCP RX	20 MBit/sec	50 MBit/sec	iperf example	05838641
TCP TX	20 MBit/sec	50 MBit/sec	iperf example	05838641

When the throughput is tested by iperf example, the sdkconfig is [examples/wifi/iperf/sdkconfig.defaults.99](#)

#### 4.24.19 Wi-Fi 80211 Packet Send

The `esp_wifi_80211_tx()` API can be used to:

- Send the beacon, probe request, probe response, action frame.
- Send the non-QoS data frame.

It cannot be used for sending encrypted or QoS frames.

#### Preconditions of Using `esp_wifi_80211_tx`

- The Wi-Fi mode is Station, or AP, or Station+AP.
- Either `esp_wifi_set_promiscuous(true)`, or `esp_wifi_start()`, or both of these APIs return `ESP_OK`. This is because we need to make sure that Wi-Fi hardware is initialized before `esp_wifi_80211_tx()` is called. In ESP32, both `esp_wifi_set_promiscuous(true)` and `esp_wifi_start()` can trigger the initialization of Wi-Fi hardware.
- The parameters of `esp_wifi_80211_tx` are hereby correctly provided.

#### Data rate

- If there is no WiFi connection, the data rate is 1Mbps.
- If there is WiFi connection and the packet is from station to AP or from AP to station, the data rate is same as the WiFi connection. Otherwise the data rate is 1Mbps.

#### Side-Effects to Avoid in Different Scenarios

Theoretically, if we do not consider the side-effects the API imposes on the Wi-Fi driver or other stations/APs, we can send a raw 802.11 packet over the air, with any destination MAC, any source MAC, any BSSID, or any other type of packet. However, robust/useful applications should avoid such side-effects. The table below provides some tips/recommendations on how to avoid the side-effects of `esp_wifi_80211_tx` in different scenarios.

Scenario	Description
No WiFi connection	<p>In this scenario, no Wi-Fi connection is set up, so there are no side-effects on the Wi-Fi driver. If <code>en_sys_seq==true</code>, the Wi-Fi driver is responsible for the sequence control. If <code>en_sys_seq==false</code>, the application needs to ensure that the buffer has the correct sequence.</p> <p>Theoretically, the MAC address can be any address. However, this may impact other stations/APs with the same MAC/BSSID.</p> <p>Side-effect example#1 The application calls <code>esp_wifi_80211_tx</code> to send a beacon with <code>BSSID == mac_x</code> in AP mode, but the <code>mac_x</code> is not the MAC of the AP interface. Moreover, there is another AP, say “other-AP”, whose <code>bssid</code> is <code>mac_x</code>. If this happens, an “unexpected behavior” may occur, because the stations which connect to the “other-AP” cannot figure out whether the beacon is from the “other-AP” or the <code>esp_wifi_80211_tx</code>.</p> <p>To avoid the above-mentioned side-effects, we recommend that:</p> <ul style="list-style-type: none"> <li>• If <code>esp_wifi_80211_tx</code> is called in Station mode, the first MAC should be a multicast MAC or the exact target-device’s MAC, while the second MAC should be that of the station interface.</li> <li>• If <code>esp_wifi_80211_tx</code> is called in AP mode, the first MAC should be a multicast MAC or the exact target-device’s MAC, while the second MAC should be that of the AP interface.</li> </ul> <p>The recommendations above are only for avoiding side-effects and can be ignored when there are good reasons for doing this.</p>
Have WiFi connection	<p>When the Wi-Fi connection is already set up, and the sequence is controlled by the application, the latter may impact the sequence control of the Wi-Fi connection, as a whole. So, the <code>en_sys_seq</code> need to be true, otherwise <code>ESP_ERR_WIFI_ARG</code> is returned.</p> <p>The MAC-address recommendations in the “No WiFi connection” scenario also apply to this scenario.</p> <p>If the WiFi mode is station mode and the MAC address1 is the MAC of AP to which the station is connected, the MAC address2 is the MAC of station interface, we say the packets is from the station to AP. On the other hand, if the WiFi mode is AP mode and the MAC address1 is the MAC of the station who connects to this AP, the MAC address2 is the MAC of AP interface, we say the packet is from the AP to station. To avoid conflicting with WiFi connections, the following checks are applied:</p> <ul style="list-style-type: none"> <li>• If the packet type is data and is from the station to AP, the ToDS bit in <code>ieee80211</code> frame control should be 1, the FromDS bit should be 0, otherwise the packet will be discarded by WiFi driver.</li> <li>• If the packet type is data and is from the AP to station, the ToDS bit in <code>ieee80211</code> frame control should be 0, the FromDS bit should be 1, otherwise the packet will be discarded by WiFi driver.</li> <li>• If the packet is from station to AP or from AP to station, the Power Management, More Data,</li> </ul>
Espressif Systems	<p>1514 Re-Transmission bits should be 0, otherwise the packet will be discarded by WiFi driver. <span style="float: right;">Release 4.0.3</span></p> <p>ESP_ERR_WIFI_ARG is returned if any check fails.</p>

### 4.24.20 Wi-Fi Sniffer Mode

The Wi-Fi sniffer mode can be enabled by `esp_wifi_set_promiscuous()`. If the sniffer mode is enabled, the following packets **can** be dumped to the application:

- 802.11 Management frame
- 802.11 Data frame, including MPDU, AMPDU, AMSDU, etc.
- 802.11 MIMO frame, for MIMO frame, the sniffer only dumps the length of the frame.
- 802.11 Control frame

The following packets will **NOT** be dumped to the application:

- 802.11 error frame, such as the frame with a CRC error, etc.

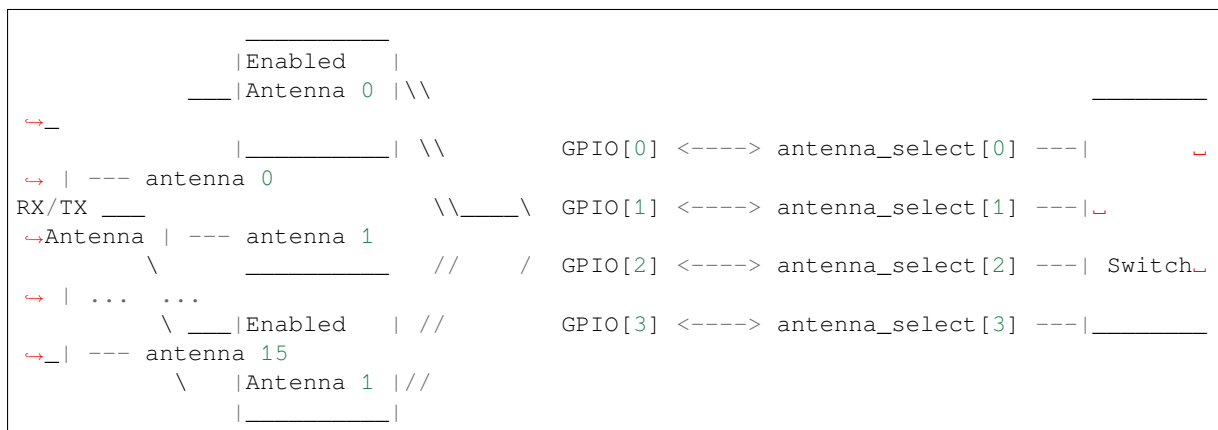
For frames that the sniffer **can** dump, the application can additionally decide which specific type of packets can be filtered to the application by using `esp_wifi_set_promiscuous_filter()` and `esp_wifi_set_promiscuous_ctrl_filter()`. By default, it will filter all 802.11 data and management frames to the application.

The Wi-Fi sniffer mode can be enabled in the Wi-Fi mode of `WIFI_MODE_NULL`, or `WIFI_MODE_STA`, or `WIFI_MODE_AP`, or `WIFI_MODE_APSTA`. In other words, the sniffer mode is active when the station is connected to the AP, or when the AP has a Wi-Fi connection. Please note that the sniffer has a **great impact** on the throughput of the station or AP Wi-Fi connection. Generally, we should **NOT** enable the sniffer, when the station/AP Wi-Fi connection experiences heavy traffic unless we have special reasons.

Another noteworthy issue about the sniffer is the callback `wifi_promiscuous_cb_t`. The callback will be called directly in the Wi-Fi driver task, so if the application has a lot of work to do for each filtered packet, the recommendation is to post an event to the application task in the callback and defer the real work to the application task.

### 4.24.21 Wi-Fi Multiple Antennas

The Wi-Fi multiple antennas selecting can be depicted as following picture:



ESP32 supports up to sixteen antennas through external antenna switch. The antenna switch can be controlled by up to four address pins - `antenna_select[0:3]`. Different input value of `antenna_select[0:3]` means selecting different antenna. E.g. the value '0b1011' means the antenna 11 is selected. The default value of `antenna_select[3:0]` is '0b0000', it means the antenna 0 is selected by default.

Up to four GPIOs are connected to the four active high `antenna_select` pins. ESP32 can select the antenna by control the GPIO[0:3]. The API `esp_wifi_set_ant_gpio()` is used to configure which GPIOs are connected to `antenna_selects`. If `GPIO[x]` is connected to `antenna_select[x]`, then `gpio_config->gpio_cfg[x].gpio_select` should be set to 1 and `gpio_config->gpio_cfg[x].gpio_num` should be provided.

Although up to sixteen antennas are supported, only one or two antennas can be simultaneously enabled for RX/TX. The API `esp_wifi_set_ant()` is used to configure which antennas are enabled.

The enabled antennas selecting algorithm is also configured by `esp_wifi_set_ant()`. The RX/TX antenna mode can be `WIFI_ANT_MODE_ANT0`, `WIFI_ANT_MODE_ANT1` or `WIFI_ANT_MODE_AUTO`. If the antenna mode is `WIFI_ANT_MODE_ANT0`, the enabled antenna 0 is selected for RX/TX data. If the antenna mode

is `WIFI_ANT_MODE_ANT1`, the enabled antenna 1 is selected for RX/TX data. Otherwise, WiFi automatically selects the antenna that has better signal from the enabled antennas.

If the RX antenna mode is `WIFI_ANT_MODE_AUTO`, the default antenna mode also needs to be set. Because the RX antenna switching only happens when some conditions are met, e.g. the RX antenna starts to switch if the RSSI is lower than  $-65\text{dBm}$  and if another antenna has better signal etc, RX uses the default antenna if the conditions are not met. If the default antenna mode is `WIFI_ANT_MODE_ANT1`, the enabled antenna 1 is used as the default RX antenna, otherwise the enabled antenna 0 is used as the default RX antenna.

**Some limitations need to be considered:**

- The TX antenna can be set to `WIFI_ANT_MODE_AUTO` only if the RX antenna mode is `WIFI_ANT_MODE_AUTO` because TX antenna selecting algorithm is based on RX antenna in `WIFI_ANT_MODE_AUTO` type.
- Currently BT doesn't support the multiple antennas feature, please don't use multiple antennas related APIs.

**Following is the recommended scenarios to use the multiple antennas:**

- In Wi-Fi mode `WIFI_MODE_STA`, both RX/TX antenna modes are configured to `WIFI_ANT_MODE_AUTO`. The WiFi driver selects the better RX/TX antenna automatically.
- The RX antenna mode is configured to `WIFI_ANT_MODE_AUTO`. The TX antenna mode is configured to `WIFI_ANT_MODE_ANT0` or `WIFI_ANT_MODE_ANT1`. The applications can choose to always select a specified antenna for TX, or implement their own TX antenna selecting algorithm, e.g. selecting the TX antenna mode based on the channel switch information etc.
- Both RX/TX antenna modes are configured to `WIFI_ANT_MODE_ANT0` or `WIFI_ANT_MODE_ANT1`.

## Wi-Fi Multiple Antennas Configuration

**Generally, following steps can be taken to configure the multiple antennas:**

- Configure which GPIOs are connected to the antenna\_selects, for example, if four antennas are supported and GPIO20/GPIO21 are connected to antenna\_select[0]/antenna\_select[1], the configurations look like:

```
wifi_ant_gpio_config_t config = {
    { .gpio_select = 1, .gpio_num = 20 },
    { .gpio_select = 1, .gpio_num = 21 }
};
```

- Configure which antennas are enabled and how RX/TX use the enabled antennas, for example, if antenna1 and antenna3 are enabled, the RX needs to select the better antenna automatically and uses antenna1 as its default antenna, the TX always selects the antenna3. The configuration looks like:

```
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
};
```

### 4.24.22 Wi-Fi Channel State Information

Channel state information (CSI) refers to the channel information of a Wi-Fi connection. In ESP32, this information consists of channel frequency responses of sub-carriers and is estimated when packets are received from the transmitter. Each channel frequency response of sub-carrier is recorded by two bytes of signed characters. The first one is imaginary part and the second one is real part. There are up to three fields of channel frequency responses according to the type of received packet. They are legacy long training field (LLTF), high throughput LTF (HT-LTF) and space time block code HT-LTF (STBC-HT-LTF). For different types of packets which are received on channels with different state, the sub-carrier index and total bytes of signed characters of CSI is shown in the following table.

channel information	secondary channel mode	anyone			below					above				
		non HT	HT		non HT	HT		non HT	HT		non HT	HT		
channel band- width	STBC	20MHz	20MHz		20MHz	20MHz		40MHz			20MHz	20MHz		40MHz
		non STBC	non STBC	STBC	non STBC	non STBC	STBC	non STBC	STBC	non STBC	non STBC	STBC	non STBC	STBC
sub- carrier index	LLTF	0~31, 32~ 1	0~31, 32~ 1	0~31, 32~ 1	0~63	0~63	0~63	0~63	0~63	-	-	-	-	-
	HT- LTF	•	0~31, 32~ 1	0~31, 32~ 1	•	0~63	0~62	0~63, 64~ 1	0~60, 60~ 1	•	-	-	0~63, 64~ 1	0~60, 60~ 1
	STBC- HT- LTF	•	•	0~31, 32~ 1	•	•	0~62	•	0~60, 60~ 1	•	•	-	62~ 1	•
total bytes		128	256	384	128	256	380	384	612	128	256	376	384	612

All of the information in the table can be found in the structure `wifi_csi_info_t`.

- Secondary channel refers to `secondary_channel` field of `rx_ctrl` field.
- Signal mode of packet refers to `sig_mode` field of `rx_ctrl` field.
- Channel bandwidth refers to `cwb` field of `rx_ctrl` field.
- STBC refers to `stbc` field of `rx_ctrl` field.
- Total bytes refers to `len` field.
- The CSI data corresponding to each Long Training Field(LTF) type is stored in a buffer starting from the `buf` field. Each item is stored as two bytes: imaginary part followed by real part. The order of each item is the same as the sub-carrier in the table. The order of LTF is: LLTF, HT-LTF, STBC-HT-LTF. However all 3 LTFs may not be present, depending on the channel and packet information (see above).
- If `first_word_invalid` field of `wifi_csi_info_t` is true, it means that the first four bytes of CSI data is invalid due to a hardware limitation in ESP32.
- More information like RSSI, noise floor of RF, receiving time and antenna is in the `rx_ctrl` field.

#### 注解:

- For STBC packet, CSI is provided for every space-time stream without CSD (cyclic shift delay). As each cyclic shift on the additional chains shall be -200ns, only the CSD angle of first space-time stream is recorded in sub-carrier 0 of HT-LTF and STBC-HT-LTF for there is no channel frequency response in sub-carrier 0. CSD[10:0] is 11 bits, ranging from  $-\pi$  to  $\pi$ .
- If LLTF, HT-LTF or STBC-HT-LTF is not enabled by calling API `esp_wifi_set_csi_config()`, the total bytes of CSI data will be fewer than that in the table. For example, if LLTF and HT-LTF is not enabled and STBC-HT-LTF is enabled, when a packet is received with the condition above/HT/40MHz/STBC, the total bytes of CSI data is 244  $((61 + 60) * 2 + 2 = 244$ , the result is aligned to four bytes and the last two bytes is invalid).

#### 4.24.23 Wi-Fi Channel State Information Configure

To use Wi-Fi CSI, the following steps need to be done.

- Select Wi-Fi CSI in menuconfig. It is “Menuconfig → Components config → Wi-Fi → WiFi CSI(Channel State Information)” .
- Set CSI receiving callback function by calling API `esp_wifi_set_csi_rx_cb()`.

- Configure CSI by calling API `esp_wifi_set_csi_config()`.
- Enable CSI by calling API `esp_wifi_set_csi()`.

The CSI receiving callback function runs from Wi-Fi task. So, do not do lengthy operations in the callback function. Instead, post necessary data to a queue and handle it from a lower priority task. Because station does not receive any packet when it is disconnected and only receives packets from AP when it is connected, it is suggested to enable sniffer mode to receive more CSI data by calling `esp_wifi_set_promiscuous()`.

#### 4.24.24 Wi-Fi HT20/40

ESP32 supports Wi-Fi bandwidth HT20 or HT40, it doesn't support HT20/40 coexist. `esp_wifi_set_bandwidth` can be used to change the default bandwidth of station or AP. The default bandwidth for ESP32 station and AP is HT40.

In station mode, the actual bandwidth is firstly negotiated during the Wi-Fi connection. It is HT40 only if both the station and the connected AP support HT40, otherwise it's HT20. If the bandwidth of connected AP is changes, the actual bandwidth is negotiated again without Wi-Fi disconnecting.

Similarly, in AP mode, the actual bandwidth is negotiated between AP and the stations that connect to the AP. It's HT40 if the AP and one of the stations support HT40, otherwise it's HT20.

In station/AP coexist mode, the station/AP can configure HT20/40 separately. If both station and AP are negotiated to HT40, the HT40 channel should be the channel of station because the station always has higher priority than AP in ESP32. E.g. the configured bandwidth of AP is HT40, the configured primary channel is 6 and the configured secondary channel is 10. The station is connected to an router whose primary channel is 6 and secondary channel is 2, then the actual channel of AP is changed to primary 6 and secondary 2 automatically.

Theoretically the HT40 can gain better throughput because the maximum raw physical (PHY) data rate for HT40 is 150Mbps while it's 72Mbps for HT20. However, if the device is used in some special environment, e.g. there are too many other Wi-Fi devices around the ESP32 device, the performance of HT40 may be degraded. So if the applications need to support same or similar scenarios, it's recommended that the bandwidth is always configured to HT20.

#### 4.24.25 Wi-Fi QoS

ESP32 supports all the mandatory features required in WFA Wi-Fi QoS Certification.

Four ACs(Access Category) are defined in Wi-Fi specification, each AC has a its own priority to access the Wi-Fi channel. Moreover a map rule is defined to map the QoS priority of other protocol, such as 802.11D or TCP/IP precedence to Wi-Fi AC.

Below is a table describes how the IP Precedences are mapped to Wi-Fi ACs in ESP32, it also indicates whether the AMPDU is supported for this AC. The table is sorted with priority descending order, namely, the AC\_VO has highest priority.

IP Precedence	Wi-Fi AC	Support AMPDU?
6, 7	AC_VO (Voice)	No
4, 5	AC_VI (Video)	Yes
3, 0	AC_BE (Best Effort)	Yes
1, 2	AC_BK (Background)	Yes

The application can make use of the QoS feature by configuring the IP precedence via socket option IP\_TOS. Here is an example to make the socket to use VI queue:

```
const int ip_precedence_vi = 4;
const int ip_precedence_offset = 5;
int priority = (ip_precedence_vi << ip_precedence_offset);
setsockopt(socket_id, IPPROTO_IP, IP_TOS, &priority, sizeof(priority));
```

**Theoretically the higher priority AC has better performance than the low priority AC, however, it's not always be true, here**

- For some really important application traffic, can put it into AC\_VO queue. Avoid sending big traffic via AC\_VO queue. On one hand, the AC\_VO queue doesn't support AMPDU and it can't get better performance than other queue if the traffic is big, on the other hand, it may impact the the management frames that also use AC\_VO queue.
- Avoid using more than two different AMPDU supported precedences, e.g. socket A uses precedence 0, socket B uses precedence 1, socket C uses precedence 2, this is a bad design because it may need much more memory. To be detailed, the Wi-Fi driver may generate a Block Ack session for each precedence and it needs more memory if the Block Ack session is setup.

#### 4.24.26 Wi-Fi AMSDU

ESP32 supports receiving AMSDU but doesn't support transmitting AMSDU. The transmitting AMSDU is not necessary since ESP32 has transmitting AMPDU.

#### 4.24.27 Wi-Fi Fragment

ESP32 supports Wi-Fi receiving fragment, but doesn't support Wi-Fi transmitting fragment. The Wi-Fi transmitting fragment will be supported in future release.

#### 4.24.28 WPS Enrolle

ESP32 supports WPS enrollee feature in Wi-Fi mode WIFI\_MODE\_STA or WIFI\_MODE\_APSTA. Currently ESP32 supports WPS enrollee type PBC and PIN.

#### 4.24.29 Wi-Fi Buffer Usage

This section is only about the dynamic buffer configuration.

##### Why Buffer Configuration Is Important

**In order to get a robust, high-performance system, we need to consider the memory usage/configuration very carefully, because**

- the available memory in ESP32 is limited.
- currently, the default type of buffer in LwIP and Wi-Fi drivers is “dynamic” , **which means that both the LwIP and Wi-Fi share memory with the application**. Programmers should always keep this in mind; otherwise, they will face a memory issue, such as “running out of heap memory” .
- it is very dangerous to run out of heap memory, as this will cause ESP32 an “undefined behavior” . Thus, enough heap memory should be reserved for the application, so that it never runs out of it.
- the Wi-Fi throughput heavily depends on memory-related configurations, such as the TCP window size, Wi-Fi RX/TX dynamic buffer number, etc.
- the peak heap memory that the ESP32 LwIP/Wi-Fi may consume depends on a number of factors, such as the maximum TCP/UDP connections that the application may have, etc.
- the total memory that the application requires is also an important factor when considering memory configuration.

Due to these reasons, there is not a good-for-all application configuration. Rather, we have to consider memory configurations separately for every different application.

##### Dynamic vs. Static Buffer

The default type of buffer in LwIP and Wi-Fi drivers is “dynamic” . Most of the time the dynamic buffer can significantly save memory. However, it makes the application programming a little more difficult, because in this case the application needs to consider memory usage in LwIP/Wi-Fi.



### Peak LwIP Dynamic Buffer

The default type of LwIP buffer is “dynamic”, and this section considers the dynamic buffer only. The peak heap memory that LwIP consumes is the **theoretically-maximum memory** that the LwIP driver consumes. Generally, the peak heap memory that the 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`
- 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:

$$\text{lwip\_dynamic\_peek\_memory} = (\text{lwip\_udp\_con\_num} * \text{lwip\_udp\_conn}) + (\text{lwip\_tcp\_con\_num} * (\text{lwip\_tcp\_tx\_win\_size} + \text{lwip\_tcp\_rx\_win\_size} + \text{lwip\_tcp\_conn}))$$

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).

### Peak Wi-Fi Dynamic Buffer

The Wi-Fi driver supports several types of buffer (refer to [Wi-Fi Buffer Configure](#)). However, this section is about the usage of the dynamic Wi-Fi buffer only. The peak heap memory that Wi-Fi consumes is the **theoretically-maximum memory** that the Wi-Fi driver consumes. Generally, the peak memory depends on:

- the number of dynamic rx buffers that are configured: `wifi_rx_dynamic_buf_num`
- the number of dynamic tx buffers that are configured: `wifi_tx_dynamic_buf_num`
- the maximum packet size that the Wi-Fi driver can receive: `wifi_rx_pkt_size_max`
- the maximum packet size that the Wi-Fi driver can send: `wifi_tx_pkt_size_max`

So, the peak memory that the Wi-Fi driver consumes can be calculated with the following formula:

$$\text{wifi\_dynamic\_peek\_memory} = (\text{wifi\_rx\_dynamic\_buf\_num} * \text{wifi\_rx\_pkt\_size\_max}) + (\text{wifi\_tx\_dynamic\_buf\_num} * \text{wifi\_tx\_pkt\_size\_max})$$

Generally, we do not need to care about the dynamic tx long buffers and dynamic tx long long buffers, because they are management frames which only have a small impact on the system.

## 4.24.30 Wi-Fi Menuconfig

### Wi-Fi Buffer Configure

If you are going to modify the default number or type of buffer, it would be helpful to also have an overview of how the buffer is allocated/freed in the data path. The following diagram shows this process in the TX direction:

#### Description:

- The application allocates the data which needs to be sent out.
- The application calls TCPIP-/Socket-related APIs to send the user data. These APIs will allocate a PBUF used in LwIP, and make a copy of the user data.
- When LwIP calls a Wi-Fi API to send the PBUF, the Wi-Fi API will allocate a “Dynamic Tx Buffer” or “Static Tx Buffer”, make a copy of the LwIP PBUF, and finally send the data.

The following diagram shows how buffer is allocated/freed in the RX direction:

#### Description:

- The Wi-Fi hardware receives a packet over the air and puts the packet content to the “Static Rx Buffer”, which is also called “RX DMA Buffer”.

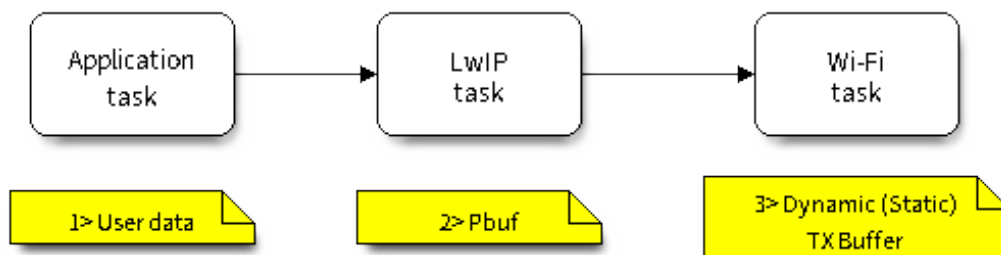


图 31: TX Buffer Allocation

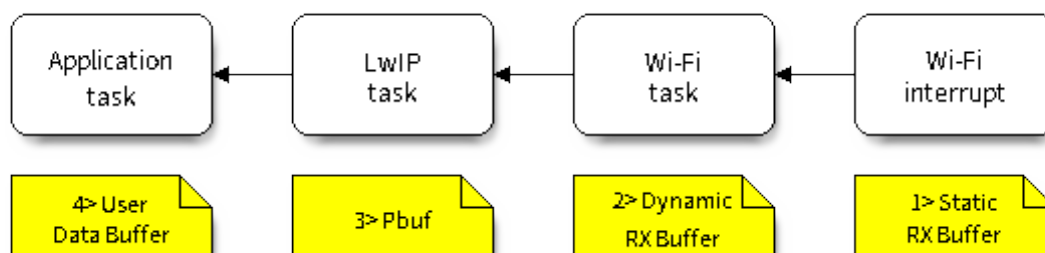


图 32: RX Buffer Allocation

- The Wi-Fi driver allocates a “Dynamic Rx Buffer” , makes a copy of the “Static Rx Buffer” , and returns the “Static Rx Buffer” to hardware.
- The Wi-Fi driver delivers the packet to the upper-layer (LwIP), and allocates a PBUF for holding the “Dynamic Rx Buffer” .
- The application receives data from LwIP.

The diagram shows the configuration of the Wi-Fi internal buffer.

Buffer Type	Alloc Type	Default	Configurable	Description
Static RX Buffer (Hardware RX Buffer)	Static	10 * 1600 Bytes	Yes	<p>This is a kind of DMA memory. It is initialized in <code>esp_wifi_init()</code> and freed in <code>esp_wifi_deinit()</code>. The 'Static Rx Buffer' forms the hardware receiving list. Upon receiving a frame over the air, hardware writes the frame into the buffer and raises an interrupt to the CPU. Then, the Wi-Fi driver reads the content from the buffer and returns the buffer back to the list.</p> <p>If the application want to reduce the the memory statically allocated by Wi-Fi, they can reduce this value from 10 to 6 to save 6400 Bytes memory. It's not recommended to reduce the configuration to a value less than 6 unless the AMPDU feature is disabled.</p>
Dynamic RX Buffer	Dynamic	32	Yes	<p>The buffer length is variable and it depends on the received frames' length. When the Wi-Fi driver receives a frame from the 'Hardware Rx Buffer', the 'Dynamic Rx Buffer' needs to be allocated from the heap. The number of the Dynamic Rx Buffer, configured in the <code>menuconfig</code>, is used to limit the total un-freed Dynamic Rx Buffer number.</p>
Dynamic TX Buffer	Dynamic	32	Yes	<p>This is a kind of DMA memory. It is allocated to the heap. When the upper-layer (LwIP) sends packets to the Wi-Fi driver, it firstly allocates</p>
Espressif Systems		1522		Release v4.0.3

## Wi-Fi NVS Flash

If the Wi-Fi NVS flash is enabled, all Wi-Fi configurations set via the Wi-Fi APIs will be stored into flash, and the Wi-Fi driver will start up with these configurations next time it powers on/reboots. However, the application can choose to disable the Wi-Fi NVS flash if it does not need to store the configurations into persistent memory, or has its own persistent storage, or simply due to debugging reasons, etc.

## Wi-Fi AMPDU

ESP32 supports both receiving and transmitting AMPDU, the AMPDU can greatly improve the Wi-Fi throughput. Generally, the AMPDU should be enabled. Disabling AMPDU is usually for debugging purposes.

### 4.24.31 Troubleshooting

Please refer to a separate document with [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截取网络封包, 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, 或是专门属于营利用的软件。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”？

- **捕捉多种网络接口**  
Wireshark 可以捕捉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捉的文件**  
Wireshark 可以打开多种网络分析软件捕捉的包。
- **支持多格式输出**  
Wireshark 可以将捕捉文件输出为多种其他捕捉软件支持的格式。
- **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，可编写一个 Shell 脚本，运行该文件即可启动 Wireshark 配置抓包网卡和信道。Shell 脚本如下：

```
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 脚本打开 Wireshark，会出现 Wireshark 抓包开始界面

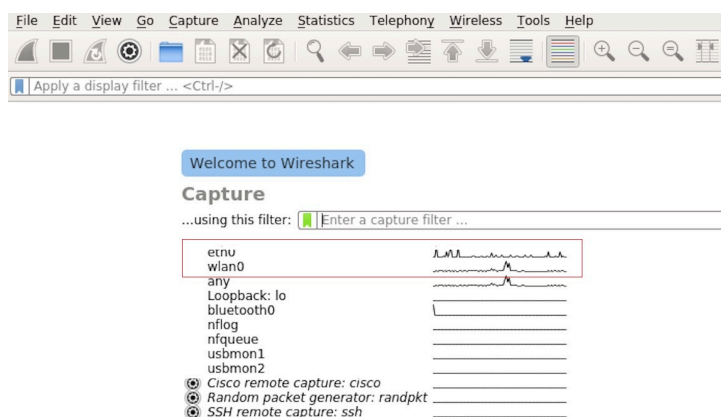


图 33: Wireshark 抓包界面

### 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中包，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。

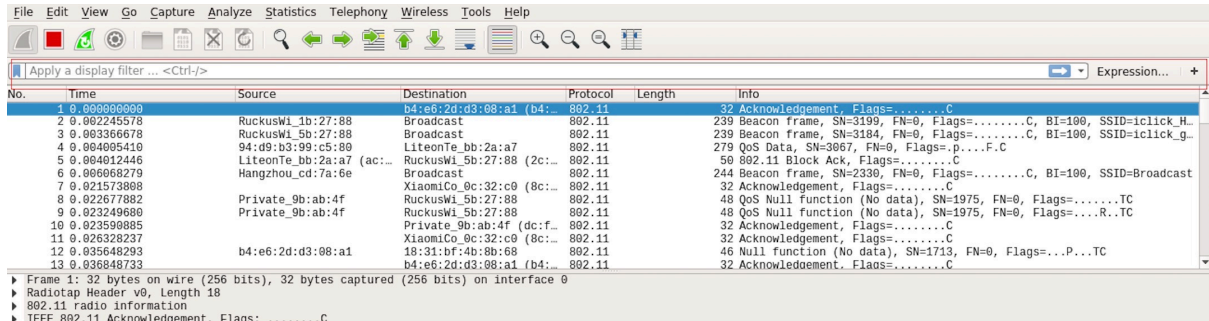


图 34: 设置 Wireshark 过滤条件

点击 *Filter* 按钮（下图的左上角蓝色按钮）会弹出 *display filter* 对话框。

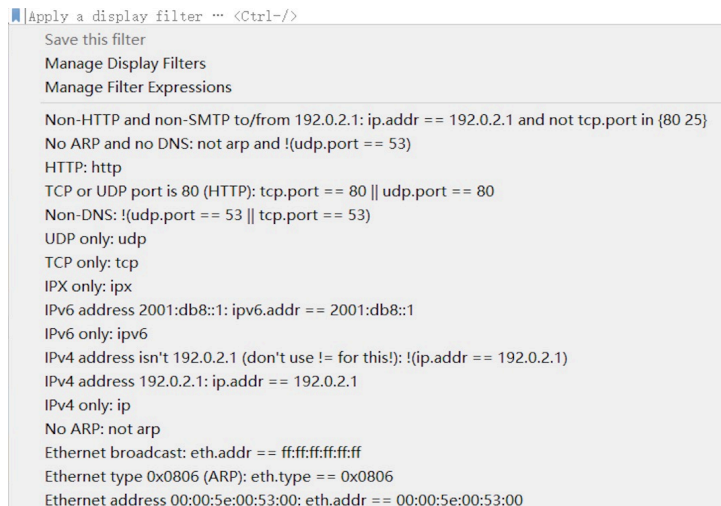


图 35: *Display Filter* 对话框

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。如果您输入的格式不正确，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，背景会变为绿色。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，即使您重新启动程序。

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。

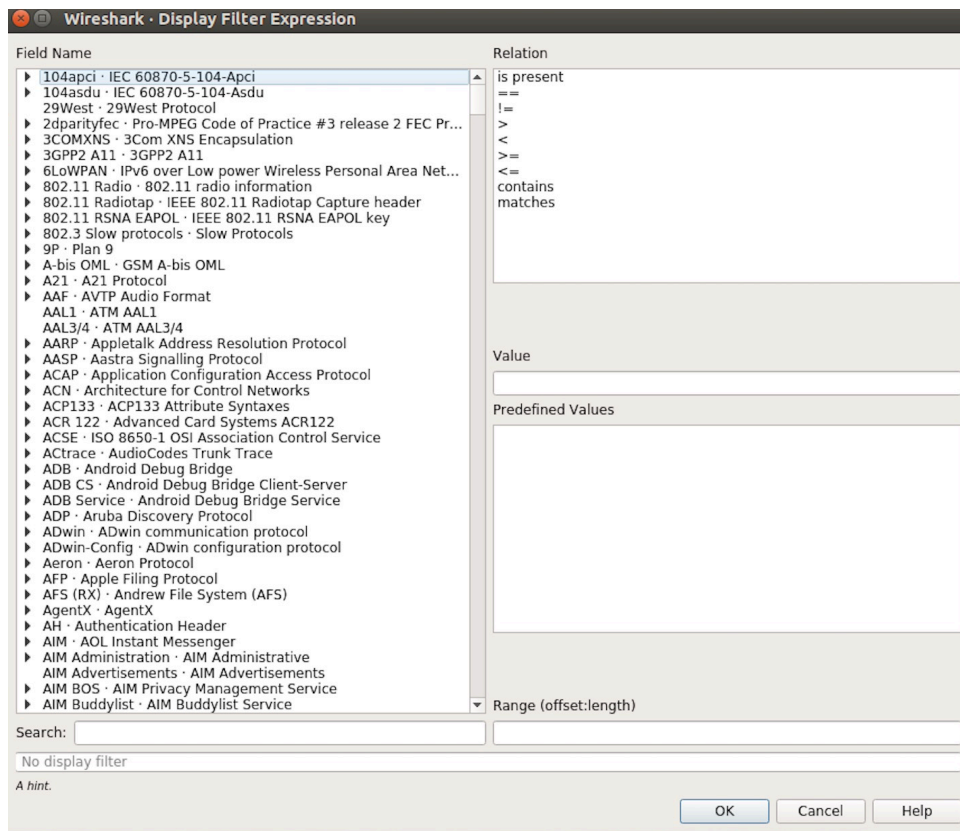


图 36: Filter Expression 对话框



图 37: 过滤条件工具栏

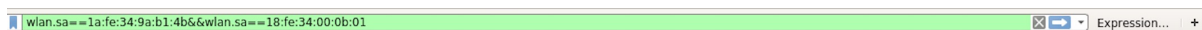


图 38: 在过滤条件工具栏中运用 MAC 地址过滤示例

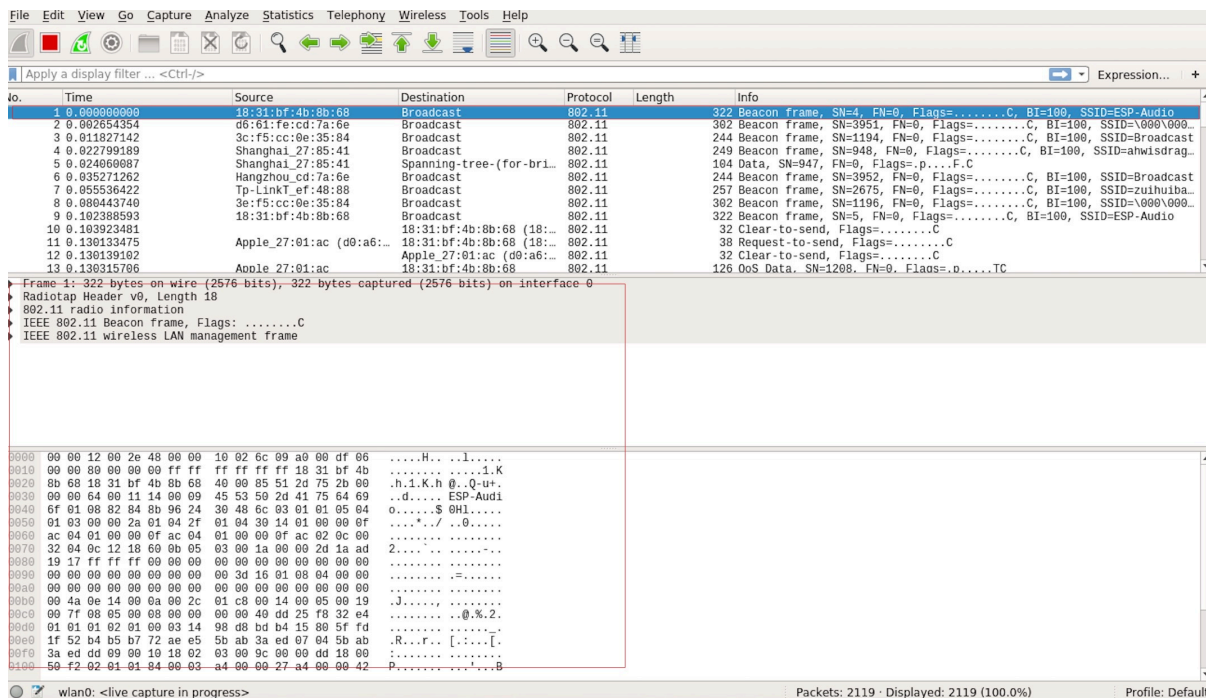


图 39: 封包列表具体信息示例

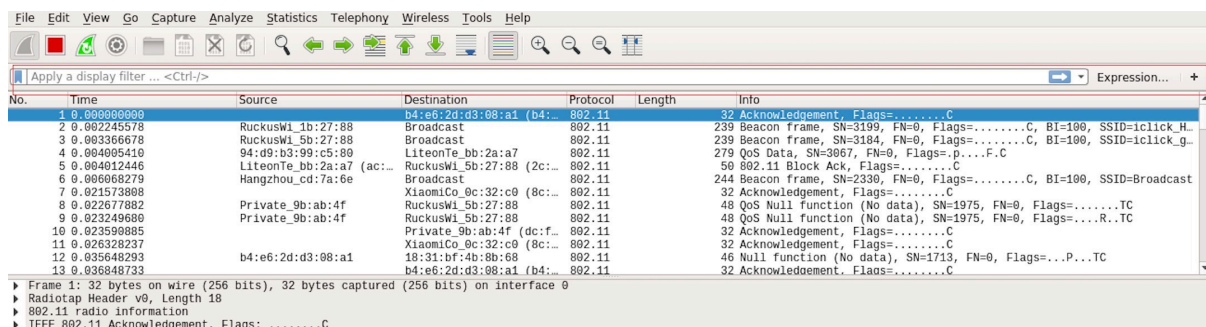


图 40: 停止包的捕捉

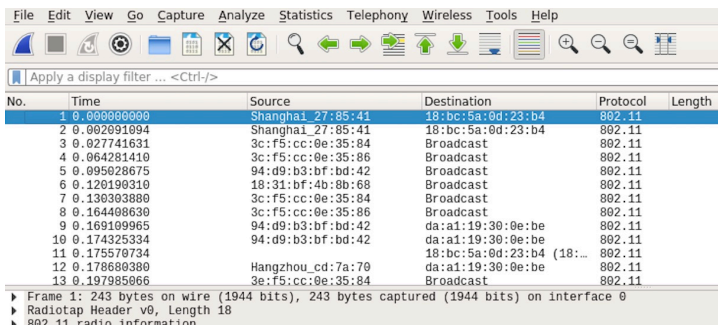


图 41: 开始或继续包的捕捉



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。

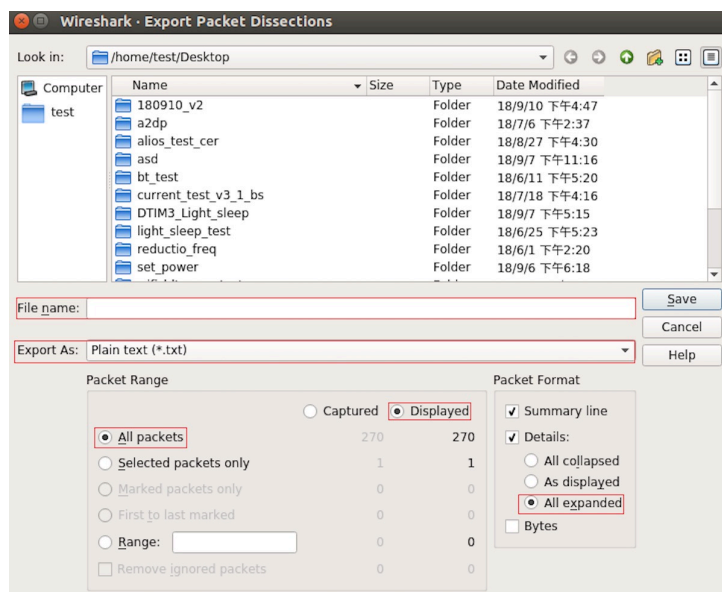


图 42: 保存捕捉包

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。

## 4.25 ESP-BLE-MESH

Bluetooth® mesh networking enables many-to-many (m:m) device communications and is optimized for creating large-scale device networks.

Devices may relay data to other devices not in direct radio range of the originating device. In this way, mesh networks can span very large physical areas and contain large numbers of devices. It is ideally suited for building automation, sensor networks, and other IoT solutions where tens, hundreds, or thousands of devices need to reliably and securely communicate with one another.

Bluetooth mesh is not a wireless communications technology, but a networking technology. This technology is dependent upon Bluetooth Low Energy (BLE) - a wireless communications protocol stack.

Built on top of Zephyr Bluetooth Mesh stack, the ESP-BLE-MESH implementation supports device provisioning and node control. It also supports such node features as Proxy, Relay, Low power and Friend.

Please see the [ESP-BLE-MESH 架构](#) for information about the implementation of ESP-BLE-MESH architecture and [ESP-BLE-MESH API Reference](#) for information about respective API.

ESP-BLE-MESH is implemented and certified based on the latest Mesh Profile v1.0.1, users can refer [here](#) for the certification details of ESP-BLE-MESH.

---

**注解:** If you are looking for Wi-Fi based implementation of mesh for ESP32, please check another product by Espressif called ESP-MESH. For more information and documentation see [ESP-MESH](#).

---

### 4.25.1 Getting Started with ESP-BLE-MESH

This section is intended to help you get started with ESP-BLE-MESH for the hardware based on the ESP32 chip by Espressif.

We are going to demonstrate process of setting and operation of a small ESP-BLE-MESH network of three nodes. This process will cover device provisioning and node configuration, and then sending on/off commands to Generic OnOff Server Models on specific nodes.

If you are new to ESP-IDF, please first set up development environment, compile, flash and run example application following top level ESP-IDF [快速入门](#) documentation.

#### What You Need

Hardware:

- Three ESP32 boards, see [options](#).
- USB cables to connect the boards.
- Computer configured with ESP-IDF.
- Mobile phone or tablet running Android or iOS.

Software:

- Example application [bluetooth/esp\\_ble\\_mesh/ble\\_mesh\\_node/onoff\\_server](#) code to load to the ESP32 boards.
- Mobile App: **nRF Mesh** for Android or iOS. Optionally you can use some other Apps:
  - [EspBleMesh](#) Android App
  - Silicon Labs Android or iOS App

#### Installation Step by Step

This is a detailed roadmap to walk you through the installation process.

**Step 1. Check Hardware** Both [ESP32-DevKitC](#) and [ESP-WROVER-KIT](#) development boards are supported for ESP-BLE-MESH implementation. You can choose particular board through menuconfig: `idf.py menuconfig`  
> Example Configuration > Board selection for ESP-BLE-MESH

---

**注解:** If you plan to use [ESP32-DevKitC](#), connect a RGB LED to GPIO pins 25, 26 and 27.

---

**Step 2. Configure Software** Enter the [bluetooth/esp\\_ble\\_mesh/ble\\_mesh\\_node/onoff\\_server](#) example directory, run `idf.py menuconfig` to select your board and then run `idf.py build` to compile the example.

**Step 3. Upload Application to Nodes** After the [bluetooth/esp\\_ble\\_mesh/ble\\_mesh\\_node/onoff\\_server](#) example is compiled successfully, users can run `idf.py flash` to upload the same generated binary files into each of the three development boards.

Once boards are powered on, the RGB LED on each board should turn **GREEN**.

**Step 4. Provision Nodes** In this section, we will use the **nRF Mesh Android App** to demonstrate how to provision an unprovisioned device. Users can also get its iOS version from the App Store.

**4.1 Scanner** The Scanner is App's functionality to search for unprovisioned devices in range. Open the App, press **Scanner** at the bottom and the search will start. After a short while we should see three unprovisioned devices displayed.

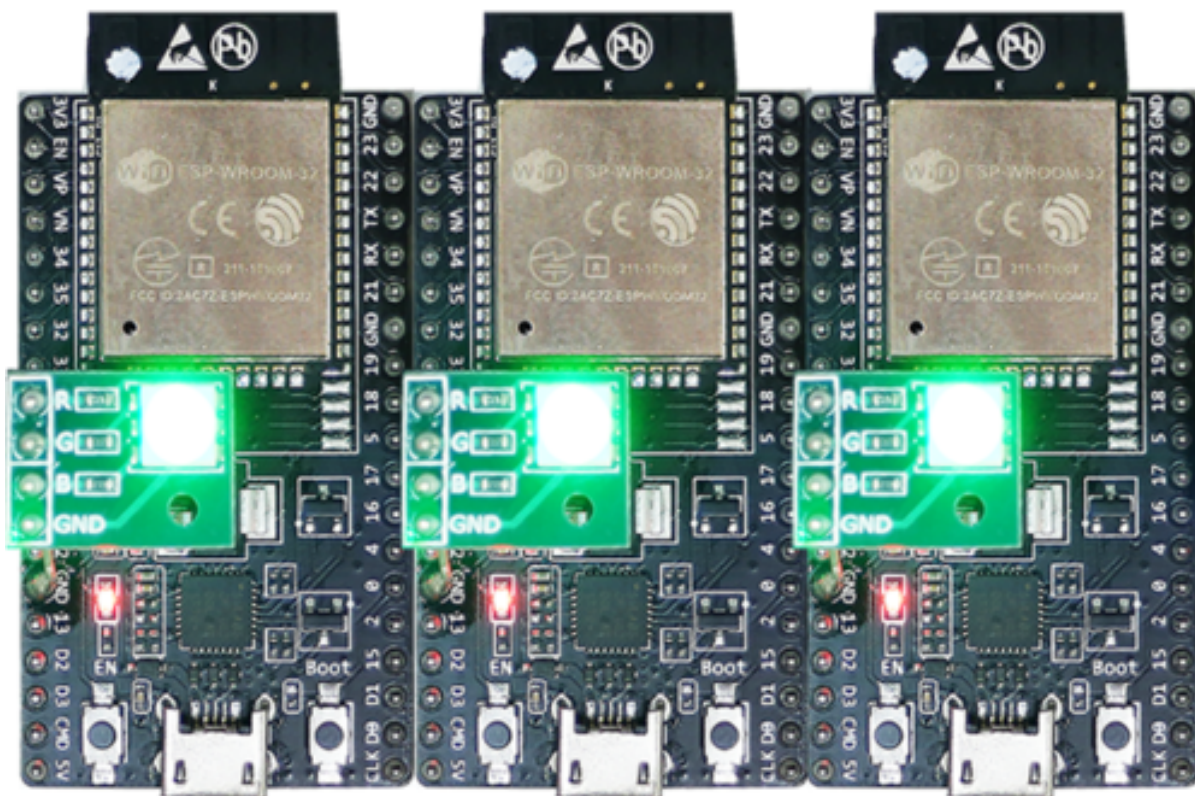


图 43: ESP-BLE-MESH Devices Power On

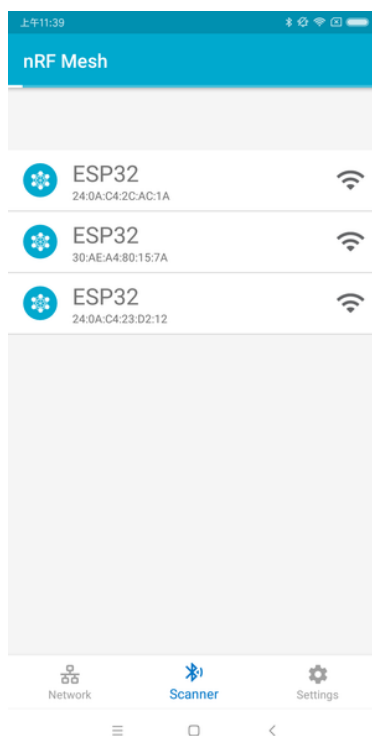


图 44: nRF Mesh - Scanner

**4.2 Identify** Users can select any unprovisioned device, then the App will try to set up a connection with the selected device. After the BLE connection is established successfully (sometimes users need to try multiple times to get connected), and proper ESP-BLE-MESH GATT Service is discovered, users can see the **IDENTIFY** interface button on the screen. The IDENTIFY operation can be used to tell users which device is going to be provisioned.

**注解:** The IDENTIFY operation also needs some cooperation on the device side, then users can see which device is in the provisioning process. Currently when pressing the **IDENTIFY** interface button, no signs can be seen from the device except from the log on the serial monitor.

After the **IDENTIFY** interface button is pressed, users can see the **PROVISION** interface button.

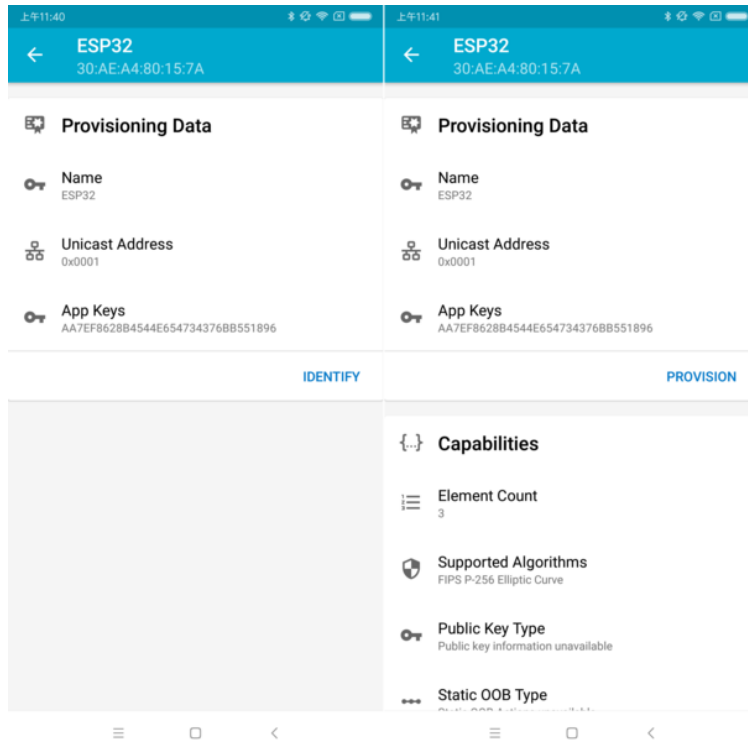


图 45: nRF Mesh - IDENTIFY - PROVISION

**4.3 Provision** Then, the App will try to provision the unprovisioned device. When the device is provisioned successfully, the RGB LED on the board will turn off, and the App will implement the following procedures:

1. Disconnect with the node
2. Try to reconnect with the node
3. Connect successfully and discover ESP-BLE-MESH GATT Service
4. Get Composition Data of the node and add AppKey to it

When all the procedures are finished, the node is configured properly. And after pressing **OK**, users can see that unicast address is assigned, and Composition Data of the node is decoded successfully.

Sometimes in procedure 2, the App may fail to reconnect with the node. In this case, after pressing **OK**, users can see that only unicast address of the node has been assigned, but no Composition Data has been got. Then users need to press **CONNECT** on the top right, and the previously provisioned node will be displayed on the screen, and users need to choose it and try to connect with the node.

After connecting successfully, the App will show the interface buttons which can be used to get Composition Data and add AppKey.

If the device is the second or the third one which has been provisioned by the App, and after pressing **CONNECT**, users can see two or three nodes on the screen. In this situation, users can choose any device to connect with, once succeed then go back to the main screen to choose the node which needs to be configured.

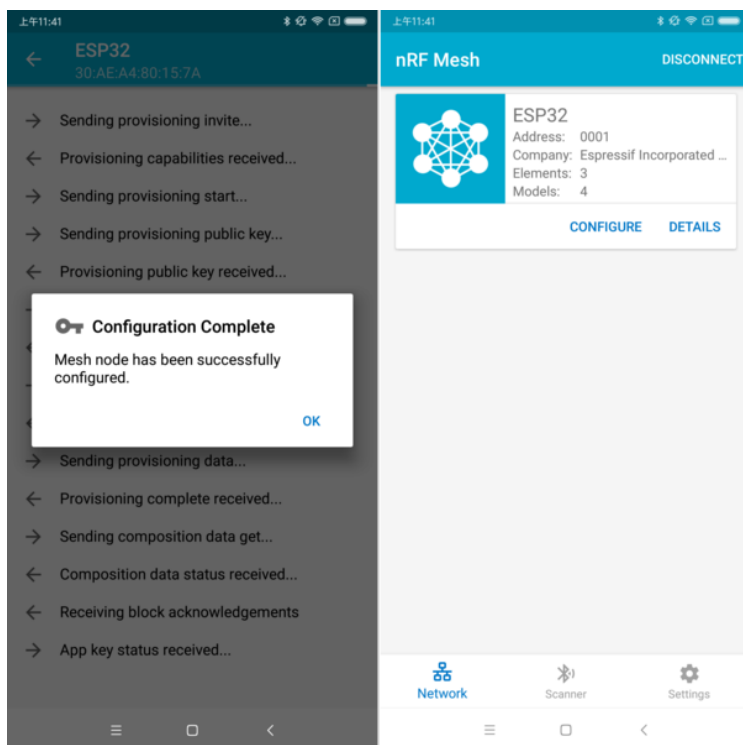


图 46: nRF Mesh - Configuration Complete

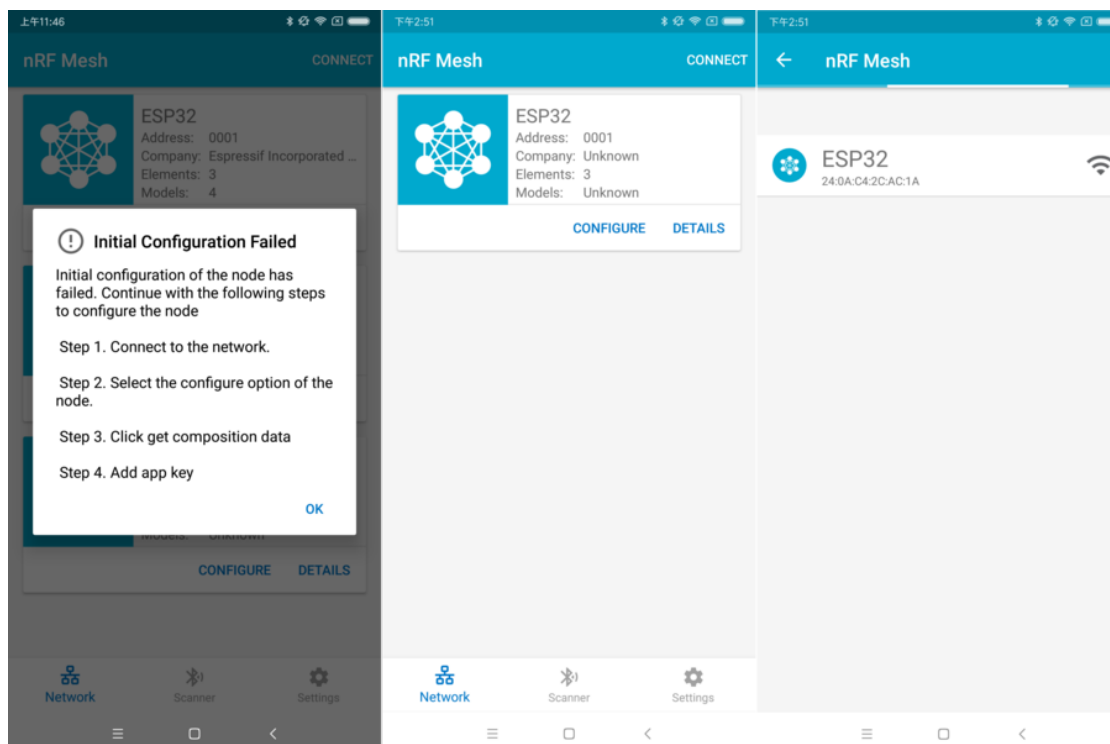


图 47: nRF Mesh - Initial Configuration Failed

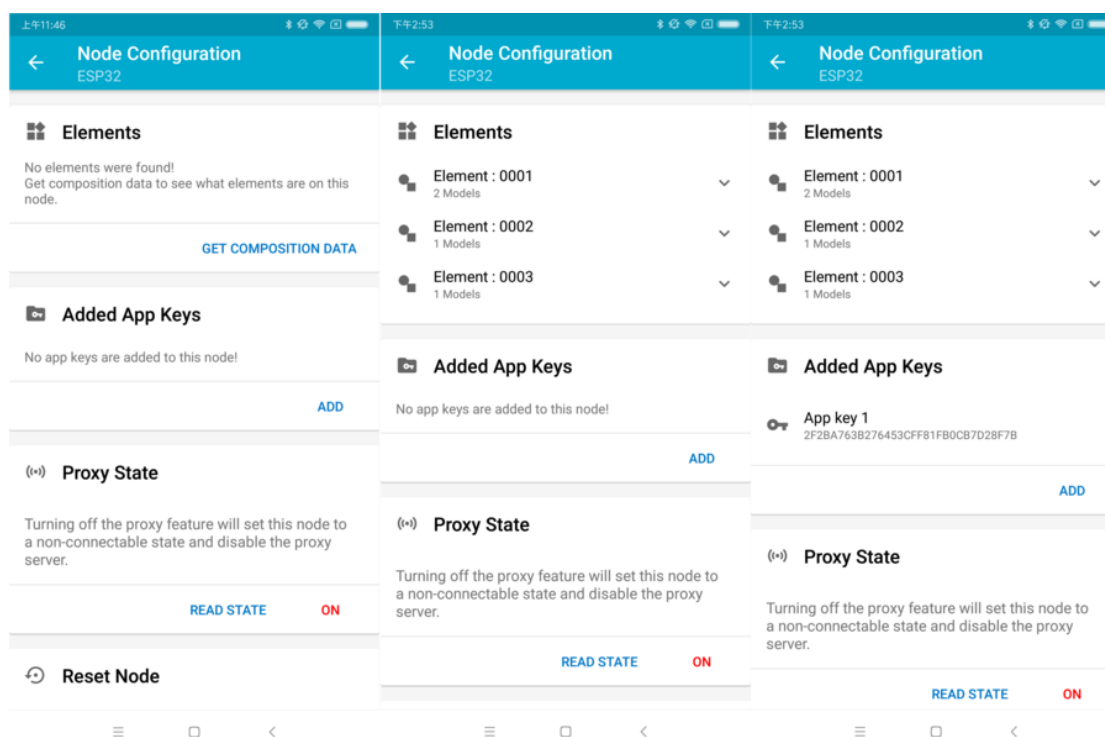


图 48: nRF Mesh - Reconnect - Initial Configuration

Here an example of three devices listed.

- The left picture shows that the third device is provisioned successfully, but the App failed to connect with it. When it tries to reconnect with the third node, three nodes are displayed on the App.
- The right picture shows that after connecting with any node successfully, the App displays the information of the three nodes. Users can see that the App has got the Composition Data of the first and the second nodes, but for the third one, only the unicast address has been assigned to it while the Composition Data is unknown.

**4.4 Configuration** When provisioning and initial configuration are finished, users can start to configure the node, such as binding AppKey with each model with the elements, setting publication information to it, etc.

Example below shows how to bind AppKey with Generic OnOff Server Model within the Primary Element.

---

**注解:** No need to bind AppKey with the Configuration Server Model, since it only uses the DevKey to encrypt messages in the Upper Transport Layer.

---

**Step 5. Operate Network** After all the Generic OnOff Server Models within the three elements are bound with proper AppKey, users can use the App to turn on/off the RGB LED.

In the [bluetooth/esp\\_ble\\_mesh/ble\\_mesh\\_node/onoff\\_server](#) example, the first Generic OnOff Server Model is used to control the **RED** color, the second one is used to control the **GREEN** color and the third one is used to control the **BLUE** color.

The following screenshot shows different board with different color on.

---

**注解:** For **nRF Mesh** iOS App [version 1.0.4], when the node contains more than one element, the App is not behaving correctly. If users try to turn on/off the second or the third Generic OnOff Server Model, the message sent by the App is destined to the first Generic OnOff Server Model within the Primary Element.

---

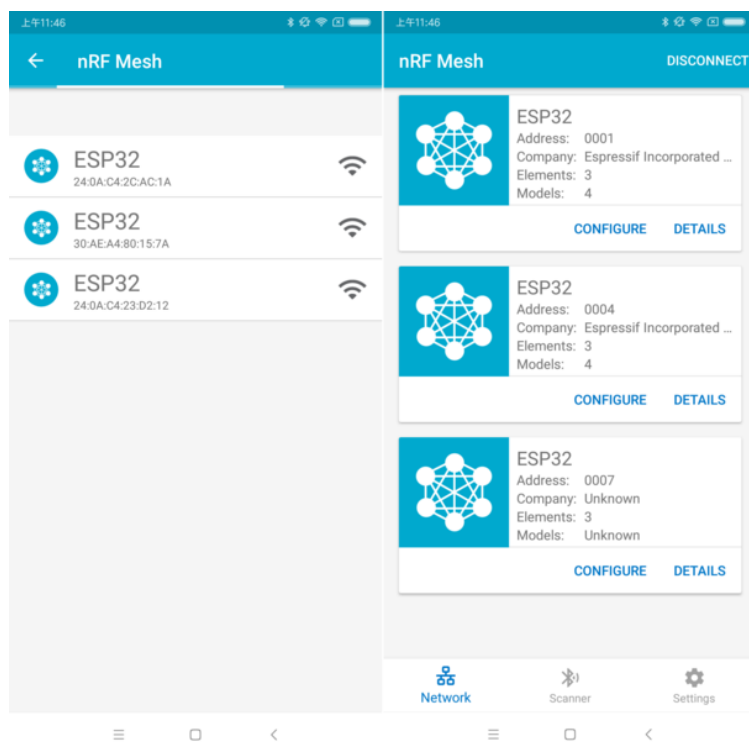


图 49: nRF Mesh - Reconnect - Three Nodes

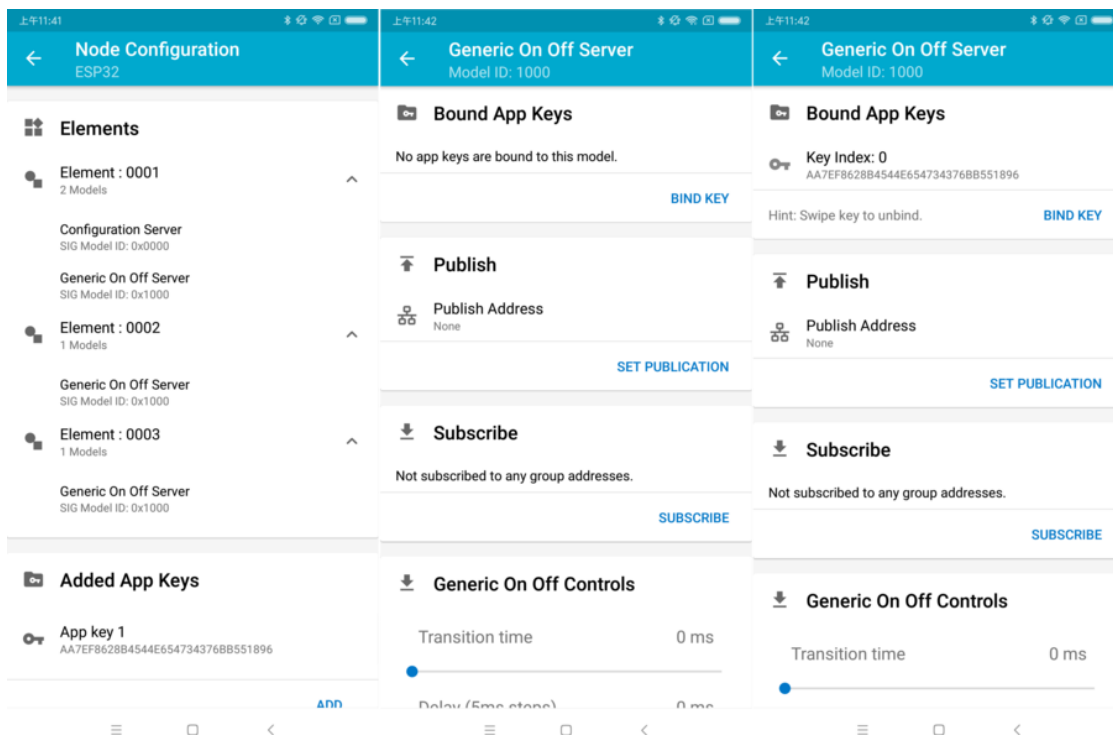


图 50: nRF Mesh - Model Bind AppKey

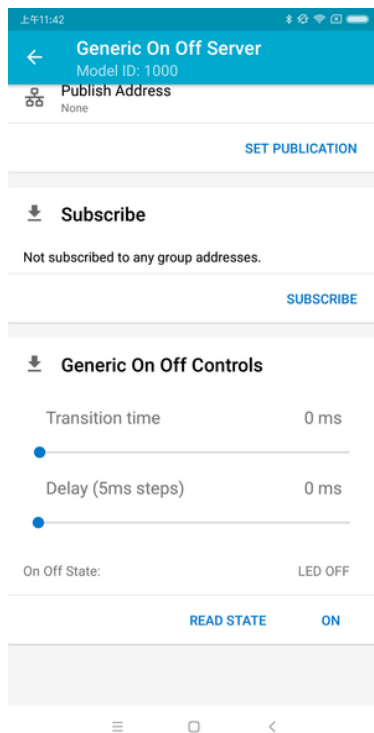


图 51: nRF Mesh - Generic OnOff Control

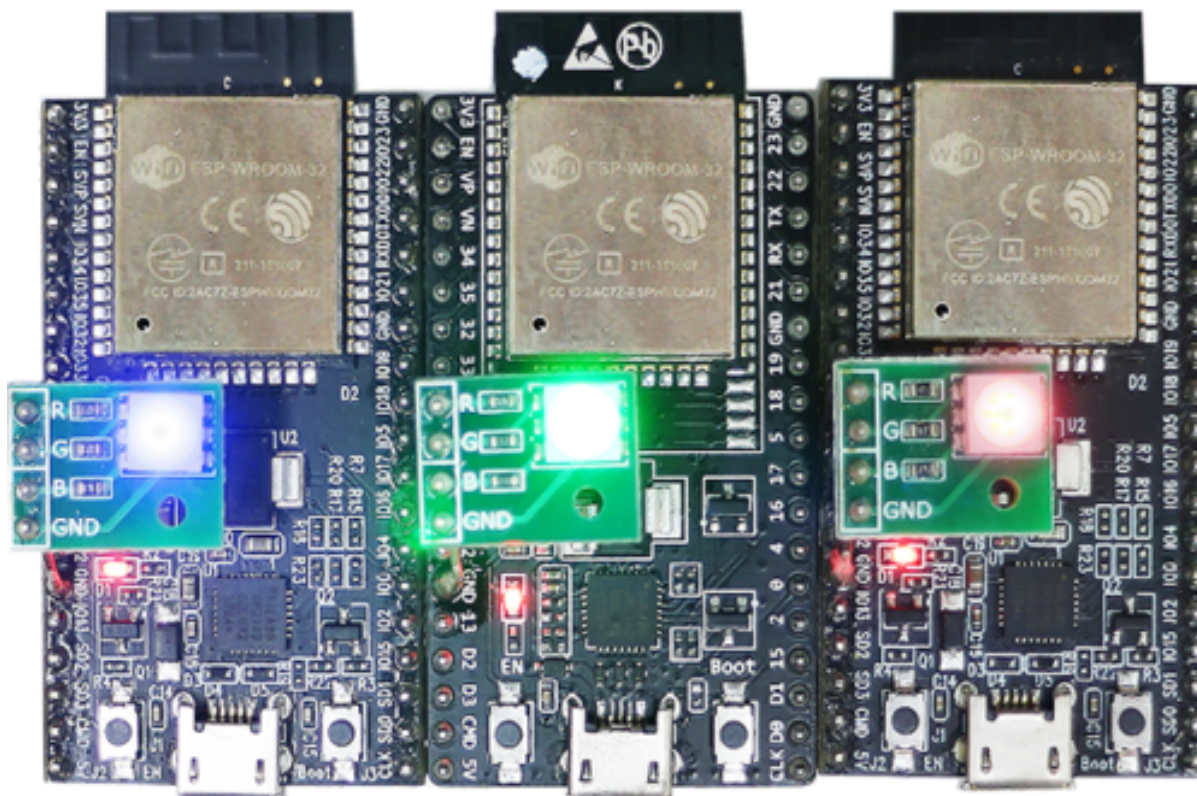


图 52: Three ESP-BLE-MESH Nodes On



## 4.25.2 ESP-BLE-MESH Examples

- [ESP-BLE-MESH Node OnOff Server](#) - shows the use of ESP-BLE-MESH as a node having a Configuration Server model and a Generic OnOff Server model. A ESP-BLE-MESH Provisioner can then provision the unprovisioned device and control a RGB LED representing on/off state, see [example code](#) .
- [ESP-BLE-MESH Node OnOff Client](#) - shows how a Generic OnOff Client model works within a node. The node has a Configuration Server model and a Generic OnOff Client model, see [example code](#) .
- [ESP-BLE-MESH Provisioner](#) - shows how a device can act as an ESP-BLE-MESH Provisioner to provision devices. The Provisioner has a Configuration Server model, a Configuration Client model and a Generic OnOff Client model, see [example code](#) .
- [ESP-BLE-MESH Fast Provisioning - Client](#) and [Server](#) - this example is used for showing how fast provisioning can be used in order to create a mesh network. It takes no more than 60 seconds to provision 100 devices, see [example client code](#) and [example server code](#) .
- [ESP-BLE-MESH and Wi-Fi Coexistence](#) - an example that demonstrates the Wi-Fi and Bluetooth (BLE/BR/EDR) coexistence feature of ESP32. Simply put, users can use the Wi-Fi while operating Bluetooth, see [example code](#) .
- [ESP-BLE-MESH Node Console](#) - an example that implements BLE Mesh node basic features. Within this example a node can be scanned and provisioned by Provisioner and reply to get/set message from Provisioner, see [example node code](#) and [example Provisioner code](#) .

## 4.25.3 ESP-BLE-MESH Demo Videos

- [Provisioning of ESP-BLE-MESH nodes using Smartphone App](#)
- [Espressif Fast Provisioning using ESP-BLE-MESH App](#)
- [Espressif ESP-BLE-MESH and Wi-Fi Coexistence](#)

## 4.25.4 ESP-BLE-MESH FAQ

- [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](#)

## 4.25.5 Related Documents

### 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](#)。

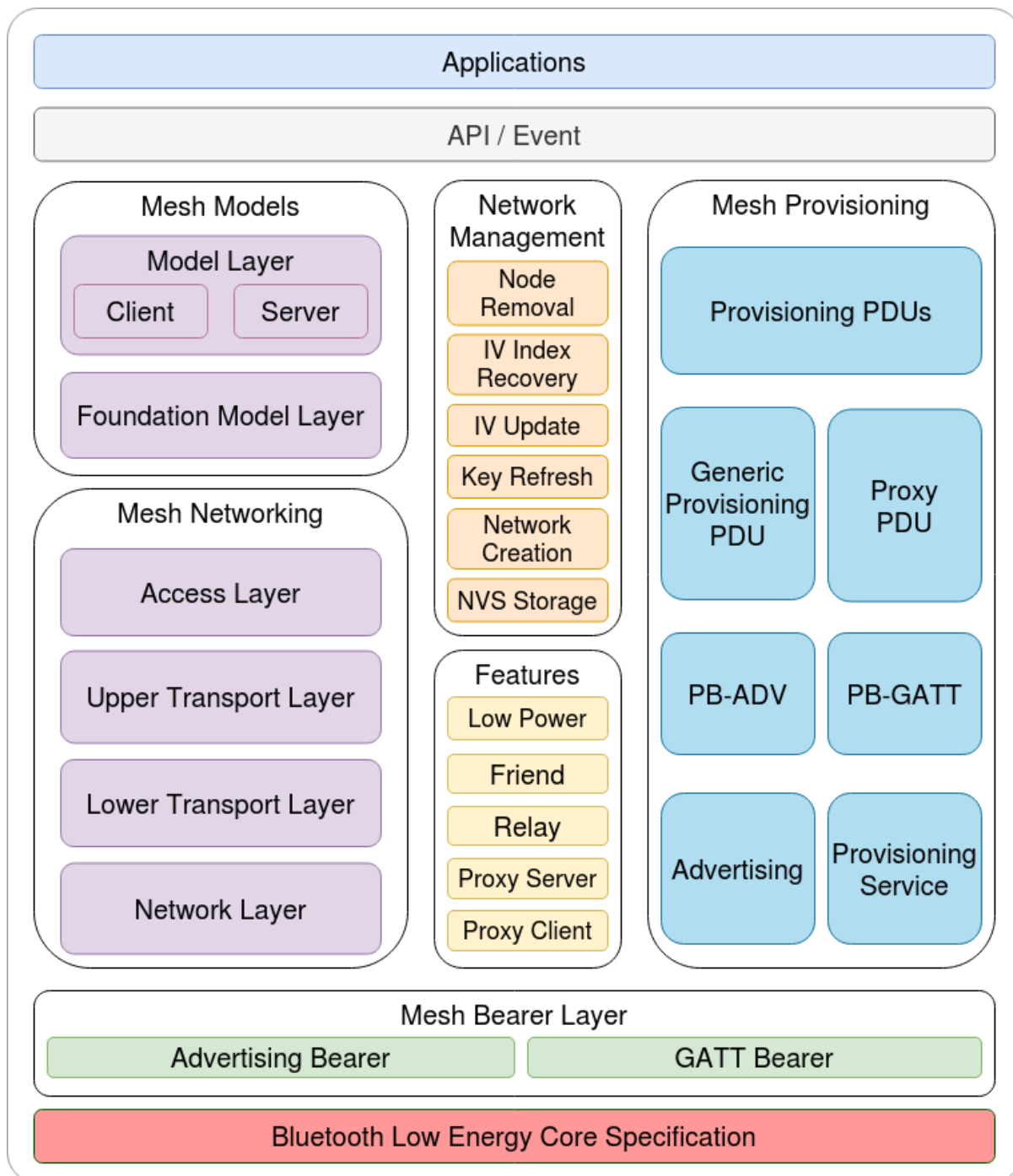


图 53: 图 1.1 ESP-BLE-MESH 架构图

ESP-BLE-MESH 架构主要由以下 5 大部分组成：

- Mesh 协议栈
  - Mesh Networking 负责 BLE Mesh 设备的网络消息处理等。
  - Mesh Provisioning 负责 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 BLE Mesh 特性，如低功耗特性、好友特性、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，如网络密钥 (NetKey)、网络索引等。
- Mesh 网络消息的分包与重组。
- 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：

表 1: 表 1.1 Mesh Networking 框架描述

层级	功能
接入层	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。
上层传输层	上层传输层对接入层进出的应用数据进行加密、解密和认证，同时也处理被称为“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。
底层传输层	底层传输层处理 PDU 的分包和重组。
网络层	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。

**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、网络索引和网络密钥)。
- 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：

表 2: 表 1.2 Mesh Provisioning 框架描述

层级	功能
Provisioning PDUs	通过配网协议处理不同层级的 Provisioning PDUs。
Generic Provisioning PDU/Proxy PDU	使用 Generic Provisioning 层或代理协议层将 Provisioning PDU 传输到未配网的设备。
PB-ADV/PB-GATT	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。
Advertising/Provisioning Service	Provisioning bearer 定义了会话建立的方式, 该方式用来将 Generic Provisioning 层的传输包传送到设备。

**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:

表 3: 表 1.3 Mesh Models 框架描述

层级	功能
模型层	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。
基础模型层	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。

**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。
- 网络索引更新程序: 用于更新节点的网络索引。
- 密钥更新程序: 用于更新节点的网络密钥、应用密钥 (AppKey) 等。
- 网络创建程序: 用于创建 mesh 网络。
- NVS 存储器: 用于存储节点的网络信息。

**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。

表 4: 表 1.3 Mesh 承载层描述

层级	功能
GATT 承载层	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。
广播承载层	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。

**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的功能，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，事件将参数传输给应用层。事件主要分为两大类。

  - **调用 API 完成的事件。**
    - \* 比如接收消息的节点。
  - **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相应处理。

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（网格模型的实现）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。

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：

表 5: 表 2.1 Mesh Networking 文件描述

文件	功能
access.c	BLE Mesh 接入层
transport.c	BLE Mesh 底层/上层传输层
net.c	BLE Mesh 网络层
adv.c	用于发送 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

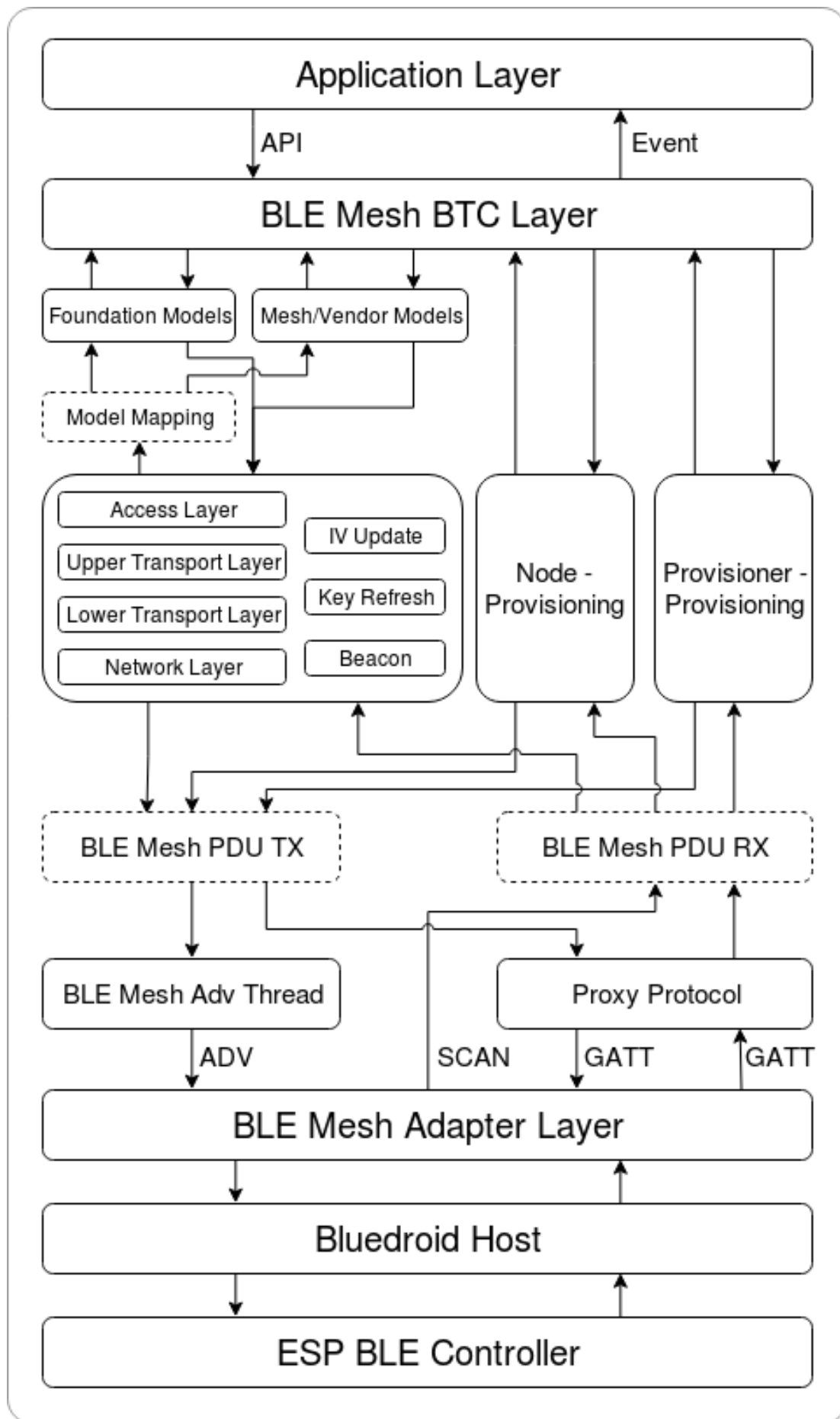


图 54: 图 2.1 ESP-BLE-MESH 架构实现图



**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：

表 6: 表 2.2 Mesh Provisioning (节点) 文件描述

文件	功能
<code>prov.c</code>	BLE Mesh 节点配网 (PB-ADV & PB-GATT)
<code>proxy_server.c</code>	BLE Mesh 节点代理服务器相关功能
<code>beacon.c</code>	用于处理 BLE Mesh Beacon 的 API

实现 Provisioner 配置功能的特定文件如表 2.3 所示：

表 7: 表 2.3 Mesh Provisioning (Provisioner) 文件描述

文件	功能
<code>provisioner_prov.c</code>	BLE Mesh Provisioner 配置入网 (PB-ADV & PB-GATT)
<code>proxy_client.c</code>	BLE Mesh 代理客户端相关功能
<code>provisioner_main.c</code>	BLE Mesh Provisioner 网络相关功能

**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。

表 8: Table 2.4 Mesh Models 文件描述

文件	功能
<code>cfg_cli.c</code>	发送 Configuration Client 消息，接收相应应答消息
<code>cfg_srv.c</code>	接收 Configuration Client 消息，发送适当应答消息
<code>health_cli.c</code>	发送 Health Client 消息，接收相应应答消息
<code>health_srv.c</code>	接收 Health Client 消息，发送适当应答消息
<code>client_common.c</code>	BLE Mesh 模型相关操作
<code>generic_client.c</code>	发送 BLE Mesh Generic Client 消息，接收相应应答消息
<code>lighting_client.c</code>	发送 BLE Mesh Lighting Client 消息，接收相应应答消息
<code>sensor_client.c</code>	发送 BLE Mesh Sensor Client 消息，接收相应应答消息
<code>time_scene_client.c</code>	发送 BLE Mesh Time Scene Client 消息，接收相应应答消息

**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

文件	功能
<code>mesh_bearer_adapt.c</code>	BLE Mesh 承载层适配文件。此文件提供用于接收和发送 BLE Mesh ADV 和 GATT 相关数据包的接口。

**注解：** `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 Examples* 开发产品。

**3. 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

文件	功能
lpn.c	BLE Mesh 低功耗功能
friend.c	BLE Mesh 好友功能
net.c	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
proxy_server.c	BLE Mesh 代理服务器相关功能
proxy_client.c	BLE Mesh 代理客户端相关功能
settings.c	BLE Mesh NVS 存储器功能
main.c	BLE Mesh 协议栈初始化，协议栈使能，节点移除相关功能

## 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。

- 配网阶段：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型。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。
- 需要和节点有相同的、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相互之间不能进行通信。
- 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, 则无法实现相互通信。

### 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值的网络密钥。
- API `esp_ble_mesh_provisioner_add_local_app_key()` 可以用来添加包含固定值或随机值的应用密钥。

### 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的地址。节点在网络配置的过程中可以知道 Provisioner 的单播地址, 因为 Provisioner 往节点发送消息时, 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:

```
#include <stdio.h>
#include <string.h>
#include <stdint.h>

//test date: 0C001A0001000800030000010501000000800100001003103F002A00
//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
// 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;
```

(下页继续)

```

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 * 2
↳4) {
        return -1;
    }

    pos_sig_base = sizeof(*head) - 1;

    for(i = 1; i < head->sig_models * 2; i = i + 2) {
        data->SIG_models[i/2] = mystr[i + pos_sig_base] | (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 = i + 2) {
        data->Vendor_models[i/2].model_id = mystr[i + pos_vnd_base] |
↳(mystr[i + pos_vnd_base + 1] << 8);
        printf("%d: %4.4x\n", i/2, data->Vendor_models[i/2].model_id);

        data->Vendor_models[i/2].vendor_id = mystr[i + pos_vnd_base + 2]
↳| (mystr[i + pos_vnd_base + 3] << 8);
        printf("%d: %4.4x\n", i/2, data->Vendor_models[i/2].vendor_id);
    }

    return 0;
}

void app_main(void)
{
    esp_ble_mesh_composition_head head = {0};
    esp_ble_mesh_composition_decode data = {0};
    uint8_t mystr[] = { 0x0C, 0x00, 0x1A, 0x00,
                        0x01, 0x00, 0x08, 0x00,
                        0x03, 0x00, 0x00, 0x01,
                        0x05, 0x01, 0x00, 0x00,
                        0x00, 0x80, 0x01, 0x00,

```

(下页继续)

(续上页)

```

                                0x00, 0x10, 0x03, 0x10,
                                0x3F, 0x00, 0x2A, 0x00};

int ret;

ret = decode_comp_data(&head, &data, mystr, sizeof(mystr));
if (ret == -1) {
    printf("decode_comp_data error");
}
}

```

### 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。

```

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) {
        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;
        }
    }
}

```

(下页继续)

```

    }
    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;
}

```

**注解：**使能了节点的 NVS 存储器后，通过该方式添加的组地址以及绑定的应用密钥在设备掉电的情况下不能保存。这些配置信息只有通过 Configuration Client Model 配置时才会保存。

### 1.17 Provisioner 如何通过分组的方式控制节点？

通常而言，在 ESP-BLE-MESH 网络中实现组控制有两种方法，即组地址方法和虚拟地址方法。假设有 10 个设备，即 5 个带蓝灯的设备 and 5 个带红灯的设备。

- 方案一：5 个蓝灯设备订阅一个组地址，5 个红灯设备订阅另一个组地址。Provisioner 往不同的组地址发送消息，即可实现分组控制设备。
- 方案二：5 个蓝灯设备订阅一个虚拟地址，5 个红灯设备订阅另一个虚拟地址，Provisioner 往不同的虚拟地址发送消息，即可实现分组控制设备。

### 1.18 Provisioner 如何将节点添加至多个子网？

节点配置期间，Provisioner 可以为节点添加多个网络密钥，拥有相同网络密钥的节点属于同一子网。Provisioner 可以通过不同的网络密钥与不同子网内的节点进行通信。

### 1.19 Provisioner 如何知道网络中的某个设备是否离线？

节点离线通常定义为：电源故障或其他原因导致的节点无法与 mesh 网络中的其他节点正常通信的情况。

ESP-BLE-MESH 网络中的节点间彼此不连接，它们通过广播通道进行通信。

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。

**注解：**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，必须包括相应的客户端模型。

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()` 可用于获取 Configuration Server Model 的 `esp_ble_mesh_cfg_client_set_state_t` 值。
- Health Client Model
  - 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()` 可用于获取 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()` 可用于获取 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()` 可用于获取 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()` 可用于获取 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()` 可用于获取 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。

---

**注解：** 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）。

---

### 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 个八位位组，因此，如果消息的 opcode 为 2 个八位位组，则该消息可以携带 9 个八位位组的有效信息。对于 vendor 消息，由于 opcode 是 3 个八位位组，剩余的有效负载长度为 8 个八位位组。

## 2.9 什么时候应该使能节点的 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 什么时候应该使能节点的 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，一旦烟雾警报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。当烟雾警报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，烟雾警报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` 速率可以稳定在 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...
- `Bluetooth` --> `Bluedriod Enable` --> BT/BLE will first malloc the memory from the PSRAM
- `Bluetooth` --> `Bluedriod Enable` --> Use dynamic memory allocation in BT/BLE stack.
- `Bluetooth` --> `Bluetooth controller` --> BLE full scan feature supported.
- `Wi-Fi` --> Software controls Wi-Fi/Bluetooth coexistence --> Wi-Fi

## 4. 快速配网

### 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 0x000 的应用密钥时，会出现该报错。

### 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

表 11: Table 1 ESP-BLE-MESH Terminology - Role

Term	Official Definition	Detailed Explanation
Unprovisioned Device	A device that is not a member of a mesh network is known as an unprovisioned device.	Examples: lighting devices, temperature control devices, manufacturing equipments and electric doors, etc.
Node	A node is a provisioned device.	The role of unprovisioned device will change to node after being provisioned to ESP-BLE-MESH network. Nodes (such as lighting devices, temperature control devices, manufacturing equipments, and electric doors) are devices that can send, receive, or relay messages in ESP-BLE-MESH network, and they can optionally support one or more subnets.
Relay Node	A node that supports the Relay feature and has the Relay feature enabled is known as a Relay node.	Relay nodes can receive and resend ESP-BLE-MESH messages, so the messages can be transferred further. Users can decide whether or not to enable forwarding function of nodes according to nodes' status. Messages can be relayed for multiple times, and each relay is considered as a "hop". Messages can hop up to 126 times, which is enough for message transmission in a wide area.
Proxy Node	A node that supports the Proxy feature and has the Proxy feature enabled is known as a Proxy node.	Proxy nodes receive messages from one bearer (it generally includes advertising bearer and GATT bearer) and resend it from another one. The purpose is to connect communication equipments that only support GATT bearer to ESP-BLE-MESH network. Generally, mobile apps need a Proxy node to access Mesh network. Without Proxy nodes, mobile apps cannot communicate with members in Mesh network.
Friend 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.	Friend node, like the backup of Low Power node (LPN), can store messages that are sent to Low Power node and security updates; the stored information will be transferred to Low Power node when Low Power node needs it. Low Power node must establish "friendship" with another node that supports the Friend Feature to reduce duty cycle of its receiver, thus power consumption of Low Power node can be reduced. Low Power node needs to find a Friend node to establish a friendship with it. The process involved is called "friendship establishment". Cooperation between Low Power node and Friend nodes enables Low Power node to schedule the use of the radio, thus Low Power node can receive messages at an appropriate or lower frequency without the need of keeping listening. Low Power node will poll Friend node to see if there is new message.
Low Power 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.	By polling, Low Power node gets information from Friend node, such as messages, security updates, and etc.
Provisioner	A node that is capable of adding a device to a mesh network.	The device that can provision unprovisioned devices is called a Provisioner. This process usually needs to be implemented through an app that is typically provided by the product manufacturer and can be used on a gateway, a smartphone, tablet or other carriers.

表 12: Table 2 ESP-BLE-MESH Terminology - Composition

Term	Official Definition	Detailed Explanation
State	A value representing a condition of an element that is exposed by an element of a node.	Each node in a ESP-BLE-MESH network has an independent set of state values that indicate certain states of the device, like brightness, and color of lighting device. Change of state value will lead to change of the physical state of devices. For example, changing the on/off state of a device is actually turning on/off the device.
Model	A model defines the basic functionality of a node.	A node may contain multiple models, and each model defines basic functionalities of nodes, like the states needed by the nodes, the messages controlling the states, and actions resulted from messages handling. The function implementation of the nodes is based on models, which can be divided into SIG Model and Vendor Model, with the former defined by SIG and latter defined by users.
Element	An addressable entity within a device.	A node can contain one or more elements, with each having a unicast address and one or more models, and the models contained by the same element must not be the same.
Composition Data State	The Composition Data state contains information about a node, the elements it includes, and the supported models.	By reading the value of the Composition Data state, users can know basic information of the node, such as the number of elements, and the models in each element. Provisioner gets this message to further provision the device, such as configuring subscription address and publishing address of nodes.

表 13: Table 3 ESP-BLE-MESH Terminology - Features

Term	Official Definition	Detailed Explanation
Low Power Feature	The ability to operate within a mesh network at significantly reduced receiver duty cycles only in conjunction with a node supporting the Friend feature.	Low Power feature reduces power consumption of nodes. When a Low Power node is searching for a Friend node, and there are multiple Friend nodes nearby, it selects the most suitable Friend node through algorithm.
Friend Feature	The ability to help a node supporting the Low Power feature to operate by storing messages destined for those nodes.	By enabling friend feature, the node can help to store information for Low Power node. The nodes enabled with friend feature may cause more power and memory consumption.
Relay Feature	The ability to receive and retransmit mesh messages over the advertising bearer to enable larger networks.	The relay feature enables ESP-BLE-MESH messages to hop among nodes for multiple times, and the transmission distance can exceed the range of direct radio transmission between two nodes, thereby covering the entire network. When a node is enabled with the relay feature to relay messages, it only relays the messages of its own subnet, and does not relay the messages of other subnets. The data integrity will not be considered when the node enabled with relay feature relays segmented messages. The node would relay every segmented message once it receives one rather than waiting for the complete message.
Proxy Feature	The ability to receive and retransmit mesh messages between GATT and advertising bearers.	The purpose of the proxy feature is to allow nodes without an advertising bearer to access the ESP-BLE-MESH network. The proxy feature is typically used in nodes that need to connect to mobile apps.

表 14: Table 4 ESP-BLE-MESH Terminology - Provisioning

Term	Official Definition	Detailed Explanation
PB-ADV	PB-ADV is a provisioning bearer used to provision a device using Generic Provisioning PDUs over the advertising channels.	PB-ADV transfers packets generated during the provisioning process over the advertising channels. This way can only be used for provisioning when provisioner and unprovisioned device both support PB-ADV.
PB-GATT	PB-GATT is a provisioning bearer used to provision a device using Proxy PDUs to encapsulate Provisioning PDUs within the Mesh Provisioning Service.	PB-GATT uses connection channels to transfer packets generated during the provisioning process. If an unprovisioned device wants to be provisioned through this method, it needs to implement the related Mesh Provisioning Service. Unprovisioned devices which don't implement such service cannot be provisioned into mesh network through PB-GATT bearer.
Provisioning	Provisioning is a process of adding an unprovisioned device to a mesh network, managed by a Provisioner.	The process of provisioning turns the “unprovisioned device” into a “node”, making it a member of the ESP-BLE-MESH network.
Authentication Method	Authentication is a step during the provisioning of nodes.	There are four authentication methods for unprovisioned devices: Output OOB, Input OOB, Static OOB, and No OOB.
Input OOB	Input Out-of-Band	For example, a Provisioner generates and displays a random number, and then prompts users to take appropriate actions to input the random number into the unprovisioned device. Taking lighting switch as an example, users can press the button for several times in a certain period of time to input the random number displayed on the Provisioner. Authentication method of the Input OOB is similar to that of Output OOB, but the role of the device is reversed.
Output OOB	Output Out-of-Band	For example, an unprovisioned device will choose a random number and output the number in a way that is compatible with its functionality. If the unprovisioned device is a bulb, it can flash a specified number of times. If the unprovisioned device has an LCD screen, the random number can display as a multi-digit value. Users who start provisioning should input the observed number to authenticate the unprovisioned device.
Static OOB	Static Out-of-Band	Authentication method of Static OOB: use Static OOB information. Use 0 as Static OOB information if No OOB information is needed. Use Static OOB information to authenticate devices which are going through provisioning if OOB information is needed.
No OOB	No Out-of-Band	Authentication method of No OOB: Set the value of the Static OOB field to 0. Using this way is like not authenticating the unprovisioned devices.



表 15: Table 5 ESP-BLE-MESH Terminology - Address

Term	Official Definition	Detailed Explanation
Unassigned Address	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.	The addresses owned by elements which has not been configured yet or no address has been allocated are unassigned addresses. These elements will not be used for messages transfer because they have no fixed address. Unassigned address is recommended to set as the value of the address before setting the address of user code.
Unicast Address	A unicast address is a unique address allocated to each element.	During provisioning, the Provisioner will assign a unicast address to each element of node within the life cycle of the nodes in the network. A unicast address may appear in the source/destination address field of a message. Messages sent to a unicast address can only be processed by the element that owns the unicast address.
Virtual Address	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.	Associated with specific UUID labels, a virtual address may serve as the publishing or subscription address of the model. A UUID label is a 128-bit value associated with elements of one or more nodes. For virtual addresses, the 15th and 14th bits are set to 1 and 0 respectively; bits from 13th to 0 are set to hash values (providing 16384 hash values). The hash is a derivation of the Label UUID. To use subscribing elements to check the full 128-bit UUID is very inefficient while hash values provide a more efficient way to determine which elements that which messages are finally sent to.
Group Address	A group address is an address that is programmed into zero or more elements	Group address is another kind of multicast address in the ESP-BLE-MESH network, which is usually used to group nodes. A message sent to the all-proxies address shall be processed by the primary element of all nodes that have the proxy functionality enabled. A message sent to the all-friends address shall be processed by the primary element of all nodes that have the friend functionality enabled. A message sent to the all-relays address shall be processed by the primary element of all nodes that have the relay functionality enabled. A message sent to the all-nodes address shall be processed by the primary element of all nodes.

表 16: Table 6 ESP-BLE-MESH Terminology - Security

Term	Official Definition	Detailed Explanation
Device Key (DevKey)	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.	The device key enables you to provision the devices, configure the nodes. The device key is used to encrypt Configuration Messages, i.e. the message transferred between the Provisioner and the node when the device is configured.
Application Key (App-Key)	Application keys are used to secure communications at the upper transport layer.	Application key is used for decryption of application data before delivering application data to application layer and encryption of them during the delivery of application layer. Some nodes in the network have a specific purpose and can restrict access to potentially sensitive data based on the needs of the application. With specific application keys, these nodes are associated with specific applications. Generally speaking, the fields using different application keys include security (access control of buildings, machine rooms and CEO offices), lighting (plant, exterior building and sidewalks) and HVAC systems. Application keys are bound to Network keys. This means application keys are only used in a context of a Network key they are bound to. An application key shall only be bound to a single Network key.
Master Security Material	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.	The corresponding friendship messages encrypted with the friendship security material: 1. Friend Poll, 2. Friend Update, 3. Friend Subscription List, add/delete/confirm, 4. The Stored Messages” sent by friend nodes to Low Power node. The corresponding friendship messages encrypted with the master security material: 1. Friend Clear, 2. Friend Clear Confirm. Based on the setup of the applications, the messages sent from the Low Power node to the friend nodes will be encrypted with the friendship security material or master security material, with the former being used by the messages transmitted between Low Power node and friend nodes and the latter being used by other network messages.

表 17: Table 7 ESP-BLE-MESH Terminology - Message

Term	Official Definition	Detailed Explanation
Reassembly / Segmentation	Segmentation and reassembly (SAR) is a method of communication network, which is divided into small units before transmitting packets and reassembled in a proper order at the communication receiving end.	The lower transport layer will automatically segment the message whose size is too big. The receiving end will return a response message, and the transmitting end will send the data packet again that the receiving end does not receive according to the response message. This is automatically completed by the lower transport layer. Unsegmented messages have at most 15 bytes, of which 4 bytes are transMIC, so the remaining is 11 bytes; in the case of segmentation, there are 12 valid bytes in the first several packets, and 8 in the last one. Special case: A shorter packet requires mandatory segmentation from lower transport layer, in which case the valid byte is 8 bytes.
Unacknowledged / Acknowledged	There are two types of messages: Unacknowledged or Acknowledged	Based on the whether or not the receiving end needs to send the response message, the messages sent are divided into two kinds. The sending end should set the maximum number of retransmission.

表 18: Table 8 ESP-BLE-MESH Terminology - Foundation Models

Term	Official Definition	Detailed Explanation
Configuration Server Model	This model is used to represent a mesh network configuration of a device.	The node must contain the Configuration Server Model, which is responsible for maintaining configuration-related states. The states that Configuration Server Model maintains include: NetKey List, AppKey List, Model to AppKey List, Node Identity, Key Refresh Phase, Heartbeat Publish, Heartbeat Subscription, Network Transmit, Relay Retransmit etc.
Configuration Client Model	The model is used to represent an element that can control and monitor the configuration of a node.	The Configuration Client Model uses messages to control the state maintained by the Configuration Server Model. The Provisioner must contain the Configuration Client Model, with which the configuration messages, like Configuration Composition Data Get can be sent.
Health Server Model	This model is used to represent a mesh network diagnostics of a device.	The Health Server Model is primarily used by devices to check their states and see if there is an error. The states maintained by Health Server model include: Current Fault, Registered Fault, Health Period, and Attention Timer.
Health Client Model	The model is used to represent an element that can control and monitor the health of a node.	The Health Client Model uses messages to control the state maintained by the Health Server Model. The model can get the self-test information of other nodes through the message “Health Fault Get” .

表 19: Table 9 ESP-BLE-MESH Terminology - Network Management

Term	Official Definition	Detailed Explanation
Key Refresh procedure	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.	Key Refresh Procedure is used to update network key and application key of ESP-BLE-MESH network. Key Refresh Procedure is used when the security of one or more network keys and/or one or more application keys is threatened or potentially threatened. Keys are usually updated after some nodes in the network are removed.
IV (Initialisation Vector) Update Procedure	A node can also use an IV Update procedure to signal to peer nodes that it is updating the IV Index.	The IV Update procedure is used to update the value of ESP-BLE-MESH network’s IV Index. This value is related to the random number required for message encryption. To ensure that the value of the random number is not repeated, this value is periodically incremented. IV Index is a 32-bit value and a shared network resource. For example, all nodes in a mesh network share the same IV Index value. Starting from 0x00000000, the IV Index increments during the IV Update procedure and maintained by a specific process, ensuring the IV Index shared in the mesh network is the same. This can be done when the node believes that it has the risk of exhausting its sequence number, or when it determines that another node is nearly exhausting its sequence number. Note: The update time must not be less than 96 hours. It can be triggered when a secure network beacon is received, or when the node determines that its sequence number is greater than a certain value.

For more terms, please see: [ESP-BLE-MESH Glossary of Terms](#).

### Bluetooth SIG Documentation

- [BLE Mesh Core Specification](#)
- [BLE Mesh Model Specification](#)
- [An Intro to Bluetooth Mesh Part 1 / Part 2](#)

- [The Fundamental Concepts of Bluetooth Mesh Networking, Part 1 / Part 2](#)
- [Bluetooth Mesh Networking: Friendship](#)
- [Management of Devices in a Bluetooth Mesh Network](#)
- [Bluetooth Mesh Security Overview](#)
- [Provisioning a Bluetooth Mesh Network Part 1 / Part 2](#)

## 4.26 ESP-MESH

This guide provides information regarding the ESP-MESH protocol. Please see the [MESH API Reference](#) for more information about API usage.

### 4.26.1 Overview

ESP-MESH is a networking protocol built atop the Wi-Fi protocol. ESP-MESH allows numerous devices (henceforth referred to as nodes) spread over a large physical area (both indoors and outdoors) to be interconnected under a single WLAN (Wireless Local-Area Network). ESP-MESH is self-organizing and self-healing meaning the network can be built and maintained autonomously.

The ESP-MESH guide is split into the following sections:

1. [Introduction](#)
2. [ESP-MESH Concepts](#)
3. [Building a Network](#)
4. [Managing a Network](#)
5. [Data Transmission](#)
6. [Channel Switching](#)
7. [Performance](#)
8. [Further Notes](#)

### 4.26.2 Introduction

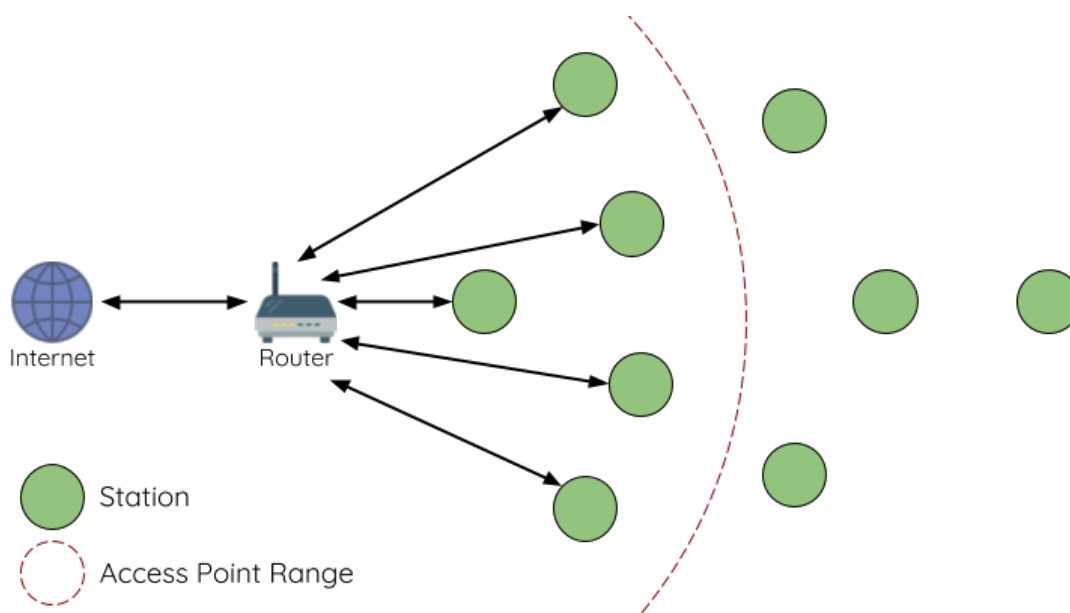


图 55: Traditional Wi-Fi Network Architectures

A traditional infrastructure Wi-Fi network is a point-to-multipoint network where a single central node known as the access point (AP) is directly connected to all other nodes known as stations. The AP is responsible for arbitrating and

forwarding transmissions between the stations. Some APs also relay transmissions to/from an external IP network via a router. Traditional infrastructure Wi-Fi networks suffer the disadvantage of limited coverage area due to the requirement that every station must be in range to directly connect with the AP. Furthermore, traditional Wi-Fi networks are susceptible to overloading as the maximum number of stations permitted in the network is limited by the capacity of the AP.

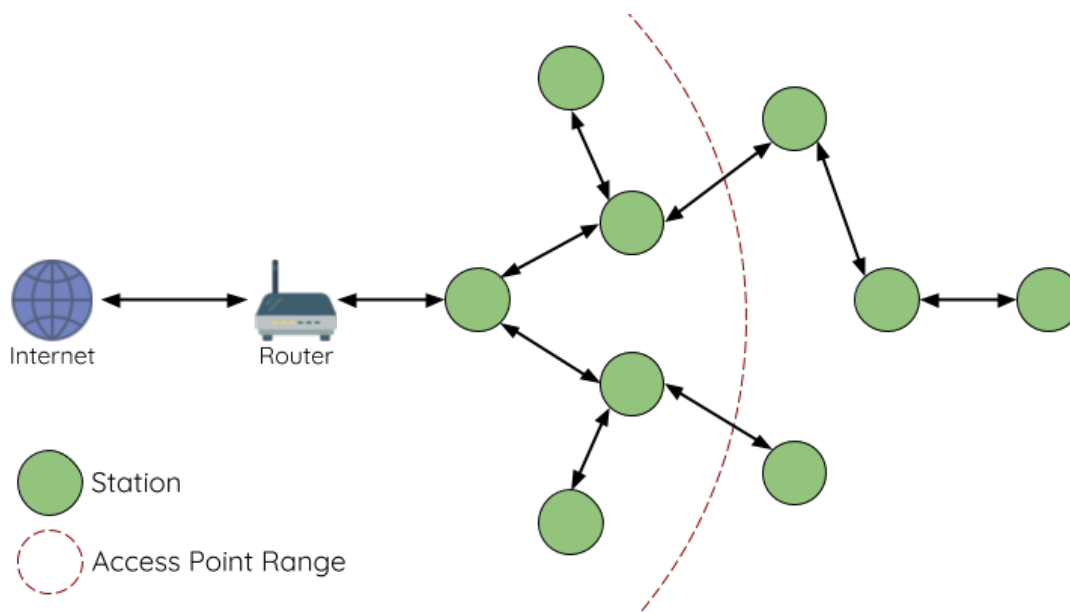


图 56: ESP-MESH Network Architecture

ESP-MESH differs from traditional infrastructure Wi-Fi networks in that nodes are not required to connect to a central node. Instead, nodes are permitted to connect with neighboring nodes. Nodes are mutually responsible for relaying each others transmissions. This allows an ESP-MESH network to have much greater coverage area as nodes can still achieve interconnectivity without needing to be in range of the central node. Likewise, ESP-MESH is also less susceptible to overloading as the number of nodes permitted on the network is no longer limited by a single central node.

### 4.26.3 ESP-MESH Concepts

## Terminology

Term	Description
Node	Any device that <b>is</b> or <b>can be</b> part of an ESP-MESH network
Root Node	The top node in the network
Child Node	A node X is a child node when it is connected to another node Y where the connection makes node X more distant from the root node than node Y (in terms of number of connections).
Parent Node	The converse notion of a child node
Sub-Child Node	Any node reachable by repeated proceeding from parent to child
Sibling Nodes	Nodes that share the same parent node
Connection	A traditional Wi-Fi association between an AP and a station. A node in ESP-MESH will use its station interface to associate with the softAP interface of another node, thus forming a connection. The connection process includes the authentication and association processes in Wi-Fi.
Up-stream Connection	The connection from a node to its parent node
Down-stream Connection	The connection from a node to one of its child nodes
Wireless Hop	The portion of the path between source and destination nodes that corresponds to a single wireless connection. A data packet that traverses a single connection is known as <b>single-hop</b> whereas traversing multiple connections is known as <b>multi-hop</b> .
Subnetwork	A subnetwork is subdivision of an ESP-MESH network which consists of a node and all of its descendant nodes. Therefore the subnetwork of the root node consists of all nodes in an ESP-MESH network.
MAC Address	Media Access Control Address used to uniquely identify each node or router within an ESP-MESH network.
DS	Distribution System (External IP Network)

## Tree Topology

ESP-MESH is built atop the infrastructure Wi-Fi protocol and can be thought of as a networking protocol that combines many individual Wi-Fi networks into a single WLAN. In Wi-Fi, stations are limited to a single connection with an AP (upstream connection) at any time, whilst an AP can be simultaneously connected to multiple stations (downstream connections). However ESP-MESH allows nodes to simultaneously act as a station and an AP. Therefore a node in ESP-MESH can have **multiple downstream connections using its softAP interface**, whilst simultaneously having a **single upstream connection using its station interface**. This naturally results in a tree network topology with a parent-child hierarchy consisting of multiple layers.

ESP-MESH is a multiple hop (multi-hop) network meaning nodes can transmit packets to other nodes in the network through one or more wireless hops. Therefore, nodes in ESP-MESH not only transmit their own packets, but simultaneously serve as relays for other nodes. Provided that a path exists between any two nodes on the physical layer (via one or more wireless hops), any pair of nodes within an ESP-MESH network can communicate.

---

**注解:** The size (total number of nodes) in an ESP-MESH network is dependent on the maximum number of layers permitted in the network, and the maximum number of downstream connections each node can have. Both of these variables can be configured to limit the size of the network.

---

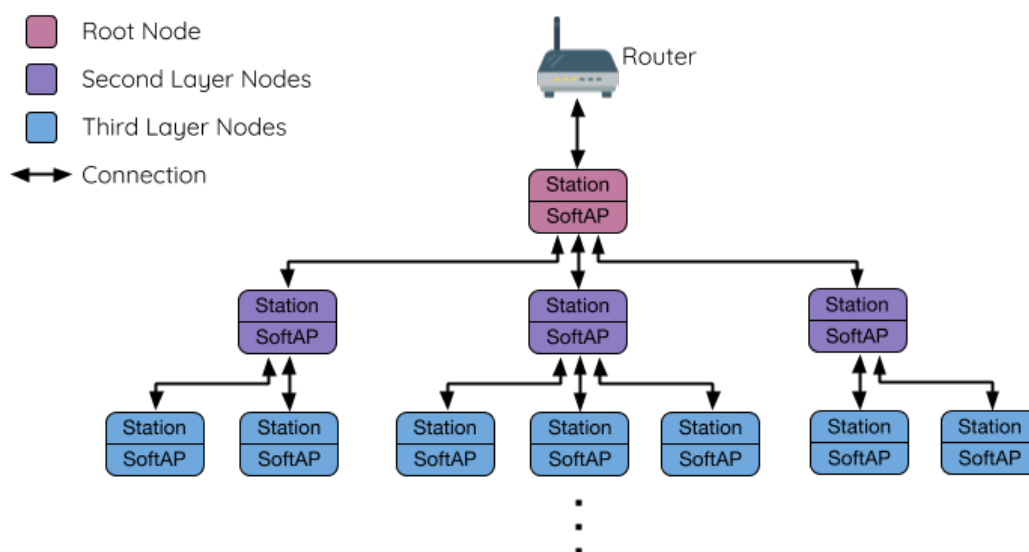


图 57: ESP-MESH Tree Topology

### Node Types

**Root Node:** The root node is the top node in the network and serves as the only interface between the ESP-MESH network and an external IP network. The root node is connected to a conventional Wi-Fi router and relays packets to/from the external IP network to nodes within the ESP-MESH network. **There can only be one root node within an ESP-MESH network** and the root node's upstream connection may only be with the router. Referring to the diagram above, node A is the root node of the network.

**Leaf Nodes:** A leaf node is a node that is not permitted to have any child nodes (no downstream connections). Therefore a leaf node can only transmit or receive its own packets, but cannot forward the packets of other nodes. If a node is situated on the network's maximum permitted layer, it will be assigned as a leaf node. This prevents the node from forming any downstream connections thus ensuring the network does not add an extra layer. Some nodes without a softAP interface (station only) will also be assigned as leaf nodes due to the requirement of a softAP interface for any downstream connections. Referring to the diagram above, nodes L/M/N are situated on the networks maximum permitted layer hence have been assigned as leaf nodes .

**Intermediate Parent Nodes:** Connected nodes that are neither the root node or a leaf node are intermediate parent nodes. An intermediate parent node must have a single upstream connection (a single parent node), but can have zero to multiple downstream connections (zero to multiple child nodes). Therefore an intermediate parent node can transmit and receive packets, but also forward packets sent from its upstream and downstream connections. Referring to the diagram above, nodes B to J are intermediate parent nodes. **Intermediate parent nodes without downstream connections such as nodes E/F/G/I/J are not equivalent to leaf nodes** as they are still permitted to form downstream connections in the future.

**Idle Nodes:** Nodes that have yet to join the network are assigned as idle nodes. Idle nodes will attempt to form an upstream connection with an intermediate parent node or attempt to become the root node under the correct circumstances (see [Automatic Root Node Selection](#)). Referring to the diagram above, nodes K and O are idle nodes.

### Beacon Frames & RSSI Thresholding

Every node in ESP-MESH that is able to form downstream connections (i.e. has a softAP interface) will periodically transmit Wi-Fi beacon frames. A node uses beacon frames to allow other nodes to detect its presence and know of its status. Idle nodes will listen for beacon frames to generate a list of potential parent nodes, one of which the idle node will form an upstream connection with. ESP-MESH uses the Vendor Information Element to store metadata such as:

- Node Type (Root, Intermediate Parent, Leaf, Idle)

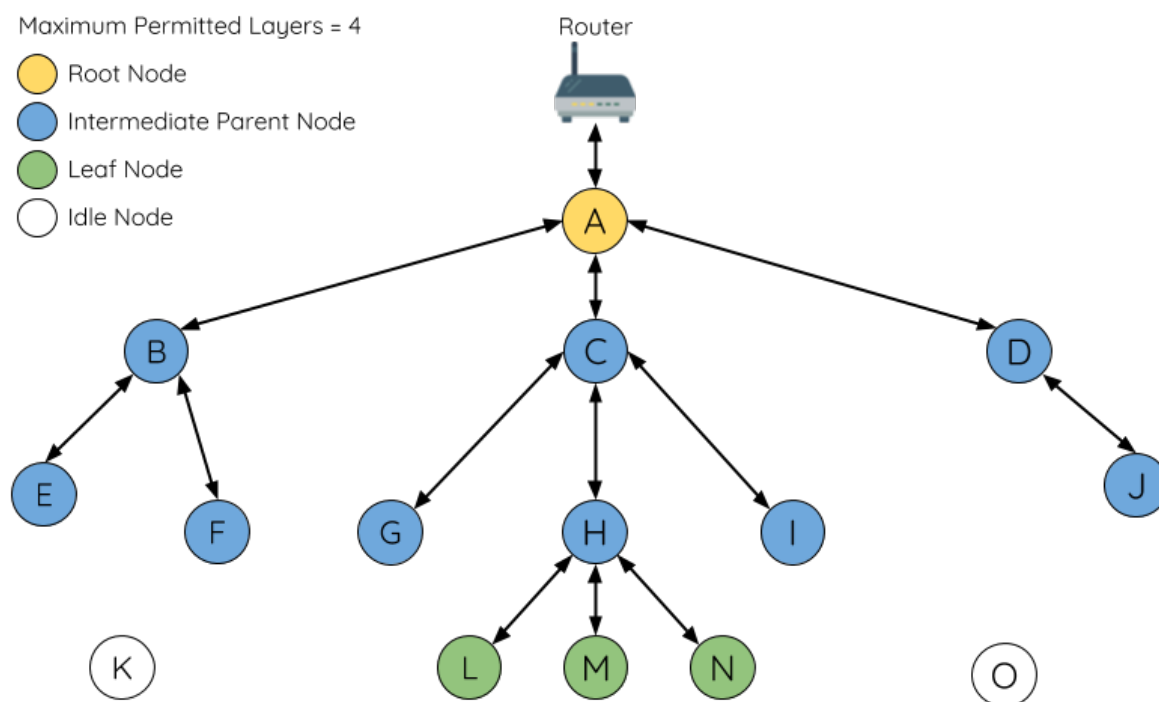


图 58: ESP-MESH Node Types

- Current layer of Node
- Maximum number of layers permitted in the network
- Current number of child nodes
- Maximum number of downstream connections to accept

The signal strength of a potential upstream connection is represented by RSSI (Received Signal Strength Indication) of the beacon frames of the potential parent node. To prevent nodes from forming a weak upstream connection, ESP-MESH implements an RSSI threshold mechanism for beacon frames. If a node detects a beacon frame with an RSSI below a preconfigured threshold, the transmitting node will be disregarded when forming an upstream connection.

**Panel A** of the illustration above demonstrates how the RSSI threshold affects the number of parent node candidates an idle node has.

**Panel B** of the illustration above demonstrates how an RF shielding object can lower the RSSI of a potential parent node. Due to the RF shielding object, the area in which the RSSI of node X is above the threshold is significantly reduced. This causes the idle node to disregard node X even though node X is physically adjacent. The idle node will instead form an upstream connection with the physically distant node Y due to a stronger RSSI.

---

**注解:** Nodes technically still receive all beacon frames on the MAC layer. The RSSI threshold is an ESP-MESH feature that simply filters out all received beacon frames that are below the preconfigured threshold.

---

### Preferred Parent Node

When an idle node has multiple parent nodes candidates (potential parent nodes), the idle node will form an upstream connection with the **preferred parent node**. The preferred parent node is determined based on the following criteria:

- Which layer the parent node candidate is situated on
- The number of downstream connections (child nodes) the parent node candidate currently has

The selection of the preferred parent node will always prioritize the parent node candidate on the shallowest layer of the network (including the root node). This helps minimize the total number of layers in an ESP-MESH network



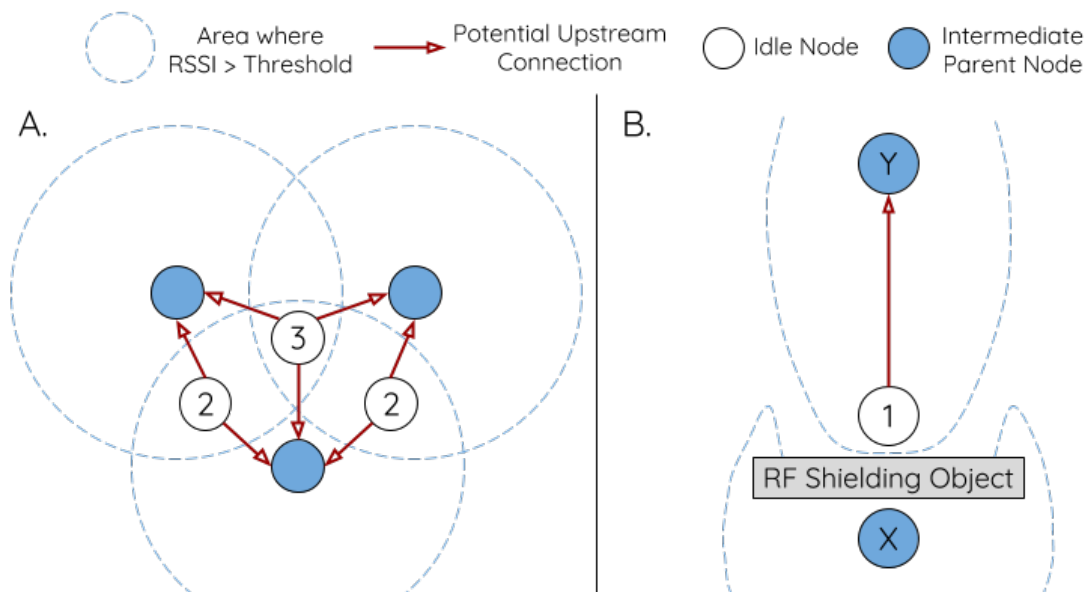


图 59: Effects of RSSI Thresholding

when upstream connections are formed. For example, given a second layer node and a third layer node, the second layer node will always be preferred.

If there are multiple parent node candidates within the same layer, the parent node candidate with the least child nodes will be preferred. This criteria has the effect of balancing the number of downstream connections amongst nodes of the same layer.

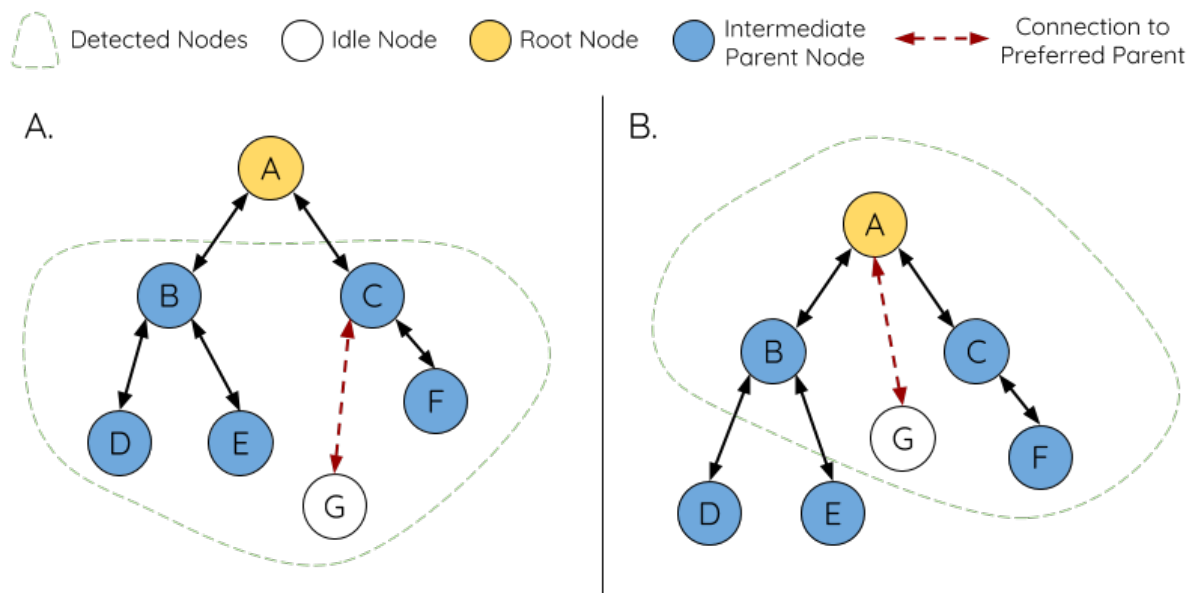


图 60: Preferred Parent Node Selection

**Panel A** of the illustration above demonstrates an example of how the idle node G selects a preferred parent node given the five parent node candidates B/C/D/E/F. Nodes on the shallowest layer are preferred, hence nodes B/C are prioritized since they are second layer nodes whereas nodes D/E/F are on the third layer. Node C is selected as the preferred parent node due it having fewer downstream connections (fewer child nodes) compared to node B.

**Panel B** of the illustration above demonstrates the case where the root node is within range of the idle node G. In other words, the root node's beacon frames are above the RSSI threshold when received by node G. The root node is always the shallowest node in an ESP-MESH network hence is always the preferred parent node given multiple

parent node candidates.

**注解:** Users may also define their own algorithm for selecting a preferred parent node, or force a node to only connect with a specific parent node (see the [Mesh Manual Networking Example](#)).

## Routing Tables

Each node within an ESP-MESH network will maintain its individual routing table used to correctly route ESP-MESH packets (see [ESP-MESH Packet](#)) to the correct destination node. The routing table of a particular node will **consist of the MAC addresses of all nodes within the particular node's subnetwork** (including the MAC address of the particular node itself). Each routing table is internally partitioned into multiple subtables with each subtable corresponding to the subnetwork of each child node.

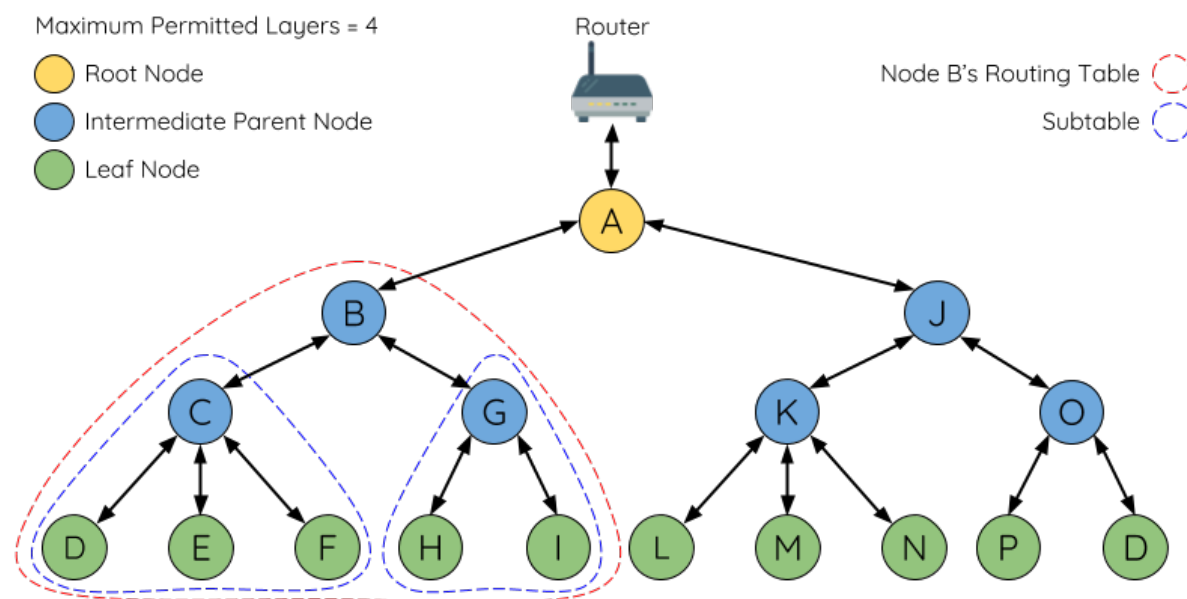


图 61: ESP-MESH Routing Tables Example

Using the diagram above as an example, the routing table of node B would consist of the MAC addresses of nodes B to I (i.e. equivalent to the subnetwork of node B). Node B's routing table is internally partitioned into two subtables containing of nodes C to F and nodes G to I (i.e. equivalent to the subnetworks of nodes C and G respectively).

**ESP-MESH utilizes routing tables to determine whether an ESP-MESH packet should be forwarded upstream or downstream based on the following rules.**

1. If the packet's destination MAC address is within the current node's routing table and is not the current node, select the subtable that contains the destination MAC address and forward the data packet downstream to the child node corresponding to the subtable.
2. If the destination MAC address is not within the current node's routing table, forward the data packet upstream to the current node's parent node. Doing so repeatedly will result in the packet arriving at the root node where the routing table should contain all nodes within the network.

**注解:** Users can call `esp_mesh_get_routing_table()` to obtain a node's routing table, or `esp_mesh_get_routing_table_size()` to obtain the size of a node's routing table.

`esp_mesh_get_subnet_nodes_list()` can be used to obtain the corresponding subtable of a specific child node. Likewise `esp_mesh_get_subnet_nodes_num()` can be used to obtain the size of the subtable.

## 4.26.4 Building a Network

### General Process

**警告:** Before the ESP-MESH network building process can begin, certain parts of the configuration must be uniform across each node in the network (see `mesh_cfg_t`). Each node must be configured with **the same Mesh Network ID, router configuration, and softAP configuration.**

An ESP-MESH network building process involves selecting a root node, then forming downstream connections layer by layer until all nodes have joined the network. The exact layout of the network can be dependent on factors such as root node selection, parent node selection, and asynchronous power-on reset. However, the ESP-MESH network building process can be generalized into the following steps:

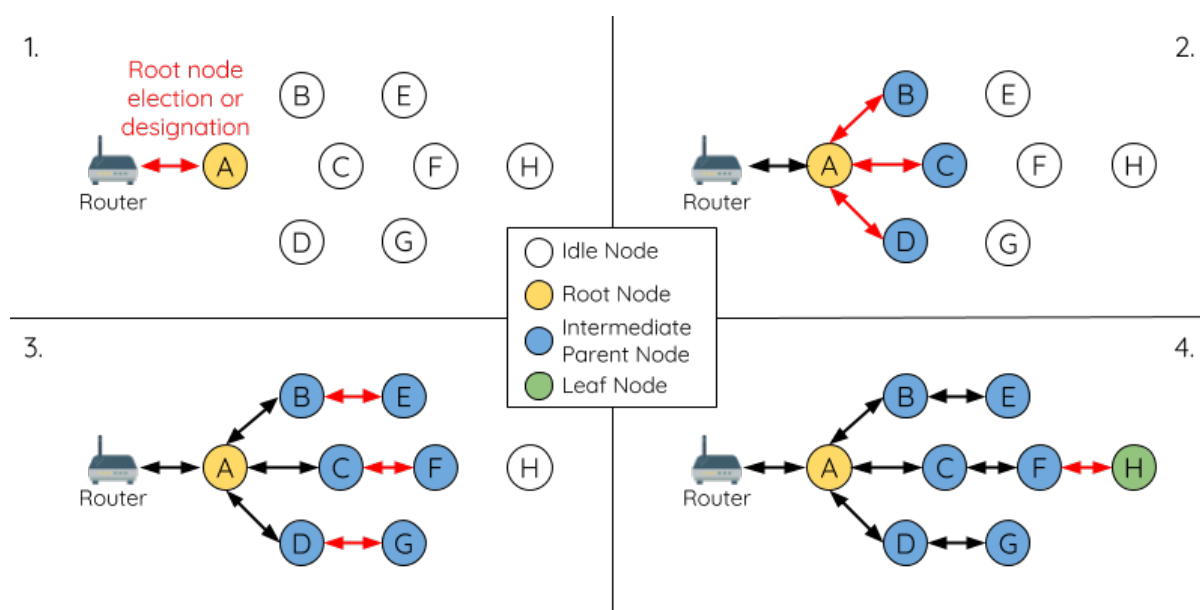


图 62: ESP-MESH Network Building Process

**1. Root Node Selection** The root node can be designated during configuration (see section on *User Designated Root Node*), or dynamically elected based on the signal strength between each node and the router (see *Automatic Root Node Selection*). Once selected, the root node will connect with the router and begin allowing downstream connections to form. Referring to the figure above, node A is selected to be the root node hence node A forms an upstream connection with the router.

**2. Second Layer Formation** Once the root node has connected to the router, idle nodes in range of the root node will begin connecting with the root node thereby forming the second layer of the network. Once connected, the second layer nodes become intermediate parent nodes (assuming maximum permitted layers > 2) hence the next layer to form. Referring to the figure above, nodes B to D are in range of the root node. Therefore nodes B to D form upstream connections with the root node and become intermediate parent nodes.

**3. Formation of remaining layers** The remaining idle nodes will connect with intermediate parent nodes within range thereby forming a new layer in the network. Once connected, the idle nodes become intermediate parent node or leaf nodes depending on the network's maximum permitted layers. This step is repeated until there are no more idle nodes within the network or until the maximum permitted layer of the network has been reached. Referring to the figure above, nodes E/F/G connect with nodes B/C/D respectively and become intermediate parent nodes themselves.

**4. Limiting Tree Depth** To prevent the network from exceeding the maximum permitted number of layers, nodes on the maximum layer will automatically become leaf nodes once connected. This prevents any other idle node from connecting with the leaf node thereby prevent a new layer form forming. However if an idle node has no other potential parent node, it will remain idle indefinitely. Referring to the figure above, the network's maximum permitted layers is set to four. Therefore when node H connects, it becomes a leaf node to prevent any downstream connections from forming.

#### Automatic Root Node Selection

The automatic selection of a root node involves an election process amongst all idle nodes based on their signal strengths with the router. Each idle node will transmit their MAC addresses and router RSSI values via Wi-Fi beacon frames. **The MAC address is used to uniquely identify each node in the network** whilst the **router RSSI** is used to indicate a node's signal strength with reference to the router.

Each node will then simultaneously scan for the beacon frames from other idle nodes. If a node detects a beacon frame with a stronger router RSSI, the node will begin transmitting the contents of that beacon frame (i.e. voting for the node with the stronger router RSSI). The process of transmission and scanning will repeat for a preconfigured minimum number of iterations (10 iterations by default) and result in the beacon frame with the strongest router RSSI being propagated throughout the network.

After all iterations, each node will individually check for its **vote percentage** (number of votes/number of nodes participating in election) to determine if it should become the root node. **If a node has a vote percentage larger than a preconfigured threshold (90% by default), the node will become a root node.**

The following diagram demonstrates how an ESP-MESH network is built when the root node is automatically selected.

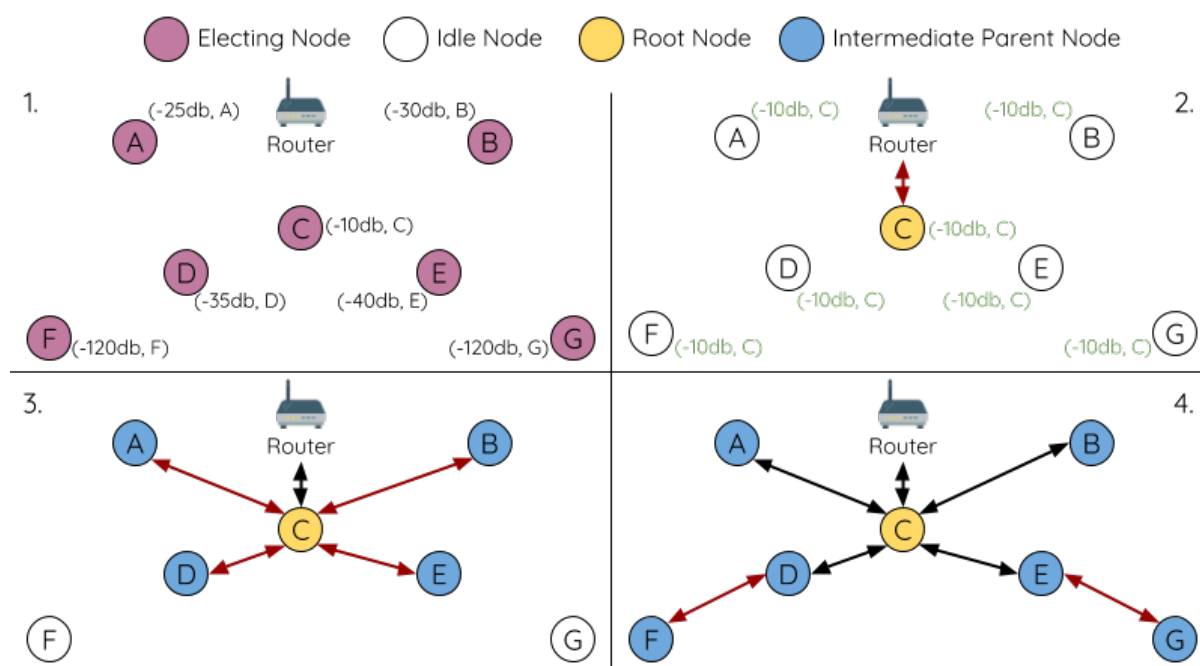


图 63: Root Node Election Example

1. On power-on reset, each node begins transmitting beacon frames consisting of their own MAC addresses and their router RSSIs.
2. Over multiple iterations of transmission and scanning, the beacon frame with the strongest router RSSI is propagated throughout the network. Node C has the strongest router RSSI (-10db) hence its beacon frame is propagated throughout the network. All nodes participating in the election vote for node C thus giving node C a vote percentage of 100%. Therefore node C becomes a root node and connects with the router.
3. Once Node C has connected with the router, nodes A/B/D/E connect with node C as it is the preferred parent node (i.e. the shallowest node). Nodes A/B/D/E form the second layer of the network.

4. Node F and G connect with nodes D and E respectively and the network building process is complete.

**注解:** The minimum number of iterations for the election process can be configured using `esp_mesh_set_attempts()`. Users should adjust the number of iterations based on the number of nodes within the network (i.e. the larger the network the larger number of scan iterations required).

**警告:** `Vote percentage threshold` can also be configured using `esp_mesh_set_vote_percentage()`. Setting a low vote percentage threshold **can result in two or more nodes becoming root nodes** within the same ESP-MESH network leading to the building of multiple networks. If such is the case, ESP-MESH has internal mechanisms to autonomously resolve the **root node conflict**. The networks of the multiple root nodes will be combined into a single network with a single root node. However, root node conflicts where two or more root nodes have the same router SSID but different router BSSID are not handled.

### User Designated Root Node

The root node can also be designated by user which will entail the designated root node to directly connect with the router and forgo the election process. When a root node is designated, all other nodes within the network must also forgo the election process to prevent the occurrence of a root node conflict. The following diagram demonstrates how an ESP-MESH network is built when the root node is designated by the user.

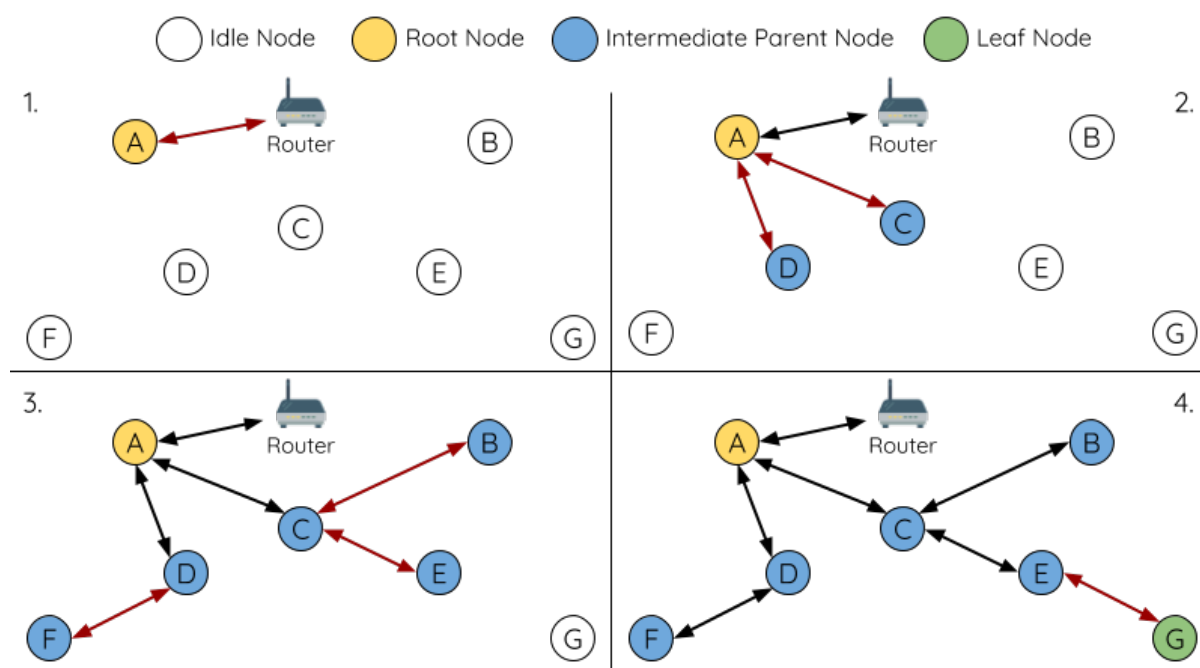


图 64: Root Node Designation Example (Root Node = A, Max Layers = 4)

1. Node A is designated the root node by the user therefore directly connects with the router. All other nodes forgo the election process.
2. Nodes C/D connect with node A as their preferred parent node. Both nodes form the second layer of the network.
3. Likewise, nodes B/E connect with node C, and node F connects with node D. Nodes B/E/F form the third layer of the network.
4. Node G connects with node E, forming the fourth layer of the network. However the maximum permitted number of layers in this network is configured as four, therefore node G becomes a leaf node to prevent any new layers from forming.

---

**注解:** When designating a root node, the root node should call `esp_mesh_set_parent()` in order to directly connect with the router. Likewise, all other nodes should call `esp_mesh_fix_root()` to forgo the election process.

---

### Parent Node Selection

By default, ESP-MESH is self organizing meaning that each node will autonomously select which potential parent node to form an upstream connection with. The autonomously selected parent node is known as the preferred parent node. The criteria used for selecting the preferred parent node is designed to reduce the number of layers in the ESP-MESH network and to balance the number of downstream connections between potential parent nodes (see section on *Preferred Parent Node*).

However ESP-MESH also allows users to disable self-organizing behavior which will allow users to define their own criteria for parent node selection, or to configure nodes to have designated parent nodes (see the [Mesh Manual Networking Example](#)).

### Asynchronous Power-on Reset

ESP-MESH network building can be affected by the order in which nodes power-on. If certain nodes within the network power-on asynchronously (i.e. separated by several minutes), **the final structure of the network could differ from the ideal case where all nodes are powered on synchronously**. Nodes that are delayed in powering on will adhere to the following rules:

**Rule 1:** If a root node already exists in the network, the delayed node will not attempt to elect a new root node, even if it has a stronger RSSI with the router. The delayed node will instead join the network like any other idle node by connecting with a preferred parent node. If the delayed node is the designated root node, all other nodes in the network will remain idle until the delayed node powers-on.

**Rule 2:** If a delayed node forms an upstream connection and becomes an intermediate parent node, it may also become the new preferred parent of other nodes (i.e. being a shallower node). This will cause the other nodes to switch their upstream connections to connect with the delayed node (see *Parent Node Switching*).

**Rule 3:** If an idle node has a designated parent node which is delayed in powering-on, the idle node will not attempt to form any upstream connections in the absence of its designated parent node. The idle node will remain idle indefinitely until its designated parent node powers-on.

The following example demonstrates the effects of asynchronous power-on with regards to network building.

1. Nodes A/C/D/F/G/H are powered-on synchronously and begin the root node election process by broadcasting their MAC addresses and router RSSIs. Node A is elected as the root node as it has the strongest RSSI.
2. Once node A becomes the root node, the remaining nodes begin forming upstream connections layer by layer with their preferred parent nodes. The result is a network with five layers.
3. Node B/E are delayed in powering-on but neither attempt to become the root node even though they have stronger router RSSIs (-20db and -10db) compared to node A. Instead both delayed nodes form upstream connections with their preferred parent nodes A and C respectively. Both Nodes B/E become intermediate parent nodes after connecting.
4. Nodes D/G switch their upstream connections as node B is the new preferred parent node due to it being on a shallower layer (second layer node). Due to the switch, the resultant network has three layers instead of the original five layers.

**Synchronous Power-On:** Had all nodes powered-on synchronously, node E would have become the root node as it has the strongest router RSSI (-10db). This would result in a significantly different network layout compared to the network formed under the conditions of asynchronous power-on. **However the synchronous power-on network layout can still be reached if the user manually switches the root node** (see `esp_mesh_waive_root()`).



图 65: Network Building with Asynchronous Power On Example

**注解:** Differences in parent node selection caused by asynchronous power-on are autonomously corrected for to some extent in ESP-MESH (see *Parent Node Switching*)

### Loop-back Avoidance, Detection, and Handling

A loop-back is the situation where a particular node forms an upstream connection with one of its descendant nodes (a node within the particular node's subnetwork). This results in a circular connection path thereby breaking the tree topology. ESP-MESH prevents loop-back during parent selection by excluding nodes already present in the selecting node's routing table (see *Routing Tables*) thus prevents a particular node from attempting to connect to any node within its subnetwork.

In the event that a loop-back occurs, ESP-MESH utilizes a path verification mechanism and energy transfer mechanism to detect the loop-back occurrence. The parent node of the upstream connection that caused the loop-back will then inform the child node of the loop-back and initiate a disconnection.

### 4.26.5 Managing a Network

**ESP-MESH is a self healing network meaning it can detect and correct for failures in network routing.** Failures occur when a parent node with one or more child nodes breaks down, or when the connection between a parent node and its child nodes becomes unstable. Child nodes in ESP-MESH will autonomously select a new parent node and form an upstream connection with it to maintain network interconnectivity. ESP-MESH can handle both Root Node Failures and Intermediate Parent Node Failures.

#### Root Node Failure

If the root node breaks down, the nodes connected with it (second layer nodes) will promptly detect the failure of the root node. The second layer nodes will initially attempt to reconnect with the root node. However after multiple failed attempts, the second layer nodes will initialize a new round of root node election. **The second layer node with the strongest router RSSI will be elected as the new root node** whilst the remaining second layer nodes will form an upstream connection with the new root node (or a neighboring parent node if not in range).

If the root node and multiple downstream layers simultaneously break down (e.g. root node, second layer, and third layer), the shallowest layer that is still functioning will initialize the root node election. The following example illustrates an example of self healing from a root node break down.

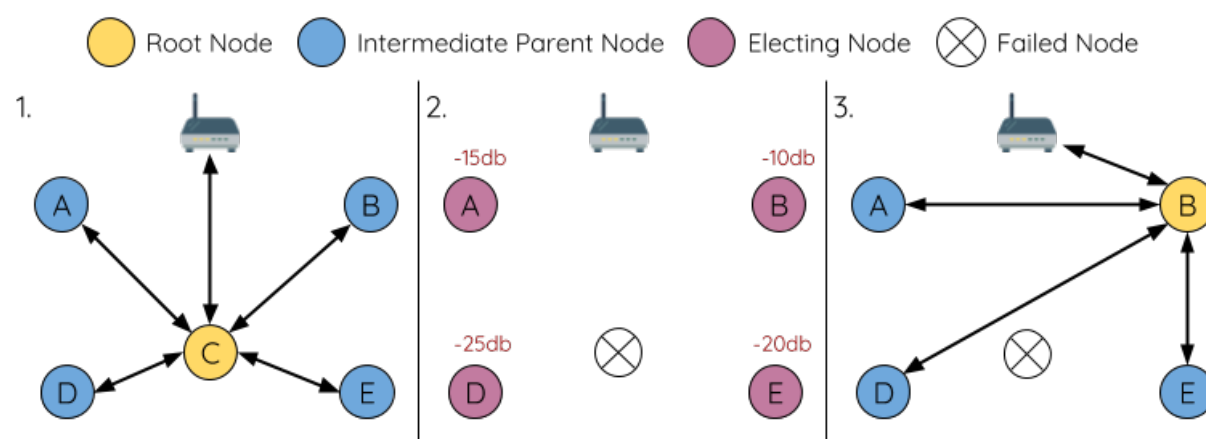


图 66: Self Healing From Root Node Failure

1. Node C is the root node of the network. Nodes A/B/D/E are second layer nodes connected to node C.
2. Node C breaks down. After multiple failed attempts to reconnect, the second layer nodes begin the election process by broadcasting their router RSSIs. Node B has the strongest router RSSI.



3. Node B is elected as the root node and begins accepting downstream connections. The remaining second layer nodes A/D/E form upstream connections with node B thus the network is healed and can continue operating normally.

**注解:** If a designated root node breaks down, the remaining nodes **will not autonomously attempt to elect a new root node** as an election process will never be attempted whilst a designated root node is used.

### Intermediate Parent Node Failure

If an intermediate parent node breaks down, the disconnected child nodes will initially attempt to reconnect with the parent node. After multiple failed attempts to reconnect, each child node will begin to scan for potential parent nodes (see *Beacon Frames & RSSI Thresholding*).

If other potential parent nodes are available, each child node will individually select a new preferred parent node (see *Preferred Parent Node*) and form an upstream connection with it. If there are no other potential parent nodes for a particular child node, it will remain idle indefinitely.

The following diagram illustrates an example of self healing from an Intermediate Parent Node break down.

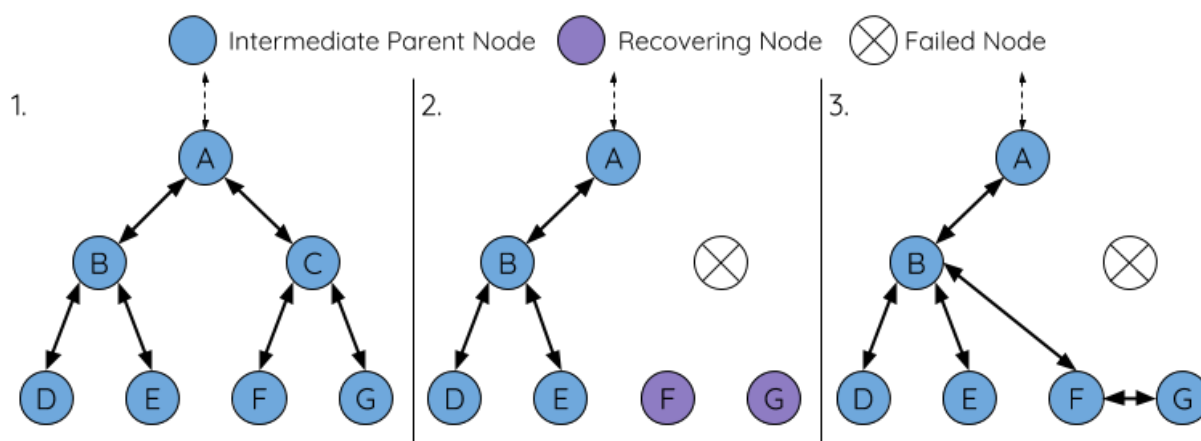


图 67: Self Healing From Intermediate Parent Node Failure

1. The following branch of the network consists of nodes A to G.
2. Node C breaks down. Nodes F/G detect the break down and attempt to reconnect with node C. After multiple failed attempts to reconnect, nodes F/G begin to select a new preferred parent node.
3. Node G is out of range from any other parent node hence remains idle for the time being. Node F is in range of nodes B/E, however node B is selected as it is the shallower node. Node F becomes an intermediate parent node after connecting with Node B thus node G can connect with node F. The network is healed, however the network routing as been affected and an extra layer has been added.

**注解:** If a child node has a designated parent node that breaks down, the child node will make no attempt to connect with a new parent node. The child node will remain idle indefinitely.

### Root Node Switching

ESP-MESH does not automatically switch the root node unless the root node breaks down. Even if the root node's router RSSI degrades to the point of disconnection, the root node will remain unchanged. Root node switching is the act of explicitly starting a new election such that a node with a stronger router RSSI will be elected as the new root node. This can be a useful method of adapting to degrading root node performance.

To trigger a root node switch, the current root node must explicitly call `esp_mesh_waive_root()` to trigger a new election. The current root node will signal all nodes within the network to begin transmitting and scanning for beacon frames (see [Automatic Root Node Selection](#)) **whilst remaining connected to the network (i.e. not idle)**. If another node receives more votes than the current root node, a root node switch will be initiated. **The root node will remain unchanged otherwise.**

A newly elected root node sends a **switch request** to the current root node which in turn will respond with an acknowledgment signifying both nodes are ready to switch. Once the acknowledgment is received, the newly elected root node will disconnect from its parent and promptly form an upstream connection with the router thereby becoming the new root node of the network. The previous root node will disconnect from the router **whilst maintaining all of its downstream connections** and enter the idle state. The previous root node will then begin scanning for potential parent nodes and selecting a preferred parent.

The following diagram illustrates an example of a root node switch.

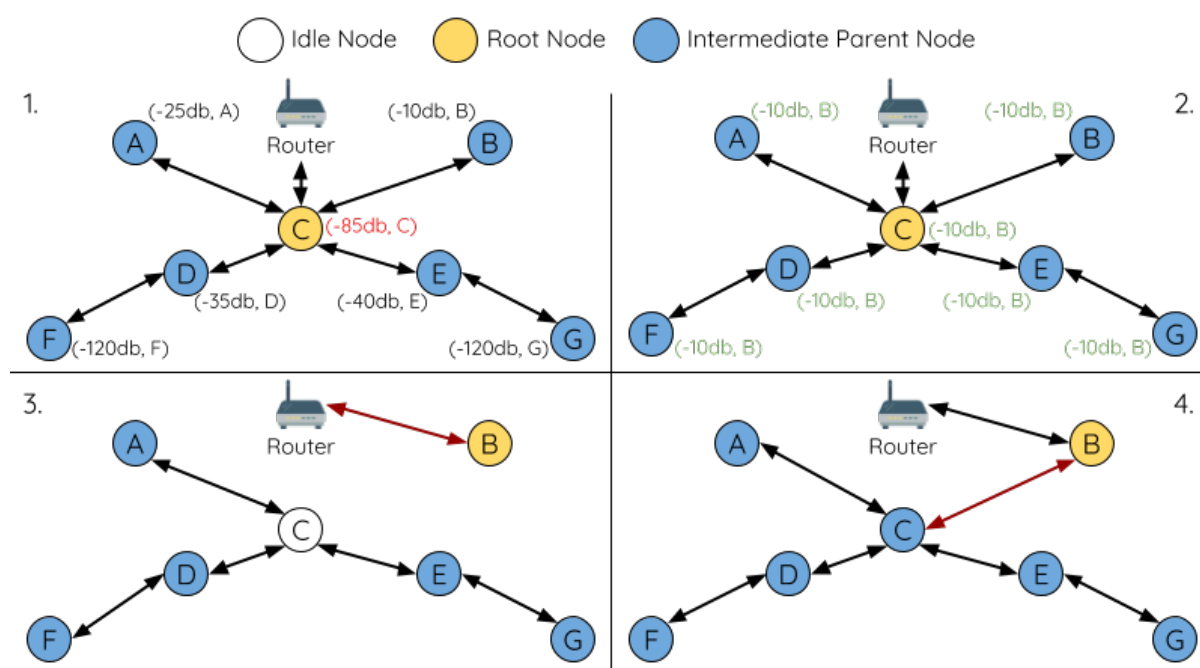


图 68: Root Node Switch Example

1. Node C is the current root node but has degraded signal strength with the router (-85db). The node C triggers a new election and all nodes begin transmitting and scanning for beacon frames **whilst still being connected**.

2. After multiple rounds of transmission and scanning, node B is elected as the new root node. Node B sends node C a **switch request** and node C responds with an acknowledgment.

3. Node B disconnects from its parent and connects with the router becoming the network's new root node. Node C disconnects from the router, enters the idle state, and begins scanning for and selecting a new preferred parent node. **Node C maintains all its downstream connections throughout this process.**

4. Node C selects node B as its preferred parent node, forms an upstream connection, and becomes a second layer node. The network layout is similar after the switch as node C still maintains the same subnetwork. However, each node in node C's subnetwork has been placed one layer deeper as a result of the switch. [Parent Node Switching](#) may adjust the network layout afterwards if any nodes have a new preferred parent node as a result of the root node switch.

**注解:** Root node switching must require an election hence is only supported when using a self-organized ESP-MESH network. In other words, root node switching cannot occur if a designated root node is used.

## Parent Node Switching

Parent Node Switching entails a child node switching its upstream connection to another parent node of a shallower layer. **Parent Node Switching occurs autonomously** meaning that a child node will change its upstream connection automatically if a potential parent node of a shallower layer becomes available (i.e. due to a *Asynchronous Power-on Reset*).

All potential parent nodes periodically transmit beacon frames (see *Beacon Frames & RSSI Thresholding*) allowing for a child node to scan for the availability of a shallower parent node. Due to parent node switching, a self-organized ESP-MESH network can dynamically adjust its network layout to ensure each connection has a good RSSI and that the number of layers in the network is minimized.

### 4.26.6 Data Transmission

#### ESP-MESH Packet

ESP-MESH network data transmissions use ESP-MESH packets. ESP-MESH packets are **entirely contained within the frame body of a Wi-Fi data frame**. A multi-hop data transmission in an ESP-MESH network will involve a single ESP-MESH packet being carried over each wireless hop by a different Wi-Fi data frame.

The following diagram shows the structure of an ESP-MESH packet and its relation with a Wi-Fi data frame.

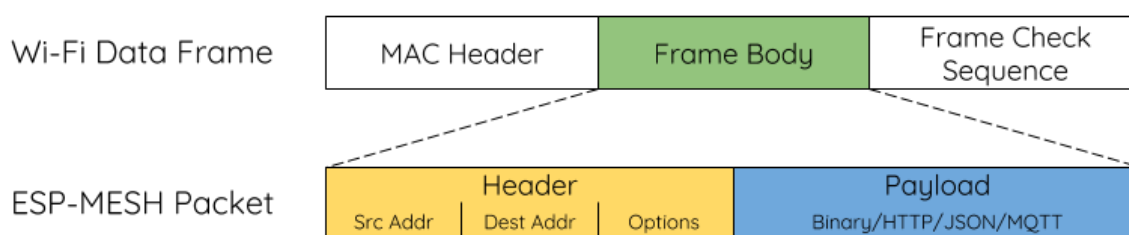


图 69: ESP-MESH Packet

The **header** of an ESP-MESH packet contains the MAC addresses of the source and destination nodes. The options field contains information pertaining to the special types of ESP-MESH packets such as a group transmission or a packet originating from the external IP network (see *MESH\_OPT\_SEND\_GROUP* and *MESH\_OPT\_RECV\_DS\_ADDR*).

The **payload** of an ESP-MESH packet contains the actual application data. This data can be raw binary data, or encoded under an application layer protocol such as HTTP, MQTT, and JSON (see *mesh\_proto\_t*).

---

**注解:** When sending an ESP-MESH packet to the external IP network, the destination address field of the header will contain the IP address and port of the target server rather than the MAC address of a node (see *mesh\_addr\_t*). Furthermore the root node will handle the formation of the outgoing TCP/IP packet.

---

## Group Control & Multicasting

Multicasting is a feature that allows a single ESP-MESH packet to be transmitted simultaneously to multiple nodes within the network. Multicasting in ESP-MESH can be achieved by either specifying a list of target nodes, or specifying a preconfigured group of nodes. Both methods of multicasting are called via *esp\_mesh\_send()*.

To multicast by specifying a list of target nodes, users must first set the ESP-MESH packet's destination address to the **Multicast-Group Address** (01:00:5E:xx:xx:xx). This signifies that the ESP-MESH packet is a multicast packet with a group of addresses, and that the address should be obtained from the header options. Users must then list the MAC addresses of the target nodes as options (see *mesh\_opt\_t* and *MESH\_OPT\_SEND\_GROUP*). This

method of multicasting requires no prior setup but can incur a large amount of overhead data as each target node's MAC address must be listed in the options field of the header.

Multicasting by group allows a ESP-MESH packet to be transmitted to a preconfigured group of nodes. Each grouping is identified by a unique ID, and a node can be placed into a group via `esp_mesh_set_group_id()`. Multicasting to a group involves setting the destination address of the ESP-MESH packet to the target group ID. Furthermore, the `MESH_DATA_GROUP` flag must set. Using groups to multicast incurs less overhead, but requires nodes to previously added into groups.

---

**注解:** During a multicast, all nodes within the network still receive the ESP-MESH packet on the MAC layer. However, nodes not included in the MAC address list or the target group will simply filter out the packet.

---

## Broadcasting

Broadcasting is a feature that allows a single ESP-MESH packet to be transmitted simultaneously to all nodes within the network. Each node essentially forwards a broadcast packet to all of its upstream and downstream connections such that the packet propagates throughout the network as quickly as possible. However, ESP-MESH utilizes the following methods to avoid wasting bandwidth during a broadcast.

1. When an intermediate parent node receives a broadcast packet from its parent, it will forward the packet to each of its child nodes whilst storing a copy of the packet for itself.
2. When an intermediate parent node is the source node of the broadcast, it will transmit the broadcast packet upstream to its parent node and downstream to each of its child nodes.
3. When an intermediate parent node receives a broadcast packet from one of its child nodes, it will forward the packet to its parent node and each of its remaining child nodes whilst storing a copy of the packet for itself.
4. When a leaf node is the source node of a broadcast, it will directly transmit the packet to its parent node.
5. When the root node is the source node of a broadcast, the root node will transmit the packet to all of its child nodes.
6. When the root node receives a broadcast packet from one of its child nodes, it will forward the packet to each of its remaining child nodes whilst storing a copy of the packet for itself.
7. When a node receives a broadcast packet with a source address matching its own MAC address, the node will discard the broadcast packet.
8. When an intermediate parent node receives a broadcast packet from its parent node which was originally transmitted from one of its child nodes, it will discard the broadcast packet

## Upstream Flow Control

ESP-MESH relies on parent nodes to control the upstream data flow of their immediate child nodes. To prevent a parent node's message buffer from overflowing due to an overload of upstream transmissions, a parent node will allocate a quota for upstream transmissions known as a **receiving window** for each of its child nodes. **Each child node must apply for a receiving window before it is permitted to transmit upstream.** The size of a receiving window can be dynamically adjusted. An upstream transmission from a child node to the parent node consists of the following steps:

1. Before each transmission, the child node sends a window request to its parent node. The window request consists of a sequence number which corresponds to the child node's data packet that is pending transmission.
2. The parent node receives the window request and compares the sequence number with the sequence number of the previous packet sent by the child node. The comparison is used to calculate the size of the receiving window which is transmitted back to the child node.
3. The child node transmits the data packet in accordance with the window size specified by the parent node. If the child node depletes its receiving window, it must obtain another receiving windows by sending a request before it is permitted to continue transmitting.

注解: ESP-MESH does not support any downstream flow control.

警告: Due to *Parent Node Switching*, packet loss may occur during upstream transmissions.

Due to the fact that the root node acts as the sole interface to an external IP network, it is critical that downstream nodes are aware of the root node's connection status with the external IP network. Failing to do so can lead to nodes attempting to pass data upstream to the root node whilst it is disconnected from the IP network. This results in unnecessary transmissions and packet loss. ESP-MESH address this issue by providing a mechanism to stabilize the throughput of outgoing data based on the connection status between the root node and the external IP network. The root node can broadcast its external IP network connection status to all other nodes by calling `esp_mesh_post_toDS_state()`.

### Bi-Directional Data Stream

The following diagram illustrates the various network layers involved in an ESP-MESH Bidirectional Data Stream.

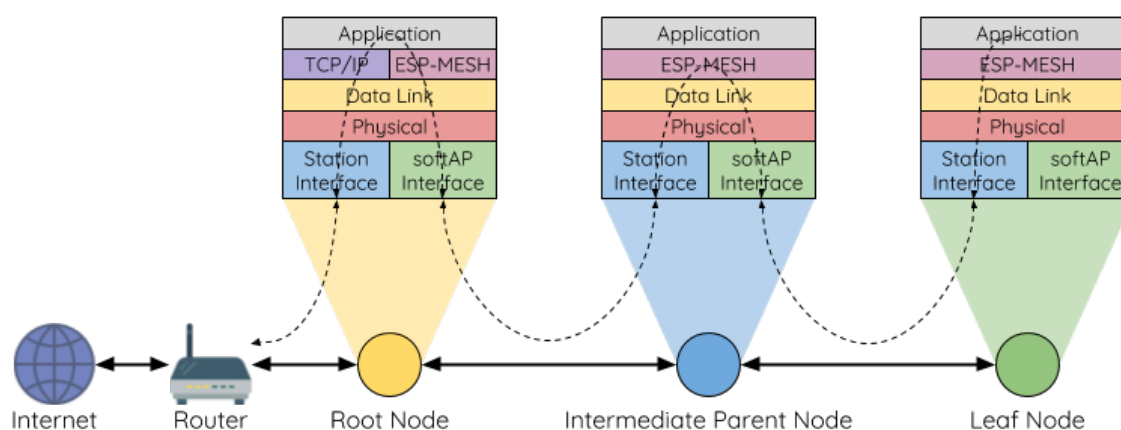


图 70: ESP-MESH Bidirectional Data Stream

Due to the use of *Routing Tables*, **ESP-MESH is able to handle pack forwarding entirely on the mesh layer**. A TCP/IP layer is only required on the root node when it transmits/receives a packet to/from an external IP network.

### 4.26.7 Channel Switching

#### Background

In traditional Wi-Fi networks, **channels** are predetermined frequency ranges. In an infrastructure basic service set (BSS), the serving AP and its connected stations must be on the same operating channels (1 to 14) in which beacons are transmitted. Physically adjacent BSS (Basic Service Sets) operating on the same channel can lead to interference and degraded performance.

In order to allow a BSS adapt to changing physical layer conditions and maintain performance, Wi-Fi contains mechanisms for **network channel switching**. A network channel switch is an attempt to move a BSS to a new operating channel whilst minimizing disruption to the BSS during this process. However it should be recognized that a channel switch may be unsuccessful in moving all stations to the new operating channel.

In an infrastructure Wi-Fi network, network channel switches are triggered by the AP with the aim of having the AP and all connected stations synchronously switch to a new channel. Network channel switching is implemented by embedding a **Channel Switch Announcement (CSA)** element within the AP's periodically transmitted beacon

frames. The CSA element is used to advertise to all connected stations regarding an upcoming network channel switch and will be included in multiple beacon frames up until the switch occurs.

A CSA element contains information regarding the **New Channel Number** and a **Channel Switch Count** which indicates the number of beacon frame intervals (TBTTs) remaining until the network channel switch occurs. Therefore, the Channel Switch Count is decremented every beacon frame and allows connected stations to synchronize their channel switch with the AP.

### ESP-MESH Network Channel Switching

ESP-MESH Network Channel Switching also utilize beacon frames that contain a CSA element. However, being a multi-hop network makes the switching process in ESP-MESH is more complex due to the fact that a beacon frame might not be able to reach all nodes within the network (i.e. in a single hop). Therefore, an ESP-MESH network relies on nodes to forward the CSA element so that it is propagated throughout the network.

When an intermediate parent node with one or more child nodes receives a beacon frame containing a CSA, the node will forward the CSA element by including the element in its next transmitted beacon frame (i.e. with the same **New Channel Number** and **Channel Switch Count**). Given that all nodes within an ESP-MESH network receive the same CSA, the nodes can synchronize their channel switches using the Channel Switch Count, albeit with a short delay due to CSA element forwarding.

An ESP-MESH network channel switch can be triggered by either the router or the root node.

**Root Node Triggered** A root node triggered channel switch can only occur when the ESP-MESH network is not connected to a router. By calling `esp_mesh_switch_channel()`, the root node will set an initial Channel Switch Count value and begin including a CSA element in its beacon frames. Each CSA element is then received by second layer nodes, and forwarded downstream in the their own beacon frames.

**Router Triggered** When an ESP-MESH network is connected to a router, the entire network must use the same channel as the router. Therefore, **the root node will not be permitted to trigger a channel switch when it is connected to a router.**

When the root node receives beacon frame containing a CSA element from the router, **the root node will set Channel Switch Count value in the CSA element to a custom value before forwarding it downstream via beacon frames.** It will also decrement the Channel Switch Count of subsequent CSA elements relative to the custom value. This custom value can be based on factors such as the number of network layers, the current number of nodes etc.

The setting the Channel Switch Count value to a custom value is due to the fact that the ESP-MESH network and its router may have a different and varying beacon intervals. Therefore, the Channel Switch Count value provided by the router is irrelevant to an ESP-MESH network. By using a custom value, nodes within the ESP-MESH network are able to switch channels synchronously relative to the ESP-MESH network's beacon interval. However, this will also result in the ESP-MESH network's channel switch being unsynchronized with the channel switch of the router and its connected stations.

### Impact of Network Channel Switching

- **Due to the ESP-MESH network channel switch being unsynchronized with the router's channel switch, there will be**
  - The ESP-MESH network's channel switch time is dependent on the ESP-MESH network's beacon interval and the root node's custom Channel Switch Count value.
  - The channel discrepancy prevents any data exchange between the root node and the router during that ESP-MESH network's switch.
  - In the ESP-MESH network, the root node and intermediate parent nodes will request their connected child nodes to stop transmissions until the channel switch takes place by setting the **Channel Switch Mode** field in the CSA element to 1.
  - Frequent router triggered network channel switches can degrade the ESP-MESH network's performance. Note that this can be caused by the ESP-MESH network itself (e.g. due to wireless medium

contention with ESP-MESH network). If this is the case, users should disable the automatic channel switching on the router and use a specified channel instead.

- **When there is a temporary channel discrepancy, the root node remains technically connected to the router.**
  - Disconnection occurs after the root node fails to receive any beacon frames or probe responses from the router over a fixed number of router beacon intervals.
  - Upon disconnection, the root node will automatically re-scan all channels for the presence of a router.
- **If the root node is unable to receive any of the router's CSA beacon frames (e.g. due to short switch time given by the router).**
  - After the router switches channels, the root node will no longer be able to receive the router's beacon frames and probe responses and result in a disconnection after a fixed number of beacon intervals.
  - The root node will re-scan all channels for the router after disconnection.
  - The root node will maintain downstream connections throughout this process.

---

**注解:** Although ESP-MESH network channel switching aims to move all nodes within the network to a new operating channel, it should be recognized that a channel switch might not successfully move all nodes (e.g. due to reasons such as node failures).

---

### Channel and Router Switching Configuration

ESP-MESH allows for autonomous channel switching to be enabled/disabled via configuration. Likewise, autonomous router switching (i.e. when a root node autonomously connects to another router) can also be enabled/disabled by configuration. Autonomous channel switching and router switching is dependent on the following configuration parameters and run-time conditions.

**Allow Channel Switch:** This parameter is set via the `allow_channel_switch` field of the `mesh_cfg_t` structure and permits an ESP-MESH network to dynamically switch channels when set.

**Preset Channel:** An ESP-MESH network can have a preset channel by setting the `channel` field of the `mesh_cfg_t` structure to the desired channel number. If this field is unset, the `allow_channel_switch` parameter is overridden such that channel switches are always permitted.

**Allow Router Switch:** This parameter is set via the `allow_router_switch` field of the `mesh_router_t` and permits an ESP-MESH to dynamically switch to a different router when set.

**Preset Router BSSID:** An ESP-MESH network can have a preset router by setting the `bssid` field of the `mesh_router_t` structure to the BSSID of the desired router. If this field is unset, the `allow_router_switch` parameter is overridden such that router switches are always permitted.

**Root Node Present:** The presence of a root node will also affect whether or a channel or router switch is permitted.

The following table illustrates how the different combinations of parameters/conditions affect whether channel switching and/or router switching is permitted. Note that X represents a “don't care” for the parameter.

Configuration and Conditions						Result						
Preset Channel	Allow Channel Switch	Preset BSSID	Router	Allow Router Switch	Root Node Present	Permitted Switches						
N	X	N		X	X	Channel & Router						
		Y		N		Channel Only						
		Y		Y		Channel & Router						
Y	Y	N		X	X	Channel & Router						
						N	N	Router Only				
						N	Y	Channel & Router				
	Y	Y	Y		N	X	Channel Only					
							N	N	None			
							N	Y	Channel Only			
					Y	Y		Y	X	Channel & Router		
										N	N	Router Only
										N	Y	Channel & Router

### 4.26.8 Performance

The performance of an ESP-MESH network can be evaluated based on multiple metrics such as the following:

**Network Building Time:** The amount of time taken to build an ESP-MESH network from scratch.

**Healing Time:** The amount of time taken for the network to detect a node break down and carry out appropriate actions to heal the network (such as generating a new root node or forming new connections).

**Per-hop latency:** The latency of data transmission over one wireless hop. In other words, the time taken to transmit a data packet from a parent node to a child node or vice versa.

**Network Node Capacity:** The total number of nodes the ESP-MESH network can simultaneously support. This number is determined by the maximum number of downstream connections a node can accept and the maximum number of layers permissible in the network.

The following table lists the common performance figures of an ESP-MESH network. However users should note that performance numbers can vary greatly between installations based on network configuration and operating environment.

Function	Description
Networking Building Time	< 60 seconds
Healing time	Root Node Break Down: < 10 seconds Child Node Break Down: < 5 seconds
Per-hop latency	10 to 30 milliseconds

**注解:** The following test conditions were used to generate the performance figures above.

- Number of test devices: **100**
- Maximum Downstream Connections to Accept: **6**
- Maximum Permissible Layers: **6**

**注解:** Throughput depends on packet error rate and hop count.



---

**注解:** The throughput of root node's access to the external IP network is directly affected by the number of nodes in the ESP-MESH network and the bandwidth of the router.

---

### 4.26.9 Further Notes

- Data transmission uses Wi-Fi WPA2-PSK encryption
- Mesh networking IE uses AES encryption

Router and internet icon made by [Smashicons](http://Smashicons) from [www.flaticon.com](http://www.flaticon.com)

## 4.27 BluFi

### 4.27.1 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 配置和证书传输到 ESP32，然后 ESP32 可基于这些信息连接到 AP 或建立 SoftAP。

BluFi 流程的关键部分包括数据的分片、加密、校验和验证。

用户可按需自定义用于对称加密、非对称加密和校验的算法。这里我们采用 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。

### 4.27.2 BluFi 流程

BluFi 配网功能包含配置 SoftAP 和 Station 两部分。

下面以配置 Station 为例说明配置步骤。BluFi 配网的配置 Station 包含广播、连接、服务发现、协商共享密钥、传输数据、回传连接状态等步骤。

### 4.27.3 ESP32 配网流程

1. ESP32 开启 GATT Server 功能，发送带有特定 *adv data* 的广播。你可以自定义该广播，该广播不属于 BluFi Profile。
2. 使用手机 APP 搜索到该特定广播，手机作为 GATT Client 连接 ESP32。你可以决定使用哪款手机 APP。
3. GATT 连接建立成功后，手机向 ESP32 发送“协商过程”数据帧（详情见 [BluFi 传输格式](#)）。
4. ESP32 收到“协商过程”数据帧后，会按照使用者自定义的协商过程来解析。
5. 手机与 ESP32 进行密钥协商。协商过程可使用 DH/RSA/ECC 等加密算法进行。
6. 协商结束后，手机端向 ESP32 发送“设置安全模式”控制帧。
7. ESP32 收到“设置安全模式”控制帧后，使用经过协商的共享密钥以及配置的安全策略对通信数据进行加密和解密。
8. 手机向 ESP32 发送“BluFi 传输格式”定义的 SSID、Password 等用于 Wi-Fi 连接的必要信息。
9. 手机向 ESP32 发送“Wi-Fi 连接请求”控制帧，ESP32 收到之后，识别为手机已将必要的信息传输完毕，准备连接 Wi-Fi。
10. ESP32 连接到 Wi-Fi 后，发送“Wi-Fi 连接状态报告”控制帧到手机，以报告连接状态。至此配网结束。

---

**注解:**

1. 安全模式设置可在任何时候进行，ESP32 收到安全模式的配置后，会根据安全模式指定的模式进行安全相关的操作。
  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。
-

#### 4.27.4 配网流程图

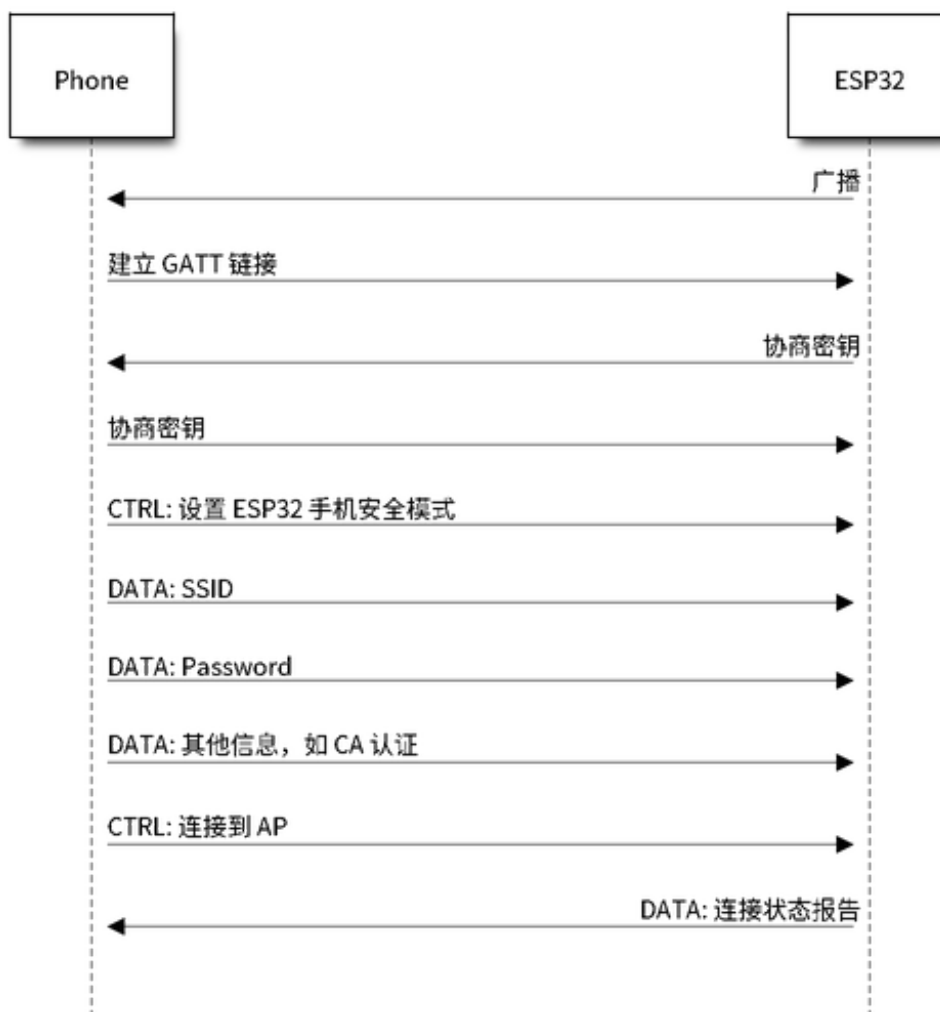


图 71: BluFi Flow Chart

#### 4.27.5 BluFi 传输格式

手机 APP 与 ESP32 之间的 BluFi 通信格式定义如下:

帧不分片情况下的标准格式 (8 bit):

Description	Value
LSB - Type	1
Frame Control	1
Sequence Number	1
Data Length	1
Data	\${Data Length}
MSB - CheckSum	2

如果 **Frame Control** 帧中的 **More Frag** 使能, 则 **Total Content Length** 为数据帧中剩余部分的总长度, 用于报告终端需要分配多少内存。

帧分片格式 (8 bit):

Description	Value	
LSB - Type	1	
FrameControl(Frag)	1	
SequenceNumber	1	
DataLength	1	
Data	Total Content Length	2
	Content	\${Data Length} - 2
MSB - CheckSum	2	

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):

Description	Value	
LSB - Type (Ack)	1	
Frame Control	1	
SequenceNumber	1	
DataLength	1	
Data	Acked Sequence Number	2
MSB - CheckSum	2	

#### 1. Type

类型域，占 1 byte。分为 Type 和 Subtype（子类型域）两部分，Type 占低 2 bit，Subtype 占高 6 bit。

- 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 (0x0 b' 00)

控制帧 (二进制)	含义	解释	备注
0x0 (b' 000000)	Ack	用来回复对方发的帧, Ack 帧的 Data 域使用回复对象帧的 Sequence 值。	Data 域使用 1 byte Sequence 值, 与恢复对象帧的 Sequence 值相同。
0x1 (b' 000001)	Set the security mode.	通知 ESP32 发送数据时使用的安全模式, 在该过程中可设置多次, 每次设置后影响后续安全模式。在不设置的情况下, ESP32 默认控制帧和数据帧均为无校验、无加密。手机到 ESP32 方向依赖于帧 Control 域。	Data 域占用 1 byte。高 4 bit 为控制帧的安全模式, 低 4bit 为数据帧的安全模式。 b' 0000: 无校验、无加密; b' 0001: 有校验、无加密; b' 0010: 无校验、有加密; b' 0011: 有校验、有加密。
0x2 (b' 000010)	Set the Wi-Fi op-mode of ESP32.	设置 ESP32 的 Wi-Fi 模式, 帧包含 opmode 信息。	data[0] 用于表示 opmode 类型, 包括: 0x00: NULL; 0x01: STA; 0x02: SoftAP; 0x03: SoftAP&STA. 如果设置有包含 AP, 请尽量优先设置 AP 模式的 SSID/Password/Max Conn Number 等。
0x3 (b' 000011)	Connect the AP.	通知 ESP32, 必要的信息已经发送完毕, 可以连接 AP。	不包含 Data 域。
0x4 (b' 000100)	Disconnect the AP.	通知 ESP32 断开与 AP 的连接	不包含 Data 域。
0x5 (b' 000101)	Get the status of Wi-Fi.	获取 ESP32 的 Wi-Fi 模式和状态等信息。	不包含 Data 域。ESP32 收到此控制帧后, 后续会通过 Wi-Fi 连接状态报告 (Wi-Fi Connection State Report) 数据帧来回复手机端当前所处的 opmode、连接状态、SSID 等信息。提供给手机端的信息由应用决定。
0x6 (b' 000110)	Disconnect the STA device from the SoftAP in SoftAP mode.	处于 SoftAP 模式时, 踢掉某个 STA 设备。	data[0~5] 为 STA 设备的 MAC 地址, 如有多个 STA, 则 [6-11] 为第二个, 依次类推。
0x7 (b' 000111)	Get the version.		
0x8 (b' 001000)	Tell ESP32 disconnect the BLE GATT link.	通知 ESP32 断开蓝牙连接。	ESP32 收到该指令后主动断开蓝牙连接。

## 1.2 数据帧 (0x1 b' 01)

数据帧 (二进制)	含义	解释
0x0 (b' 000000)	Negotiation data.	用来发送协商数据, 传输到应用层注册的回调函数。
0x1 (b' 000001)	BSSID for STA mode.	STA 将要连接的 AP 的 BSSID (用于隐藏 SSID)。
0x2 (b' 000010)	SSID for STA mode.	STA 将要连接的 AP 的 SSID。
0x3 (b' 000011)	Password for STA mode.	STA 将要连接的 AP 的密码。
0x4 (b' 000100)	SSID for SoftAP mode.	SoftAP 模式使用的 SSID。
0x5 (b' 000101)	Password for Soft-AP mode.	SoftAP 模式使用的密码。
0x6 (b' 000110)	Max connection number for SoftAP mode.	AP 模式的最大连接数。
0x7 (b' 000111)	Authentication mode for SoftAP mode.	AP 模式的认证模式。
0x8 (b' 001000)	Channel for SoftAP mode.	SoftAP 模式的通道数量。
0x9 (b' 001001)	Username.	使用企业级加密时, Client 端的用户名。
0xa (b' 001010)	CA certification.	进行企业级加密时使用的 CA 证书。
0xb (b' 001011)	Client certification.	进行企业级加密时, Client 端的证书。 可包含或不包含私钥, 由证书内容决定。
0xc (b' 001100)	Server certification.	进行企业级加密时, Server 端的证书。 可包含或不包含私钥, 由证书内容决定。
0xd (b' 001101)	Client private key.	进行企业级加密时, Client 端的私钥。
0xe (b' 001110)	Server private key.	进行企业级加密时, Server 端的私钥。
0xf (b' 001111)	Wi-Fi connection state report.	通知手机 ESP32 的 Wi-Fi 状态, 包括 STA 状态和 SoftAP 状态, 用于手机配置。但收到手机询问 Wi-Fi 状态时, 除了回复此帧外, 还可回复其他数据帧。
0x10 (b' 010000)	Version.	
0x11 (b' 010001)	Wi-Fi list.	通知手机 ESP32 周围的 Wi-Fi 热点列表。
0x12 (b' 010010)	Report error.	通知手机 BluFi 过程出现异常错误。

数据帧 (二进制)	含义	解释
0x13 (b' 010011)	Custom data.	用户发送或者接收自定义数据。

## 2. Frame Control

帧控制域，占 1 byte，每个 bit 表示不同含义。

位	含义
0x01	表示帧是否加密。
	1 表示加密，0 表示未加密。
	加密部分帧括完整的 DATA 域加密之前的明文（不帧含末尾的校验）。
	控制帧暂不加密，故控制帧此位为 0。
0x02	表示帧 Data 域结尾是否帧含校验（例如 SHA1,MD5,CRC 等）需要校验的数据域包括 sequnce + data length + 明文 data。
	控制帧和数据帧都可以包含校验位或不包含。
0x04	表示数据方向。
	0 表示手机发向 ESP32；
	1 表示 ESP32 发向手机。
0x08	表示是否要求对方回复 ack。
	0 表示不要求；
	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。
	如果是 Frag 帧，则告知当前 content 部分 + 后续 content 部分的总长度，位于 Data 域的前 2 字节（即最大支持 64K 的 content 数据）。
0x10~0x80 保留	

## 3. Sequence Control

序列控制域。帧发送时，无论帧的类型是什么，序列 (Sequence) 都会自动加 1，用来防止重放攻击 (Replay Attack)。每次重现连接后，序列清零。

## 4. Length

Data 域的长度，不包含 CheckSum。

## 5. Data

不同的 Type 或 Subtype，Data 域的含义均不同。请参考上方表格。

## 6. CheckSum

此域为 2 byte 的校验，用来校验『序列 + 数据长度 + 明文数据』。

### 4.27.6 ESP32 端的安全实现

#### 1. 保证数据安全

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）一个共享密钥。

#### 2. 保证数据完整性

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。有些算法如 DH，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)

加入帧发送序列 (Sequence)，并且序列参与数据校验。

在 ESP32 端的代码中，你可以决定和开发密钥协商等安全处理的流程参考上述流程图)。手机应用向 ESP32 发送协商数据，将传送给应用层处理。如果应用层不处理，可使用 BluFi 提供的 DH 加密算法来磋商密钥。应用层需向 BluFi 注册以下几个与安全相关的函数：

```
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)，处理完成后，需要将待发送的数据使用 output\_data 和 output\_len 传出。

BluFi 会在调用完 negotiate\_data\_handler 后，发送 negotiate\_data\_handler 传出的 output\_data。

这里的两个『\*』，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，通过 need\_free 通知是否需要释放内存。

```
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 bit 序列 (sequence)，可作为 iv 的某 8 bit 来使用。

```
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 bit 序列 (sequence)，可作为 iv 的某 8 bit 来使用。

```
typedef uint16_t (*esp_blufi_checksum_func_t)(uint8_t iv8, uint8_t *data, int len);
```

该函数用来计算 CheckSum，返回值为 CheckSum 的值。BluFi 会使用该函数返回值与包末尾的 CheckSum 做比较。

## 4.27.7 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

## 4.28 片外 RAM

### 4.28.1 简介

ESP32 提供了 520 KB 的片上 SRAM，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 存储器以供用户使用。片外 RAM 被添加到内存映射中，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。ESP32 支持多种类型的 RAM 芯片，但 ESP32 SDK 当前仅支持 ESP-PSRAM32 芯片。

ESP-PSRAM32 芯片的工作电压为 1.8 V，只能与 1.8 V flash 并联使用。请确保在启动时将 MTDI 管脚设置为高电平，或者将 ESP32 中的熔丝设置为始终使用 1.8 V 的 VDD\_SIO 电平，否则有可能会损坏 PSRAM 和/或 flash 芯片。

要将 ESP-PSRAM 芯片连接到 ESP32D0W\*，请连接以下信号：

- PSRAM /CE (pin 1) > ESP32 GPIO 16

- PSRAM SO (pin 2) > flash DO
- PSRAM SIO[2] (pin 3) > flash WP
- PSRAM SI (pin 5) > flash DI
- PSRAM SCLK (pin 6) > ESP32 GPIO 17
- PSRAM SIO[3] (pin 7) > flash HOLD
- PSRAM Vcc (pin 8) > ESP32 VCC\_SDIO

ESP32D2W\* 芯片的连接方式有待确定。

---

**注解：**乐鑫同时提供 ESP32-WROVER 模组，内部搭载 ESP32 芯片，集成 1.8 V flash 和 ESP-PSRAM32，可直接用于终端产品 PCB 中。

---

### 4.28.3 配置片外 RAM

ESP-IDF 完全支持将外部存储器集成到您的应用程序中。您可以将 ESP-IDF 配置成启动并完成初始化后以多种方式处理片外 RAM：

- 集成片外 RAM 到 ESP32 内存映射
- 添加片外 RAM 到内存分配程序
- 调用 `malloc()` 分配片外 RAM（默认）
- 允许 `.bss` 段放入片外存储器

#### 集成片外 RAM 到 ESP32 内存映射

在 `CONFIG_SPIRAM_USE` 中选择 “Integrate RAM into ESP32 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 被映射到以 0x3F800000 起始的数据地址空间（字节可寻址），空间大小正好为 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，同时应用程序全权负责管理片外 RAM，包括协调 Buffer 的使用、防止发生损坏等。

#### 添加片外 RAM 到内存分配程序

在 `CONFIG_SPIRAM_USE` 中选择 “Make RAM allocatable using heap\_caps\_malloc( ..., MALLOC\_CAP\_SPIRAM)” 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内存分配程序里携带 `MALLOC_CAP_SPIRAM` 的标志

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()` 返回的 RAM 中。

这允许应用程序使用片外 RAM 而无需重写代码以使用 `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 Buffer 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` 定义一个内部存储池，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` 旗标从该池中分配存储器。

### 允许 .bss 段放入片外存储器

设置 `CONFIG_SPIRAM_ALLOW_BSS_SEG_EXTERNAL_MEMORY` 启用该选项，此选项配置与上面三个选项互不影响。

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` 库中零初始化的数据（BSS 段）。

`EXT_RAM_ATTR` 宏应用于任何静态声明（未初始化为非零值）之后，可以将附加数据从内部 BSS 段移到片外 RAM。

启用此选项可以减少 BSS 段占用的内部静态存储。

剩余的片外 RAM 也可以通过上述方法添加到内存分配程序中。

## 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 描述符，也不能用作 DMA 读写操作的缓冲区 (Buffer)。与 DMA 搭配使用的 Buffer 必须先使用 `heap_caps_malloc(size, MALLOC_CAP_DMA)` 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，即频繁在片外 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，访问大块数据可以“挤出” flash cache，可能会降低代码执行速度。
- 片外 RAM 不可用作任务堆栈存储器。因此 `xTaskCreate()` 及类似函数将始终为堆栈和任务 TCB 分配片上存储器，而 `xTaskCreateStatic()` 类型的函数将检查传递的 Buffer 是否属于片上存储器。但对于不以任何方式直接或间接调用 ROM 中代码的任务，`menuconfig` 选项 `CONFIG_SPIRAM_ALLOW_STACK_EXTERNAL_MEMORY` 将消除 `xTaskCreateStatic` 中的检查，从而允许任务堆栈存储在外部 RAM 中。但是，不建议使用此方法。
- 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`，`CONFIG_SPIRAM_IGNORE_NOTFOUND` 选项将不能使用，这是因为在链接时，链接器已经向片外 RAM 分配符号。
- 时钟频率为 80 MHz 时，片外 RAM 须占用 HSPI 总线或 VSPI 总线。请使用 `CONFIG_SPIRAM_OCCUPY_SPI_HOST` 选择要用的 SPI 主机。

## 4.28.5 芯片版本

有些 ESP32 芯片版本存在某些已知问题，可能会影响片外 RAM 的使用。请参考 ESP32 勘误表，查看详细信息。为了解决这些问题，ESP-IDF 采取了以下措施：

## ESP32 rev v0

ESP-IDF 尚未提供针对此版本硅片 bug 的解决方法，因此在 ESP32 rev v0 中，ESP-IDF 无法将片外 PSRAM 映射到 ESP32 主内存映射中。

## ESP32 rev v1

当某些机器指令序列在片外存储器位置上运行时，此芯片版本中的错误可能会引发芯片故障（详情见 ESP32 勘误表 第 3.2 章节）。为了解决这个问题，用于编译 ESP-IDF 项目的 GCC 编译器扩展了一个旗标：`-mfix-esp32-psram-cache-issue`。在命令行中将此旗标传递给 GCC，编译器对这些序列进行处理，然后仅输出可以安全执行的代码。如需启用此旗标，请选择 `CONFIG_SPIRAM_CACHE_WORKAROUND`。

ESP-IDF 还采取了其他措施确保不同时使用 PSRAM 访问和出错指令集：

- 链接到使用 GCC 旗标重新编译的 Newlib 版本；
- 避免使用某些 ROM 函数；
- 为 Wi-Fi 栈分配静态内存。

## 4.29 链接脚本生成机制

### 4.29.1 概述

ESP32 的代码和数据可以存放在多个内存区域。通常，代码和只读数据存放在 flash 区域，可写数据存放在内存中。我们经常需要更改代码或者数据的默认映射区域，例如为了提高性能，将关键部分的代码和只读数据放置到内存中，或者将代码、数据和只读数据存放到 RTC 内存中以便在唤醒桩和 ULP 协处理器中使用。

IDF 的链接脚本生成机制允许用户在组件级别定义代码和数据的存放区域。组件通过链接片段文件描述如何映射目标文件的输入段（甚至可以是某个具体的函数或者数据）。在构建应用程序时，链接片段文件会被收集、解析并处理，然后扩充到链接脚本模板中形成最终的链接脚本文件，该链接脚本会被用于链接最终的二进制应用程序。

### 4.29.2 快速上手

本节将指导如何快速将代码和数据放入 RAM 和 RTC 内存中，并演示如何使这些放置规则依赖于项目的配置。本节内容重在指导快速入门，因此并未在使用前详细介绍所有涉及的术语和概念，但在首次提及此类术语或概念时，均提供了相应链接，以帮助您的理解。

#### 准备工作

**Make** 在组件目录中新建一个链接片段文件，该文件是一个扩展名为 `.lf` 的文本文件。为了能够让构建系统收集到此片段文件，需要为组件添加一个条目，在调用 `register_component` 之前设置 `COMPONENT_ADD_LDFRAGMENTS` 变量的值，使其指向刚才的链接片段文件。

```
# 文件路径相对于组件的 Makefile
COMPONENT_ADD_LDFRAGMENTS += "path/to/linker_fragment_file.lf" "path/to/another_
↪linker_fragment_file.lf"
```

**CMake** 对于 CMake 构建系统来说，需要在调用 `register_component` 之前设置 `COMPONENT_ADD_LDFRAGMENTS` 变量的值，使其指向链接片段文件。

```
# 文件路径相对于组件的 CMakeLists.txt
set (COMPONENT_ADD_LDFRAGMENTS "path/to/linker_fragment_file.lf" "path/to/another_
↳linker_fragment_file.lf")

register_component ()
```

也可以使用函数 `ldgen_add_fragment_files` 在项目的 `CMakeLists.txt` 文件或者组件的 `project_include.cmake` 文件中指定该片段文件

```
ldgen_add_fragment_files(target files ...)
```

### 指定放置区域

链接脚本生成机制允许指定以下条目的存放位置：

- 组件中的一个或多个目标文件
- 一个或多个函数/变量（使用它们的名字来指定）
- 整个组件库

在继续讲解之前，假设我们的组件包含以下内容：

- 一个名为 `component` 的组件，在构建期间被归档为 `libcomponent.a` 库文件
- 该库中有三个存档的目标文件：`object1.o`、`object2.o` 和 `object3.o`
- `object1.o` 中定义了 `function1` 函数，`object2.o` 中定义了 `function2` 函数
- 在其中的一个 IDF `KConfig` 文件中存在 `PERFORMANCE_MODE` 和 `PERFORMANCE_LEVEL` 两个配置，相应地，项目的 `sdkconfig` 文件会通过 `CONFIG_PERFORMANCE_MODE` 和 `CONFIG_PERFORMANCE_LEVEL` 这两个宏来指示当前设置的值

在新建的链接片段文件中输入以下内容：

```
[mapping]
archive: libcomponent.a
entries:
```

这会创建一个空的 *mapping* 片段，它并不会执行任何操作。在链接期间，会使用默认的存放规则来映射 `libcomponent.a`，除非填充了 `entries` 字段。

**放置目标文件** 假设整个 `object1.o` 目标文件对性能至关重要，所以最好把它放在 RAM 中。另一方面，假设“`object2.o`”目标文件包含有从深度睡眠唤醒所需的数据，因此需要将它存放到 RTC 内存中。可以在链接片段文件中写入以下内容：

```
[mapping]
archive: libcomponent.a
entries:
    object1 (noflash)      # 将所有代码和只读数据放置在 IRAM 和 DRAM 中
    object2 (rtc)         # 将所有代码、数据和只读数据放置到 RTC 快速内存和 RTC 慢速内存中
```

那么 `object3.o` 放在哪里呢？由于未指定放置规则，它会被存放到默认区域。

**放置函数和数据** 假设在 `object1.o` 目标文件中只有 `function1` 是与性能密切相关，且在 `object2.o` 目标文件中只有 `function2` 需要在深度睡眠唤醒后执行。可以在链接片段文件中写入以下内容：

```
[mapping]
archive: libcomponent.a
entries:
    object1:function1 (noflash)
    object2:function2 (rtc)
```

object1.o 和 object2.o 的剩余函数以及整个 object3.o 目标文件会被存放到默认区域。指定数据存放区域的方法很类似，仅需将：之后的函数名，替换为变量名即可。

**警告：** 使用符号名来指定放置区域有一定的**局限**。因此，您也可以将相关代码和数据集中在源文件中，然后根据**使用目标文件的放置规则**进行放置。

**放置整个组件** 在这个例子中，假设我们需要将整个组件存放到 RAM 中，可以这样写：

```
[mapping]
archive: libcomponent.a
entries:
    * (noflash)
```

类似的，下面的写法可以将整个组件存放到 RTC 内存中：

```
[mapping]
archive: libcomponent.a
entries:
    * (rtc)
```

**依赖于具体配置的存放方式** 假设只有当 sdkconfig 文件中存在 CONFIG\_PERFORMANCE\_MODE == y 时，整个组件才会被放置到指定区域，可以这样写：

```
[mapping]
archive: libcomponent.a
entries:
    : PERFORMANCE_MODE = y
    * (noflash)
```

其含义可以通过如下伪代码来表述：

```
if PERFORMANCE_MODE = y
    place entire libcomponent.a in RAM
else
    use default placements
```

此外，您还可以设置多个判断条件。假设有如下需求：当 CONFIG\_PERFORMANCE\_LEVEL == 1 时，只有 object1.o 存放到 RAM 中；当 CONFIG\_PERFORMANCE\_LEVEL == 2 时，object1.o 和 object2.o 会被存放到 RAM 中；当 CONFIG\_PERFORMANCE\_LEVEL == 3 时，归档中的所有目标文件都会被存放到 RAM 中；当这三个条件都不满足时，将整个组件库存放到 RTC 内存中。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]
archive: libcomponent.a
entries:
    : PERFORMANCE_LEVEL = 3
    * (noflash)
    : PERFORMANCE_LEVEL = 2
    object1 (noflash)
    object2 (noflash)
    : PERFORMANCE_LEVEL = 1
    object1 (noflash)
    : default
    * (rtc)
```

用伪代码可以表述为：

```

if CONFIG_PERFORMANCE_LEVEL == 3
    place entire libcomponent.a in RAM
else if CONFIG_PERFORMANCE_LEVEL == 2
    only place object1.o and object2.o in RAM
else if CONFIG_PERFORMANCE_LEVEL == 1
    only place object1.o in RAM
else
    place entire libcomponent.a in RTC memory

```

条件测试还支持[其他操作](#)。

### 默认的存放规则

到目前为止，“默认存放规则”一直作为未指定 `rtc` 和 `noflash` 存放规则时的备选放置方式。`noflash` 或者 `rtc` 标记不仅仅是链接脚本生成机制中的关键字，实际上还是由用户指定且被称为 *scheme* 片段的对象。由于这些存放规则非常常用，所以 IDF 中已经预定义了这些规则。

类似地，还有一个名为 `default` 的 *scheme* 片段，它定义了默认的存放规则，详情请见[默认 \*scheme\*](#)。

---

**注解：**有关使用此功能的 IDF 组件的示例，请参阅 [freertos/CMakeLists.txt](#)。为了提高性能，`freertos` 组件通过该机制将所有目标文件中的代码、字面量和只读数据存放到 `IRAM` 中。

---

快速入门指南到此结束，下面的文章将进一步详细讨论这个机制，例如它的组件、基本概念、语法、如何集成到构建系统中等等。以下部分有助于创建自定义的映射或者修改默认行为。

## 4.29.3 组件

### 链接片段文件

“链接片段文件”包含称为“片段”的对象，每个片段含有多条信息，放在一起时即可形成寻访规则，共同描述目标文件各个段在二进制输出文件中的存放位置。

换言之，处理“链接片段文件”也就是在 GNU LD 的 `SECTIONS` 命令中，创建段的存放规则，并将其放在一个内部 `target token` 中。

下面讨论三种类型的片段。

---

**注解：**片段具有名称属性（`mapping` 片段除外）并且是全局可见的。片段的命名遵循 C 语言的基本变量命名规则，即区分大小写；必须以字母或者下划线开头；允许非初始字符使用字母、数字和下划线；不能使用空格等特殊字符。此外，每种片段都有自己的独立命名空间，如果多个片的类型和名称相同，就会引发异常。

---

**I. sections 片段** `sections` 片段定义了 GCC 编译器输出的目标文件段的列表，可以是默认的段（比如 `.text` 段、`.data` 段），也可以是用户通过 `__attribute__` 关键字自定义的段。

此外，用户还可以在某一类段后增加一个 `+`，表示囊括列表中的“所有这类段”和“所有以这类段开头的段”。相较于显式地罗列所有的段，我们更推荐使用这种方式。

### 语法

```

[sections:name]
entries:
    .section+
    .section
    ...

```

## 示例

```
# 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.*
```

**II. scheme 片段** scheme 片段定义了为每个 sections 指定的 target。

## 语法

```
[scheme:name]
entries:
    sections -> target
    sections -> target
    ...
```

## 示例

```
[scheme:noflash]
entries:
    text -> iram0_text           # 名为 text 的 sections 片段下的所有条目均归入 iram0_
↔text
    rodata -> dram0_data       # 名为 rodata 的 sections 片段下的所有条目均归入 dram0_
↔data
```

## default scheme

注意, 有一个名为 default 的 scheme 很特殊, 特殊在于 catch-all 存放规则都是从这个 scheme 中的条目生成的。这意味着, 如果该 scheme 有一条 text -> flash\_text 条目, 则将为目标 flash\_text 生成如下的存放规则:

```
*(.literal .literal.* .text .text.*)
```

此后, 这些生成的 catch-all 规则将用于未指定映射规则的情况。

**注解:** default scheme 是在 [esp32/ld/esp32\\_fragments.lf](#) 文件中定义的, 此外, 快速上手指南中提到的内置 noflash scheme 片段和 rtc scheme 片段也是在这个文件中定义的。

**III. mapping 片段** mapping 片段定义了可映射实体 (即目标文件、函数名、变量名) 对应的 scheme 片段。具体来说, mapping 片段有两种类型的条目, 分别为映射条目和条件条目。

**注解:** mapping 片段没有具体的名称属性, 内部会根据归档条目的值构造其名称。

## 语法

```
[mapping]
archive: archive           # 构建后输出的存档文件的名称 (即 libxxx.a)
```

(下页继续)

```

entries:
  : condition                # 条件条目, 非默认
  object:symbol (scheme)    # 映射条目, Type I
  object (scheme)          # 映射条目, Type II
  * (scheme)                # 映射条目, Type III

# 为了提高可读性, 可以适当增加分隔行或注释, 非必须

: default                  # 条件条目, 默认
* (scheme)                 # 映射条目, Type III

```

## 映射条目

mapping 片段的映射条目共有三种类型, 分别为:

**Type I** 同时指定了目标文件名和符号名。其中, 符号名可以是函数名或者变量名。

**Type II** 仅指定了目标文件名。

**Type III** 指定了 \*, 也就是指定了归档文件中所有目标文件。

接下来, 让我们通过展开一个 Type II 映射条目, 更好地理解映射条目的含义。最初:

```
object (scheme)
```

接着, 让我们根据条目定义, 将这个 scheme 片段展开:

```
object (sections -> target,
        sections -> target,
        ...)
```

然后再根据条目定义, 将这个 sections 片段展开:

```
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

```

## 有关 Type I 映射条目的局限性

Type I 映射条目可以工作的大前提是编译器必须支持 `-ffunction-sections` 和 `-ffdata-sections` 选项。因此, 如果用户主动禁用了这两个选项, Type I 映射条目就无法工作。此外, 值得注意的是, Type I 映射条目的实现还与输出段有关。因此, 有时及时用户在编译时没有选择禁用这两个选项, 也有可能无法使用 Type I 映射条目。

例如, 当使用 `-ffunction-sections` 选项时, 编译器会给每个函数都输出一个单独的段, 根据段名的构造规则, 这些段的名称应该类似 `.text.{func_name}` 或 `.literal.{func_name}`。然而, 对于函数中的字符串文字, 情况并非如此, 因为它们会使用池化后或者新创建的段名。

当使用 `-fdata-sections` 选项时, 编译器会给每一个全局可见的数据输出一个单独的段, 名字类似于 `.data.{var_name}`、`.rodata.{var_name}` 或者 `.bss.{var_name}`。这种情况下, Type I 映射条目可以使用。然而, 对于在函数作用域中声明的静态数据, 编译器在为其生成段名时会同时使用其变量名和其他信息, 因此当涉及在函数作用域中定义的静态数据时就会出现这个问题。

## 条件条目

条件条目允许根据具体项目配置生成不同的链接脚本。也就是说, 可以根据一些配置表达式的值, 选择使用一套不同的映射条目。由于检查配置的过程是通过 `tools/kconfig_new/kconfiglib.py` 文件中的 `eval_string` 完成的, 因此条件表达式也必须遵循 `eval_string` 的语法和限制。

在一个 `mapping` 片段中，跟着一个条件条目后定义的所有映射条目均属于该条件条目，直至下一个条件条目的出现或者是该 `mapping` 片段的结束。在检查配置时，编译器将逐条检查这个 `mapping` 片段中的所有条件条目，直至找到一个满足条件的条件条目（即表达式为 `TRUE`），然后使用该条件条目下定义的映射条目。另外，尽管每个映射都已包含一个隐式的空映射，但用户还是可以自定义一个默认条件，即所有条件条目均不满足时（即没有表达式为 `TRUE`）使用的映射条目。

### 示例

```
[scheme:noflash]
entries:
    text -> iram0_text
    rodata -> dram0_data

[mapping:lwip]
archive: liblwip.a
entries:
    : LWIP_IRAM_OPTIMIZATION = y           # 如果 CONFIG_LWIP_IRAM_OPTIMIZATION 在_
    ↪sdkconfig 中被定义为 'y'
    ip4:ip4_route_src_hook (noflash)      # 将 ip4.o:ip4_route_src_hook, ip4.o:ip4_
    ↪route_src 和
    ip4:ip4_route_src (noflash)           # ip4.o:ip4_route 映射到 noflash scheme
    ip4:ip4_route (noflash)               # 该 scheme 会将他们存放到 RAM 中

    : default                             # 否则不使用特殊的映射规则
```

### 链接脚本模板

链接脚本模板与其他链接脚本没有本质区别，但带有特定的标记语法，可以指示放置生成的存放规则的位置，是指定存放规则的放置位置的框架。

### 语法

如需引用一个 `target token` 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脚本模板的摘录。该链接脚定义了一个输出段 `.iram0.text`，里面包含一个引用目标 `iram0_text` 的标记。

```
.iram0.text :
{
    /* 标记 IRAM 的边界 */
    _iram_text_start = ABSOLUTE(.);

    /* 引用 iram0_text */
    mapping[iram0_text]

    _iram_text_end = ABSOLUTE(.);
} > iram0_0_seg
```

下面，让我们更具体一点。假设某个链接脚本生成器收集到了以下片段：

```
[sections:text]
    .text+
    .literal+

[sections:iram]
    .iram1+

[scheme:default]
```

(下页继续)



```

entries:
  text -> flash_text
  iram -> iram0_text

[scheme:noflash]
entries:
  text -> iram0_text

[mapping:freertos]
archive: libfreertos.a
entries:
  * (noflash)

```

则该脚本生成器生成的链接脚本文件，其摘录应如下所示：

```

.iram0.text :
{
  /* 标记 IRAM 的边界 */
  _iram_text_start = ABSOLUTE(.);

  /* 将链接片段处理生成的存放规则放置在模板标记的位置处 */
  *(.iram1 .iram1.*)
  *libfreertos.a:(.literal .text .literal.* .text.*)

  _iram_text_end = ABSOLUTE(.);
} > iram0_0_seg

```

```
*libfreertos.a:(.literal .text .literal.* .text.*)
```

这是从 `freertos mapping` 片段的 `* (noflash)` 条目中生成的规则。`libfreertos.a` 归档文件下的所有目标文件的 `text` 段会被收集到 `iram0_text` 目标下（假设采用 `noflash scheme`），并放在模板中被 `iram0_text` 标记的地方。

```
*(.iram1 .iram1.*)
```

这是从 `default scheme` 的 `iram -> iram0_text` 条目生成的规则，因为 `default scheme` 指定了一个 `iram -> iram0_text` 条目，因此生成的规则也将放在被 `iram0_text` 标记的地方。值得注意的是，由于该规则是从 `default scheme` 中生成的，因此在同一目标下收集的所有规则下排在第一位。

#### 4.29.4 与构建系统的集成

链接脚本是在应用程序的构建过程中生成的，此时尚未链接形成最终的二进制文件。实现该机制的工具位于 `$(IDF_PATH)/tools/ldgen` 目录下。

##### 链接脚本模板

目前使用的链接脚本模板是 [esp32/ld/esp32.project.ld.in](https://github.com/espressif/esp32/blob/master/ld/esp32.project.ld.in)，仅用于应用程序的构建，生成的链接脚本文件将放在同一组件的构建目录下。值得注意的是，修改此链接描述文件模板会触发应用程序的二进制文件的重新链接。

##### 链接片段文件

任何组件都可以将片段文件添加到构建系统中，方法有两种：设置 `COMPONENT_ADD_LDFRAGMENTS` 变量或者使用 `ldgen_add_fragment_files` 函数（仅限 `CMake`），具体可以参考[添加片段文件](#)小节中的介绍。值得注意的是，修改构建系统中的任何片段文件都会触发应用程序的二进制文件的重新链接。

## 4.30 工具

### 4.30.1 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 3.6 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`, `make`, `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

---

**注解:** Versions of ESP-IDF released 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:

```
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`:

```
docker run --rm -v $PWD:/project -w /project espressif/idf:v4.0 idf.py build
```

**注解:** At the time of writing, v4.0 release of ESP-IDF does not exist, yet, so the above command will not work. You can check the up-to-date list of available tags at <https://hub.docker.com/r/espressif/idf/tags>.

---

**Building a project with GNU Make** Same as for CMake, except that the build command is different:

```
docker run --rm -v $PWD:/project -w /project espressif/idf make defconfig all -j4
```

**注解:** If the `sdkconfig` file does not exist, the default behavior of GNU Make build system is to open the `menuconfig` UI. This may be not desired in automated build environments. To ensure that the `sdkconfig` file exists, `defconfig` target is added before `all`.

---

**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
```

**注解:** 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>).

---

## Chapter 5

# Libraries and Frameworks

### 5.1 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:

#### 5.1.1 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`.

#### 5.1.2 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.

#### 5.1.3 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.



## Chapter 6

# Contributions Guide

We welcome contributions to the esp-idf project!

### 6.1 How to Contribute

Contributions to esp-idf - fixing bugs, adding features, adding documentation - are welcome. We accept contributions via [Github Pull Requests](#).

### 6.2 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.
- Does any new code conform to the esp-idf [Style Guide](#)?
- 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.

### 6.3 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.

### 6.4 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.

## 6.5 Related Documents

### 6.5.1 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

**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.

```
void function1()
{
    do_one_thing();
    do_another_thing();
}
// INCORRECT, don't place empty line here
// place empty line here
void function2()
{
    // INCORRECT, don't use an empty line here
    int var = 0;
    while (var < SOME_CONSTANT) {
        do_stuff(&var);
    }
}
```

**Horizontal space** Always add single space after conditional and loop keywords:

```
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:

```
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;                                       // correct
++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:

```
gpio_matrix_in(PIN_CAM_D6,    I2S0I_DATA_IN14_IDX, false);
gpio_matrix_in(PIN_CAM_D7,    I2S0I_DATA_IN15_IDX, false);
gpio_matrix_in(PIN_CAM_HREF,  I2S0I_H_ENABLE_IDX,  false);
gpio_matrix_in(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:

```
// This is correct:
void function(int arg)
{
}

// NOT like this:
void function(int arg) {
}
```

- Within a function, place opening brace on the same line with conditional and loop statements:

```
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:



```
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:

```
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:

```
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](#)>. However because MSYS2 uses Unix-style line endings, it is often easier to configure your text editor to use LF (Unix style) endings when editing ESP-IDF source files.

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
```

## 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](#).

## Documenting code

Please see the guide here: [Documenting Code](#).

## 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 namespaced with a per-component or per-unit prefix, to avoid naming collisions. ie `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 (ie shortening `data` to `dat`), unless the resulting name would otherwise be very long.

## Structure

To be written.

## Language features

To be written.

## 6.5.2 Documenting Code

The purpose of this description is to provide quick summary on documentation style used in [espressif/esp-idf](#) repository and how to add new documentation.

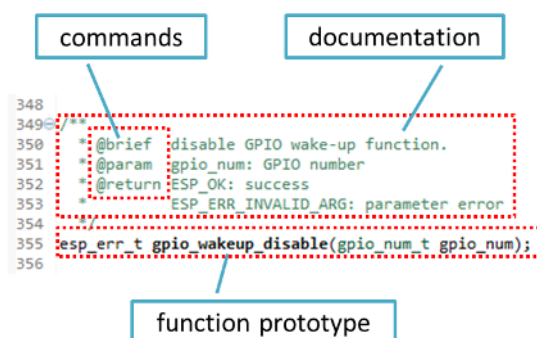
## Introduction

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:

```
/**
 * @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.

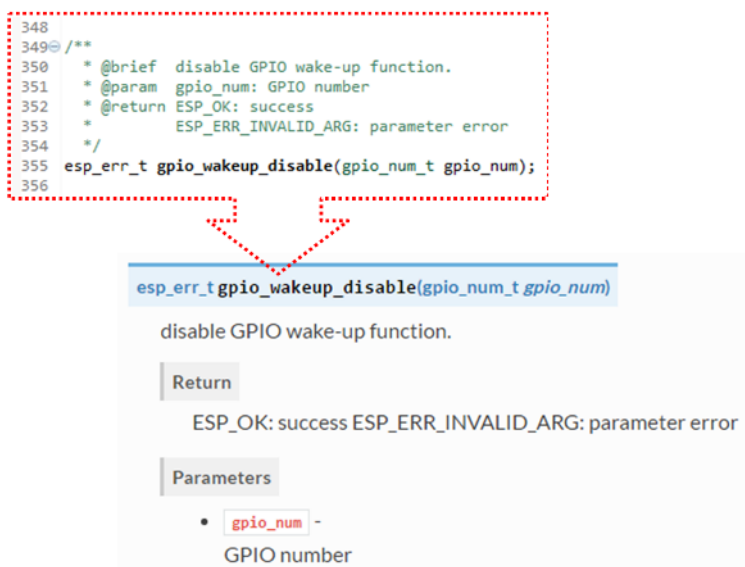


Doxygen supports couple of formatting styles. It also gives you great flexibility on level of details to include in documentation. To get familiar with available features, please check data rich and very well organized [Doxygen Manual](#).

## 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:



## 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 on 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.

```

41 @/**
42  * @brief Set log level for given tag
43  *
44  * If logging for given component has already been enabled, changes previous setting.
45  *
46  * @param tag Tag of the log entries to enable. Must be a non-NULL zero terminated string.
47  *           Value "" resets log level for all tags to the given value.
48  *
49  * @param level Selects log level to enable.
50  *             Only logs at this and lower levels will be shown.
51  */
52 void esp_log_level_set(const char* tag, esp_log_level_t level);

```

do not add data type

white spaces are compressed

a line break that will render

this line break will not render

```

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.

Parameters

- **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 levels will be shown.

4. If function has void input or does not return any value, then skip @param or @return

```

26 @/**
27  * @brief Initialize BT controller
28  *
29  * This function should be called only once,
30  * before any other BT functions are called.
31  */
32 void bt_controller_init(void);

```

do not add data type

```

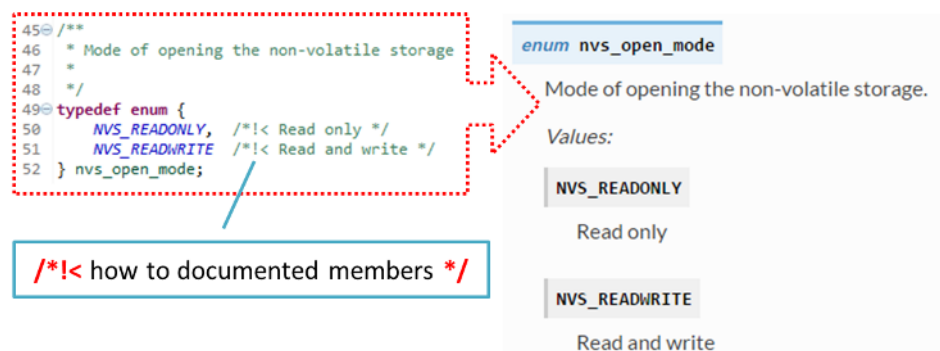
void bt_controller_init(void)

```

Initialize BT controller.

This function should be called only once, before any other BT functions are called.

5. When documenting a define as well as members of a struct or enum, place specific comment like below after each member.



- 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
*

```

- Overview of functionality of documented header file, or group of files that make a library, should be placed in the same directory in a separate README.rst file. If directory contains header files for different APIs, then the file name should be apiname-readme.rst.

### Go one extra mile

There is couple of tips, how you can make your documentation even better and more useful to the reader.

- 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.
* @endcode
*

```

The code snippet should be enclosed in a comment block of the function that it illustrates.

- 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.

- To provide common description to a group of similar functions, enclose them using /\*\*@{ \*/ and /\*\*@} \*/ markup commands:

```

/**@{ */
/**
* @brief common description of similar functions

```

(下页继续)

```

*
*/
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 e.g. 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.

```

*
* [ESP32 Technical Reference] (https://espressif.com/sites/default/files/→documentation/esp32\_technical\_reference\_manual\_en.pdf)
*

```

**注解:** Code snippets, notes, links, etc. will not make it to the documentation, if not enclosed in a comment block associated with one of documented objects.

6. Prepare one or more complete code examples together with description. Place description in a separate file `README.md` in specific folder of [examples](#) directory.

## Linking Examples

When linking to examples on GitHub do not use absolute / hardcoded URLs. Instead, use docutils custom roles that will generate links for you. These auto-generated links point to the tree or blob for the git commit ID (or tag) of the repository. This is needed to ensure that links do not get broken when files in master branch are moved around or deleted.

The following roles are provided:

- `:idf:`path`` - points to directory inside ESP-IDF
- `:idf_file:`path`` - points to file inside ESP-IDF
- `:idf_raw:`path`` - points to raw view of the file inside ESP-IDF
- `:component:`path`` - points to directory inside ESP-IDF components dir
- `:component_file:`path`` - points to file inside ESP-IDF components dir
- `:component_raw:`path`` - points to raw view of the file inside ESP-IDF components dir
- `:example:`path`` - points to directory inside ESP-IDF examples dir
- `:example_file:`path`` - points to file inside ESP-IDF examples dir
- `:example_raw:`path`` - points to raw view of the file inside ESP-IDF examples dir

Example implementation:

```

* :example:`get-started/hello_world`
* :example:`Hello World! <get-started/hello_world>`

```

How it renders:

- [get-started/hello\\_world](#)
- [Hello World!](#)

A check is added to the CI build script, which searches RST files for presence of hard-coded links (identified by `tree/master`, `blob/master`, or `raw/master` part of the URL). This check can be run manually: `cd docs` and then `make gh-linkcheck`.

## Linking Language Versions

Switching between documentation in different languages may be done using `:link_to_translation: custom` role. The role placed on a page of documentation provides a link to the same page in a language specified as a parameter. Examples below show how to enter links to Chinese and English versions of documentation:

```
:link_to_translation:`zh_CN: 中文版`  
:link_to_translation:`en:English`
```

The language is specified using standard abbreviations like `en` or `zh_CN`. The text after last semicolon is not standardized and may be entered depending on the context where the link is placed, e.g.:

```
:link_to_translation:`en:see description in English`
```

## Add Illustrations

Consider adding diagrams and pictures to illustrate described concepts.

Sometimes it is better to add an illustration than writing a lengthy paragraph to describe a complex idea, a data structure or an algorithm. This repository is using [blockdiag](#) suite of tools to generate diagram images from simple text files.

The following types of diagrams are supported:

- [Block diagram](#)
- [Sequence diagram](#)
- [Activity diagram](#)
- [Logical network diagram](#)

With this suite of tools it is possible to generate beautiful diagram images from simple text format (similar to `graphviz`'s DOT format). The diagram elements are laid out automatically. The diagram code is then converted into “.png” graphics and integrated “behind the scenes” into **Sphinx** documents.

For the diagram preparation you can use an on-line [interactive shell](#) that instantly shows the rendered image.

Below are couple of diagram examples:

- Simple **block diagram** / `blockdiag` - [Wi-Fi Buffer Configuration](#)
- Slightly more complicated **block diagram** - [Wi-Fi programming model](#)
- **Sequence diagram** / `seqdiag` - [Scan for a Specific AP in All Channels](#)
- **Packet diagram** / `packetdiag` - [NVS Page Structure](#)

Try them out by modifying the source code and see the diagram instantly rendering below.

---

**注解:** There may be slight differences in rendering of font used by the [interactive shell](#) compared to the font used in the `esp-idf` documentation.

---

## Add Notes

Working on a document, you might need to:

- Place some suggestions on what should be added or modified in future.
- Leave a reminder for yourself or somebody else to follow up.

In this case, add a `todo` note to your reST file using the directive `.. todo::`. For example:

```
.. todo::  
  
    Add a package diagram.
```

If you add `.. todoclist::` to a reST file, the directive will be replaced by a list of all todo notes from the whole documentation.

By default, the directives `.. todo::` and `.. todoclist::` are ignored by documentation builders. If you want the notes and the list of notes to be visible in your locally built documentation, do the following:

1. Open your local `conf_common.py` file.
2. Find the parameter `todo_include_todos`.
3. Change its value from `False` to `True`.

Before pushing your changes to origin, please set the value of `todo_include_todos` back to `False`.

For more details about the extension, see [sphinx.ext.todo](#) documentation.

### Put it all together

Once documentation is ready, follow instruction in [API Documentation Template](#) and create a single file, that will merge all individual pieces of prepared documentation. Finally add a link to this file to respective `.. toctree::` in `index.rst` file located in `/docs` folder or subfolders.

### OK, but I am new to Sphinx!

1. No worries. All the software you need is well documented. It is also open source and free. Start by checking [Sphinx](#) documentation. If you are not clear how to write using rst markup language, see [reStructuredText Primer](#). You can also use markdown (.md) files, and find out about more about the specific markdown syntax that we use on 'Recommonmark parser' s documentation page <<https://recommonmark.readthedocs.io/en/latest/>>.
2. Check the source files of this documentation to understand what is behind of what you see now on the screen. Sources are maintained on GitHub in [espressif/esp-idf](#) repository in `docs` folder. You can go directly to the source file of this page by scrolling up and clicking the link in the top right corner. When on GitHub, see what s really inside, open source files by clicking `Raw` button.
3. You will likely want to see how documentation builds and looks like before posting it on the GitHub. There are two options to do so:
  - Install [Sphinx](#), [Breathe](#), [Blockdiag](#) and [Doxygen](#) to build it locally, see chapter below.
  - Set up an account on [Read the Docs](#) and build documentation in the cloud. Read the Docs provides document building and hosting for free and their service works really quick and great.
4. To preview documentation before building, use [Sublime Text](#) editor together with [OmniMarkupPreviewer](#) plugin.

### Setup for building documentation locally

**Install Dependencies** You can setup environment to build documentation locally on your PC by installing:

1. Doxygen - <https://www.stack.nl/~dimitri/doxygen/>
2. Sphinx - <https://github.com/sphinx-doc/sphinx/#readme-for-sphinx>
3. Breathe - <https://github.com/michaeljones/breathe#breathe>
4. Document theme "sphinx\_rtd\_theme" - [https://github.com/rtfd/sphinx\\_rtd\\_theme](https://github.com/rtfd/sphinx_rtd_theme)
5. Custom 404 page "sphinx-notfound-page" - <https://github.com/rtfd/sphinx-notfound-page>
6. Blockdiag - <http://blockdiag.com/en/index.html>
7. Recommonmark - <https://github.com/rtfd/recommonmark>

The package "sphinx\_rtd\_theme" is added to have the same "look and feel" of [ESP32 Programming Guide](#) documentation like on the "Read the Docs" hosting site.

Do not worry about being confronted with several packages to install. Besides Doxygen, all remaining packages are written in Python. Therefore installation of all of them is combined into one simple step.

Installation of Doxygen is OS dependent:

#### Linux



```
sudo apt-get install doxygen
```

**Windows** - install in MSYS2 console

```
pacman -S doxygen
```

**MacOS**

```
brew install doxygen
```

---

**注解:** If you are installing on Windows MSYS2 system (Linux and MacOS users should skip this note, Windows users who don't use MSYS2 will need to find other alternatives), **before** going further, execute two extra steps below. These steps are required to install dependencies of “blockdiag” discussed under [Add Illustrations](#).

1. Update all the system packages:

```
$ pacman -Syu
```

This process will likely require restarting of the MSYS2 MINGW32 console and repeating above commands, until update is complete.

2. Install *pillow*, that is one of dependences of the *blockdiag*:

```
$ pacman -S mingw32/mingw-w64-i686-python2-pillow
```

Check the log on the screen that `mingw-w64-i686-python2-pillow-4.3.0-1` is installed. Previous versions of *pillow* will not work.

A downside of Windows installation is that fonts of the *blockdiag pictures* <[add-illustrations](#)> do not render correctly, you will see some random characters instead. Until this issue is fixed, you can use the [interactive shell](#) to see how the complete picture looks like.

---

All remaining applications are [Python](#) packages and you can install them in one step as follows:

```
cd ~/esp/esp-idf/docs
pip install --user -r requirements.txt
```

---

**注解:** Installation steps assume that ESP-IDF is placed in `~/esp/esp-idf` directory, that is default location of ESP-IDF used in documentation.

---

Change to directory with files for specific language:

```
cd en
```

Now you should be ready to build documentation by invoking:

```
make html
```

This may take couple of minutes. After completion, documentation will be placed in `~/esp/esp-idf/docs/en/_build/html` folder. To see it, open `index.html` in a web browser.

## 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](#)
- [Documentation Add-ons and Extensions Reference](#)

### 6.5.3 Documentation Add-ons and Extensions Reference

This documentation is created using [Sphinx](#) application that renders text source files in [reStructuredText](#) (`.rst`) format located in `docs` directory. For some more details on that process, please refer to section [Documenting Code](#).

Besides Sphinx there are several other applications that help to provide nicely formatted and easy to navigate documentation. These applications are listed in section [Setup for building documentation locally](#) with the installed version numbers provided in file `docs/requirements.txt`.

On top of that we have created a couple of custom add-ons and extensions to help integrate documentation with underlining [ESP-IDF](#) repository and further improve navigation as well as maintenance of documentation.

The purpose of this section is to provide a quick reference to the add-ons and the extensions.

#### Documentation Folder Structure

- The ESP-IDF repository contains a dedicated documentation folder `docs` in the root.
- The `docs` folder contains localized documentation in `docs/en` (English) and `docs/zh_CN` (simplified Chinese) subfolders.
- Graphics files and fonts common to localized documentation are contained in `docs/_static` subfolder
- Remaining files in the root of `docs` as well as `docs/en` and `docs/zh_CN` provide configuration and scripts used to automate documentation processing including the add-ons and extensions.
- Several folders and files are generated dynamically during documentations build and placed primarily in `docs/[lang]/_build` folders. These folders are temporary and not visible in [ESP-IDF](#) repository,

#### Add-ons and Extensions Reference

**`docs/conf_common.py`** This file contains configuration common to each localized documentation (e.g. English, Chinese). The contents of this file is imported to standard Sphinx configuration file `conf.py` located in respective language folders (e.g. `docs/en`, `docs/zh_CN`) during build for each language.

**`docs/check_doc_warnings.sh`** If there are any warnings reported during documentation build, then the build is failed. The warnings should be resolved before merging any documentation updates. This script is doing check for warnings in respective log file to fail the build. See also description of `sphinx-known-warnings.txt` below.

**`docs/check_lang_folder_sync.sh`** To reduce potential discrepancies when maintaining concurrent language version, the structure and filenames of language folders `docs/en` and `docs/zh_CN` folders should be kept identical. The script `check_lang_folder_sync.sh` is run on each documentation build to verify if this condition is met.

---

**注解:** If a new content is provided in e.g. English, and there is no any translation yet, then the corresponding file in `zh_CN` folder should contain an `.. include::` directive pointing to the source file in English. This will automatically include the English version visible to Chinese readers. For example if a file `docs/zh_CN/contribute/documenting-code.rst` does not have a Chinese translation, then it should contain `.. include:: ../../en/contribute/documenting-code.rst` instead.

---

**`docs/docs_common.mk`** It contains the common code which is included into the language-specific `Makefiles`. Note that this file contains couple of customizations comparing to what is provided within standard Sphinx installation, e.g. `gh-linkcheck` command has been added.

**`docs/gen-dxd.py`** A Python script that generates API reference files based on Doxygen `xml` output. The files have an `inc` extension and are located in `docs/[lang]/_build/inc` directory created dynamically when documentation is build. Please refer to [Documenting Code](#) and [API Documentation Template](#), section **API Reference** for additional details on this process.

**docs/gen-toolchain-links.py** There couple of places in documentation that provide links to download the toolchain. To provide one source of this information and reduce effort to manually update several files, this script generates toolchain download links and toolchain unpacking code snippets based on information found in `tools/toolchain_versions.mk`.

**docs/gen-version-specific-includes.py** Another Python script to automatically generate reStructuredText Text `.inc` snippets with version-based content for this ESP-IDF version.

**docs/html\_redirects.py** During documentation lifetime some source files are moved between folders or renamed. This Python script is adding a mechanism to redirect documentation pages that have changed URL by generating in the Sphinx output static HTML redirect pages. The script is used together with a redirection list `html_redirect_pages` defined in file `docs/conf_common.py`.

**docs/link-roles.py** This is an implementation of a custom [Sphinx Roles](#) to help linking from documentation to specific files and folders in ESP-IDF. For description of implemented roles please see [Linking Examples](#) and [Linking Language Versions](#).

**docs/local\_util.py** A collection of utility functions useful primarily when building documentation locally (see [Setup for building documentation locally](#)) to reduce the time to generate documentation on a second and subsequent builds. The utility functions check what Doxygen `xml` input files have been changed and copy these files to destination folders, so only the changed files are used during build process.

**docs/sphinx-known-warnings.txt** There are couple of spurious Sphinx warnings that cannot be resolved without doing update to the Sphinx source code itself. For such specific cases respective warnings are documented in `sphinx-known-warnings.txt` file, that is checked during documentation build, to ignore the spurious warnings.

**tools/gen\_esp\_err\_to\_name.py** This script is traversing the ESP-IDF directory structure looking for error codes and messages in source code header files to generate an `.inc` file to include in documentation under [Error Codes Reference](#).

**tools/kconfig\_new/confgen.py** Options to configure ESP-IDF's [components](#) are contained in `Kconfig` files located inside directories of individual components, e.g. `components/bt/Kconfig`. This script is traversing the `component` directories to collect configuration options and generate an `.inc` file to include in documentation under [Configuration Options Reference](#).

## Related Documents

- [Documenting Code](#)

## 6.5.4 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要功能。
- 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 Makefile 和 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令要遵循[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，或者定义为宏或常量。
- 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。

## 6.5.5 API Documentation Template

---

注解：INSTRUCTIONS

1. Use this file ([docs/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

---

注解：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

### 注解: 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

### 注解: 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 script `docs/gen-dxd.py` for all header files listed in the `INPUT` statement of `docs/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. 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`.
5. To show contents of `*.inc` file in documentation, include it as follows:

```
.. include:: /_build/inc/esp_wifi.inc
```

For example see [docs/en/api-reference/wifi/esp\\_wifi.rst](docs/en/api-reference/wifi/esp_wifi.rst)

6. Optionally, rather than using `*.inc` files, you may want to describe API in your own way. See <docs/en/api-guides/ulp.rst> for example.

Below is the list of common `.. doxygen...::` directives:

- Functions - `.. doxygenfunction:: name_of_function`
- Unions - `.. doxygenunion:: 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* as follows:

```
* :component_file:`path_to/header_file.h`
```

7. In any case, to generate API reference, the file `docs/Doxyfile` should be updated with paths to `*.h` headers that are being documented.
8. When changes are committed and documentation is built, check how this section has been rendered. *Correct annotations* in respective header files, if required.

## 6.5.6 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 at [CONTRIBUTING.rst](#)

**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 [angus@espressif.com](mailto:angus@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 (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:	

**Us**

Date:	
Name:	
Title:	
Address:	





## Chapter 7

# 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。

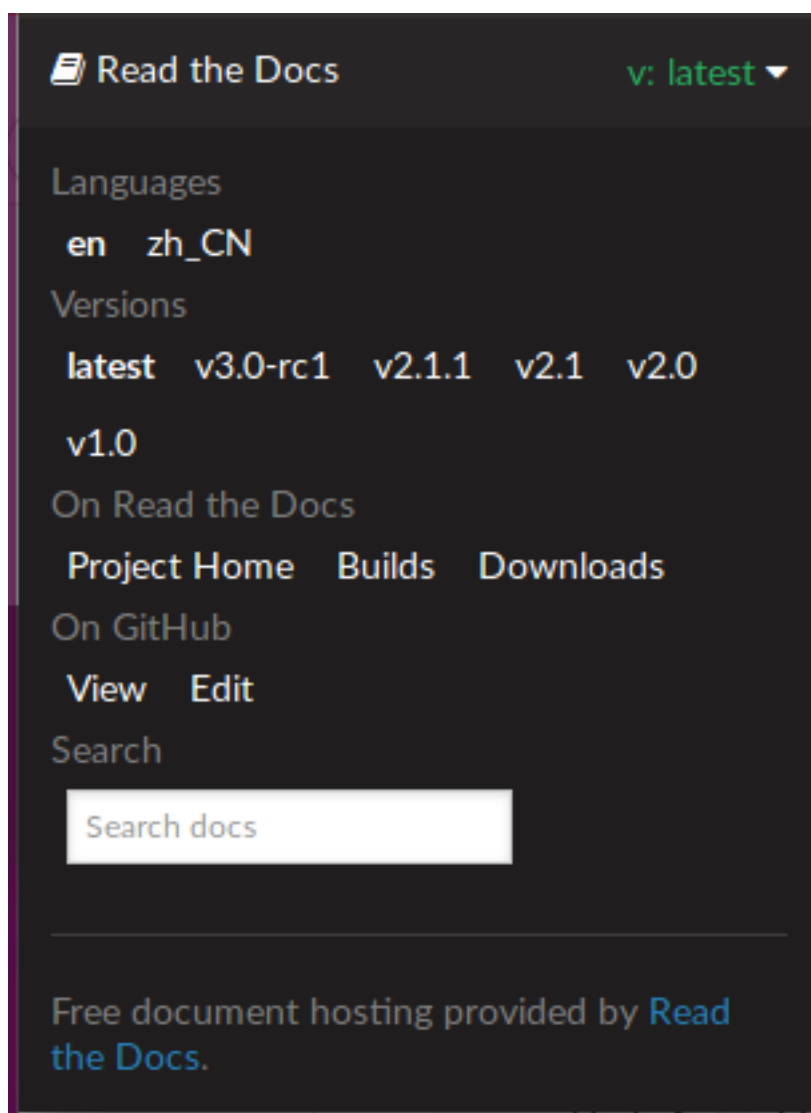
### 7.1 发布版本

您可以通过以下链接访问各个版本的配套文档：

- 最新稳定版 ESP-IDF： [https://docs.espressif.com/projects/esp-idf/zh\\_CN/stable/](https://docs.espressif.com/projects/esp-idf/zh_CN/stable/)
- 最新版 ESP-IDF（即 master 分支）： [https://docs.espressif.com/projects/esp-idf/zh\\_CN/latest/](https://docs.espressif.com/projects/esp-idf/zh_CN/latest/)

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。

此外，您还可以直接前往文档页面，查看部分 ESP-IDF 版本的配套文档，具体可通过点击页面左下角的小三角标志，在不同版本间切换。



## 7.2 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](#)）。
- 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 的所有最新特性，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](#)。

有关如何更新 ESP-IDF 本地副本的内容，请参考[更新 ESP-IDF](#) 章节。

## 7.3 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您的工程代码，并重新测试您的工程，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。  
升级至一个新的 **Bugfix** 版本（例 v3.0 升级至 v3.0.1）意味着您不需要更新您的工程代码，仅需测试与本次发布修复 **bug**（列表见 [发布说明页面](#)）直接相关的特性。

## 7.4 查看当前版本

查看 ESP-IDF 本地副本的版本，请使用 `git` 命令：

```
cd $IDF_PATH
git describe --dirty
```

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 默认引导程序可以在设备启动时打印 ESP-IDF 的版本，但注意代码中的版本信息仅会在源代码重新编译时才会更新，因此打印出来的版本可能并不是最新的。

几个 ESP-IDF 版本的例子：

版本字符串	含义
v3.2-dev-306-gbeb3611ca	<p><b>master</b> 分支上的预发布版本。</p> <ul style="list-style-type: none"> <li>- v3.2-dev: 为 v3.2 进行的开发。</li> <li>- 306: v3.2 开发启动后的 <b>commit</b> 数量。</li> <li>- beb3611ca: <b>commit</b> 标识符。</li> </ul>
v3.0.2	稳定版本，标签为 v3.0.2。
v3.1-beta1-75-g346d6b0ea	<p>v3.1 的 <b>beta</b> 测试版本（可参考<a href="#">更新至一个发布分支</a>）。</p> <ul style="list-style-type: none"> <li>- v3.1-beta1 - 预发布标签。</li> <li>- 75: 添加预发布 <b>beta</b> 标签后的 <b>commit</b> 数量。</li> <li>- 346d6b0ea: <b>commit</b> 标识符。</li> </ul>
v3.0.1-dirty	<p>稳定版本，标签为 v3.0.1。</p> <ul style="list-style-type: none"> <li>- <code>dirty</code> 代表 ESP-IDF 的本地副本有修改。</li> </ul>

## 7.5 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）上进行。**master** 分支上的 ESP-IDF 版本总带有 `-dev` 标签，表示“正在开发中”，例 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，则将该版本签出至一个新分支（例 `release/v3.1`）。此外，该分支还打上预发布标签（例 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。

- 当测试快结束时，该发布分支上将增加一个 rc 标签，代表候选发布 (Release Candidate)，例 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，该版本中发现的 bug 都将在该发布分支上进行修复。人工测试完成后，该分支将增加一个 bugfix 版本标签（例 v3.1.1），并体现在 [发布说明页面](#)。

## 7.6 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](#)。

---

**注解：**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。

---

### 7.6.1 更新至一个稳定发布版本

（推荐量产用户）如需更新至一个新的 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（例 v3.0.1 或 v3.0.2）时，请将新的 bugfix 版本更新至您的 ESP-IDF 目录：

```
cd $IDF_PATH
git fetch
git checkout vX.Y.Z
git submodule update --init --recursive
```

- 如有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。具体命令与上方描述一致。

---

**注解：**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。

---

### 7.6.2 更新至一个预发布版本

您也可以将您的本地副本签出（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。

### 7.6.3 更新至 master 分支

---

**注解：**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：

- 本地签出至 master 分支:

```
cd $IDF_PATH
git checkout master
git pull
git submodule update --init --recursive
```

- 此外，您还应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，您可能需要更改项代码，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，确保本地副本的及时更新。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](#)。

---

### 7.6.4 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](#)。

举例，您可以关注 ESP-IDF v3.1 分支，随时关注该分支上的 bugfix 版本发布（例 v3.1.1 等）：

```
cd $IDF_PATH
git fetch
git checkout release/v3.1
git pull
git submodule update --init --recursive
```

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。

---

**注解：**发布分支并不会会有专门的配套文档，建议您使用与本分支最接近的版本。

---



## Chapter 8

# 资源

- 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 [ESP32 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。
- 通过 [Arduino](#) 平台开发应用，请参考 [ESP32 Wi-Fi 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 9

## Copyrights and Licenses

### 9.1 Software Copyrights

All original source code in this repository is Copyright (C) 2015-2018 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.

#### 9.1.1 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) 2015 Real Time Engineers Ltd and is licensed under the GNU General Public License V2 with the FreeRTOS Linking Exception, 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, Errno Consulting and licensed under the BSD license.
- [JSMN](#) JSON Parser (components/jsmn) Copyright (c) 2010 Serge A. Zaitsev and licensed under the MIT license.
- [argtable3](#) argument parsing library Copyright (C) 1998-2001,2003-2011,2013 Stewart Heitmann and licensed under 3-clause BSD license.
- [linenoise](#) line editing library Copyright (c) 2010-2014 Salvatore Sanfilippo, Copyright (c) 2010-2013 Pieter Noordhuis, licensed under 2-clause BSD license.
- [libcoap](#) COAP library Copyright (c) 2010-2017 Olaf Bergmann and others, is licensed under 2-clause BSD license.
- [libexpat](#) XML parsing library Copyright (c) 1998-2000 Thai Open Source Software Center Ltd and Clark Cooper, Copyright (c) 2001-2017 Expat maintainers, is licensed under MIT 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.
- [libsodium](#) library, Copyright (c) 2013-2018 Frank Denis, is licensed under ISC license.
- [micro-ecc](#) library, Copyright (c) 2014 Kenneth MacKay, is licensed under 2-clause BSD license.
- [nghttp2](#) library, Copyright (c) 2012, 2014, 2015, 2016 Tatsuhiro Tsujikawa, Copyright (c) 2012, 2014, 2015, 2016 nghttp2 contributors, is licensed under MIT license.

- [Mbed TLS](#) library, Copyright (C) 2006-2018 ARM Limited, is licensed under Apache License 2.0.
- [SPIFFS](#) library, Copyright (c) 2013-2017 Peter Andersson, is licensed under MIT license.
- [SD/MMC driver](#) is derived from [OpenBSD SD/MMC driver](#), Copyright (c) 2006 Uwe Stuehler, and is licensed under BSD license.
- [Asio](#) , Copyright (c) 2003-2018 Christopher M. Kohlhoff is licensed under the Boost Software 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.
- [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.

### 9.1.2 Build Tools

This is the list of licenses for tools included in this repository, which are used to build applications. The tools do not become part of the application (firmware), so their license does not affect licensing of the application.

- [esptool.py](#) is Copyright (C) 2014-2016 Fredrik Ahlberg, Angus Gratton and is licensed under the GNU General Public License v2, as described in [LICENSE file](#).
- [KConfig](#) is Copyright (C) 2002 Roman Zippel and others, and is licensed under the GNU General Public License V2.

## 9.2 ROM Source Code Copyrights

ESP32 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.

### 9.3 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.

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.

## 9.4 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.

## 9.5 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 10

# 关于本指南

本指南为 乐鑫公司 ESP32 系列芯片 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 32 位处理器，运算能力最高可达 600 DMIPS。

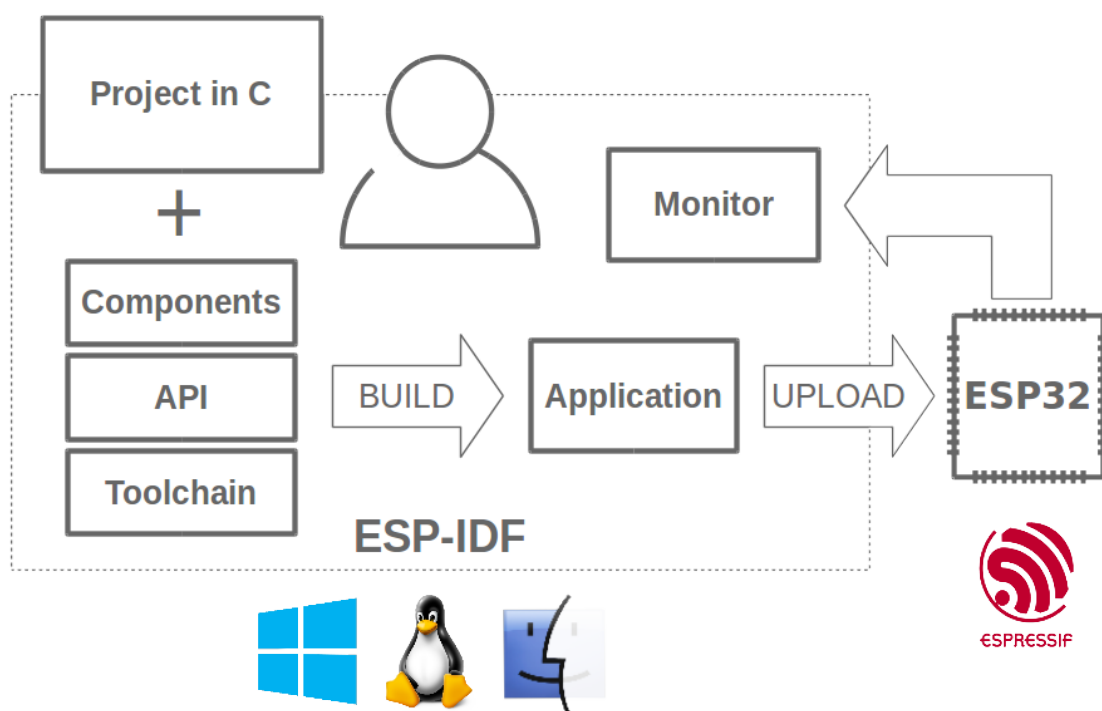


图 1: 乐鑫物联网综合开发框架

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 应用程序提供工具链、API、组件和工作流的支持。



## Chapter 11

# Switch Between Languages/切换语言

The ESP-IDF Programming Manual is now available in two languages. Please refer to the English version if there is any discrepancy.

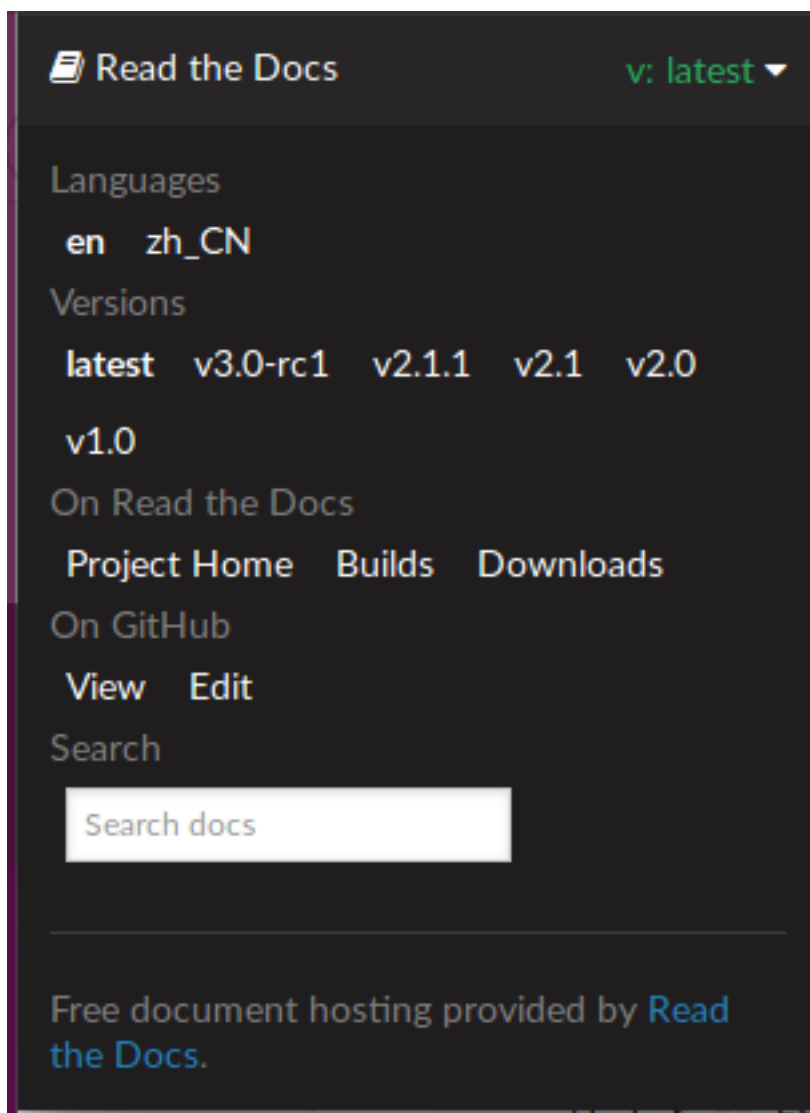
《ESP-IDF 编程手册》部分文档现在有两种语言的版本。如有出入请以英文版本为准。

- English/英文
- Chinese/中文

You can easily change from one language to another by the panel on the sidebar like below. Just click on the **Read the Docs** title button on the left-bottom corner if it is folded.

如下图所示，你可使用边栏的面板进行语言的切换。如果该面板被折叠，点击左下角 **Read the Docs** 标题按钮来显示它。





- [genindex](#)

# 索引

## 符号

[anonymous] (C++ *enum*), 106, 303

## A

- ADC1\_CHANNEL\_0 (C++ *enumerator*), 565
- ADC1\_CHANNEL\_0\_GPIO\_NUM (C 宏), 570
- ADC1\_CHANNEL\_1 (C++ *enumerator*), 566
- ADC1\_CHANNEL\_1\_GPIO\_NUM (C 宏), 570
- ADC1\_CHANNEL\_2 (C++ *enumerator*), 566
- ADC1\_CHANNEL\_2\_GPIO\_NUM (C 宏), 570
- ADC1\_CHANNEL\_3 (C++ *enumerator*), 566
- ADC1\_CHANNEL\_3\_GPIO\_NUM (C 宏), 570
- ADC1\_CHANNEL\_4 (C++ *enumerator*), 566
- ADC1\_CHANNEL\_4\_GPIO\_NUM (C 宏), 570
- ADC1\_CHANNEL\_5 (C++ *enumerator*), 566
- ADC1\_CHANNEL\_5\_GPIO\_NUM (C 宏), 570
- ADC1\_CHANNEL\_6 (C++ *enumerator*), 566
- ADC1\_CHANNEL\_6\_GPIO\_NUM (C 宏), 570
- ADC1\_CHANNEL\_7 (C++ *enumerator*), 566
- ADC1\_CHANNEL\_7\_GPIO\_NUM (C 宏), 570
- ADC1\_CHANNEL\_MAX (C++ *enumerator*), 566
- adc1\_channel\_t (C++ *enum*), 565
- adc1\_config\_channel\_atten (C++ *function*), 561
- adc1\_config\_width (C++ *function*), 561
- adc1\_get\_raw (C++ *function*), 562
- ADC1\_GPIO32\_CHANNEL (C 宏), 570
- ADC1\_GPIO33\_CHANNEL (C 宏), 570
- ADC1\_GPIO34\_CHANNEL (C 宏), 570
- ADC1\_GPIO35\_CHANNEL (C 宏), 570
- ADC1\_GPIO36\_CHANNEL (C 宏), 570
- ADC1\_GPIO37\_CHANNEL (C 宏), 570
- ADC1\_GPIO38\_CHANNEL (C 宏), 570
- ADC1\_GPIO39\_CHANNEL (C 宏), 570
- adc1\_pad\_get\_io\_num (C++ *function*), 561
- adc1\_ulp\_enable (C++ *function*), 563
- ADC2\_CHANNEL\_0 (C++ *enumerator*), 566
- ADC2\_CHANNEL\_0\_GPIO\_NUM (C 宏), 570
- ADC2\_CHANNEL\_1 (C++ *enumerator*), 566
- ADC2\_CHANNEL\_1\_GPIO\_NUM (C 宏), 570
- ADC2\_CHANNEL\_2 (C++ *enumerator*), 566
- ADC2\_CHANNEL\_2\_GPIO\_NUM (C 宏), 570
- ADC2\_CHANNEL\_3 (C++ *enumerator*), 566
- ADC2\_CHANNEL\_3\_GPIO\_NUM (C 宏), 570
- ADC2\_CHANNEL\_4 (C++ *enumerator*), 566
- ADC2\_CHANNEL\_4\_GPIO\_NUM (C 宏), 570
- ADC2\_CHANNEL\_5 (C++ *enumerator*), 566
- ADC2\_CHANNEL\_5\_GPIO\_NUM (C 宏), 570
- ADC2\_CHANNEL\_6 (C++ *enumerator*), 566
- ADC2\_CHANNEL\_6\_GPIO\_NUM (C 宏), 570
- ADC2\_CHANNEL\_7 (C++ *enumerator*), 566
- ADC2\_CHANNEL\_7\_GPIO\_NUM (C 宏), 570
- ADC2\_CHANNEL\_8 (C++ *enumerator*), 566
- ADC2\_CHANNEL\_8\_GPIO\_NUM (C 宏), 570
- ADC2\_CHANNEL\_9 (C++ *enumerator*), 566
- ADC2\_CHANNEL\_9\_GPIO\_NUM (C 宏), 570
- ADC2\_CHANNEL\_MAX (C++ *enumerator*), 566
- adc2\_channel\_t (C++ *enum*), 566
- adc2\_config\_channel\_atten (C++ *function*), 564
- adc2\_get\_raw (C++ *function*), 564
- ADC2\_GPIO0\_CHANNEL (C 宏), 570
- ADC2\_GPIO12\_CHANNEL (C 宏), 570
- ADC2\_GPIO13\_CHANNEL (C 宏), 570
- ADC2\_GPIO14\_CHANNEL (C 宏), 570
- ADC2\_GPIO15\_CHANNEL (C 宏), 570
- ADC2\_GPIO25\_CHANNEL (C 宏), 570
- ADC2\_GPIO26\_CHANNEL (C 宏), 570
- ADC2\_GPIO27\_CHANNEL (C 宏), 570
- ADC2\_GPIO2\_CHANNEL (C 宏), 570
- ADC2\_GPIO4\_CHANNEL (C 宏), 570
- adc2\_pad\_get\_io\_num (C++ *function*), 564
- adc2\_vref\_to\_gpio (C++ *function*), 564
- ADC\_ATTEN\_0db (C 宏), 565
- ADC\_ATTEN\_11db (C 宏), 565
- ADC\_ATTEN\_2\_5db (C 宏), 565
- ADC\_ATTEN\_6db (C 宏), 565
- ADC\_ATTEN\_DB\_0 (C++ *enumerator*), 565
- ADC\_ATTEN\_DB\_11 (C++ *enumerator*), 565
- ADC\_ATTEN\_DB\_2\_5 (C++ *enumerator*), 565
- ADC\_ATTEN\_DB\_6 (C++ *enumerator*), 565
- ADC\_ATTEN\_MAX (C++ *enumerator*), 565
- adc\_atten\_t (C++ *enum*), 565
- adc\_bits\_width\_t (C++ *enum*), 565
- ADC\_CHANNEL\_0 (C++ *enumerator*), 566
- ADC\_CHANNEL\_1 (C++ *enumerator*), 566
- ADC\_CHANNEL\_2 (C++ *enumerator*), 566
- ADC\_CHANNEL\_3 (C++ *enumerator*), 567
- ADC\_CHANNEL\_4 (C++ *enumerator*), 567
- ADC\_CHANNEL\_5 (C++ *enumerator*), 567
- ADC\_CHANNEL\_6 (C++ *enumerator*), 567
- ADC\_CHANNEL\_7 (C++ *enumerator*), 567
- ADC\_CHANNEL\_8 (C++ *enumerator*), 567
- ADC\_CHANNEL\_9 (C++ *enumerator*), 567

ADC\_CHANNEL\_MAX (C++ enumerator), 567  
 adc\_channel\_t (C++ enum), 566  
 ADC\_ENCODE\_11BIT (C++ enumerator), 567  
 ADC\_ENCODE\_12BIT (C++ enumerator), 567  
 ADC\_ENCODE\_MAX (C++ enumerator), 567  
 adc\_gpio\_init (C++ function), 563  
 ADC\_I2S\_DATA\_SRC\_ADC (C++ enumerator), 567  
 ADC\_I2S\_DATA\_SRC\_IO\_SIG (C++ enumerator), 567  
 ADC\_I2S\_DATA\_SRC\_MAX (C++ enumerator), 567  
 adc\_i2s\_encode\_t (C++ enum), 567  
 adc\_i2s\_mode\_init (C++ function), 563  
 adc\_i2s\_source\_t (C++ enum), 567  
 adc\_power\_acquire (C++ function), 562  
 adc\_power\_off (C++ function), 562  
 adc\_power\_on (C++ function), 562  
 adc\_power\_release (C++ function), 563  
 adc\_set\_clk\_div (C++ function), 563  
 adc\_set\_data\_inv (C++ function), 563  
 adc\_set\_data\_width (C++ function), 561  
 adc\_set\_i2s\_data\_source (C++ function), 563  
 ADC\_UNIT\_1 (C++ enumerator), 567  
 ADC\_UNIT\_2 (C++ enumerator), 567  
 ADC\_UNIT\_ALTER (C++ enumerator), 567  
 ADC\_UNIT\_BOTH (C++ enumerator), 567  
 ADC\_UNIT\_MAX (C++ enumerator), 567  
 adc\_unit\_t (C++ enum), 567  
 ADC\_WIDTH\_10Bit (C 宏), 565  
 ADC\_WIDTH\_11Bit (C 宏), 565  
 ADC\_WIDTH\_12Bit (C 宏), 565  
 ADC\_WIDTH\_9Bit (C 宏), 565  
 ADC\_WIDTH\_BIT\_10 (C++ enumerator), 565  
 ADC\_WIDTH\_BIT\_11 (C++ enumerator), 565  
 ADC\_WIDTH\_BIT\_12 (C++ enumerator), 565  
 ADC\_WIDTH\_BIT\_9 (C++ enumerator), 565  
 ADC\_WIDTH\_MAX (C++ enumerator), 565  
 ADD\_DEV\_FLUSHABLE\_DEV\_FLAG (C 宏), 285  
 ADD\_DEV\_RM\_AFTER\_PROV\_FLAG (C 宏), 285  
 ADD\_DEV\_START\_PROV\_NOW\_FLAG (C 宏), 285  
 ADV\_CHNL\_37 (C++ enumerator), 136  
 ADV\_CHNL\_38 (C++ enumerator), 136  
 ADV\_CHNL\_39 (C++ enumerator), 136  
 ADV\_CHNL\_ALL (C++ enumerator), 136  
 ADV\_FILTER\_ALLOW\_SCAN\_ANY\_CON\_ANY (C++ enumerator), 136  
 ADV\_FILTER\_ALLOW\_SCAN\_ANY\_CON\_WLST (C++ enumerator), 136  
 ADV\_FILTER\_ALLOW\_SCAN\_WLST\_CON\_ANY (C++ enumerator), 136  
 ADV\_FILTER\_ALLOW\_SCAN\_WLST\_CON\_WLST (C++ enumerator), 136  
 ADV\_TYPE\_DIRECT\_IND\_HIGH (C++ enumerator), 136  
 ADV\_TYPE\_DIRECT\_IND\_LOW (C++ enumerator), 136  
 ADV\_TYPE\_IND (C++ enumerator), 136  
 ADV\_TYPE\_NONCONN\_IND (C++ enumerator), 136  
 ADV\_TYPE\_SCAN\_IND (C++ enumerator), 136

## B

BD\_ADDR (C++ type), 296  
 BD\_ADDR\_LEN (C 宏), 284  
 BLE\_ADDR\_TYPE\_PUBLIC (C++ enumerator), 111  
 BLE\_ADDR\_TYPE\_RANDOM (C++ enumerator), 111  
 BLE\_ADDR\_TYPE\_RPA\_PUBLIC (C++ enumerator), 111  
 BLE\_ADDR\_TYPE\_RPA\_RANDOM (C++ enumerator), 111  
 BLE\_BIT (C 宏), 133  
 BLE\_HCI\_UART\_H4\_ACL (C 宏), 250  
 BLE\_HCI\_UART\_H4\_CMD (C 宏), 250  
 BLE\_HCI\_UART\_H4\_EVT (C 宏), 250  
 BLE\_HCI\_UART\_H4\_NONE (C 宏), 250  
 BLE\_HCI\_UART\_H4\_SCO (C 宏), 250  
 BLE\_SCAN\_DUPLICATE\_DISABLE (C++ enumerator), 137  
 BLE\_SCAN\_DUPLICATE\_ENABLE (C++ enumerator), 137  
 BLE\_SCAN\_DUPLICATE\_MAX (C++ enumerator), 138  
 BLE\_SCAN\_FILTER\_ALLOW\_ALL (C++ enumerator), 137  
 BLE\_SCAN\_FILTER\_ALLOW\_ONLY\_WLST (C++ enumerator), 137  
 BLE\_SCAN\_FILTER\_ALLOW\_UND\_RPA\_DIR (C++ enumerator), 137  
 BLE\_SCAN\_FILTER\_ALLOW\_WLIST\_PRA\_DIR (C++ enumerator), 137  
 BLE\_SCAN\_TYPE\_ACTIVE (C++ enumerator), 137  
 BLE\_SCAN\_TYPE\_PASSIVE (C++ enumerator), 137  
 BLE\_UUID128\_VAL\_LENGTH (C 宏), 845  
 BLE\_WL\_ADDR\_TYPE\_PUBLIC (C++ enumerator), 111  
 BLE\_WL\_ADDR\_TYPE\_RANDOM (C++ enumerator), 111  
 BT\_CONTROLLER\_INIT\_CONFIG\_DEFAULT (C 宏), 106  
 BT\_OCTET32 (C++ type), 296  
 BT\_OCTET32\_LEN (C 宏), 284

## C

can\_clear\_receive\_queue (C++ function), 582  
 can\_clear\_transmit\_queue (C++ function), 582  
 can\_driver\_install (C++ function), 580  
 can\_driver\_uninstall (C++ function), 580  
 can\_filter\_config\_t (C++ class), 583  
 can\_filter\_config\_t::acceptance\_code (C++ member), 583  
 can\_filter\_config\_t::acceptance\_mask (C++ member), 583  
 can\_filter\_config\_t::single\_filter (C++ member), 583  
 can\_general\_config\_t (C++ class), 582  
 can\_general\_config\_t::alerts\_enabled (C++ member), 583

- can\_general\_config\_t::bus\_off\_io (C++ member), 583  
 can\_general\_config\_t::clkout\_divider (C++ member), 583  
 can\_general\_config\_t::clkout\_io (C++ member), 583  
 can\_general\_config\_t::intr\_flags (C++ member), 583  
 can\_general\_config\_t::mode (C++ member), 582  
 can\_general\_config\_t::rx\_io (C++ member), 583  
 can\_general\_config\_t::rx\_queue\_len (C++ member), 583  
 can\_general\_config\_t::tx\_io (C++ member), 582  
 can\_general\_config\_t::tx\_queue\_len (C++ member), 583  
 can\_get\_status\_info (C++ function), 582  
 can\_initiate\_recovery (C++ function), 582  
 can\_message\_t (C++ class), 584  
 can\_message\_t::data (C++ member), 584  
 can\_message\_t::data\_length\_code (C++ member), 584  
 can\_message\_t::flags (C++ member), 584  
 can\_message\_t::identifier (C++ member), 584  
 CAN\_MODE\_LISTEN\_ONLY (C++ enumerator), 584  
 CAN\_MODE\_NO\_ACK (C++ enumerator), 584  
 CAN\_MODE\_NORMAL (C++ enumerator), 584  
 can\_mode\_t (C++ enum), 584  
 can\_read\_alerts (C++ function), 581  
 can\_receive (C++ function), 581  
 can\_reconfigure\_alerts (C++ function), 581  
 can\_start (C++ function), 580  
 CAN\_STATE\_BUS\_OFF (C++ enumerator), 585  
 CAN\_STATE\_RECOVERING (C++ enumerator), 585  
 CAN\_STATE\_RUNNING (C++ enumerator), 585  
 CAN\_STATE\_STOPPED (C++ enumerator), 585  
 can\_state\_t (C++ enum), 585  
 can\_status\_info\_t (C++ class), 583  
 can\_status\_info\_t::arb\_lost\_count (C++ member), 584  
 can\_status\_info\_t::bus\_error\_count (C++ member), 584  
 can\_status\_info\_t::msgs\_to\_rx (C++ member), 584  
 can\_status\_info\_t::msgs\_to\_tx (C++ member), 584  
 can\_status\_info\_t::rx\_error\_counter (C++ member), 584  
 can\_status\_info\_t::rx\_missed\_count (C++ member), 584  
 can\_status\_info\_t::state (C++ member), 584  
 can\_status\_info\_t::tx\_error\_counter (C++ member), 584  
 can\_status\_info\_t::tx\_failed\_count (C++ member), 584  
 can\_stop (C++ function), 580  
 can\_timing\_config\_t (C++ class), 583  
 can\_timing\_config\_t::brp (C++ member), 583  
 can\_timing\_config\_t::sjw (C++ member), 583  
 can\_timing\_config\_t::triple\_sampling (C++ member), 583  
 can\_timing\_config\_t::tseg\_1 (C++ member), 583  
 can\_timing\_config\_t::tseg\_2 (C++ member), 583  
 can\_transmit (C++ function), 580  
 CHIP\_ESP32 (C++ enumerator), 1139  
 CHIP\_FEATURE\_BLE (C 宏), 1138  
 CHIP\_FEATURE\_BT (C 宏), 1138  
 CHIP\_FEATURE\_EMB\_FLASH (C 宏), 1138  
 CHIP\_FEATURE\_WIFI\_BGN (C 宏), 1138  
 CONFIG\_EFUSE\_CUSTOM\_TABLE, 1063  
 CONFIG\_EFUSE\_MAX\_BLK\_LEN, 1064  
 CONFIG\_EFUSE\_VIRTUAL, 1067  
 CONFIG\_ESPTOOLPY\_FLASHSIZE, 861  
 CONFIG\_FEATURE\_CACHE\_TX\_BUF\_BIT (C 宏), 476  
 CONFIG\_FEATURE\_WPA3\_SAE\_BIT (C 宏), 476  
 CONFIG\_HEAP\_TRACING\_STACK\_DEPTH (C 宏), 1052  
 CONFIG\_LOG\_DEFAULT\_LEVEL, 1078  
 CONFIG\_LWIP\_USE\_ONLY\_LWIP\_SELECT, 914  
 CONFIG\_SPIRAM\_BANKSWITCH\_ENABLE, 1053  
 CONFIG\_SPIRAM\_BANKSWITCH\_RESERVE, 1053
- ## D
- DAC\_CHANNEL\_1 (C++ enumerator), 586  
 DAC\_CHANNEL\_1\_GPIO\_NUM (C 宏), 587  
 DAC\_CHANNEL\_2 (C++ enumerator), 586  
 DAC\_CHANNEL\_2\_GPIO\_NUM (C 宏), 587  
 DAC\_CHANNEL\_MAX (C++ enumerator), 586  
 dac\_channel\_t (C++ enum), 586  
 DAC\_GPIO25\_CHANNEL (C 宏), 587  
 DAC\_GPIO26\_CHANNEL (C 宏), 587  
 dac\_i2s\_disable (C++ function), 586  
 dac\_i2s\_enable (C++ function), 586  
 dac\_output\_disable (C++ function), 586  
 dac\_output\_enable (C++ function), 586  
 dac\_output\_voltage (C++ function), 586  
 dac\_pad\_get\_io\_num (C++ function), 585  
 DEFAULT\_HTTP\_BUF\_SIZE (C 宏), 780  
 DEL\_DEV\_ADDR\_FLAG (C 宏), 285  
 DEL\_DEV\_UUID\_FLAG (C 宏), 285
- ## E
- eAbortSleep (C++ enumerator), 960  
 eBlocked (C++ enumerator), 959  
 eDeleted (C++ enumerator), 959  
 EFUSE\_BLK0 (C++ enumerator), 1072  
 EFUSE\_BLK1 (C++ enumerator), 1072

- EFUSE\_BLK2 (C++ *enumerator*), 1073
- EFUSE\_BLK3 (C++ *enumerator*), 1073
- EFUSE\_CODE\_SCHEME\_SELECTOR, 1065
- EFUSE\_CODING\_SCHEME\_3\_4 (C++ *enumerator*), 1073
- EFUSE\_CODING\_SCHEME\_NONE (C++ *enumerator*), 1073
- EFUSE\_CODING\_SCHEME\_REPEAT (C++ *enumerator*), 1073
- eIncrement (C++ *enumerator*), 960
- eNoAction (C++ *enumerator*), 959
- eNoTasksWaitingTimeout (C++ *enumerator*), 960
- eNotifyAction (C++ *enum*), 959
- eReady (C++ *enumerator*), 959
- eRunning (C++ *enumerator*), 959
- eSetBits (C++ *enumerator*), 959
- eSetValueWithoutOverwrite (C++ *enumerator*), 960
- eSetValueWithOverwrite (C++ *enumerator*), 960
- eSleepModeStatus (C++ *enum*), 960
- ESP\_A2D\_AUDIO\_CFG\_EVT (C++ *enumerator*), 208
- ESP\_A2D\_AUDIO\_STATE\_EVT (C++ *enumerator*), 207
- ESP\_A2D\_AUDIO\_STATE\_REMOTE\_SUSPEND (C++ *enumerator*), 207
- ESP\_A2D\_AUDIO\_STATE\_STARTED (C++ *enumerator*), 207
- ESP\_A2D\_AUDIO\_STATE\_STOPPED (C++ *enumerator*), 207
- esp\_a2d\_audio\_state\_t (C++ *enum*), 207
- esp\_a2d\_cb\_event\_t (C++ *enum*), 207
- esp\_a2d\_cb\_param\_t (C++ *union*), 204
- esp\_a2d\_cb\_param\_t::a2d\_audio\_cfg\_param (C++ *class*), 204
- esp\_a2d\_cb\_param\_t::a2d\_audio\_cfg\_param::mcc (C++ *member*), 204
- esp\_a2d\_cb\_param\_t::a2d\_audio\_cfg\_param::remodbda (C++ *member*), 204
- esp\_a2d\_cb\_param\_t::a2d\_audio\_stat\_param (C++ *class*), 204
- esp\_a2d\_cb\_param\_t::a2d\_audio\_stat\_param::remodbda (C++ *member*), 204
- esp\_a2d\_cb\_param\_t::a2d\_audio\_stat\_param::state (C++ *member*), 204
- esp\_a2d\_cb\_param\_t::a2d\_conn\_stat\_param (C++ *class*), 205
- esp\_a2d\_cb\_param\_t::a2d\_conn\_stat\_param::remodbda (C++ *member*), 205
- esp\_a2d\_cb\_param\_t::a2d\_conn\_stat\_param::state (C++ *member*), 205
- esp\_a2d\_cb\_param\_t::a2d\_prof\_stat (C++ *member*), 204
- esp\_a2d\_cb\_param\_t::a2d\_prof\_stat\_param (C++ *class*), 205
- esp\_a2d\_cb\_param\_t::a2d\_prof\_stat\_param::init\_state (C++ *member*), 205
- esp\_a2d\_cb\_param\_t::audio\_cfg (C++ *member*), 204
- esp\_a2d\_cb\_param\_t::audio\_stat (C++ *member*), 204
- esp\_a2d\_cb\_param\_t::conn\_stat (C++ *member*), 204
- esp\_a2d\_cb\_param\_t::media\_ctrl\_stat (C++ *member*), 204
- esp\_a2d\_cb\_param\_t::media\_ctrl\_stat\_param (C++ *class*), 205
- esp\_a2d\_cb\_param\_t::media\_ctrl\_stat\_param::cmd (C++ *member*), 205
- esp\_a2d\_cb\_param\_t::media\_ctrl\_stat\_param::status (C++ *member*), 205
- esp\_a2d\_cb\_t (C++ *type*), 206
- ESP\_A2D\_CIE\_LEN\_ATRAC (C 宏), 206
- ESP\_A2D\_CIE\_LEN\_M12 (C 宏), 206
- ESP\_A2D\_CIE\_LEN\_M24 (C 宏), 206
- ESP\_A2D\_CIE\_LEN\_SBC (C 宏), 206
- ESP\_A2D\_CONNECTION\_STATE\_CONNECTED (C++ *enumerator*), 206
- ESP\_A2D\_CONNECTION\_STATE\_CONNECTING (C++ *enumerator*), 206
- ESP\_A2D\_CONNECTION\_STATE\_DISCONNECTED (C++ *enumerator*), 206
- ESP\_A2D\_CONNECTION\_STATE\_DISCONNECTING (C++ *enumerator*), 206
- ESP\_A2D\_CONNECTION\_STATE\_EVT (C++ *enumerator*), 207
- esp\_a2d\_connection\_state\_t (C++ *enum*), 206
- ESP\_A2D\_DEINIT\_SUCCESS (C++ *enumerator*), 207
- ESP\_A2D\_DISC\_RSN\_ABNORMAL (C++ *enumerator*), 206
- ESP\_A2D\_DISC\_RSN\_NORMAL (C++ *enumerator*), 206
- esp\_a2d\_disc\_rsn\_t (C++ *enum*), 206
- esp\_a2d\_init\_state\_t (C++ *enum*), 207
- ESP\_A2D\_INIT\_SUCCESS (C++ *enumerator*), 207
- esp\_a2d\_mcc\_t (C++ *class*), 205
- esp\_a2d\_mcc\_t::cie (C++ *member*), 205
- esp\_a2d\_mcc\_t::type (C++ *member*), 205
- ESP\_A2D\_MCT\_ATRAC (C 宏), 206
- ESP\_A2D\_MCT\_M12 (C 宏), 205
- ESP\_A2D\_MCT\_M24 (C 宏), 205
- ESP\_A2D\_MCT\_NON\_A2DP (C 宏), 206
- ESP\_A2D\_MCT\_SBC (C 宏), 205
- esp\_a2d\_mct\_t (C++ *type*), 206
- esp\_a2d\_media\_ctrl (C++ *function*), 203
- ESP\_A2D\_MEDIA\_CTRL\_ACK\_BUSY (C++ *enumerator*), 207
- ESP\_A2D\_MEDIA\_CTRL\_ACK\_EVT (C++ *enumerator*), 208
- ESP\_A2D\_MEDIA\_CTRL\_ACK\_FAILURE (C++ *enumerator*), 207

- ESP\_A2D\_MEDIA\_CTRL\_ACK\_SUCCESS (C++ enumerator), 207
- esp\_a2d\_media\_ctrl\_ack\_t (C++ enum), 207
- ESP\_A2D\_MEDIA\_CTRL\_CHECK\_SRC\_RDY (C++ enumerator), 207
- ESP\_A2D\_MEDIA\_CTRL\_NONE (C++ enumerator), 207
- ESP\_A2D\_MEDIA\_CTRL\_START (C++ enumerator), 207
- ESP\_A2D\_MEDIA\_CTRL\_STOP (C++ enumerator), 207
- ESP\_A2D\_MEDIA\_CTRL\_SUSPEND (C++ enumerator), 207
- esp\_a2d\_media\_ctrl\_t (C++ enum), 207
- ESP\_A2D\_PROF\_STATE\_EVT (C++ enumerator), 208
- esp\_a2d\_register\_callback (C++ function), 202
- esp\_a2d\_sink\_connect (C++ function), 203
- esp\_a2d\_sink\_data\_cb\_t (C++ type), 206
- esp\_a2d\_sink\_deinit (C++ function), 202
- esp\_a2d\_sink\_disconnect (C++ function), 203
- esp\_a2d\_sink\_init (C++ function), 202
- esp\_a2d\_sink\_register\_data\_callback (C++ function), 202
- esp\_a2d\_source\_connect (C++ function), 203
- esp\_a2d\_source\_data\_cb\_t (C++ type), 206
- esp\_a2d\_source\_deinit (C++ function), 203
- esp\_a2d\_source\_disconnect (C++ function), 204
- esp\_a2d\_source\_init (C++ function), 203
- esp\_a2d\_source\_register\_data\_callback (C++ function), 203
- esp\_adc\_cal\_characteristics\_t (C++ class), 569
- esp\_adc\_cal\_characteristics\_t::adc\_num (C++ member), 569
- esp\_adc\_cal\_characteristics\_t::atten (C++ member), 569
- esp\_adc\_cal\_characteristics\_t::bit\_width (C++ member), 569
- esp\_adc\_cal\_characteristics\_t::coeff\_a (C++ member), 569
- esp\_adc\_cal\_characteristics\_t::coeff\_b (C++ member), 569
- esp\_adc\_cal\_characteristics\_t::high\_curve (C++ member), 569
- esp\_adc\_cal\_characteristics\_t::low\_curve (C++ member), 569
- esp\_adc\_cal\_characteristics\_t::vref (C++ member), 569
- esp\_adc\_cal\_characterize (C++ function), 568
- esp\_adc\_cal\_check\_efuse (C++ function), 568
- esp\_adc\_cal\_get\_voltage (C++ function), 568
- esp\_adc\_cal\_raw\_to\_voltage (C++ function), 568
- ESP\_ADC\_CAL\_VAL\_DEFAULT\_VREF (C++ enumerator), 569
- ESP\_ADC\_CAL\_VAL\_EFUSE\_TP (C++ enumerator), 569
- ESP\_ADC\_CAL\_VAL\_EFUSE\_VREF (C++ enumerator), 569
- esp\_adc\_cal\_value\_t (C++ enum), 569
- ESP\_APP\_DESC\_MAGIC\_WORD (C 宏), 1132
- esp\_app\_desc\_t (C++ class), 1131
- esp\_app\_desc\_t::app\_elf\_sha256 (C++ member), 1131
- esp\_app\_desc\_t::date (C++ member), 1131
- esp\_app\_desc\_t::idf\_ver (C++ member), 1131
- esp\_app\_desc\_t::magic\_word (C++ member), 1131
- esp\_app\_desc\_t::project\_name (C++ member), 1131
- esp\_app\_desc\_t::reserv1 (C++ member), 1131
- esp\_app\_desc\_t::reserv2 (C++ member), 1131
- esp\_app\_desc\_t::secure\_version (C++ member), 1131
- esp\_app\_desc\_t::time (C++ member), 1131
- esp\_app\_desc\_t::version (C++ member), 1131
- ESP\_APP\_ID\_MAX (C 宏), 110
- ESP\_APP\_ID\_MIN (C 宏), 110
- esp\_apprtrace\_buffer\_get (C++ function), 1095
- esp\_apprtrace\_buffer\_put (C++ function), 1096
- esp\_apprtrace\_dest\_t (C++ enum), 1098
- ESP\_APPTRACE\_DEST\_TRAX (C++ enumerator), 1098
- ESP\_APPTRACE\_DEST\_UART0 (C++ enumerator), 1098
- esp\_apprtrace\_down\_buffer\_config (C++ function), 1095
- esp\_apprtrace\_down\_buffer\_get (C++ function), 1097
- esp\_apprtrace\_down\_buffer\_put (C++ function), 1097
- esp\_apprtrace\_fclose (C++ function), 1097
- esp\_apprtrace\_flush (C++ function), 1096
- esp\_apprtrace\_flush\_nolock (C++ function), 1096
- esp\_apprtrace\_fopen (C++ function), 1097
- esp\_apprtrace\_fread (C++ function), 1098
- esp\_apprtrace\_fseek (C++ function), 1098
- esp\_apprtrace\_fstop (C++ function), 1098
- esp\_apprtrace\_ftell (C++ function), 1098
- esp\_apprtrace\_fwrite (C++ function), 1097
- esp\_apprtrace\_host\_is\_connected (C++ function), 1097
- esp\_apprtrace\_init (C++ function), 1095
- esp\_apprtrace\_read (C++ function), 1096
- esp\_apprtrace\_vprintf (C++ function), 1096



- tion), 208
- ESP\_AVRC\_CT\_REMOTE\_FEATURES\_EVT (C++ enumerator), 220
- esp\_avrc\_ct\_send\_get\_rn\_capabilities\_cmd (C++ function), 209
- esp\_avrc\_ct\_send\_metadata\_cmd (C++ function), 209
- esp\_avrc\_ct\_send\_passthrough\_cmd (C++ function), 210
- esp\_avrc\_ct\_send\_register\_notification\_cmd (C++ function), 209
- esp\_avrc\_ct\_send\_set\_absolute\_volume\_cmd (C++ function), 209
- esp\_avrc\_ct\_send\_set\_player\_value\_cmd (C++ function), 208
- ESP\_AVRC\_CT\_SET\_ABSOLUTE\_VOLUME\_RSP\_EVT (C++ enumerator), 220
- ESP\_AVRC\_FEAT\_ADV\_CTRL (C++ enumerator), 216
- ESP\_AVRC\_FEAT\_BROWSE (C++ enumerator), 216
- ESP\_AVRC\_FEAT\_FLAG\_BROWSING (C++ enumerator), 217
- ESP\_AVRC\_FEAT\_FLAG\_CAT1 (C++ enumerator), 216
- ESP\_AVRC\_FEAT\_FLAG\_CAT2 (C++ enumerator), 217
- ESP\_AVRC\_FEAT\_FLAG\_CAT3 (C++ enumerator), 217
- ESP\_AVRC\_FEAT\_FLAG\_CAT4 (C++ enumerator), 217
- ESP\_AVRC\_FEAT\_FLAG\_COVER\_ART\_GET\_IMAGE (C++ enumerator), 217
- ESP\_AVRC\_FEAT\_FLAG\_COVER\_ART\_GET\_IMAGE\_LINKED\_THUMB (C++ enumerator), 217
- ESP\_AVRC\_FEAT\_FLAG\_COVER\_ART\_GET\_LINKED\_THUMB (C++ enumerator), 217
- ESP\_AVRC\_FEAT\_META\_DATA (C++ enumerator), 216
- ESP\_AVRC\_FEAT\_RCCT (C++ enumerator), 216
- ESP\_AVRC\_FEAT\_RCTG (C++ enumerator), 216
- ESP\_AVRC\_FEAT\_VENDOR (C++ enumerator), 216
- esp\_avrc\_feature\_flag\_t (C++ enum), 216
- esp\_avrc\_features\_t (C++ enum), 216
- ESP\_AVRC\_MD\_ATTR\_ALBUM (C++ enumerator), 221
- ESP\_AVRC\_MD\_ATTR\_ARTIST (C++ enumerator), 221
- ESP\_AVRC\_MD\_ATTR\_GENRE (C++ enumerator), 221
- esp\_avrc\_md\_attr\_mask\_t (C++ enum), 221
- ESP\_AVRC\_MD\_ATTR\_NUM\_TRACKS (C++ enumerator), 221
- ESP\_AVRC\_MD\_ATTR\_PLAYING\_TIME (C++ enumerator), 221
- ESP\_AVRC\_MD\_ATTR\_TITLE (C++ enumerator), 221
- ESP\_AVRC\_MD\_ATTR\_TRACK\_NUM (C++ enumerator), 221
- ESP\_AVRC\_PLAYBACK\_ERROR (C++ enumerator), 224
- ESP\_AVRC\_PLAYBACK\_FWD\_SEEK (C++ enumerator), 224
- ESP\_AVRC\_PLAYBACK\_PAUSED (C++ enumerator), 224
- ESP\_AVRC\_PLAYBACK\_PLAYING (C++ enumerator), 224
- ESP\_AVRC\_PLAYBACK\_REV\_SEEK (C++ enumerator), 224
- esp\_avrc\_playback\_stat\_t (C++ enum), 224
- ESP\_AVRC\_PLAYBACK\_STOPPED (C++ enumerator), 224
- esp\_avrc\_ps\_attr\_ids\_t (C++ enum), 222
- esp\_avrc\_ps\_eq\_value\_ids\_t (C++ enum), 222
- ESP\_AVRC\_PS\_EQUALIZER (C++ enumerator), 222
- ESP\_AVRC\_PS\_EQUALIZER\_OFF (C++ enumerator), 222
- ESP\_AVRC\_PS\_EQUALIZER\_ON (C++ enumerator), 222
- ESP\_AVRC\_PS\_MAX\_ATTR (C++ enumerator), 222
- ESP\_AVRC\_PS\_REPEAT\_GROUP (C++ enumerator), 222
- ESP\_AVRC\_PS\_REPEAT\_MODE (C++ enumerator), 222
- ESP\_AVRC\_PS\_REPEAT\_OFF (C++ enumerator), 222
- ESP\_AVRC\_PS\_REPEAT\_SINGLE (C++ enumerator), 222
- esp\_avrc\_ps\_rpt\_value\_ids\_t (C++ enum), 222
- ESP\_AVRC\_PS\_SCAN\_ALL (C++ enumerator), 223
- ESP\_AVRC\_PS\_SCAN\_GROUP (C++ enumerator), 223
- ESP\_AVRC\_PS\_SCAN\_MODE (C++ enumerator), 223
- ESP\_AVRC\_PS\_SCAN\_OFF (C++ enumerator), 223
- esp\_avrc\_ps\_scn\_value\_ids\_t (C++ enum), 223
- esp\_avrc\_ps\_shf\_value\_ids\_t (C++ enum), 223
- ESP\_AVRC\_PS\_SHUFFLE\_ALL (C++ enumerator), 223
- ESP\_AVRC\_PS\_SHUFFLE\_GROUP (C++ enumerator), 223
- ESP\_AVRC\_PS\_SHUFFLE\_MODE (C++ enumerator), 223
- ESP\_AVRC\_PS\_SHUFFLE\_OFF (C++ enumerator), 223
- esp\_avrc\_psth\_bit\_mask\_operation (C++ function), 211
- esp\_avrc\_psth\_bit\_mask\_t (C++ class), 215
- esp\_avrc\_psth\_bit\_mask\_t::bits (C++ member), 216
- ESP\_AVRC\_PSTH\_FILTER\_ALLOWED\_CMD (C++ enumerator), 219
- ESP\_AVRC\_PSTH\_FILTER\_SUPPORT\_MAX (C++ enumerator), 220



- ESP\_AVRC\_PSTH\_FILTER\_SUPPORTED\_CMD (C++ enumerator), 219
- esp\_avrc\_psth\_filter\_t (C++ enum), 219
- ESP\_AVRC\_PT\_CMD\_0 (C++ enumerator), 217
- ESP\_AVRC\_PT\_CMD\_1 (C++ enumerator), 218
- ESP\_AVRC\_PT\_CMD\_2 (C++ enumerator), 218
- ESP\_AVRC\_PT\_CMD\_3 (C++ enumerator), 218
- ESP\_AVRC\_PT\_CMD\_4 (C++ enumerator), 218
- ESP\_AVRC\_PT\_CMD\_5 (C++ enumerator), 218
- ESP\_AVRC\_PT\_CMD\_6 (C++ enumerator), 218
- ESP\_AVRC\_PT\_CMD\_7 (C++ enumerator), 218
- ESP\_AVRC\_PT\_CMD\_8 (C++ enumerator), 218
- ESP\_AVRC\_PT\_CMD\_9 (C++ enumerator), 218
- ESP\_AVRC\_PT\_CMD\_ANGLE (C++ enumerator), 219
- ESP\_AVRC\_PT\_CMD\_BACKWARD (C++ enumerator), 219
- ESP\_AVRC\_PT\_CMD\_CHAN\_DOWN (C++ enumerator), 218
- ESP\_AVRC\_PT\_CMD\_CHAN\_UP (C++ enumerator), 218
- ESP\_AVRC\_PT\_CMD\_CLEAR (C++ enumerator), 218
- ESP\_AVRC\_PT\_CMD\_CONT\_MENU (C++ enumerator), 217
- ESP\_AVRC\_PT\_CMD\_DISP\_INFO (C++ enumerator), 218
- ESP\_AVRC\_PT\_CMD\_DOT (C++ enumerator), 218
- ESP\_AVRC\_PT\_CMD\_DOWN (C++ enumerator), 217
- ESP\_AVRC\_PT\_CMD\_EJECT (C++ enumerator), 219
- ESP\_AVRC\_PT\_CMD\_ENTER (C++ enumerator), 218
- ESP\_AVRC\_PT\_CMD\_EXIT (C++ enumerator), 217
- ESP\_AVRC\_PT\_CMD\_F1 (C++ enumerator), 219
- ESP\_AVRC\_PT\_CMD\_F2 (C++ enumerator), 219
- ESP\_AVRC\_PT\_CMD\_F3 (C++ enumerator), 219
- ESP\_AVRC\_PT\_CMD\_F4 (C++ enumerator), 219
- ESP\_AVRC\_PT\_CMD\_F5 (C++ enumerator), 219
- ESP\_AVRC\_PT\_CMD\_FAST\_FORWARD (C++ enumerator), 219
- ESP\_AVRC\_PT\_CMD\_FAV\_MENU (C++ enumerator), 217
- ESP\_AVRC\_PT\_CMD\_FORWARD (C++ enumerator), 219
- ESP\_AVRC\_PT\_CMD\_HELP (C++ enumerator), 218
- ESP\_AVRC\_PT\_CMD\_INPUT\_SEL (C++ enumerator), 218
- ESP\_AVRC\_PT\_CMD\_LEFT (C++ enumerator), 217
- ESP\_AVRC\_PT\_CMD\_LEFT\_DOWN (C++ enumerator), 217
- ESP\_AVRC\_PT\_CMD\_LEFT\_UP (C++ enumerator), 217
- ESP\_AVRC\_PT\_CMD\_MUTE (C++ enumerator), 219
- ESP\_AVRC\_PT\_CMD\_PAGE\_DOWN (C++ enumerator), 218
- ESP\_AVRC\_PT\_CMD\_PAGE\_UP (C++ enumerator), 218
- ESP\_AVRC\_PT\_CMD\_PAUSE (C++ enumerator), 219
- ESP\_AVRC\_PT\_CMD\_PLAY (C++ enumerator), 219
- ESP\_AVRC\_PT\_CMD\_POWER (C++ enumerator), 218
- ESP\_AVRC\_PT\_CMD\_PREV\_CHAN (C++ enumerator), 218
- ESP\_AVRC\_PT\_CMD\_RECORD (C++ enumerator), 219
- ESP\_AVRC\_PT\_CMD\_REWIND (C++ enumerator), 219
- ESP\_AVRC\_PT\_CMD\_RIGHT (C++ enumerator), 217
- ESP\_AVRC\_PT\_CMD\_RIGHT\_DOWN (C++ enumerator), 217
- ESP\_AVRC\_PT\_CMD\_RIGHT\_UP (C++ enumerator), 217
- ESP\_AVRC\_PT\_CMD\_ROOT\_MENU (C++ enumerator), 217
- ESP\_AVRC\_PT\_CMD\_SELECT (C++ enumerator), 217
- ESP\_AVRC\_PT\_CMD\_SETUP\_MENU (C++ enumerator), 217
- ESP\_AVRC\_PT\_CMD\_SOUND\_SEL (C++ enumerator), 218
- ESP\_AVRC\_PT\_CMD\_STATE\_PRESSED (C++ enumerator), 220
- ESP\_AVRC\_PT\_CMD\_STATE\_RELEASED (C++ enumerator), 220
- esp\_avrc\_pt\_cmd\_state\_t (C++ enum), 220
- ESP\_AVRC\_PT\_CMD\_STOP (C++ enumerator), 219
- ESP\_AVRC\_PT\_CMD\_SUBPICT (C++ enumerator), 219
- esp\_avrc\_pt\_cmd\_t (C++ enum), 217
- ESP\_AVRC\_PT\_CMD\_UP (C++ enumerator), 217
- ESP\_AVRC\_PT\_CMD\_VENDOR (C++ enumerator), 219
- ESP\_AVRC\_PT\_CMD\_VOL\_DOWN (C++ enumerator), 219
- ESP\_AVRC\_PT\_CMD\_VOL\_UP (C++ enumerator), 218
- ESP\_AVRC\_RN\_ADDRESSED\_PLAYER\_CHANGE (C++ enumerator), 221
- ESP\_AVRC\_RN\_APP\_SETTING\_CHANGE (C++ enumerator), 221
- ESP\_AVRC\_RN\_AVAILABLE\_PLAYERS\_CHANGE (C++ enumerator), 221
- ESP\_AVRC\_RN\_BATTERY\_STATUS\_CHANGE (C++ enumerator), 221
- ESP\_AVRC\_RN\_CAP\_ALLOWED\_EVT (C++ enumerator), 222
- ESP\_AVRC\_RN\_CAP\_MAX (C++ enumerator), 222
- ESP\_AVRC\_RN\_CAP\_SUPPORTED\_EVT (C++ enumerator), 222
- esp\_avrc\_rn\_event\_ids\_t (C++ enum), 221
- esp\_avrc\_rn\_evt\_bit\_mask\_operation (C++ function), 211
- esp\_avrc\_rn\_evt\_cap\_mask\_t (C++ class), 216
- esp\_avrc\_rn\_evt\_cap\_mask\_t::bits (C++ member), 216
- esp\_avrc\_rn\_evt\_cap\_t (C++ enum), 222
- ESP\_AVRC\_RN\_MAX\_EVT (C++ enumerator), 222
- ESP\_AVRC\_RN\_NOW\_PLAYING\_CHANGE (C++

- enumerator*), 221
- esp\_avrc\_rn\_param\_t (C++ union), 212
- esp\_avrc\_rn\_param\_t::batt (C++ member), 212
- esp\_avrc\_rn\_param\_t::elm\_id (C++ member), 212
- esp\_avrc\_rn\_param\_t::play\_pos (C++ member), 212
- esp\_avrc\_rn\_param\_t::playback (C++ member), 212
- esp\_avrc\_rn\_param\_t::volume (C++ member), 212
- ESP\_AVRC\_RN\_PLAY\_POS\_CHANGED (C++ enumerator), 221
- ESP\_AVRC\_RN\_PLAY\_STATUS\_CHANGE (C++ enumerator), 221
- ESP\_AVRC\_RN\_RSP\_CHANGED (C++ enumerator), 222
- ESP\_AVRC\_RN\_RSP\_INTERIM (C++ enumerator), 222
- esp\_avrc\_rn\_rsp\_t (C++ enum), 222
- ESP\_AVRC\_RN\_SYSTEM\_STATUS\_CHANGE (C++ enumerator), 221
- ESP\_AVRC\_RN\_TRACK\_CHANGE (C++ enumerator), 221
- ESP\_AVRC\_RN\_TRACK\_REACHED\_END (C++ enumerator), 221
- ESP\_AVRC\_RN\_TRACK\_REACHED\_START (C++ enumerator), 221
- ESP\_AVRC\_RN\_UIDS\_CHANGE (C++ enumerator), 221
- ESP\_AVRC\_RN\_VOLUME\_CHANGE (C++ enumerator), 221
- ESP\_AVRC\_RSP\_ACCEPT (C++ enumerator), 223
- ESP\_AVRC\_RSP\_CHANGED (C++ enumerator), 223
- ESP\_AVRC\_RSP\_IMPL\_STBL (C++ enumerator), 223
- ESP\_AVRC\_RSP\_IN\_TRANS (C++ enumerator), 223
- ESP\_AVRC\_RSP\_INTERIM (C++ enumerator), 223
- ESP\_AVRC\_RSP\_NOT\_IMPL (C++ enumerator), 223
- ESP\_AVRC\_RSP\_REJECT (C++ enumerator), 223
- esp\_avrc\_rsp\_t (C++ enum), 223
- esp\_avrc\_tg\_cb\_event\_t (C++ enum), 220
- esp\_avrc\_tg\_cb\_param\_t (C++ union), 214
- esp\_avrc\_tg\_cb\_param\_t::avrc\_tg\_conn\_status\_param (C++ class), 214
- esp\_avrc\_tg\_cb\_param\_t::avrc\_tg\_conn\_status\_param (C++ member), 215
- esp\_avrc\_tg\_cb\_param\_t::avrc\_tg\_conn\_status\_param (C++ member), 215
- esp\_avrc\_tg\_cb\_param\_t::avrc\_tg\_psth\_cmd (C++ class), 215
- esp\_avrc\_tg\_cb\_param\_t::avrc\_tg\_psth\_cmd (C++ member), 215
- esp\_avrc\_tg\_cb\_param\_t::avrc\_tg\_psth\_cmd (C++ member), 215
- esp\_avrc\_tg\_cb\_param\_t::avrc\_tg\_reg\_ntf\_param (C++ class), 215
- esp\_avrc\_tg\_cb\_param\_t::avrc\_tg\_reg\_ntf\_param::ev (C++ member), 215
- esp\_avrc\_tg\_cb\_param\_t::avrc\_tg\_reg\_ntf\_param::ev (C++ member), 215
- esp\_avrc\_tg\_cb\_param\_t::avrc\_tg\_rmt\_feats\_param (C++ class), 215
- esp\_avrc\_tg\_cb\_param\_t::avrc\_tg\_rmt\_feats\_param:: (C++ member), 215
- esp\_avrc\_tg\_cb\_param\_t::avrc\_tg\_rmt\_feats\_param:: (C++ member), 215
- esp\_avrc\_tg\_cb\_param\_t::avrc\_tg\_rmt\_feats\_param:: (C++ member), 215
- esp\_avrc\_tg\_cb\_param\_t::avrc\_tg\_rmt\_feats\_param:: (C++ member), 215
- esp\_avrc\_tg\_cb\_param\_t::avrc\_tg\_set\_abs\_vol\_param (C++ class), 215
- esp\_avrc\_tg\_cb\_param\_t::avrc\_tg\_set\_abs\_vol\_param (C++ member), 215
- esp\_avrc\_tg\_cb\_param\_t::conn\_stat (C++ member), 214
- esp\_avrc\_tg\_cb\_param\_t::psth\_cmd (C++ member), 214
- esp\_avrc\_tg\_cb\_param\_t::reg\_ntf (C++ member), 214
- esp\_avrc\_tg\_cb\_param\_t::rmt\_feats (C++ member), 214
- esp\_avrc\_tg\_cb\_param\_t::set\_abs\_vol (C++ member), 214
- esp\_avrc\_tg\_cb\_t (C++ type), 216
- ESP\_AVRC\_TG\_CONNECTION\_STATE\_EVT (C++ enumerator), 220
- esp\_avrc\_tg\_deinit (C++ function), 210
- esp\_avrc\_tg\_get\_psth\_cmd\_filter (C++ function), 210
- esp\_avrc\_tg\_get\_rn\_evt\_cap (C++ function), 211
- esp\_avrc\_tg\_init (C++ function), 210
- ESP\_AVRC\_TG\_PASSTHROUGH\_CMD\_EVT (C++ enumerator), 220
- esp\_avrc\_tg\_register\_callback (C++ function), 210
- ESP\_AVRC\_TG\_REGISTER\_NOTIFICATION\_EVT (C++ enumerator), 220
- ESP\_AVRC\_TG\_REMOTE\_FEATURES\_EVT (C++ enumerator), 220
- esp\_avrc\_tg\_send\_rn\_rsp (C++ function), 212
- ESP\_AVRC\_TG\_SET\_ABSOLUTE\_VOLUME\_CMD\_EVT (C++ enumerator), 220
- esp\_avrc\_tg\_set\_psth\_cmd\_filter (C++ function), 211
- esp\_avrc\_tg\_set\_rn\_evt\_cap (C++ function), 211
- ESP\_AVRC\_TRANS\_LABEL\_MAX (C 宏), 216
- esp\_base\_mac\_addr\_get (C++ function), 1137
- esp\_base\_mac\_addr\_set (C++ function), 1137
- ESP\_BD\_ADDR\_HEX (C 宏), 110
- ESP\_BD\_ADDR\_KEY\_LEN (C 宏), 109
- ESP\_BD\_ADDR\_STR (C 宏), 110
- esp\_base\_mac\_addr\_t (C++ type), 110
- ESP\_BLE\_AD\_MANUFACTURER\_SPECIFIC\_TYPE

- (C++ *enumerator*), 136
- ESP\_BLE\_AD\_TYPE\_128SERVICE\_DATA (C++ *enumerator*), 135
- ESP\_BLE\_AD\_TYPE\_128SOL\_SRV\_UUID (C++ *enumerator*), 135
- ESP\_BLE\_AD\_TYPE\_128SRV\_CMPL (C++ *enumerator*), 135
- ESP\_BLE\_AD\_TYPE\_128SRV\_PART (C++ *enumerator*), 135
- ESP\_BLE\_AD\_TYPE\_16SRV\_CMPL (C++ *enumerator*), 135
- ESP\_BLE\_AD\_TYPE\_16SRV\_PART (C++ *enumerator*), 135
- ESP\_BLE\_AD\_TYPE\_32SERVICE\_DATA (C++ *enumerator*), 135
- ESP\_BLE\_AD\_TYPE\_32SOL\_SRV\_UUID (C++ *enumerator*), 135
- ESP\_BLE\_AD\_TYPE\_32SRV\_CMPL (C++ *enumerator*), 135
- ESP\_BLE\_AD\_TYPE\_32SRV\_PART (C++ *enumerator*), 135
- ESP\_BLE\_AD\_TYPE\_ADV\_INT (C++ *enumerator*), 135
- ESP\_BLE\_AD\_TYPE\_APPEARANCE (C++ *enumerator*), 135
- ESP\_BLE\_AD\_TYPE\_CHAN\_MAP\_UPDATE (C++ *enumerator*), 136
- ESP\_BLE\_AD\_TYPE\_DEV\_CLASS (C++ *enumerator*), 135
- ESP\_BLE\_AD\_TYPE\_FLAG (C++ *enumerator*), 135
- ESP\_BLE\_AD\_TYPE\_INDOOR\_POSITION (C++ *enumerator*), 136
- ESP\_BLE\_AD\_TYPE\_INT\_RANGE (C++ *enumerator*), 135
- ESP\_BLE\_AD\_TYPE\_LE\_DEV\_ADDR (C++ *enumerator*), 135
- ESP\_BLE\_AD\_TYPE\_LE\_ROLE (C++ *enumerator*), 135
- ESP\_BLE\_AD\_TYPE\_LE\_SECURE\_CONFIRM (C++ *enumerator*), 136
- ESP\_BLE\_AD\_TYPE\_LE\_SECURE\_RANDOM (C++ *enumerator*), 136
- ESP\_BLE\_AD\_TYPE\_LE\_SUPPORT\_FEATURE (C++ *enumerator*), 136
- ESP\_BLE\_AD\_TYPE\_NAME\_CMPL (C++ *enumerator*), 135
- ESP\_BLE\_AD\_TYPE\_NAME\_SHORT (C++ *enumerator*), 135
- ESP\_BLE\_AD\_TYPE\_PUBLIC\_TARGET (C++ *enumerator*), 135
- ESP\_BLE\_AD\_TYPE\_RANDOM\_TARGET (C++ *enumerator*), 135
- ESP\_BLE\_AD\_TYPE\_SERVICE\_DATA (C++ *enumerator*), 135
- ESP\_BLE\_AD\_TYPE\_SM\_OOB\_FLAG (C++ *enumerator*), 135
- ESP\_BLE\_AD\_TYPE\_SM\_TK (C++ *enumerator*), 135
- ESP\_BLE\_AD\_TYPE\_SOL\_SRV\_UUID (C++ *enumerator*), 135
- ESP\_BLE\_AD\_TYPE\_SPAIR\_C256 (C++ *enumerator*), 135
- ESP\_BLE\_AD\_TYPE\_SPAIR\_R256 (C++ *enumerator*), 135
- ESP\_BLE\_AD\_TYPE\_TRANS\_DISC\_DATA (C++ *enumerator*), 136
- ESP\_BLE\_AD\_TYPE\_TX\_PWR (C++ *enumerator*), 135
- ESP\_BLE\_AD\_TYPE\_URI (C++ *enumerator*), 136
- esp\_ble\_addr\_type\_t (C++ *enum*), 111
- esp\_ble\_adv\_channel\_t (C++ *enum*), 136
- ESP\_BLE\_ADV\_DATA\_LEN\_MAX (C 宏), 133
- esp\_ble\_adv\_data\_t (C++ *class*), 126
- esp\_ble\_adv\_data\_t::appearance (C++ *member*), 126
- esp\_ble\_adv\_data\_t::flag (C++ *member*), 126
- esp\_ble\_adv\_data\_t::include\_name (C++ *member*), 126
- esp\_ble\_adv\_data\_t::include\_txpower (C++ *member*), 126
- esp\_ble\_adv\_data\_t::manufacturer\_len (C++ *member*), 126
- esp\_ble\_adv\_data\_t::max\_interval (C++ *member*), 126
- esp\_ble\_adv\_data\_t::min\_interval (C++ *member*), 126
- esp\_ble\_adv\_data\_t::p\_manufacturer\_data (C++ *member*), 126
- esp\_ble\_adv\_data\_t::p\_service\_data (C++ *member*), 126
- esp\_ble\_adv\_data\_t::p\_service\_uuid (C++ *member*), 126
- esp\_ble\_adv\_data\_t::service\_data\_len (C++ *member*), 126
- esp\_ble\_adv\_data\_t::service\_uuid\_len (C++ *member*), 126
- esp\_ble\_adv\_data\_t::set\_scan\_rsp (C++ *member*), 126
- esp\_ble\_adv\_data\_type (C++ *enum*), 135
- esp\_ble\_adv\_filter\_t (C++ *enum*), 136
- ESP\_BLE\_ADV\_FLAG\_BREDR\_NOT\_SPT (C 宏), 131
- ESP\_BLE\_ADV\_FLAG\_DMT\_CONTROLLER\_SPT (C 宏), 131
- ESP\_BLE\_ADV\_FLAG\_DMT\_HOST\_SPT (C 宏), 131
- ESP\_BLE\_ADV\_FLAG\_GEN\_DISC (C 宏), 131
- ESP\_BLE\_ADV\_FLAG\_LIMIT\_DISC (C 宏), 131
- ESP\_BLE\_ADV\_FLAG\_NON\_LIMIT\_DISC (C 宏), 131
- esp\_ble\_adv\_params\_t (C++ *class*), 125
- esp\_ble\_adv\_params\_t::adv\_filter\_policy (C++ *member*), 126
- esp\_ble\_adv\_params\_t::adv\_int\_max (C++ *member*), 125
- esp\_ble\_adv\_params\_t::adv\_int\_min (C++ *member*), 125

- esp\_ble\_adv\_params\_t::adv\_type (C++ member), 125  
 esp\_ble\_adv\_params\_t::channel\_map (C++ member), 126  
 esp\_ble\_adv\_params\_t::own\_addr\_type (C++ member), 125  
 esp\_ble\_adv\_params\_t::peer\_addr (C++ member), 125  
 esp\_ble\_adv\_params\_t::peer\_addr\_type (C++ member), 125  
 esp\_ble\_adv\_type\_t (C++ enum), 136  
 ESP\_BLE\_APP\_ENC\_KEY\_SIZE (C++ enumerator), 137  
 ESP\_BLE\_APPEARANCE\_BLOOD\_PRESSURE\_ARM (C 宏), 132  
 ESP\_BLE\_APPEARANCE\_BLOOD\_PRESSURE\_WRIST (C 宏), 132  
 ESP\_BLE\_APPEARANCE\_CYCLING\_CADENCE (C 宏), 133  
 ESP\_BLE\_APPEARANCE\_CYCLING\_COMPUTER (C 宏), 132  
 ESP\_BLE\_APPEARANCE\_CYCLING\_POWER (C 宏), 133  
 ESP\_BLE\_APPEARANCE\_CYCLING\_SPEED (C 宏), 133  
 ESP\_BLE\_APPEARANCE\_CYCLING\_SPEED\_CADENCE (C 宏), 133  
 ESP\_BLE\_APPEARANCE\_GENERIC\_BARCODE\_SCANNER (C 宏), 132  
 ESP\_BLE\_APPEARANCE\_GENERIC\_BLOOD\_PRESSURE (C 宏), 132  
 ESP\_BLE\_APPEARANCE\_GENERIC\_CLOCK (C 宏), 132  
 ESP\_BLE\_APPEARANCE\_GENERIC\_COMPUTER (C 宏), 132  
 ESP\_BLE\_APPEARANCE\_GENERIC\_CONTINUOUS\_GLUCOSE\_MONITOR (C 宏), 133  
 ESP\_BLE\_APPEARANCE\_GENERIC\_CYCLING (C 宏), 132  
 ESP\_BLE\_APPEARANCE\_GENERIC\_DISPLAY (C 宏), 132  
 ESP\_BLE\_APPEARANCE\_GENERIC\_EYEGLASSES (C 宏), 132  
 ESP\_BLE\_APPEARANCE\_GENERIC\_GLUCOSE (C 宏), 132  
 ESP\_BLE\_APPEARANCE\_GENERIC\_HEART\_RATE (C 宏), 132  
 ESP\_BLE\_APPEARANCE\_GENERIC\_HID (C 宏), 132  
 ESP\_BLE\_APPEARANCE\_GENERIC\_INSULIN\_PUMP (C 宏), 133  
 ESP\_BLE\_APPEARANCE\_GENERIC\_KEYRING (C 宏), 132  
 ESP\_BLE\_APPEARANCE\_GENERIC\_MEDIA\_PLAYER (C 宏), 132  
 ESP\_BLE\_APPEARANCE\_GENERIC\_MEDICATION\_DELIVERY (C 宏), 133  
 ESP\_BLE\_APPEARANCE\_GENERIC\_OUTDOOR\_SPORTS (C 宏), 133  
 ESP\_BLE\_APPEARANCE\_GENERIC\_PERSONAL\_MOBILITY\_DEVICE (C 宏), 133  
 ESP\_BLE\_APPEARANCE\_GENERIC\_PHONE (C 宏), 132  
 ESP\_BLE\_APPEARANCE\_GENERIC\_PULSE\_OXIMETER (C 宏), 133  
 ESP\_BLE\_APPEARANCE\_GENERIC\_REMOTE (C 宏), 132  
 ESP\_BLE\_APPEARANCE\_GENERIC\_TAG (C 宏), 132  
 ESP\_BLE\_APPEARANCE\_GENERIC\_THERMOMETER (C 宏), 132  
 ESP\_BLE\_APPEARANCE\_GENERIC\_WALKING (C 宏), 132  
 ESP\_BLE\_APPEARANCE\_GENERIC\_WATCH (C 宏), 132  
 ESP\_BLE\_APPEARANCE\_GENERIC\_WEIGHT (C 宏), 133  
 ESP\_BLE\_APPEARANCE\_HEART\_RATE\_BELT (C 宏), 132  
 ESP\_BLE\_APPEARANCE\_HID\_BARCODE\_SCANNER (C 宏), 132  
 ESP\_BLE\_APPEARANCE\_HID\_CARD\_READER (C 宏), 132  
 ESP\_BLE\_APPEARANCE\_HID\_DIGITAL\_PEN (C 宏), 132  
 ESP\_BLE\_APPEARANCE\_HID\_DIGITIZER\_TABLET (C 宏), 132  
 ESP\_BLE\_APPEARANCE\_HID\_GAMEPAD (C 宏), 132  
 ESP\_BLE\_APPEARANCE\_HID\_JOYSTICK (C 宏), 132  
 ESP\_BLE\_APPEARANCE\_HID\_KEYBOARD (C 宏), 132  
 ESP\_BLE\_APPEARANCE\_HID\_MOUSE (C 宏), 132  
 ESP\_BLE\_APPEARANCE\_INSULIN\_PEN (C 宏), 133  
 ESP\_BLE\_APPEARANCE\_INSULIN\_PUMP\_DURABLE\_PUMP (C 宏), 133  
 ESP\_BLE\_APPEARANCE\_INSULIN\_PUMP\_PATCH\_PUMP (C 宏), 133  
 ESP\_BLE\_APPEARANCE\_MOBILITY\_SCOOTER (C 宏), 133  
 ESP\_BLE\_APPEARANCE\_OUTDOOR\_SPORTS\_LOCATION (C 宏), 133  
 ESP\_BLE\_APPEARANCE\_OUTDOOR\_SPORTS\_LOCATION\_AND\_NAVIGATION (C 宏), 133  
 ESP\_BLE\_APPEARANCE\_OUTDOOR\_SPORTS\_LOCATION\_POD (C 宏), 133  
 ESP\_BLE\_APPEARANCE\_OUTDOOR\_SPORTS\_LOCATION\_POD\_AND\_NAVIGATION (C 宏), 133  
 ESP\_BLE\_APPEARANCE\_POWERED\_WHEELCHAIR (C 宏), 133  
 ESP\_BLE\_APPEARANCE\_PULSE\_OXIMETER\_FINGERTIP (C 宏), 133  
 ESP\_BLE\_APPEARANCE\_PULSE\_OXIMETER\_WRIST (C 宏), 133

- ESP\_BLE\_APPEARANCE\_SPORTS\_WATCH (C 宏), 132
- ESP\_BLE\_APPEARANCE\_THERMOMETER\_EAR (C 宏), 132
- ESP\_BLE\_APPEARANCE\_UNKNOWN (C 宏), 132
- ESP\_BLE\_APPEARANCE\_WALKING\_IN\_SHOE (C 宏), 132
- ESP\_BLE\_APPEARANCE\_WALKING\_ON\_HIP (C 宏), 132
- ESP\_BLE\_APPEARANCE\_WALKING\_ON\_SHOE (C 宏), 132
- esp\_ble\_auth\_cmpl\_t (C++ class), 130
- esp\_ble\_auth\_cmpl\_t::addr\_type (C++ member), 131
- esp\_ble\_auth\_cmpl\_t::auth\_mode (C++ member), 131
- esp\_ble\_auth\_cmpl\_t::bd\_addr (C++ member), 130
- esp\_ble\_auth\_cmpl\_t::dev\_type (C++ member), 131
- esp\_ble\_auth\_cmpl\_t::fail\_reason (C++ member), 131
- esp\_ble\_auth\_cmpl\_t::key (C++ member), 130
- esp\_ble\_auth\_cmpl\_t::key\_present (C++ member), 130
- esp\_ble\_auth\_cmpl\_t::key\_type (C++ member), 130
- esp\_ble\_auth\_cmpl\_t::success (C++ member), 130
- esp\_ble\_auth\_req\_t (C++ type), 133
- esp\_ble\_bond\_dev\_t (C++ class), 130
- esp\_ble\_bond\_dev\_t::bd\_addr (C++ member), 130
- esp\_ble\_bond\_dev\_t::bond\_key (C++ member), 130
- esp\_ble\_bond\_key\_info\_t (C++ class), 129
- esp\_ble\_bond\_key\_info\_t::key\_mask (C++ member), 129
- esp\_ble\_bond\_key\_info\_t::pcsrk\_key (C++ member), 129
- esp\_ble\_bond\_key\_info\_t::penc\_key (C++ member), 129
- esp\_ble\_bond\_key\_info\_t::pid\_key (C++ member), 130
- esp\_ble\_confirm\_reply (C++ function), 118
- ESP\_BLE\_CONN\_INT\_MAX (C 宏), 109
- ESP\_BLE\_CONN\_INT\_MIN (C 宏), 109
- ESP\_BLE\_CONN\_LATENCY\_MAX (C 宏), 109
- ESP\_BLE\_CONN\_PARAM\_UNDEF (C 宏), 109
- ESP\_BLE\_CONN\_SUP\_TOUT\_MAX (C 宏), 109
- ESP\_BLE\_CONN\_SUP\_TOUT\_MIN (C 宏), 109
- esp\_ble\_conn\_update\_params\_t (C++ class), 127
- esp\_ble\_conn\_update\_params\_t::bda (C++ member), 127
- esp\_ble\_conn\_update\_params\_t::latency (C++ member), 127
- esp\_ble\_conn\_update\_params\_t::max\_int (C++ member), 127
- esp\_ble\_conn\_update\_params\_t::min\_int (C++ member), 127
- esp\_ble\_conn\_update\_params\_t::timeout (C++ member), 127
- ESP\_BLE\_CSR\_KEY\_MASK (C 宏), 110
- esp\_ble\_duplicate\_exceptional\_info\_type\_t (C++ enum), 139
- ESP\_BLE\_DUPLICATE\_EXCEPTIONAL\_LIST\_ADD (C++ enumerator), 138
- ESP\_BLE\_DUPLICATE\_EXCEPTIONAL\_LIST\_CLEAN (C++ enumerator), 139
- ESP\_BLE\_DUPLICATE\_EXCEPTIONAL\_LIST\_REMOVE (C++ enumerator), 138
- ESP\_BLE\_DUPLICATE\_SCAN\_EXCEPTIONAL\_ADDR\_LIST (C++ enumerator), 139
- ESP\_BLE\_DUPLICATE\_SCAN\_EXCEPTIONAL\_ALL\_LIST (C++ enumerator), 139
- ESP\_BLE\_DUPLICATE\_SCAN\_EXCEPTIONAL\_INFO\_ADV\_ADDR (C++ enumerator), 139
- ESP\_BLE\_DUPLICATE\_SCAN\_EXCEPTIONAL\_INFO\_MESH\_BEACON (C++ enumerator), 139
- ESP\_BLE\_DUPLICATE\_SCAN\_EXCEPTIONAL\_INFO\_MESH\_LINK (C++ enumerator), 139
- ESP\_BLE\_DUPLICATE\_SCAN\_EXCEPTIONAL\_INFO\_MESH\_PROXY (C++ enumerator), 139
- ESP\_BLE\_DUPLICATE\_SCAN\_EXCEPTIONAL\_INFO\_MESH\_PROXY\_SRV (C++ enumerator), 139
- ESP\_BLE\_DUPLICATE\_SCAN\_EXCEPTIONAL\_MESH\_BEACON\_TYPE (C++ enumerator), 139
- ESP\_BLE\_DUPLICATE\_SCAN\_EXCEPTIONAL\_MESH\_LINK\_ID\_LIST (C++ enumerator), 139
- ESP\_BLE\_DUPLICATE\_SCAN\_EXCEPTIONAL\_MESH\_PROXY\_SRV (C++ enumerator), 139
- ESP\_BLE\_ENC\_KEY\_MASK (C 宏), 109
- ESP\_BLE\_EVT\_CONN\_ADV (C++ enumerator), 138
- ESP\_BLE\_EVT\_CONN\_DIR\_ADV (C++ enumerator), 138
- ESP\_BLE\_EVT\_DISC\_ADV (C++ enumerator), 138
- ESP\_BLE\_EVT\_NON\_CONN\_ADV (C++ enumerator), 138
- ESP\_BLE\_EVT\_SCAN\_RSP (C++ enumerator), 138
- esp\_ble\_evt\_type\_t (C++ enum), 138
- esp\_ble\_gap\_add\_duplicate\_scan\_exceptional\_device (C++ function), 116
- esp\_ble\_gap\_cb\_param\_t (C++ union), 120
- esp\_ble\_gap\_cb\_param\_t::adv\_data\_cmpl (C++ member), 120
- esp\_ble\_gap\_cb\_param\_t::adv\_data\_raw\_cmpl (C++ member), 120
- esp\_ble\_gap\_cb\_param\_t::adv\_start\_cmpl (C++ member), 120
- esp\_ble\_gap\_cb\_param\_t::adv\_stop\_cmpl (C++ member), 120
- esp\_ble\_gap\_cb\_param\_t::ble\_adv\_data\_cmpl\_evt\_param

(C++ class), 121  
 esp\_ble\_gap\_cb\_param\_t::ble\_adv\_data\_cmpl\_evt\_param (C++ member), 121  
 esp\_ble\_gap\_cb\_param\_t::ble\_adv\_data\_raw\_cmpl\_evt\_param (C++ class), 121  
 esp\_ble\_gap\_cb\_param\_t::ble\_adv\_data\_raw\_cmpl\_evt\_param (C++ member), 121  
 esp\_ble\_gap\_cb\_param\_t::ble\_adv\_start\_cmpl\_evt\_param (C++ class), 121  
 esp\_ble\_gap\_cb\_param\_t::ble\_adv\_start\_cmpl\_evt\_param (C++ member), 121  
 esp\_ble\_gap\_cb\_param\_t::ble\_adv\_stop\_cmpl\_evt\_param (C++ class), 121  
 esp\_ble\_gap\_cb\_param\_t::ble\_adv\_stop\_cmpl\_evt\_param (C++ member), 121  
 esp\_ble\_gap\_cb\_param\_t::ble\_clear\_bond\_cmpl\_evt\_param (C++ class), 121  
 esp\_ble\_gap\_cb\_param\_t::ble\_clear\_bond\_cmpl\_evt\_param (C++ member), 122  
 esp\_ble\_gap\_cb\_param\_t::ble\_get\_bond\_dev\_name\_cmpl\_evt\_param (C++ class), 122  
 esp\_ble\_gap\_cb\_param\_t::ble\_get\_bond\_dev\_name\_cmpl\_evt\_param (C++ member), 122  
 esp\_ble\_gap\_cb\_param\_t::ble\_get\_bond\_dev\_name\_cmpl\_evt\_num (C++ member), 122  
 esp\_ble\_gap\_cb\_param\_t::ble\_get\_bond\_dev\_name\_cmpl\_evt\_num (C++ class), 123  
 esp\_ble\_gap\_cb\_param\_t::ble\_get\_bond\_dev\_name\_cmpl\_evt\_num (C++ member), 122  
 esp\_ble\_gap\_cb\_param\_t::ble\_get\_bond\_dev\_name\_cmpl\_evt\_num (C++ class), 124  
 esp\_ble\_gap\_cb\_param\_t::ble\_local\_privacy\_cmpl\_evt\_param (C++ class), 122  
 esp\_ble\_gap\_cb\_param\_t::ble\_local\_privacy\_cmpl\_evt\_param (C++ member), 122  
 esp\_ble\_gap\_cb\_param\_t::ble\_pkt\_data\_len\_cmpl\_evt\_param (C++ class), 122  
 esp\_ble\_gap\_cb\_param\_t::ble\_pkt\_data\_len\_cmpl\_evt\_param (C++ member), 122  
 esp\_ble\_gap\_cb\_param\_t::ble\_pkt\_data\_len\_cmpl\_evt\_param (C++ class), 124  
 esp\_ble\_gap\_cb\_param\_t::ble\_pkt\_data\_len\_cmpl\_evt\_param (C++ member), 124  
 esp\_ble\_gap\_cb\_param\_t::ble\_pkt\_data\_len\_cmpl\_evt\_param (C++ class), 124  
 esp\_ble\_gap\_cb\_param\_t::ble\_pkt\_data\_len\_cmpl\_evt\_param (C++ member), 124  
 esp\_ble\_gap\_cb\_param\_t::ble\_read\_rssi\_cmpl\_evt\_param (C++ class), 122  
 esp\_ble\_gap\_cb\_param\_t::ble\_read\_rssi\_cmpl\_evt\_param (C++ member), 122  
 esp\_ble\_gap\_cb\_param\_t::ble\_read\_rssi\_cmpl\_evt\_param (C++ class), 120  
 esp\_ble\_gap\_cb\_param\_t::ble\_read\_rssi\_cmpl\_evt\_param (C++ member), 122  
 esp\_ble\_gap\_cb\_param\_t::ble\_read\_rssi\_cmpl\_evt\_param (C++ class), 124  
 esp\_ble\_gap\_cb\_param\_t::ble\_read\_rssi\_cmpl\_evt\_param (C++ member), 124  
 esp\_ble\_gap\_cb\_param\_t::ble\_remove\_bond\_cmpl\_evt\_param (C++ class), 122  
 esp\_ble\_gap\_cb\_param\_t::ble\_remove\_bond\_cmpl\_evt\_param (C++ member), 123  
 esp\_ble\_gap\_cb\_param\_t::ble\_remove\_bond\_cmpl\_evt\_param (C++ class), 124  
 esp\_ble\_gap\_cb\_param\_t::ble\_remove\_bond\_cmpl\_evt\_param (C++ member), 123  
 esp\_ble\_gap\_cb\_param\_t::ble\_scan\_param\_cmpl\_evt\_param (C++ class), 123  
 esp\_ble\_gap\_cb\_param\_t::ble\_scan\_param\_cmpl\_evt\_param (C++ member), 123  
 esp\_ble\_gap\_cb\_param\_t::ble\_scan\_result\_evt\_param (C++ class), 123  
 esp\_ble\_gap\_cb\_param\_t::ble\_scan\_result\_evt\_param (C++ member), 124

- (C++ member), 124
- esp\_ble\_gap\_cb\_param\_t::ble\_update\_conn\_params (C++ function), 115
- esp\_ble\_gap\_cb\_param\_t::ble\_update\_duplicate\_scan\_params (C++ member), 125
- esp\_ble\_gap\_cb\_param\_t::ble\_update\_duplicate\_scan\_exceptional\_dev\_list (C++ class), 125
- esp\_ble\_gap\_cb\_param\_t::ble\_update\_duplicate\_scan\_exceptional\_dev\_list\_cmpl\_data\_param::device\_info (C++ member), 125
- esp\_ble\_gap\_cb\_param\_t::ble\_update\_duplicate\_scan\_exceptional\_dev\_list\_cmpl\_data\_param::length (C++ member), 125
- esp\_ble\_gap\_cb\_param\_t::ble\_update\_duplicate\_scan\_exceptional\_dev\_list\_cmpl\_data\_param::status (C++ member), 125
- esp\_ble\_gap\_cb\_param\_t::ble\_update\_duplicate\_scan\_exceptional\_dev\_list\_cmpl\_data\_param::subcode (C++ member), 125
- esp\_ble\_gap\_cb\_param\_t::ble\_update\_whitelist (C++ class), 125
- esp\_ble\_gap\_cb\_param\_t::ble\_update\_whitelist\_cmpl\_evt\_param::status (C++ member), 125
- esp\_ble\_gap\_cb\_param\_t::ble\_update\_whitelist\_cmpl\_evt\_param::rsp (C++ member), 125
- esp\_ble\_gap\_cb\_param\_t::clear\_bond\_dev (C++ member), 121
- esp\_ble\_gap\_cb\_param\_t::get\_bond\_dev\_cmpl (C++ member), 121
- esp\_ble\_gap\_cb\_param\_t::local\_privacy\_cmpl (C++ member), 121
- esp\_ble\_gap\_cb\_param\_t::pkt\_data\_len\_cmpl (C++ member), 120
- esp\_ble\_gap\_cb\_param\_t::read\_rssi\_cmpl (C++ member), 121
- esp\_ble\_gap\_cb\_param\_t::remove\_bond\_dev\_cmpl (C++ member), 121
- esp\_ble\_gap\_cb\_param\_t::scan\_param\_cmpl (C++ member), 120
- esp\_ble\_gap\_cb\_param\_t::scan\_rsp\_data\_cmpl (C++ member), 120
- esp\_ble\_gap\_cb\_param\_t::scan\_rsp\_data\_rsp\_cmpl (C++ member), 120
- esp\_ble\_gap\_cb\_param\_t::scan\_rst (C++ member), 120
- esp\_ble\_gap\_cb\_param\_t::scan\_start\_cmpl (C++ member), 120
- esp\_ble\_gap\_cb\_param\_t::scan\_stop\_cmpl (C++ member), 120
- esp\_ble\_gap\_cb\_param\_t::set\_rand\_addr\_cmpl (C++ member), 120
- esp\_ble\_gap\_cb\_param\_t::update\_conn\_params (C++ member), 120
- esp\_ble\_gap\_cb\_param\_t::update\_duplicate\_scan\_exceptional\_dev\_list\_cmpl (C++ member), 121
- esp\_ble\_gap\_cb\_param\_t::update\_whitelist\_cmpl (C++ member), 121
- esp\_ble\_gap\_clean\_duplicate\_scan\_exceptional\_dev\_list (C++ function), 117
- esp\_ble\_gap\_clear\_rand\_addr (C++ function), 115
- esp\_ble\_gap\_config\_adv\_data (C++ function), 113
- esp\_ble\_gap\_config\_adv\_data\_raw (C++ function), 116
- esp\_ble\_gap\_get\_local\_used\_addr (C++ function), 116
- esp\_ble\_gap\_get\_whitelist\_size (C++ function), 116
- esp\_ble\_gap\_read\_rssi (C++ function), 116
- esp\_ble\_gap\_set\_device\_name (C++ function), 116
- esp\_ble\_gap\_set\_pkt\_data\_len (C++ function), 114
- esp\_ble\_gap\_set\_prefer\_conn\_params (C++ function), 115
- esp\_ble\_gap\_set\_rand\_addr (C++ function), 114
- esp\_ble\_gap\_set\_scan\_params (C++ function), 113
- esp\_ble\_gap\_set\_security\_param (C++ function), 117
- esp\_ble\_gap\_start\_advertising (C++ function), 114
- esp\_ble\_gap\_start\_scanning (C++ function), 114
- esp\_ble\_gap\_stop\_advertising (C++ function), 114
- esp\_ble\_gap\_stop\_scanning (C++ function), 114
- esp\_ble\_gap\_update\_conn\_params (C++ function), 114
- esp\_ble\_gap\_update\_whitelist (C++ function), 115
- esp\_ble\_gattc\_app\_register (C++ function), 162
- esp\_ble\_gattc\_app\_unregister (C++ function), 162
- esp\_ble\_gattc\_app\_start\_data\_cmpl (C++ function), 169
- esp\_ble\_gattc\_cache\_clean (C++ function), 170
- esp\_ble\_gattc\_cache\_get\_addr\_list (C++ function), 169
- esp\_ble\_gattc\_cache\_refresh (C++ function), 169
- esp\_ble\_gattc\_cb\_param\_t (C++ union), 170
- esp\_ble\_gattc\_cb\_param\_t::cfg\_mtu (C++ member), 170

`esp_ble_gattc_cb_param_t::close` (C++ member), 170

`esp_ble_gattc_cb_param_t::congest` (C++ member), 170

`esp_ble_gattc_cb_param_t::connect` (C++ member), 170

`esp_ble_gattc_cb_param_t::dis_srvc_cmpl` (C++ member), 171

`esp_ble_gattc_cb_param_t::disconnect` (C++ member), 170

`esp_ble_gattc_cb_param_t::exec_cmpl` (C++ member), 170

`esp_ble_gattc_cb_param_t::gattc_cfg_mtuevt_param_t` (C++ class), 171

`esp_ble_gattc_cb_param_t::gattc_cfg_mtuevt_param_t::conn_param_t` (C++ member), 171

`esp_ble_gattc_cb_param_t::gattc_cfg_mtuevt_param_t::mtb_param_t` (C++ member), 173

`esp_ble_gattc_cb_param_t::gattc_cfg_mtuevt_param_t::status_param_t` (C++ member), 173

`esp_ble_gattc_cb_param_t::gattc_close_evt_param_t` (C++ class), 171

`esp_ble_gattc_cb_param_t::gattc_close_evt_param_t::conn_param_t` (C++ member), 173

`esp_ble_gattc_cb_param_t::gattc_close_evt_param_t::desc_param_t` (C++ member), 173

`esp_ble_gattc_cb_param_t::gattc_close_evt_param_t::mtb_param_t` (C++ member), 173

`esp_ble_gattc_cb_param_t::gattc_close_evt_param_t::status_param_t` (C++ member), 173

`esp_ble_gattc_cb_param_t::gattc_close_evt_param_t::strat_param_t` (C++ class), 173

`esp_ble_gattc_cb_param_t::gattc_congest_evt_param_t` (C++ class), 171

`esp_ble_gattc_cb_param_t::gattc_congest_evt_param_t::congest_param_t` (C++ member), 173

`esp_ble_gattc_cb_param_t::gattc_congest_evt_param_t::conn_param_t` (C++ member), 173

`esp_ble_gattc_cb_param_t::gattc_connect_evt_param_t` (C++ class), 171

`esp_ble_gattc_cb_param_t::gattc_connect_evt_param_t::conn_param_t` (C++ member), 173

`esp_ble_gattc_cb_param_t::gattc_connect_evt_param_t::mtb_param_t` (C++ member), 173

`esp_ble_gattc_cb_param_t::gattc_dis_srvc_cmpl_evt_param_t` (C++ class), 172

`esp_ble_gattc_cb_param_t::gattc_dis_srvc_cmpl_evt_param_t::conn_param_t` (C++ member), 173

`esp_ble_gattc_cb_param_t::gattc_dis_srvc_cmpl_evt_param_t::dis_srvc_cmpl_param_t` (C++ member), 174

`esp_ble_gattc_cb_param_t::gattc_disconnect_evt_param_t` (C++ class), 172

`esp_ble_gattc_cb_param_t::gattc_disconnect_evt_param_t::conn_param_t` (C++ member), 174

`esp_ble_gattc_cb_param_t::gattc_disconnect_evt_param_t::dis_srvc_cmpl_param_t` (C++ member), 174

`esp_ble_gattc_cb_param_t::gattc_disconnect_evt_param_t::mtb_param_t` (C++ member), 174

`esp_ble_gattc_cb_param_t::gattc_disconnect_evt_param_t::status_param_t` (C++ member), 174



esp\_ble\_gattc\_cb\_param\_t::gattc\_reg\_evt\_param: gattc\_cb\_param\_t::notify (C++  
 (C++ class), 174 member), 170  
 esp\_ble\_gattc\_cb\_param\_t::gattc\_reg\_evt\_param: gattc\_cb\_param\_t::open (C++  
 (C++ member), 174 member), 170  
 esp\_ble\_gattc\_cb\_param\_t::gattc\_reg\_evt\_param: gattc\_cb\_param\_t::queue\_full  
 (C++ member), 174 (C++ member), 171  
 esp\_ble\_gattc\_cb\_param\_t::gattc\_reg\_for\_notify: gattc\_cb\_param\_t::read (C++  
 (C++ class), 174 member), 170  
 esp\_ble\_gattc\_cb\_param\_t::gattc\_reg\_for\_notify: gattc\_cb\_param\_t::handle\_reg (C++  
 (C++ member), 174 member), 170  
 esp\_ble\_gattc\_cb\_param\_t::gattc\_reg\_for\_notify: gattc\_cb\_param\_t::status\_reg\_for\_notify  
 (C++ member), 174 (C++ member), 170  
 esp\_ble\_gattc\_cb\_param\_t::gattc\_search\_evt\_param: gattc\_cb\_param\_t::search\_cmpl  
 (C++ class), 174 (C++ member), 170  
 esp\_ble\_gattc\_cb\_param\_t::gattc\_search\_evt\_param: gattc\_cb\_param\_t::search\_res  
 (C++ member), 174 (C++ member), 170  
 esp\_ble\_gattc\_cb\_param\_t::gattc\_search\_evt\_param: gattc\_cb\_param\_t::search\_res\_count  
 (C++ member), 174 (C++ member), 170  
 esp\_ble\_gattc\_cb\_param\_t::gattc\_search\_evt\_param: gattc\_cb\_param\_t::start\_srvc\_chg  
 (C++ member), 174 (C++ member), 170  
 esp\_ble\_gattc\_cb\_param\_t::gattc\_search\_evt\_param: gattc\_cb\_param\_t::unreg\_for\_notify  
 (C++ class), 174 (C++ member), 170  
 esp\_ble\_gattc\_cb\_param\_t::gattc\_search\_evt\_param: gattc\_cb\_param\_t::write (C++  
 (C++ member), 175 member), 170  
 esp\_ble\_gattc\_cb\_param\_t::gattc\_search\_evt\_param: close (C++ function), 163  
 (C++ member), 175 esp\_ble\_gattc\_execute\_write (C++ func-  
 tion), 168  
 esp\_ble\_gattc\_cb\_param\_t::gattc\_search\_res\_evt\_param: is\_primary  
 (C++ member), 175 esp\_ble\_gattc\_get\_all\_char (C++ function),  
 164  
 esp\_ble\_gattc\_cb\_param\_t::gattc\_search\_res\_evt\_param: srvc\_id  
 (C++ member), 175 esp\_ble\_gattc\_get\_all\_descr (C++ func-  
 tion), 164  
 esp\_ble\_gattc\_cb\_param\_t::gattc\_search\_res\_evt\_param: start\_handle  
 (C++ member), 175 esp\_ble\_gattc\_get\_attr\_count (C++ func-  
 tion), 166  
 esp\_ble\_gattc\_cb\_param\_t::gattc\_set\_assoc\_add\_evt\_param  
 (C++ class), 175 esp\_ble\_gattc\_get\_char\_by\_uuid (C++  
 function), 164  
 esp\_ble\_gattc\_cb\_param\_t::gattc\_set\_assoc\_add\_evt\_param: status  
 (C++ member), 175 esp\_ble\_gattc\_get\_db (C++ function), 166  
 esp\_ble\_gattc\_cb\_param\_t::gattc\_srvc\_chg\_evt\_param: gattc\_get\_descr\_by\_char\_handle  
 (C++ class), 175 (C++ function), 165  
 esp\_ble\_gattc\_cb\_param\_t::gattc\_srvc\_chg\_evt\_param: gattc\_get\_descr\_by\_uuid (C++  
 (C++ member), 175 function), 165  
 esp\_ble\_gattc\_cb\_param\_t::gattc\_unreg\_evt\_param: include\_service  
 (C++ class), 175 (C++ function), 165  
 esp\_ble\_gattc\_cb\_param\_t::gattc\_unreg\_evt\_param: srvc\_handle (C++ function),  
 (C++ member), 175 163  
 esp\_ble\_gattc\_cb\_param\_t::gattc\_unreg\_evt\_param: param (C++ function), 163  
 (C++ member), 175 esp\_ble\_gattc\_prepare\_write (C++ func-  
 tion), 168  
 esp\_ble\_gattc\_cb\_param\_t::gattc\_write\_evt\_param: prepare\_write\_char\_descr  
 (C++ class), 175 (C++ function), 168  
 esp\_ble\_gattc\_cb\_param\_t::gattc\_write\_evt\_param: read\_by\_type (C++ function),  
 (C++ member), 175 167  
 esp\_ble\_gattc\_cb\_param\_t::gattc\_write\_evt\_param: handle  
 (C++ member), 175 esp\_ble\_gattc\_read\_char (C++ function), 166  
 esp\_ble\_gattc\_cb\_param\_t::gattc\_write\_evt\_param: read\_char\_descr (C++ func-  
 (C++ member), 175 tion), 167  
 esp\_ble\_gattc\_cb\_param\_t::gattc\_write\_evt\_param: read\_multiple (C++ func-  
 (C++ member), 175 tion), 167  
 esp\_ble\_gattc\_cb\_param\_t::get\_addr\_list esp\_ble\_gattc\_register\_callback (C++  
 (C++ member), 171 function), 162



esp\_ble\_gatts\_cb\_param\_t::gatts\_connect\_evt\_param (C++ class), 157  
 esp\_ble\_gatts\_cb\_param\_t::gatts\_connect\_evt\_param (C++ member), 157  
 esp\_ble\_gatts\_cb\_param\_t::gatts\_create\_evt\_param (C++ class), 157  
 esp\_ble\_gatts\_cb\_param\_t::gatts\_create\_evt\_param (C++ member), 157  
 esp\_ble\_gatts\_cb\_param\_t::gatts\_delete\_evt\_param (C++ class), 157  
 esp\_ble\_gatts\_cb\_param\_t::gatts\_delete\_evt\_param (C++ member), 157  
 esp\_ble\_gatts\_cb\_param\_t::gatts\_disconnect\_evt\_param (C++ class), 157  
 esp\_ble\_gatts\_cb\_param\_t::gatts\_disconnect\_evt\_param (C++ member), 157  
 esp\_ble\_gatts\_cb\_param\_t::gatts\_exec\_write\_evt\_param (C++ class), 157  
 esp\_ble\_gatts\_cb\_param\_t::gatts\_exec\_write\_evt\_param (C++ member), 158  
 esp\_ble\_gatts\_cb\_param\_t::gatts\_mtu\_evt\_param (C++ class), 158  
 esp\_ble\_gatts\_cb\_param\_t::gatts\_mtu\_evt\_param (C++ member), 158  
 esp\_ble\_gatts\_cb\_param\_t::gatts\_open\_evt\_param (C++ class), 158  
 esp\_ble\_gatts\_cb\_param\_t::gatts\_open\_evt\_param (C++ member), 158  
 esp\_ble\_gatts\_cb\_param\_t::gatts\_read\_evt\_param (C++ class), 158  
 esp\_ble\_gatts\_cb\_param\_t::gatts\_read\_evt\_param (C++ member), 158  
 esp\_ble\_gatts\_cb\_param\_t::gatts\_send\_service\_changed\_evt\_param (C++ class), 159  
 esp\_ble\_gatts\_cb\_param\_t::gatts\_send\_service\_changed\_evt\_param (C++ member), 159  
 esp\_ble\_gatts\_cb\_param\_t::gatts\_set\_attr\_val\_evt\_param (C++ class), 159  
 esp\_ble\_gatts\_cb\_param\_t::gatts\_set\_attr\_val\_evt\_param (C++ member), 159  
 esp\_ble\_gatts\_cb\_param\_t::gatts\_start\_evt\_param (C++ class), 159  
 esp\_ble\_gatts\_cb\_param\_t::gatts\_start\_evt\_param (C++ member), 159  
 esp\_ble\_gatts\_cb\_param\_t::gatts\_stop\_evt\_param (C++ class), 159  
 esp\_ble\_gatts\_cb\_param\_t::gatts\_stop\_evt\_param (C++ member), 160  
 esp\_ble\_gatts\_cb\_param\_t::gatts\_write\_evt\_param (C++ class), 160  
 esp\_ble\_gatts\_cb\_param\_t::gatts\_write\_evt\_param (C++ member), 160

- esp\_ble\_gatts\_cb\_param\_t::gatts\_write\_data (C++ member), 160  
 esp\_ble\_gatts\_cb\_param\_t::gatts\_write\_data (C++ member), 160  
 esp\_ble\_gatts\_cb\_param\_t::gatts\_write\_data (C++ member), 160  
 esp\_ble\_gatts\_cb\_param\_t::mtu (C++ member), 154  
 esp\_ble\_gatts\_cb\_param\_t::open (C++ member), 154  
 esp\_ble\_gatts\_cb\_param\_t::read (C++ member), 154  
 esp\_ble\_gatts\_cb\_param\_t::reg (C++ member), 154  
 esp\_ble\_gatts\_cb\_param\_t::rsp (C++ member), 154  
 esp\_ble\_gatts\_cb\_param\_t::service\_change (C++ member), 155  
 esp\_ble\_gatts\_cb\_param\_t::set\_attr\_val (C++ member), 154  
 esp\_ble\_gatts\_cb\_param\_t::start (C++ member), 154  
 esp\_ble\_gatts\_cb\_param\_t::stop (C++ member), 154  
 esp\_ble\_gatts\_cb\_param\_t::write (C++ member), 154  
 esp\_ble\_gatts\_close (C++ function), 153  
 esp\_ble\_gatts\_create\_attr\_tab (C++ function), 151  
 esp\_ble\_gatts\_create\_service (C++ function), 150  
 esp\_ble\_gatts\_delete\_service (C++ function), 152  
 esp\_ble\_gatts\_get\_attr\_value (C++ function), 153  
 esp\_ble\_gatts\_open (C++ function), 153  
 esp\_ble\_gatts\_register\_callback (C++ function), 150  
 esp\_ble\_gatts\_send\_indicate (C++ function), 152  
 esp\_ble\_gatts\_send\_response (C++ function), 152  
 esp\_ble\_gatts\_send\_service\_change\_indicate (C++ function), 153  
 esp\_ble\_gatts\_set\_attr\_value (C++ function), 153  
 esp\_ble\_gatts\_start\_service (C++ function), 152  
 esp\_ble\_gatts\_stop\_service (C++ function), 152  
 esp\_ble\_get\_bond\_device\_list (C++ function), 118  
 esp\_ble\_get\_bond\_device\_num (C++ function), 118  
 esp\_ble\_get\_current\_conn\_params (C++ function), 119  
 ESP\_BLE\_HOST\_STATUS\_CHECK (C 宏), 282  
 ESP\_BLE\_HOST\_STATUS\_ENABLED (C 宏), 282  
 ESP\_BLE\_HOST\_STATUS\_DISABLED (C 宏), 110  
 esp\_ble\_io\_cap\_t (C++ type), 133  
 ESP\_BLE\_LOCAL\_ID\_KEYS\_PARAM (C 宏), 109  
 esp\_ble\_key\_mask\_t (C++ type), 110  
 esp\_ble\_key\_val (C++ class), 130  
 esp\_ble\_key\_t::bd\_addr (C++ member), 130  
 esp\_ble\_key\_t::key\_type (C++ member), 130  
 esp\_ble\_key\_t::p\_key\_value (C++ member), 130  
 esp\_ble\_key\_type\_t (C++ type), 133  
 esp\_ble\_key\_value\_t (C++ union), 119  
 esp\_ble\_key\_value\_t::lcsr\_key (C++ member), 119  
 esp\_ble\_key\_value\_t::lenc\_key (C++ member), 119  
 esp\_ble\_key\_value\_t::pcsr\_key (C++ member), 119  
 esp\_ble\_key\_value\_t::penc\_key (C++ member), 119  
 esp\_ble\_key\_value\_t::pid\_key (C++ member), 119  
 esp\_ble\_lcsr\_keys (C++ class), 129  
 esp\_ble\_lcsr\_keys::counter (C++ member), 129  
 esp\_ble\_lcsr\_keys::csr\_key (C++ member), 129  
 esp\_ble\_lcsr\_keys::div (C++ member), 129  
 esp\_ble\_lcsr\_keys::sec\_level (C++ member), 129  
 esp\_ble\_lenc\_keys\_t (C++ class), 128  
 esp\_ble\_lenc\_keys\_t::div (C++ member), 129  
 esp\_ble\_lenc\_keys\_t::key\_size (C++ member), 129  
 esp\_ble\_lenc\_keys\_t::ltk (C++ member), 129  
 esp\_ble\_lenc\_keys\_t::sec\_level (C++ member), 129  
 ESP\_BLE\_LINK\_KEY\_MASK (C 宏), 110  
 esp\_ble\_local\_id\_keys\_t (C++ class), 130  
 esp\_ble\_local\_id\_keys\_t::dhk (C++ member), 130  
 esp\_ble\_local\_id\_keys\_t::ir (C++ member), 130  
 esp\_ble\_local\_id\_keys\_t::irk (C++ member), 130  
 ESP\_BLE\_MESH\_ACTUATOR\_BLOCKED\_ERROR (C 宏), 382  
 ESP\_BLE\_MESH\_ACTUATOR\_BLOCKED\_WARNING (C 宏), 382  
 ESP\_BLE\_MESH\_ADDR\_ALL\_NODES (C 宏), 282  
 ESP\_BLE\_MESH\_ADDR\_FRIENDS (C 宏), 282  
 ESP\_BLE\_MESH\_ADDR\_IS\_GROUP (C 宏), 283  
 ESP\_BLE\_MESH\_ADDR\_IS\_RFU (C 宏), 283  
 ESP\_BLE\_MESH\_ADDR\_IS\_UNICAST (C 宏), 283  
 ESP\_BLE\_MESH\_ADDR\_IS\_VIRTUAL (C 宏), 283  
 ESP\_BLE\_MESH\_ADDR\_PROXIES (C 宏), 282  
 ESP\_BLE\_MESH\_ADDR\_RELAYS (C 宏), 282

- ESP\_BLE\_MESH\_ADDR\_TYPE\_PUBLIC (C 宏), 284
- ESP\_BLE\_MESH\_ADDR\_TYPE\_RANDOM (C 宏), 284
- ESP\_BLE\_MESH\_ADDR\_TYPE\_RPA\_PUBLIC (C 宏), 284
- ESP\_BLE\_MESH\_ADDR\_TYPE\_RPA\_RANDOM (C 宏), 284
- esp\_ble\_mesh\_addr\_type\_t (C++ type), 296
- ESP\_BLE\_MESH\_ADDR\_UNASSIGNED (C 宏), 282
- ESP\_BLE\_MESH\_BATTERY\_LOW\_ERROR (C 宏), 381
- ESP\_BLE\_MESH\_BATTERY\_LOW\_WARNING (C 宏), 381
- esp\_ble\_mesh\_bd\_addr\_t (C++ type), 296
- ESP\_BLE\_MESH\_BEACON\_DISABLED (C 宏), 282
- ESP\_BLE\_MESH\_BEACON\_ENABLED (C 宏), 282
- ESP\_BLE\_MESH\_BEEP (C++ enumerator), 298
- ESP\_BLE\_MESH\_BLINK (C++ enumerator), 298
- esp\_ble\_mesh\_cb\_t (C++ type), 296
- esp\_ble\_mesh\_cb\_type\_t (C++ enum), 297
- esp\_ble\_mesh\_cfg\_app\_key\_add\_t (C++ class), 329
- esp\_ble\_mesh\_cfg\_app\_key\_add\_t::app\_idx (C++ member), 329
- esp\_ble\_mesh\_cfg\_app\_key\_add\_t::app\_key (C++ member), 329
- esp\_ble\_mesh\_cfg\_app\_key\_add\_t::net\_idx (C++ member), 329
- esp\_ble\_mesh\_cfg\_app\_key\_delete\_t (C++ class), 333
- esp\_ble\_mesh\_cfg\_app\_key\_delete\_t::app\_idx (C++ member), 333
- esp\_ble\_mesh\_cfg\_app\_key\_delete\_t::net\_idx (C++ member), 333
- esp\_ble\_mesh\_cfg\_app\_key\_get\_t (C++ class), 327
- esp\_ble\_mesh\_cfg\_app\_key\_get\_t::net\_idx (C++ member), 327
- esp\_ble\_mesh\_cfg\_app\_key\_list\_cb\_t (C++ class), 338
- esp\_ble\_mesh\_cfg\_app\_key\_list\_cb\_t::app\_idx (C++ member), 339
- esp\_ble\_mesh\_cfg\_app\_key\_list\_cb\_t::net\_idx (C++ member), 338
- esp\_ble\_mesh\_cfg\_app\_key\_list\_cb\_t::status (C++ member), 338
- esp\_ble\_mesh\_cfg\_app\_key\_status\_cb\_t (C++ class), 336
- esp\_ble\_mesh\_cfg\_app\_key\_status\_cb\_t::app\_idx (C++ member), 336
- esp\_ble\_mesh\_cfg\_app\_key\_status\_cb\_t::net\_idx (C++ member), 336
- esp\_ble\_mesh\_cfg\_app\_key\_status\_cb\_t::status (C++ member), 336
- esp\_ble\_mesh\_cfg\_app\_key\_update\_t (C++ class), 332
- esp\_ble\_mesh\_cfg\_app\_key\_update\_t::app\_idx (C++ member), 333
- esp\_ble\_mesh\_cfg\_app\_key\_update\_t::app\_key (C++ member), 333
- (C++ member), 333
- esp\_ble\_mesh\_cfg\_app\_key\_update\_t::net\_idx (C++ member), 333
- esp\_ble\_mesh\_cfg\_beacon\_set\_t (C++ class), 328
- esp\_ble\_mesh\_cfg\_beacon\_set\_t::beacon (C++ member), 328
- esp\_ble\_mesh\_cfg\_beacon\_status\_cb\_t (C++ class), 334
- esp\_ble\_mesh\_cfg\_beacon\_status\_cb\_t::beacon (C++ member), 334
- esp\_ble\_mesh\_cfg\_client\_cb\_event\_t (C++ enum), 344
- esp\_ble\_mesh\_cfg\_client\_cb\_param\_t (C++ class), 340
- esp\_ble\_mesh\_cfg\_client\_cb\_param\_t::error\_code (C++ member), 340
- esp\_ble\_mesh\_cfg\_client\_cb\_param\_t::params (C++ member), 340
- esp\_ble\_mesh\_cfg\_client\_cb\_param\_t::status\_cb (C++ member), 340
- esp\_ble\_mesh\_cfg\_client\_cb\_t (C++ type), 343
- esp\_ble\_mesh\_cfg\_client\_common\_cb\_param\_t (C++ union), 323
- esp\_ble\_mesh\_cfg\_client\_common\_cb\_param\_t::app\_key (C++ member), 324
- esp\_ble\_mesh\_cfg\_client\_common\_cb\_param\_t::app\_key (C++ member), 324
- esp\_ble\_mesh\_cfg\_client\_common\_cb\_param\_t::beacon (C++ member), 323
- esp\_ble\_mesh\_cfg\_client\_common\_cb\_param\_t::comp\_d (C++ member), 323
- esp\_ble\_mesh\_cfg\_client\_common\_cb\_param\_t::default (C++ member), 323
- esp\_ble\_mesh\_cfg\_client\_common\_cb\_param\_t::friend (C++ member), 324
- esp\_ble\_mesh\_cfg\_client\_common\_cb\_param\_t::gatt\_p (C++ member), 323
- esp\_ble\_mesh\_cfg\_client\_common\_cb\_param\_t::heartb (C++ member), 324
- esp\_ble\_mesh\_cfg\_client\_common\_cb\_param\_t::heartb (C++ member), 324
- esp\_ble\_mesh\_cfg\_client\_common\_cb\_param\_t::kr\_ph (C++ member), 324
- esp\_ble\_mesh\_cfg\_client\_common\_cb\_param\_t::lpn\_ti (C++ member), 324
- esp\_ble\_mesh\_cfg\_client\_common\_cb\_param\_t::model (C++ member), 324
- esp\_ble\_mesh\_cfg\_client\_common\_cb\_param\_t::model (C++ member), 324
- esp\_ble\_mesh\_cfg\_client\_common\_cb\_param\_t::model (C++ member), 324
- esp\_ble\_mesh\_cfg\_client\_common\_cb\_param\_t::model (C++ member), 324
- esp\_ble\_mesh\_cfg\_client\_common\_cb\_param\_t::net\_tr

(C++ member), 324

esp\_ble\_mesh\_cfg\_client\_common\_cb\_params\_t (C++ member), 324

esp\_ble\_mesh\_cfg\_client\_common\_cb\_params\_t (C++ member), 324

esp\_ble\_mesh\_cfg\_client\_common\_cb\_params\_t (C++ member), 324

ESP\_BLE\_MESH\_CFG\_CLIENT\_EVT\_MAX (C++ enumerator), 344

ESP\_BLE\_MESH\_CFG\_CLIENT\_GET\_STATE\_EVT (C++ enumerator), 344

esp\_ble\_mesh\_cfg\_client\_get\_state\_t (C++ union), 321

esp\_ble\_mesh\_cfg\_client\_get\_state\_t::app\_key\_get (C++ member), 322

esp\_ble\_mesh\_cfg\_client\_get\_state\_t::comp\_data\_mesh (C++ member), 321

esp\_ble\_mesh\_cfg\_client\_get\_state\_t::krmphase\_get (C++ member), 322

esp\_ble\_mesh\_cfg\_client\_get\_state\_t::lpspdlle\_get (C++ member), 322

esp\_ble\_mesh\_cfg\_client\_get\_state\_t::model\_pub\_mesh (C++ member), 321

esp\_ble\_mesh\_cfg\_client\_get\_state\_t::node\_identif (C++ member), 322

esp\_ble\_mesh\_cfg\_client\_get\_state\_t::sigmodel\_mesh\_get (C++ member), 322

esp\_ble\_mesh\_cfg\_client\_get\_state\_t::sigmodel\_mesh\_get (C++ member), 321

esp\_ble\_mesh\_cfg\_client\_get\_state\_t::sigmodel\_mesh\_get (C++ member), 322

ESP\_BLE\_MESH\_CFG\_CLIENT\_PUBLISH\_EVT (C++ enumerator), 344

ESP\_BLE\_MESH\_CFG\_CLIENT\_SET\_STATE\_EVT (C++ enumerator), 344

esp\_ble\_mesh\_cfg\_client\_set\_state\_t (C++ union), 322

esp\_ble\_mesh\_cfg\_client\_set\_state\_t::app\_key\_add (C++ member), 322

esp\_ble\_mesh\_cfg\_client\_set\_state\_t::app\_key\_delete (C++ member), 323

esp\_ble\_mesh\_cfg\_client\_set\_state\_t::app\_key\_update (C++ member), 323

esp\_ble\_mesh\_cfg\_client\_set\_state\_t::beepble (C++ member), 322

esp\_ble\_mesh\_cfg\_client\_set\_state\_t::default\_ttl\_set (C++ member), 322

esp\_ble\_mesh\_cfg\_client\_set\_state\_t::frispdlle (C++ member), 322

esp\_ble\_mesh\_cfg\_client\_set\_state\_t::gateway\_mesh (C++ member), 322

esp\_ble\_mesh\_cfg\_client\_set\_state\_t::hospblet\_mesh\_set (C++ member), 323

esp\_ble\_mesh\_cfg\_client\_set\_state\_t::hospblet\_mesh\_set (C++ member), 323

esp\_ble\_mesh\_cfg\_client\_set\_state\_t::kr\_phase\_set (C++ member), 323

esp\_ble\_mesh\_cfg\_client\_set\_state\_t::model\_app\_bi (C++ member), 322

esp\_ble\_mesh\_cfg\_client\_set\_state\_t::model\_app\_un (C++ member), 323

esp\_ble\_mesh\_cfg\_client\_set\_state\_t::model\_pub\_se (C++ member), 322

esp\_ble\_mesh\_cfg\_client\_set\_state\_t::model\_pub\_va (C++ member), 323

esp\_ble\_mesh\_cfg\_client\_set\_state\_t::model\_sub\_ad (C++ member), 322

esp\_ble\_mesh\_cfg\_client\_set\_state\_t::model\_sub\_de (C++ member), 322

esp\_ble\_mesh\_cfg\_client\_set\_state\_t::model\_sub\_de (C++ member), 323

esp\_ble\_mesh\_cfg\_client\_set\_state\_t::model\_sub\_ov (C++ member), 323

esp\_ble\_mesh\_cfg\_client\_set\_state\_t::model\_sub\_va (C++ member), 323

esp\_ble\_mesh\_cfg\_client\_set\_state\_t::model\_sub\_va (C++ member), 323

esp\_ble\_mesh\_cfg\_client\_set\_state\_t::net\_key\_add (C++ member), 322

esp\_ble\_mesh\_cfg\_client\_set\_state\_t::net\_key\_dele (C++ member), 323

esp\_ble\_mesh\_cfg\_client\_set\_state\_t::net\_key\_upda (C++ member), 323

esp\_ble\_mesh\_cfg\_client\_set\_state\_t::net\_transmit (C++ member), 323

esp\_ble\_mesh\_cfg\_client\_set\_state\_t::node\_identit (C++ member), 323

ESP\_BLE\_MESH\_CFG\_CLIENT\_TIMEOUT\_EVT (C++ enumerator), 344

esp\_ble\_mesh\_cfg\_comp\_data\_status\_cb\_t (C++ class), 334

esp\_ble\_mesh\_cfg\_comp\_data\_status\_cb\_t::compositi (C++ member), 335

esp\_ble\_mesh\_cfg\_comp\_data\_status\_cb\_t::page (C++ member), 335

esp\_ble\_mesh\_cfg\_comp\_data\_status\_cb\_t::page (C++ class), 326

esp\_ble\_mesh\_cfg\_comp\_data\_status\_cb\_t::page (C++ member), 326

esp\_ble\_mesh\_cfg\_default\_ttl\_set\_t (C++ class), 328

esp\_ble\_mesh\_cfg\_default\_ttl\_set\_t::ttl (C++ member), 328

esp\_ble\_mesh\_cfg\_default\_ttl\_set\_t::ttl (C++ member), 328

esp\_ble\_mesh\_cfg\_default\_ttl\_status\_cb\_t (C++ class), 335

esp\_ble\_mesh\_cfg\_default\_ttl\_status\_cb\_t::default (C++ member), 335

esp\_ble\_mesh\_cfg\_friend\_set\_t (C++

class), 328  
 esp\_ble\_mesh\_cfg\_friend\_set\_t::friend\_status (C++ member), 328  
 esp\_ble\_mesh\_cfg\_friend\_status\_cb\_t (C++ class), 337  
 esp\_ble\_mesh\_cfg\_friend\_status\_cb\_t::friend\_status (C++ member), 337  
 esp\_ble\_mesh\_cfg\_gatt\_proxy\_set\_t (C++ class), 328  
 esp\_ble\_mesh\_cfg\_gatt\_proxy\_set\_t::gatt\_proxy (C++ member), 328  
 esp\_ble\_mesh\_cfg\_gatt\_proxy\_status\_cb\_t (C++ class), 335  
 esp\_ble\_mesh\_cfg\_gatt\_proxy\_status\_cb\_t::gatt\_proxy (C++ member), 335  
 esp\_ble\_mesh\_cfg\_hb\_pub\_status\_cb\_t (C++ class), 337  
 esp\_ble\_mesh\_cfg\_hb\_pub\_status\_cb\_t::count (C++ member), 337  
 esp\_ble\_mesh\_cfg\_hb\_pub\_status\_cb\_t::data (C++ member), 337  
 esp\_ble\_mesh\_cfg\_hb\_pub\_status\_cb\_t::features (C++ member), 337  
 esp\_ble\_mesh\_cfg\_hb\_pub\_status\_cb\_t::net\_idx (C++ member), 337  
 esp\_ble\_mesh\_cfg\_hb\_pub\_status\_cb\_t::period (C++ member), 337  
 esp\_ble\_mesh\_cfg\_hb\_pub\_status\_cb\_t::status (C++ member), 337  
 esp\_ble\_mesh\_cfg\_hb\_pub\_status\_cb\_t::ttl (C++ member), 337  
 esp\_ble\_mesh\_cfg\_hb\_sub\_status\_cb\_t (C++ class), 337  
 esp\_ble\_mesh\_cfg\_hb\_sub\_status\_cb\_t::count (C++ member), 338  
 esp\_ble\_mesh\_cfg\_hb\_sub\_status\_cb\_t::data (C++ member), 337  
 esp\_ble\_mesh\_cfg\_hb\_sub\_status\_cb\_t::max\_hop (C++ member), 338  
 esp\_ble\_mesh\_cfg\_hb\_sub\_status\_cb\_t::min\_hop (C++ member), 338  
 esp\_ble\_mesh\_cfg\_hb\_sub\_status\_cb\_t::period (C++ member), 338  
 esp\_ble\_mesh\_cfg\_hb\_sub\_status\_cb\_t::status (C++ member), 337  
 esp\_ble\_mesh\_cfg\_heartbeat\_pub\_set\_t (C++ class), 334  
 esp\_ble\_mesh\_cfg\_heartbeat\_pub\_set\_t::count (C++ member), 334  
 esp\_ble\_mesh\_cfg\_heartbeat\_pub\_set\_t::data (C++ member), 334  
 esp\_ble\_mesh\_cfg\_heartbeat\_pub\_set\_t::feature (C++ member), 334  
 esp\_ble\_mesh\_cfg\_heartbeat\_pub\_set\_t::net\_idx (C++ member), 334  
 esp\_ble\_mesh\_cfg\_heartbeat\_pub\_set\_t::period (C++ member), 334  
 esp\_ble\_mesh\_cfg\_heartbeat\_sub\_set\_t (C++ class), 334  
 esp\_ble\_mesh\_cfg\_heartbeat\_sub\_set\_t::period (C++ member), 334  
 esp\_ble\_mesh\_cfg\_heartbeat\_sub\_set\_t::src (C++ member), 334  
 esp\_ble\_mesh\_cfg\_kr\_phase\_get\_t (C++ class), 328  
 esp\_ble\_mesh\_cfg\_kr\_phase\_get\_t::net\_idx (C++ member), 328  
 esp\_ble\_mesh\_cfg\_kr\_phase\_set\_t (C++ class), 333  
 esp\_ble\_mesh\_cfg\_kr\_phase\_set\_t::net\_idx (C++ member), 334  
 esp\_ble\_mesh\_cfg\_kr\_phase\_set\_t::transition (C++ member), 334  
 esp\_ble\_mesh\_cfg\_kr\_phase\_status\_cb\_t (C++ class), 339  
 esp\_ble\_mesh\_cfg\_kr\_phase\_status\_cb\_t::net\_idx (C++ member), 339  
 esp\_ble\_mesh\_cfg\_kr\_phase\_status\_cb\_t::phase (C++ member), 339  
 esp\_ble\_mesh\_cfg\_kr\_phase\_status\_cb\_t::status (C++ member), 339  
 esp\_ble\_mesh\_cfg\_lpn\_polltimeout\_get\_t (C++ class), 328  
 esp\_ble\_mesh\_cfg\_lpn\_polltimeout\_get\_t::lpn\_addr (C++ member), 328  
 esp\_ble\_mesh\_cfg\_lpn\_pollto\_status\_cb\_t (C++ class), 339  
 esp\_ble\_mesh\_cfg\_lpn\_pollto\_status\_cb\_t::lpn\_addr (C++ member), 339  
 esp\_ble\_mesh\_cfg\_lpn\_pollto\_status\_cb\_t::poll\_time (C++ member), 339  
 esp\_ble\_mesh\_cfg\_mod\_app\_status\_cb\_t (C++ class), 336  
 esp\_ble\_mesh\_cfg\_mod\_app\_status\_cb\_t::app\_idx (C++ member), 337  
 esp\_ble\_mesh\_cfg\_mod\_app\_status\_cb\_t::company\_id (C++ member), 337  
 esp\_ble\_mesh\_cfg\_mod\_app\_status\_cb\_t::element\_addr (C++ member), 337  
 esp\_ble\_mesh\_cfg\_mod\_app\_status\_cb\_t::model\_id (C++ member), 337  
 esp\_ble\_mesh\_cfg\_mod\_app\_status\_cb\_t::status (C++ member), 337  
 esp\_ble\_mesh\_cfg\_model\_app\_bind\_t (C++ class), 329  
 esp\_ble\_mesh\_cfg\_model\_app\_bind\_t::company\_id (C++ member), 329  
 esp\_ble\_mesh\_cfg\_model\_app\_bind\_t::element\_addr (C++ member), 329  
 esp\_ble\_mesh\_cfg\_model\_app\_bind\_t::model\_app\_idx (C++ member), 329





(C++ member), 330  
esp\_ble\_mesh\_cfg\_model\_sub\_delete\_t::element\_addr (C++ member), 330  
esp\_ble\_mesh\_cfg\_model\_sub\_delete\_t::model\_id (C++ member), 330  
esp\_ble\_mesh\_cfg\_model\_sub\_delete\_t::sub\_delete (C++ member), 330  
esp\_ble\_mesh\_cfg\_model\_sub\_delete\_t::sub\_delete\_mesh\_cfg\_model\_sub\_va\_overwrite\_t::compan (C++ member), 331  
esp\_ble\_mesh\_cfg\_model\_sub\_delete\_t::sub\_delete\_mesh\_cfg\_model\_sub\_va\_overwrite\_t::elemen (C++ member), 331  
esp\_ble\_mesh\_cfg\_model\_sub\_delete\_t::sub\_delete\_mesh\_cfg\_model\_sub\_va\_overwrite\_t::label\_ (C++ member), 331  
esp\_ble\_mesh\_cfg\_model\_sub\_delete\_t::sub\_delete\_mesh\_cfg\_model\_sub\_va\_overwrite\_t::model\_ (C++ member), 331  
esp\_ble\_mesh\_cfg\_model\_sub\_delete\_t::sub\_delete\_mesh\_cfg\_net\_key\_add\_t (C++ class), 329  
esp\_ble\_mesh\_cfg\_model\_sub\_delete\_t::status (C++ member), 329  
esp\_ble\_mesh\_cfg\_model\_sub\_delete\_t::sub\_delete\_mesh\_cfg\_net\_key\_add\_t::net\_key (C++ member), 329  
esp\_ble\_mesh\_cfg\_model\_sub\_overwrite\_t esp\_ble\_mesh\_cfg\_net\_key\_delete\_t (C++ class), 332  
esp\_ble\_mesh\_cfg\_model\_sub\_overwrite\_t::element\_addr esp\_ble\_mesh\_cfg\_net\_key\_delete\_t::net\_idx (C++ member), 332  
esp\_ble\_mesh\_cfg\_model\_sub\_overwrite\_t::element\_addr esp\_ble\_mesh\_cfg\_net\_key\_list\_cb\_t (C++ class), 338  
esp\_ble\_mesh\_cfg\_model\_sub\_overwrite\_t::element\_addr esp\_ble\_mesh\_cfg\_net\_key\_list\_cb\_t::net\_idx (C++ member), 338  
esp\_ble\_mesh\_cfg\_model\_sub\_overwrite\_t::sub\_delete esp\_ble\_mesh\_cfg\_net\_key\_status\_cb\_t (C++ class), 336  
esp\_ble\_mesh\_cfg\_model\_sub\_status\_cb\_t esp\_ble\_mesh\_cfg\_net\_key\_status\_cb\_t::net\_idx (C++ member), 336  
esp\_ble\_mesh\_cfg\_model\_sub\_status\_cb\_t::element\_addr esp\_ble\_mesh\_cfg\_net\_key\_status\_cb\_t::status (C++ member), 336  
esp\_ble\_mesh\_cfg\_model\_sub\_status\_cb\_t::element\_addr esp\_ble\_mesh\_cfg\_net\_key\_update\_t (C++ class), 332  
esp\_ble\_mesh\_cfg\_model\_sub\_status\_cb\_t::element\_addr esp\_ble\_mesh\_cfg\_net\_key\_update\_t::net\_idx (C++ member), 332  
esp\_ble\_mesh\_cfg\_model\_sub\_status\_cb\_t::status esp\_ble\_mesh\_cfg\_net\_key\_update\_t::net\_key (C++ member), 332  
esp\_ble\_mesh\_cfg\_model\_sub\_status\_cb\_t::sub\_delete esp\_ble\_mesh\_cfg\_net\_trans\_status\_cb\_t (C++ class), 338  
esp\_ble\_mesh\_cfg\_model\_sub\_va\_add\_t esp\_ble\_mesh\_cfg\_net\_trans\_status\_cb\_t::net\_trans (C++ member), 338  
esp\_ble\_mesh\_cfg\_model\_sub\_va\_add\_t::compan esp\_ble\_mesh\_cfg\_net\_trans\_status\_cb\_t::net\_trans (C++ member), 338  
esp\_ble\_mesh\_cfg\_model\_sub\_va\_add\_t::element\_addr esp\_ble\_mesh\_cfg\_net\_transmit\_set\_t (C++ class), 334  
esp\_ble\_mesh\_cfg\_model\_sub\_va\_add\_t::label esp\_ble\_mesh\_cfg\_net\_transmit\_set\_t::net\_transmit (C++ member), 334  
esp\_ble\_mesh\_cfg\_model\_sub\_va\_add\_t::model\_id esp\_ble\_mesh\_cfg\_node\_id\_status\_cb\_t (C++ class), 339  
esp\_ble\_mesh\_cfg\_model\_sub\_va\_delete\_t esp\_ble\_mesh\_cfg\_node\_id\_status\_cb\_t::identity (C++ member), 339  
esp\_ble\_mesh\_cfg\_model\_sub\_va\_delete\_t::element\_addr esp\_ble\_mesh\_cfg\_node\_id\_status\_cb\_t::net\_idx (C++ member), 339  
esp\_ble\_mesh\_cfg\_model\_sub\_va\_delete\_t::element\_addr esp\_ble\_mesh\_cfg\_node\_id\_status\_cb\_t::status (C++ member), 339  
esp\_ble\_mesh\_cfg\_model\_sub\_va\_delete\_t::sub\_delete esp\_ble\_mesh\_cfg\_node\_identity\_get\_t



- esp\_ble\_mesh\_cfg\_srv::relay\_retransmit (C++ member), 325  
 esp\_ble\_mesh\_cfg\_srv::src (C++ member), 326  
 esp\_ble\_mesh\_cfg\_srv::timer (C++ member), 325  
 esp\_ble\_mesh\_cfg\_srv::ttl (C++ member), 326  
 esp\_ble\_mesh\_cfg\_srv\_t (C++ type), 343  
 ESP\_BLE\_MESH\_CFG\_STATUS\_CANNOT\_BIND (C 宏), 290  
 ESP\_BLE\_MESH\_CFG\_STATUS\_CANNOT\_REMOVE (C 宏), 290  
 ESP\_BLE\_MESH\_CFG\_STATUS\_CANNOT\_SET (C 宏), 290  
 ESP\_BLE\_MESH\_CFG\_STATUS\_CANNOT\_UPDATE (C 宏), 290  
 ESP\_BLE\_MESH\_CFG\_STATUS\_FEATURE\_NOT\_SUPPORTED (C 宏), 290  
 ESP\_BLE\_MESH\_CFG\_STATUS\_INSUFFICIENT\_RESOURCES (C 宏), 289  
 ESP\_BLE\_MESH\_CFG\_STATUS\_INVALID\_ADDRESSES (C 宏), 289  
 ESP\_BLE\_MESH\_CFG\_STATUS\_INVALID\_APPKEY (C 宏), 289  
 ESP\_BLE\_MESH\_CFG\_STATUS\_INVALID\_BINDING (C 宏), 290  
 ESP\_BLE\_MESH\_CFG\_STATUS\_INVALID\_MODEL (C 宏), 289  
 ESP\_BLE\_MESH\_CFG\_STATUS\_INVALID\_NETKEY (C 宏), 289  
 ESP\_BLE\_MESH\_CFG\_STATUS\_INVALID\_PUBLISH\_PARAMS (C 宏), 289  
 ESP\_BLE\_MESH\_CFG\_STATUS\_KEY\_INDEX\_ALREADY\_STORED (C 宏), 289  
 ESP\_BLE\_MESH\_CFG\_STATUS\_NOT\_A\_SUBSCRIBE\_MODEL (C 宏), 289  
 ESP\_BLE\_MESH\_CFG\_STATUS\_STORAGE\_FAILURE (C 宏), 290  
 ESP\_BLE\_MESH\_CFG\_STATUS\_SUCCESS (C 宏), 289  
 esp\_ble\_mesh\_cfg\_status\_t (C++ type), 297  
 ESP\_BLE\_MESH\_CFG\_STATUS\_TEMP\_UNABLE\_TO\_CHANGE\_STATE (C 宏), 290  
 ESP\_BLE\_MESH\_CFG\_STATUS\_UNSPECIFIED\_ERROR (C 宏), 290  
 esp\_ble\_mesh\_cfg\_vnd\_model\_app\_get\_t (C++ class), 327  
 esp\_ble\_mesh\_cfg\_vnd\_model\_app\_get\_t::company (C++ member), 327  
 esp\_ble\_mesh\_cfg\_vnd\_model\_app\_get\_t::element\_count (C++ member), 327  
 esp\_ble\_mesh\_cfg\_vnd\_model\_app\_get\_t::model\_id (C++ member), 327  
 esp\_ble\_mesh\_cfg\_vnd\_model\_sub\_get\_t (C++ class), 327  
 esp\_ble\_mesh\_cfg\_vnd\_model\_sub\_get\_t::company (C++ member), 327  
 esp\_ble\_mesh\_cfg\_vnd\_model\_sub\_get\_t::element\_add (C++ member), 327  
 esp\_ble\_mesh\_cfg\_vnd\_model\_sub\_get\_t::model\_id (C++ member), 327  
 ESP\_BLE\_MESH\_CID\_NVAL (C 宏), 282  
 esp\_ble\_mesh\_client\_common\_param\_t (C++ class), 280  
 esp\_ble\_mesh\_client\_common\_param\_t::ctx (C++ member), 280  
 esp\_ble\_mesh\_client\_common\_param\_t::model (C++ member), 280  
 esp\_ble\_mesh\_client\_common\_param\_t::msg\_role (C++ member), 280  
 esp\_ble\_mesh\_client\_common\_param\_t::msg\_timeout (C++ member), 280  
 esp\_ble\_mesh\_client\_common\_param\_t::opcode (C++ member), 280  
 esp\_ble\_mesh\_client\_model\_deinit (C++ function), 308  
 esp\_ble\_mesh\_client\_model\_init (C++ function), 308  
 ESP\_BLE\_MESH\_CLIENT\_MODEL\_RECV\_PUBLISH\_MSG\_EVT (C++ enumerator), 303  
 esp\_ble\_mesh\_client\_model\_send\_msg (C++ function), 308  
 ESP\_BLE\_MESH\_CLIENT\_MODEL\_SEND\_TIMEOUT\_EVT (C++ enumerator), 303  
 esp\_ble\_mesh\_client\_op\_pair\_t (C++ class), 279  
 esp\_ble\_mesh\_client\_op\_pair\_t::cli\_op (C++ member), 279  
 esp\_ble\_mesh\_client\_op\_pair\_t::status\_op (C++ member), 279  
 esp\_ble\_mesh\_client\_t (C++ class), 279  
 esp\_ble\_mesh\_client\_t::internal\_data (C++ member), 280  
 esp\_ble\_mesh\_client\_t::model (C++ member), 280  
 esp\_ble\_mesh\_client\_t::msg\_role (C++ member), 280  
 esp\_ble\_mesh\_client\_t::op\_pair (C++ member), 280  
 esp\_ble\_mesh\_client\_t::op\_pair\_size (C++ member), 280  
 esp\_ble\_mesh\_client\_t::publish\_status (C++ member), 280  
 esp\_ble\_mesh\_comp\_t (C++ class), 277  
 esp\_ble\_mesh\_comp\_t::cid (C++ member), 277  
 esp\_ble\_mesh\_comp\_t::element\_count (C++ member), 277  
 esp\_ble\_mesh\_comp\_t::elements (C++ member), 277  
 esp\_ble\_mesh\_comp\_t::pid (C++ member), 277  
 esp\_ble\_mesh\_comp\_t::vid (C++ member), 277  
 ESP\_BLE\_MESH\_CONDENSATION\_ERROR (C 宏), 282

- 381
- ESP\_BLE\_MESH\_CONDENSATION\_WARNING (C 宏), 381
- esp\_ble\_mesh\_config\_client\_get\_state (C++ function), 321
- esp\_ble\_mesh\_config\_client\_set\_state (C++ function), 321
- ESP\_BLE\_MESH\_CONFIGURATION\_ERROR (C 宏), 381
- ESP\_BLE\_MESH\_CONFIGURATION\_WARNING (C 宏), 381
- esp\_ble\_mesh\_deinit (C++ function), 304
- ESP\_BLE\_MESH\_DEINIT\_MESH\_COMP\_EVT (C++ enumerator), 303
- esp\_ble\_mesh\_deinit\_param\_t (C++ class), 274
- esp\_ble\_mesh\_deinit\_param\_t::erase\_flash (C++ member), 274
- esp\_ble\_mesh\_dev\_add\_flag\_t (C++ type), 296
- esp\_ble\_mesh\_dev\_role\_t (C++ enum), 299
- esp\_ble\_mesh\_device\_delete\_t (C++ class), 277
- esp\_ble\_mesh\_device\_delete\_t::addr (C++ member), 277
- esp\_ble\_mesh\_device\_delete\_t::addr\_type (C++ member), 277
- esp\_ble\_mesh\_device\_delete\_t::flag (C++ member), 278
- esp\_ble\_mesh\_device\_delete\_t::uuid (C++ member), 278
- ESP\_BLE\_MESH\_DEVICE\_DROPPED\_ERROR (C 宏), 382
- ESP\_BLE\_MESH\_DEVICE\_DROPPED\_WARNING (C 宏), 382
- ESP\_BLE\_MESH\_DEVICE\_MOVED\_ERROR (C 宏), 382
- ESP\_BLE\_MESH\_DEVICE\_MOVED\_WARNING (C 宏), 382
- ESP\_BLE\_MESH\_DEVICE\_NAME\_MAX\_LEN (C 宏), 282
- ESP\_BLE\_MESH\_DISPLAY\_NUMBER (C++ enumerator), 298
- ESP\_BLE\_MESH\_DISPLAY\_STRING (C++ enumerator), 298
- esp\_ble\_mesh\_elem\_t (C++ class), 274
- esp\_ble\_mesh\_elem\_t::element\_addr (C++ member), 274
- esp\_ble\_mesh\_elem\_t::location (C++ member), 274
- esp\_ble\_mesh\_elem\_t::sig\_model\_count (C++ member), 274
- esp\_ble\_mesh\_elem\_t::sig\_models (C++ member), 274
- esp\_ble\_mesh\_elem\_t::vnd\_model\_count (C++ member), 274
- esp\_ble\_mesh\_elem\_t::vnd\_models (C++ member), 274
- ESP\_BLE\_MESH\_ELEMENT (C 宏), 284
- ESP\_BLE\_MESH\_ELEMENT\_NOT\_CALIBRATED\_ERROR (C 宏), 381
- ESP\_BLE\_MESH\_ELEMENT\_NOT\_CALIBRATED\_WARNING (C 宏), 381
- ESP\_BLE\_MESH\_EMPTY\_ERROR (C 宏), 382
- ESP\_BLE\_MESH\_EMPTY\_WARNING (C 宏), 382
- ESP\_BLE\_MESH\_ENTER\_NUMBER (C++ enumerator), 298
- ESP\_BLE\_MESH\_ENTER\_STRING (C++ enumerator), 298
- esp\_ble\_mesh\_fast\_prov\_action\_t (C++ enum), 299
- esp\_ble\_mesh\_fast\_prov\_info\_t (C++ class), 279
- esp\_ble\_mesh\_fast\_prov\_info\_t::flags (C++ member), 279
- esp\_ble\_mesh\_fast\_prov\_info\_t::iv\_index (C++ member), 279
- esp\_ble\_mesh\_fast\_prov\_info\_t::match\_len (C++ member), 279
- esp\_ble\_mesh\_fast\_prov\_info\_t::match\_val (C++ member), 279
- esp\_ble\_mesh\_fast\_prov\_info\_t::net\_idx (C++ member), 279
- esp\_ble\_mesh\_fast\_prov\_info\_t::offset (C++ member), 279
- esp\_ble\_mesh\_fast\_prov\_info\_t::unicast\_max (C++ member), 279
- esp\_ble\_mesh\_fast\_prov\_info\_t::unicast\_min (C++ member), 279
- ESP\_BLE\_MESH\_FEATURE\_ALL\_SUPPORTED (C 宏), 283
- ESP\_BLE\_MESH\_FEATURE\_FRIEND (C 宏), 283
- ESP\_BLE\_MESH\_FEATURE\_LOW\_POWER (C 宏), 283
- ESP\_BLE\_MESH\_FEATURE\_PROXY (C 宏), 283
- ESP\_BLE\_MESH\_FEATURE\_RELAY (C 宏), 283
- esp\_ble\_mesh\_find\_element (C++ function), 305
- esp\_ble\_mesh\_find\_sig\_model (C++ function), 305
- esp\_ble\_mesh\_find\_vendor\_model (C++ function), 305
- ESP\_BLE\_MESH\_FRIEND\_DISABLED (C 宏), 283
- ESP\_BLE\_MESH\_FRIEND\_ENABLED (C 宏), 283
- ESP\_BLE\_MESH\_FRIEND\_FRIENDSHIP\_ESTABLISH\_EVT (C++ enumerator), 302
- ESP\_BLE\_MESH\_FRIEND\_FRIENDSHIP\_TERMINATE\_EVT (C++ enumerator), 302
- ESP\_BLE\_MESH\_FRIEND\_NOT\_SUPPORTED (C 宏), 283
- ESP\_BLE\_MESH\_GATT\_PROXY\_DISABLED (C 宏), 282
- ESP\_BLE\_MESH\_GATT\_PROXY\_ENABLED (C 宏), 283
- ESP\_BLE\_MESH\_GATT\_PROXY\_NOT\_SUPPORTED (C 宏), 283

ESP\_BLE\_MESH\_GEN\_ADMIN\_ACCESS\_READ (C++ enumerator), 374  
 ESP\_BLE\_MESH\_GEN\_ADMIN\_ACCESS\_READ\_WRITE (C++ enumerator), 374  
 ESP\_BLE\_MESH\_GEN\_ADMIN\_ACCESS\_WRITE (C++ enumerator), 374  
 ESP\_BLE\_MESH\_GEN\_ADMIN\_NOT\_USER\_PROP (C++ enumerator), 374  
 esp\_ble\_mesh\_gen\_admin\_prop\_access\_t (C++ enum), 374  
 esp\_ble\_mesh\_gen\_admin\_prop\_srv\_t (C++ class), 362  
 esp\_ble\_mesh\_gen\_admin\_prop\_srv\_t::model (C++ member), 362  
 esp\_ble\_mesh\_gen\_admin\_prop\_srv\_t::property\_ids (C++ member), 362  
 esp\_ble\_mesh\_gen\_admin\_prop\_srv\_t::property\_value (C++ member), 362  
 esp\_ble\_mesh\_gen\_admin\_prop\_srv\_t::rsp\_ctrl (C++ member), 362  
 esp\_ble\_mesh\_gen\_admin\_properties\_status\_cb\_t (C++ class), 355  
 esp\_ble\_mesh\_gen\_admin\_properties\_status\_cb\_t::model (C++ member), 356  
 esp\_ble\_mesh\_gen\_admin\_property\_get\_t (C++ class), 352  
 esp\_ble\_mesh\_gen\_admin\_property\_get\_t::model (C++ member), 352  
 esp\_ble\_mesh\_gen\_admin\_property\_set\_t (C++ class), 352  
 esp\_ble\_mesh\_gen\_admin\_property\_set\_t::model (C++ member), 352  
 esp\_ble\_mesh\_gen\_admin\_property\_set\_t::msg (C++ member), 352  
 esp\_ble\_mesh\_gen\_admin\_property\_set\_t::msg\_len (C++ member), 352  
 esp\_ble\_mesh\_gen\_admin\_property\_set\_t::msg\_type (C++ member), 352  
 esp\_ble\_mesh\_gen\_admin\_property\_status\_cb\_t (C++ class), 356  
 esp\_ble\_mesh\_gen\_admin\_property\_status\_cb\_t::admin\_property (C++ member), 347  
 esp\_ble\_mesh\_gen\_admin\_property\_status\_cb\_t::admin\_property\_id (C++ member), 347  
 esp\_ble\_mesh\_gen\_admin\_property\_status\_cb\_t::battery\_status (C++ member), 346  
 esp\_ble\_mesh\_gen\_admin\_property\_status\_cb\_t::client\_property\_value (C++ member), 347  
 esp\_ble\_mesh\_gen\_admin\_property\_status\_cb\_t::def\_trans\_time (C++ member), 346  
 esp\_ble\_mesh\_gen\_battery\_srv\_t (C++ class), 360  
 esp\_ble\_mesh\_gen\_battery\_srv\_t::model (C++ member), 360  
 esp\_ble\_mesh\_gen\_battery\_srv\_t::rsp\_ctrl (C++ member), 360  
 esp\_ble\_mesh\_gen\_battery\_srv\_t::state (C++ member), 360  
 esp\_ble\_mesh\_gen\_battery\_state\_t (C++ class), 360  
 esp\_ble\_mesh\_gen\_battery\_state\_t::battery\_level (C++ member), 360  
 esp\_ble\_mesh\_gen\_battery\_state\_t::time\_to\_charge (C++ member), 360  
 esp\_ble\_mesh\_gen\_battery\_state\_t::time\_to\_discharge (C++ member), 360  
 esp\_ble\_mesh\_gen\_battery\_status\_cb\_t (C++ class), 354  
 esp\_ble\_mesh\_gen\_battery\_status\_cb\_t::battery\_level (C++ member), 354  
 esp\_ble\_mesh\_gen\_battery\_status\_cb\_t::flags (C++ member), 354  
 esp\_ble\_mesh\_gen\_battery\_status\_cb\_t::time\_to\_charge (C++ member), 354  
 esp\_ble\_mesh\_gen\_battery\_status\_cb\_t::time\_to\_discharge (C++ member), 354  
 esp\_ble\_mesh\_gen\_client\_prop\_srv\_t (C++ class), 362  
 esp\_ble\_mesh\_gen\_client\_prop\_srv\_t::id\_count (C++ member), 363  
 esp\_ble\_mesh\_gen\_client\_prop\_srv\_t::model (C++ member), 363  
 esp\_ble\_mesh\_gen\_client\_prop\_srv\_t::property\_ids (C++ member), 363  
 esp\_ble\_mesh\_gen\_client\_prop\_srv\_t::rsp\_ctrl (C++ member), 363  
 esp\_ble\_mesh\_gen\_client\_properties\_get\_t (C++ class), 352  
 esp\_ble\_mesh\_gen\_client\_properties\_get\_t::property (C++ member), 353  
 esp\_ble\_mesh\_gen\_client\_properties\_status\_cb\_t (C++ class), 356  
 esp\_ble\_mesh\_gen\_client\_properties\_status\_cb\_t::property (C++ member), 356  
 esp\_ble\_mesh\_gen\_client\_status\_cb\_t (C++ union), 346  
 esp\_ble\_mesh\_gen\_client\_status\_cb\_t::admin\_property (C++ member), 347  
 esp\_ble\_mesh\_gen\_client\_status\_cb\_t::admin\_property\_id (C++ member), 347  
 esp\_ble\_mesh\_gen\_client\_status\_cb\_t::battery\_status (C++ member), 346  
 esp\_ble\_mesh\_gen\_client\_status\_cb\_t::client\_property\_value (C++ member), 347  
 esp\_ble\_mesh\_gen\_client\_status\_cb\_t::def\_trans\_time (C++ member), 346  
 esp\_ble\_mesh\_gen\_client\_status\_cb\_t::level\_status (C++ member), 346  
 esp\_ble\_mesh\_gen\_client\_status\_cb\_t::location\_global (C++ member), 346  
 esp\_ble\_mesh\_gen\_client\_status\_cb\_t::location\_local (C++ member), 346  
 esp\_ble\_mesh\_gen\_client\_status\_cb\_t::manufacturer (C++ member), 347  
 esp\_ble\_mesh\_gen\_client\_status\_cb\_t::manufacturer\_id (C++ member), 347  
 esp\_ble\_mesh\_gen\_client\_status\_cb\_t::onoff\_status (C++ member), 346

---

esp\_ble\_mesh\_gen\_client\_status\_cb\_t::on\_ble\_mesh\_gen\_level\_srv\_t (C++ class),  
 (C++ member), 346

esp\_ble\_mesh\_gen\_client\_status\_cb\_t::on\_ble\_mesh\_gen\_level\_srv\_t::last  
 (C++ member), 346

esp\_ble\_mesh\_gen\_client\_status\_cb\_t::on\_ble\_mesh\_gen\_level\_srv\_t::model  
 (C++ member), 346

esp\_ble\_mesh\_gen\_client\_status\_cb\_t::on\_ble\_mesh\_gen\_level\_srv\_t::rsp\_ctrl  
 (C++ member), 346

esp\_ble\_mesh\_gen\_client\_status\_cb\_t::on\_ble\_mesh\_gen\_level\_srv\_t::state  
 (C++ member), 346

esp\_ble\_mesh\_gen\_client\_status\_cb\_t::on\_ble\_mesh\_gen\_level\_srv\_t::transition  
 (C++ member), 346

esp\_ble\_mesh\_gen\_client\_status\_cb\_t::on\_ble\_mesh\_gen\_level\_srv\_t::tt\_delta\_level  
 (C++ member), 346

esp\_ble\_mesh\_gen\_def\_trans\_time\_set\_t esp\_ble\_mesh\_gen\_level\_state\_t (C++  
 (C++ class), 350 class), 357

esp\_ble\_mesh\_gen\_def\_trans\_time\_set\_t::start\_time esp\_ble\_mesh\_gen\_level\_state\_t::last\_delta  
 (C++ member), 350 (C++ member), 357

esp\_ble\_mesh\_gen\_def\_trans\_time\_srv\_t esp\_ble\_mesh\_gen\_level\_state\_t::last\_level  
 (C++ class), 358 (C++ member), 357

esp\_ble\_mesh\_gen\_def\_trans\_time\_srv\_t::model esp\_ble\_mesh\_gen\_level\_state\_t::level  
 (C++ member), 358 (C++ member), 357

esp\_ble\_mesh\_gen\_def\_trans\_time\_srv\_t::on\_ble\_mesh\_gen\_level\_state\_t::move\_start  
 (C++ member), 358 (C++ member), 357

esp\_ble\_mesh\_gen\_def\_trans\_time\_srv\_t::state esp\_ble\_mesh\_gen\_level\_state\_t::positive  
 (C++ member), 358 (C++ member), 358

esp\_ble\_mesh\_gen\_def\_trans\_time\_state\_t esp\_ble\_mesh\_gen\_level\_state\_t::target\_level  
 (C++ class), 358 (C++ member), 357

esp\_ble\_mesh\_gen\_def\_trans\_time\_state\_t::start\_time esp\_ble\_mesh\_gen\_level\_status\_cb\_t  
 (C++ member), 358 (C++ class), 353

esp\_ble\_mesh\_gen\_def\_trans\_time\_status\_cb\_t esp\_ble\_mesh\_gen\_level\_status\_cb\_t::op\_en  
 (C++ class), 353 (C++ member), 353

esp\_ble\_mesh\_gen\_def\_trans\_time\_status\_cb\_t::present\_level  
 (C++ member), 353 (C++ member), 353

esp\_ble\_mesh\_gen\_def\_trans\_time\_status\_cb\_t::remain\_time  
 (C++ member), 353 (C++ member), 353

esp\_ble\_mesh\_gen\_def\_trans\_time\_status\_cb\_t::target\_level  
 (C++ member), 353 (C++ member), 353

esp\_ble\_mesh\_gen\_delta\_set\_t (C++ class), esp\_ble\_mesh\_gen\_loc\_global\_set\_t  
 349 (C++ class), 351

esp\_ble\_mesh\_gen\_delta\_set\_t::delay esp\_ble\_mesh\_gen\_loc\_global\_set\_t::global\_altitud  
 (C++ member), 350 (C++ member), 351

esp\_ble\_mesh\_gen\_delta\_set\_t::level esp\_ble\_mesh\_gen\_loc\_global\_set\_t::global\_latitud  
 (C++ member), 349 (C++ member), 351

esp\_ble\_mesh\_gen\_delta\_set\_t::op\_en esp\_ble\_mesh\_gen\_loc\_global\_set\_t::global\_longitud  
 (C++ member), 349 (C++ member), 351

esp\_ble\_mesh\_gen\_delta\_set\_t::tid esp\_ble\_mesh\_gen\_loc\_global\_set\_t::global\_longitud  
 (C++ member), 350 (C++ member), 351

esp\_ble\_mesh\_gen\_delta\_set\_t::trans\_time esp\_ble\_mesh\_gen\_loc\_global\_status\_cb\_t  
 (C++ member), 350 (C++ class), 354

esp\_ble\_mesh\_gen\_level\_set\_t (C++ class), esp\_ble\_mesh\_gen\_loc\_global\_status\_cb\_t::global\_a  
 349 (C++ member), 355

esp\_ble\_mesh\_gen\_level\_set\_t::delay esp\_ble\_mesh\_gen\_loc\_global\_status\_cb\_t::global\_l  
 (C++ member), 349 (C++ member), 355

esp\_ble\_mesh\_gen\_level\_set\_t::level esp\_ble\_mesh\_gen\_loc\_global\_status\_cb\_t::global\_l  
 (C++ member), 349 (C++ member), 355

esp\_ble\_mesh\_gen\_level\_set\_t::op\_en esp\_ble\_mesh\_gen\_loc\_global\_status\_cb\_t::global\_l  
 (C++ member), 349 (C++ member), 355

esp\_ble\_mesh\_gen\_level\_set\_t::tid esp\_ble\_mesh\_gen\_loc\_local\_set\_t (C++  
 (C++ member), 349 class), 351

esp\_ble\_mesh\_gen\_level\_set\_t::trans\_time esp\_ble\_mesh\_gen\_loc\_local\_set\_t::floor\_number  
 (C++ member), 349 (C++ member), 351

---

esp\_ble\_mesh\_gen\_loc\_local\_set\_t::local\_status\_cb\_t (C++ member), 351  
 esp\_ble\_mesh\_gen\_loc\_local\_set\_t::local\_status\_cb\_t (C++ member), 351  
 esp\_ble\_mesh\_gen\_loc\_local\_set\_t::local\_status\_cb\_t (C++ member), 351  
 esp\_ble\_mesh\_gen\_loc\_local\_set\_t::uncertainty\_cb\_t (C++ member), 351  
 esp\_ble\_mesh\_gen\_loc\_local\_status\_cb\_t (C++ class), 355  
 esp\_ble\_mesh\_gen\_loc\_local\_status\_cb\_t (C++ member), 355  
 esp\_ble\_mesh\_gen\_loc\_local\_status\_cb\_t (C++ member), 355  
 esp\_ble\_mesh\_gen\_loc\_local\_status\_cb\_t (C++ member), 355  
 esp\_ble\_mesh\_gen\_loc\_local\_status\_cb\_t (C++ member), 355  
 esp\_ble\_mesh\_gen\_loc\_local\_status\_cb\_t (C++ member), 355  
 esp\_ble\_mesh\_gen\_loc\_local\_status\_cb\_t (C++ member), 355  
 esp\_ble\_mesh\_gen\_location\_setup\_srv\_t (C++ class), 361  
 esp\_ble\_mesh\_gen\_location\_setup\_srv\_t (C++ member), 361  
 esp\_ble\_mesh\_gen\_location\_setup\_srv\_t (C++ member), 361  
 esp\_ble\_mesh\_gen\_location\_setup\_srv\_t (C++ member), 361  
 esp\_ble\_mesh\_gen\_location\_srv\_t (C++ class), 361  
 esp\_ble\_mesh\_gen\_location\_srv\_t::model (C++ member), 361  
 esp\_ble\_mesh\_gen\_location\_srv\_t::rsp\_ctrl (C++ member), 361  
 esp\_ble\_mesh\_gen\_location\_srv\_t::state (C++ member), 361  
 esp\_ble\_mesh\_gen\_location\_state\_t (C++ class), 360  
 esp\_ble\_mesh\_gen\_location\_state\_t (C++ member), 361  
 esp\_ble\_mesh\_gen\_location\_state\_t::global\_status\_cb\_t (C++ member), 361  
 esp\_ble\_mesh\_gen\_location\_state\_t::global\_status\_cb\_t (C++ member), 361  
 esp\_ble\_mesh\_gen\_location\_state\_t::global\_status\_cb\_t (C++ member), 361  
 esp\_ble\_mesh\_gen\_location\_state\_t::local\_status\_cb\_t (C++ member), 361  
 esp\_ble\_mesh\_gen\_location\_state\_t::local\_status\_cb\_t (C++ member), 361  
 esp\_ble\_mesh\_gen\_location\_state\_t::uncertainty\_cb\_t (C++ member), 361  
 ESP\_BLE\_MESH\_GEN\_MANU\_ACCESS\_READ (C++ enumerator), 374  
 ESP\_BLE\_MESH\_GEN\_MANU\_NOT\_USER\_PROP (C++ enumerator), 374  
 esp\_ble\_mesh\_gen\_manu\_prop\_access\_t (C++ enum), 374  
 esp\_ble\_mesh\_gen\_manu\_prop\_srv\_t (C++ class), 362  
 esp\_ble\_mesh\_gen\_manu\_prop\_srv\_t::model (C++ member), 362  
 esp\_ble\_mesh\_gen\_manu\_prop\_srv\_t::properties (C++ member), 362  
 esp\_ble\_mesh\_gen\_manu\_prop\_srv\_t::property\_count (C++ member), 362  
 esp\_ble\_mesh\_gen\_manu\_prop\_srv\_t::rsp\_ctrl (C++ member), 362  
 esp\_ble\_mesh\_gen\_manufacturer\_properties\_status\_cb\_t (C++ class), 356  
 esp\_ble\_mesh\_gen\_manufacturer\_properties\_status\_cb\_t (C++ member), 356  
 esp\_ble\_mesh\_gen\_manufacturer\_property\_get\_t (C++ class), 352  
 esp\_ble\_mesh\_gen\_manufacturer\_property\_get\_t::properties (C++ member), 352  
 esp\_ble\_mesh\_gen\_manufacturer\_property\_set\_t (C++ class), 352  
 esp\_ble\_mesh\_gen\_manufacturer\_property\_set\_t::properties (C++ member), 352  
 esp\_ble\_mesh\_gen\_manufacturer\_property\_set\_t::use\_model (C++ member), 352  
 esp\_ble\_mesh\_gen\_manufacturer\_property\_status\_cb\_t (C++ class), 356  
 esp\_ble\_mesh\_gen\_manufacturer\_property\_status\_cb\_t (C++ member), 356  
 esp\_ble\_mesh\_gen\_manufacturer\_property\_status\_cb\_t (C++ member), 356  
 esp\_ble\_mesh\_gen\_move\_set\_t (C++ class), 350  
 esp\_ble\_mesh\_gen\_move\_set\_t::delay (C++ member), 350  
 esp\_ble\_mesh\_gen\_move\_set\_t::delta\_level (C++ member), 350  
 esp\_ble\_mesh\_gen\_move\_set\_t::op\_en (C++ member), 350  
 esp\_ble\_mesh\_gen\_move\_set\_t::tid (C++ member), 350  
 esp\_ble\_mesh\_gen\_move\_set\_t::trans\_time (C++ member), 350  
 esp\_ble\_mesh\_gen\_onoff\_set\_t (C++ class), 349  
 esp\_ble\_mesh\_gen\_onoff\_set\_t::delay (C++ member), 349  
 esp\_ble\_mesh\_gen\_onoff\_set\_t::onoff (C++ member), 349  
 esp\_ble\_mesh\_gen\_onoff\_set\_t::op\_en (C++ member), 349  
 esp\_ble\_mesh\_gen\_onoff\_set\_t::tid (C++ member), 349

esp\_ble\_mesh\_gen\_onoff\_set\_t::trans\_time (C++ member), 349  
 esp\_ble\_mesh\_gen\_onoff\_srv\_t (C++ class), 357  
 esp\_ble\_mesh\_gen\_onoff\_srv\_t::last (C++ member), 357  
 esp\_ble\_mesh\_gen\_onoff\_srv\_t::model (C++ member), 357  
 esp\_ble\_mesh\_gen\_onoff\_srv\_t::rsp\_ctrl (C++ member), 357  
 esp\_ble\_mesh\_gen\_onoff\_srv\_t::state (C++ member), 357  
 esp\_ble\_mesh\_gen\_onoff\_srv\_t::transition (C++ member), 357  
 esp\_ble\_mesh\_gen\_onoff\_state\_t (C++ class), 357  
 esp\_ble\_mesh\_gen\_onoff\_state\_t::onoff (C++ member), 357  
 esp\_ble\_mesh\_gen\_onoff\_state\_t::target\_time (C++ member), 357  
 esp\_ble\_mesh\_gen\_onoff\_status\_cb\_t (C++ class), 353  
 esp\_ble\_mesh\_gen\_onoff\_status\_cb\_t::op\_err (C++ member), 353  
 esp\_ble\_mesh\_gen\_onoff\_status\_cb\_t::present\_time (C++ member), 353  
 esp\_ble\_mesh\_gen\_onoff\_status\_cb\_t::remain\_time (C++ member), 353  
 esp\_ble\_mesh\_gen\_onoff\_status\_cb\_t::target\_time (C++ member), 353  
 esp\_ble\_mesh\_gen\_onpowerup\_set\_t (C++ class), 350  
 esp\_ble\_mesh\_gen\_onpowerup\_set\_t::onpowerup (C++ member), 350  
 esp\_ble\_mesh\_gen\_onpowerup\_state\_t (C++ class), 358  
 esp\_ble\_mesh\_gen\_onpowerup\_state\_t::onpowerup (C++ member), 358  
 esp\_ble\_mesh\_gen\_onpowerup\_state\_t::onpowerup\_time (C++ member), 358  
 esp\_ble\_mesh\_gen\_onpowerup\_status\_cb\_t (C++ class), 353  
 esp\_ble\_mesh\_gen\_onpowerup\_status\_cb\_t::onpowerup (C++ member), 353  
 esp\_ble\_mesh\_gen\_onpowerup\_status\_cb\_t::onpowerup\_time (C++ member), 353  
 esp\_ble\_mesh\_gen\_power\_default\_set\_t (C++ class), 351  
 esp\_ble\_mesh\_gen\_power\_default\_set\_t::power (C++ member), 351  
 esp\_ble\_mesh\_gen\_power\_default\_status\_cb\_t (C++ class), 354  
 esp\_ble\_mesh\_gen\_power\_default\_status\_cb\_t::power (C++ member), 354  
 esp\_ble\_mesh\_gen\_power\_last\_status\_cb\_t (C++ class), 354  
 esp\_ble\_mesh\_gen\_power\_last\_status\_cb\_t::power (C++ member), 354  
 esp\_ble\_mesh\_gen\_power\_level\_set\_t (C++ class), 350  
 esp\_ble\_mesh\_gen\_power\_level\_set\_t::delay (C++ member), 351  
 esp\_ble\_mesh\_gen\_power\_level\_set\_t::op\_en (C++ member), 350  
 esp\_ble\_mesh\_gen\_power\_level\_set\_t::power (C++ member), 350  
 esp\_ble\_mesh\_gen\_power\_level\_set\_t::tid (C++ member), 350  
 esp\_ble\_mesh\_gen\_power\_level\_set\_t::trans\_time (C++ member), 350  
 esp\_ble\_mesh\_gen\_power\_level\_setup\_srv\_t (C++ class), 360  
 esp\_ble\_mesh\_gen\_power\_level\_setup\_srv\_t::model (C++ member), 360  
 esp\_ble\_mesh\_gen\_power\_level\_setup\_srv\_t::rsp\_ctrl (C++ member), 360  
 esp\_ble\_mesh\_gen\_power\_level\_setup\_srv\_t::state (C++ member), 360  
 esp\_ble\_mesh\_gen\_power\_level\_srv\_t (C++ class), 359  
 esp\_ble\_mesh\_gen\_power\_level\_srv\_t::last (C++ member), 360  
 esp\_ble\_mesh\_gen\_power\_level\_srv\_t::model (C++ member), 359  
 esp\_ble\_mesh\_gen\_power\_level\_srv\_t::rsp\_ctrl (C++ member), 359  
 esp\_ble\_mesh\_gen\_power\_level\_srv\_t::state (C++ member), 359  
 esp\_ble\_mesh\_gen\_power\_level\_srv\_t::transition (C++ member), 360  
 esp\_ble\_mesh\_gen\_power\_level\_srv\_t::tt\_delta\_level (C++ member), 360  
 esp\_ble\_mesh\_gen\_power\_level\_state\_t (C++ class), 359  
 esp\_ble\_mesh\_gen\_power\_level\_state\_t::power\_actual (C++ member), 359  
 esp\_ble\_mesh\_gen\_power\_level\_state\_t::power\_default (C++ member), 359  
 esp\_ble\_mesh\_gen\_power\_level\_state\_t::power\_last (C++ member), 359  
 esp\_ble\_mesh\_gen\_power\_level\_state\_t::power\_range (C++ member), 359  
 esp\_ble\_mesh\_gen\_power\_level\_state\_t::power\_range\_min (C++ member), 359  
 esp\_ble\_mesh\_gen\_power\_level\_state\_t::status\_code (C++ member), 359  
 esp\_ble\_mesh\_gen\_power\_level\_state\_t::target\_power (C++ member), 359  
 esp\_ble\_mesh\_gen\_power\_level\_status\_cb\_t (C++ class), 353  
 esp\_ble\_mesh\_gen\_power\_level\_status\_cb\_t::op\_en (C++ member), 354  
 esp\_ble\_mesh\_gen\_power\_level\_status\_cb\_t::present\_time (C++ member), 354  
 esp\_ble\_mesh\_gen\_power\_level\_status\_cb\_t::remain\_time (C++ member), 354  
 esp\_ble\_mesh\_gen\_power\_level\_status\_cb\_t::target\_time (C++ member), 354  
 esp\_ble\_mesh\_gen\_power\_onoff\_setup\_srv\_t (C++ class), 359



esp\_ble\_mesh\_gen\_power\_onoff\_setup\_srv\_t::mode (C++ member), 359  
 esp\_ble\_mesh\_gen\_power\_onoff\_setup\_srv\_t::role (C++ member), 359  
 esp\_ble\_mesh\_gen\_power\_onoff\_setup\_srv\_t::state (C++ member), 359  
 esp\_ble\_mesh\_gen\_power\_onoff\_srv\_t (C++ class), 358  
 esp\_ble\_mesh\_gen\_power\_onoff\_srv\_t::mode (C++ member), 359  
 esp\_ble\_mesh\_gen\_power\_onoff\_srv\_t::rsp (C++ member), 359  
 esp\_ble\_mesh\_gen\_power\_onoff\_srv\_t::state (C++ member), 359  
 esp\_ble\_mesh\_gen\_power\_range\_set\_t (C++ class), 351  
 esp\_ble\_mesh\_gen\_power\_range\_set\_t::range (C++ member), 351  
 esp\_ble\_mesh\_gen\_power\_range\_set\_t::range (C++ member), 351  
 esp\_ble\_mesh\_gen\_power\_range\_status\_cb\_t (C++ class), 354  
 esp\_ble\_mesh\_gen\_power\_range\_status\_cb\_t (C++ member), 354  
 esp\_ble\_mesh\_gen\_power\_range\_status\_cb\_t (C++ member), 354  
 esp\_ble\_mesh\_gen\_power\_range\_status\_cb\_t (C++ member), 354  
 esp\_ble\_mesh\_gen\_power\_range\_status\_cb\_t (C++ member), 354  
 ESP\_BLE\_MESH\_GEN\_USER\_ACCESS\_PROHIBIT (C++ enumerator), 374  
 ESP\_BLE\_MESH\_GEN\_USER\_ACCESS\_READ (C++ enumerator), 374  
 ESP\_BLE\_MESH\_GEN\_USER\_ACCESS\_READ\_WRITE (C++ enumerator), 374  
 ESP\_BLE\_MESH\_GEN\_USER\_ACCESS\_WRITE (C++ enumerator), 374  
 esp\_ble\_mesh\_gen\_user\_prop\_access\_t (C++ enum), 374  
 esp\_ble\_mesh\_gen\_user\_prop\_srv\_t (C++ class), 362  
 esp\_ble\_mesh\_gen\_user\_prop\_srv\_t::mode (C++ member), 362  
 esp\_ble\_mesh\_gen\_user\_prop\_srv\_t::property (C++ member), 362  
 esp\_ble\_mesh\_gen\_user\_prop\_srv\_t::property (C++ member), 362  
 esp\_ble\_mesh\_gen\_user\_prop\_srv\_t::rsp (C++ member), 362  
 esp\_ble\_mesh\_gen\_user\_properties\_status\_cb\_t (C++ class), 355  
 esp\_ble\_mesh\_gen\_user\_properties\_status\_cb\_t (C++ member), 355  
 esp\_ble\_mesh\_gen\_user\_property\_get\_t (C++ class), 351  
 esp\_ble\_mesh\_gen\_user\_property\_get\_t::property (C++ member), 352  
 esp\_ble\_mesh\_gen\_user\_property\_set\_t (C++ class), 352  
 esp\_ble\_mesh\_gen\_user\_property\_set\_t::property\_id (C++ member), 352  
 esp\_ble\_mesh\_gen\_user\_property\_set\_t::property\_value (C++ member), 352  
 esp\_ble\_mesh\_gen\_user\_property\_status\_cb\_t (C++ class), 355  
 esp\_ble\_mesh\_gen\_user\_property\_status\_cb\_t::open (C++ member), 355  
 esp\_ble\_mesh\_gen\_user\_property\_status\_cb\_t::property (C++ member), 355  
 esp\_ble\_mesh\_gen\_user\_property\_status\_cb\_t::property (C++ member), 355  
 esp\_ble\_mesh\_gen\_user\_property\_status\_cb\_t::user (C++ member), 355  
 esp\_ble\_mesh\_generic\_client\_cb\_event\_t (C++ enum), 374  
 esp\_ble\_mesh\_generic\_client\_cb\_param\_t (C++ class), 356  
 esp\_ble\_mesh\_generic\_client\_cb\_param\_t::error\_code (C++ member), 357  
 esp\_ble\_mesh\_generic\_client\_cb\_param\_t::params (C++ member), 357  
 esp\_ble\_mesh\_generic\_client\_cb\_param\_t::status\_cb (C++ member), 357  
 esp\_ble\_mesh\_generic\_client\_cb\_param\_t (C++ member), 357  
 ESP\_BLE\_MESH\_GENERIC\_CLIENT\_EVT\_MAX (C++ enumerator), 374  
 esp\_ble\_mesh\_generic\_client\_get\_state (C++ function), 344  
 ESP\_BLE\_MESH\_GENERIC\_CLIENT\_GET\_STATE\_EVT (C++ enumerator), 374  
 esp\_ble\_mesh\_generic\_client\_get\_state\_t (C++ union), 345  
 esp\_ble\_mesh\_generic\_client\_get\_state\_t::admin\_priv (C++ member), 345  
 esp\_ble\_mesh\_generic\_client\_get\_state\_t::client\_priv (C++ member), 345  
 esp\_ble\_mesh\_generic\_client\_get\_state\_t::manufact (C++ member), 345  
 esp\_ble\_mesh\_generic\_client\_get\_state\_t::user\_priv (C++ member), 345  
 ESP\_BLE\_MESH\_GENERIC\_CLIENT\_PUBLISH\_EVT (C++ enumerator), 374  
 esp\_ble\_mesh\_generic\_client\_set\_state (C++ function), 344  
 ESP\_BLE\_MESH\_GENERIC\_CLIENT\_SET\_STATE\_EVT (C++ enumerator), 374  
 esp\_ble\_mesh\_generic\_client\_set\_state\_t (C++ union), 345  
 esp\_ble\_mesh\_generic\_client\_set\_state\_t::admin\_priv (C++ member), 346  
 esp\_ble\_mesh\_generic\_client\_set\_state\_t::def\_trans (C++ member), 345  
 esp\_ble\_mesh\_generic\_client\_set\_state\_t::delta\_send (C++ member), 345  
 esp\_ble\_mesh\_generic\_client\_set\_state\_t::level\_send (C++ member), 345



esp\_ble\_mesh\_generic\_server\_state\_change esp\_ble\_mesh\_health\_attention\_on\_cb\_t::time  
 (C++ member), 347 (C++ member), 380  
 esp\_ble\_mesh\_generic\_server\_state\_change esp\_ble\_mesh\_health\_attention\_set\_t  
 (C++ member), 347 (C++ class), 378  
 esp\_ble\_mesh\_generic\_server\_state\_change esp\_ble\_mesh\_health\_attention\_set\_t::attention  
 (C++ member), 347 (C++ member), 378  
 esp\_ble\_mesh\_generic\_server\_state\_change esp\_ble\_mesh\_health\_attention\_status\_cb\_t  
 (C++ member), 347 (C++ class), 379  
 esp\_ble\_mesh\_generic\_server\_state\_change esp\_ble\_mesh\_health\_attention\_status\_cb\_t::attent  
 (C++ member), 347 (C++ member), 379  
 esp\_ble\_mesh\_generic\_server\_state\_change esp\_ble\_mesh\_health\_client\_cb\_event\_t  
 (C++ member), 347 (C++ enum), 383  
 esp\_ble\_mesh\_generic\_server\_state\_change esp\_ble\_mesh\_health\_client\_cb\_param\_t  
 (C++ member), 347 (C++ class), 379  
 esp\_ble\_mesh\_generic\_server\_state\_change esp\_ble\_mesh\_health\_client\_cb\_param\_t::error\_code  
 (C++ member), 347 (C++ member), 379  
 esp\_ble\_mesh\_generic\_server\_state\_change esp\_ble\_mesh\_health\_client\_cb\_param\_t::params  
 (C++ member), 347 (C++ member), 379  
 esp\_ble\_mesh\_generic\_server\_state\_change esp\_ble\_mesh\_health\_client\_cb\_param\_t::status\_cb  
 (C++ member), 347 (C++ member), 379  
 esp\_ble\_mesh\_generic\_server\_state\_change esp\_ble\_mesh\_health\_client\_cb\_t (C++  
 (C++ member), 347 type), 382  
 esp\_ble\_mesh\_generic\_server\_state\_change esp\_ble\_mesh\_health\_client\_common\_cb\_param\_t  
 (C++ member), 347 (C++ union), 376  
 esp\_ble\_mesh\_generic\_server\_state\_change esp\_ble\_mesh\_health\_client\_common\_cb\_param\_t::att  
 (C++ member), 347 (C++ member), 376  
 esp\_ble\_mesh\_get\_composition\_data esp\_ble\_mesh\_health\_client\_common\_cb\_param\_t::cur  
 (C++ function), 305 (C++ member), 376  
 esp\_ble\_mesh\_get\_element\_count (C++ esp\_ble\_mesh\_health\_client\_common\_cb\_param\_t::faul  
 function), 305 (C++ member), 376  
 esp\_ble\_mesh\_get\_fast\_prov\_app\_key esp\_ble\_mesh\_health\_client\_common\_cb\_param\_t::per  
 (C++ function), 314 (C++ member), 376  
 esp\_ble\_mesh\_get\_model\_publish\_period ESP\_BLE\_MESH\_HEALTH\_CLIENT\_EVT\_MAX  
 (C++ function), 305 (C++ enumerator), 383  
 esp\_ble\_mesh\_get\_primary\_element\_address esp\_ble\_mesh\_health\_client\_get\_state  
 (C++ function), 305 (C++ function), 375  
 ESP\_BLE\_MESH\_GET\_PUBLISH\_TRANSMIT\_COUNT ESP\_BLE\_MESH\_HEALTH\_CLIENT\_GET\_STATE\_EVT  
 (C 宏), 284 (C++ enumerator), 383  
 ESP\_BLE\_MESH\_GET\_PUBLISH\_TRANSMIT\_INTERVAL esp\_ble\_mesh\_health\_client\_get\_state\_t  
 (C 宏), 284 (C++ union), 376  
 ESP\_BLE\_MESH\_GET\_SENSOR\_DATA\_FORMAT (C esp\_ble\_mesh\_health\_client\_get\_state\_t::fault\_get  
 宏), 438 (C++ member), 376  
 ESP\_BLE\_MESH\_GET\_SENSOR\_DATA\_LENGTH (C ESP\_BLE\_MESH\_HEALTH\_CLIENT\_PUBLISH\_EVT  
 宏), 438 (C++ enumerator), 383  
 ESP\_BLE\_MESH\_GET\_SENSOR\_DATA\_PROPERTY\_ID esp\_ble\_mesh\_health\_client\_set\_state  
 (C 宏), 438 (C++ function), 375  
 ESP\_BLE\_MESH\_GET\_TRANSMIT\_COUNT (C 宏), ESP\_BLE\_MESH\_HEALTH\_CLIENT\_SET\_STATE\_EVT  
 283 (C++ enumerator), 383  
 ESP\_BLE\_MESH\_GET\_TRANSMIT\_INTERVAL (C esp\_ble\_mesh\_health\_client\_set\_state\_t  
 宏), 283 (C++ union), 376  
 esp\_ble\_mesh\_health\_attention\_off\_cb\_t esp\_ble\_mesh\_health\_client\_set\_state\_t::attention  
 (C++ class), 380 (C++ member), 376  
 esp\_ble\_mesh\_health\_attention\_off\_cb\_t::mode esp\_ble\_mesh\_health\_client\_set\_state\_t::fault\_cle  
 (C++ member), 380 (C++ member), 376  
 esp\_ble\_mesh\_health\_attention\_on\_cb\_t esp\_ble\_mesh\_health\_client\_set\_state\_t::fault\_tes  
 (C++ class), 380 (C++ member), 376  
 esp\_ble\_mesh\_health\_attention\_on\_cb\_t::mode esp\_ble\_mesh\_health\_client\_set\_state\_t::period\_se  
 (C++ member), 380 (C++ member), 376

ESP\_BLE\_MESH\_HEALTH\_CLIENT\_TIMEOUT\_EVT esp\_ble\_mesh\_health\_period\_set\_t::fast\_period\_div  
(C++ enumerator), 383 (C++ member), 378

esp\_ble\_mesh\_health\_current\_status\_cb\_t esp\_ble\_mesh\_health\_period\_status\_cb\_t  
(C++ class), 378 (C++ class), 379

esp\_ble\_mesh\_health\_current\_status\_cb\_t esp\_ble\_mesh\_health\_period\_status\_cb\_t::fast\_peri  
(C++ member), 379 (C++ member), 379

esp\_ble\_mesh\_health\_current\_status\_cb\_t ESP\_BLE\_MESH\_HEALTH\_PUB\_DEFINE (C 宏),  
(C++ member), 379 381

esp\_ble\_mesh\_health\_current\_status\_cb\_t ESP\_BLE\_MESH\_HEALTH\_SERVER\_ATTENTION\_OFF\_EVT  
(C++ member), 379 (C++ enumerator), 383

ESP\_BLE\_MESH\_HEALTH\_FAULT\_ARRAY\_SIZE ESP\_BLE\_MESH\_HEALTH\_SERVER\_ATTENTION\_ON\_EVT  
(C 宏), 382 (C++ enumerator), 383

esp\_ble\_mesh\_health\_fault\_clear\_cb\_t esp\_ble\_mesh\_health\_server\_cb\_event\_t  
(C++ class), 380 (C++ enum), 383

esp\_ble\_mesh\_health\_fault\_clear\_cb\_t::comp esp\_ble\_mesh\_health\_server\_cb\_param\_t  
(C++ member), 380 (C++ union), 376

esp\_ble\_mesh\_health\_fault\_clear\_cb\_t::mode esp\_ble\_mesh\_health\_server\_cb\_param\_t::attention  
(C++ member), 380 (C++ member), 377

esp\_ble\_mesh\_health\_fault\_clear\_t esp\_ble\_mesh\_health\_server\_cb\_param\_t::attention  
(C++ class), 378 (C++ member), 377

esp\_ble\_mesh\_health\_fault\_clear\_t::comp esp\_ble\_mesh\_health\_server\_cb\_param\_t::fault\_clea  
(C++ member), 378 (C++ member), 376

esp\_ble\_mesh\_health\_fault\_get\_t (C++ esp\_ble\_mesh\_health\_server\_cb\_param\_t::fault\_test  
class), 378 (C++ member), 377

esp\_ble\_mesh\_health\_fault\_get\_t::comp esp\_ble\_mesh\_health\_server\_cb\_param\_t::fault\_upda  
(C++ member), 378 (C++ member), 376

esp\_ble\_mesh\_health\_fault\_status\_cb\_t esp\_ble\_mesh\_health\_server\_cb\_t (C++  
(C++ class), 379 type), 382

esp\_ble\_mesh\_health\_fault\_status\_cb\_t::ESP\_BLE\_MESH\_HEALTH\_SERVER\_EVT\_MAX  
(C++ member), 379 (C++ enumerator), 383

esp\_ble\_mesh\_health\_fault\_status\_cb\_t::ESP\_BLE\_MESH\_HEALTH\_SERVER\_FAULT\_CLEAR\_EVT  
(C++ member), 379 (C++ enumerator), 383

esp\_ble\_mesh\_health\_fault\_status\_cb\_t::ESP\_BLE\_MESH\_HEALTH\_SERVER\_FAULT\_TEST\_EVT  
(C++ member), 379 (C++ enumerator), 383

esp\_ble\_mesh\_health\_fault\_test\_cb\_t esp\_ble\_mesh\_health\_server\_fault\_update  
(C++ class), 380 (C++ function), 375

esp\_ble\_mesh\_health\_fault\_test\_cb\_t::comp ESP\_BLE\_MESH\_HEALTH\_SERVER\_FAULT\_UPDATE\_COMP\_EVT  
(C++ member), 380 (C++ enumerator), 383

esp\_ble\_mesh\_health\_fault\_test\_cb\_t::mode esp\_ble\_mesh\_health\_srv\_cb\_t (C++ class),  
(C++ member), 380 377

esp\_ble\_mesh\_health\_fault\_test\_cb\_t::test esp\_ble\_mesh\_health\_srv\_cb\_t::attention\_off  
(C++ member), 380 (C++ member), 377

esp\_ble\_mesh\_health\_fault\_test\_t (C++ esp\_ble\_mesh\_health\_srv\_cb\_t::attention\_on  
class), 378 (C++ member), 377

esp\_ble\_mesh\_health\_fault\_test\_t::comp esp\_ble\_mesh\_health\_srv\_cb\_t::fault\_clear  
(C++ member), 378 (C++ member), 377

esp\_ble\_mesh\_health\_fault\_test\_t::test esp\_ble\_mesh\_health\_srv\_cb\_t::fault\_test  
(C++ member), 378 (C++ member), 377

esp\_ble\_mesh\_health\_fault\_update\_comp\_cb\_t esp\_ble\_mesh\_health\_srv\_t (C++ class), 377  
(C++ class), 379 esp\_ble\_mesh\_health\_srv\_t::attention\_timer

esp\_ble\_mesh\_health\_fault\_update\_comp\_cb\_t::e ESP\_BLE\_MESH\_HEALTH\_SRV\_EVT\_MAX  
(C++ member), 380 (C++ enumerator), 377

esp\_ble\_mesh\_health\_fault\_update\_comp\_cb\_t::e ESP\_BLE\_MESH\_HEALTH\_SRV\_EVT\_FAULT\_CLEAR  
(C++ member), 380 (C++ enumerator), 378

esp\_ble\_mesh\_health\_model\_status\_t esp\_ble\_mesh\_health\_srv\_t::attention\_timer\_start  
(C++ type), 297 (C++ member), 377

esp\_ble\_mesh\_health\_period\_set\_t (C++ esp\_ble\_mesh\_health\_srv\_t::health\_cb  
class), 378 (C++ member), 377

esp\_ble\_mesh\_health\_period\_set\_t (C++ esp\_ble\_mesh\_health\_srv\_t::health\_test  
class), 378 (C++ member), 378

esp\_ble\_mesh\_health\_period\_set\_t (C++ esp\_ble\_mesh\_health\_srv\_t::model (C++  
class), 378

- member*), 377
- ESP\_BLE\_MESH\_HEALTH\_STANDARD\_TEST (C 宏), 381
- esp\_ble\_mesh\_health\_test\_t (C++ class), 377
- esp\_ble\_mesh\_health\_test\_t::company\_id (C++ member), 377
- esp\_ble\_mesh\_health\_test\_t::current\_faults (C++ member), 377
- esp\_ble\_mesh\_health\_test\_t::id\_count (C++ member), 377
- esp\_ble\_mesh\_health\_test\_t::prev\_test\_id (C++ member), 377
- esp\_ble\_mesh\_health\_test\_t::registered\_faults (C++ member), 377
- esp\_ble\_mesh\_health\_test\_t::test\_ids (C++ member), 377
- ESP\_BLE\_MESH\_HEARTBEAT\_FILTER\_ACCEPTLIST (C 宏), 285
- ESP\_BLE\_MESH\_HEARTBEAT\_FILTER\_ADD (C 宏), 285
- esp\_ble\_mesh\_heartbeat\_filter\_info\_t (C++ class), 279
- esp\_ble\_mesh\_heartbeat\_filter\_info\_t::hb\_dst (C++ member), 279
- esp\_ble\_mesh\_heartbeat\_filter\_info\_t::hb\_src (C++ member), 279
- ESP\_BLE\_MESH\_HEARTBEAT\_FILTER\_REJECTLIST (C 宏), 285
- ESP\_BLE\_MESH\_HEARTBEAT\_FILTER\_REMOVE (C 宏), 285
- ESP\_BLE\_MESH\_HEARTBEAT\_MESSAGE\_RECV\_EVT (C++ enumerator), 302
- ESP\_BLE\_MESH\_HOUSING\_OPENED\_ERROR (C 宏), 382
- ESP\_BLE\_MESH\_HOUSING\_OPENED\_WARNING (C 宏), 382
- esp\_ble\_mesh\_init (C++ function), 304
- esp\_ble\_mesh\_input\_action\_t (C++ enum), 298
- ESP\_BLE\_MESH\_INPUT\_NO\_CHANGE\_ERROR (C 宏), 382
- ESP\_BLE\_MESH\_INPUT\_NO\_CHANGE\_WARNING (C 宏), 382
- ESP\_BLE\_MESH\_INPUT\_OOB (C++ enumerator), 298
- ESP\_BLE\_MESH\_INPUT\_TOO\_HIGH\_ERROR (C 宏), 382
- ESP\_BLE\_MESH\_INPUT\_TOO\_HIGH\_WARNING (C 宏), 382
- ESP\_BLE\_MESH\_INPUT\_TOO\_LOW\_ERROR (C 宏), 382
- ESP\_BLE\_MESH\_INPUT\_TOO\_LOW\_WARNING (C 宏), 382
- ESP\_BLE\_MESH\_INTERNAL\_BUS\_ERROR (C 宏), 382
- ESP\_BLE\_MESH\_INTERNAL\_BUS\_WARNING (C 宏), 382
- ESP\_BLE\_MESH\_INVALID\_NODE\_INDEX (C 宏), 283
- ESP\_BLE\_MESH\_INVALID\_SCENE\_NUMBER (C 宏), 459
- ESP\_BLE\_MESH\_INVALID\_SENSOR\_PROPERTY\_ID (C 宏), 437
- ESP\_BLE\_MESH\_INVALID\_SENSOR\_SETTING\_PROPERTY\_ID (C 宏), 437
- ESP\_BLE\_MESH\_INVALID\_SETTINGS\_IDX (C 宏), 282
- esp\_ble\_mesh\_is\_model\_subscribed\_to\_group (C++ function), 305
- ESP\_BLE\_MESH\_KEY\_ANY (C 宏), 282
- ESP\_BLE\_MESH\_KEY\_DEV (C 宏), 282
- ESP\_BLE\_MESH\_KEY\_PRIMARY (C 宏), 282
- ESP\_BLE\_MESH\_KEY\_UNUSED (C 宏), 282
- esp\_ble\_mesh\_last\_msg\_info\_t (C++ class), 281
- esp\_ble\_mesh\_last\_msg\_info\_t::dst (C++ member), 281
- esp\_ble\_mesh\_last\_msg\_info\_t::src (C++ member), 281
- esp\_ble\_mesh\_last\_msg\_info\_t::tid (C++ member), 281
- esp\_ble\_mesh\_last\_msg\_info\_t::timestamp (C++ member), 281
- ESP\_BLE\_MESH\_LC\_FADE (C++ enumerator), 424
- ESP\_BLE\_MESH\_LC\_FADE\_ON (C++ enumerator), 424
- ESP\_BLE\_MESH\_LC\_FADE\_STANDBY\_AUTO (C++ enumerator), 424
- ESP\_BLE\_MESH\_LC\_FADE\_STANDBY\_MANUAL (C++ enumerator), 424
- ESP\_BLE\_MESH\_LC\_OFF (C++ enumerator), 424
- ESP\_BLE\_MESH\_LC\_PROLONG (C++ enumerator), 424
- ESP\_BLE\_MESH\_LC\_RUN (C++ enumerator), 424
- ESP\_BLE\_MESH\_LC\_STANDBY (C++ enumerator), 424
- esp\_ble\_mesh\_lc\_state\_t (C++ enum), 424
- esp\_ble\_mesh\_light\_client\_cb\_event\_t (C++ enum), 424
- esp\_ble\_mesh\_light\_client\_cb\_param\_t (C++ class), 400
- esp\_ble\_mesh\_light\_client\_cb\_param\_t::error\_code (C++ member), 401
- esp\_ble\_mesh\_light\_client\_cb\_param\_t::params (C++ member), 401
- esp\_ble\_mesh\_light\_client\_cb\_param\_t::status\_cb (C++ member), 401
- esp\_ble\_mesh\_light\_client\_cb\_t (C++ type), 424
- ESP\_BLE\_MESH\_LIGHT\_CLIENT\_EVT\_MAX (C++ enumerator), 424
- esp\_ble\_mesh\_light\_client\_get\_state (C++ function), 383
- ESP\_BLE\_MESH\_LIGHT\_CLIENT\_GET\_STATE\_EVT (C++ enumerator), 424

---

esp\_ble\_mesh\_light\_client\_get\_state\_t esp\_ble\_mesh\_light\_client\_status\_cb\_t::ctl\_temper  
 (C++ union), 384 (C++ member), 385  
 esp\_ble\_mesh\_light\_client\_get\_state\_t: esp\_ble\_mesh\_light\_client\_status\_cb\_t::ctl\_temper  
 (C++ member), 384 (C++ member), 385  
 ESP\_BLE\_MESH\_LIGHT\_CLIENT\_PUBLISH\_EVT esp\_ble\_mesh\_light\_client\_status\_cb\_t::hsl\_defaul  
 (C++ enumerator), 424 (C++ member), 386  
 esp\_ble\_mesh\_light\_client\_set\_state esp\_ble\_mesh\_light\_client\_status\_cb\_t::hsl\_hue\_st  
 (C++ function), 383 (C++ member), 386  
 ESP\_BLE\_MESH\_LIGHT\_CLIENT\_SET\_STATE\_EVT esp\_ble\_mesh\_light\_client\_status\_cb\_t::hsl\_range\_  
 (C++ enumerator), 424 (C++ member), 386  
 esp\_ble\_mesh\_light\_client\_set\_state\_t esp\_ble\_mesh\_light\_client\_status\_cb\_t::hsl\_satura  
 (C++ union), 384 (C++ member), 386  
 esp\_ble\_mesh\_light\_client\_set\_state\_t: esp\_ble\_mesh\_light\_client\_status\_cb\_t::hsl\_status  
 (C++ member), 384 (C++ member), 386  
 esp\_ble\_mesh\_light\_client\_set\_state\_t: esp\_ble\_mesh\_light\_client\_status\_cb\_t::hsl\_target  
 (C++ member), 384 (C++ member), 386  
 esp\_ble\_mesh\_light\_client\_set\_state\_t: esp\_ble\_mesh\_light\_client\_status\_cb\_t::lc\_light\_o  
 (C++ member), 384 (C++ member), 386  
 esp\_ble\_mesh\_light\_client\_set\_state\_t: esp\_ble\_mesh\_light\_client\_status\_cb\_t::lc\_mode\_st  
 (C++ member), 384 (C++ member), 386  
 esp\_ble\_mesh\_light\_client\_set\_state\_t: esp\_ble\_mesh\_light\_client\_status\_cb\_t::lc\_om\_stat  
 (C++ member), 385 (C++ member), 386  
 esp\_ble\_mesh\_light\_client\_set\_state\_t: esp\_ble\_mesh\_light\_client\_status\_cb\_t::lc\_propert  
 (C++ member), 384 (C++ member), 386  
 esp\_ble\_mesh\_light\_client\_set\_state\_t: esp\_ble\_mesh\_light\_client\_status\_cb\_t::lightness\_  
 (C++ member), 385 (C++ member), 385  
 esp\_ble\_mesh\_light\_client\_set\_state\_t: esp\_ble\_mesh\_light\_client\_status\_cb\_t::lightness\_  
 (C++ member), 385 (C++ member), 385  
 esp\_ble\_mesh\_light\_client\_set\_state\_t: esp\_ble\_mesh\_light\_client\_status\_cb\_t::lightness\_  
 (C++ member), 384 (C++ member), 385  
 esp\_ble\_mesh\_light\_client\_set\_state\_t: esp\_ble\_mesh\_light\_client\_status\_cb\_t::lightness\_  
 (C++ member), 385 (C++ member), 385  
 esp\_ble\_mesh\_light\_client\_set\_state\_t: esp\_ble\_mesh\_light\_client\_status\_cb\_t::lightness\_  
 (C++ member), 385 (C++ member), 385  
 esp\_ble\_mesh\_light\_client\_set\_state\_t: esp\_ble\_mesh\_light\_client\_status\_cb\_t::lightness\_  
 (C++ member), 385 (C++ member), 385  
 esp\_ble\_mesh\_light\_client\_set\_state\_t: esp\_ble\_mesh\_light\_client\_status\_cb\_t::xyl\_defaul  
 (C++ member), 385 (C++ member), 386  
 esp\_ble\_mesh\_light\_client\_set\_state\_t: esp\_ble\_mesh\_light\_client\_status\_cb\_t::xyl\_range\_  
 (C++ member), 385 (C++ member), 386  
 esp\_ble\_mesh\_light\_client\_set\_state\_t: esp\_ble\_mesh\_light\_client\_status\_cb\_t::xyl\_status  
 (C++ member), 384 (C++ member), 386  
 esp\_ble\_mesh\_light\_client\_set\_state\_t: esp\_ble\_mesh\_light\_client\_status\_cb\_t::xyl\_target  
 (C++ member), 384 (C++ member), 386  
 esp\_ble\_mesh\_light\_client\_set\_state\_t: ESP\_BLE\_MESH\_LIGHT\_CLIENT\_TIMEOUT\_EVT  
 (C++ member), 384 (C++ enumerator), 424  
 esp\_ble\_mesh\_light\_client\_set\_state\_t: esp\_ble\_mesh\_light\_control\_t (C++ class),  
 (C++ member), 384 409  
 esp\_ble\_mesh\_light\_client\_set\_state\_t: esp\_ble\_mesh\_light\_control\_t::prop\_state  
 (C++ member), 385 (C++ member), 410  
 esp\_ble\_mesh\_light\_client\_set\_state\_t: esp\_ble\_mesh\_light\_control\_t::state  
 (C++ member), 385 (C++ member), 410  
 esp\_ble\_mesh\_light\_client\_set\_state\_t: esp\_ble\_mesh\_light\_control\_t::state\_machine  
 (C++ member), 385 (C++ member), 410  
 esp\_ble\_mesh\_light\_client\_status\_cb\_t esp\_ble\_mesh\_light\_ctl\_default\_set\_t  
 (C++ union), 385 (C++ class), 391  
 esp\_ble\_mesh\_light\_client\_status\_cb\_t: esp\_ble\_mesh\_light\_ctl\_default\_set\_t::delta\_uv  
 (C++ member), 386 (C++ member), 391  
 esp\_ble\_mesh\_light\_client\_status\_cb\_t: esp\_ble\_mesh\_light\_ctl\_default\_set\_t::lightness  
 (C++ member), 385 (C++ member), 391

esp\_ble\_mesh\_light\_ctl\_default\_set\_t::temperature esp\_ble\_mesh\_light\_ctl\_state\_t::delta\_uv\_default  
 (C++ member), 391 (C++ member), 402  
 esp\_ble\_mesh\_light\_ctl\_default\_status\_delta esp\_ble\_mesh\_light\_ctl\_state\_t::lightness  
 (C++ class), 396 (C++ member), 402  
 esp\_ble\_mesh\_light\_ctl\_default\_status\_delta esp\_ble\_mesh\_light\_ctl\_state\_t::lightness\_default  
 (C++ member), 397 (C++ member), 402  
 esp\_ble\_mesh\_light\_ctl\_default\_status\_delta esp\_ble\_mesh\_light\_ctl\_state\_t::status\_code  
 (C++ member), 397 (C++ member), 402  
 esp\_ble\_mesh\_light\_ctl\_default\_status\_delta esp\_ble\_mesh\_light\_ctl\_state\_t::target\_delta\_uv  
 (C++ member), 397 (C++ member), 402  
 ESP\_BLE\_MESH\_LIGHT\_CTL\_LIGHTNESS\_STATE esp\_ble\_mesh\_light\_ctl\_state\_t::target\_lightness  
 (C++ enumerator), 303 (C++ member), 402  
 esp\_ble\_mesh\_light\_ctl\_set\_t (C++ class), esp\_ble\_mesh\_light\_ctl\_state\_t::target\_temperature  
 390 (C++ member), 402  
 esp\_ble\_mesh\_light\_ctl\_set\_t::ctl\_delta esp\_ble\_mesh\_light\_ctl\_state\_t::temperature  
 (C++ member), 390 (C++ member), 402  
 esp\_ble\_mesh\_light\_ctl\_set\_t::ctl\_lightness esp\_ble\_mesh\_light\_ctl\_state\_t::temperature\_default  
 (C++ member), 390 (C++ member), 402  
 esp\_ble\_mesh\_light\_ctl\_set\_t::ctl\_temperature esp\_ble\_mesh\_light\_ctl\_state\_t::temperature\_range  
 (C++ member), 390 (C++ member), 402  
 esp\_ble\_mesh\_light\_ctl\_set\_t::delay esp\_ble\_mesh\_light\_ctl\_state\_t::temperature\_range  
 (C++ member), 390 (C++ member), 402  
 esp\_ble\_mesh\_light\_ctl\_set\_t::op\_en esp\_ble\_mesh\_light\_ctl\_status\_cb\_t  
 (C++ member), 390 (C++ class), 395  
 esp\_ble\_mesh\_light\_ctl\_set\_t::tid esp\_ble\_mesh\_light\_ctl\_status\_cb\_t::op\_en  
 (C++ member), 390 (C++ member), 396  
 esp\_ble\_mesh\_light\_ctl\_set\_t::trans\_time esp\_ble\_mesh\_light\_ctl\_status\_cb\_t::present\_ctl\_l  
 (C++ member), 390 (C++ member), 396  
 esp\_ble\_mesh\_light\_ctl\_setup\_srv\_t esp\_ble\_mesh\_light\_ctl\_status\_cb\_t::present\_ctl\_t  
 (C++ class), 403 (C++ member), 396  
 esp\_ble\_mesh\_light\_ctl\_setup\_srv\_t::mode esp\_ble\_mesh\_light\_ctl\_status\_cb\_t::remain\_time  
 (C++ member), 403 (C++ member), 396  
 esp\_ble\_mesh\_light\_ctl\_setup\_srv\_t::rsp\_ctrl esp\_ble\_mesh\_light\_ctl\_status\_cb\_t::target\_ctl\_li  
 (C++ member), 403 (C++ member), 396  
 esp\_ble\_mesh\_light\_ctl\_setup\_srv\_t::state esp\_ble\_mesh\_light\_ctl\_status\_cb\_t::target\_ctl\_te  
 (C++ member), 403 (C++ member), 396  
 esp\_ble\_mesh\_light\_ctl\_srv\_t (C++ class), ESP\_BLE\_MESH\_LIGHT\_CTL\_TEMP\_DELTA\_UV\_STATE  
 402 (C++ enumerator), 303  
 esp\_ble\_mesh\_light\_ctl\_srv\_t::last esp\_ble\_mesh\_light\_ctl\_temp\_srv\_t  
 (C++ member), 403 (C++ class), 403  
 esp\_ble\_mesh\_light\_ctl\_srv\_t::model esp\_ble\_mesh\_light\_ctl\_temp\_srv\_t::last  
 (C++ member), 403 (C++ member), 403  
 esp\_ble\_mesh\_light\_ctl\_srv\_t::rsp\_ctrl esp\_ble\_mesh\_light\_ctl\_temp\_srv\_t::model  
 (C++ member), 403 (C++ member), 403  
 esp\_ble\_mesh\_light\_ctl\_srv\_t::state esp\_ble\_mesh\_light\_ctl\_temp\_srv\_t::rsp\_ctrl  
 (C++ member), 403 (C++ member), 403  
 esp\_ble\_mesh\_light\_ctl\_srv\_t::transition esp\_ble\_mesh\_light\_ctl\_temp\_srv\_t::state  
 (C++ member), 403 (C++ member), 403  
 esp\_ble\_mesh\_light\_ctl\_srv\_t::tt\_delta\_delta esp\_ble\_mesh\_light\_ctl\_temp\_srv\_t::transition  
 (C++ member), 403 (C++ member), 403  
 esp\_ble\_mesh\_light\_ctl\_srv\_t::tt\_delta\_delta esp\_ble\_mesh\_light\_ctl\_temp\_srv\_t::tt\_delta\_delta  
 (C++ member), 403 (C++ member), 403  
 esp\_ble\_mesh\_light\_ctl\_srv\_t::tt\_delta\_delta esp\_ble\_mesh\_light\_ctl\_temp\_srv\_t::tt\_delta\_tempe  
 (C++ member), 403 (C++ member), 403  
 esp\_ble\_mesh\_light\_ctl\_state\_t (C++ esp\_ble\_mesh\_light\_ctl\_temperature\_range\_set\_t  
 class), 402 (C++ class), 391  
 esp\_ble\_mesh\_light\_ctl\_state\_t::delta\_uv esp\_ble\_mesh\_light\_ctl\_temperature\_range\_set\_t::r  
 (C++ member), 402 (C++ member), 391

---

esp\_ble\_mesh\_light\_ctl\_temperature\_range\_set\_t (C++ member), 391  
 esp\_ble\_mesh\_light\_ctl\_temperature\_range\_set\_t (C++ class), 396  
 esp\_ble\_mesh\_light\_ctl\_temperature\_range\_set\_t (C++ member), 396  
 esp\_ble\_mesh\_light\_ctl\_temperature\_range\_set\_t (C++ member), 396  
 esp\_ble\_mesh\_light\_ctl\_temperature\_range\_set\_t (C++ member), 396  
 esp\_ble\_mesh\_light\_ctl\_temperature\_range\_set\_t (C++ member), 396  
 esp\_ble\_mesh\_light\_ctl\_temperature\_set\_delta\_hsl\_hue\_srv\_t (C++ member), 405  
 esp\_ble\_mesh\_light\_ctl\_temperature\_set\_delta\_hsl\_hue\_srv\_t (C++ member), 405  
 esp\_ble\_mesh\_light\_ctl\_temperature\_set\_delta\_hsl\_hue\_srv\_t (C++ member), 405  
 esp\_ble\_mesh\_light\_ctl\_temperature\_set\_delta\_hsl\_hue\_srv\_t (C++ member), 405  
 esp\_ble\_mesh\_light\_ctl\_temperature\_set\_delta\_hsl\_hue\_srv\_t (C++ member), 405  
 esp\_ble\_mesh\_light\_ctl\_temperature\_set\_delta\_hsl\_hue\_srv\_t (C++ member), 405  
 esp\_ble\_mesh\_light\_ctl\_temperature\_set\_delta\_hsl\_hue\_srv\_t (C++ member), 405  
 esp\_ble\_mesh\_light\_ctl\_temperature\_set\_delta\_hsl\_hue\_srv\_t (C++ member), 405  
 esp\_ble\_mesh\_light\_ctl\_temperature\_set\_delta\_hsl\_hue\_srv\_t (C++ member), 405  
 ESP\_BLE\_MESH\_LIGHT\_HSL\_HUE\_STATE (C++ enumerator), 303  
 esp\_ble\_mesh\_light\_ctl\_temperature\_status\_delta\_hsl\_hue\_status\_cb\_t (C++ class), 397  
 esp\_ble\_mesh\_light\_ctl\_temperature\_status\_delta\_hsl\_hue\_status\_cb\_t (C++ member), 397  
 esp\_ble\_mesh\_light\_ctl\_temperature\_status\_delta\_hsl\_hue\_status\_cb\_t (C++ member), 397  
 esp\_ble\_mesh\_light\_ctl\_temperature\_status\_delta\_hsl\_hue\_status\_cb\_t (C++ member), 398  
 esp\_ble\_mesh\_light\_ctl\_temperature\_status\_delta\_hsl\_hue\_status\_cb\_t (C++ member), 397  
 esp\_ble\_mesh\_light\_ctl\_temperature\_status\_delta\_hsl\_hue\_status\_cb\_t (C++ member), 397  
 ESP\_BLE\_MESH\_LIGHT\_HSL\_HUE\_TARGET\_STATE (C++ enumerator), 303  
 esp\_ble\_mesh\_light\_ctl\_temperature\_status\_delta\_hsl\_hue\_target\_set\_t (C++ class), 392  
 esp\_ble\_mesh\_light\_hsl\_default\_set\_t (C++ class), 392  
 esp\_ble\_mesh\_light\_hsl\_default\_set\_t (C++ member), 392  
 esp\_ble\_mesh\_light\_hsl\_default\_set\_t (C++ member), 392  
 esp\_ble\_mesh\_light\_hsl\_default\_set\_t (C++ member), 392  
 esp\_ble\_mesh\_light\_hsl\_default\_status\_delta\_hsl\_hue\_status\_cb\_t (C++ class), 398  
 esp\_ble\_mesh\_light\_hsl\_default\_status\_delta\_hsl\_hue\_status\_cb\_t (C++ member), 398  
 esp\_ble\_mesh\_light\_hsl\_default\_status\_delta\_hsl\_hue\_status\_cb\_t (C++ member), 398  
 esp\_ble\_mesh\_light\_hsl\_default\_status\_delta\_hsl\_hue\_status\_cb\_t (C++ member), 398  
 esp\_ble\_mesh\_light\_hsl\_hue\_set\_t (C++ class), 391  
 esp\_ble\_mesh\_light\_hsl\_hue\_set\_t (C++ member), 392  
 esp\_ble\_mesh\_light\_hsl\_range\_set\_t (C++ member), 392  
 esp\_ble\_mesh\_light\_hsl\_range\_set\_t (C++ member), 392  
 esp\_ble\_mesh\_light\_hsl\_range\_set\_t (C++ member), 392  
 esp\_ble\_mesh\_light\_hsl\_range\_set\_t (C++ member), 392  
 esp\_ble\_mesh\_light\_hsl\_range\_status\_cb\_t (C++ class), 398  
 esp\_ble\_mesh\_light\_hsl\_range\_status\_cb\_t (C++ member), 398  
 esp\_ble\_mesh\_light\_hsl\_range\_status\_cb\_t (C++ member), 398  
 esp\_ble\_mesh\_light\_hsl\_range\_status\_cb\_t (C++ member), 398  
 esp\_ble\_mesh\_light\_hsl\_range\_status\_cb\_t (C++ member), 398  
 esp\_ble\_mesh\_light\_hsl\_range\_status\_cb\_t (C++ member), 398



esp\_ble\_mesh\_light\_hsl\_sat\_srv\_t (C++ class), 405  
 esp\_ble\_mesh\_light\_hsl\_sat\_srv\_t::last (C++ member), 405  
 esp\_ble\_mesh\_light\_hsl\_sat\_srv\_t::model (C++ member), 405  
 esp\_ble\_mesh\_light\_hsl\_sat\_srv\_t::rsp\_ctrl (C++ member), 404  
 esp\_ble\_mesh\_light\_hsl\_sat\_srv\_t::state (C++ member), 405  
 esp\_ble\_mesh\_light\_hsl\_sat\_srv\_t::transition (C++ member), 404  
 esp\_ble\_mesh\_light\_hsl\_sat\_srv\_t::tt\_delta\_hue (C++ member), 406  
 esp\_ble\_mesh\_light\_hsl\_saturation\_set\_t (C++ class), 392  
 esp\_ble\_mesh\_light\_hsl\_saturation\_set\_t::delay (C++ member), 392  
 esp\_ble\_mesh\_light\_hsl\_saturation\_set\_t::enable (C++ member), 392  
 esp\_ble\_mesh\_light\_hsl\_saturation\_set\_t::saturation (C++ member), 392  
 ESP\_BLE\_MESH\_LIGHT\_HSL\_STATE (C++ enumerator), 303  
 esp\_ble\_mesh\_light\_hsl\_saturation\_set\_t::state (C++ member), 392  
 ESP\_BLE\_MESH\_LIGHT\_HSL\_SATURATION\_STATE (C++ enumerator), 303  
 esp\_ble\_mesh\_light\_hsl\_saturation\_status\_cb\_t (C++ class), 398  
 esp\_ble\_mesh\_light\_hsl\_saturation\_status\_cb\_t::hue\_default (C++ member), 398  
 esp\_ble\_mesh\_light\_hsl\_saturation\_status\_cb\_t::hue\_range\_max (C++ member), 398  
 esp\_ble\_mesh\_light\_hsl\_saturation\_status\_cb\_t::hue\_range\_min (C++ member), 398  
 esp\_ble\_mesh\_light\_hsl\_saturation\_status\_cb\_t::lightness (C++ member), 398  
 esp\_ble\_mesh\_light\_hsl\_saturation\_status\_cb\_t::lightness\_default (C++ member), 398  
 esp\_ble\_mesh\_light\_hsl\_set\_t (C++ class), 391  
 esp\_ble\_mesh\_light\_hsl\_set\_t::delay (C++ member), 391  
 esp\_ble\_mesh\_light\_hsl\_set\_t::hsl\_hue (C++ member), 391  
 esp\_ble\_mesh\_light\_hsl\_set\_t::hsl\_lightness (C++ member), 391  
 esp\_ble\_mesh\_light\_hsl\_set\_t::hsl\_saturation (C++ member), 391  
 esp\_ble\_mesh\_light\_hsl\_set\_t::op\_en (C++ member), 391  
 esp\_ble\_mesh\_light\_hsl\_set\_t::tid (C++ member), 391  
 esp\_ble\_mesh\_light\_hsl\_set\_t::trans\_time (C++ member), 391  
 esp\_ble\_mesh\_light\_hsl\_setup\_srv\_t (C++ class), 405  
 esp\_ble\_mesh\_light\_hsl\_setup\_srv\_t::model (C++ member), 405  
 esp\_ble\_mesh\_light\_hsl\_state\_t (C++ class), 403  
 esp\_ble\_mesh\_light\_hsl\_state\_t::hue (C++ member), 404  
 esp\_ble\_mesh\_light\_hsl\_state\_t::hue\_default (C++ member), 404  
 esp\_ble\_mesh\_light\_hsl\_state\_t::hue\_range\_max (C++ member), 404  
 esp\_ble\_mesh\_light\_hsl\_state\_t::hue\_range\_min (C++ member), 404  
 esp\_ble\_mesh\_light\_hsl\_state\_t::lightness (C++ member), 404  
 esp\_ble\_mesh\_light\_hsl\_state\_t::lightness\_default (C++ member), 404  
 esp\_ble\_mesh\_light\_hsl\_state\_t::saturation (C++ member), 404  
 esp\_ble\_mesh\_light\_hsl\_state\_t::saturation\_default (C++ member), 404  
 esp\_ble\_mesh\_light\_hsl\_state\_t::saturation\_range (C++ member), 404  
 esp\_ble\_mesh\_light\_hsl\_state\_t::saturation\_range\_min (C++ member), 404  
 esp\_ble\_mesh\_light\_hsl\_state\_t::status\_code (C++ member), 404  
 esp\_ble\_mesh\_light\_hsl\_state\_t::target\_hue (C++ member), 404  
 esp\_ble\_mesh\_light\_hsl\_state\_t::target\_lightness (C++ member), 404  
 esp\_ble\_mesh\_light\_hsl\_state\_t::target\_saturation (C++ member), 404  
 esp\_ble\_mesh\_light\_hsl\_status\_cb\_t (C++ class), 397  
 esp\_ble\_mesh\_light\_hsl\_status\_cb\_t::hsl\_hue (C++ member), 397



---

esp\_ble\_mesh\_light\_lc\_setup\_srv\_t (C++ class), 410  
 esp\_ble\_mesh\_light\_lc\_setup\_srv\_t::lc (C++ member), 410  
 esp\_ble\_mesh\_light\_lc\_setup\_srv\_t::mode (C++ member), 410  
 esp\_ble\_mesh\_light\_lc\_setup\_srv\_t::rsp\_ctrl (C++ member), 410  
 esp\_ble\_mesh\_light\_lc\_srv\_t (C++ class), 410  
 esp\_ble\_mesh\_light\_lc\_srv\_t::last (C++ member), 410  
 esp\_ble\_mesh\_light\_lc\_srv\_t::lc (C++ member), 410  
 esp\_ble\_mesh\_light\_lc\_srv\_t::model (C++ member), 410  
 esp\_ble\_mesh\_light\_lc\_srv\_t::rsp\_ctrl (C++ member), 410  
 esp\_ble\_mesh\_light\_lc\_srv\_t::transition (C++ member), 410  
 esp\_ble\_mesh\_light\_lc\_state\_machine\_t (C++ class), 409  
 esp\_ble\_mesh\_light\_lc\_state\_machine\_t::enable (C++ member), 409  
 esp\_ble\_mesh\_light\_lc\_state\_machine\_t::enable\_by\_light (C++ member), 409  
 esp\_ble\_mesh\_light\_lc\_state\_machine\_t::enable\_by\_magnitude (C++ member), 409  
 esp\_ble\_mesh\_light\_lc\_state\_machine\_t::enable\_by\_state (C++ member), 409  
 esp\_ble\_mesh\_light\_lc\_state\_machine\_t::enable\_by\_timer (C++ member), 409  
 esp\_ble\_mesh\_light\_lc\_state\_machine\_t::enable\_time (C++ member), 409  
 esp\_ble\_mesh\_light\_lc\_state\_t (C++ class), 407  
 esp\_ble\_mesh\_light\_lc\_state\_t::ambient\_light\_level (C++ member), 407  
 esp\_ble\_mesh\_light\_lc\_state\_t::light\_on\_off (C++ member), 407  
 esp\_ble\_mesh\_light\_lc\_state\_t::linear\_output (C++ member), 407  
 esp\_ble\_mesh\_light\_lc\_state\_t::mode (C++ member), 407  
 esp\_ble\_mesh\_light\_lc\_state\_t::occupancy (C++ member), 407  
 esp\_ble\_mesh\_light\_lc\_state\_t::occupancy\_mode (C++ member), 407  
 esp\_ble\_mesh\_light\_lc\_state\_t::target\_light\_level (C++ member), 407  
 ESP\_BLE\_MESH\_LIGHT\_LIGHTNESS\_ACTUAL\_STATE (C++ enumerator), 303  
 esp\_ble\_mesh\_light\_lightness\_default\_set\_t (C++ class), 389  
 esp\_ble\_mesh\_light\_lightness\_default\_set\_t::lightness (C++ member), 389  
 esp\_ble\_mesh\_light\_lightness\_default\_set\_t::lightness\_time (C++ member), 389  
 esp\_ble\_mesh\_light\_lightness\_default\_status\_cb\_t (C++ class), 395  
 esp\_ble\_mesh\_light\_lightness\_default\_status\_cb\_t::lightness (C++ member), 395  
 esp\_ble\_mesh\_light\_lightness\_last\_status\_cb\_t (C++ class), 395  
 esp\_ble\_mesh\_light\_lightness\_last\_status\_cb\_t::lightness (C++ member), 395  
 esp\_ble\_mesh\_light\_lightness\_linear\_set\_t (C++ class), 389  
 esp\_ble\_mesh\_light\_lightness\_linear\_set\_t::delay (C++ member), 389  
 esp\_ble\_mesh\_light\_lightness\_linear\_set\_t::lightness (C++ member), 389  
 esp\_ble\_mesh\_light\_lightness\_linear\_set\_t::open (C++ member), 389  
 esp\_ble\_mesh\_light\_lightness\_linear\_set\_t::tid (C++ member), 389  
 esp\_ble\_mesh\_light\_lightness\_linear\_set\_t::transition (C++ member), 389  
 ESP\_BLE\_MESH\_LIGHT\_LIGHTNESS\_LINEAR\_STATE (C++ enumerator), 303  
 esp\_ble\_mesh\_light\_lightness\_linear\_status\_cb\_t (C++ class), 395  
 esp\_ble\_mesh\_light\_lightness\_linear\_status\_cb\_t::lightness (C++ member), 395  
 esp\_ble\_mesh\_light\_lightness\_linear\_status\_cb\_t::lightness\_time (C++ member), 395  
 esp\_ble\_mesh\_light\_lightness\_linear\_status\_cb\_t::state (C++ member), 395  
 esp\_ble\_mesh\_light\_lightness\_range\_set\_t (C++ class), 390  
 esp\_ble\_mesh\_light\_lightness\_range\_set\_t::range\_min (C++ member), 390  
 esp\_ble\_mesh\_light\_lightness\_range\_set\_t::range\_max (C++ member), 390  
 esp\_ble\_mesh\_light\_lightness\_range\_status\_cb\_t (C++ class), 395  
 esp\_ble\_mesh\_light\_lightness\_range\_status\_cb\_t::range\_min (C++ member), 395  
 esp\_ble\_mesh\_light\_lightness\_range\_status\_cb\_t::range\_max (C++ member), 395  
 esp\_ble\_mesh\_light\_lightness\_set\_t (C++ class), 389  
 esp\_ble\_mesh\_light\_lightness\_set\_t::delay (C++ member), 389  
 esp\_ble\_mesh\_light\_lightness\_set\_t::lightness (C++ member), 389  
 esp\_ble\_mesh\_light\_lightness\_set\_t::open (C++ member), 389  
 esp\_ble\_mesh\_light\_lightness\_set\_t::tid (C++ member), 389  
 esp\_ble\_mesh\_light\_lightness\_set\_t::transition\_time (C++ member), 389

esp\_ble\_mesh\_light\_lightness\_setup\_srv\_t esp\_ble\_mesh\_light\_xyl\_default\_set\_t  
 (C++ class), 402 (C++ class), 393  
 esp\_ble\_mesh\_light\_lightness\_setup\_srv\_t::act esp\_ble\_mesh\_light\_xyl\_default\_set\_t::lightness  
 (C++ member), 402 (C++ member), 393  
 esp\_ble\_mesh\_light\_lightness\_setup\_srv\_t::idle esp\_ble\_mesh\_light\_xyl\_default\_set\_t::xyl\_x  
 (C++ member), 402 (C++ member), 393  
 esp\_ble\_mesh\_light\_lightness\_setup\_srv\_t::state esp\_ble\_mesh\_light\_xyl\_default\_set\_t::xyl\_y  
 (C++ member), 402 (C++ member), 393  
 esp\_ble\_mesh\_light\_lightness\_srv\_t esp\_ble\_mesh\_light\_xyl\_default\_status\_cb\_t  
 (C++ class), 401 (C++ class), 399  
 esp\_ble\_mesh\_light\_lightness\_srv\_t::act esp\_ble\_mesh\_light\_xyl\_default\_status\_cb\_t::light  
 (C++ member), 401 (C++ member), 399  
 esp\_ble\_mesh\_light\_lightness\_srv\_t::last esp\_ble\_mesh\_light\_xyl\_default\_status\_cb\_t::xyl\_x  
 (C++ member), 401 (C++ member), 399  
 esp\_ble\_mesh\_light\_lightness\_srv\_t::line esp\_ble\_mesh\_light\_xyl\_default\_status\_cb\_t::xyl\_y  
 (C++ member), 401 (C++ member), 399  
 esp\_ble\_mesh\_light\_lightness\_srv\_t::mode ESP\_BLE\_MESH\_LIGHT\_XYL\_LIGHTNESS\_STATE  
 (C++ member), 401 (C++ enumerator), 303  
 esp\_ble\_mesh\_light\_lightness\_srv\_t::rsp esp\_ble\_mesh\_light\_xyl\_range\_set\_t  
 (C++ member), 401 (C++ class), 393  
 esp\_ble\_mesh\_light\_lightness\_srv\_t::state esp\_ble\_mesh\_light\_xyl\_range\_set\_t::xyl\_x\_range\_m  
 (C++ member), 401 (C++ member), 393  
 esp\_ble\_mesh\_light\_lightness\_srv\_t::tt esp\_ble\_mesh\_light\_xyl\_range\_set\_t::xyl\_x\_range\_m  
 (C++ member), 401 (C++ member), 393  
 esp\_ble\_mesh\_light\_lightness\_srv\_t::tt esp\_ble\_mesh\_light\_xyl\_range\_set\_t::xyl\_y\_range\_m  
 (C++ member), 402 (C++ member), 393  
 esp\_ble\_mesh\_light\_lightness\_state\_t esp\_ble\_mesh\_light\_xyl\_range\_set\_t::xyl\_y\_range\_m  
 (C++ class), 401 (C++ member), 393  
 esp\_ble\_mesh\_light\_lightness\_state\_t::light esp\_ble\_mesh\_light\_xyl\_range\_status\_cb\_t  
 (C++ member), 401 (C++ class), 399  
 esp\_ble\_mesh\_light\_lightness\_state\_t::light esp\_ble\_mesh\_light\_xyl\_range\_status\_cb\_t::status  
 (C++ member), 401 (C++ member), 399  
 esp\_ble\_mesh\_light\_lightness\_state\_t::light esp\_ble\_mesh\_light\_xyl\_range\_status\_cb\_t::xyl\_x\_r  
 (C++ member), 401 (C++ member), 399  
 esp\_ble\_mesh\_light\_lightness\_state\_t::light esp\_ble\_mesh\_light\_xyl\_range\_status\_cb\_t::xyl\_x\_r  
 (C++ member), 401 (C++ member), 399  
 esp\_ble\_mesh\_light\_lightness\_state\_t::light esp\_ble\_mesh\_light\_xyl\_range\_status\_cb\_t::xyl\_x\_r  
 (C++ member), 401 (C++ member), 400  
 esp\_ble\_mesh\_light\_lightness\_state\_t::light esp\_ble\_mesh\_light\_xyl\_range\_status\_cb\_t::xyl\_y\_r  
 (C++ member), 401 (C++ member), 400  
 esp\_ble\_mesh\_light\_lightness\_state\_t::state esp\_ble\_mesh\_light\_xyl\_set\_t (C++ class),  
 (C++ member), 401 392  
 esp\_ble\_mesh\_light\_lightness\_state\_t::temp esp\_ble\_mesh\_light\_xyl\_set\_t::delay  
 (C++ member), 401 (C++ member), 393  
 esp\_ble\_mesh\_light\_lightness\_state\_t::temp esp\_ble\_mesh\_light\_xyl\_set\_t::op\_en  
 (C++ member), 401 (C++ member), 393  
 esp\_ble\_mesh\_light\_lightness\_status\_cb\_t esp\_ble\_mesh\_light\_xyl\_set\_t::tid  
 (C++ class), 394 (C++ member), 393  
 esp\_ble\_mesh\_light\_lightness\_status\_cb\_t esp\_ble\_mesh\_light\_xyl\_set\_t::trans\_time  
 (C++ member), 395 (C++ member), 393  
 esp\_ble\_mesh\_light\_lightness\_status\_cb\_t esp\_ble\_mesh\_light\_xyl\_set\_t::xyl\_lightness  
 (C++ member), 395 (C++ member), 393  
 esp\_ble\_mesh\_light\_lightness\_status\_cb\_t esp\_ble\_mesh\_light\_xyl\_set\_t::xyl\_x  
 (C++ member), 395 (C++ member), 393  
 esp\_ble\_mesh\_light\_lightness\_status\_cb\_t esp\_ble\_mesh\_light\_xyl\_set\_t::xyl\_y  
 (C++ member), 395 (C++ member), 393  
 esp\_ble\_mesh\_light\_message\_opcode\_t esp\_ble\_mesh\_light\_xyl\_setup\_srv\_t  
 (C++ type), 297 (C++ class), 407

---

esp\_ble\_mesh\_light\_xyl\_setup\_srv\_t::model (C++ member), 407  
 esp\_ble\_mesh\_light\_xyl\_setup\_srv\_t::model (C++ member), 407  
 esp\_ble\_mesh\_light\_xyl\_setup\_srv\_t::state (C++ member), 407  
 esp\_ble\_mesh\_light\_xyl\_srv\_t (C++ class), 406  
 esp\_ble\_mesh\_light\_xyl\_srv\_t::last (C++ member), 407  
 esp\_ble\_mesh\_light\_xyl\_srv\_t::model (C++ member), 406  
 esp\_ble\_mesh\_light\_xyl\_srv\_t::rsp\_ctrl (C++ member), 406  
 esp\_ble\_mesh\_light\_xyl\_srv\_t::state (C++ member), 406  
 esp\_ble\_mesh\_light\_xyl\_srv\_t::transition (C++ member), 407  
 esp\_ble\_mesh\_light\_xyl\_srv\_t::tt\_delta (C++ member), 407  
 esp\_ble\_mesh\_light\_xyl\_srv\_t::tt\_delta (C++ member), 407  
 esp\_ble\_mesh\_light\_xyl\_srv\_t::tt\_delta (C++ member), 407  
 esp\_ble\_mesh\_light\_xyl\_state\_t (C++ class), 406  
 esp\_ble\_mesh\_light\_xyl\_state\_t::lightness (C++ member), 406  
 esp\_ble\_mesh\_light\_xyl\_state\_t::lightness (C++ member), 406  
 esp\_ble\_mesh\_light\_xyl\_state\_t::status (C++ member), 406  
 esp\_ble\_mesh\_light\_xyl\_state\_t::target (C++ member), 406  
 esp\_ble\_mesh\_light\_xyl\_state\_t::target (C++ member), 406  
 esp\_ble\_mesh\_light\_xyl\_state\_t::target (C++ member), 406  
 esp\_ble\_mesh\_light\_xyl\_state\_t::x (C++ member), 406  
 esp\_ble\_mesh\_light\_xyl\_state\_t::x\_default (C++ member), 406  
 esp\_ble\_mesh\_light\_xyl\_state\_t::x\_range (C++ member), 406  
 esp\_ble\_mesh\_light\_xyl\_state\_t::x\_range (C++ member), 406  
 esp\_ble\_mesh\_light\_xyl\_state\_t::y (C++ member), 406  
 esp\_ble\_mesh\_light\_xyl\_state\_t::y\_default (C++ member), 406  
 esp\_ble\_mesh\_light\_xyl\_state\_t::y\_range (C++ member), 406  
 esp\_ble\_mesh\_light\_xyl\_state\_t::y\_range (C++ member), 406  
 esp\_ble\_mesh\_light\_xyl\_status\_cb\_t (C++ class), 398  
 esp\_ble\_mesh\_light\_xyl\_status\_cb\_t::op (C++ member), 399  
 esp\_ble\_mesh\_light\_xyl\_status\_cb\_t::remain\_time (C++ member), 399  
 esp\_ble\_mesh\_light\_xyl\_status\_cb\_t::xyl\_lightness (C++ member), 399  
 esp\_ble\_mesh\_light\_xyl\_status\_cb\_t::xyl\_x (C++ member), 399  
 esp\_ble\_mesh\_light\_xyl\_status\_cb\_t::xyl\_y (C++ member), 399  
 esp\_ble\_mesh\_light\_xyl\_target\_status\_cb\_t (C++ class), 399  
 esp\_ble\_mesh\_light\_xyl\_target\_status\_cb\_t::op\_en (C++ member), 399  
 esp\_ble\_mesh\_light\_xyl\_target\_status\_cb\_t::remain (C++ member), 399  
 esp\_ble\_mesh\_light\_xyl\_target\_status\_cb\_t::target (C++ member), 399  
 esp\_ble\_mesh\_light\_xyl\_target\_status\_cb\_t::target (C++ member), 399  
 esp\_ble\_mesh\_light\_xyl\_target\_status\_cb\_t::target (C++ member), 399  
 esp\_ble\_mesh\_lighting\_server\_cb\_event\_t (C++ enum), 424  
 esp\_ble\_mesh\_lighting\_server\_cb\_param\_t (C++ class), 420  
 esp\_ble\_mesh\_lighting\_server\_cb\_param\_t::ctx (C++ member), 420  
 esp\_ble\_mesh\_lighting\_server\_cb\_param\_t::model (C++ member), 420  
 esp\_ble\_mesh\_lighting\_server\_cb\_param\_t::value (C++ member), 420  
 esp\_ble\_mesh\_lighting\_server\_cb\_t (C++ type), 424  
 esp\_ble\_mesh\_lighting\_server\_cb\_value\_t (C++ union), 388  
 esp\_ble\_mesh\_lighting\_server\_cb\_value\_t::get (C++ member), 389  
 esp\_ble\_mesh\_lighting\_server\_cb\_value\_t::set (C++ member), 389  
 esp\_ble\_mesh\_lighting\_server\_cb\_value\_t::state\_ch (C++ member), 389  
 esp\_ble\_mesh\_lighting\_server\_cb\_value\_t::status (C++ member), 389  
 ESP\_BLE\_MESH\_LIGHTING\_SERVER\_EVT\_MAX (C++ enumerator), 425  
 ESP\_BLE\_MESH\_LIGHTING\_SERVER\_RECV\_GET\_MSG\_EVT (C++ enumerator), 425  
 esp\_ble\_mesh\_lighting\_server\_recv\_get\_msg\_t (C++ union), 387  
 esp\_ble\_mesh\_lighting\_server\_recv\_get\_msg\_t::lc\_p (C++ member), 387  
 ESP\_BLE\_MESH\_LIGHTING\_SERVER\_RECV\_SET\_MSG\_EVT (C++ enumerator), 425  
 esp\_ble\_mesh\_lighting\_server\_recv\_set\_msg\_t (C++ union), 387  
 esp\_ble\_mesh\_lighting\_server\_recv\_set\_msg\_t::ctl (C++ member), 388  
 esp\_ble\_mesh\_lighting\_server\_recv\_set\_msg\_t::ctl (C++ member), 388

esp\_ble\_mesh\_lighting\_server\_rcv\_set\_resp\_t::etm\_stemp\_lighting\_server\_state\_change\_t::hsl  
 (C++ member), 388 (C++ member), 387  
 esp\_ble\_mesh\_lighting\_server\_rcv\_set\_resp\_t::etm\_stemp\_lighting\_server\_state\_change\_t::hsl  
 (C++ member), 388 (C++ member), 387  
 esp\_ble\_mesh\_lighting\_server\_rcv\_set\_resp\_t::etm\_stemp\_lighting\_server\_state\_change\_t::hsl  
 (C++ member), 388 (C++ member), 387  
 esp\_ble\_mesh\_lighting\_server\_rcv\_set\_resp\_t::etm\_stemp\_faulting\_server\_state\_change\_t::lc  
 (C++ member), 388 (C++ member), 387  
 esp\_ble\_mesh\_lighting\_server\_rcv\_set\_resp\_t::etm\_stemp\_lighting\_server\_state\_change\_t::lc\_m  
 (C++ member), 388 (C++ member), 387  
 esp\_ble\_mesh\_lighting\_server\_rcv\_set\_resp\_t::etm\_stemp\_lighting\_server\_state\_change\_t::lc\_o  
 (C++ member), 388 (C++ member), 387  
 esp\_ble\_mesh\_lighting\_server\_rcv\_set\_resp\_t::etm\_stemp\_lighting\_server\_state\_change\_t::lc\_p  
 (C++ member), 388 (C++ member), 387  
 esp\_ble\_mesh\_lighting\_server\_rcv\_set\_resp\_t::etm\_stemp\_lighting\_server\_state\_change\_t::ligh  
 (C++ member), 388 (C++ member), 386  
 esp\_ble\_mesh\_lighting\_server\_rcv\_set\_resp\_t::etm\_stemp\_lighting\_server\_state\_change\_t::ligh  
 (C++ member), 388 (C++ member), 386  
 esp\_ble\_mesh\_lighting\_server\_rcv\_set\_resp\_t::etm\_stemp\_lighting\_server\_state\_change\_t::ligh  
 (C++ member), 388 (C++ member), 386  
 esp\_ble\_mesh\_lighting\_server\_rcv\_set\_resp\_t::etm\_stemp\_lighting\_server\_state\_change\_t::ligh  
 (C++ member), 388 (C++ member), 386  
 esp\_ble\_mesh\_lighting\_server\_rcv\_set\_resp\_t::etm\_stemp\_lighting\_server\_state\_change\_t::sens  
 (C++ member), 387 (C++ member), 387  
 esp\_ble\_mesh\_lighting\_server\_rcv\_set\_resp\_t::etm\_stemp\_lighting\_server\_state\_change\_t::xyl  
 (C++ member), 388 (C++ member), 387  
 esp\_ble\_mesh\_lighting\_server\_rcv\_set\_resp\_t::etm\_stemp\_lighting\_server\_state\_change\_t::xyl  
 (C++ member), 387 (C++ member), 387  
 esp\_ble\_mesh\_lighting\_server\_rcv\_set\_resp\_t::etm\_stemp\_lighting\_server\_state\_change\_t::xyl  
 (C++ member), 388 (C++ member), 387  
 esp\_ble\_mesh\_lighting\_server\_rcv\_set\_resp\_t::xylmesh\_lpn\_disable (C++ function),  
 (C++ member), 388 307  
 esp\_ble\_mesh\_lighting\_server\_rcv\_set\_resp\_t::xylmesh\_lpn\_disable (C++ member), 388  
 (C++ member), 388 (C++ enumerator), 302  
 esp\_ble\_mesh\_lighting\_server\_rcv\_set\_resp\_t::xylmesh\_lpn\_enable (C++ function), 307  
 (C++ member), 388 ESP\_BLE\_MESH\_LPN\_ENABLE\_COMP\_EVT (C++  
 ESP\_BLE\_MESH\_LIGHTING\_SERVER\_RECV\_STATUS\_MSG\_ENUMERATOR), 302  
 (C++ enumerator), 425 ESP\_BLE\_MESH\_LPN\_FRIENDSHIP\_ESTABLISH\_EVT  
 esp\_ble\_mesh\_lighting\_server\_rcv\_status\_msg\_t (C++ enumerator), 302  
 (C++ union), 388 ESP\_BLE\_MESH\_LPN\_FRIENDSHIP\_TERMINATE\_EVT  
 esp\_ble\_mesh\_lighting\_server\_rcv\_status\_msg\_t (C++ enumerator), 302  
 (C++ member), 388 esp\_ble\_mesh\_lpn\_poll (C++ function), 307  
 ESP\_BLE\_MESH\_LIGHTING\_SERVER\_STATE\_CHANGE\_EVT\_MESH\_LPN\_POLL\_COMP\_EVT (C++  
 (C++ enumerator), 425 enumerator), 302  
 esp\_ble\_mesh\_lighting\_server\_state\_change\_t::etm\_stemp\_faulting\_server\_state\_change\_t::lc  
 (C++ union), 386 宏), 382  
 esp\_ble\_mesh\_lighting\_server\_state\_change\_t::etm\_stemp\_faulting\_server\_state\_change\_t::lc  
 (C++ member), 387 (C 宏), 382  
 esp\_ble\_mesh\_lighting\_server\_state\_change\_t::etm\_stemp\_faulting\_server\_state\_change\_t::lc  
 (C++ member), 386 ESP\_BLE\_MESH\_MEMORY\_ERROR (C 宏), 381  
 (C++ member), 387 ESP\_BLE\_MESH\_MEMORY\_WARNING (C 宏), 381  
 esp\_ble\_mesh\_lighting\_server\_state\_change\_t::etm\_stemp\_faulting\_server\_state\_change\_t::lc  
 (C++ member), 387 ESP\_BLE\_MESH\_MIC\_SHORT (C 宏), 282  
 esp\_ble\_mesh\_lighting\_server\_state\_change\_t::etm\_stemp\_faulting\_server\_state\_change\_t::lc  
 (C++ member), 386 esp\_ble\_mesh\_model::cb (C++ class), 275  
 (C++ member), 387 esp\_ble\_mesh\_model::cb (C++ member), 276  
 esp\_ble\_mesh\_lighting\_server\_state\_change\_t::etm\_stemp\_faulting\_server\_state\_change\_t::lc  
 (C++ member), 387 宏), 276  
 esp\_ble\_mesh\_lighting\_server\_state\_change\_t::etm\_stemp\_faulting\_server\_state\_change\_t::lc  
 (C++ member), 387 esp\_ble\_mesh\_model::element\_idx (C++  
 (C++ member), 387 member), 276



- ESP\_BLE\_MESH\_MODEL\_GEN\_LOCATION\_CLI (C 宏), 370
- ESP\_BLE\_MESH\_MODEL\_GEN\_LOCATION\_SETUP\_SRV (C 宏), 372
- ESP\_BLE\_MESH\_MODEL\_GEN\_LOCATION\_SRV (C 宏), 372
- ESP\_BLE\_MESH\_MODEL\_GEN\_MANUFACTURER\_PROP\_SRV (C 宏), 373
- ESP\_BLE\_MESH\_MODEL\_GEN\_ONOFF\_CLI (C 宏), 370
- ESP\_BLE\_MESH\_MODEL\_GEN\_ONOFF\_SRV (C 宏), 371
- ESP\_BLE\_MESH\_MODEL\_GEN\_POWER\_LEVEL\_CLI (C 宏), 370
- ESP\_BLE\_MESH\_MODEL\_GEN\_POWER\_LEVEL\_SETUP\_SRV (C 宏), 372
- ESP\_BLE\_MESH\_MODEL\_GEN\_POWER\_LEVEL\_SRV (C 宏), 372
- ESP\_BLE\_MESH\_MODEL\_GEN\_POWER\_ONOFF\_CLI (C 宏), 370
- ESP\_BLE\_MESH\_MODEL\_GEN\_POWER\_ONOFF\_SETUP\_SRV (C 宏), 371
- ESP\_BLE\_MESH\_MODEL\_GEN\_POWER\_ONOFF\_SRV (C 宏), 371
- ESP\_BLE\_MESH\_MODEL\_GEN\_PROPERTY\_CLI (C 宏), 371
- ESP\_BLE\_MESH\_MODEL\_GEN\_USER\_PROP\_SRV (C 宏), 373
- ESP\_BLE\_MESH\_MODEL\_HEALTH\_CLI (C 宏), 381
- ESP\_BLE\_MESH\_MODEL\_HEALTH\_SRV (C 宏), 380
- ESP\_BLE\_MESH\_MODEL\_ID\_CONFIG\_CLI (C 宏), 285
- ESP\_BLE\_MESH\_MODEL\_ID\_CONFIG\_SRV (C 宏), 285
- ESP\_BLE\_MESH\_MODEL\_ID\_GEN\_ADMIN\_PROP\_SRV (C 宏), 286
- ESP\_BLE\_MESH\_MODEL\_ID\_GEN\_BATTERY\_CLI (C 宏), 286
- ESP\_BLE\_MESH\_MODEL\_ID\_GEN\_BATTERY\_SRV (C 宏), 286
- ESP\_BLE\_MESH\_MODEL\_ID\_GEN\_CLIENT\_PROP\_SRV (C 宏), 286
- ESP\_BLE\_MESH\_MODEL\_ID\_GEN\_DEF\_TRANS\_TIME\_SRV (C 宏), 285
- ESP\_BLE\_MESH\_MODEL\_ID\_GEN\_DEF\_TRANS\_TIME\_SRV (C 宏), 285
- ESP\_BLE\_MESH\_MODEL\_ID\_GEN\_LEVEL\_CLI (C 宏), 285
- ESP\_BLE\_MESH\_MODEL\_ID\_GEN\_LEVEL\_SRV (C 宏), 285
- ESP\_BLE\_MESH\_MODEL\_ID\_GEN\_LOCATION\_CLI (C 宏), 286
- ESP\_BLE\_MESH\_MODEL\_ID\_GEN\_LOCATION\_SETUP\_SRV (C 宏), 286
- ESP\_BLE\_MESH\_MODEL\_ID\_GEN\_LOCATION\_SRV (C 宏), 286
- ESP\_BLE\_MESH\_MODEL\_ID\_GEN\_MANUFACTURER\_PROP\_SRV (C 宏), 286
- ESP\_BLE\_MESH\_MODEL\_ID\_GEN\_ONOFF\_CLI (C 宏), 285
- ESP\_BLE\_MESH\_MODEL\_ID\_GEN\_ONOFF\_SRV (C 宏), 285
- ESP\_BLE\_MESH\_MODEL\_ID\_GEN\_POWER\_LEVEL\_CLI (C 宏), 286
- ESP\_BLE\_MESH\_MODEL\_ID\_GEN\_POWER\_LEVEL\_SETUP\_SRV (C 宏), 286
- ESP\_BLE\_MESH\_MODEL\_ID\_GEN\_POWER\_LEVEL\_SRV (C 宏), 286
- ESP\_BLE\_MESH\_MODEL\_ID\_GEN\_POWER\_ONOFF\_CLI (C 宏), 286
- ESP\_BLE\_MESH\_MODEL\_ID\_GEN\_POWER\_ONOFF\_SETUP\_SRV (C 宏), 285
- ESP\_BLE\_MESH\_MODEL\_ID\_GEN\_POWER\_ONOFF\_SRV (C 宏), 285
- ESP\_BLE\_MESH\_MODEL\_ID\_GEN\_PROP\_CLI (C 宏), 286
- ESP\_BLE\_MESH\_MODEL\_ID\_GEN\_USER\_PROP\_SRV (C 宏), 286
- ESP\_BLE\_MESH\_MODEL\_ID\_HEALTH\_CLI (C 宏), 285
- ESP\_BLE\_MESH\_MODEL\_ID\_HEALTH\_SRV (C 宏), 285
- ESP\_BLE\_MESH\_MODEL\_ID\_LIGHT\_CTL\_CLI (C 宏), 286
- ESP\_BLE\_MESH\_MODEL\_ID\_LIGHT\_CTL\_SETUP\_SRV (C 宏), 286
- ESP\_BLE\_MESH\_MODEL\_ID\_LIGHT\_CTL\_SRV (C 宏), 286
- ESP\_BLE\_MESH\_MODEL\_ID\_LIGHT\_CTL\_TEMP\_SRV (C 宏), 286
- ESP\_BLE\_MESH\_MODEL\_ID\_LIGHT\_HSL\_CLI (C 宏), 286
- ESP\_BLE\_MESH\_MODEL\_ID\_LIGHT\_HSL\_HUE\_SRV (C 宏), 286
- ESP\_BLE\_MESH\_MODEL\_ID\_LIGHT\_HSL\_SAT\_SRV (C 宏), 286
- ESP\_BLE\_MESH\_MODEL\_ID\_LIGHT\_HSL\_SETUP\_SRV (C 宏), 286
- ESP\_BLE\_MESH\_MODEL\_ID\_LIGHT\_HSL\_SRV (C 宏), 286
- ESP\_BLE\_MESH\_MODEL\_ID\_LIGHT\_LC\_CLI (C 宏), 287
- ESP\_BLE\_MESH\_MODEL\_ID\_LIGHT\_LC\_SETUP\_SRV (C 宏), 287
- ESP\_BLE\_MESH\_MODEL\_ID\_LIGHT\_LC\_SRV (C 宏), 287
- ESP\_BLE\_MESH\_MODEL\_ID\_LIGHT\_LIGHTNESS\_CLI (C 宏), 286
- ESP\_BLE\_MESH\_MODEL\_ID\_LIGHT\_LIGHTNESS\_SETUP\_SRV (C 宏), 286
- ESP\_BLE\_MESH\_MODEL\_ID\_LIGHT\_LIGHTNESS\_SRV (C 宏), 286
- ESP\_BLE\_MESH\_MODEL\_ID\_LIGHT\_XYL\_CLI (C 宏), 287
- ESP\_BLE\_MESH\_MODEL\_ID\_LIGHT\_XYL\_SETUP\_SRV (C 宏), 286



- ESP\_BLE\_MESH\_MODEL\_ID\_LIGHT\_XYL\_SRV (C 宏), 286
- ESP\_BLE\_MESH\_MODEL\_ID\_SCENE\_CLI (C 宏), 286
- ESP\_BLE\_MESH\_MODEL\_ID\_SCENE\_SETUP\_SRV (C 宏), 286
- ESP\_BLE\_MESH\_MODEL\_ID\_SCENE\_SRV (C 宏), 286
- ESP\_BLE\_MESH\_MODEL\_ID\_SCHEDULER\_CLI (C 宏), 286
- ESP\_BLE\_MESH\_MODEL\_ID\_SCHEDULER\_SETUP\_SRV (C 宏), 286
- ESP\_BLE\_MESH\_MODEL\_ID\_SCHEDULER\_SRV (C 宏), 286
- ESP\_BLE\_MESH\_MODEL\_ID\_SENSOR\_CLI (C 宏), 286
- ESP\_BLE\_MESH\_MODEL\_ID\_SENSOR\_SETUP\_SRV (C 宏), 286
- ESP\_BLE\_MESH\_MODEL\_ID\_SENSOR\_SRV (C 宏), 286
- ESP\_BLE\_MESH\_MODEL\_ID\_TIME\_CLI (C 宏), 286
- ESP\_BLE\_MESH\_MODEL\_ID\_TIME\_SETUP\_SRV (C 宏), 286
- ESP\_BLE\_MESH\_MODEL\_ID\_TIME\_SRV (C 宏), 286
- ESP\_BLE\_MESH\_MODEL\_LIGHT\_CTL\_CLI (C 宏), 421
- ESP\_BLE\_MESH\_MODEL\_LIGHT\_CTL\_SETUP\_SRV (C 宏), 422
- ESP\_BLE\_MESH\_MODEL\_LIGHT\_CTL\_SRV (C 宏), 421
- ESP\_BLE\_MESH\_MODEL\_LIGHT\_CTL\_TEMP\_SRV (C 宏), 422
- ESP\_BLE\_MESH\_MODEL\_LIGHT\_HSL\_CLI (C 宏), 421
- ESP\_BLE\_MESH\_MODEL\_LIGHT\_HSL\_HUE\_SRV (C 宏), 423
- ESP\_BLE\_MESH\_MODEL\_LIGHT\_HSL\_SAT\_SRV (C 宏), 423
- ESP\_BLE\_MESH\_MODEL\_LIGHT\_HSL\_SETUP\_SRV (C 宏), 422
- ESP\_BLE\_MESH\_MODEL\_LIGHT\_HSL\_SRV (C 宏), 422
- ESP\_BLE\_MESH\_MODEL\_LIGHT\_LC\_CLI (C 宏), 421
- ESP\_BLE\_MESH\_MODEL\_LIGHT\_LC\_SETUP\_SRV (C 宏), 423
- ESP\_BLE\_MESH\_MODEL\_LIGHT\_LC\_SRV (C 宏), 423
- ESP\_BLE\_MESH\_MODEL\_LIGHT\_LIGHTNESS\_CLI (C 宏), 420
- ESP\_BLE\_MESH\_MODEL\_LIGHT\_LIGHTNESS\_SETUP\_SRV (C 宏), 421
- ESP\_BLE\_MESH\_MODEL\_LIGHT\_LIGHTNESS\_SRV (C 宏), 421
- ESP\_BLE\_MESH\_MODEL\_LIGHT\_XYL\_CLI (C 宏), 421
- ESP\_BLE\_MESH\_MODEL\_LIGHT\_XYL\_SETUP\_SRV (C 宏), 423
- ESP\_BLE\_MESH\_MODEL\_LIGHT\_XYL\_SRV (C 宏), 423
- esp\_ble\_mesh\_model\_msg\_opcode\_init (C++ function), 308
- ESP\_BLE\_MESH\_MODEL\_NONE (C 宏), 285
- ESP\_BLE\_MESH\_MODEL\_OP (C 宏), 285
- ESP\_BLE\_MESH\_MODEL\_OP\_1 (C 宏), 284
- ESP\_BLE\_MESH\_MODEL\_OP\_2 (C 宏), 284
- ESP\_BLE\_MESH\_MODEL\_OP\_3 (C 宏), 284
- ESP\_BLE\_MESH\_MODEL\_OP\_APP\_KEY\_ADD (C 宏), 288
- ESP\_BLE\_MESH\_MODEL\_OP\_APP\_KEY\_DELETE (C 宏), 288
- ESP\_BLE\_MESH\_MODEL\_OP\_APP\_KEY\_GET (C 宏), 287
- ESP\_BLE\_MESH\_MODEL\_OP\_APP\_KEY\_LIST (C 宏), 289
- ESP\_BLE\_MESH\_MODEL\_OP\_APP\_KEY\_STATUS (C 宏), 289
- ESP\_BLE\_MESH\_MODEL\_OP\_APP\_KEY\_UPDATE (C 宏), 288
- ESP\_BLE\_MESH\_MODEL\_OP\_ATTENTION\_GET (C 宏), 290
- ESP\_BLE\_MESH\_MODEL\_OP\_ATTENTION\_SET (C 宏), 290
- ESP\_BLE\_MESH\_MODEL\_OP\_ATTENTION\_SET\_UNACK (C 宏), 290
- ESP\_BLE\_MESH\_MODEL\_OP\_ATTENTION\_STATUS (C 宏), 290
- ESP\_BLE\_MESH\_MODEL\_OP\_BEACON\_GET (C 宏), 287
- ESP\_BLE\_MESH\_MODEL\_OP\_BEACON\_SET (C 宏), 287
- ESP\_BLE\_MESH\_MODEL\_OP\_BEACON\_STATUS (C 宏), 289
- ESP\_BLE\_MESH\_MODEL\_OP\_COMPOSITION\_DATA\_GET (C 宏), 287
- ESP\_BLE\_MESH\_MODEL\_OP\_COMPOSITION\_DATA\_STATUS (C 宏), 289
- ESP\_BLE\_MESH\_MODEL\_OP\_DEFAULT\_TTL\_GET (C 宏), 287
- ESP\_BLE\_MESH\_MODEL\_OP\_DEFAULT\_TTL\_SET (C 宏), 287
- ESP\_BLE\_MESH\_MODEL\_OP\_DEFAULT\_TTL\_STATUS (C 宏), 289
- ESP\_BLE\_MESH\_MODEL\_OP\_END (C 宏), 285
- ESP\_BLE\_MESH\_MODEL\_OP\_FRIEND\_GET (C 宏), 287
- ESP\_BLE\_MESH\_MODEL\_OP\_FRIEND\_SET (C 宏), 288
- ESP\_BLE\_MESH\_MODEL\_OP\_FRIEND\_STATUS (C 宏), 289
- ESP\_BLE\_MESH\_MODEL\_OP\_GATT\_PROXY\_GET (C 宏), 287
- ESP\_BLE\_MESH\_MODEL\_OP\_GATT\_PROXY\_SET (C 宏), 288



ESP\_BLE\_MESH\_MODEL\_OP\_GEN\_POWER\_RANGE\_SET ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_CTL\_DEFAULT\_SET\_UNACK  
(C 宏), 291 (C 宏), 294

ESP\_BLE\_MESH\_MODEL\_OP\_GEN\_USER\_PROPERTY\_GET ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_CTL\_DEFAULT\_STATUS  
(C 宏), 292 (C 宏), 294

ESP\_BLE\_MESH\_MODEL\_OP\_GEN\_USER\_PROPERTY\_SET ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_CTL\_GET (C  
(C 宏), 292 宏), 294

ESP\_BLE\_MESH\_MODEL\_OP\_GEN\_USER\_PROPERTY\_ENABLE ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_CTL\_SET (C  
(C 宏), 292 宏), 294

ESP\_BLE\_MESH\_MODEL\_OP\_GEN\_USER\_PROPERTY\_DISABLE ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_CTL\_SET\_UNACK  
(C 宏), 292 (C 宏), 294

ESP\_BLE\_MESH\_MODEL\_OP\_GEN\_USER\_PROPERTY\_ENABLE\_BLINKS ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_CTL\_STATUS  
(C 宏), 292 (C 宏), 294

ESP\_BLE\_MESH\_MODEL\_OP\_GEN\_USER\_PROPERTY\_DISABLE\_BLINKS ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_CTL\_TEMPERATURE\_GET  
(C 宏), 292 (C 宏), 294

ESP\_BLE\_MESH\_MODEL\_OP\_HEALTH\_CURRENT\_STATUS ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_CTL\_TEMPERATURE\_RANGE  
(C 宏), 290 (C 宏), 294

ESP\_BLE\_MESH\_MODEL\_OP\_HEALTH\_FAULT\_CLEAR ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_CTL\_TEMPERATURE\_RANGE  
(C 宏), 290 (C 宏), 294

ESP\_BLE\_MESH\_MODEL\_OP\_HEALTH\_FAULT\_CLEAR\_UNACK ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_CTL\_TEMPERATURE\_RANGE  
(C 宏), 290 (C 宏), 294

ESP\_BLE\_MESH\_MODEL\_OP\_HEALTH\_FAULT\_GET ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_CTL\_TEMPERATURE\_RANGE  
(C 宏), 290 (C 宏), 294

ESP\_BLE\_MESH\_MODEL\_OP\_HEALTH\_FAULT\_STATUS ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_CTL\_TEMPERATURE\_SET  
(C 宏), 290 (C 宏), 294

ESP\_BLE\_MESH\_MODEL\_OP\_HEALTH\_FAULT\_TEST ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_CTL\_TEMPERATURE\_SET\_U  
(C 宏), 290 (C 宏), 294

ESP\_BLE\_MESH\_MODEL\_OP\_HEALTH\_FAULT\_TEST\_UNACK ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_CTL\_TEMPERATURE\_STATU  
(C 宏), 290 (C 宏), 294

ESP\_BLE\_MESH\_MODEL\_OP\_HEALTH\_PERIOD\_GET ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_HSL\_DEFAULT\_GET  
(C 宏), 290 (C 宏), 295

ESP\_BLE\_MESH\_MODEL\_OP\_HEALTH\_PERIOD\_SET ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_HSL\_DEFAULT\_SET  
(C 宏), 290 (C 宏), 295

ESP\_BLE\_MESH\_MODEL\_OP\_HEALTH\_PERIOD\_SET\_UNACK ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_HSL\_DEFAULT\_SET\_UNACK  
(C 宏), 290 (C 宏), 295

ESP\_BLE\_MESH\_MODEL\_OP\_HEALTH\_PERIOD\_STATUS ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_HSL\_DEFAULT\_STATUS  
(C 宏), 290 (C 宏), 295

ESP\_BLE\_MESH\_MODEL\_OP\_HEARTBEAT\_PUB\_GET ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_HSL\_GET (C  
(C 宏), 287 宏), 294

ESP\_BLE\_MESH\_MODEL\_OP\_HEARTBEAT\_PUB\_SET ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_HSL\_HUE\_GET  
(C 宏), 288 (C 宏), 294

ESP\_BLE\_MESH\_MODEL\_OP\_HEARTBEAT\_PUB\_STATUS ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_HSL\_HUE\_SET  
(C 宏), 289 (C 宏), 294

ESP\_BLE\_MESH\_MODEL\_OP\_HEARTBEAT\_SUB\_GET ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_HSL\_HUE\_SET\_UNACK  
(C 宏), 287 (C 宏), 294

ESP\_BLE\_MESH\_MODEL\_OP\_HEARTBEAT\_SUB\_SET ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_HSL\_HUE\_STATUS  
(C 宏), 288 (C 宏), 294

ESP\_BLE\_MESH\_MODEL\_OP\_HEARTBEAT\_SUB\_STATUS ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_HSL\_RANGE\_GET  
(C 宏), 289 (C 宏), 295

ESP\_BLE\_MESH\_MODEL\_OP\_KEY\_REFRESH\_PHASE\_SET ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_HSL\_RANGE\_SET  
(C 宏), 287 (C 宏), 295

ESP\_BLE\_MESH\_MODEL\_OP\_KEY\_REFRESH\_PHASE\_SET\_UNACK ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_HSL\_RANGE\_SET\_UNACK  
(C 宏), 288 (C 宏), 295

ESP\_BLE\_MESH\_MODEL\_OP\_KEY\_REFRESH\_PHASE\_STATUS ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_HSL\_RANGE\_STATUS  
(C 宏), 289 (C 宏), 295

ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_CTL\_DEFAULT\_SET ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_HSL\_SATURATION\_GET  
(C 宏), 294 (C 宏), 295

ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_CTL\_DEFAULT\_SET\_UNACK ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_HSL\_SATURATION\_SET  
(C 宏), 294 (C 宏), 295

ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_HSL\_SATURATION\_GET (C 宏), 295

ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_HSL\_SATURATION\_SET (C 宏), 294

ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_HSL\_STATUS\_GET (C 宏), 295

ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_HSL\_STATUS\_SET (C 宏), 294

ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_HSL\_TARGET\_GET (C 宏), 295

ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_HSL\_TARGET\_SET (C 宏), 294

ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_LC\_LIGHT\_ONOFF\_GET (C 宏), 296

ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_LC\_LIGHT\_ONOFF\_SET (C 宏), 294

ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_LC\_LIGHT\_ONOFF\_STATUS\_GET (C 宏), 296

ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_LC\_LIGHT\_ONOFF\_STATUS\_SET (C 宏), 294

ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_LC\_LIGHT\_ONOFF\_UNACK (C 宏), 293

ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_LC\_LIGHT\_ONOFF\_UNACK\_STATUS\_GET (C 宏), 296

ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_LC\_LIGHT\_ONOFF\_UNACK\_STATUS\_SET (C 宏), 293

ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_LC\_MODE\_GET (C 宏), 295

ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_LC\_MODE\_SET (C 宏), 294

ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_LC\_MODE\_STATUS\_GET (C 宏), 295

ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_LC\_MODE\_STATUS\_SET (C 宏), 294

ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_LC\_OM\_GET (C 宏), 295

ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_LC\_OM\_SET (C 宏), 296

ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_LC\_OM\_STATUS\_GET (C 宏), 295

ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_LC\_OM\_STATUS\_SET (C 宏), 296

ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_LC\_PROPERTY\_GET (C 宏), 296

ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_LC\_PROPERTY\_SET (C 宏), 295

ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_LC\_PROPERTY\_STATUS\_GET (C 宏), 296

ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_LC\_PROPERTY\_STATUS\_SET (C 宏), 295

ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_LC\_PROPERTY\_UNACK (C 宏), 295

ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_LC\_PROPERTY\_UNACK\_STATUS\_GET (C 宏), 296

ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_LC\_PROPERTY\_UNACK\_STATUS\_SET (C 宏), 295

ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_LIGHTNESS\_DEFAULT\_GET (C 宏), 294

ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_LIGHTNESS\_DEFAULT\_SET (C 宏), 295

ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_LIGHTNESS\_DEFAULT\_STATUS\_GET (C 宏), 294

ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_LIGHTNESS\_DEFAULT\_STATUS\_SET (C 宏), 295

ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_LIGHTNESS\_DEFAULT\_UNACK (C 宏), 294

ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_LIGHTNESS\_DEFAULT\_UNACK\_STATUS\_GET (C 宏), 294

ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_LIGHTNESS\_DEFAULT\_UNACK\_STATUS\_SET (C 宏), 295

ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_LIGHTNESS\_GET (C 宏), 293

ESP\_BLE\_MESH\_MODEL\_OP\_LIGHT\_LIGHTNESS\_SET (C 宏), 288

ESP\_BLE\_MESH\_MODEL\_OP\_MODEL\_APP\_STATUS ESP\_BLE\_MESH\_MODEL\_OP\_RELAY\_SET (C 宏),  
 (C 宏), 289 288  
 ESP\_BLE\_MESH\_MODEL\_OP\_MODEL\_APP\_UNBIND ESP\_BLE\_MESH\_MODEL\_OP\_RELAY\_STATUS (C  
 (C 宏), 289 宏), 289  
 ESP\_BLE\_MESH\_MODEL\_OP\_MODEL\_PUB\_GET (C ESP\_BLE\_MESH\_MODEL\_OP\_SCENE\_DELETE (C  
 宏), 287 宏), 293  
 ESP\_BLE\_MESH\_MODEL\_OP\_MODEL\_PUB\_SET (C ESP\_BLE\_MESH\_MODEL\_OP\_SCENE\_DELETE\_UNACK  
 宏), 288 (C 宏), 293  
 ESP\_BLE\_MESH\_MODEL\_OP\_MODEL\_PUB\_STATUS ESP\_BLE\_MESH\_MODEL\_OP\_SCENE\_GET (C 宏),  
 (C 宏), 289 293  
 ESP\_BLE\_MESH\_MODEL\_OP\_MODEL\_PUB\_VIRTUAL\_ADDRESS ESP\_BLE\_MESH\_MODEL\_OP\_SCENE\_RECALL (C  
 (C 宏), 288 宏), 293  
 ESP\_BLE\_MESH\_MODEL\_OP\_MODEL\_SUB\_ADD (C ESP\_BLE\_MESH\_MODEL\_OP\_SCENE\_RECALL\_UNACK  
 宏), 288 (C 宏), 293  
 ESP\_BLE\_MESH\_MODEL\_OP\_MODEL\_SUB\_DELETE ESP\_BLE\_MESH\_MODEL\_OP\_SCENE\_REGISTER\_GET  
 (C 宏), 288 (C 宏), 293  
 ESP\_BLE\_MESH\_MODEL\_OP\_MODEL\_SUB\_DELETE\_FORCE ESP\_BLE\_MESH\_MODEL\_OP\_SCENE\_REGISTER\_STATUS  
 (C 宏), 289 (C 宏), 293  
 ESP\_BLE\_MESH\_MODEL\_OP\_MODEL\_SUB\_OVERWRITE ESP\_BLE\_MESH\_MODEL\_OP\_SCENE\_STATUS (C  
 (C 宏), 288 宏), 293  
 ESP\_BLE\_MESH\_MODEL\_OP\_MODEL\_SUB\_STATUS ESP\_BLE\_MESH\_MODEL\_OP\_SCENE\_STORE (C  
 (C 宏), 289 宏), 293  
 ESP\_BLE\_MESH\_MODEL\_OP\_MODEL\_SUB\_VIRTUAL\_ADDRESS ESP\_BLE\_MESH\_MODEL\_OP\_SCENE\_STORE\_UNACK  
 (C 宏), 288 (C 宏), 293  
 ESP\_BLE\_MESH\_MODEL\_OP\_MODEL\_SUB\_VIRTUAL\_ADDRESS\_OVERRIDE ESP\_BLE\_MESH\_MODEL\_OP\_SCHEDULER\_ACT\_GET  
 (C 宏), 288 (C 宏), 293  
 ESP\_BLE\_MESH\_MODEL\_OP\_MODEL\_SUB\_VIRTUAL\_ADDRESS\_OVERRIDE\_WRITE ESP\_BLE\_MESH\_MODEL\_OP\_SCHEDULER\_ACT\_SET  
 (C 宏), 288 (C 宏), 293  
 ESP\_BLE\_MESH\_MODEL\_OP\_NET\_KEY\_ADD (C ESP\_BLE\_MESH\_MODEL\_OP\_SCHEDULER\_ACT\_SET\_UNACK  
 宏), 288 (C 宏), 293  
 ESP\_BLE\_MESH\_MODEL\_OP\_NET\_KEY\_DELETE ESP\_BLE\_MESH\_MODEL\_OP\_SCHEDULER\_ACT\_STATUS  
 (C 宏), 288 (C 宏), 293  
 ESP\_BLE\_MESH\_MODEL\_OP\_NET\_KEY\_GET (C ESP\_BLE\_MESH\_MODEL\_OP\_SCHEDULER\_GET (C  
 宏), 287 宏), 293  
 ESP\_BLE\_MESH\_MODEL\_OP\_NET\_KEY\_LIST (C ESP\_BLE\_MESH\_MODEL\_OP\_SCHEDULER\_STATUS  
 宏), 289 (C 宏), 293  
 ESP\_BLE\_MESH\_MODEL\_OP\_NET\_KEY\_STATUS ESP\_BLE\_MESH\_MODEL\_OP\_SENSOR\_CADENCE\_GET  
 (C 宏), 289 (C 宏), 292  
 ESP\_BLE\_MESH\_MODEL\_OP\_NET\_KEY\_UPDATE ESP\_BLE\_MESH\_MODEL\_OP\_SENSOR\_CADENCE\_SET  
 (C 宏), 288 (C 宏), 292  
 ESP\_BLE\_MESH\_MODEL\_OP\_NETWORK\_TRANSMIT\_ERROR ESP\_BLE\_MESH\_MODEL\_OP\_SENSOR\_CADENCE\_SET\_UNACK  
 (C 宏), 287 (C 宏), 292  
 ESP\_BLE\_MESH\_MODEL\_OP\_NETWORK\_TRANSMIT\_ERROR\_CLEAR ESP\_BLE\_MESH\_MODEL\_OP\_SENSOR\_CADENCE\_STATUS  
 (C 宏), 289 (C 宏), 292  
 ESP\_BLE\_MESH\_MODEL\_OP\_NETWORK\_TRANSMIT\_ERROR\_CLEAR\_WRITE ESP\_BLE\_MESH\_MODEL\_OP\_SENSOR\_COLUMN\_GET  
 (C 宏), 289 (C 宏), 292  
 ESP\_BLE\_MESH\_MODEL\_OP\_NODE\_IDENTITY\_GET ESP\_BLE\_MESH\_MODEL\_OP\_SENSOR\_COLUMN\_STATUS  
 (C 宏), 287 (C 宏), 292  
 ESP\_BLE\_MESH\_MODEL\_OP\_NODE\_IDENTITY\_SET ESP\_BLE\_MESH\_MODEL\_OP\_SENSOR\_DESCRIPTOR\_GET  
 (C 宏), 288 (C 宏), 292  
 ESP\_BLE\_MESH\_MODEL\_OP\_NODE\_IDENTITY\_STATUS ESP\_BLE\_MESH\_MODEL\_OP\_SENSOR\_DESCRIPTOR\_STATUS  
 (C 宏), 289 (C 宏), 292  
 ESP\_BLE\_MESH\_MODEL\_OP\_NODE\_RESET (C 宏), ESP\_BLE\_MESH\_MODEL\_OP\_SENSOR\_GET (C 宏),  
 288 292  
 ESP\_BLE\_MESH\_MODEL\_OP\_NODE\_RESET\_STATUS ESP\_BLE\_MESH\_MODEL\_OP\_SENSOR\_SERIES\_GET  
 (C 宏), 289 (C 宏), 292  
 ESP\_BLE\_MESH\_MODEL\_OP\_RELAY\_GET (C 宏), ESP\_BLE\_MESH\_MODEL\_OP\_SENSOR\_SERIES\_STATUS  
 287 (C 宏), 292

- ESP\_BLE\_MESH\_MODEL\_OP\_SENSOR\_SETTING\_GET (C 宏), 287  
(C 宏), 293
- ESP\_BLE\_MESH\_MODEL\_OP\_SENSOR\_SETTING\_SET (C 宏), 289  
(C 宏), 293
- ESP\_BLE\_MESH\_MODEL\_OP\_SENSOR\_SETTING\_SET\_UNACKNOWLEDGED (C 宏), 293
- ESP\_BLE\_MESH\_MODEL\_OP\_SENSOR\_SETTING\_STATUS (C 宏), 293
- ESP\_BLE\_MESH\_MODEL\_OP\_SENSOR\_SETTINGS\_GET (C 宏), 293  
(C 宏), 293
- ESP\_BLE\_MESH\_MODEL\_OP\_SENSOR\_SETTINGS\_STATUS (C 宏), 293  
(C 宏), 293
- ESP\_BLE\_MESH\_MODEL\_OP\_SENSOR\_STATUS (C 宏), 292
- ESP\_BLE\_MESH\_MODEL\_OP\_SIG\_MODEL\_APP\_GET (C 宏), 287  
(C 宏), 287
- ESP\_BLE\_MESH\_MODEL\_OP\_SIG\_MODEL\_APP\_LIST (C 宏), 289  
(C 宏), 289
- ESP\_BLE\_MESH\_MODEL\_OP\_SIG\_MODEL\_SUB\_GET (C 宏), 287  
(C 宏), 287
- ESP\_BLE\_MESH\_MODEL\_OP\_SIG\_MODEL\_SUB\_LIST (C 宏), 289  
(C 宏), 289
- esp\_ble\_mesh\_model\_op\_t (C++ class), 275
- esp\_ble\_mesh\_model\_op\_t::min\_len (C++ member), 275
- esp\_ble\_mesh\_model\_op\_t::opcode (C++ member), 275
- esp\_ble\_mesh\_model\_op\_t::param\_cb (C++ member), 275
- ESP\_BLE\_MESH\_MODEL\_OP\_TAI\_UTC\_DELTA\_GET (C 宏), 293  
(C 宏), 293
- ESP\_BLE\_MESH\_MODEL\_OP\_TAI\_UTC\_DELTA\_SET (C 宏), 293  
(C 宏), 293
- ESP\_BLE\_MESH\_MODEL\_OP\_TAI\_UTC\_DELTA\_STATUS (C 宏), 293  
(C 宏), 293
- ESP\_BLE\_MESH\_MODEL\_OP\_TIME\_GET (C 宏), 293  
(C 宏), 293
- ESP\_BLE\_MESH\_MODEL\_OP\_TIME\_ROLE\_GET (C 宏), 293  
(C 宏), 293
- ESP\_BLE\_MESH\_MODEL\_OP\_TIME\_ROLE\_SET (C 宏), 293  
(C 宏), 293
- ESP\_BLE\_MESH\_MODEL\_OP\_TIME\_ROLE\_STATUS (C 宏), 293  
(C 宏), 293
- ESP\_BLE\_MESH\_MODEL\_OP\_TIME\_SET (C 宏), 293  
(C 宏), 293
- ESP\_BLE\_MESH\_MODEL\_OP\_TIME\_STATUS (C 宏), 293  
(C 宏), 293
- ESP\_BLE\_MESH\_MODEL\_OP\_TIME\_ZONE\_GET (C 宏), 293  
(C 宏), 293
- ESP\_BLE\_MESH\_MODEL\_OP\_TIME\_ZONE\_SET (C 宏), 293  
(C 宏), 293
- ESP\_BLE\_MESH\_MODEL\_OP\_TIME\_ZONE\_STATUS (C 宏), 293  
(C 宏), 293
- ESP\_BLE\_MESH\_MODEL\_OP\_VENDOR\_MODEL\_APP\_GET (C 宏), 287  
(C 宏), 287
- ESP\_BLE\_MESH\_MODEL\_OP\_VENDOR\_MODEL\_APP\_LIST (C 宏), 289  
(C 宏), 289
- ESP\_BLE\_MESH\_MODEL\_OP\_VENDOR\_MODEL\_SUB\_GET (C 宏), 287  
(C 宏), 287
- ESP\_BLE\_MESH\_MODEL\_OP\_VENDOR\_MODEL\_SUB\_LIST (C 宏), 289  
(C 宏), 289
- ESP\_BLE\_MESH\_MODEL\_OP\_VENDOR\_MODEL\_STATUS (C 宏), 292  
(C 宏), 292
- ESP\_BLE\_MESH\_MODEL\_OPERATION\_EVT (C++ enumerator), 303
- ESP\_BLE\_MESH\_MODEL\_PUB\_DEFINE (C 宏), 284
- esp\_ble\_mesh\_model\_pub\_t (C++ class), 274
- esp\_ble\_mesh\_model\_pub\_t::app\_idx (C++ member), 274
- esp\_ble\_mesh\_model\_pub\_t::count (C++ member), 275
- esp\_ble\_mesh\_model\_pub\_t::cred (C++ member), 274
- esp\_ble\_mesh\_model\_pub\_t::dev\_role (C++ member), 275
- esp\_ble\_mesh\_model\_pub\_t::fast\_period (C++ member), 275
- esp\_ble\_mesh\_model\_pub\_t::model (C++ member), 274
- esp\_ble\_mesh\_model\_pub\_t::msg (C++ member), 275
- esp\_ble\_mesh\_model\_pub\_t::period (C++ member), 275
- esp\_ble\_mesh\_model\_pub\_t::period\_div (C++ member), 275
- esp\_ble\_mesh\_model\_pub\_t::period\_start (C++ member), 275
- esp\_ble\_mesh\_model\_pub\_t::publish\_addr (C++ member), 274
- esp\_ble\_mesh\_model\_pub\_t::retransmit (C++ member), 275
- esp\_ble\_mesh\_model\_pub\_t::send\_rel (C++ member), 274
- esp\_ble\_mesh\_model\_pub\_t::timer (C++ member), 275
- esp\_ble\_mesh\_model\_pub\_t::ttl (C++ member), 275
- esp\_ble\_mesh\_model\_pub\_t::update (C++ member), 275
- esp\_ble\_mesh\_model\_publish (C++ function), 309
- ESP\_BLE\_MESH\_MODEL\_PUBLISH\_COMP\_EVT (C++ enumerator), 303
- ESP\_BLE\_MESH\_MODEL\_PUBLISH\_UPDATE\_EVT (C++ enumerator), 303
- ESP\_BLE\_MESH\_MODEL\_SCENE\_CLI (C 宏), 458
- ESP\_BLE\_MESH\_MODEL\_SCENE\_SETUP\_SRV (C 宏), 459
- ESP\_BLE\_MESH\_MODEL\_SCENE\_SRV (C 宏), 458
- ESP\_BLE\_MESH\_MODEL\_SCHEDULER\_CLI (C 宏), 458
- ESP\_BLE\_MESH\_MODEL\_SCHEDULER\_SETUP\_SRV (C 宏), 459
- ESP\_BLE\_MESH\_MODEL\_SCHEDULER\_SRV (C 宏), 459
- ESP\_BLE\_MESH\_MODEL\_SEND\_COMP\_EVT (C++ enumerator), 303
- ESP\_BLE\_MESH\_MODEL\_SENSOR\_CLI (C 宏), 437

- ESP\_BLE\_MESH\_MODEL\_SENSOR\_SETUP\_SRV (C 宏), 437
- ESP\_BLE\_MESH\_MODEL\_SENSOR\_SRV (C 宏), 437
- ESP\_BLE\_MESH\_MODEL\_STATUS\_CANNOT\_SET\_RANGE\_MAX (C 宏), 296
- ESP\_BLE\_MESH\_MODEL\_STATUS\_CANNOT\_SET\_RANGE\_MIN (C 宏), 296
- ESP\_BLE\_MESH\_MODEL\_STATUS\_SUCCESS (C 宏), 296
- esp\_ble\_mesh\_model\_status\_t (C++ type), 297
- esp\_ble\_mesh\_model\_subscribe\_group\_addr (C++ function), 305
- ESP\_BLE\_MESH\_MODEL\_SUBSCRIBE\_GROUP\_ADDR\_COMPLETE (C++ enumerator), 302
- esp\_ble\_mesh\_model\_t (C++ type), 296
- ESP\_BLE\_MESH\_MODEL\_TIME\_CLI (C 宏), 458
- ESP\_BLE\_MESH\_MODEL\_TIME\_SETUP\_SRV (C 宏), 458
- ESP\_BLE\_MESH\_MODEL\_TIME\_SRV (C 宏), 458
- esp\_ble\_mesh\_model\_unsubscribe\_group\_addr (C++ function), 306
- ESP\_BLE\_MESH\_MODEL\_UNSUBSCRIBE\_GROUP\_ADDR\_COMPLETE (C++ enumerator), 302
- esp\_ble\_mesh\_msg\_ctx\_t (C++ class), 276
- esp\_ble\_mesh\_msg\_ctx\_t::addr (C++ member), 276
- esp\_ble\_mesh\_msg\_ctx\_t::app\_idx (C++ member), 276
- esp\_ble\_mesh\_msg\_ctx\_t::model (C++ member), 277
- esp\_ble\_mesh\_msg\_ctx\_t::net\_idx (C++ member), 276
- esp\_ble\_mesh\_msg\_ctx\_t::rcv\_dst (C++ member), 276
- esp\_ble\_mesh\_msg\_ctx\_t::rcv\_op (C++ member), 277
- esp\_ble\_mesh\_msg\_ctx\_t::rcv\_rssi (C++ member), 276
- esp\_ble\_mesh\_msg\_ctx\_t::rcv\_ttl (C++ member), 276
- esp\_ble\_mesh\_msg\_ctx\_t::send\_rel (C++ member), 276
- esp\_ble\_mesh\_msg\_ctx\_t::send\_ttl (C++ member), 276
- esp\_ble\_mesh\_msg\_ctx\_t::srv\_send (C++ member), 277
- ESP\_BLE\_MESH\_NET\_PRIMARY (C 宏), 282
- ESP\_BLE\_MESH\_NO\_FAULT (C 宏), 381
- ESP\_BLE\_MESH\_NO\_INPUT (C++ enumerator), 298
- ESP\_BLE\_MESH\_NO\_LOAD\_ERROR (C 宏), 381
- ESP\_BLE\_MESH\_NO\_LOAD\_WARNING (C 宏), 381
- ESP\_BLE\_MESH\_NO\_OOB (C++ enumerator), 298
- ESP\_BLE\_MESH\_NO\_OUTPUT (C++ enumerator), 298
- esp\_ble\_mesh\_node\_add\_local\_app\_key (C++ function), 306
- ESP\_BLE\_MESH\_NODE\_ADD\_LOCAL\_APP\_KEY\_COMPLETE\_EVT (C++ enumerator), 300
- esp\_ble\_mesh\_node\_add\_local\_net\_key (C++ function), 306
- ESP\_BLE\_MESH\_NODE\_ADD\_LOCAL\_NET\_KEY\_COMP\_EVT (C++ enumerator), 300
- esp\_ble\_mesh\_node\_bind\_app\_key\_to\_local\_model (C++ function), 306
- ESP\_BLE\_MESH\_NODE\_BIND\_APP\_KEY\_TO\_MODEL\_COMP\_EVT (C++ enumerator), 300
- esp\_ble\_mesh\_node\_get\_local\_app\_key (C++ function), 306
- esp\_ble\_mesh\_node\_get\_local\_net\_key (C++ function), 306
- ESP\_BLE\_MESH\_NODE\_IDENTITY\_NOT\_SUPPORTED (C 宏), 283
- ESP\_BLE\_MESH\_NODE\_IDENTITY\_RUNNING (C 宏), 283
- ESP\_BLE\_MESH\_NODE\_IDENTITY\_STOPPED (C 宏), 283
- esp\_ble\_mesh\_node\_input\_number (C++ function), 315
- esp\_ble\_mesh\_node\_input\_string (C++ function), 315
- esp\_ble\_mesh\_node\_is\_provisioned (C++ function), 315
- esp\_ble\_mesh\_node\_local\_reset (C++ function), 309
- ESP\_BLE\_MESH\_NODE\_NAME\_MAX\_LEN (C 宏), 282
- ESP\_BLE\_MESH\_NODE\_PROV\_COMPLETE\_EVT (C++ enumerator), 299
- esp\_ble\_mesh\_node\_prov\_disable (C++ function), 315
- ESP\_BLE\_MESH\_NODE\_PROV\_DISABLE\_COMP\_EVT (C++ enumerator), 299
- esp\_ble\_mesh\_node\_prov\_enable (C++ function), 315
- ESP\_BLE\_MESH\_NODE\_PROV\_ENABLE\_COMP\_EVT (C++ enumerator), 299
- ESP\_BLE\_MESH\_NODE\_PROV\_INPUT\_EVT (C++ enumerator), 299
- ESP\_BLE\_MESH\_NODE\_PROV\_INPUT\_NUMBER\_COMP\_EVT (C++ enumerator), 300
- ESP\_BLE\_MESH\_NODE\_PROV\_INPUT\_STRING\_COMP\_EVT (C++ enumerator), 300
- ESP\_BLE\_MESH\_NODE\_PROV\_LINK\_CLOSE\_EVT (C++ enumerator), 299
- ESP\_BLE\_MESH\_NODE\_PROV\_LINK\_OPEN\_EVT (C++ enumerator), 299
- ESP\_BLE\_MESH\_NODE\_PROV\_OOB\_PUB\_KEY\_EVT (C++ enumerator), 299
- ESP\_BLE\_MESH\_NODE\_PROV\_OUTPUT\_NUMBER\_EVT (C++ enumerator), 299
- ESP\_BLE\_MESH\_NODE\_PROV\_OUTPUT\_STRING\_EVT (C++ enumerator), 299
- ESP\_BLE\_MESH\_NODE\_PROV\_RESET\_EVT (C++ enumerator), 299
- ESP\_BLE\_MESH\_NODE\_PROV\_SET\_OOB\_PUB\_KEY\_COMP\_EVT (C++ enumerator), 299

- (C++ *enumerator*), 300
- ESP\_BLE\_MESH\_NODE\_PROXY\_GATT\_DISABLE\_COMPLETE\_EVT (C++ *enumerator*), 300
- ESP\_BLE\_MESH\_NODE\_PROXY\_GATT\_ENABLE\_COMPLETE\_EVT (C++ *enumerator*), 300
- ESP\_BLE\_MESH\_NODE\_PROXY\_IDENTITY\_ENABLE\_COMPLETE\_EVT (C++ *enumerator*), 300
- esp\_ble\_mesh\_node\_set\_oob\_pub\_key (C++ *function*), 315
- ESP\_BLE\_MESH\_NODE\_SET\_UNPROV\_DEV\_NAME\_COMPLETE\_EVT (C++ *enumerator*), 299
- esp\_ble\_mesh\_node\_t (C++ *class*), 278
- esp\_ble\_mesh\_node\_t::addr (C++ *member*), 278
- esp\_ble\_mesh\_node\_t::addr\_type (C++ *member*), 278
- esp\_ble\_mesh\_node\_t::comp\_data (C++ *member*), 279
- esp\_ble\_mesh\_node\_t::comp\_length (C++ *member*), 278
- esp\_ble\_mesh\_node\_t::dev\_key (C++ *member*), 278
- esp\_ble\_mesh\_node\_t::dev\_uuid (C++ *member*), 278
- esp\_ble\_mesh\_node\_t::element\_num (C++ *member*), 278
- esp\_ble\_mesh\_node\_t::flags (C++ *member*), 278
- esp\_ble\_mesh\_node\_t::iv\_index (C++ *member*), 278
- esp\_ble\_mesh\_node\_t::name (C++ *member*), 278
- esp\_ble\_mesh\_node\_t::net\_idx (C++ *member*), 278
- esp\_ble\_mesh\_node\_t::oob\_info (C++ *member*), 278
- esp\_ble\_mesh\_node\_t::unicast\_addr (C++ *member*), 278
- ESP\_BLE\_MESH\_OCTET16\_LEN (C 宏), 282
- esp\_ble\_mesh\_octet16\_t (C++ *type*), 296
- ESP\_BLE\_MESH\_OCTET8\_LEN (C 宏), 282
- esp\_ble\_mesh\_octet8\_t (C++ *type*), 296
- esp\_ble\_mesh\_oob\_method\_t (C++ *enum*), 298
- esp\_ble\_mesh\_opcode\_config\_client\_get\_tesp\_ble\_mesh\_prov\_cb\_param\_t (C++ *type*), 296
- esp\_ble\_mesh\_opcode\_config\_client\_set\_tesp\_ble\_mesh\_prov\_cb\_param\_t (C++ *type*), 296
- esp\_ble\_mesh\_opcode\_config\_status\_t (C++ *type*), 297
- esp\_ble\_mesh\_opcode\_health\_client\_get\_tesp\_ble\_mesh\_prov\_cb\_param\_t (C++ *type*), 297
- esp\_ble\_mesh\_opcode\_health\_client\_set\_tesp\_ble\_mesh\_prov\_cb\_param\_t (C++ *type*), 297
- esp\_ble\_mesh\_opcode\_t (C++ *type*), 297
- esp\_ble\_mesh\_output\_action\_t (C++ *enum*), 298
- ESP\_BLE\_MESH\_OUTPUT\_OOB (C++ *enumerator*), 298
- ESP\_BLE\_MESH\_OVERFLOW\_ERROR (C 宏), 382
- ESP\_BLE\_MESH\_OVERFLOW\_WARNING (C 宏), 382
- ESP\_BLE\_MESH\_OVERHEAT\_ERROR (C 宏), 381
- ESP\_BLE\_MESH\_OVERHEAT\_WARNING (C 宏), 381
- ESP\_BLE\_MESH\_OVERLOAD\_ERROR (C 宏), 381
- ESP\_BLE\_MESH\_OVERLOAD\_WARNING (C 宏), 381
- ESP\_BLE\_MESH\_POWER\_SUPPLY\_INTERRUPTED\_ERROR (C 宏), 381
- ESP\_BLE\_MESH\_POWER\_SUPPLY\_INTERRUPTED\_WARNING (C 宏), 381
- ESP\_BLE\_MESH\_PROV (C 宏), 284
- ESP\_BLE\_MESH\_PROV\_ADV (C++ *enumerator*), 298
- esp\_ble\_mesh\_prov\_adv\_cb\_t (C++ *type*), 319
- esp\_ble\_mesh\_prov\_bearer\_t (C++ *enum*), 298
- esp\_ble\_mesh\_prov\_cb\_event\_t (C++ *enum*), 299
- esp\_ble\_mesh\_prov\_cb\_param\_t (C++ *union*), 251
- esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_deinit\_mes (C++ *class*), 256
- esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_deinit\_mes (C++ *member*), 256
- esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_friend\_fri (C++ *class*), 256
- esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_friend\_fri (C++ *member*), 256
- esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_friend\_fri (C++ *class*), 256
- esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_friend\_fri (C++ *member*), 257
- esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_friend\_fri (C++ *member*), 257
- esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_friend\_fri (C++ *member*), 257
- esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_friend\_fri (C++ *member*), 257
- esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_heartbeat (C++ *class*), 257
- esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_heartbeat (C++ *member*), 257
- esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_heartbeat (C++ *member*), 257
- esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_input\_evt (C++ *class*), 257
- esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_input\_evt (C++ *member*), 257
- esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_input\_evt (C++ *member*), 257
- esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_input\_num (C++ *class*), 257
- esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_input\_num (C++ *member*), 257
- esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_input\_num (C++ *member*), 257
- esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_input\_stri (C++ *class*), 257
- esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_input\_stri (C++ *member*), 257
- esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_input\_stri (C++ *member*), 257
- esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_link\_close (C++ *class*), 257
- esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_link\_close (C++ *member*), 258
- esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_link\_open









(C++ class), 268  
 esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_espoxylechiesit\_mevow\_dfi\_ltaerrasttat:de\_iraaitamneshoncomp  
 (C++ member), 268 (C++ member), 256  
 esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_espoxylechiesit\_mevow\_dfi\_ltaerrasttat:enablam::filter\_typ  
 (C++ member), 268 (C++ member), 254  
 esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_espoxylechiesit\_mevow\_dfi\_ltaerrasttat:esr\_pacode::list\_size  
 (C++ member), 269 (C++ member), 253  
 esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_espoxylechiesit\_mevow\_dfi\_ltaerrasttat:ESPoBiam\_MESH\_FRND\_F  
 (C++ member), 268 (C++ enumerator), 257  
 esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_espoxylechiesit\_mevow\_dfi\_ltaerrasttat:ESPoBiam\_MESH\_FRNDaC  
 (C++ member), 268 (C++ enumerator), 256  
 esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_espoxylechiesit\_mevow\_dfi\_ltaerrasttat:ESPoBiam\_MESH\_FRND\_F  
 (C++ class), 269 (C++ enumerator), 256  
 esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_espoxylechiesit\_mevow\_dfi\_ltaerrasttat:ESPoBiam\_MESH\_FRNDaC  
 (C++ member), 269 (C++ enumerator), 257  
 esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_espoxylechiesit\_mevow\_dfi\_ltaerrasttat:ESPoBiam\_MESH\_FRNDaC  
 (C++ member), 269 (C++ enumerator), 256  
 esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_espoxylechiesit\_mevow\_dfi\_ltaerrasttat:ESPoBiam\_MESH\_FRNDaC  
 (C++ member), 269 (C++ member), 254  
 esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_espoxylechiesit\_mvstvfoidltaerrasttat:rfomplpfaria  
 (C++ class), 269 (C++ member), 255  
 esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_espoxylechiesit\_mvstvfoidltaerrasttat:rfomplpfaria:dstoipn\_thear  
 (C++ member), 269 (C++ member), 255  
 esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_espoxylechiesit\_mvstvfoidltaerrasttat:rdmgsparam::err\_code  
 (C++ member), 269 (C++ member), 254  
 esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_espoxylechiesit\_mvstvfoidltaerrasttat:rdmgsparam::net\_idx  
 (C++ member), 269 (C++ member), 254  
 esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_espoxylegatesdisabiechicpqpaparam:heartbeat\_msg\_recv  
 (C++ class), 269 (C++ member), 255  
 esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_espoxylegatesdisabiechicpqpaparam:hops\_code  
 (C++ member), 269 (C++ member), 254  
 esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_espoxylegatesrenable\_compaparam::index  
 (C++ class), 269 (C++ member), 254  
 esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_espoxylegatesrenable\_compaparam::errit\_cdtile  
 (C++ member), 269 (C++ member), 254  
 esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_espoxyleidmeshitporenablecompaparam::disable\_comp  
 (C++ class), 269 (C++ member), 255  
 esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_espoxyleidmeshitporenablecompaparam::enablecomp  
 (C++ member), 269 (C++ member), 255  
 esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_esp\_ble\_mesh\_prov\_act\_idn\_param:paralpn\_friendship\_esta  
 (C++ class), 269 (C++ member), 255  
 esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_esp\_ble\_mesh\_prov\_act\_idn\_param:paralpn\_sfraters\_sitp\_dre  
 (C++ member), 270 (C++ member), 255  
 esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_esp\_ble\_mesh\_prov\_act\_idn\_param:lpn\_poll\_comp  
 (C++ class), 270 (C++ member), 255  
 esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_esp\_ble\_mesh\_prov\_act\_idn\_param:mstaalt\_uslmajrup\_add  
 (C++ member), 270 (C++ member), 256  
 esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_esp\_ble\_mesh\_prov\_act\_idn\_param:mstaalt\_us\_sut\_gdrup\_a  
 (C++ member), 270 (C++ member), 256  
 esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_esp\_ble\_mesh\_prov\_act\_idn\_param:mstaalt\_us\_unipkesty\_co  
 (C++ member), 270 (C++ member), 252  
 esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_esp\_ble\_mesh\_keyvcomp\_param\_t::node\_add\_net\_key\_co  
 (C++ class), 270 (C++ member), 252  
 esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_esp\_ble\_mesh\_keyvcomp\_param\_t:emnodeidnd\_app\_key\_t  
 (C++ member), 270 (C++ member), 252  
 esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_esp\_ble\_mesh\_keyvcomp\_param\_t:emnodeidnd\_app\_key\_t  
 (C++ class), 270 (C++ member), 252  
 esp\_ble\_mesh\_prov\_cb\_param\_t::ble\_mesh\_esp\_ble\_mesh\_keyvcomp\_param\_t:emnodeidnd\_app\_key\_t  
 (C++ member), 270 (C++ member), 252

---

<code>(C++ member)</code> , 251	<code>(C++ member)</code> , 254
<code>esp_ble_mesh_prov_cb_param_t::node_provisioner_open_send</code> <code>(C++ member)</code> , 251	<code>esp_ble_mesh_prov_cb_param_t::provisioner_open_send</code> <code>(C++ member)</code> , 255
<code>esp_ble_mesh_prov_cb_param_t::node_provisioner_open_send</code> <code>(C++ member)</code> , 252	<code>esp_ble_mesh_prov_cb_param_t::provisioner_open_send</code> <code>(C++ member)</code> , 255
<code>esp_ble_mesh_prov_cb_param_t::node_provisioner_prov_de</code> <code>(C++ member)</code> , 252	<code>esp_ble_mesh_prov_cb_param_t::provisioner_prov_co</code> <code>(C++ member)</code> , 253
<code>esp_ble_mesh_prov_cb_param_t::node_provisioner_prov_de</code> <code>(C++ member)</code> , 252	<code>esp_ble_mesh_prov_cb_param_t::provisioner_prov_de</code> <code>(C++ member)</code> , 253
<code>esp_ble_mesh_prov_cb_param_t::node_provisioner_prov_di</code> <code>(C++ member)</code> , 252	<code>esp_ble_mesh_prov_cb_param_t::provisioner_prov_di</code> <code>(C++ member)</code> , 252
<code>esp_ble_mesh_prov_cb_param_t::node_provisioner_prov_en</code> <code>(C++ member)</code> , 252	<code>esp_ble_mesh_prov_cb_param_t::provisioner_prov_en</code> <code>(C++ member)</code> , 252
<code>esp_ble_mesh_prov_cb_param_t::node_provisioner_prov_in</code> <code>(C++ member)</code> , 252	<code>esp_ble_mesh_prov_cb_param_t::provisioner_prov_in</code> <code>(C++ member)</code> , 252
<code>esp_ble_mesh_prov_cb_param_t::node_provisioner_prov_in</code> <code>(C++ member)</code> , 252	<code>esp_ble_mesh_prov_cb_param_t::provisioner_prov_in</code> <code>(C++ member)</code> , 253
<code>esp_ble_mesh_prov_cb_param_t::node_provisioner_prov_in</code> <code>(C++ member)</code> , 252	<code>esp_ble_mesh_prov_cb_param_t::provisioner_prov_in</code> <code>(C++ member)</code> , 253
<code>esp_ble_mesh_prov_cb_param_t::node_provisioner_prov_li</code> <code>(C++ member)</code> , 252	<code>esp_ble_mesh_prov_cb_param_t::provisioner_prov_li</code> <code>(C++ member)</code> , 253
<code>esp_ble_mesh_prov_cb_param_t::node_provisioner_prov_li</code> <code>(C++ member)</code> , 252	<code>esp_ble_mesh_prov_cb_param_t::provisioner_prov_li</code> <code>(C++ member)</code> , 252
<code>esp_ble_mesh_prov_cb_param_t::node_provisioner_prov_ou</code> <code>(C++ member)</code> , 252	<code>esp_ble_mesh_prov_cb_param_t::provisioner_prov_ou</code> <code>(C++ member)</code> , 252
<code>esp_ble_mesh_prov_cb_param_t::node_provisioner_prov_re</code> <code>(C++ member)</code> , 252	<code>esp_ble_mesh_prov_cb_param_t::provisioner_prov_re</code> <code>(C++ member)</code> , 252
<code>esp_ble_mesh_prov_cb_param_t::node_set_device_name</code> <code>(C++ member)</code> , 251	<code>esp_ble_mesh_prov_cb_param_t::provisioner_prov_re</code> <code>(C++ member)</code> , 253
<code>esp_ble_mesh_prov_cb_param_t::op</code> ( <code>C++ member</code> ), 254	<code>esp_ble_mesh_prov_cb_param_t::provisioner_recv_he</code> <code>(C++ member)</code> , 254
<code>esp_ble_mesh_prov_cb_param_t::prov_register</code> <code>(C++ member)</code> , 251	<code>esp_ble_mesh_prov_cb_param_t::provisioner_recv_un</code> <code>(C++ member)</code> , 252
<code>esp_ble_mesh_prov_cb_param_t::provisioner_set_dev</code> <code>(C++ member)</code> , 253	<code>esp_ble_mesh_prov_cb_param_t::provisioner_set_dev</code> <code>(C++ member)</code> , 253
<code>esp_ble_mesh_prov_cb_param_t::provisioner_set_he</code> <code>(C++ member)</code> , 253	<code>esp_ble_mesh_prov_cb_param_t::provisioner_set_he</code> <code>(C++ member)</code> , 254
<code>esp_ble_mesh_prov_cb_param_t::provisioner_set_he</code> <code>(C++ member)</code> , 253	<code>esp_ble_mesh_prov_cb_param_t::provisioner_set_he</code> <code>(C++ member)</code> , 254
<code>esp_ble_mesh_prov_cb_param_t::provisioner_set_nod</code> <code>(C++ member)</code> , 253	<code>esp_ble_mesh_prov_cb_param_t::provisioner_set_nod</code> <code>(C++ member)</code> , 253
<code>esp_ble_mesh_prov_cb_param_t::provisioner_set_pri</code> <code>(C++ member)</code> , 255	<code>esp_ble_mesh_prov_cb_param_t::provisioner_set_pri</code> <code>(C++ member)</code> , 253
<code>esp_ble_mesh_prov_cb_param_t::provisioner_set_pro</code> <code>(C++ member)</code> , 255	<code>esp_ble_mesh_prov_cb_param_t::provisioner_set_pro</code> <code>(C++ member)</code> , 253
<code>esp_ble_mesh_prov_cb_param_t::provisioner_set_st</code> <code>(C++ member)</code> , 253	<code>esp_ble_mesh_prov_cb_param_t::provisioner_set_st</code> <code>(C++ member)</code> , 253
<code>esp_ble_mesh_prov_cb_param_t::provisioner_store_n</code> <code>(C++ member)</code> , 253	<code>esp_ble_mesh_prov_cb_param_t::provisioner_store_n</code> <code>(C++ member)</code> , 253
<code>esp_ble_mesh_prov_cb_param_t::provisioner_update</code> <code>(C++ member)</code> , 253	<code>esp_ble_mesh_prov_cb_param_t::provisioner_update</code> <code>(C++ member)</code> , 253
<code>esp_ble_mesh_prov_cb_param_t::provisioner_update</code> <code>(C++ member)</code> , 255	<code>esp_ble_mesh_prov_cb_param_t::provisioner_update</code> <code>(C++ member)</code> , 253
<code>esp_ble_mesh_prov_cb_param_t::provisioner_update</code> <code>(C++ member)</code> , 255	<code>esp_ble_mesh_prov_cb_param_t::proxy_client_add_fi</code> <code>(C++ member)</code> , 256
<code>esp_ble_mesh_prov_cb_param_t::provisioner_update</code> <code>(C++ member)</code> , 254	<code>esp_ble_mesh_prov_cb_param_t::proxy_client_conne</code> <code>(C++ member)</code> , 256
<code>esp_ble_mesh_prov_cb_param_t::provisioner_update</code> <code>(C++ member)</code> , 254	<code>esp_ble_mesh_prov_cb_param_t::proxy_client_conne</code> <code>(C++ member)</code> , 256

- (C++ member), 255
- esp\_ble\_mesh\_prov\_cb\_param\_t::proxy\_client\_disconnect (C++ member), 256
- esp\_ble\_mesh\_prov\_cb\_param\_t::proxy\_client\_disconnect (C++ member), 256
- esp\_ble\_mesh\_prov\_cb\_param\_t::proxy\_client\_register (C++ member), 255
- esp\_ble\_mesh\_prov\_cb\_param\_t::proxy\_client\_register (C++ member), 256
- esp\_ble\_mesh\_prov\_cb\_param\_t::proxy\_client\_remove (C++ member), 256
- esp\_ble\_mesh\_prov\_cb\_param\_t::proxy\_client\_remove (C++ member), 256
- esp\_ble\_mesh\_prov\_cb\_param\_t::proxy\_client\_set (C++ member), 256
- esp\_ble\_mesh\_prov\_cb\_param\_t::rssi (C++ member), 254
- esp\_ble\_mesh\_prov\_cb\_param\_t::rx\_ttl (C++ member), 254
- esp\_ble\_mesh\_prov\_cb\_param\_t::set\_fast\_provisioner\_add\_local\_app\_key (C++ member), 255
- esp\_ble\_mesh\_prov\_cb\_param\_t::set\_fast\_provisioner\_add\_local\_net\_key (C++ member), 255
- esp\_ble\_mesh\_prov\_cb\_param\_t::type (C++ member), 254
- esp\_ble\_mesh\_prov\_cb\_param\_t::uid (C++ member), 255
- esp\_ble\_mesh\_prov\_cb\_param\_t::[anonymous] (C++ enum), 256
- esp\_ble\_mesh\_prov\_cb\_t (C++ type), 318
- esp\_ble\_mesh\_prov\_data\_info\_t (C++ class), 278
- esp\_ble\_mesh\_prov\_data\_info\_t::flag (C++ member), 278
- esp\_ble\_mesh\_prov\_data\_info\_t::flags (C++ member), 278
- esp\_ble\_mesh\_prov\_data\_info\_t::iv\_index (C++ member), 278
- esp\_ble\_mesh\_prov\_data\_info\_t::net\_idx (C++ member), 278
- ESP\_BLE\_MESH\_PROV\_EVT\_MAX (C++ enumerator), 303
- ESP\_BLE\_MESH\_PROV\_GATT (C++ enumerator), 298
- ESP\_BLE\_MESH\_PROV\_INPUT\_OOB\_MAX\_LEN (C 宏), 284
- ESP\_BLE\_MESH\_PROV\_OOB\_2D\_CODE (C++ enumerator), 298
- ESP\_BLE\_MESH\_PROV\_OOB\_BAR\_CODE (C++ enumerator), 298
- ESP\_BLE\_MESH\_PROV\_OOB\_IN\_BOX (C++ enumerator), 298
- ESP\_BLE\_MESH\_PROV\_OOB\_IN\_MANUAL (C++ enumerator), 299
- esp\_ble\_mesh\_prov\_oob\_info\_t (C++ enum), 298
- ESP\_BLE\_MESH\_PROV\_OOB\_NFC (C++ enumerator), 298
- ESP\_BLE\_MESH\_PROV\_OOB\_NUMBER (C++ enumerator), 298
- ESP\_BLE\_MESH\_PROV\_OOB\_ON\_BOX (C++ enumerator), 298
- ESP\_BLE\_MESH\_PROV\_OOB\_ON\_DEV (C++ enumerator), 298
- ESP\_BLE\_MESH\_PROV\_OOB\_ON\_PAPER (C++ enumerator), 298
- ESP\_BLE\_MESH\_PROV\_OOB\_OTHER (C++ enumerator), 298
- ESP\_BLE\_MESH\_PROV\_OOB\_STRING (C++ enumerator), 298
- ESP\_BLE\_MESH\_PROV\_OOB\_URI (C++ enumerator), 298
- ESP\_BLE\_MESH\_PROV\_OUTPUT\_OOB\_MAX\_LEN (C 宏), 284
- ESP\_BLE\_MESH\_PROV\_REGISTER\_COMP\_EVT (C++ enumerator), 299
- ESP\_BLE\_MESH\_PROV\_STATIC\_OOB\_MAX\_LEN (C 宏), 284
- esp\_ble\_mesh\_prov\_t (C++ class), 277
- esp\_ble\_mesh\_provisioner\_add\_local\_app\_key (C++ function), 311
- ESP\_BLE\_MESH\_PROVISIONER\_ADD\_LOCAL\_APP\_KEY\_COMP\_EVT (C++ enumerator), 301
- esp\_ble\_mesh\_provisioner\_add\_local\_net\_key (C++ function), 311
- ESP\_BLE\_MESH\_PROVISIONER\_ADD\_LOCAL\_NET\_KEY\_COMP\_EVT (C++ enumerator), 301
- esp\_ble\_mesh\_provisioner\_add\_unprov\_dev (C++ function), 316
- ESP\_BLE\_MESH\_PROVISIONER\_ADD\_UNPROV\_DEV\_COMP\_EVT (C++ enumerator), 300
- esp\_ble\_mesh\_provisioner\_bind\_app\_key\_to\_local\_model (C++ function), 311
- ESP\_BLE\_MESH\_PROVISIONER\_BIND\_APP\_KEY\_TO\_MODEL\_CODE (C++ enumerator), 301
- esp\_ble\_mesh\_provisioner\_close\_settings\_with\_index (C++ function), 313
- ESP\_BLE\_MESH\_PROVISIONER\_CLOSE\_SETTINGS\_WITH\_INDEX (C++ enumerator), 301
- esp\_ble\_mesh\_provisioner\_close\_settings\_with\_uid (C++ function), 313
- ESP\_BLE\_MESH\_PROVISIONER\_CLOSE\_SETTINGS\_WITH\_UID (C++ enumerator), 301
- esp\_ble\_mesh\_provisioner\_delete\_dev (C++ function), 317
- ESP\_BLE\_MESH\_PROVISIONER\_DELETE\_DEV\_COMP\_EVT (C++ enumerator), 300
- esp\_ble\_mesh\_provisioner\_delete\_node\_with\_addr (C++ function), 310
- ESP\_BLE\_MESH\_PROVISIONER\_DELETE\_NODE\_WITH\_ADDR\_CODE (C++ enumerator), 301
- esp\_ble\_mesh\_provisioner\_delete\_node\_with\_uuid (C++ function), 310
- ESP\_BLE\_MESH\_PROVISIONER\_DELETE\_NODE\_WITH\_UUID\_CODE (C++ enumerator), 301
- esp\_ble\_mesh\_provisioner\_delete\_settings\_with\_index (C++ function), 314
- ESP\_BLE\_MESH\_PROVISIONER\_DELETE\_SETTINGS\_WITH\_INDEX (C++ enumerator), 301

(C++ enumerator), 302  
 esp\_ble\_mesh\_provisioner\_delete\_settings ESP\_BLE\_MESH\_PROVISIONER\_PROV\_ENABLE\_COMP\_EVT  
 (C++ function), 314 (C++ enumerator), 300  
 ESP\_BLE\_MESH\_PROVISIONER\_DELETE\_SETTINGS ESP\_BLE\_MESH\_PROVISIONER\_PROV\_INPUT\_EVT  
 (C++ enumerator), 302 (C++ enumerator), 300  
 esp\_ble\_mesh\_provisioner\_direct\_erase\_settings ESP\_BLE\_MESH\_PROVISIONER\_PROV\_INPUT\_NUMBER\_COMP\_EVT  
 (C++ function), 313 (C++ enumerator), 301  
 ESP\_BLE\_MESH\_PROVISIONER\_DIRECT\_ERASE\_SETTINGS ESP\_BLE\_MESH\_PROVISIONER\_PROV\_INPUT\_STRING\_COMP\_EVT  
 (C++ enumerator), 301 (C++ enumerator), 301  
 ESP\_BLE\_MESH\_PROVISIONER\_ENABLE\_HEARTBEAT ESP\_BLE\_MESH\_PROVISIONER\_PROV\_LINK\_CLOSE\_EVT  
 (C++ enumerator), 301 (C++ enumerator), 300  
 esp\_ble\_mesh\_provisioner\_get\_free\_settings ESP\_BLE\_MESH\_PROVISIONER\_PROV\_LINK\_OPEN\_EVT  
 (C++ function), 314 (C++ enumerator), 300  
 esp\_ble\_mesh\_provisioner\_get\_local\_app\_key ESP\_BLE\_MESH\_PROVISIONER\_PROV\_OUTPUT\_EVT  
 (C++ function), 311 (C++ enumerator), 300  
 esp\_ble\_mesh\_provisioner\_get\_local\_net\_key ESP\_BLE\_MESH\_PROVISIONER\_PROV\_READ\_OOB\_PUB\_KEY\_COMP\_EVT  
 (C++ function), 312 (C++ enumerator), 301  
 esp\_ble\_mesh\_provisioner\_get\_node\_index ESP\_BLE\_MESH\_PROVISIONER\_PROV\_READ\_OOB\_PUB\_KEY\_EVT  
 (C++ function), 309 (C++ enumerator), 300  
 esp\_ble\_mesh\_provisioner\_get\_node\_name esp\_ble\_mesh\_provisioner\_read\_oob\_pub\_key  
 (C++ function), 309 (C++ function), 315  
 esp\_ble\_mesh\_provisioner\_get\_node\_table esp\_ble\_mesh\_provisioner\_rcv\_heartbeat  
 (C++ function), 310 (C++ function), 312  
 esp\_ble\_mesh\_provisioner\_get\_node\_with\_uuid ESP\_BLE\_MESH\_PROVISIONER\_RECV\_HEARTBEAT\_MESSAGE\_EVT  
 (C++ function), 310 (C++ enumerator), 301  
 esp\_ble\_mesh\_provisioner\_get\_node\_with\_uuid ESP\_BLE\_MESH\_PROVISIONER\_RECV\_UNPROV\_ADV\_PKT\_EVT  
 (C++ function), 310 (C++ enumerator), 300  
 esp\_ble\_mesh\_provisioner\_get\_node\_with\_uuid esp\_ble\_mesh\_provisioner\_set\_dev\_uuid\_match  
 (C++ function), 310 (C++ function), 317  
 esp\_ble\_mesh\_provisioner\_get\_prov\_node ESP\_BLE\_MESH\_PROVISIONER\_SET\_DEV\_UUID\_MATCH\_COMP\_EVT  
 (C++ function), 310 (C++ enumerator), 300  
 esp\_ble\_mesh\_provisioner\_get\_settings\_index esp\_ble\_mesh\_provisioner\_set\_heartbeat\_filter\_info  
 (C++ function), 314 (C++ function), 312  
 esp\_ble\_mesh\_provisioner\_get\_settings\_index ESP\_BLE\_MESH\_PROVISIONER\_SET\_HEARTBEAT\_FILTER\_INFO  
 (C++ function), 314 (C++ enumerator), 301  
 esp\_ble\_mesh\_provisioner\_input\_number esp\_ble\_mesh\_provisioner\_set\_heartbeat\_filter\_type  
 (C++ function), 316 (C++ function), 312  
 esp\_ble\_mesh\_provisioner\_input\_string ESP\_BLE\_MESH\_PROVISIONER\_SET\_HEARTBEAT\_FILTER\_TYPE  
 (C++ function), 316 (C++ enumerator), 301  
 esp\_ble\_mesh\_provisioner\_open\_settings esp\_ble\_mesh\_provisioner\_set\_node\_name  
 (C++ function), 313 (C++ function), 309  
 ESP\_BLE\_MESH\_PROVISIONER\_OPEN\_SETTINGS ESP\_BLE\_MESH\_PROVISIONER\_SET\_NODE\_NAME\_COMP\_EVT  
 (C++ enumerator), 301 (C++ enumerator), 301  
 esp\_ble\_mesh\_provisioner\_open\_settings esp\_ble\_mesh\_provisioner\_set\_primary\_elem\_addr  
 (C++ function), 313 (C++ function), 318  
 ESP\_BLE\_MESH\_PROVISIONER\_OPEN\_SETTINGS ESP\_BLE\_MESH\_PROVISIONER\_SET\_PRIMARY\_ELEM\_ADDR\_COMP\_EVT  
 (C++ enumerator), 301 (C++ enumerator), 301  
 ESP\_BLE\_MESH\_PROVISIONER\_PROV\_COMPLETE esp\_ble\_mesh\_provisioner\_set\_prov\_data\_info  
 (C++ enumerator), 300 (C++ function), 318  
 ESP\_BLE\_MESH\_PROVISIONER\_PROV\_DEV\_WITH\_UUID ESP\_BLE\_MESH\_PROVISIONER\_SET\_PROV\_DATA\_INFO\_COMP\_EVT  
 (C++ enumerator), 300 (C++ enumerator), 300  
 esp\_ble\_mesh\_provisioner\_prov\_device\_with\_uuid esp\_ble\_mesh\_provisioner\_set\_static\_oob\_value  
 (C++ function), 317 (C++ function), 318  
 esp\_ble\_mesh\_provisioner\_prov\_disable ESP\_BLE\_MESH\_PROVISIONER\_SET\_STATIC\_OOB\_VALUE\_COMP\_EVT  
 (C++ function), 316 (C++ enumerator), 301  
 ESP\_BLE\_MESH\_PROVISIONER\_PROV\_DISABLE\_COMP\_EVT esp\_ble\_mesh\_provisioner\_store\_node\_comp\_data  
 (C++ enumerator), 300 (C++ function), 310  
 esp\_ble\_mesh\_provisioner\_prov\_enable ESP\_BLE\_MESH\_PROVISIONER\_STORE\_NODE\_COMP\_DATA\_COMP\_EVT

- (C++ enumerator), 301
- esp\_ble\_mesh\_provisioner\_update\_local\_app\_key (C++ function), 311
- ESP\_BLE\_MESH\_PROVISIONER\_UPDATE\_LOCAL\_APP\_KEY (C++ enumerator), 301
- esp\_ble\_mesh\_provisioner\_update\_local\_app\_key\_mesh\_register\_health\_server\_callback (C++ function), 311
- ESP\_BLE\_MESH\_PROVISIONER\_UPDATE\_LOCAL\_APP\_KEY\_MESH\_REGISTER\_HEALTH\_SERVER\_CALLBACK (C++ enumerator), 301
- esp\_ble\_mesh\_provisioner\_update\_local\_app\_key\_mesh\_register\_light\_client\_callback (C++ function), 312
- ESP\_BLE\_MESH\_PROVISIONER\_UPDATE\_LOCAL\_APP\_KEY\_MESH\_REGISTER\_LIGHT\_CLIENT\_CALLBACK (C++ enumerator), 301
- esp\_ble\_mesh\_provisioner\_update\_local\_app\_key\_mesh\_register\_lighting\_server\_callback (C++ function), 312
- ESP\_BLE\_MESH\_PROVISIONER\_UPDATE\_LOCAL\_APP\_KEY\_MESH\_REGISTER\_LIGHTING\_SERVER\_CALLBACK (C++ enumerator), 301
- esp\_ble\_mesh\_provisioner\_update\_local\_app\_key\_mesh\_register\_prov\_callback (C++ function), 312
- ESP\_BLE\_MESH\_PROVISIONER\_UPDATE\_LOCAL\_APP\_KEY\_MESH\_REGISTER\_PROV\_CALLBACK (C++ enumerator), 301
- esp\_ble\_mesh\_proxy\_client\_add\_filter\_add\_app\_key\_mesh\_register\_sensor\_client\_callback (C++ function), 320
- ESP\_BLE\_MESH\_PROXY\_CLIENT\_ADD\_FILTER\_ADD\_APP\_KEY\_MESH\_REGISTER\_SENSOR\_CLIENT\_CALLBACK (C++ enumerator), 302
- esp\_ble\_mesh\_proxy\_client\_add\_filter\_add\_app\_key\_mesh\_register\_sensor\_server\_callback (C++ function), 320
- ESP\_BLE\_MESH\_PROXY\_CLIENT\_ADD\_FILTER\_ADD\_APP\_KEY\_MESH\_REGISTER\_SENSOR\_SERVER\_CALLBACK (C++ enumerator), 302
- esp\_ble\_mesh\_proxy\_client\_connect esp\_ble\_mesh\_register\_time\_scene\_client\_callback (C++ function), 319
- ESP\_BLE\_MESH\_PROXY\_CLIENT\_CONNECT esp\_ble\_mesh\_register\_time\_scene\_client\_callback (C++ enumerator), 302
- ESP\_BLE\_MESH\_PROXY\_CLIENT\_CONNECTED\_EVENT esp\_ble\_mesh\_register\_time\_scene\_server\_callback (C++ function), 302
- (C++ enumerator), 302
- esp\_ble\_mesh\_proxy\_client\_disconnect ESP\_BLE\_MESH\_RELAY\_DISABLED (C 宏), 282
- (C++ function), 319
- ESP\_BLE\_MESH\_RELAY\_ENABLED (C 宏), 282
- ESP\_BLE\_MESH\_PROXY\_CLIENT\_DISCONNECT\_EVENT ESP\_BLE\_MESH\_RELAY\_NOT\_SUPPORTED (C 宏), 282
- (C++ enumerator), 302
- ESP\_BLE\_MESH\_PROXY\_CLIENT\_DISCONNECTED\_EVENT ESP\_BLE\_MESH\_SAMPLE\_FUNC\_ACCUMULATED (C++ enumerator), 440
- (C++ enumerator), 302
- ESP\_BLE\_MESH\_PROXY\_CLIENT\_DISCONNECTED\_EVENT ESP\_BLE\_MESH\_SAMPLE\_FUNC\_ARITHMETIC\_MEAN (C++ enumerator), 440
- (C++ enumerator), 302
- ESP\_BLE\_MESH\_PROXY\_CLIENT\_RECV\_ADV\_PKT\_EVENT ESP\_BLE\_MESH\_SAMPLE\_FUNC\_COUNT (C++ enumerator), 440
- (C++ enumerator), 302
- ESP\_BLE\_MESH\_PROXY\_CLIENT\_RECV\_FILTER\_EVENT ESP\_BLE\_MESH\_SAMPLE\_FUNC\_INSTANTANEOUS (C++ enumerator), 440
- (C++ enumerator), 302
- esp\_ble\_mesh\_proxy\_client\_remove\_filter\_remove\_app\_key\_mesh\_sample\_func\_maximum (C++ function), 320
- (C++ function), 320
- ESP\_BLE\_MESH\_PROXY\_CLIENT\_REMOVE\_FILTER\_REMOVE\_APP\_KEY\_MESH\_SAMPLE\_FUNC\_MINIMUM (C++ enumerator), 440
- (C++ enumerator), 302
- esp\_ble\_mesh\_proxy\_client\_set\_filter\_type\_remove\_app\_key\_mesh\_sample\_func\_rms (C++ function), 320
- (C++ function), 320
- ESP\_BLE\_MESH\_PROXY\_CLIENT\_SET\_FILTER\_TYPE\_REMOVE\_APP\_KEY\_MESH\_SAMPLE\_FUNC\_UNSPECIFIED (C++ enumerator), 440
- (C++ enumerator), 302
- esp\_ble\_mesh\_proxy\_filter\_type\_t (C++ enum), 299
- esp\_ble\_mesh\_proxy\_gatt\_disable (C++ function), 319
- esp\_ble\_mesh\_proxy\_gatt\_enable (C++ function), 319
- esp\_ble\_mesh\_proxy\_identity\_enable (C++ function), 319
- ESP\_BLE\_MESH\_PUBLISH\_TRANSMIT (C 宏), 283
- ESP\_BLE\_MESH\_PUSH (C++ enumerator), 298
- esp\_ble\_mesh\_register\_config\_client\_callback esp\_ble\_mesh\_scene\_delete\_t (C++ class), 445
- (C++ function), 321
- esp\_ble\_mesh\_register\_config\_server\_callback esp\_ble\_mesh\_scene\_delete\_t::scene\_number (C++ member), 445
- (C++ function), 321
- esp\_ble\_mesh\_register\_custom\_model\_callback ESP\_BLE\_MESH\_SCENE\_NOT\_FOUND (C 宏), 461
- (C++ function), 307
- esp\_ble\_mesh\_register\_generic\_client\_callback ESP\_BLE\_MESH\_SCENE\_NUMBER\_LEN (C 宏), 459
- (C++ function), 344
- esp\_ble\_mesh\_register\_generic\_server\_callback esp\_ble\_mesh\_scene\_recall\_t (C++ class), 445
- (C++ function), 345
- esp\_ble\_mesh\_register\_health\_client\_callback esp\_ble\_mesh\_scene\_recall\_t::delay (C++ member), 445
- (C++ class), 447
- esp\_ble\_mesh\_scene\_recall\_t::op\_en (C++ member), 445
- esp\_ble\_mesh\_scene\_recall\_t::scene\_number (C++ member), 445
- esp\_ble\_mesh\_scene\_recall\_t::tid (C++ member), 445
- esp\_ble\_mesh\_scene\_recall\_t::trans\_time (C++ member), 445
- ESP\_BLE\_MESH\_SCENE\_REG\_FULL (C 宏), 460
- esp\_ble\_mesh\_scene\_register\_status\_cb\_t (C++ class), 447



---

esp\_ble\_mesh\_scene\_register\_status\_cb\_t (C++ member), 447  
 esp\_ble\_mesh\_scene\_register\_status\_cb\_t::current\_scene (C++ member), 447  
 esp\_ble\_mesh\_scene\_register\_status\_cb\_t::open (C++ member), 447  
 esp\_ble\_mesh\_scene\_register\_status\_cb\_t::remain\_time (C++ member), 447  
 esp\_ble\_mesh\_scene\_register\_status\_cb\_t::status\_code (C++ member), 447  
 esp\_ble\_mesh\_scene\_register\_status\_cb\_t::target\_scene (C++ member), 447  
 esp\_ble\_mesh\_scene\_register\_t (C++ class), 449  
 esp\_ble\_mesh\_scene\_register\_t::scene\_number (C++ member), 450  
 esp\_ble\_mesh\_scene\_register\_t::scene\_type (C++ member), 450  
 esp\_ble\_mesh\_scene\_register\_t::scene\_value (C++ member), 450  
 esp\_ble\_mesh\_scene\_setup\_srv\_t (C++ class), 451  
 esp\_ble\_mesh\_scene\_setup\_srv\_t::model (C++ member), 451  
 esp\_ble\_mesh\_scene\_setup\_srv\_t::rsp\_ctrl (C++ member), 451  
 esp\_ble\_mesh\_scene\_setup\_srv\_t::state (C++ member), 451  
 esp\_ble\_mesh\_scene\_srv\_t (C++ class), 451  
 esp\_ble\_mesh\_scene\_srv\_t::last (C++ member), 451  
 esp\_ble\_mesh\_scene\_srv\_t::model (C++ member), 451  
 esp\_ble\_mesh\_scene\_srv\_t::rsp\_ctrl (C++ member), 451  
 esp\_ble\_mesh\_scene\_srv\_t::state (C++ member), 451  
 esp\_ble\_mesh\_scene\_srv\_t::transition (C++ member), 451  
 esp\_ble\_mesh\_scene\_status\_cb\_t (C++ class), 447  
 esp\_ble\_mesh\_scene\_status\_cb\_t::current\_scene (C++ member), 447  
 esp\_ble\_mesh\_scene\_status\_cb\_t::open (C++ member), 447  
 esp\_ble\_mesh\_scene\_status\_cb\_t::remain\_time (C++ member), 447  
 esp\_ble\_mesh\_scene\_status\_cb\_t::status\_code (C++ member), 447  
 esp\_ble\_mesh\_scene\_status\_cb\_t::target\_scene (C++ member), 447  
 esp\_ble\_mesh\_scene\_store\_t (C++ class), 444  
 esp\_ble\_mesh\_scene\_store\_t::scene\_number (C++ member), 445  
 ESP\_BLE\_MESH\_SCENE\_SUCCESS (C 宏), 460  
 esp\_ble\_mesh\_scenes\_state\_t (C++ class), 450  
 esp\_ble\_mesh\_scenes\_state\_t::current\_scene (C++ member), 450  
 esp\_ble\_mesh\_scenes\_state\_t::in\_progress (C++ member), 451  
 esp\_ble\_mesh\_scenes\_state\_t::scene\_count (C++ member), 450  
 esp\_ble\_mesh\_schedule\_register\_t (C++ class), 451  
 esp\_ble\_mesh\_schedule\_register\_t::action (C++ member), 452  
 esp\_ble\_mesh\_schedule\_register\_t::day (C++ member), 451  
 esp\_ble\_mesh\_schedule\_register\_t::day\_of\_week (C++ member), 452  
 esp\_ble\_mesh\_schedule\_register\_t::hour (C++ member), 451  
 esp\_ble\_mesh\_schedule\_register\_t::in\_use (C++ member), 451  
 esp\_ble\_mesh\_schedule\_register\_t::minute (C++ member), 451  
 esp\_ble\_mesh\_schedule\_register\_t::month (C++ member), 451  
 esp\_ble\_mesh\_schedule\_register\_t::scene\_number (C++ member), 452  
 esp\_ble\_mesh\_schedule\_register\_t::second (C++ member), 452  
 esp\_ble\_mesh\_schedule\_register\_t::trans\_time (C++ member), 451  
 esp\_ble\_mesh\_schedule\_register\_t::year (C++ member), 451  
 ESP\_BLE\_MESH\_SCHEDULE\_ACT\_NO\_ACTION (C 宏), 460  
 ESP\_BLE\_MESH\_SCHEDULE\_ACT\_SCENE\_RECALL (C 宏), 460  
 ESP\_BLE\_MESH\_SCHEDULE\_ACT\_TURN\_OFF (C 宏), 460  
 ESP\_BLE\_MESH\_SCHEDULE\_ACT\_TURN\_ON (C 宏), 460  
 ESP\_BLE\_MESH\_SCHEDULE\_DAY\_ANY\_DAY (C 宏), 460  
 ESP\_BLE\_MESH\_SCHEDULE\_ENTRY\_MAX\_INDEX (C 宏), 460  
 ESP\_BLE\_MESH\_SCHEDULE\_HOUR\_ANY\_HOUR (C 宏), 460  
 ESP\_BLE\_MESH\_SCHEDULE\_HOUR\_ONCE\_A\_DAY (C 宏), 460  
 ESP\_BLE\_MESH\_SCHEDULE\_SCENE\_NO\_SCENE (C 宏), 460  
 ESP\_BLE\_MESH\_SCHEDULE\_SEC\_ANY\_OF\_HOUR (C 宏), 460  
 ESP\_BLE\_MESH\_SCHEDULE\_SEC\_ANY\_OF\_MIN (C 宏), 460  
 ESP\_BLE\_MESH\_SCHEDULE\_SEC EVERY\_15\_MIN (C 宏), 460  
 ESP\_BLE\_MESH\_SCHEDULE\_SEC EVERY\_15\_SEC (C 宏), 460  
 ESP\_BLE\_MESH\_SCHEDULE\_SEC EVERY\_20\_MIN (C 宏), 460



esp\_ble\_mesh\_sensor\_cadence\_status\_cb\_t esp\_ble\_mesh\_sensor\_client\_set\_state\_t  
 (C++ member), 430 (C++ union), 426  
 esp\_ble\_mesh\_sensor\_cadence\_t (C++ esp\_ble\_mesh\_sensor\_client\_set\_state\_t::cadence\_s  
 class), 432 (C++ member), 426  
 esp\_ble\_mesh\_sensor\_cadence\_t::fast\_cadence esp\_ble\_mesh\_sensor\_client\_set\_state\_t::setting\_s  
 (C++ member), 432 (C++ member), 426  
 esp\_ble\_mesh\_sensor\_cadence\_t::fast\_cadence esp\_ble\_mesh\_sensor\_client\_status\_cb\_t  
 (C++ member), 432 (C++ union), 426  
 esp\_ble\_mesh\_sensor\_cadence\_t::min\_interval esp\_ble\_mesh\_sensor\_client\_status\_cb\_t::cadence\_s  
 (C++ member), 432 (C++ member), 426  
 esp\_ble\_mesh\_sensor\_cadence\_t::period esp\_ble\_mesh\_sensor\_client\_status\_cb\_t::column\_st  
 (C++ member), 432 (C++ member), 427  
 esp\_ble\_mesh\_sensor\_cadence\_t::trigger esp\_ble\_mesh\_sensor\_client\_status\_cb\_t::descripto  
 (C++ member), 432 (C++ member), 426  
 esp\_ble\_mesh\_sensor\_cadence\_t::trigger esp\_ble\_mesh\_sensor\_client\_status\_cb\_t::sensor\_st  
 (C++ member), 432 (C++ member), 427  
 esp\_ble\_mesh\_sensor\_cadence\_t::trigger\_type esp\_ble\_mesh\_sensor\_client\_status\_cb\_t::series\_st  
 (C++ member), 432 (C++ member), 427  
 esp\_ble\_mesh\_sensor\_client\_cb\_event\_t esp\_ble\_mesh\_sensor\_client\_status\_cb\_t::setting\_s  
 (C++ enum), 439 (C++ member), 426  
 esp\_ble\_mesh\_sensor\_client\_cb\_param\_t esp\_ble\_mesh\_sensor\_client\_status\_cb\_t::settings\_  
 (C++ class), 431 (C++ member), 426  
 esp\_ble\_mesh\_sensor\_client\_cb\_param\_t::ESP\_BLE\_MESH\_SENSOR\_CLIENT\_TIMEOUT\_EVT  
 (C++ member), 431 (C++ enumerator), 439  
 esp\_ble\_mesh\_sensor\_client\_cb\_param\_t::esp\_ble\_mesh\_sensor\_column\_get\_t (C++  
 (C++ member), 431 class), 429  
 esp\_ble\_mesh\_sensor\_client\_cb\_param\_t::esp\_ble\_mesh\_sensor\_column\_get\_t::property\_id  
 (C++ member), 431 (C++ member), 429  
 esp\_ble\_mesh\_sensor\_client\_cb\_t (C++ esp\_ble\_mesh\_sensor\_column\_get\_t::raw\_value\_x  
 type), 439 (C++ member), 429  
 ESP\_BLE\_MESH\_SENSOR\_CLIENT\_EVT\_MAX esp\_ble\_mesh\_sensor\_column\_status\_cb\_t  
 (C++ enumerator), 439 (C++ class), 431  
 esp\_ble\_mesh\_sensor\_client\_get\_state esp\_ble\_mesh\_sensor\_column\_status\_cb\_t::property\_  
 (C++ function), 425 (C++ member), 431  
 ESP\_BLE\_MESH\_SENSOR\_CLIENT\_GET\_STATE\_EVT esp\_ble\_mesh\_sensor\_column\_status\_cb\_t::sensor\_co  
 (C++ enumerator), 439 (C++ member), 431  
 esp\_ble\_mesh\_sensor\_client\_get\_state\_t ESP\_BLE\_MESH\_SENSOR\_DATA\_FORMAT\_A (C  
 (C++ union), 426 宏), 438  
 esp\_ble\_mesh\_sensor\_client\_get\_state\_t::ESP\_BLE\_MESH\_SENSOR\_DATA\_FORMAT\_A\_MPID  
 (C++ member), 426 (C 宏), 438  
 esp\_ble\_mesh\_sensor\_client\_get\_state\_t::ESP\_BLE\_MESH\_SENSOR\_DATA\_FORMAT\_A\_MPID\_LEN  
 (C++ member), 426 (C 宏), 438  
 esp\_ble\_mesh\_sensor\_client\_get\_state\_t::ESP\_BLE\_MESH\_SENSOR\_DATA\_FORMAT\_B (C  
 (C++ member), 426 宏), 438  
 esp\_ble\_mesh\_sensor\_client\_get\_state\_t::ESP\_BLE\_MESH\_SENSOR\_DATA\_FORMAT\_B\_MPID  
 (C++ member), 426 (C 宏), 439  
 esp\_ble\_mesh\_sensor\_client\_get\_state\_t::ESP\_BLE\_MESH\_SENSOR\_DATA\_FORMAT\_B\_MPID\_LEN  
 (C++ member), 426 (C 宏), 438  
 esp\_ble\_mesh\_sensor\_client\_get\_state\_t::esp\_ble\_mesh\_sensor\_data\_t (C++ class),  
 (C++ member), 426 432  
 esp\_ble\_mesh\_sensor\_client\_get\_state\_t::esp\_ble\_mesh\_sensor\_data\_t::format  
 (C++ member), 426 (C++ member), 432  
 ESP\_BLE\_MESH\_SENSOR\_CLIENT\_PUBLISH\_EVT esp\_ble\_mesh\_sensor\_data\_t::length  
 (C++ enumerator), 439 (C++ member), 433  
 esp\_ble\_mesh\_sensor\_client\_set\_state esp\_ble\_mesh\_sensor\_data\_t::raw\_value  
 (C++ function), 425 (C++ member), 433  
 ESP\_BLE\_MESH\_SENSOR\_CLIENT\_SET\_STATE\_EVT ESP\_BLE\_MESH\_SENSOR\_DATA\_ZERO\_LEN (C  
 (C++ enumerator), 439 宏), 438

- esp\_ble\_mesh\_sensor\_descriptor\_get\_t (C++ member), 430  
 (C++ class), 428
- esp\_ble\_mesh\_sensor\_descriptor\_get\_t::op\_en (C++ member), 430  
 (C++ member), 428
- esp\_ble\_mesh\_sensor\_descriptor\_get\_t::property\_id (C++ class), 431  
 (C++ member), 428
- ESP\_BLE\_MESH\_SENSOR\_DESCRIPTOR\_LEN (C 宏), 437
- esp\_ble\_mesh\_sensor\_descriptor\_status\_cb\_t (C++ member), 431  
 (C++ class), 430
- esp\_ble\_mesh\_sensor\_descriptor\_status\_cb\_t::data (C++ member), 440  
 (C++ member), 430
- esp\_ble\_mesh\_sensor\_descriptor\_t (C++ class), 431
- esp\_ble\_mesh\_sensor\_descriptor\_t::measure\_per (C++ member), 436  
 (C++ member), 432
- esp\_ble\_mesh\_sensor\_descriptor\_t::negative\_to (C++ member), 436  
 (C++ member), 431
- esp\_ble\_mesh\_sensor\_descriptor\_t::positive\_to (C++ member), 436  
 (C++ member), 431
- esp\_ble\_mesh\_sensor\_descriptor\_t::sampling\_func (C++ member), 439  
 (C++ member), 432
- esp\_ble\_mesh\_sensor\_descriptor\_t::update\_interval (C++ union), 427  
 (C++ member), 432
- ESP\_BLE\_MESH\_SENSOR\_DIVISOR\_TRIGGER\_TYPE\_LEN (C++ member), 428  
 (C 宏), 438
- esp\_ble\_mesh\_sensor\_get\_t (C++ class), 429
- esp\_ble\_mesh\_sensor\_get\_t::op\_en (C++ member), 429
- esp\_ble\_mesh\_sensor\_get\_t::property\_id (C++ member), 429
- esp\_ble\_mesh\_sensor\_message\_opcode\_t (C++ type), 297
- ESP\_BLE\_MESH\_SENSOR\_NOT\_APPL\_MEASURE\_PERIOD (C 宏), 437
- ESP\_BLE\_MESH\_SENSOR\_NOT\_APPL\_UPDATE\_INTERVAL (C 宏), 437
- ESP\_BLE\_MESH\_SENSOR\_PERIOD\_DIVISOR\_MAX (C 宏), 438
- ESP\_BLE\_MESH\_SENSOR\_PROPERTY\_ID\_LEN (C 宏), 437
- esp\_ble\_mesh\_sensor\_sample\_func (C++ enum), 440
- esp\_ble\_mesh\_sensor\_series\_column\_t (C++ class), 433
- esp\_ble\_mesh\_sensor\_series\_column\_t::column\_id (C++ member), 433
- esp\_ble\_mesh\_sensor\_series\_column\_t::raw\_value (C++ member), 433
- esp\_ble\_mesh\_sensor\_series\_column\_t::raw\_value\_max (C++ member), 433
- esp\_ble\_mesh\_sensor\_series\_get\_t (C++ class), 429
- esp\_ble\_mesh\_sensor\_series\_get\_t::op\_en (C++ member), 430
- esp\_ble\_mesh\_sensor\_series\_get\_t::property\_id (C++ member), 430
- esp\_ble\_mesh\_sensor\_series\_get\_t::raw\_value (C++ member), 430
- esp\_ble\_mesh\_sensor\_series\_get\_t::raw\_value\_x2 (C++ member), 430
- esp\_ble\_mesh\_sensor\_series\_status\_cb\_t (C++ member), 431
- esp\_ble\_mesh\_sensor\_series\_status\_cb\_t::property\_id (C++ member), 431
- esp\_ble\_mesh\_sensor\_series\_status\_cb\_t::sensor\_id (C++ member), 431
- esp\_ble\_mesh\_sensor\_server\_cb\_event\_t (C++ member), 436
- esp\_ble\_mesh\_sensor\_server\_cb\_param\_t (C++ class), 436
- esp\_ble\_mesh\_sensor\_server\_cb\_param\_t::ctx (C++ member), 436
- esp\_ble\_mesh\_sensor\_server\_cb\_param\_t::model (C++ member), 436
- esp\_ble\_mesh\_sensor\_server\_cb\_param\_t::value (C++ member), 436
- esp\_ble\_mesh\_sensor\_server\_cb\_t (C++ class), 439
- esp\_ble\_mesh\_sensor\_server\_cb\_value\_t (C++ member), 439
- esp\_ble\_mesh\_sensor\_server\_cb\_value\_t::get (C++ member), 428
- esp\_ble\_mesh\_sensor\_server\_cb\_value\_t::set (C++ member), 428
- esp\_ble\_mesh\_sensor\_server\_cb\_value\_t::state\_change (C++ member), 428
- ESP\_BLE\_MESH\_SENSOR\_SERVER\_EVT\_MAX (C++ enumerator), 440
- ESP\_BLE\_MESH\_SENSOR\_SERVER\_RECV\_GET\_MSG\_EVT (C++ enumerator), 440
- esp\_ble\_mesh\_sensor\_server\_recv\_get\_msg\_t (C++ union), 427
- esp\_ble\_mesh\_sensor\_server\_recv\_get\_msg\_t::sensor\_id (C++ member), 427
- esp\_ble\_mesh\_sensor\_server\_recv\_get\_msg\_t::sensor\_property\_id (C++ member), 427
- esp\_ble\_mesh\_sensor\_server\_recv\_get\_msg\_t::sensor\_value (C++ member), 427
- ESP\_BLE\_MESH\_SENSOR\_SERVER\_RECV\_SET\_MSG\_EVT (C++ enumerator), 440
- esp\_ble\_mesh\_sensor\_server\_recv\_set\_msg\_t (C++ union), 427
- esp\_ble\_mesh\_sensor\_server\_recv\_set\_msg\_t::sensor\_id (C++ member), 427
- esp\_ble\_mesh\_sensor\_server\_recv\_set\_msg\_t::sensor\_property\_id (C++ member), 427
- ESP\_BLE\_MESH\_SENSOR\_SERVER\_STATE\_CHANGE\_EVT (C++ enumerator), 440

(C++ enumerator), 440

esp\_ble\_mesh\_sensor\_server\_state\_change (C++ union), 427

esp\_ble\_mesh\_sensor\_server\_state\_change (C++ member), 427

ESP\_BLE\_MESH\_SENSOR\_SETTING\_ACCESS\_LEN (C 宏), 437

ESP\_BLE\_MESH\_SENSOR\_SETTING\_ACCESS\_READ (C 宏), 437

esp\_ble\_mesh\_sensor\_setting\_get\_t (C++ class), 429

esp\_ble\_mesh\_sensor\_setting\_get\_t::sensor\_property (C++ member), 429

esp\_ble\_mesh\_sensor\_setting\_get\_t::sensor\_set (C++ member), 429

ESP\_BLE\_MESH\_SENSOR\_SETTING\_PROPERTY\_ID\_LEN (C 宏), 437

esp\_ble\_mesh\_sensor\_setting\_set\_t (C++ class), 429

esp\_ble\_mesh\_sensor\_setting\_set\_t::sensor\_property (C++ member), 429

esp\_ble\_mesh\_sensor\_setting\_set\_t::sensor\_set (C++ member), 429

esp\_ble\_mesh\_sensor\_setting\_status\_cb\_t (C++ class), 430

esp\_ble\_mesh\_sensor\_setting\_status\_cb\_t::open (C++ member), 430

esp\_ble\_mesh\_sensor\_setting\_status\_cb\_t::sensor\_set (C++ member), 430

esp\_ble\_mesh\_sensor\_setting\_status\_cb\_t::sensor\_set\_binding\_property\_id (C++ member), 430

esp\_ble\_mesh\_sensor\_setting\_status\_cb\_t::sensor\_set (C++ member), 431

esp\_ble\_mesh\_sensor\_setting\_t (C++ class), 432

esp\_ble\_mesh\_sensor\_setting\_t::access (C++ member), 432

esp\_ble\_mesh\_sensor\_setting\_t::property\_id (C++ member), 432

esp\_ble\_mesh\_sensor\_setting\_t::raw (C++ member), 432

esp\_ble\_mesh\_sensor\_settings\_get\_t (C++ class), 428

esp\_ble\_mesh\_sensor\_settings\_get\_t::sensor\_property (C++ member), 429

esp\_ble\_mesh\_sensor\_settings\_status\_cb\_t (C++ class), 430

esp\_ble\_mesh\_sensor\_settings\_status\_cb\_t::sensor\_property (C++ member), 430

esp\_ble\_mesh\_sensor\_settings\_status\_cb\_t::sensor\_set (C++ member), 430

(C++ member), 430

esp\_ble\_mesh\_sensor\_setup\_srv\_t (C++ class), 434

esp\_ble\_mesh\_sensor\_setup\_srv\_t::model (C++ member), 434

esp\_ble\_mesh\_sensor\_setup\_srv\_t::rsp\_ctrl (C++ member), 434

esp\_ble\_mesh\_sensor\_setup\_srv\_t::state\_count (C++ member), 434

esp\_ble\_mesh\_sensor\_setup\_srv\_t::states (C++ member), 434

esp\_ble\_mesh\_sensor\_srv\_t (C++ class), 433

esp\_ble\_mesh\_sensor\_srv\_t::model (C++ member), 433

esp\_ble\_mesh\_sensor\_srv\_t::rsp\_ctrl (C++ member), 433

esp\_ble\_mesh\_sensor\_srv\_t::state\_count (C++ member), 433

esp\_ble\_mesh\_sensor\_srv\_t::states (C++ member), 434

esp\_ble\_mesh\_sensor\_state\_t (C++ class), 433

esp\_ble\_mesh\_sensor\_state\_t::cadence (C++ member), 433

esp\_ble\_mesh\_sensor\_state\_t::descriptor (C++ member), 433

esp\_ble\_mesh\_sensor\_state\_t::sensor\_data (C++ member), 433

esp\_ble\_mesh\_sensor\_state\_t::sensor\_property\_id (C++ member), 433

esp\_ble\_mesh\_sensor\_state\_t::series\_column (C++ member), 433

esp\_ble\_mesh\_sensor\_state\_t::setting\_count (C++ member), 433

esp\_ble\_mesh\_sensor\_state\_t::settings (C++ member), 433

esp\_ble\_mesh\_sensor\_status\_cb\_t (C++ class), 431

esp\_ble\_mesh\_sensor\_status\_cb\_t::binding\_property\_id (C++ member), 431

esp\_ble\_mesh\_sensor\_status\_cb\_t::marshalled\_sensor\_set (C++ member), 431

ESP\_BLE\_MESH\_SENSOR\_STATUS\_MIN\_INTERVAL\_LEN (C 宏), 438

ESP\_BLE\_MESH\_SENSOR\_STATUS\_MIN\_INTERVAL\_MAX (C 宏), 438

ESP\_BLE\_MESH\_SENSOR\_STATUS\_TRIGGER\_TYPE\_CHAR (C 宏), 438

ESP\_BLE\_MESH\_SENSOR\_STATUS\_TRIGGER\_TYPE\_UINT16 (C 宏), 438

ESP\_BLE\_MESH\_SENSOR\_UNSPECIFIED\_NEG\_TOLERANCE (C 宏), 437

ESP\_BLE\_MESH\_SENSOR\_UNSPECIFIED\_POS\_TOLERANCE (C 宏), 437

ESP\_BLE\_MESH\_SERVER\_AUTO\_RSP (C 宏), 296

ESP\_BLE\_MESH\_SERVER\_FLAG\_MAX (C++ enumerator), 303

esp\_ble\_mesh\_server\_model\_send\_msg (C++ function), 308

ESP\_BLE\_MESH\_SERVER\_MODIFY\_STATE\_MAX









(C++ member), 456  
 esp\_ble\_mesh\_server\_rcv\_scene\_recall\_test\_delete\_mesh\_server\_rcv\_sensor\_descriptor\_get\_t:  
 (C++ member), 456  
 esp\_ble\_mesh\_server\_rcv\_scene\_recall\_test\_delete\_mesh\_server\_rcv\_sensor\_get\_t  
 (C++ member), 456  
 esp\_ble\_mesh\_server\_rcv\_scene\_recall\_test\_delete\_mesh\_server\_rcv\_sensor\_get\_t::op\_en  
 (C++ member), 456  
 esp\_ble\_mesh\_server\_rcv\_scene\_store\_t esp\_ble\_mesh\_server\_rcv\_sensor\_get\_t::property\_i  
 (C++ class), 456  
 esp\_ble\_mesh\_server\_rcv\_scene\_store\_t::esp\_ble\_mesh\_server\_rcv\_sensor\_series\_get\_t  
 (C++ member), 456  
 esp\_ble\_mesh\_server\_rcv\_scheduler\_act esp\_ble\_mesh\_server\_rcv\_sensor\_series\_get\_t  
 (C++ class), 455  
 esp\_ble\_mesh\_server\_rcv\_scheduler\_act esp\_ble\_mesh\_server\_rcv\_sensor\_series\_get\_t::op  
 (C++ member), 455  
 esp\_ble\_mesh\_server\_rcv\_scheduler\_act esp\_ble\_mesh\_server\_rcv\_sensor\_series\_get\_t::pro  
 (C++ member), 455  
 esp\_ble\_mesh\_server\_rcv\_scheduler\_act esp\_ble\_mesh\_server\_rcv\_sensor\_series\_get\_t::raw  
 (C++ class), 456  
 esp\_ble\_mesh\_server\_rcv\_scheduler\_act esp\_ble\_mesh\_server\_rcv\_sensor\_setting\_get\_t  
 (C++ member), 457  
 esp\_ble\_mesh\_server\_rcv\_scheduler\_act esp\_ble\_mesh\_server\_rcv\_sensor\_setting\_get\_t::pr  
 (C++ member), 457  
 esp\_ble\_mesh\_server\_rcv\_scheduler\_act esp\_ble\_mesh\_server\_rcv\_sensor\_setting\_get\_t::se  
 (C++ member), 457  
 esp\_ble\_mesh\_server\_rcv\_scheduler\_act esp\_ble\_mesh\_server\_rcv\_sensor\_setting\_set\_t  
 (C++ member), 457  
 esp\_ble\_mesh\_server\_rcv\_scheduler\_act esp\_ble\_mesh\_server\_rcv\_sensor\_setting\_set\_t::pr  
 (C++ member), 456  
 esp\_ble\_mesh\_server\_rcv\_scheduler\_act esp\_ble\_mesh\_server\_rcv\_sensor\_setting\_set\_t::se  
 (C++ member), 457  
 esp\_ble\_mesh\_server\_rcv\_scheduler\_act esp\_ble\_mesh\_server\_rcv\_sensor\_setting\_set\_t::se  
 (C++ member), 456  
 esp\_ble\_mesh\_server\_rcv\_scheduler\_act esp\_ble\_mesh\_server\_rcv\_sensor\_settings\_get\_t  
 (C++ member), 457  
 esp\_ble\_mesh\_server\_rcv\_scheduler\_act esp\_ble\_mesh\_server\_rcv\_sensor\_settings\_get\_t::p  
 (C++ member), 457  
 esp\_ble\_mesh\_server\_rcv\_scheduler\_act esp\_ble\_mesh\_server\_rcv\_sensor\_status\_t  
 (C++ member), 457  
 esp\_ble\_mesh\_server\_rcv\_scheduler\_act esp\_ble\_mesh\_server\_rcv\_sensor\_status\_t::data  
 (C++ member), 456  
 esp\_ble\_mesh\_server\_rcv\_sensor\_cadence esp\_ble\_mesh\_server\_rcv\_tai\_utc\_delta\_set\_t  
 (C++ class), 435  
 esp\_ble\_mesh\_server\_rcv\_sensor\_cadence esp\_ble\_mesh\_server\_rcv\_tai\_utc\_delta\_set\_t::pad  
 (C++ member), 435  
 esp\_ble\_mesh\_server\_rcv\_sensor\_cadence esp\_ble\_mesh\_server\_rcv\_tai\_utc\_delta\_set\_t::tai  
 (C++ class), 436  
 esp\_ble\_mesh\_server\_rcv\_sensor\_cadence esp\_ble\_mesh\_server\_rcv\_tai\_utc\_delta\_set\_t::tai  
 (C++ member), 436  
 esp\_ble\_mesh\_server\_rcv\_sensor\_cadence esp\_ble\_mesh\_server\_rcv\_time\_role\_set\_t  
 (C++ member), 436  
 esp\_ble\_mesh\_server\_rcv\_sensor\_column esp\_ble\_mesh\_server\_rcv\_time\_role\_set\_t::time\_ro  
 (C++ class), 435  
 esp\_ble\_mesh\_server\_rcv\_sensor\_column esp\_ble\_mesh\_server\_rcv\_time\_set\_t  
 (C++ member), 436  
 esp\_ble\_mesh\_server\_rcv\_sensor\_column esp\_ble\_mesh\_server\_rcv\_time\_set\_t::subsecond  
 (C++ member), 436  
 esp\_ble\_mesh\_server\_rcv\_sensor\_descriptor esp\_ble\_mesh\_server\_rcv\_time\_set\_t::tai\_seconds  
 (C++ class), 435  
 esp\_ble\_mesh\_server\_rcv\_sensor\_descriptor esp\_ble\_mesh\_server\_rcv\_time\_set\_t::tai\_utc\_delt

(C++ member), 455

esp\_ble\_mesh\_server\_rcv\_time\_set\_t::time\_authonmesh\_server\_state\_value\_t::light\_hsl (C++ member), 455

esp\_ble\_mesh\_server\_rcv\_time\_set\_t::time\_authonmesh\_server\_state\_value\_t::light\_hsl\_hue (C++ member), 455

esp\_ble\_mesh\_server\_rcv\_time\_set\_t::time\_authonmesh\_server\_state\_value\_t::light\_hsl\_lightness (C++ member), 455

esp\_ble\_mesh\_server\_rcv\_time\_status\_t esp\_ble\_mesh\_server\_state\_value\_t::light\_hsl\_saturation (C++ class), 457

esp\_ble\_mesh\_server\_rcv\_time\_status\_t esp\_ble\_mesh\_server\_state\_value\_t::light\_lc\_lightness (C++ member), 457

esp\_ble\_mesh\_server\_rcv\_time\_status\_t esp\_ble\_mesh\_server\_state\_value\_t::light\_lightness (C++ member), 457

esp\_ble\_mesh\_server\_rcv\_time\_status\_t esp\_ble\_mesh\_server\_state\_value\_t::light\_lightness (C++ member), 457

esp\_ble\_mesh\_server\_rcv\_time\_status\_t esp\_ble\_mesh\_server\_state\_value\_t::light\_xyl\_lightness (C++ member), 457

esp\_ble\_mesh\_server\_rcv\_time\_status\_t esp\_ble\_mesh\_server\_state\_value\_t::lightness (C++ member), 457

esp\_ble\_mesh\_server\_rcv\_time\_status\_t esp\_ble\_mesh\_server\_state\_value\_t::onoff (C++ member), 457

esp\_ble\_mesh\_server\_rcv\_time\_zone\_set esp\_ble\_mesh\_server\_state\_value\_t::onpowerup (C++ class), 455

esp\_ble\_mesh\_server\_rcv\_time\_zone\_set esp\_ble\_mesh\_server\_state\_value\_t::power (C++ member), 455

esp\_ble\_mesh\_server\_rcv\_time\_zone\_set esp\_ble\_mesh\_server\_state\_value\_t::saturation (C++ member), 455

ESP\_BLE\_MESH\_SERVER\_RSP\_BY\_APP (C 宏), 296

esp\_ble\_mesh\_server\_rsp\_ctrl\_t (C++ class), 281

esp\_ble\_mesh\_server\_rsp\_ctrl\_t::get\_autorespble\_mesh\_set\_fast\_prov\_action (C++ member), 281

esp\_ble\_mesh\_server\_rsp\_ctrl\_t::set\_autorespble\_mesh\_set\_fast\_prov\_action (C++ member), 281

esp\_ble\_mesh\_server\_rsp\_ctrl\_t::status espble\_mesh\_set\_fast\_prov\_info (C++ member), 281

esp\_ble\_mesh\_server\_state\_type\_t (C++ enum), 303

esp\_ble\_mesh\_server\_state\_value\_t esp\_ble\_mesh\_set\_unprovisioned\_device\_name (C++ union), 270

esp\_ble\_mesh\_server\_state\_value\_t::deltaESP\_BLE\_MESH\_SETTINGS\_UID\_SIZE (C 宏), (C++ member), 271

esp\_ble\_mesh\_server\_state\_value\_t::genESP\_BLE\_MESH\_SIG\_MODEL (C 宏), (C++ member), 270

esp\_ble\_mesh\_server\_state\_value\_t::gen\_onoff (C++ class), 341

esp\_ble\_mesh\_server\_state\_value\_t::gen\_onpowerup (C++ member), 341

esp\_ble\_mesh\_server\_state\_value\_t::gen\_powerup (C++ member), 341

esp\_ble\_mesh\_server\_state\_value\_t::hue (C++ member), 341

esp\_ble\_mesh\_server\_state\_value\_t::level (C++ class), 342

esp\_ble\_mesh\_server\_state\_value\_t::light\_ctl (C++ member), 342

esp\_ble\_mesh\_server\_state\_value\_t::light\_ctl (C++ member), 342

esp\_ble\_mesh\_server\_state\_value\_t::light\_hsl (C++ member), 271

esp\_ble\_mesh\_server\_state\_value\_t::light\_hsl\_hue (C++ member), 271

esp\_ble\_mesh\_server\_state\_value\_t::light\_hsl\_lightness (C++ member), 271

esp\_ble\_mesh\_server\_state\_value\_t::light\_hsl\_saturation (C++ member), 271

esp\_ble\_mesh\_server\_state\_value\_t::light\_lc\_lightness (C++ member), 271

esp\_ble\_mesh\_server\_state\_value\_t::light\_lightness (C++ member), 271

esp\_ble\_mesh\_server\_state\_value\_t::light\_lightness (C++ member), 271

esp\_ble\_mesh\_server\_state\_value\_t::light\_xyl\_lightness (C++ member), 271

esp\_ble\_mesh\_server\_state\_value\_t::lightness (C++ member), 271

esp\_ble\_mesh\_server\_state\_value\_t::onoff (C++ member), 270

esp\_ble\_mesh\_server\_state\_value\_t::onpowerup (C++ member), 270

esp\_ble\_mesh\_server\_state\_value\_t::power (C++ member), 271

esp\_ble\_mesh\_server\_state\_value\_t::saturation (C++ member), 271

esp\_ble\_mesh\_server\_state\_value\_t::temperature (C++ member), 271

ESP\_BLE\_MESH\_SERVER\_TRANS\_TIMER\_START (C++ enumerator), 303

ESP\_BLE\_MESH\_SET\_FAST\_PROV\_ACTION (C++ function), 318

ESP\_BLE\_MESH\_SET\_FAST\_PROV\_ACTION\_COMP\_EVT (C++ enumerator), 302

espble\_mesh\_set\_fast\_prov\_info (C++ function), 318

ESP\_BLE\_MESH\_SET\_FAST\_PROV\_INFO\_COMP\_EVT (C++ enumerator), 302

esp\_ble\_mesh\_set\_unprovisioned\_device\_name (C++ function), 315

ESP\_BLE\_MESH\_SETTINGS\_UID\_SIZE (C 宏), 282

ESP\_BLE\_MESH\_SIG\_MODEL (C 宏), 284

esp\_ble\_mesh\_state\_change\_cfg\_appkey\_add\_t (C++ class), 341

esp\_ble\_mesh\_state\_change\_cfg\_appkey\_add\_t::appkey (C++ member), 341

esp\_ble\_mesh\_state\_change\_cfg\_appkey\_add\_t::appkey (C++ member), 341

esp\_ble\_mesh\_state\_change\_cfg\_appkey\_add\_t::netindex (C++ member), 341

esp\_ble\_mesh\_state\_change\_cfg\_appkey\_delete\_t (C++ class), 342

esp\_ble\_mesh\_state\_change\_cfg\_appkey\_delete\_t::appkey (C++ member), 342

esp\_ble\_mesh\_state\_change\_cfg\_appkey\_delete\_t::netindex (C++ member), 342



esp\_ble\_mesh\_state\_change\_gen\_loc\_globals esp\_ble\_mesh\_state\_change\_light\_ctl\_default\_set\_t  
 (C++ member), 364 (C++ member), 412  
 esp\_ble\_mesh\_state\_change\_gen\_loc\_locals esp\_ble\_mesh\_state\_change\_light\_ctl\_set\_t  
 (C++ class), 364 (C++ class), 411  
 esp\_ble\_mesh\_state\_change\_gen\_loc\_locals esp\_ble\_mesh\_state\_change\_light\_ctl\_set\_t::delta  
 (C++ member), 365 (C++ member), 411  
 esp\_ble\_mesh\_state\_change\_gen\_loc\_locals esp\_ble\_mesh\_state\_change\_light\_ctl\_set\_t::lightn  
 (C++ member), 364 (C++ member), 411  
 esp\_ble\_mesh\_state\_change\_gen\_loc\_locals esp\_ble\_mesh\_state\_change\_light\_ctl\_set\_t::temper  
 (C++ member), 365 (C++ member), 411  
 esp\_ble\_mesh\_state\_change\_gen\_loc\_locals esp\_ble\_mesh\_state\_change\_light\_ctl\_temperature\_r  
 (C++ member), 364 (C++ class), 411  
 esp\_ble\_mesh\_state\_change\_gen\_loc\_locals esp\_ble\_mesh\_state\_change\_light\_ctl\_temperature\_r  
 (C++ member), 365 (C++ member), 411  
 esp\_ble\_mesh\_state\_change\_gen\_manu\_prop esp\_ble\_mesh\_state\_change\_light\_ctl\_temperature\_r  
 (C++ class), 365 (C++ member), 411  
 esp\_ble\_mesh\_state\_change\_gen\_manu\_prop esp\_ble\_mesh\_state\_change\_light\_ctl\_temperature\_s  
 (C++ member), 365 (C++ class), 411  
 esp\_ble\_mesh\_state\_change\_gen\_manu\_prop esp\_ble\_mesh\_state\_change\_light\_ctl\_temperature\_s  
 (C++ member), 365 (C++ member), 411  
 esp\_ble\_mesh\_state\_change\_gen\_move\_set esp\_ble\_mesh\_state\_change\_light\_ctl\_temperature\_s  
 (C++ class), 363 (C++ member), 411  
 esp\_ble\_mesh\_state\_change\_gen\_move\_set esp\_ble\_mesh\_state\_change\_light\_hsl\_default\_set\_t  
 (C++ member), 363 (C++ class), 412  
 esp\_ble\_mesh\_state\_change\_gen\_onoff\_set esp\_ble\_mesh\_state\_change\_light\_hsl\_default\_set\_t  
 (C++ class), 363 (C++ member), 412  
 esp\_ble\_mesh\_state\_change\_gen\_onoff\_set esp\_ble\_mesh\_state\_change\_light\_hsl\_default\_set\_t  
 (C++ member), 363 (C++ member), 412  
 esp\_ble\_mesh\_state\_change\_gen\_onpowerup\_set esp\_ble\_mesh\_state\_change\_light\_hsl\_default\_set\_t  
 (C++ class), 363 (C++ member), 412  
 esp\_ble\_mesh\_state\_change\_gen\_onpowerup\_set esp\_ble\_mesh\_state\_change\_light\_hsl\_hue\_set\_t  
 (C++ member), 364 (C++ class), 412  
 esp\_ble\_mesh\_state\_change\_gen\_power\_def esp\_ble\_mesh\_state\_change\_light\_hsl\_hue\_set\_t::hu  
 (C++ class), 364 (C++ member), 412  
 esp\_ble\_mesh\_state\_change\_gen\_power\_def esp\_ble\_mesh\_state\_change\_light\_hsl\_range\_set\_t  
 (C++ member), 364 (C++ class), 412  
 esp\_ble\_mesh\_state\_change\_gen\_power\_level\_set esp\_ble\_mesh\_state\_change\_light\_hsl\_range\_set\_t::  
 (C++ class), 364 (C++ member), 413  
 esp\_ble\_mesh\_state\_change\_gen\_power\_level\_set esp\_ble\_mesh\_state\_change\_light\_hsl\_range\_set\_t::  
 (C++ member), 364 (C++ member), 413  
 esp\_ble\_mesh\_state\_change\_gen\_power\_range\_set esp\_ble\_mesh\_state\_change\_light\_hsl\_range\_set\_t::  
 (C++ class), 364 (C++ member), 413  
 esp\_ble\_mesh\_state\_change\_gen\_power\_range\_set esp\_ble\_mesh\_state\_change\_light\_hsl\_range\_set\_t::  
 (C++ member), 364 (C++ member), 413  
 esp\_ble\_mesh\_state\_change\_gen\_power\_range\_set esp\_ble\_mesh\_state\_change\_light\_hsl\_saturation\_se  
 (C++ member), 364 (C++ class), 412  
 esp\_ble\_mesh\_state\_change\_gen\_user\_prop esp\_ble\_mesh\_state\_change\_light\_hsl\_saturation\_se  
 (C++ class), 365 (C++ member), 412  
 esp\_ble\_mesh\_state\_change\_gen\_user\_prop esp\_ble\_mesh\_state\_change\_light\_hsl\_set\_t  
 (C++ member), 365 (C++ class), 412  
 esp\_ble\_mesh\_state\_change\_gen\_user\_prop esp\_ble\_mesh\_state\_change\_light\_hsl\_set\_t::hue  
 (C++ member), 365 (C++ member), 412  
 esp\_ble\_mesh\_state\_change\_light\_ctl\_def esp\_ble\_mesh\_state\_change\_light\_hsl\_set\_t::lightn  
 (C++ class), 411 (C++ member), 412  
 esp\_ble\_mesh\_state\_change\_light\_ctl\_def esp\_ble\_mesh\_state\_change\_light\_hsl\_set\_t::satura  
 (C++ member), 412 (C++ member), 412  
 esp\_ble\_mesh\_state\_change\_light\_ctl\_def esp\_ble\_mesh\_state\_change\_light\_lc\_light\_onoff\_se  
 (C++ member), 412 (C++ class), 414

esp\_ble\_mesh\_state\_change\_light\_lc\_lightness\_def\_mesh\_state\_change\_light\_xyl\_set\_t::y  
 (C++ member), 414 (C++ member), 413  
 esp\_ble\_mesh\_state\_change\_light\_lc\_mode\_def\_mesh\_state\_change\_scene\_delete\_t  
 (C++ class), 413 (C++ class), 454  
 esp\_ble\_mesh\_state\_change\_light\_lc\_mode\_def\_mesh\_state\_change\_scene\_delete\_t::scene\_n  
 (C++ member), 414 (C++ member), 454  
 esp\_ble\_mesh\_state\_change\_light\_lc\_om\_def\_mesh\_state\_change\_scene\_recall\_t  
 (C++ class), 414 (C++ class), 454  
 esp\_ble\_mesh\_state\_change\_light\_lc\_om\_def\_mesh\_state\_change\_scene\_recall\_t::scene\_n  
 (C++ member), 414 (C++ member), 454  
 esp\_ble\_mesh\_state\_change\_light\_lc\_prop\_def\_mesh\_state\_change\_scene\_store\_t  
 (C++ class), 414 (C++ class), 454  
 esp\_ble\_mesh\_state\_change\_light\_lc\_prop\_def\_mesh\_state\_change\_scene\_store\_t::scene\_nu  
 (C++ member), 414 (C++ member), 454  
 esp\_ble\_mesh\_state\_change\_light\_lc\_prop\_def\_mesh\_state\_change\_scene\_store\_t::scene\_ar  
 (C++ member), 414 (C++ class), 454  
 esp\_ble\_mesh\_state\_change\_light\_lightness\_def\_mesh\_state\_change\_scheduler\_act\_set\_t::ac  
 (C++ class), 411 (C++ member), 455  
 esp\_ble\_mesh\_state\_change\_light\_lightness\_def\_mesh\_state\_change\_scheduler\_act\_set\_t::da  
 (C++ member), 411 (C++ member), 454  
 esp\_ble\_mesh\_state\_change\_light\_lightness\_def\_mesh\_state\_change\_scheduler\_act\_set\_t::da  
 (C++ class), 410 (C++ member), 454  
 esp\_ble\_mesh\_state\_change\_light\_lightness\_def\_mesh\_state\_change\_scheduler\_act\_set\_t::ho  
 (C++ member), 411 (C++ member), 454  
 esp\_ble\_mesh\_state\_change\_light\_lightness\_def\_mesh\_state\_change\_scheduler\_act\_set\_t::in  
 (C++ class), 411 (C++ member), 454  
 esp\_ble\_mesh\_state\_change\_light\_lightness\_def\_mesh\_state\_change\_scheduler\_act\_set\_t::mi  
 (C++ member), 411 (C++ member), 454  
 esp\_ble\_mesh\_state\_change\_light\_lightness\_def\_mesh\_state\_change\_scheduler\_act\_set\_t::mo  
 (C++ member), 411 (C++ member), 454  
 esp\_ble\_mesh\_state\_change\_light\_lightness\_def\_mesh\_state\_change\_scheduler\_act\_set\_t::sc  
 (C++ class), 410 (C++ member), 455  
 esp\_ble\_mesh\_state\_change\_light\_lightness\_def\_mesh\_state\_change\_scheduler\_act\_set\_t::se  
 (C++ member), 410 (C++ member), 454  
 esp\_ble\_mesh\_state\_change\_light\_xyl\_def\_mesh\_state\_change\_scheduler\_act\_set\_t::tr  
 (C++ class), 413 (C++ member), 455  
 esp\_ble\_mesh\_state\_change\_light\_xyl\_def\_mesh\_state\_change\_scheduler\_act\_set\_t::ye  
 (C++ member), 413 (C++ member), 454  
 esp\_ble\_mesh\_state\_change\_light\_xyl\_def\_mesh\_state\_change\_sensor\_cadence\_set\_t  
 (C++ member), 413 (C++ class), 434  
 esp\_ble\_mesh\_state\_change\_light\_xyl\_def\_mesh\_state\_change\_sensor\_cadence\_set\_t::f  
 (C++ member), 413 (C++ member), 434  
 esp\_ble\_mesh\_state\_change\_light\_xyl\_range\_def\_mesh\_state\_change\_sensor\_cadence\_set\_t::f  
 (C++ class), 413 (C++ member), 434  
 esp\_ble\_mesh\_state\_change\_light\_xyl\_range\_def\_mesh\_state\_change\_sensor\_cadence\_set\_t::m  
 (C++ member), 413 (C++ member), 434  
 esp\_ble\_mesh\_state\_change\_light\_xyl\_range\_def\_mesh\_state\_change\_sensor\_cadence\_set\_t::p  
 (C++ member), 413 (C++ member), 434  
 esp\_ble\_mesh\_state\_change\_light\_xyl\_range\_def\_mesh\_state\_change\_sensor\_cadence\_set\_t::p  
 (C++ member), 413 (C++ member), 434  
 esp\_ble\_mesh\_state\_change\_light\_xyl\_range\_def\_mesh\_state\_change\_sensor\_cadence\_set\_t::t  
 (C++ member), 413 (C++ member), 434  
 esp\_ble\_mesh\_state\_change\_light\_xyl\_set\_def\_mesh\_state\_change\_sensor\_cadence\_set\_t::t  
 (C++ class), 413 (C++ member), 434  
 esp\_ble\_mesh\_state\_change\_light\_xyl\_set\_def\_mesh\_state\_change\_sensor\_cadence\_set\_t::t  
 (C++ member), 413 (C++ member), 434  
 esp\_ble\_mesh\_state\_change\_light\_xyl\_set\_def\_mesh\_state\_change\_sensor\_setting\_set\_t  
 (C++ member), 413 (C++ class), 434



- (C++ member), 447
- esp\_ble\_mesh\_tai\_utc\_delta\_status\_cb\_t:establish\_time\_scene\_client\_set\_state\_t::time  
(C++ member), 446
- esp\_ble\_mesh\_tai\_utc\_delta\_status\_cb\_t:establish\_time\_scene\_client\_set\_state\_t::time  
(C++ member), 447
- ESP\_BLE\_MESH\_TAMPER\_ERROR (C 宏), 382
- ESP\_BLE\_MESH\_TAMPER\_WARNING (C 宏), 382
- ESP\_BLE\_MESH\_TIME\_AUTHORITY (C 宏), 460
- ESP\_BLE\_MESH\_TIME\_CLINET (C 宏), 460
- ESP\_BLE\_MESH\_TIME\_NONE (C 宏), 460
- ESP\_BLE\_MESH\_TIME\_RELAY (C 宏), 460
- esp\_ble\_mesh\_time\_role\_set\_t (C++ class), 444
- esp\_ble\_mesh\_time\_role\_set\_t::time\_role  
(C++ member), 444
- esp\_ble\_mesh\_time\_role\_status\_cb\_t  
(C++ class), 447
- esp\_ble\_mesh\_time\_role\_status\_cb\_t::time\_role  
(C++ member), 447
- esp\_ble\_mesh\_time\_scene\_client\_cb\_event\_t  
(C++ enum), 461
- esp\_ble\_mesh\_time\_scene\_client\_cb\_param\_t  
(C++ class), 448
- esp\_ble\_mesh\_time\_scene\_client\_cb\_param\_t:role\_mode  
(C++ member), 448
- esp\_ble\_mesh\_time\_scene\_client\_cb\_param\_t:role\_mode  
(C++ member), 448
- esp\_ble\_mesh\_time\_scene\_client\_cb\_param\_t:status  
(C++ member), 448
- esp\_ble\_mesh\_time\_scene\_client\_cb\_t  
(C++ type), 461
- ESP\_BLE\_MESH\_TIME\_SCENE\_CLIENT\_EVT\_MAX  
(C++ enumerator), 461
- esp\_ble\_mesh\_time\_scene\_client\_get\_status  
(C++ function), 440
- ESP\_BLE\_MESH\_TIME\_SCENE\_CLIENT\_GET\_STATUS\_EVT  
(C++ enumerator), 461
- esp\_ble\_mesh\_time\_scene\_client\_get\_status  
(C++ union), 441
- esp\_ble\_mesh\_time\_scene\_client\_get\_status:role\_mode  
(C++ member), 441
- ESP\_BLE\_MESH\_TIME\_SCENE\_CLIENT\_PUBLISH\_EVT  
(C++ enumerator), 461
- esp\_ble\_mesh\_time\_scene\_client\_set\_status  
(C++ function), 441
- ESP\_BLE\_MESH\_TIME\_SCENE\_CLIENT\_SET\_STATUS\_EVT  
(C++ enumerator), 461
- esp\_ble\_mesh\_time\_scene\_client\_set\_status  
(C++ union), 441
- esp\_ble\_mesh\_time\_scene\_client\_set\_status:role\_mode  
(C++ member), 441
- esp\_ble\_mesh\_time\_scene\_client\_set\_status:role\_mode  
(C++ member), 441
- esp\_ble\_mesh\_time\_scene\_client\_set\_status:role\_mode  
(C++ member), 441
- (C++ member), 441
- (C++ member), 441
- (C++ member), 441
- esp\_ble\_mesh\_time\_scene\_client\_set\_state\_t::time  
(C++ member), 441
- (C++ member), 441
- esp\_ble\_mesh\_time\_scene\_client\_status\_cb\_t  
(C++ union), 442
- esp\_ble\_mesh\_time\_scene\_client\_status\_cb\_t::scene  
(C++ member), 442
- esp\_ble\_mesh\_time\_scene\_client\_status\_cb\_t::scene  
(C++ member), 442
- esp\_ble\_mesh\_time\_scene\_client\_status\_cb\_t::sched  
(C++ member), 442
- esp\_ble\_mesh\_time\_scene\_client\_status\_cb\_t::sched  
(C++ member), 442
- esp\_ble\_mesh\_time\_scene\_client\_status\_cb\_t::tai\_u  
(C++ member), 442
- esp\_ble\_mesh\_time\_scene\_client\_status\_cb\_t::time  
(C++ member), 442
- esp\_ble\_mesh\_time\_scene\_client\_status\_cb\_t::time  
(C++ member), 442
- esp\_ble\_mesh\_time\_scene\_client\_status\_cb\_t::time  
(C++ member), 442
- ESP\_BLE\_MESH\_TIME\_SCENE\_CLIENT\_TIMEOUT\_EVT  
(C++ enumerator), 461
- esp\_ble\_mesh\_time\_scene\_message\_opcode\_t  
(C++ type), 297
- esp\_ble\_mesh\_time\_scene\_server\_cb\_event\_t  
(C++ enum), 461
- esp\_ble\_mesh\_time\_scene\_server\_cb\_param\_t  
(C++ class), 457
- esp\_ble\_mesh\_time\_scene\_server\_cb\_param\_t::ctx  
(C++ member), 457
- esp\_ble\_mesh\_time\_scene\_server\_cb\_param\_t::model  
(C++ member), 457
- esp\_ble\_mesh\_time\_scene\_server\_cb\_param\_t::value  
(C++ member), 457
- esp\_ble\_mesh\_time\_scene\_server\_cb\_t  
(C++ type), 461
- esp\_ble\_mesh\_time\_scene\_server\_cb\_value\_t  
(C++ union), 443
- esp\_ble\_mesh\_time\_scene\_server\_cb\_value\_t::get  
(C++ member), 443
- esp\_ble\_mesh\_time\_scene\_server\_cb\_value\_t::set  
(C++ member), 443
- esp\_ble\_mesh\_time\_scene\_server\_cb\_value\_t::state  
(C++ member), 443
- esp\_ble\_mesh\_time\_scene\_server\_cb\_value\_t::status  
(C++ member), 443
- ESP\_BLE\_MESH\_TIME\_SCENE\_SERVER\_EVT\_MAX  
(C++ enumerator), 462
- ESP\_BLE\_MESH\_TIME\_SCENE\_SERVER\_RECV\_GET\_MSG\_EVT  
(C++ enumerator), 461
- esp\_ble\_mesh\_time\_scene\_server\_recv\_get\_msg\_t  
(C++ union), 442
- esp\_ble\_mesh\_time\_scene\_server\_recv\_get\_msg\_t::sc

- (C++ member), 443
- ESP\_BLE\_MESH\_TIME\_SCENE\_SERVER\_RECV\_SET\_MSG\_EVT (C++ member), 444
- (C++ enumerator), 461
- esp\_ble\_mesh\_time\_scene\_server\_recv\_set\_msg\_t (C++ member), 444
- (C++ union), 443
- esp\_ble\_mesh\_time\_scene\_server\_recv\_set\_msg\_t (C++ member), 443
- esp\_ble\_mesh\_time\_scene\_server\_recv\_set\_msg\_t (C++ member), 443
- esp\_ble\_mesh\_time\_scene\_server\_recv\_set\_msg\_t (C++ member), 443
- esp\_ble\_mesh\_time\_scene\_server\_recv\_set\_msg\_t (C++ member), 443
- esp\_ble\_mesh\_time\_scene\_server\_recv\_set\_msg\_t (C++ member), 443
- esp\_ble\_mesh\_time\_scene\_server\_recv\_set\_msg\_t (C++ member), 443
- esp\_ble\_mesh\_time\_scene\_server\_recv\_set\_msg\_t (C++ member), 443
- esp\_ble\_mesh\_time\_scene\_server\_recv\_set\_msg\_t (C++ member), 443
- esp\_ble\_mesh\_time\_scene\_server\_recv\_set\_msg\_t (C++ member), 443
- esp\_ble\_mesh\_time\_scene\_server\_recv\_set\_msg\_t (C++ member), 443
- ESP\_BLE\_MESH\_TIME\_SCENE\_SERVER\_RECV\_STATUS\_MS\_EVT (C++ member), 448
- (C++ enumerator), 461
- esp\_ble\_mesh\_time\_scene\_server\_recv\_status\_ms\_evt (C++ member), 449
- (C++ union), 443
- esp\_ble\_mesh\_time\_scene\_server\_recv\_status\_ms\_evt (C++ member), 448
- (C++ member), 443
- ESP\_BLE\_MESH\_TIME\_SCENE\_SERVER\_STATE\_CHANGE\_EVT (C++ member), 449
- (C++ enumerator), 461
- esp\_ble\_mesh\_time\_scene\_server\_state\_change\_t (C++ member), 449
- (C++ union), 442
- esp\_ble\_mesh\_time\_scene\_server\_state\_change\_t (C++ member), 449
- (C++ member), 442
- esp\_ble\_mesh\_time\_scene\_server\_state\_change\_t (C++ member), 449
- (C++ member), 442
- esp\_ble\_mesh\_time\_scene\_server\_state\_change\_t (C++ member), 449
- (C++ member), 442
- esp\_ble\_mesh\_time\_scene\_server\_state\_change\_t (C++ member), 449
- (C++ member), 442
- esp\_ble\_mesh\_time\_scene\_server\_state\_change\_t (C++ member), 449
- (C++ member), 442
- esp\_ble\_mesh\_time\_scene\_server\_state\_change\_t (C++ member), 449
- (C++ member), 442
- esp\_ble\_mesh\_time\_set\_t (C++ class), 444
- esp\_ble\_mesh\_time\_set\_t::sub\_second (C++ member), 444
- esp\_ble\_mesh\_time\_set\_t::tai\_seconds (C++ member), 444
- esp\_ble\_mesh\_time\_set\_t::tai\_utc\_delta (C++ member), 444
- esp\_ble\_mesh\_time\_set\_t::time\_authority (C++ member), 444
- esp\_ble\_mesh\_time\_set\_t::time\_zone\_offset (C++ member), 444
- esp\_ble\_mesh\_time\_set\_t::uncertainty (C++ member), 444
- esp\_ble\_mesh\_time\_setup\_srv\_t (C++ class), 449
- esp\_ble\_mesh\_time\_setup\_srv\_t::delete (C++ member), 449
- esp\_ble\_mesh\_time\_setup\_srv\_t::model (C++ member), 449
- esp\_ble\_mesh\_time\_setup\_srv\_t::rsp\_ctrl (C++ member), 449
- esp\_ble\_mesh\_time\_setup\_srv\_t::state (C++ member), 449
- esp\_ble\_mesh\_time\_srv\_t (C++ class), 449
- esp\_ble\_mesh\_time\_srv\_t::model (C++ member), 449
- esp\_ble\_mesh\_time\_srv\_t::rsp\_ctrl (C++ member), 449
- esp\_ble\_mesh\_time\_srv\_t::state (C++ member), 449
- esp\_ble\_mesh\_time\_state\_t (C++ class), 448
- esp\_ble\_mesh\_time\_state\_t::subsecond (C++ member), 448
- esp\_ble\_mesh\_time\_state\_t::tai\_delta\_change (C++ member), 448
- esp\_ble\_mesh\_time\_state\_t::tai\_seconds (C++ member), 448
- esp\_ble\_mesh\_time\_state\_t::tai\_utc\_delta\_curr (C++ member), 448
- esp\_ble\_mesh\_time\_state\_t::tai\_utc\_delta\_new (C++ member), 448
- esp\_ble\_mesh\_time\_state\_t::tai\_zone\_change (C++ member), 448
- esp\_ble\_mesh\_time\_state\_t::time (C++ member), 448
- esp\_ble\_mesh\_time\_state\_t::time\_authority (C++ member), 448
- esp\_ble\_mesh\_time\_state\_t::time\_role (C++ member), 448
- esp\_ble\_mesh\_time\_state\_t::time\_set (C++ member), 448
- esp\_ble\_mesh\_time\_state\_t::time\_zone\_offset\_curr (C++ member), 448
- esp\_ble\_mesh\_time\_state\_t::time\_zone\_offset\_new (C++ member), 448
- esp\_ble\_mesh\_time\_state\_t::uncertainty (C++ member), 448
- esp\_ble\_mesh\_time\_status\_cb\_t (C++ class), 446
- esp\_ble\_mesh\_time\_status\_cb\_t::sub\_second (C++ member), 446
- esp\_ble\_mesh\_time\_status\_cb\_t::tai\_seconds (C++ member), 446
- esp\_ble\_mesh\_time\_status\_cb\_t::tai\_utc\_delta (C++ member), 446
- esp\_ble\_mesh\_time\_status\_cb\_t::time\_authority (C++ member), 446
- esp\_ble\_mesh\_time\_status\_cb\_t::time\_zone\_offset (C++ member), 446
- esp\_ble\_mesh\_time\_status\_cb\_t::uncertainty (C++ member), 446



- esp\_ble\_mesh\_time\_zone\_set\_t (C++ class), 444  
 esp\_ble\_mesh\_time\_zone\_set\_t::tai\_zone\_change (C++ member), 444  
 esp\_ble\_mesh\_time\_zone\_set\_t::time\_zone\_offset (C++ member), 444  
 esp\_ble\_mesh\_time\_zone\_status\_cb\_t (C++ class), 446  
 esp\_ble\_mesh\_time\_zone\_status\_cb\_t::tai\_zone\_change (C++ member), 446  
 esp\_ble\_mesh\_time\_zone\_status\_cb\_t::time\_zone\_offset (C++ member), 446  
 esp\_ble\_mesh\_time\_zone\_status\_cb\_t::time\_zone\_offset\_keys (C++ member), 446  
 ESP\_BLE\_MESH\_TRANSMIT (C 宏), 283  
 ESP\_BLE\_MESH\_TTL\_DEFAULT (C 宏), 282  
 ESP\_BLE\_MESH\_TTL\_MAX (C 宏), 282  
 ESP\_BLE\_MESH\_TWIST (C++ enumerator), 298  
 ESP\_BLE\_MESH\_TYPE\_COMPLETE\_CB (C++ enumerator), 298  
 ESP\_BLE\_MESH\_TYPE\_INTPUT\_CB (C++ enumerator), 298  
 ESP\_BLE\_MESH\_TYPE\_LINK\_CLOSE\_CB (C++ enumerator), 298  
 ESP\_BLE\_MESH\_TYPE\_LINK\_OPEN\_CB (C++ enumerator), 298  
 ESP\_BLE\_MESH\_TYPE\_OUTPUT\_NUM\_CB (C++ enumerator), 297  
 ESP\_BLE\_MESH\_TYPE\_OUTPUT\_STR\_CB (C++ enumerator), 297  
 ESP\_BLE\_MESH\_TYPE\_PROV\_CB (C++ enumerator), 297  
 ESP\_BLE\_MESH\_TYPE\_RESET\_CB (C++ enumerator), 298  
 ESP\_BLE\_MESH\_UNKNOWN\_TAI\_DELTA\_CHANGE (C 宏), 459  
 ESP\_BLE\_MESH\_UNKNOWN\_TAI\_SECONDS (C 宏), 459  
 ESP\_BLE\_MESH\_UNKNOWN\_TAI\_ZONE\_CHANGE (C 宏), 459  
 esp\_ble\_mesh\_unprov\_dev\_add\_t (C++ class), 277  
 esp\_ble\_mesh\_unprov\_dev\_add\_t::addr (C++ member), 277  
 esp\_ble\_mesh\_unprov\_dev\_add\_t::addr\_type (C++ member), 277  
 esp\_ble\_mesh\_unprov\_dev\_add\_t::bearer (C++ member), 277  
 esp\_ble\_mesh\_unprov\_dev\_add\_t::oob\_info (C++ member), 277  
 esp\_ble\_mesh\_unprov\_dev\_add\_t::uuid (C++ member), 277  
 ESP\_BLE\_MESH\_VENDOR\_MODEL (C 宏), 284  
 ESP\_BLE\_MESH\_VIBRATE (C++ enumerator), 298  
 ESP\_BLE\_MESH\_VIBRATION\_ERROR (C 宏), 381  
 ESP\_BLE\_MESH\_VIBRATION\_WARNING (C 宏), 381  
 ESP\_BLE\_ONLY\_ACCEPT\_SPECIFIED\_AUTH\_DISABLE (C 宏), 131  
 ESP\_BLE\_ONLY\_ACCEPT\_SPECIFIED\_AUTH\_ENABLE (C 宏), 131  
 ESP\_BLE\_OOB\_DISABLE (C 宏), 131  
 ESP\_BLE\_OOB\_ENABLE (C 宏), 131  
 esp\_ble\_oob\_req\_reply (C++ function), 119  
 esp\_ble\_passkey\_reply (C++ function), 118  
 esp\_ble\_pcsr\_keys\_t (C++ class), 128  
 esp\_ble\_pcsr\_keys\_t::counter (C++ member), 128  
 esp\_ble\_pcsr\_keys\_t::csr (C++ member), 128  
 esp\_ble\_pcsr\_keys\_t::sec\_level (C++ member), 128  
 esp\_ble\_penc\_keys\_t (C++ class), 128  
 esp\_ble\_penc\_keys\_t::ediv (C++ member), 128  
 esp\_ble\_penc\_keys\_t::key\_size (C++ member), 128  
 esp\_ble\_penc\_keys\_t::ltk (C++ member), 128  
 esp\_ble\_penc\_keys\_t::rand (C++ member), 128  
 esp\_ble\_penc\_keys\_t::sec\_level (C++ member), 128  
 esp\_ble\_pid\_keys\_t (C++ class), 128  
 esp\_ble\_pid\_keys\_t::addr\_type (C++ member), 128  
 esp\_ble\_pid\_keys\_t::irk (C++ member), 128  
 esp\_ble\_pid\_keys\_t::static\_addr (C++ member), 128  
 esp\_ble\_pkt\_data\_length\_params\_t (C++ class), 127  
 esp\_ble\_pkt\_data\_length\_params\_t::rx\_len (C++ member), 128  
 esp\_ble\_pkt\_data\_length\_params\_t::tx\_len (C++ member), 128  
 esp\_ble\_power\_type\_t (C++ enum), 107  
 ESP\_BLE\_PWR\_TYPE\_ADV (C++ enumerator), 107  
 ESP\_BLE\_PWR\_TYPE\_CONN\_HDL0 (C++ enumerator), 107  
 ESP\_BLE\_PWR\_TYPE\_CONN\_HDL1 (C++ enumerator), 107  
 ESP\_BLE\_PWR\_TYPE\_CONN\_HDL2 (C++ enumerator), 107  
 ESP\_BLE\_PWR\_TYPE\_CONN\_HDL3 (C++ enumerator), 107  
 ESP\_BLE\_PWR\_TYPE\_CONN\_HDL4 (C++ enumerator), 107  
 ESP\_BLE\_PWR\_TYPE\_CONN\_HDL5 (C++ enumerator), 107  
 ESP\_BLE\_PWR\_TYPE\_CONN\_HDL6 (C++ enumerator), 107  
 ESP\_BLE\_PWR\_TYPE\_CONN\_HDL7 (C++ enumerator), 107  
 ESP\_BLE\_PWR\_TYPE\_CONN\_HDL8 (C++ enumerator), 107  
 ESP\_BLE\_PWR\_TYPE\_DEFAULT (C++ enumerator), 107

- 107
- ESP\_BLE\_PWR\_TYPE\_NUM (C++ *enumerator*), 107
- ESP\_BLE\_PWR\_TYPE\_SCAN (C++ *enumerator*), 107
- esp\_ble\_remove\_bond\_device (C++ *function*), 118
- esp\_ble\_resolve\_adv\_data (C++ *function*), 116
- ESP\_BLE\_SCA\_100PPM (C++ *enumerator*), 106
- ESP\_BLE\_SCA\_150PPM (C++ *enumerator*), 106
- ESP\_BLE\_SCA\_20PPM (C++ *enumerator*), 107
- ESP\_BLE\_SCA\_250PPM (C++ *enumerator*), 106
- ESP\_BLE\_SCA\_30PPM (C++ *enumerator*), 106
- ESP\_BLE\_SCA\_500PPM (C++ *enumerator*), 106
- ESP\_BLE\_SCA\_50PPM (C++ *enumerator*), 106
- ESP\_BLE\_SCA\_75PPM (C++ *enumerator*), 106
- esp\_ble\_scan\_duplicate\_list\_flush (C++ *function*), 104
- esp\_ble\_scan\_duplicate\_t (C++ *enum*), 137
- esp\_ble\_scan\_filter\_t (C++ *enum*), 137
- ESP\_BLE\_SCAN\_PARAM\_UNDEF (C 宏), 109
- esp\_ble\_scan\_params\_t (C++ *class*), 126
- esp\_ble\_scan\_params\_t::own\_addr\_type (C++ *member*), 126
- esp\_ble\_scan\_params\_t::scan\_duplicate (C++ *member*), 127
- esp\_ble\_scan\_params\_t::scan\_filter\_policy (C++ *member*), 127
- esp\_ble\_scan\_params\_t::scan\_interval (C++ *member*), 127
- esp\_ble\_scan\_params\_t::scan\_type (C++ *member*), 126
- esp\_ble\_scan\_params\_t::scan\_window (C++ *member*), 127
- ESP\_BLE\_SCAN\_RSP\_DATA\_LEN\_MAX (C 宏), 133
- esp\_ble\_scan\_type\_t (C++ *enum*), 137
- esp\_ble\_sec\_act\_t (C++ *enum*), 136
- ESP\_BLE\_SEC\_ENCRYPT (C++ *enumerator*), 136
- ESP\_BLE\_SEC\_ENCRYPT\_MITM (C++ *enumerator*), 136
- ESP\_BLE\_SEC\_ENCRYPT\_NO\_MITM (C++ *enumerator*), 136
- esp\_ble\_sec\_key\_notif\_t (C++ *class*), 129
- esp\_ble\_sec\_key\_notif\_t::bd\_addr (C++ *member*), 129
- esp\_ble\_sec\_key\_notif\_t::passkey (C++ *member*), 129
- esp\_ble\_sec\_req\_t (C++ *class*), 129
- esp\_ble\_sec\_req\_t::bd\_addr (C++ *member*), 129
- esp\_ble\_sec\_t (C++ *union*), 120
- esp\_ble\_sec\_t::auth\_cmpl (C++ *member*), 120
- esp\_ble\_sec\_t::ble\_id\_keys (C++ *member*), 120
- esp\_ble\_sec\_t::ble\_key (C++ *member*), 120
- esp\_ble\_sec\_t::ble\_req (C++ *member*), 120
- esp\_ble\_sec\_t::key\_notif (C++ *member*), 120
- esp\_ble\_set\_encryption (C++ *function*), 118
- ESP\_BLE\_SM\_AUTHEN\_REQ\_MODE (C++ *enumerator*), 136
- ESP\_BLE\_SM\_CLEAR\_STATIC\_PASSKEY (C++ *enumerator*), 137
- ESP\_BLE\_SM\_IOCTLAP\_MODE (C++ *enumerator*), 137
- ESP\_BLE\_SM\_MAX\_KEY\_SIZE (C++ *enumerator*), 137
- ESP\_BLE\_SM\_MAX\_PARAM (C++ *enumerator*), 137
- ESP\_BLE\_SM\_MIN\_KEY\_SIZE (C++ *enumerator*), 137
- ESP\_BLE\_SM\_ONLY\_ACCEPT\_SPECIFIED\_SEC\_AUTH (C++ *enumerator*), 137
- ESP\_BLE\_SM\_OOB\_SUPPORT (C++ *enumerator*), 137
- esp\_ble\_sm\_param\_t (C++ *enum*), 136
- ESP\_BLE\_SM\_PASSKEY (C++ *enumerator*), 136
- ESP\_BLE\_SM\_SET\_INIT\_KEY (C++ *enumerator*), 137
- ESP\_BLE\_SM\_SET\_RSP\_KEY (C++ *enumerator*), 137
- ESP\_BLE\_SM\_SET\_STATIC\_PASSKEY (C++ *enumerator*), 137
- esp\_ble\_tx\_power\_get (C++ *function*), 101
- esp\_ble\_tx\_power\_set (C++ *function*), 101
- ESP\_BLE\_WHITELIST\_ADD (C++ *enumerator*), 138
- ESP\_BLE\_WHITELIST\_REMOVE (C++ *enumerator*), 138
- esp\_ble\_wl\_addr\_type\_t (C++ *enum*), 111
- esp\_ble\_wl\_operation\_t (C++ *enum*), 138
- esp\_blueandroid\_deinit (C++ *function*), 112
- esp\_blueandroid\_disable (C++ *function*), 112
- esp\_blueandroid\_enable (C++ *function*), 111
- esp\_blueandroid\_get\_status (C++ *function*), 111
- esp\_blueandroid\_init (C++ *function*), 112
- ESP\_BLUEDROID\_STATUS\_CHECK (C 宏), 109
- ESP\_BLUEDROID\_STATUS\_ENABLED (C++ *enumerator*), 112
- ESP\_BLUEDROID\_STATUS\_INITIALIZED (C++ *enumerator*), 112
- esp\_blueandroid\_status\_t (C++ *enum*), 112
- ESP\_BLUEDROID\_STATUS\_UNINITIALIZED (C++ *enumerator*), 112
- esp\_blufi\_ap\_record\_t (C++ *class*), 185
- esp\_blufi\_ap\_record\_t::rssi (C++ *member*), 185
- esp\_blufi\_ap\_record\_t::ssid (C++ *member*), 185
- esp\_blufi\_callbacks\_t (C++ *class*), 185
- esp\_blufi\_callbacks\_t::checksum\_func (C++ *member*), 185
- esp\_blufi\_callbacks\_t::decrypt\_func (C++ *member*), 185
- esp\_blufi\_callbacks\_t::encrypt\_func (C++ *member*), 185
- esp\_blufi\_callbacks\_t::event\_cb (C++ *member*), 185



- esp\_blufi\_cb\_param\_t::client\_cert  
 (C++ member), 180
- esp\_blufi\_cb\_param\_t::client\_pkey  
 (C++ member), 180
- esp\_blufi\_cb\_param\_t::connect (C++  
 member), 179
- esp\_blufi\_cb\_param\_t::custom\_data  
 (C++ member), 180
- esp\_blufi\_cb\_param\_t::deinit\_finish  
 (C++ member), 179
- esp\_blufi\_cb\_param\_t::disconnect (C++  
 member), 179
- esp\_blufi\_cb\_param\_t::init\_finish  
 (C++ member), 179
- esp\_blufi\_cb\_param\_t::report\_error  
 (C++ member), 180
- esp\_blufi\_cb\_param\_t::server\_cert  
 (C++ member), 180
- esp\_blufi\_cb\_param\_t::server\_pkey  
 (C++ member), 180
- esp\_blufi\_cb\_param\_t::softap\_auth\_mode  
 (C++ member), 180
- esp\_blufi\_cb\_param\_t::softap\_channel  
 (C++ member), 180
- esp\_blufi\_cb\_param\_t::softap\_max\_conn\_num  
 (C++ member), 180
- esp\_blufi\_cb\_param\_t::softap\_passwd  
 (C++ member), 180
- esp\_blufi\_cb\_param\_t::softap\_ssid  
 (C++ member), 180
- esp\_blufi\_cb\_param\_t::sta\_bssid (C++  
 member), 179
- esp\_blufi\_cb\_param\_t::sta\_passwd (C++  
 member), 180
- esp\_blufi\_cb\_param\_t::sta\_ssid (C++  
 member), 180
- esp\_blufi\_cb\_param\_t::username (C++  
 member), 180
- esp\_blufi\_cb\_param\_t::wifi\_mode (C++  
 member), 179
- ESP\_BLUFI\_CHECKSUM\_ERROR (C++ enumerator),  
 187
- esp\_blufi\_checksum\_func\_t (C++ type), 186
- esp\_blufi\_close (C++ function), 179
- ESP\_BLUFI\_DATA\_FORMAT\_ERROR (C++ enumer-  
 ator), 187
- ESP\_BLUFI\_DECRYPT\_ERROR (C++ enumerator),  
 187
- esp\_blufi\_decrypt\_func\_t (C++ type), 186
- ESP\_BLUFI\_DEINIT\_FAILED (C++ enumerator),  
 187
- ESP\_BLUFI\_DEINIT\_OK (C++ enumerator), 187
- esp\_blufi\_deinit\_state\_t (C++ enum), 187
- ESP\_BLUFI\_DH\_MALLOC\_ERROR (C++ enumera-  
 tor), 187
- ESP\_BLUFI\_DH\_PARAM\_ERROR (C++ enumerator),  
 187
- ESP\_BLUFI\_ENCRYPT\_ERROR (C++ enumerator),  
 187
- esp\_blufi\_encrypt\_func\_t (C++ type), 186
- esp\_blufi\_error\_state\_t (C++ enum), 187
- ESP\_BLUFI\_EVENT\_BLE\_CONNECT (C++ enumer-  
 ator), 186
- ESP\_BLUFI\_EVENT\_BLE\_DISCONNECT (C++  
 enumerator), 186
- esp\_blufi\_event\_cb\_t (C++ type), 185
- ESP\_BLUFI\_EVENT\_DEAUTHENTICATE\_STA  
 (C++ enumerator), 186
- ESP\_BLUFI\_EVENT\_DEINIT\_FINISH (C++ enu-  
 merator), 186
- ESP\_BLUFI\_EVENT\_GET\_WIFI\_LIST (C++ enu-  
 merator), 187
- ESP\_BLUFI\_EVENT\_GET\_WIFI\_STATUS (C++  
 enumerator), 186
- ESP\_BLUFI\_EVENT\_INIT\_FINISH (C++ enumer-  
 ator), 186
- ESP\_BLUFI\_EVENT\_RECV\_CA\_CERT (C++ enu-  
 merator), 187
- ESP\_BLUFI\_EVENT\_RECV\_CLIENT\_CERT (C++  
 enumerator), 187
- ESP\_BLUFI\_EVENT\_RECV\_CLIENT\_PRIV\_KEY  
 (C++ enumerator), 187
- ESP\_BLUFI\_EVENT\_RECV\_CUSTOM\_DATA (C++  
 enumerator), 187
- ESP\_BLUFI\_EVENT\_RECV\_SERVER\_CERT (C++  
 enumerator), 187
- ESP\_BLUFI\_EVENT\_RECV\_SERVER\_PRIV\_KEY  
 (C++ enumerator), 187
- ESP\_BLUFI\_EVENT\_RECV\_SLAVE\_DISCONNECT\_BLE  
 (C++ enumerator), 187
- ESP\_BLUFI\_EVENT\_RECV\_SOFTAP\_AUTH\_MODE  
 (C++ enumerator), 186
- ESP\_BLUFI\_EVENT\_RECV\_SOFTAP\_CHANNEL  
 (C++ enumerator), 187
- ESP\_BLUFI\_EVENT\_RECV\_SOFTAP\_MAX\_CONN\_NUM  
 (C++ enumerator), 186
- ESP\_BLUFI\_EVENT\_RECV\_SOFTAP\_PASSWD  
 (C++ enumerator), 186
- ESP\_BLUFI\_EVENT\_RECV\_SOFTAP\_SSID (C++  
 enumerator), 186
- ESP\_BLUFI\_EVENT\_RECV\_STA\_BSSID (C++  
 enumerator), 186
- ESP\_BLUFI\_EVENT\_RECV\_STA\_PASSWD (C++  
 enumerator), 186
- ESP\_BLUFI\_EVENT\_RECV\_STA\_SSID (C++ enu-  
 merator), 186
- ESP\_BLUFI\_EVENT\_RECV\_USERNAME (C++ enu-  
 merator), 187
- ESP\_BLUFI\_EVENT\_REPORT\_ERROR (C++ enu-  
 merator), 187
- ESP\_BLUFI\_EVENT\_REQ\_CONNECT\_TO\_AP  
 (C++ enumerator), 186
- ESP\_BLUFI\_EVENT\_REQ\_DISCONNECT\_FROM\_AP  
 (C++ enumerator), 186
- ESP\_BLUFI\_EVENT\_SET\_WIFI\_OPMODE (C++  
 enumerator), 186

- esp\_blufi\_extra\_info\_t (C++ class), 184  
 esp\_blufi\_extra\_info\_t::softap\_authmode (C++ member), 185  
 esp\_blufi\_extra\_info\_t::softap\_authmode\_set (C++ member), 185  
 esp\_blufi\_extra\_info\_t::softap\_channel (C++ member), 185  
 esp\_blufi\_extra\_info\_t::softap\_channel\_set (C++ member), 185  
 esp\_blufi\_extra\_info\_t::softap\_max\_conn\_num (C++ member), 185  
 esp\_blufi\_extra\_info\_t::softap\_max\_conn\_num\_set (C++ member), 185  
 esp\_blufi\_extra\_info\_t::softap\_passwd (C++ member), 184  
 esp\_blufi\_extra\_info\_t::softap\_passwd\_len (C++ member), 184  
 esp\_blufi\_extra\_info\_t::softap\_ssid (C++ member), 184  
 esp\_blufi\_extra\_info\_t::softap\_ssid\_len (C++ member), 184  
 esp\_blufi\_extra\_info\_t::sta\_bssid (C++ member), 184  
 esp\_blufi\_extra\_info\_t::sta\_bssid\_set (C++ member), 184  
 esp\_blufi\_extra\_info\_t::sta\_passwd (C++ member), 184  
 esp\_blufi\_extra\_info\_t::sta\_passwd\_len (C++ member), 184  
 esp\_blufi\_extra\_info\_t::sta\_ssid (C++ member), 184  
 esp\_blufi\_extra\_info\_t::sta\_ssid\_len (C++ member), 184  
 esp\_blufi\_get\_version (C++ function), 179  
 ESP\_BLUFI\_INIT\_FAILED (C++ enumerator), 187  
 ESP\_BLUFI\_INIT\_OK (C++ enumerator), 187  
 ESP\_BLUFI\_INIT\_SECURITY\_ERROR (C++ enumerator), 187  
 esp\_blufi\_init\_state\_t (C++ enum), 187  
 ESP\_BLUFI\_MAKE\_PUBLIC\_ERROR (C++ enumerator), 187  
 esp\_blufi\_negotiate\_data\_handler\_t (C++ type), 185  
 esp\_blufi\_profile\_deinit (C++ function), 178  
 esp\_blufi\_profile\_init (C++ function), 178  
 ESP\_BLUFI\_READ\_PARAM\_ERROR (C++ enumerator), 187  
 esp\_blufi\_register\_callbacks (C++ function), 178  
 esp\_blufi\_send\_custom\_data (C++ function), 179  
 esp\_blufi\_send\_error\_info (C++ function), 179  
 esp\_blufi\_send\_wifi\_conn\_report (C++ function), 178  
 esp\_blufi\_send\_wifi\_list (C++ function), 179  
 ESP\_BLUFI\_SEQUENCE\_ERROR (C++ enumerator), 187  
 ESP\_BLUFI\_STA\_CONN\_FAIL (C++ enumerator), 187  
 esp\_blufi\_sta\_conn\_state\_t (C++ enum), 187  
 ESP\_BLUFI\_STA\_CONN\_SUCCESS (C++ enumerator), 187  
 esp\_bredr\_sco\_datapath\_set (C++ function), 102  
 esp\_bredr\_tx\_power\_get (C++ function), 102  
 esp\_bredr\_tx\_power\_set (C++ function), 101  
 ESP\_BT\_CLR\_COD\_SERVICE\_CLASS (C++ enumerator), 199  
 ESP\_BT\_COD\_FORMAT\_TYPE\_1 (C 宏), 198  
 ESP\_BT\_COD\_FORMAT\_TYPE\_BIT\_MASK (C 宏), 198  
 ESP\_BT\_COD\_FORMAT\_TYPE\_BIT\_OFFSET (C 宏), 198  
 ESP\_BT\_COD\_MAJOR\_DEV\_AV (C++ enumerator), 200  
 ESP\_BT\_COD\_MAJOR\_DEV\_BIT\_MASK (C 宏), 198  
 ESP\_BT\_COD\_MAJOR\_DEV\_BIT\_OFFSET (C 宏), 198  
 ESP\_BT\_COD\_MAJOR\_DEV\_COMPUTER (C++ enumerator), 200  
 ESP\_BT\_COD\_MAJOR\_DEV\_HEALTH (C++ enumerator), 200  
 ESP\_BT\_COD\_MAJOR\_DEV\_IMAGING (C++ enumerator), 200  
 ESP\_BT\_COD\_MAJOR\_DEV\_LAN\_NAP (C++ enumerator), 200  
 ESP\_BT\_COD\_MAJOR\_DEV\_MISC (C++ enumerator), 200  
 ESP\_BT\_COD\_MAJOR\_DEV\_PERIPHERAL (C++ enumerator), 200  
 ESP\_BT\_COD\_MAJOR\_DEV\_PHONE (C++ enumerator), 200  
 esp\_bt\_cod\_major\_dev\_t (C++ enum), 200  
 ESP\_BT\_COD\_MAJOR\_DEV\_TOY (C++ enumerator), 200  
 ESP\_BT\_COD\_MAJOR\_DEV\_UNCATEGORIZED (C++ enumerator), 200  
 ESP\_BT\_COD\_MAJOR\_DEV\_WEARABLE (C++ enumerator), 200  
 ESP\_BT\_COD\_MINOR\_DEV\_BIT\_MASK (C 宏), 198  
 ESP\_BT\_COD\_MINOR\_DEV\_BIT\_OFFSET (C 宏), 198  
 esp\_bt\_cod\_mode\_t (C++ enum), 198  
 ESP\_BT\_COD\_SRVC\_AUDIO (C++ enumerator), 200  
 ESP\_BT\_COD\_SRVC\_BIT\_MASK (C 宏), 198  
 ESP\_BT\_COD\_SRVC\_BIT\_OFFSET (C 宏), 198  
 ESP\_BT\_COD\_SRVC\_CAPTURING (C++ enumerator), 200  
 ESP\_BT\_COD\_SRVC\_INFORMATION (C++ enumerator), 200  
 ESP\_BT\_COD\_SRVC\_LMTD\_DISCOVER (C++ enumerator), 199

- ESP\_BT\_COD\_SRVC\_NETWORKING (C++ *enumerator*), 199
- ESP\_BT\_COD\_SRVC\_NONE (C++ *enumerator*), 199
- ESP\_BT\_COD\_SRVC\_OBJ\_TRANSFER (C++ *enumerator*), 200
- ESP\_BT\_COD\_SRVC\_POSITIONING (C++ *enumerator*), 199
- ESP\_BT\_COD\_SRVC\_RENDERING (C++ *enumerator*), 200
- esp\_bt\_cod\_srvc\_t (C++ *enum*), 199
- ESP\_BT\_COD\_SRVC\_TELEPHONY (C++ *enumerator*), 200
- esp\_bt\_cod\_t (C++ *class*), 196
- esp\_bt\_cod\_t::major (C++ *member*), 196
- esp\_bt\_cod\_t::minor (C++ *member*), 196
- esp\_bt\_cod\_t::reserved\_2 (C++ *member*), 196
- esp\_bt\_cod\_t::reserved\_8 (C++ *member*), 196
- esp\_bt\_cod\_t::service (C++ *member*), 196
- ESP\_BT\_CONNECTABLE (C++ *enumerator*), 199
- esp\_bt\_connection\_mode\_t (C++ *enum*), 199
- ESP\_BT\_CONTROLLER\_CONFIG\_MAGIC\_VAL (C++ *宏*), 106
- esp\_bt\_controller\_config\_t (C++ *class*), 105
- esp\_bt\_controller\_config\_t::auto\_latency (C++ *member*), 105
- esp\_bt\_controller\_config\_t::ble\_max\_conn (C++ *member*), 105
- esp\_bt\_controller\_config\_t::ble\_sca (C++ *member*), 105
- esp\_bt\_controller\_config\_t::bt\_legacy\_adv (C++ *member*), 105
- esp\_bt\_controller\_config\_t::bt\_max\_acl\_entries (C++ *member*), 105
- esp\_bt\_controller\_config\_t::bt\_max\_sync\_conn (C++ *member*), 105
- esp\_bt\_controller\_config\_t::bt\_sco\_datapath (C++ *member*), 105
- esp\_bt\_controller\_config\_t::controller\_debug\_flag (C++ *member*), 105
- esp\_bt\_controller\_config\_t::controller\_duplication (C++ *member*), 105
- esp\_bt\_controller\_config\_t::controller\_stack\_size (C++ *member*), 196
- esp\_bt\_controller\_config\_t::hci\_uart\_baudrate (C++ *member*), 105
- esp\_bt\_controller\_config\_t::hci\_uart\_n (C++ *member*), 105
- esp\_bt\_controller\_config\_t::magic (C++ *member*), 105
- esp\_bt\_controller\_config\_t::mesh\_adv\_size (C++ *member*), 105
- esp\_bt\_controller\_config\_t::mode (C++ *member*), 105
- esp\_bt\_controller\_config\_t::normal\_adv\_size (C++ *member*), 105
- esp\_bt\_controller\_config\_t::pcm\_polar (C++ *member*), 105
- esp\_bt\_controller\_config\_t::pcm\_role (C++ *member*), 105
- esp\_bt\_controller\_config\_t::scan\_duplicate\_mode (C++ *member*), 105
- esp\_bt\_controller\_config\_t::scan\_duplicate\_type (C++ *member*), 105
- esp\_bt\_controller\_config\_t::send\_adv\_reserved\_size (C++ *member*), 105
- esp\_bt\_controller\_deinit (C++ *function*), 102
- esp\_bt\_controller\_disable (C++ *function*), 102
- esp\_bt\_controller\_enable (C++ *function*), 102
- esp\_bt\_controller\_get\_status (C++ *function*), 102
- esp\_bt\_controller\_init (C++ *function*), 102
- esp\_bt\_controller\_mem\_release (C++ *function*), 103
- ESP\_BT\_CONTROLLER\_STATUS\_ENABLED (C++ *enumerator*), 107
- ESP\_BT\_CONTROLLER\_STATUS\_IDLE (C++ *enumerator*), 107
- ESP\_BT\_CONTROLLER\_STATUS\_INITED (C++ *enumerator*), 107
- ESP\_BT\_CONTROLLER\_STATUS\_NUM (C++ *enumerator*), 107
- esp\_bt\_controller\_status\_t (C++ *enum*), 107
- esp\_bt\_dev\_get\_address (C++ *function*), 112
- esp\_bt\_dev\_set\_device\_name (C++ *function*), 113
- esp\_bt\_dev\_type\_t (C++ *enum*), 111
- ESP\_BT\_DEVICE\_TYPE\_BLE (C++ *enumerator*), 111
- ESP\_BT\_DEVICE\_TYPE\_BREDR (C++ *enumerator*), 111
- ESP\_BT\_DEVICE\_TYPE\_DUMO (C++ *enumerator*), 111
- esp\_bt\_discovery\_mode\_t (C++ *enum*), 199
- esp\_bt\_duplicate\_exceptional\_subcode\_type\_t (C++ *enum*), 138
- esp\_bt\_stack\_data\_t (C++ *class*), 196
- esp\_bt\_eir\_data\_t::fec\_required (C++ *member*), 196
- esp\_bt\_eir\_data\_t::flag (C++ *member*), 196
- esp\_bt\_eir\_data\_t::include\_txpower (C++ *member*), 196
- esp\_bt\_eir\_data\_t::include\_uuid (C++ *member*), 196
- esp\_bt\_eir\_data\_t::manufacturer\_len (C++ *member*), 196
- esp\_bt\_eir\_data\_t::p\_manufacturer\_data (C++ *member*), 196
- esp\_bt\_eir\_data\_t::p\_url (C++ *member*), 196

- esp\_bt\_eir\_data\_t::url\_len (C++ member), 196
- ESP\_BT\_EIR\_FLAG\_BREDR\_NOT\_SPT (C 宏), 197
- ESP\_BT\_EIR\_FLAG\_DMT\_CONTROLLER\_SPT (C 宏), 197
- ESP\_BT\_EIR\_FLAG\_DMT\_HOST\_SPT (C 宏), 197
- ESP\_BT\_EIR\_FLAG\_GEN\_DISC (C 宏), 197
- ESP\_BT\_EIR\_FLAG\_LIMIT\_DISC (C 宏), 197
- ESP\_BT\_EIR\_MAX\_LEN (C 宏), 197
- ESP\_BT\_EIR\_TYPE\_CMPL\_128BITS\_UUID (C 宏), 197
- ESP\_BT\_EIR\_TYPE\_CMPL\_16BITS\_UUID (C 宏), 197
- ESP\_BT\_EIR\_TYPE\_CMPL\_32BITS\_UUID (C 宏), 197
- ESP\_BT\_EIR\_TYPE\_CMPL\_LOCAL\_NAME (C 宏), 197
- ESP\_BT\_EIR\_TYPE\_FLAGS (C 宏), 197
- ESP\_BT\_EIR\_TYPE\_INCMPL\_128BITS\_UUID (C 宏), 197
- ESP\_BT\_EIR\_TYPE\_INCMPL\_16BITS\_UUID (C 宏), 197
- ESP\_BT\_EIR\_TYPE\_INCMPL\_32BITS\_UUID (C 宏), 197
- ESP\_BT\_EIR\_TYPE\_MANU\_SPECIFIC (C 宏), 197
- ESP\_BT\_EIR\_TYPE\_MAX\_NUM (C 宏), 197
- ESP\_BT\_EIR\_TYPE\_SHORT\_LOCAL\_NAME (C 宏), 197
- esp\_bt\_eir\_type\_t (C++ type), 198
- ESP\_BT\_EIR\_TYPE\_TX\_POWER\_LEVEL (C 宏), 197
- ESP\_BT\_EIR\_TYPE\_URL (C 宏), 197
- ESP\_BT\_GAP\_AUTH\_CMPL\_EVT (C++ enumerator), 201
- esp\_bt\_gap\_cancel\_discovery (C++ function), 189
- esp\_bt\_gap\_cb\_event\_t (C++ enum), 201
- esp\_bt\_gap\_cb\_param\_t (C++ union), 192
- esp\_bt\_gap\_cb\_param\_t::auth\_cmpl (C++ member), 192
- esp\_bt\_gap\_cb\_param\_t::auth\_cmpl\_param (C++ class), 193
- esp\_bt\_gap\_cb\_param\_t::auth\_cmpl\_param::bda (C++ member), 193
- esp\_bt\_gap\_cb\_param\_t::auth\_cmpl\_param::event\_name (C++ member), 193
- esp\_bt\_gap\_cb\_param\_t::auth\_cmpl\_param::status (C++ member), 193
- esp\_bt\_gap\_cb\_param\_t::bt\_remove\_bond\_data (C++ class), 193
- esp\_bt\_gap\_cb\_param\_t::bt\_remove\_bond\_data::bda (C++ member), 193
- esp\_bt\_gap\_cb\_param\_t::bt\_remove\_bond\_data::status (C++ member), 193
- esp\_bt\_gap\_cb\_param\_t::cfm\_req (C++ member), 192
- esp\_bt\_gap\_cb\_param\_t::cfm\_req\_param (C++ class), 193
- esp\_bt\_gap\_cb\_param\_t::cfm\_req\_param::bda (C++ member), 193
- esp\_bt\_gap\_cb\_param\_t::cfm\_req\_param::num\_val (C++ member), 193
- esp\_bt\_gap\_cb\_param\_t::config\_eir\_data (C++ member), 192
- esp\_bt\_gap\_cb\_param\_t::config\_eir\_data\_param (C++ class), 193
- esp\_bt\_gap\_cb\_param\_t::config\_eir\_data\_param::eir\_data (C++ member), 194
- esp\_bt\_gap\_cb\_param\_t::config\_eir\_data\_param::eir\_data\_param (C++ member), 193
- esp\_bt\_gap\_cb\_param\_t::config\_eir\_data\_param::status (C++ member), 193
- esp\_bt\_gap\_cb\_param\_t::disc\_res (C++ member), 192
- esp\_bt\_gap\_cb\_param\_t::disc\_res\_param (C++ class), 194
- esp\_bt\_gap\_cb\_param\_t::disc\_res\_param::bda (C++ member), 194
- esp\_bt\_gap\_cb\_param\_t::disc\_res\_param::num\_prop (C++ member), 194
- esp\_bt\_gap\_cb\_param\_t::disc\_res\_param::prop (C++ member), 194
- esp\_bt\_gap\_cb\_param\_t::disc\_st\_chg (C++ member), 192
- esp\_bt\_gap\_cb\_param\_t::disc\_state\_changed\_param (C++ class), 194
- esp\_bt\_gap\_cb\_param\_t::disc\_state\_changed\_param::bda (C++ member), 194
- esp\_bt\_gap\_cb\_param\_t::key\_notif (C++ member), 193
- esp\_bt\_gap\_cb\_param\_t::key\_notif\_param (C++ class), 194
- esp\_bt\_gap\_cb\_param\_t::key\_notif\_param::bda (C++ member), 194
- esp\_bt\_gap\_cb\_param\_t::key\_notif\_param::passkey (C++ member), 194
- esp\_bt\_gap\_cb\_param\_t::key\_req (C++ member), 193
- esp\_bt\_gap\_cb\_param\_t::key\_req\_param (C++ class), 194
- esp\_bt\_gap\_cb\_param\_t::key\_req\_param::bda (C++ member), 194
- esp\_bt\_gap\_cb\_param\_t::pin\_req (C++ member), 192
- esp\_bt\_gap\_cb\_param\_t::pin\_req\_param (C++ class), 194
- esp\_bt\_gap\_cb\_param\_t::pin\_req\_param::bda (C++ member), 194
- esp\_bt\_gap\_cb\_param\_t::pin\_req\_param::min\_16\_digits (C++ member), 194
- esp\_bt\_gap\_cb\_param\_t::qos\_cmpl (C++ member), 193
- esp\_bt\_gap\_cb\_param\_t::qos\_cmpl\_param (C++ class), 194
- esp\_bt\_gap\_cb\_param\_t::qos\_cmpl\_param::bda (C++ member), 195

- esp\_bt\_gap\_cb\_param\_t::qos\_cmpl\_param::stat (C++ member), 195  
 esp\_bt\_gap\_cb\_param\_t::qos\_cmpl\_param::ESP\_BT\_GAP\_DISC\_RES\_EVT (C++ enumerator), (C++ member), 195  
 esp\_bt\_gap\_cb\_param\_t::read\_rssi\_delta ESP\_BT\_GAP\_DISC\_STATE\_CHANGED\_EVT (C++ member), 192 (C++ enumerator), 201  
 esp\_bt\_gap\_cb\_param\_t::read\_rssi\_delta ESP\_BT\_GAP\_DISCOVERY\_STARTED (C++ enum- (C++ class), 195 (C++ member), 195 (C++ member), 195  
 esp\_bt\_gap\_cb\_param\_t::read\_rssi\_delta ESP\_BT\_GAP\_DISCOVERY\_STOPPED (C++ enum- (C++ member), 195 (C++ member), 195  
 esp\_bt\_gap\_cb\_param\_t::read\_rssi\_delta ESP\_BT\_GAP\_EIR\_DATA\_LEN (C 宏), 197  
 esp\_bt\_gap\_cb\_param\_t::read\_rssi\_delta ESP\_BT\_GAP\_EVT\_MAX (C++ enumerator), 201  
 esp\_bt\_gap\_cb\_param\_t::remove\_bond\_dev esp\_bt\_gap\_get\_bond\_device\_list (C++ (C++ member), 193 (C++ member), 193 function), 191  
 esp\_bt\_gap\_cb\_param\_t::rmt\_srvc\_rec esp\_bt\_gap\_get\_bond\_device\_num (C++ (C++ member), 192 (C++ member), 192 function), 191  
 esp\_bt\_gap\_cb\_param\_t::rmt\_srvc\_rec\_param esp\_bt\_gap\_get\_cod (C++ function), 190  
 esp\_bt\_gap\_cb\_param\_t::rmt\_srvc\_rec\_param::bd function), 188  
 esp\_bt\_gap\_cb\_param\_t::rmt\_srvc\_rec\_param::st function), 188  
 esp\_bt\_gap\_cb\_param\_t::rmt\_srvc\_rec\_param::stat function), 188  
 esp\_bt\_gap\_cb\_param\_t::rmt\_srvc\_rec\_param::uid function), 188  
 esp\_bt\_gap\_cb\_param\_t::rmt\_srvc\_rec\_param::stat ESP\_BT\_GAP\_KEY\_NOTIF\_EVT (C++ enumerator), (C++ member), 195 201  
 esp\_bt\_gap\_cb\_param\_t::rmt\_srvc\_rec\_param::stat ESP\_BT\_GAP\_KEY\_REQ\_EVT (C++ enumerator), (C++ member), 195 201  
 esp\_bt\_gap\_cb\_param\_t::rmt\_srvc\_rec\_param::uid ESP\_BT\_GAP\_MAX\_BDNAME\_LEN (C 宏), 197  
 esp\_bt\_gap\_cb\_t (C++ type), 198  
 ESP\_BT\_GAP\_CFM\_REQ\_EVT (C++ enumerator), ESP\_BT\_GAP\_MAX\_INQ\_LEN (C 宏), 198  
 201  
 ESP\_BT\_GAP\_MIN\_INQ\_LEN (C 宏), 198  
 esp\_bt\_gap\_config\_eir\_data (C++ function), 190  
 esp\_bt\_gap\_pin\_reply (C++ function), 191  
 ESP\_BT\_GAP\_PIN\_REQ\_EVT (C++ enumerator), 201  
 ESP\_BT\_GAP\_QOS\_CMPL\_EVT (C++ enumerator), 201  
 ESP\_BT\_GAP\_CONFIG\_EIR\_DATA\_EVT (C++ enumerator), 201  
 esp\_bt\_gap\_read\_rssi\_delta (C++ function), 190  
 ESP\_BT\_GAP\_DEV\_PROP\_BDNAME (C++ enumera- ESP\_BT\_GAP\_READ\_RSSI\_DELTA\_EVT (C++ tor), 199 (C++ member), 199 (C++ member), 199  
 ESP\_BT\_GAP\_DEV\_PROP\_COD (C++ enumerator), 199  
 ESP\_BT\_GAP\_DEV\_PROP\_EIR (C++ enumerator), 199  
 ESP\_BT\_GAP\_DEV\_PROP\_RSSI (C++ enumerator), 199  
 esp\_bt\_gap\_dev\_prop\_t (C++ class), 196  
 esp\_bt\_gap\_dev\_prop\_t::len (C++ member), 196  
 esp\_bt\_gap\_dev\_prop\_t::type (C++ mem- member), 196  
 esp\_bt\_gap\_dev\_prop\_t::val (C++ member), 196  
 ESP\_BT\_GAP\_REMOVE\_BOND\_DEV\_COMPLETE\_EVT (C++ enumerator), 201  
 esp\_bt\_gap\_remove\_bond\_device (C++ func- tion), 190  
 esp\_bt\_gap\_resolve\_eir\_data (C++ func- tion), 190  
 ESP\_BT\_GAP\_RMT\_SRVC\_REC\_EVT (C++ enum- ator), 201  
 ESP\_BT\_GAP\_RMT\_SRVC\_EVT (C++ enumerator), 201



- 201
- ESP\_BT\_GAP\_RSSI\_HIGH\_THRLD (C 宏), 197
- ESP\_BT\_GAP\_RSSI\_LOW\_THRLD (C 宏), 197
- esp\_bt\_gap\_set\_cod (C++ function), 190
- esp\_bt\_gap\_set\_pin (C++ function), 191
- esp\_bt\_gap\_set\_qos (C++ function), 192
- esp\_bt\_gap\_set\_scan\_mode (C++ function), 188
- esp\_bt\_gap\_set\_security\_param (C++ function), 191
- esp\_bt\_gap\_ssp\_confirm\_reply (C++ function), 192
- esp\_bt\_gap\_ssp\_passkey\_reply (C++ function), 191
- esp\_bt\_gap\_start\_discovery (C++ function), 189
- ESP\_BT\_GENERAL\_DISCOVERABLE (C++ enumerator), 199
- esp\_bt\_get\_mac (C++ function), 102
- ESP\_BT\_HF\_CLIENT\_NUMBER\_LEN (C 宏), 245
- ESP\_BT\_HF\_CLIENT\_OPERATOR\_NAME\_LEN (C 宏), 245
- ESP\_BT\_INIT\_COD (C++ enumerator), 199
- ESP\_BT\_INQ\_MODE\_GENERAL\_INQUIRY (C++ enumerator), 201
- ESP\_BT\_INQ\_MODE\_LIMITED\_INQUIRY (C++ enumerator), 201
- esp\_bt\_inq\_mode\_t (C++ enum), 201
- ESP\_BT\_IO\_CAP\_IN (C 宏), 198
- ESP\_BT\_IO\_CAP\_IO (C 宏), 198
- ESP\_BT\_IO\_CAP\_NONE (C 宏), 198
- ESP\_BT\_IO\_CAP\_OUT (C 宏), 197
- esp\_bt\_io\_cap\_t (C++ type), 198
- ESP\_BT\_LIMITED\_DISCOVERABLE (C++ enumerator), 199
- esp\_bt\_mem\_release (C++ function), 103
- ESP\_BT\_MODE\_BLE (C++ enumerator), 106
- ESP\_BT\_MODE\_BTDM (C++ enumerator), 106
- ESP\_BT\_MODE\_CLASSIC\_BT (C++ enumerator), 106
- ESP\_BT\_MODE\_IDLE (C++ enumerator), 106
- esp\_bt\_mode\_t (C++ enum), 106
- ESP\_BT\_NON\_CONNECTABLE (C++ enumerator), 199
- ESP\_BT\_NON\_DISCOVERABLE (C++ enumerator), 199
- ESP\_BT\_OCTET16\_LEN (C 宏), 109
- esp\_bt\_octet16\_t (C++ type), 110
- ESP\_BT\_OCTET8\_LEN (C 宏), 109
- esp\_bt\_octet8\_t (C++ type), 110
- ESP\_BT\_PIN\_CODE\_LEN (C 宏), 197
- esp\_bt\_pin\_code\_t (C++ type), 198
- ESP\_BT\_PIN\_TYPE\_FIXED (C++ enumerator), 200
- esp\_bt\_pin\_type\_t (C++ enum), 200
- ESP\_BT\_PIN\_TYPE\_VARIABLE (C++ enumerator), 200
- ESP\_BT\_SET\_COD\_ALL (C++ enumerator), 199
- ESP\_BT\_SET\_COD\_MAJOR\_MINOR (C++ enumerator), 198
- ESP\_BT\_SET\_COD\_SERVICE\_CLASS (C++ enumerator), 199
- esp\_bt\_sleep\_disable (C++ function), 104
- esp\_bt\_sleep\_enable (C++ function), 104
- ESP\_BT\_SP\_IOCAP\_MODE (C++ enumerator), 200
- esp\_bt\_sp\_param\_t (C++ enum), 200
- ESP\_BT\_STATUS\_AUTH\_FAILURE (C++ enumerator), 110
- ESP\_BT\_STATUS\_AUTH\_REJECTED (C++ enumerator), 110
- ESP\_BT\_STATUS\_BUSY (C++ enumerator), 110
- ESP\_BT\_STATUS\_CONTROL\_LE\_DATA\_LEN\_UNSUPPORTED (C++ enumerator), 111
- ESP\_BT\_STATUS\_DONE (C++ enumerator), 110
- ESP\_BT\_STATUS\_EIR\_TOO\_LARGE (C++ enumerator), 111
- ESP\_BT\_STATUS\_ERR\_ILLEGAL\_PARAMETER\_FMT (C++ enumerator), 111
- ESP\_BT\_STATUS\_FAIL (C++ enumerator), 110
- ESP\_BT\_STATUS\_INVALID\_STATIC\_RAND\_ADDR (C++ enumerator), 110
- ESP\_BT\_STATUS\_MEMORY\_FULL (C++ enumerator), 111
- ESP\_BT\_STATUS\_NOMEM (C++ enumerator), 110
- ESP\_BT\_STATUS\_NOT\_READY (C++ enumerator), 110
- ESP\_BT\_STATUS\_PARAM\_OUT\_OF\_RANGE (C++ enumerator), 110
- ESP\_BT\_STATUS\_PARM\_INVALID (C++ enumerator), 110
- ESP\_BT\_STATUS\_PEER\_LE\_DATA\_LEN\_UNSUPPORTED (C++ enumerator), 111
- ESP\_BT\_STATUS\_PENDING (C++ enumerator), 110
- ESP\_BT\_STATUS\_RMT\_DEV\_DOWN (C++ enumerator), 110
- ESP\_BT\_STATUS\_SUCCESS (C++ enumerator), 110
- esp\_bt\_status\_t (C++ enum), 110
- ESP\_BT\_STATUS\_TIMEOUT (C++ enumerator), 111
- ESP\_BT\_STATUS\_UNACCEPT\_CONN\_INTERVAL (C++ enumerator), 110
- ESP\_BT\_STATUS\_UNHANDLED (C++ enumerator), 110
- ESP\_BT\_STATUS\_UNSUPPORTED (C++ enumerator), 110
- esp\_bt\_uuid\_t (C++ class), 109
- esp\_bt\_uuid\_t::len (C++ member), 109
- esp\_bt\_uuid\_t::uuid (C++ member), 109
- ESP\_CHIP\_ID\_ESP32 (C++ enumerator), 1132
- ESP\_CHIP\_ID\_INVALID (C++ enumerator), 1132
- esp\_chip\_id\_t (C++ enum), 1132
- esp\_chip\_info (C++ function), 1138
- esp\_chip\_info\_t (C++ class), 1138
- esp\_chip\_info\_t::cores (C++ member), 1138
- esp\_chip\_info\_t::features (C++ member), 1138
- esp\_chip\_info\_t::model (C++ member), 1138

- esp\_chip\_info\_t::revision (C++ member), 1138
- esp\_chip\_model\_t (C++ enum), 1139
- esp\_deep\_sleep (C++ function), 1110
- esp\_deep\_sleep\_disable\_rom\_logging (C++ function), 1111
- esp\_deep\_sleep\_start (C++ function), 1110
- esp\_deep\_sleep\_wake\_stub\_fn\_t (C++ type), 1111
- ESP\_DEFAULT\_GATT\_IF (C 宏), 109
- esp\_default\_wake\_deep\_sleep (C++ function), 1111
- esp\_deregister\_freertos\_idle\_hook (C++ function), 1032
- esp\_deregister\_freertos\_idle\_hook\_for\_cpu (C++ function), 1032
- esp\_deregister\_freertos\_tick\_hook (C++ function), 1032
- esp\_deregister\_freertos\_tick\_hook\_for\_cpu (C++ function), 1032
- esp\_derive\_local\_mac (C++ function), 1138
- esp\_duplicate\_info\_t (C++ type), 133
- esp\_duplicate\_scan\_exceptional\_list\_types\_t (C++ enum), 139
- ESP\_EARLY\_LOGD (C 宏), 1081
- ESP\_EARLY\_LOGE (C 宏), 1081
- ESP\_EARLY\_LOGI (C 宏), 1081
- ESP\_EARLY\_LOGV (C 宏), 1081
- ESP\_EARLY\_LOGW (C 宏), 1081
- esp\_efuse\_apply\_34\_encoding (C++ function), 1072
- esp\_efuse\_block\_t (C++ enum), 1072
- esp\_efuse\_burn\_new\_values (C++ function), 1072
- esp\_efuse\_check\_secure\_version (C++ function), 1072
- esp\_efuse\_coding\_scheme\_t (C++ enum), 1073
- esp\_efuse\_desc\_t (C++ class), 1072
- esp\_efuse\_desc\_t::bit\_count (C++ member), 1072
- esp\_efuse\_desc\_t::bit\_start (C++ member), 1072
- esp\_efuse\_desc\_t::efuse\_block (C++ member), 1072
- esp\_efuse\_disable\_basic\_rom\_console (C++ function), 1072
- esp\_efuse\_get\_chip\_ver (C++ function), 1071
- esp\_efuse\_get\_coding\_scheme (C++ function), 1071
- esp\_efuse\_get\_field\_size (C++ function), 1070
- esp\_efuse\_get\_pkg\_ver (C++ function), 1071
- esp\_efuse\_init (C++ function), 1072
- esp\_efuse\_mac\_get\_custom (C++ function), 1137
- esp\_efuse\_mac\_get\_default (C++ function), 1137
- esp\_efuse\_read\_block (C++ function), 1071
- esp\_efuse\_read\_field\_blob (C++ function), 1069
- esp\_efuse\_read\_field\_cnt (C++ function), 1069
- esp\_efuse\_read\_reg (C++ function), 1070
- esp\_efuse\_read\_secure\_version (C++ function), 1072
- esp\_efuse\_reset (C++ function), 1072
- esp\_efuse\_set\_read\_protect (C++ function), 1070
- esp\_efuse\_set\_write\_protect (C++ function), 1070
- esp\_efuse\_update\_secure\_version (C++ function), 1072
- esp\_efuse\_write\_block (C++ function), 1071
- esp\_efuse\_write\_field\_blob (C++ function), 1069
- esp\_efuse\_write\_field\_cnt (C++ function), 1070
- esp\_efuse\_write\_random\_key (C++ function), 1072
- esp\_efuse\_write\_reg (C++ function), 1071
- ESP\_ERR\_CODING (C 宏), 1072
- ESP\_ERR\_EFUSE (C 宏), 1072
- ESP\_ERR\_EFUSE\_CNT\_IS\_FULL (C 宏), 1072
- ESP\_ERR\_EFUSE\_REPEATED\_PROG (C 宏), 1072
- ESP\_ERR\_ESP\_TLS\_BASE (C 宏), 769
- ESP\_ERR\_ESP\_TLS\_CANNOT\_CREATE\_SOCKET (C 宏), 769
- ESP\_ERR\_ESP\_TLS\_CANNOT\_RESOLVE\_HOSTNAME (C 宏), 769
- ESP\_ERR\_ESP\_TLS\_CONNECTION\_TIMEOUT (C 宏), 769
- ESP\_ERR\_ESP\_TLS\_FAILED\_CONNECT\_TO\_HOST (C 宏), 769
- ESP\_ERR\_ESP\_TLS\_SOCKET\_SETOPT\_FAILED (C 宏), 769
- ESP\_ERR\_ESP\_TLS\_UNSUPPORTED\_PROTOCOL\_FAMILY (C 宏), 769
- ESP\_ERR\_ESPNOW\_ARG (C 宏), 502
- ESP\_ERR\_ESPNOW\_BASE (C 宏), 502
- ESP\_ERR\_ESPNOW\_EXIST (C 宏), 502
- ESP\_ERR\_ESPNOW\_FULL (C 宏), 502
- ESP\_ERR\_ESPNOW\_IF (C 宏), 502
- ESP\_ERR\_ESPNOW\_INTERNAL (C 宏), 502
- ESP\_ERR\_ESPNOW\_NO\_MEM (C 宏), 502
- ESP\_ERR\_ESPNOW\_NOT\_FOUND (C 宏), 502
- ESP\_ERR\_ESPNOW\_NOT\_INIT (C 宏), 502
- ESP\_ERR\_FLASH\_BASE (C 宏), 1128
- ESP\_ERR\_HTTP\_BASE (C 宏), 780
- ESP\_ERR\_HTTP\_CONNECT (C 宏), 780
- ESP\_ERR\_HTTP\_CONNECTING (C 宏), 780
- ESP\_ERR\_HTTP\_EAGAIN (C 宏), 780
- ESP\_ERR\_HTTP\_FETCH\_HEADER (C 宏), 780
- ESP\_ERR\_HTTP\_INVALID\_TRANSPORT (C 宏), 780
- ESP\_ERR\_HTTP\_MAX\_REDIRECT (C 宏), 780

- ESP\_ERR\_HTTP\_WRITE\_DATA (C 宏), 780
- ESP\_ERR\_HTTPD\_ALLOC\_MEM (C 宏), 804
- ESP\_ERR\_HTTPD\_BASE (C 宏), 804
- ESP\_ERR\_HTTPD\_HANDLER\_EXISTS (C 宏), 804
- ESP\_ERR\_HTTPD\_HANDLERS\_FULL (C 宏), 804
- ESP\_ERR\_HTTPD\_INVALID\_REQ (C 宏), 804
- ESP\_ERR\_HTTPD\_RESP\_HDR (C 宏), 804
- ESP\_ERR\_HTTPD\_RESP\_SEND (C 宏), 804
- ESP\_ERR\_HTTPD\_RESULT\_TRUNC (C 宏), 804
- ESP\_ERR\_HTTPD\_TASK (C 宏), 804
- ESP\_ERR\_HTTPS\_OTA\_BASE (C 宏), 1124
- ESP\_ERR\_HTTPS\_OTA\_IN\_PROGRESS (C 宏), 1124
- ESP\_ERR\_INVALID\_ARG (C 宏), 1127
- ESP\_ERR\_INVALID\_CRC (C 宏), 1127
- ESP\_ERR\_INVALID\_MAC (C 宏), 1127
- ESP\_ERR\_INVALID\_RESPONSE (C 宏), 1127
- ESP\_ERR\_INVALID\_SIZE (C 宏), 1127
- ESP\_ERR\_INVALID\_STATE (C 宏), 1127
- ESP\_ERR\_INVALID\_VERSION (C 宏), 1127
- ESP\_ERR\_MBEDTLS\_CERT\_PARTLY\_OK (C 宏), 769
- ESP\_ERR\_MBEDTLS\_CTR\_DRBG\_SEED\_FAILED (C 宏), 769
- ESP\_ERR\_MBEDTLS\_PK\_PARSE\_KEY\_FAILED (C 宏), 769
- ESP\_ERR\_MBEDTLS\_SSL\_CONF\_ALPN\_PROTOCOLS\_FAILED (C 宏), 769
- ESP\_ERR\_MBEDTLS\_SSL\_CONF\_OWN\_CERT\_FAILED (C 宏), 769
- ESP\_ERR\_MBEDTLS\_SSL\_CONF\_PSK\_FAILED (C 宏), 769
- ESP\_ERR\_MBEDTLS\_SSL\_CONFIG\_DEFAULTS\_FAILED (C 宏), 769
- ESP\_ERR\_MBEDTLS\_SSL\_HANDSHAKE\_FAILED (C 宏), 769
- ESP\_ERR\_MBEDTLS\_SSL\_SET\_HOSTNAME\_FAILED (C 宏), 769
- ESP\_ERR\_MBEDTLS\_SSL\_SETUP\_FAILED (C 宏), 769
- ESP\_ERR\_MBEDTLS\_SSL\_WRITE\_FAILED (C 宏), 769
- ESP\_ERR\_MBEDTLS\_X509\_CRT\_PARSE\_FAILED (C 宏), 769
- ESP\_ERR\_MESH\_ARGUMENT (C 宏), 527
- ESP\_ERR\_MESH\_BASE (C 宏), 1128
- ESP\_ERR\_MESH\_DISCARD (C 宏), 528
- ESP\_ERR\_MESH\_DISCARD\_DUPLICATE (C 宏), 528
- ESP\_ERR\_MESH\_DISCONNECTED (C 宏), 527
- ESP\_ERR\_MESH\_EXCEED\_MTU (C 宏), 527
- ESP\_ERR\_MESH\_INTERFACE (C 宏), 527
- ESP\_ERR\_MESH\_NO\_MEMORY (C 宏), 527
- ESP\_ERR\_MESH\_NO\_PARENT\_FOUND (C 宏), 527
- ESP\_ERR\_MESH\_NO\_ROUTE\_FOUND (C 宏), 527
- ESP\_ERR\_MESH\_NOT\_ALLOWED (C 宏), 527
- ESP\_ERR\_MESH\_NOT\_CONFIG (C 宏), 527
- ESP\_ERR\_MESH\_NOT\_INIT (C 宏), 527
- ESP\_ERR\_MESH\_NOT\_START (C 宏), 527
- ESP\_ERR\_MESH\_NOT\_SUPPORT (C 宏), 527
- ESP\_ERR\_MESH\_OPTION\_NULL (C 宏), 527
- ESP\_ERR\_MESH\_OPTION\_UNKNOWN (C 宏), 527
- ESP\_ERR\_MESH\_QUEUE\_FAIL (C 宏), 527
- ESP\_ERR\_MESH\_QUEUE\_FULL (C 宏), 527
- ESP\_ERR\_MESH\_TIMEOUT (C 宏), 527
- ESP\_ERR\_MESH\_VOTING (C 宏), 528
- ESP\_ERR\_MESH\_WIFI\_NOT\_START (C 宏), 527
- ESP\_ERR\_MESH\_XON\_NO\_WINDOW (C 宏), 527
- ESP\_ERR\_NO\_MEM (C 宏), 1127
- ESP\_ERR\_NOT\_FOUND (C 宏), 1127
- ESP\_ERR\_NOT\_SUPPORTED (C 宏), 1127
- ESP\_ERR\_NVS\_BASE (C 宏), 903
- ESP\_ERR\_NVS\_CONTENT\_DIFFERS (C 宏), 904
- ESP\_ERR\_NVS\_CORRUPT\_KEY\_PART (C 宏), 904
- ESP\_ERR\_NVS\_ENCR\_NOT\_SUPPORTED (C 宏), 903
- ESP\_ERR\_NVS\_INVALID\_HANDLE (C 宏), 903
- ESP\_ERR\_NVS\_INVALID\_LENGTH (C 宏), 903
- ESP\_ERR\_NVS\_INVALID\_NAME (C 宏), 903
- ESP\_ERR\_NVS\_INVALID\_STATE (C 宏), 903
- ESP\_ERR\_NVS\_KEY\_TOO\_LONG (C 宏), 903
- ESP\_ERR\_NVS\_KEYS\_NOT\_INITIALIZED (C 宏), 903
- ESP\_ERR\_NVS\_NEW\_VERSION\_FOUND (C 宏), 903
- ESP\_ERR\_NVS\_NO\_FREE\_PAGES (C 宏), 903
- ESP\_ERR\_NVS\_NOT\_ENOUGH\_SPACE (C 宏), 903
- ESP\_ERR\_NVS\_NOT\_FOUND (C 宏), 903
- ESP\_ERR\_NVS\_NOT\_INITIALIZED (C 宏), 903
- ESP\_ERR\_NVS\_PAGE\_FULL (C 宏), 903
- ESP\_ERR\_NVS\_PART\_NOT\_FOUND (C 宏), 903
- ESP\_ERR\_NVS\_READ\_ONLY (C 宏), 903
- ESP\_ERR\_NVS\_REMOVE\_FAILED (C 宏), 903
- ESP\_ERR\_NVS\_TYPE\_MISMATCH (C 宏), 903
- ESP\_ERR\_NVS\_VALUE\_TOO\_LONG (C 宏), 903
- ESP\_ERR\_NVS\_XTS\_CFG\_FAILED (C 宏), 903
- ESP\_ERR\_NVS\_XTS\_CFG\_NOT\_FOUND (C 宏), 903
- ESP\_ERR\_NVS\_XTS\_DECR\_FAILED (C 宏), 903
- ESP\_ERR\_NVS\_XTS\_ENCR\_FAILED (C 宏), 903
- ESP\_ERR\_OTA\_BASE (C 宏), 1121
- ESP\_ERR\_OTA\_PARTITION\_CONFLICT (C 宏), 1121
- ESP\_ERR\_OTA\_ROLLBACK\_FAILED (C 宏), 1121
- ESP\_ERR\_OTA\_ROLLBACK\_INVALID\_STATE (C 宏), 1121
- ESP\_ERR\_OTA\_SELECT\_INFO\_INVALID (C 宏), 1121
- ESP\_ERR\_OTA\_SMALL\_SEC\_VER (C 宏), 1121
- ESP\_ERR\_OTA\_VALIDATE\_FAILED (C 宏), 1121
- esp\_err\_t (C++ type), 1128
- ESP\_ERR\_TCPIP\_ADAPTER\_BASE (C 宏), 554
- ESP\_ERR\_TCPIP\_ADAPTER\_DHCP\_ALREADY\_STARTED (C 宏), 554
- ESP\_ERR\_TCPIP\_ADAPTER\_DHCP\_ALREADY\_STOPPED (C 宏), 554
- ESP\_ERR\_TCPIP\_ADAPTER\_DHCP\_NOT\_STOPPED (C 宏), 554

- ESP\_ERR\_TCPIP\_ADAPTER\_DHCP\_START\_FAILED (C 宏), 554
- ESP\_ERR\_TCPIP\_ADAPTER\_IF\_NOT\_READY (C 宏), 554
- ESP\_ERR\_TCPIP\_ADAPTER\_INVALID\_PARAMS (C 宏), 554
- ESP\_ERR\_TCPIP\_ADAPTER\_NO\_MEM (C 宏), 554
- ESP\_ERR\_TIMEOUT (C 宏), 1127
- esp\_err\_to\_name (C++ function), 1127
- esp\_err\_to\_name\_r (C++ function), 1127
- ESP\_ERR\_ULP\_BASE (C 宏), 1462
- ESP\_ERR\_ULP\_BRANCH\_OUT\_OF\_RANGE (C 宏), 1462
- ESP\_ERR\_ULP\_DUPLICATE\_LABEL (C 宏), 1462
- ESP\_ERR\_ULP\_INVALID\_LOAD\_ADDR (C 宏), 1462
- ESP\_ERR\_ULP\_SIZE\_TOO\_BIG (C 宏), 1462
- ESP\_ERR\_ULP\_UNDEFINED\_LABEL (C 宏), 1462
- ESP\_ERR\_WIFI\_BASE (C 宏), 1127
- ESP\_ERR\_WIFI\_CONN (C 宏), 475
- ESP\_ERR\_WIFI\_IF (C 宏), 475
- ESP\_ERR\_WIFI\_INIT\_STATE (C 宏), 476
- ESP\_ERR\_WIFI\_MAC (C 宏), 475
- ESP\_ERR\_WIFI\_MODE (C 宏), 475
- ESP\_ERR\_WIFI\_NOT\_ASSOC (C 宏), 476
- ESP\_ERR\_WIFI\_NOT\_CONNECT (C 宏), 476
- ESP\_ERR\_WIFI\_NOT\_INIT (C 宏), 475
- ESP\_ERR\_WIFI\_NOT\_STARTED (C 宏), 475
- ESP\_ERR\_WIFI\_NOT\_STOPPED (C 宏), 475
- ESP\_ERR\_WIFI\_NV\_S (C 宏), 475
- ESP\_ERR\_WIFI\_PASSWORD (C 宏), 476
- ESP\_ERR\_WIFI\_POST (C 宏), 476
- ESP\_ERR\_WIFI\_SSID (C 宏), 475
- ESP\_ERR\_WIFI\_STATE (C 宏), 475
- ESP\_ERR\_WIFI\_STOP\_STATE (C 宏), 476
- ESP\_ERR\_WIFI\_TIMEOUT (C 宏), 476
- ESP\_ERR\_WIFI\_TX\_DISALLOW (C 宏), 476
- ESP\_ERR\_WIFI\_WAKE\_FAIL (C 宏), 476
- ESP\_ERR\_WIFI\_WOULD\_BLOCK (C 宏), 476
- ESP\_ERROR\_CHECK (C 宏), 1128
- ESP\_ERROR\_CHECK\_WITHOUT\_ABORT (C 宏), 1128
- esp\_esptouch\_set\_timeout (C++ function), 495
- esp\_eth\_config\_t (C++ class), 534
- esp\_eth\_config\_t::check\_link\_period\_ms (C++ member), 534
- esp\_eth\_config\_t::mac (C++ member), 534
- esp\_eth\_config\_t::on\_lowlevel\_deinit\_done (C++ member), 535
- esp\_eth\_config\_t::on\_lowlevel\_init\_done (C++ member), 534
- esp\_eth\_config\_t::phy (C++ member), 534
- esp\_eth\_config\_t::stack\_input (C++ member), 534
- esp\_eth\_decrease\_reference (C++ function), 534
- esp\_eth\_driver\_install (C++ function), 532
- esp\_eth\_driver\_uninstall (C++ function), 532
- esp\_eth\_handle\_t (C++ type), 535
- esp\_eth\_increase\_reference (C++ function), 534
- esp\_eth\_io\_cmd\_t (C++ enum), 537
- esp\_eth\_ioctl (C++ function), 533
- esp\_eth\_mac\_s (C++ class), 538
- esp\_eth\_mac\_s::deinit (C++ member), 538
- esp\_eth\_mac\_s::del (C++ member), 541
- esp\_eth\_mac\_s::get\_addr (C++ member), 540
- esp\_eth\_mac\_s::init (C++ member), 538
- esp\_eth\_mac\_s::read\_phy\_reg (C++ member), 539
- esp\_eth\_mac\_s::receive (C++ member), 539
- esp\_eth\_mac\_s::set\_addr (C++ member), 540
- esp\_eth\_mac\_s::set\_duplex (C++ member), 540
- esp\_eth\_mac\_s::set\_link (C++ member), 540
- esp\_eth\_mac\_s::set\_mediator (C++ member), 538
- esp\_eth\_mac\_s::set\_promiscuous (C++ member), 540
- esp\_eth\_mac\_s::set\_speed (C++ member), 540
- esp\_eth\_mac\_s::start (C++ member), 538
- esp\_eth\_mac\_s::stop (C++ member), 539
- esp\_eth\_mac\_s::transmit (C++ member), 539
- esp\_eth\_mac\_s::write\_phy\_reg (C++ member), 539
- esp\_eth\_mac\_t (C++ type), 541
- esp\_eth\_mediator\_s (C++ class), 535
- esp\_eth\_mediator\_s::on\_state\_changed (C++ member), 536
- esp\_eth\_mediator\_s::phy\_reg\_read (C++ member), 535
- esp\_eth\_mediator\_s::phy\_reg\_write (C++ member), 535
- esp\_eth\_mediator\_s::stack\_input (C++ member), 536
- esp\_eth\_mediator\_t (C++ type), 536
- ESP\_ETH\_PHY\_ADDR\_AUTO (C 宏), 544
- esp\_eth\_phy\_new\_dp83848 (C++ function), 542
- esp\_eth\_phy\_new\_ip101 (C++ function), 542
- esp\_eth\_phy\_new\_lan8720 (C++ function), 542
- esp\_eth\_phy\_new\_rtl8201 (C++ function), 542
- esp\_eth\_phy\_s (C++ class), 542
- esp\_eth\_phy\_s::deinit (C++ member), 543
- esp\_eth\_phy\_s::del (C++ member), 544
- esp\_eth\_phy\_s::get\_addr (C++ member), 544
- esp\_eth\_phy\_s::get\_link (C++ member), 543
- esp\_eth\_phy\_s::init (C++ member), 543
- esp\_eth\_phy\_s::negotiate (C++ member), 543
- esp\_eth\_phy\_s::pwrctl (C++ member), 543
- esp\_eth\_phy\_s::reset (C++ member), 542
- esp\_eth\_phy\_s::reset\_hw (C++ member), 543
- esp\_eth\_phy\_s::set\_addr (C++ member), 543

- esp\_eth\_phy\_s::set\_mediator (C++ member), 542  
 esp\_eth\_phy\_t (C++ type), 544  
 esp\_eth\_receive (C++ function), 533  
 esp\_eth\_start (C++ function), 533  
 esp\_eth\_state\_t (C++ enum), 536  
 esp\_eth\_stop (C++ function), 533  
 esp\_eth\_transmit (C++ function), 533  
 ESP\_EVENT\_ANY\_BASE (C 宏), 1090  
 ESP\_EVENT\_ANY\_ID (C 宏), 1090  
 esp\_event\_base\_t (C++ type), 1090  
 ESP\_EVENT\_DECLARE\_BASE (C 宏), 1090  
 ESP\_EVENT\_DEFINE\_BASE (C 宏), 1090  
 esp\_event\_dump (C++ function), 1089  
 esp\_event\_handler\_register (C++ function), 1087  
 esp\_event\_handler\_register\_with (C++ function), 1087  
 esp\_event\_handler\_t (C++ type), 1090  
 esp\_event\_handler\_unregister (C++ function), 1088  
 esp\_event\_handler\_unregister\_with (C++ function), 1088  
 esp\_event\_loop\_args\_t (C++ class), 1090  
 esp\_event\_loop\_args\_t::queue\_size (C++ member), 1090  
 esp\_event\_loop\_args\_t::task\_core\_id (C++ member), 1090  
 esp\_event\_loop\_args\_t::task\_name (C++ member), 1090  
 esp\_event\_loop\_args\_t::task\_priority (C++ member), 1090  
 esp\_event\_loop\_args\_t::task\_stack\_size (C++ member), 1090  
 esp\_event\_loop\_create (C++ function), 1086  
 esp\_event\_loop\_create\_default (C++ function), 1086  
 esp\_event\_loop\_delete (C++ function), 1086  
 esp\_event\_loop\_delete\_default (C++ function), 1086  
 esp\_event\_loop\_handle\_t (C++ type), 1090  
 esp\_event\_loop\_init (C++ function), 1091  
 esp\_event\_loop\_run (C++ function), 1086  
 esp\_event\_loop\_set\_cb (C++ function), 1091  
 esp\_event\_post (C++ function), 1088  
 esp\_event\_post\_to (C++ function), 1089  
 esp\_event\_process\_default (C++ function), 1091  
 esp\_event\_send (C++ function), 1091  
 esp\_event\_set\_default\_eth\_handlers (C++ function), 1091  
 esp\_event\_set\_default\_wifi\_handlers (C++ function), 1091  
 ESP\_EXT1\_WAKEUP\_ALL\_LOW (C++ enumerator), 1111  
 ESP\_EXT1\_WAKEUP\_ANY\_HIGH (C++ enumerator), 1111  
 ESP\_FAIL (C 宏), 1127  
 esp\_fill\_random (C++ function), 1137  
 ESP\_FLASH\_10MHZ (C++ enumerator), 871  
 ESP\_FLASH\_20MHZ (C++ enumerator), 871  
 ESP\_FLASH\_26MHZ (C++ enumerator), 871  
 ESP\_FLASH\_40MHZ (C++ enumerator), 871  
 ESP\_FLASH\_5MHZ (C++ enumerator), 871  
 ESP\_FLASH\_80MHZ (C++ enumerator), 871  
 esp\_flash\_chip\_driver\_initialized (C++ function), 865  
 ESP\_FLASH\_ENC\_MODE\_DEVELOPMENT (C++ enumerator), 879  
 ESP\_FLASH\_ENC\_MODE\_DISABLED (C++ enumerator), 879  
 ESP\_FLASH\_ENC\_MODE\_RELEASE (C++ enumerator), 879  
 esp\_flash\_enc\_mode\_t (C++ enum), 879  
 esp\_flash\_encrypt\_check\_and\_update (C++ function), 878  
 esp\_flash\_encrypt\_region (C++ function), 878  
 esp\_flash\_encryption\_enabled (C++ function), 878  
 esp\_flash\_encryption\_init\_checks (C++ function), 878  
 esp\_flash\_erase\_chip (C++ function), 866  
 esp\_flash\_erase\_region (C++ function), 866  
 esp\_flash\_get\_chip\_write\_protect (C++ function), 866  
 esp\_flash\_get\_protectable\_regions (C++ function), 867  
 esp\_flash\_get\_protected\_region (C++ function), 867  
 esp\_flash\_get\_size (C++ function), 866  
 esp\_flash\_init (C++ function), 865  
 esp\_flash\_io\_mode\_t (C++ enum), 872  
 esp\_flash\_is\_quad\_mode (C++ function), 868  
 esp\_flash\_os\_functions\_t (C++ class), 869  
 esp\_flash\_os\_functions\_t::delay\_us (C++ member), 869  
 esp\_flash\_os\_functions\_t::end (C++ member), 869  
 esp\_flash\_os\_functions\_t::region\_protected (C++ member), 869  
 esp\_flash\_os\_functions\_t::start (C++ member), 869  
 esp\_flash\_os\_functions\_t::yield (C++ member), 869  
 esp\_flash\_read (C++ function), 867  
 esp\_flash\_read\_encrypted (C++ function), 868  
 esp\_flash\_read\_id (C++ function), 865  
 esp\_flash\_region\_t (C++ class), 868  
 esp\_flash\_region\_t::offset (C++ member), 869  
 esp\_flash\_region\_t::size (C++ member), 869  
 esp\_flash\_set\_chip\_write\_protect (C++ function), 866

- esp\_flash\_set\_protected\_region (C++ function), 867
- ESP\_FLASH\_SPEED\_MAX (C++ enumerator), 871
- ESP\_FLASH\_SPEED\_MIN (C 宏), 871
- esp\_flash\_speed\_t (C++ enum), 871
- esp\_flash\_spi\_device\_config\_t (C++ class), 865
- esp\_flash\_spi\_device\_config\_t::cs\_id (C++ member), 865
- esp\_flash\_spi\_device\_config\_t::cs\_io\_num (C++ member), 865
- esp\_flash\_spi\_device\_config\_t::host\_id (C++ member), 865
- esp\_flash\_spi\_device\_config\_t::input\_delay\_ns (C++ member), 865
- esp\_flash\_spi\_device\_config\_t::io\_mode (C++ member), 865
- esp\_flash\_spi\_device\_config\_t::speed (C++ member), 865
- esp\_flash\_t (C++ class), 869
- esp\_flash\_t (C++ type), 869
- esp\_flash\_t::chip\_drv (C++ member), 869
- esp\_flash\_t::chip\_id (C++ member), 869
- esp\_flash\_t::host (C++ member), 869
- esp\_flash\_t::os\_func (C++ member), 869
- esp\_flash\_t::os\_func\_data (C++ member), 869
- esp\_flash\_t::read\_mode (C++ member), 869
- esp\_flash\_t::size (C++ member), 869
- esp\_flash\_write (C++ function), 868
- esp\_flash\_write\_encrypted (C++ function), 868
- esp\_flash\_write\_protect\_crypt\_cnt (C++ function), 878
- esp\_freertos\_idle\_cb\_t (C++ type), 1033
- esp\_freertos\_tick\_cb\_t (C++ type), 1033
- ESP\_GAP\_BLE\_ADD\_WHITELIST\_COMPLETE\_EVT (C 宏), 133
- ESP\_GAP\_BLE\_ADV\_DATA\_RAW\_SET\_COMPLETE\_EVT (C++ enumerator), 134
- ESP\_GAP\_BLE\_ADV\_DATA\_SET\_COMPLETE\_EVT (C++ enumerator), 134
- ESP\_GAP\_BLE\_ADV\_START\_COMPLETE\_EVT (C++ enumerator), 134
- ESP\_GAP\_BLE\_ADV\_STOP\_COMPLETE\_EVT (C++ enumerator), 134
- ESP\_GAP\_BLE\_AUTH\_CMPL\_EVT (C++ enumerator), 134
- esp\_gap\_ble\_cb\_event\_t (C++ enum), 134
- esp\_gap\_ble\_cb\_t (C++ type), 133
- ESP\_GAP\_BLE\_CLEAR\_BOND\_DEV\_COMPLETE\_EVT (C++ enumerator), 134
- ESP\_GAP\_BLE\_EVT\_MAX (C++ enumerator), 135
- ESP\_GAP\_BLE\_GET\_BOND\_DEV\_COMPLETE\_EVT (C++ enumerator), 135
- ESP\_GAP\_BLE\_KEY\_EVT (C++ enumerator), 134
- ESP\_GAP\_BLE\_LOCAL\_ER\_EVT (C++ enumerator), 134
- ESP\_GAP\_BLE\_LOCAL\_IR\_EVT (C++ enumerator), 134
- ESP\_GAP\_BLE\_NC\_REQ\_EVT (C++ enumerator), 134
- ESP\_GAP\_BLE\_OOB\_REQ\_EVT (C++ enumerator), 134
- ESP\_GAP\_BLE\_PASSKEY\_NOTIF\_EVT (C++ enumerator), 134
- ESP\_GAP\_BLE\_PASSKEY\_REQ\_EVT (C++ enumerator), 134
- ESP\_GAP\_BLE\_READ\_RSSI\_COMPLETE\_EVT (C++ enumerator), 135
- ESP\_GAP\_BLE\_REMOVE\_BOND\_DEV\_COMPLETE\_EVT (C++ enumerator), 134
- ESP\_GAP\_BLE\_SCAN\_PARAM\_SET\_COMPLETE\_EVT (C++ enumerator), 134
- ESP\_GAP\_BLE\_SCAN\_RESULT\_EVT (C++ enumerator), 134
- ESP\_GAP\_BLE\_SCAN\_RSP\_DATA\_RAW\_SET\_COMPLETE\_EVT (C++ enumerator), 134
- ESP\_GAP\_BLE\_SCAN\_RSP\_DATA\_SET\_COMPLETE\_EVT (C++ enumerator), 134
- ESP\_GAP\_BLE\_SCAN\_START\_COMPLETE\_EVT (C++ enumerator), 134
- ESP\_GAP\_BLE\_SCAN\_STOP\_COMPLETE\_EVT (C++ enumerator), 134
- ESP\_GAP\_BLE\_SEC\_REQ\_EVT (C++ enumerator), 134
- esp\_gap\_ble\_set\_authorization (C++ function), 119
- ESP\_GAP\_BLE\_SET\_LOCAL\_PRIVACY\_COMPLETE\_EVT (C++ enumerator), 134
- ESP\_GAP\_BLE\_SET\_PKT\_LENGTH\_COMPLETE\_EVT (C++ enumerator), 134
- ESP\_GAP\_BLE\_SET\_STATIC\_RAND\_ADDR\_EVT (C++ enumerator), 134
- ESP\_GAP\_BLE\_UPDATE\_CONN\_PARAMS\_EVT (C++ enumerator), 134
- ESP\_GAP\_BLE\_UPDATE\_DUPLICATE\_EXCEPTIONAL\_LIST\_COMPLETE\_EVT (C++ enumerator), 135
- ESP\_GAP\_BLE\_UPDATE\_WHITELIST\_COMPLETE\_EVT (C++ enumerator), 135
- esp\_gap\_conn\_params\_t (C++ class), 127
- esp\_gap\_conn\_params\_t::interval (C++ member), 127
- esp\_gap\_conn\_params\_t::latency (C++ member), 127
- esp\_gap\_conn\_params\_t::timeout (C++ member), 127
- ESP\_GAP\_SEARCH\_DI\_DISC\_CMPL\_EVT (C++ enumerator), 138
- ESP\_GAP\_SEARCH\_DISC\_BLE\_RES\_EVT (C++ enumerator), 138
- ESP\_GAP\_SEARCH\_DISC\_CMPL\_EVT (C++ enumerator), 138
- ESP\_GAP\_SEARCH\_DISC\_RES\_EVT (C++ enumerator), 138
- esp\_gap\_search\_evt\_t (C++ enum), 138

- ESP\_GAP\_SEARCH\_INQ\_CMPL\_EVT (C++ *enumerator*), 138
- ESP\_GAP\_SEARCH\_INQ\_DISCARD\_NUM\_EVT (C++ *enumerator*), 138
- ESP\_GAP\_SEARCH\_INQ\_RES\_EVT (C++ *enumerator*), 138
- ESP\_GAP\_SEARCH\_SEARCH\_CANCEL\_CMPL\_EVT (C++ *enumerator*), 138
- ESP\_GATT\_ALREADY\_OPEN (C++ *enumerator*), 148
- ESP\_GATT\_APP\_RSP (C++ *enumerator*), 148
- ESP\_GATT\_ATTR\_HANDLE\_MAX (C 宏), 146
- ESP\_GATT\_AUTH\_FAIL (C++ *enumerator*), 148
- ESP\_GATT\_AUTH\_REQ\_MITM (C++ *enumerator*), 149
- ESP\_GATT\_AUTH\_REQ\_NO\_MITM (C++ *enumerator*), 149
- ESP\_GATT\_AUTH\_REQ\_NONE (C++ *enumerator*), 149
- ESP\_GATT\_AUTH\_REQ\_SIGNED\_MITM (C++ *enumerator*), 149
- ESP\_GATT\_AUTH\_REQ\_SIGNED\_NO\_MITM (C++ *enumerator*), 149
- esp\_gatt\_auth\_req\_t (C++ *enum*), 149
- ESP\_GATT\_AUTO\_RSP (C 宏), 147
- ESP\_GATT\_BODY\_SENSOR\_LOCATION (C 宏), 146
- ESP\_GATT\_BUSY (C++ *enumerator*), 148
- ESP\_GATT\_CANCEL (C++ *enumerator*), 148
- ESP\_GATT\_CCC\_CFG\_ERR (C++ *enumerator*), 148
- ESP\_GATT\_CHAR\_PROP\_BIT\_AUTH (C 宏), 147
- ESP\_GATT\_CHAR\_PROP\_BIT\_BROADCAST (C 宏), 146
- ESP\_GATT\_CHAR\_PROP\_BIT\_EXT\_PROP (C 宏), 147
- ESP\_GATT\_CHAR\_PROP\_BIT\_INDICATE (C 宏), 147
- ESP\_GATT\_CHAR\_PROP\_BIT\_NOTIFY (C 宏), 147
- ESP\_GATT\_CHAR\_PROP\_BIT\_READ (C 宏), 147
- ESP\_GATT\_CHAR\_PROP\_BIT\_WRITE (C 宏), 147
- ESP\_GATT\_CHAR\_PROP\_BIT\_WRITE\_NR (C 宏), 147
- esp\_gatt\_char\_prop\_t (C++ *type*), 147
- ESP\_GATT\_CMD\_STARTED (C++ *enumerator*), 148
- ESP\_GATT\_CONGESTED (C++ *enumerator*), 148
- ESP\_GATT\_CONN\_CONN\_CANCEL (C++ *enumerator*), 149
- ESP\_GATT\_CONN\_FAIL\_ESTABLISH (C++ *enumerator*), 149
- ESP\_GATT\_CONN\_L2C\_FAILURE (C++ *enumerator*), 148
- ESP\_GATT\_CONN\_LMP\_TIMEOUT (C++ *enumerator*), 149
- ESP\_GATT\_CONN\_NONE (C++ *enumerator*), 149
- esp\_gatt\_conn\_params\_t (C++ *class*), 142
- esp\_gatt\_conn\_params\_t::interval (C++ *member*), 142
- esp\_gatt\_conn\_params\_t::latency (C++ *member*), 142
- esp\_gatt\_conn\_params\_t::timeout (C++ *member*), 142
- esp\_gatt\_conn\_reason\_t (C++ *enum*), 148
- ESP\_GATT\_CONN\_TERMINATE\_LOCAL\_HOST (C++ *enumerator*), 149
- ESP\_GATT\_CONN\_TERMINATE\_PEER\_USER (C++ *enumerator*), 149
- ESP\_GATT\_CONN\_TIMEOUT (C++ *enumerator*), 148
- ESP\_GATT\_CONN\_UNKNOWN (C++ *enumerator*), 148
- ESP\_GATT\_DB\_ALL (C++ *enumerator*), 150
- esp\_gatt\_db\_attr\_type\_t (C++ *enum*), 149
- ESP\_GATT\_DB\_CHARACTERISTIC (C++ *enumerator*), 149
- ESP\_GATT\_DB\_DESCRIPTOR (C++ *enumerator*), 149
- ESP\_GATT\_DB\_FULL (C++ *enumerator*), 148
- ESP\_GATT\_DB\_INCLUDED\_SERVICE (C++ *enumerator*), 150
- ESP\_GATT\_DB\_PRIMARY\_SERVICE (C++ *enumerator*), 149
- ESP\_GATT\_DB\_SECONDARY\_SERVICE (C++ *enumerator*), 149
- ESP\_GATT\_DUP\_REG (C++ *enumerator*), 148
- ESP\_GATT\_ENCRYPTED\_MITM (C++ *enumerator*), 148
- ESP\_GATT\_ENCRYPTED\_NO\_MITM (C++ *enumerator*), 148
- ESP\_GATT\_ERR\_UNLIKELY (C++ *enumerator*), 148
- ESP\_GATT\_ERROR (C++ *enumerator*), 148
- ESP\_GATT\_HEART\_RATE\_CNTL\_POINT (C 宏), 146
- ESP\_GATT\_HEART\_RATE\_MEAS (C 宏), 146
- esp\_gatt\_id\_t (C++ *class*), 140
- esp\_gatt\_id\_t::inst\_id (C++ *member*), 140
- esp\_gatt\_id\_t::uuid (C++ *member*), 140
- ESP\_GATT\_IF\_NONE (C 宏), 147
- esp\_gatt\_if\_t (C++ *type*), 147
- ESP\_GATT\_ILLEGAL\_HANDLE (C 宏), 146
- ESP\_GATT\_ILLEGAL\_PARAMETER (C++ *enumerator*), 148
- ESP\_GATT\_ILLEGAL\_UUID (C 宏), 146
- ESP\_GATT\_INSUF\_AUTHENTICATION (C++ *enumerator*), 147
- ESP\_GATT\_INSUF\_AUTHORIZATION (C++ *enumerator*), 147
- ESP\_GATT\_INSUF\_ENCRYPTION (C++ *enumerator*), 148
- ESP\_GATT\_INSUF\_KEY\_SIZE (C++ *enumerator*), 148
- ESP\_GATT\_INSUF\_RESOURCE (C++ *enumerator*), 148
- ESP\_GATT\_INTERNAL\_ERROR (C++ *enumerator*), 148
- ESP\_GATT\_INVALID\_ATTR\_LEN (C++ *enumerator*), 148
- ESP\_GATT\_INVALID\_CFG (C++ *enumerator*), 148
- ESP\_GATT\_INVALID\_HANDLE (C++ *enumerator*), 147
- ESP\_GATT\_INVALID\_OFFSET (C++ *enumerator*),

- 147
- ESP\_GATT\_INVALID\_PDU (C++ *enumerator*), 147
- ESP\_GATT\_MAX\_ATTR\_LEN (C 宏), 147
- ESP\_GATT\_MAX\_READ\_MULTI\_HANDLES (C 宏), 146
- ESP\_GATT\_MORE (C++ *enumerator*), 148
- ESP\_GATT\_NO\_RESOURCES (C++ *enumerator*), 148
- ESP\_GATT\_NOT\_ENCRYPTED (C++ *enumerator*), 148
- ESP\_GATT\_NOT\_FOUND (C++ *enumerator*), 147
- ESP\_GATT\_NOT\_LONG (C++ *enumerator*), 147
- ESP\_GATT\_OK (C++ *enumerator*), 147
- ESP\_GATT\_OUT\_OF\_RANGE (C++ *enumerator*), 148
- ESP\_GATT\_PENDING (C++ *enumerator*), 148
- ESP\_GATT\_PERM\_READ (C 宏), 146
- ESP\_GATT\_PERM\_READ\_AUTHORIZATION (C 宏), 146
- ESP\_GATT\_PERM\_READ\_ENC\_MITM (C 宏), 146
- ESP\_GATT\_PERM\_READ\_ENCRYPTED (C 宏), 146
- esp\_gatt\_perm\_t (C++ *type*), 147
- ESP\_GATT\_PERM\_WRITE (C 宏), 146
- ESP\_GATT\_PERM\_WRITE\_AUTHORIZATION (C 宏), 146
- ESP\_GATT\_PERM\_WRITE\_ENC\_MITM (C 宏), 146
- ESP\_GATT\_PERM\_WRITE\_ENCRYPTED (C 宏), 146
- ESP\_GATT\_PERM\_WRITE\_SIGNED (C 宏), 146
- ESP\_GATT\_PERM\_WRITE\_SIGNED\_MITM (C 宏), 146
- ESP\_GATT\_PRC\_IN\_PROGRESS (C++ *enumerator*), 148
- ESP\_GATT\_PREP\_WRITE\_CANCEL (C 宏), 160
- ESP\_GATT\_PREP\_WRITE\_CANCEL (C++ *enumerator*), 147
- ESP\_GATT\_PREP\_WRITE\_EXEC (C 宏), 160
- ESP\_GATT\_PREP\_WRITE\_EXEC (C++ *enumerator*), 147
- esp\_gatt\_prep\_write\_type (C++ *enum*), 147
- ESP\_GATT\_PREPARE\_Q\_FULL (C++ *enumerator*), 147
- ESP\_GATT\_READ\_NOT\_PERMIT (C++ *enumerator*), 147
- ESP\_GATT\_REQ\_NOT\_SUPPORTED (C++ *enumerator*), 147
- ESP\_GATT\_RSP\_BY\_APP (C 宏), 147
- esp\_gatt\_rsp\_t (C++ *union*), 139
- esp\_gatt\_rsp\_t::attr\_value (C++ *member*), 140
- esp\_gatt\_rsp\_t::handle (C++ *member*), 140
- ESP\_GATT\_SERVICE\_FROM\_NVS\_FLASH (C++ *enumerator*), 149
- ESP\_GATT\_SERVICE\_FROM\_REMOTE\_DEVICE (C++ *enumerator*), 149
- ESP\_GATT\_SERVICE\_FROM\_UNKNOWN (C++ *enumerator*), 149
- ESP\_GATT\_SERVICE\_STARTED (C++ *enumerator*), 148
- esp\_gatt\_srvc\_id\_t (C++ *class*), 140
- esp\_gatt\_srvc\_id\_t::id (C++ *member*), 140
- esp\_gatt\_srvc\_id\_t::is\_primary (C++ *member*), 140
- ESP\_GATT\_STACK\_RSP (C++ *enumerator*), 148
- esp\_gatt\_status\_t (C++ *enum*), 147
- ESP\_GATT\_UNKNOWN\_ERROR (C++ *enumerator*), 148
- ESP\_GATT\_UNSUPPORT\_GRP\_TYPE (C++ *enumerator*), 148
- ESP\_GATT\_UUID\_ALERT\_LEVEL (C 宏), 145
- ESP\_GATT\_UUID\_ALERT\_NTF\_SVC (C 宏), 144
- ESP\_GATT\_UUID\_ALERT\_STATUS (C 宏), 145
- ESP\_GATT\_UUID\_Automation\_IO\_SVC (C 宏), 144
- ESP\_GATT\_UUID\_BATTERY\_LEVEL (C 宏), 146
- ESP\_GATT\_UUID\_BATTERY\_SERVICE\_SVC (C 宏), 144
- ESP\_GATT\_UUID\_BLOOD\_PRESSURE\_SVC (C 宏), 144
- ESP\_GATT\_UUID\_BODY\_COMPOSITION (C 宏), 144
- ESP\_GATT\_UUID\_BOND\_MANAGEMENT\_SVC (C 宏), 144
- ESP\_GATT\_UUID\_CHAR\_AGG\_FORMAT (C 宏), 145
- ESP\_GATT\_UUID\_CHAR\_CLIENT\_CONFIG (C 宏), 144
- ESP\_GATT\_UUID\_CHAR\_DECLARE (C 宏), 144
- ESP\_GATT\_UUID\_CHAR\_DESCRIPTION (C 宏), 144
- ESP\_GATT\_UUID\_CHAR\_EXT\_PROP (C 宏), 144
- ESP\_GATT\_UUID\_CHAR\_PRESENT\_FORMAT (C 宏), 144
- ESP\_GATT\_UUID\_CHAR\_SRVR\_CONFIG (C 宏), 144
- ESP\_GATT\_UUID\_CHAR\_VALID\_RANGE (C 宏), 145
- ESP\_GATT\_UUID\_CONT\_GLUCESE\_MONITOR\_SVC (C 宏), 144
- ESP\_GATT\_UUID\_CSC\_FEATURE (C 宏), 146
- ESP\_GATT\_UUID\_CSC\_MEASUREMENT (C 宏), 146
- ESP\_GATT\_UUID\_CURRENT\_TIME (C 宏), 145
- ESP\_GATT\_UUID\_CURRENT\_TIME\_SVC (C 宏), 144
- ESP\_GATT\_UUID\_CYCLING\_POWER\_SVC (C 宏), 144
- ESP\_GATT\_UUID\_CYCLING\_SPEED\_CADENCE\_SVC (C 宏), 144
- ESP\_GATT\_UUID\_DEVICE\_INFO\_SVC (C 宏), 144
- ESP\_GATT\_UUID\_ENV\_SENSING\_CONFIG\_DESCR (C 宏), 145
- ESP\_GATT\_UUID\_ENV\_SENSING\_MEASUREMENT\_DESCR (C 宏), 145
- ESP\_GATT\_UUID\_ENV\_SENSING\_TRIGGER\_DESCR (C 宏), 145
- ESP\_GATT\_UUID\_ENVIRONMENTAL\_SENSING\_SVC (C 宏), 144
- ESP\_GATT\_UUID\_EXT\_RPT\_REF\_DESCR (C 宏), 145
- ESP\_GATT\_UUID\_FW\_VERSION\_STR (C 宏), 145



- ESP\_GATT\_UUID\_GAP\_CENTRAL\_ADDR\_RESOL (C 宏), 145
- ESP\_GATT\_UUID\_GAP\_DEVICE\_NAME (C 宏), 145
- ESP\_GATT\_UUID\_GAP\_ICON (C 宏), 145
- ESP\_GATT\_UUID\_GAP\_PREF\_CONN\_PARAM (C 宏), 145
- ESP\_GATT\_UUID\_GATT\_SRV\_CHGD (C 宏), 145
- ESP\_GATT\_UUID\_GLUCOSE\_SVC (C 宏), 144
- ESP\_GATT\_UUID\_GM\_CONTEXT (C 宏), 145
- ESP\_GATT\_UUID\_GM\_CONTROL\_POINT (C 宏), 145
- ESP\_GATT\_UUID\_GM\_FEATURE (C 宏), 145
- ESP\_GATT\_UUID\_GM\_MEASUREMENT (C 宏), 145
- ESP\_GATT\_UUID\_HEALTH\_THERMOM\_SVC (C 宏), 144
- ESP\_GATT\_UUID\_HEART\_RATE\_SVC (C 宏), 144
- ESP\_GATT\_UUID\_HID\_BT\_KB\_INPUT (C 宏), 146
- ESP\_GATT\_UUID\_HID\_BT\_KB\_OUTPUT (C 宏), 146
- ESP\_GATT\_UUID\_HID\_BT\_MOUSE\_INPUT (C 宏), 146
- ESP\_GATT\_UUID\_HID\_CONTROL\_POINT (C 宏), 146
- ESP\_GATT\_UUID\_HID\_INFORMATION (C 宏), 145
- ESP\_GATT\_UUID\_HID\_PROTO\_MODE (C 宏), 146
- ESP\_GATT\_UUID\_HID\_REPORT (C 宏), 146
- ESP\_GATT\_UUID\_HID\_REPORT\_MAP (C 宏), 146
- ESP\_GATT\_UUID\_HID\_SVC (C 宏), 144
- ESP\_GATT\_UUID\_HW\_VERSION\_STR (C 宏), 145
- ESP\_GATT\_UUID\_IEEE\_DATA (C 宏), 145
- ESP\_GATT\_UUID\_IMMEDIATE\_ALERT\_SVC (C 宏), 144
- ESP\_GATT\_UUID\_INCLUDE\_SERVICE (C 宏), 144
- ESP\_GATT\_UUID\_LINK\_LOSS\_SVC (C 宏), 144
- ESP\_GATT\_UUID\_LOCAL\_TIME\_INFO (C 宏), 145
- ESP\_GATT\_UUID\_LOCATION\_AND\_NAVIGATION\_SVC (C 宏), 144
- ESP\_GATT\_UUID\_MANU\_NAME (C 宏), 145
- ESP\_GATT\_UUID\_MODEL\_NUMBER\_STR (C 宏), 145
- ESP\_GATT\_UUID\_NEXT\_DST\_CHANGE\_SVC (C 宏), 144
- ESP\_GATT\_UUID\_NUM\_DIGITALS\_DESCR (C 宏), 145
- ESP\_GATT\_UUID\_NW\_STATUS (C 宏), 145
- ESP\_GATT\_UUID\_NW\_TRIGGER (C 宏), 145
- ESP\_GATT\_UUID\_PHONE\_ALERT\_STATUS\_SVC (C 宏), 144
- ESP\_GATT\_UUID\_PNP\_ID (C 宏), 145
- ESP\_GATT\_UUID\_PRI\_SERVICE (C 宏), 144
- ESP\_GATT\_UUID\_REF\_TIME\_INFO (C 宏), 145
- ESP\_GATT\_UUID\_REF\_TIME\_UPDATE\_SVC (C 宏), 144
- ESP\_GATT\_UUID\_RINGER\_CP (C 宏), 145
- ESP\_GATT\_UUID\_RINGER\_SETTING (C 宏), 145
- ESP\_GATT\_UUID\_RPT\_REF\_DESCR (C 宏), 145
- ESP\_GATT\_UUID\_RSC\_FEATURE (C 宏), 146
- ESP\_GATT\_UUID\_RSC\_MEASUREMENT (C 宏), 146
- ESP\_GATT\_UUID\_RUNNING\_SPEED\_CADENCE\_SVC (C 宏), 144
- ESP\_GATT\_UUID\_SC\_CONTROL\_POINT (C 宏), 146
- ESP\_GATT\_UUID\_SCAN\_INT\_WINDOW (C 宏), 146
- ESP\_GATT\_UUID\_SCAN\_PARAMETERS\_SVC (C 宏), 144
- ESP\_GATT\_UUID\_SCAN\_REFRESH (C 宏), 146
- ESP\_GATT\_UUID\_SEC\_SERVICE (C 宏), 144
- ESP\_GATT\_UUID\_SENSOR\_LOCATION (C 宏), 146
- ESP\_GATT\_UUID\_SERIAL\_NUMBER\_STR (C 宏), 145
- ESP\_GATT\_UUID\_SW\_VERSION\_STR (C 宏), 145
- ESP\_GATT\_UUID\_SYSTEM\_ID (C 宏), 145
- ESP\_GATT\_UUID\_TIME\_TRIGGER\_DESCR (C 宏), 145
- ESP\_GATT\_UUID\_TX\_POWER\_LEVEL (C 宏), 145
- ESP\_GATT\_UUID\_TX\_POWER\_SVC (C 宏), 144
- ESP\_GATT\_UUID\_USER\_DATA\_SVC (C 宏), 144
- ESP\_GATT\_UUID\_VALUE\_TRIGGER\_DESCR (C 宏), 145
- ESP\_GATT\_UUID\_WEIGHT\_SCALE\_SVC (C 宏), 144
- esp\_gatt\_value\_t (C++ class), 141
- esp\_gatt\_value\_t::auth\_req (C++ member), 142
- esp\_gatt\_value\_t::handle (C++ member), 142
- esp\_gatt\_value\_t::len (C++ member), 142
- esp\_gatt\_value\_t::offset (C++ member), 142
- esp\_gatt\_value\_t::value (C++ member), 142
- ESP\_GATT\_WRITE\_NOT\_PERMIT (C++ enumerator), 147
- ESP\_GATT\_WRITE\_TYPE\_NO\_RSP (C++ enumerator), 149
- ESP\_GATT\_WRITE\_TYPE\_RSP (C++ enumerator), 149
- esp\_gatt\_write\_type\_t (C++ enum), 149
- ESP\_GATT\_WRONG\_STATE (C++ enumerator), 148
- ESP\_GATTC\_ACL\_EVT (C++ enumerator), 176
- ESP\_GATTC\_ADV\_DATA\_EVT (C++ enumerator), 177
- ESP\_GATTC\_ADV\_VSC\_EVT (C++ enumerator), 177
- ESP\_GATTC\_BTH\_SCAN\_CFG\_EVT (C++ enumerator), 177
- ESP\_GATTC\_BTH\_SCAN\_DIS\_EVT (C++ enumerator), 177
- ESP\_GATTC\_BTH\_SCAN\_ENB\_EVT (C++ enumerator), 177
- ESP\_GATTC\_BTH\_SCAN\_PARAM\_EVT (C++ enumerator), 177
- ESP\_GATTC\_BTH\_SCAN\_RD\_EVT (C++ enumerator), 177
- ESP\_GATTC\_BTH\_SCAN\_THR\_EVT (C++ enumerator), 177
- ESP\_GATTC\_CANCEL\_OPEN\_EVT (C++ enumerator), 176

- esp\_gattc\_cb\_event\_t (C++ enum), 176  
 esp\_gattc\_cb\_t (C++ type), 176  
 ESP\_GATTC\_CFG\_MTU\_EVT (C++ enumerator), 177  
 esp\_gattc\_char\_elem\_t (C++ class), 143  
 esp\_gattc\_char\_elem\_t::char\_handle (C++ member), 143  
 esp\_gattc\_char\_elem\_t::properties (C++ member), 143  
 esp\_gattc\_char\_elem\_t::uuid (C++ member), 143  
 ESP\_GATTC\_CLOSE\_EVT (C++ enumerator), 176  
 ESP\_GATTC\_CONGEST\_EVT (C++ enumerator), 177  
 ESP\_GATTC\_CONNECT\_EVT (C++ enumerator), 177  
 esp\_gattc\_db\_elem\_t (C++ class), 142  
 esp\_gattc\_db\_elem\_t::attribute\_handle (C++ member), 142  
 esp\_gattc\_db\_elem\_t::end\_handle (C++ member), 142  
 esp\_gattc\_db\_elem\_t::properties (C++ member), 142  
 esp\_gattc\_db\_elem\_t::start\_handle (C++ member), 142  
 esp\_gattc\_db\_elem\_t::type (C++ member), 142  
 esp\_gattc\_db\_elem\_t::uuid (C++ member), 143  
 esp\_gattc\_descr\_elem\_t (C++ class), 143  
 esp\_gattc\_descr\_elem\_t::handle (C++ member), 143  
 esp\_gattc\_descr\_elem\_t::uuid (C++ member), 143  
 ESP\_GATTC\_DIS\_SRVC\_CMPL\_EVT (C++ enumerator), 178  
 ESP\_GATTC\_DISCONNECT\_EVT (C++ enumerator), 177  
 ESP\_GATTC\_ENC\_CMPL\_CB\_EVT (C++ enumerator), 177  
 ESP\_GATTC\_EXEC\_EVT (C++ enumerator), 176  
 ESP\_GATTC\_GET\_ADDR\_LIST\_EVT (C++ enumerator), 178  
 esp\_gattc\_incl\_svc\_elem\_t (C++ class), 143  
 esp\_gattc\_incl\_svc\_elem\_t::handle (C++ member), 143  
 esp\_gattc\_incl\_svc\_elem\_t::incl\_srvc\_elem\_handle (C++ member), 143  
 esp\_gattc\_incl\_svc\_elem\_t::incl\_srvc\_start\_handle (C++ member), 143  
 esp\_gattc\_incl\_svc\_elem\_t::uuid (C++ member), 143  
 ESP\_GATTC\_MULT\_ADV\_DATA\_EVT (C++ enumerator), 177  
 ESP\_GATTC\_MULT\_ADV\_DIS\_EVT (C++ enumerator), 177  
 ESP\_GATTC\_MULT\_ADV\_ENB\_EVT (C++ enumerator), 177  
 ESP\_GATTC\_MULT\_ADV\_UPD\_EVT (C++ enumerator), 177  
 esp\_gattc\_multi\_t (C++ class), 142  
 esp\_gattc\_multi\_t::handles (C++ member), 142  
 esp\_gattc\_multi\_t::num\_attr (C++ member), 142  
 ESP\_GATTC\_NOTIFY\_EVT (C++ enumerator), 176  
 ESP\_GATTC\_OPEN\_EVT (C++ enumerator), 176  
 ESP\_GATTC\_PREP\_WRITE\_EVT (C++ enumerator), 176  
 ESP\_GATTC\_QUEUE\_FULL\_EVT (C++ enumerator), 178  
 ESP\_GATTC\_READ\_CHAR\_EVT (C++ enumerator), 176  
 ESP\_GATTC\_READ\_DESCR\_EVT (C++ enumerator), 176  
 ESP\_GATTC\_READ\_MULTIPLE\_EVT (C++ enumerator), 177  
 ESP\_GATTC\_REG\_EVT (C++ enumerator), 176  
 ESP\_GATTC\_REG\_FOR\_NOTIFY\_EVT (C++ enumerator), 177  
 ESP\_GATTC\_SCAN\_FLT\_CFG\_EVT (C++ enumerator), 177  
 ESP\_GATTC\_SCAN\_FLT\_PARAM\_EVT (C++ enumerator), 177  
 ESP\_GATTC\_SCAN\_FLT\_STATUS\_EVT (C++ enumerator), 177  
 ESP\_GATTC\_SEARCH\_CMPL\_EVT (C++ enumerator), 176  
 ESP\_GATTC\_SEARCH\_RES\_EVT (C++ enumerator), 176  
 esp\_gattc\_service\_elem\_t (C++ class), 143  
 esp\_gattc\_service\_elem\_t::end\_handle (C++ member), 143  
 esp\_gattc\_service\_elem\_t::is\_primary (C++ member), 143  
 esp\_gattc\_service\_elem\_t::start\_handle (C++ member), 143  
 esp\_gattc\_service\_elem\_t::uuid (C++ member), 143  
 ESP\_GATTC\_SET\_ASSOC\_EVT (C++ enumerator), 178  
 ESP\_GATTC\_SRVC\_CHG\_EVT (C++ enumerator), 176  
 ESP\_GATTC\_UNREG\_EVT (C++ enumerator), 176  
 ESP\_GATTC\_UNREG\_FOR\_NOTIFY\_EVT (C++ enumerator), 177  
 ESP\_GATTC\_WRITE\_CHAR\_EVT (C++ enumerator), 176  
 ESP\_GATTC\_WRITE\_DESCR\_EVT (C++ enumerator), 176  
 ESP\_GATTS\_ADD\_CHAR\_DESCR\_EVT (C++ enumerator), 161  
 ESP\_GATTS\_ADD\_CHAR\_EVT (C++ enumerator), 161  
 ESP\_GATTS\_ADD\_INCL\_SRVC\_EVT (C++ enumerator), 161  
 esp\_gatts\_attr\_db\_t (C++ class), 141  
 esp\_gatts\_attr\_db\_t::att\_desc (C++ member), 141

- esp\_gatts\_attr\_db\_t::attr\_control (C++ member), 141  
 ESP\_GATTS\_CANCEL\_OPEN\_EVT (C++ enumerator), 161  
 esp\_gatts\_cb\_event\_t (C++ enum), 160  
 esp\_gatts\_cb\_t (C++ type), 160  
 ESP\_GATTS\_CLOSE\_EVT (C++ enumerator), 161  
 ESP\_GATTS\_CONF\_EVT (C++ enumerator), 161  
 ESP\_GATTS\_CONGEST\_EVT (C++ enumerator), 161  
 ESP\_GATTS\_CONNECT\_EVT (C++ enumerator), 161  
 ESP\_GATTS\_CREAT\_ATTR\_TAB\_EVT (C++ enumerator), 162  
 ESP\_GATTS\_CREATE\_EVT (C++ enumerator), 161  
 ESP\_GATTS\_DELETE\_EVT (C++ enumerator), 161  
 ESP\_GATTS\_DISCONNECT\_EVT (C++ enumerator), 161  
 ESP\_GATTS\_EXEC\_WRITE\_EVT (C++ enumerator), 161  
 esp\_gatts\_incl128\_svc\_desc\_t (C++ class), 141  
 esp\_gatts\_incl128\_svc\_desc\_t::end\_hdl (C++ member), 141  
 esp\_gatts\_incl128\_svc\_desc\_t::start\_hdl (C++ member), 141  
 esp\_gatts\_incl\_svc\_desc\_t (C++ class), 141  
 esp\_gatts\_incl\_svc\_desc\_t::end\_hdl (C++ member), 141  
 esp\_gatts\_incl\_svc\_desc\_t::start\_hdl (C++ member), 141  
 esp\_gatts\_incl\_svc\_desc\_t::uuid (C++ member), 141  
 ESP\_GATTS\_LISTEN\_EVT (C++ enumerator), 161  
 ESP\_GATTS\_MTU\_EVT (C++ enumerator), 161  
 ESP\_GATTS\_OPEN\_EVT (C++ enumerator), 161  
 ESP\_GATTS\_READ\_EVT (C++ enumerator), 161  
 ESP\_GATTS\_REG\_EVT (C++ enumerator), 161  
 ESP\_GATTS\_RESPONSE\_EVT (C++ enumerator), 161  
 ESP\_GATTS\_SEND\_SERVICE\_CHANGE\_EVT (C++ enumerator), 162  
 ESP\_GATTS\_SET\_ATTR\_VAL\_EVT (C++ enumerator), 162  
 ESP\_GATTS\_START\_EVT (C++ enumerator), 161  
 ESP\_GATTS\_STOP\_EVT (C++ enumerator), 161  
 ESP\_GATTS\_UNREG\_EVT (C++ enumerator), 161  
 ESP\_GATTS\_WRITE\_EVT (C++ enumerator), 161  
 esp\_gcov\_dump (C++ function), 1098  
 esp\_get\_deep\_sleep\_wake\_stub (C++ function), 1111  
 esp\_get\_flash\_encryption\_mode (C++ function), 878  
 esp\_get\_free\_heap\_size (C++ function), 1136  
 esp\_get\_free\_internal\_heap\_size (C++ function), 1136  
 esp\_get\_idf\_version (C++ function), 1139  
 esp\_get\_minimum\_free\_heap\_size (C++ function), 1136  
 ESP\_HF\_AT\_RESPONSE\_CODE\_BLACKLISTED (C++ enumerator), 235  
 ESP\_HF\_AT\_RESPONSE\_CODE\_BUSY (C++ enumerator), 235  
 ESP\_HF\_AT\_RESPONSE\_CODE\_CME (C++ enumerator), 235  
 ESP\_HF\_AT\_RESPONSE\_CODE\_DELAYED (C++ enumerator), 235  
 ESP\_HF\_AT\_RESPONSE\_CODE\_ERR (C++ enumerator), 235  
 ESP\_HF\_AT\_RESPONSE\_CODE\_NO\_ANSWER (C++ enumerator), 235  
 ESP\_HF\_AT\_RESPONSE\_CODE\_NO\_CARRIER (C++ enumerator), 235  
 ESP\_HF\_AT\_RESPONSE\_CODE\_OK (C++ enumerator), 235  
 esp\_hf\_at\_response\_code\_t (C++ enum), 235  
 ESP\_HF\_BTRH\_CMD\_ACCEPT (C++ enumerator), 234  
 ESP\_HF\_BTRH\_CMD\_HOLD (C++ enumerator), 234  
 ESP\_HF\_BTRH\_CMD\_REJECT (C++ enumerator), 234  
 esp\_hf\_btrh\_cmd\_t (C++ enum), 234  
 ESP\_HF\_BTRH\_STATUS\_ACCEPTED (C++ enumerator), 234  
 ESP\_HF\_BTRH\_STATUS\_HELD (C++ enumerator), 234  
 ESP\_HF\_BTRH\_STATUS\_REJECTED (C++ enumerator), 234  
 esp\_hf\_btrh\_status\_t (C++ enum), 234  
 ESP\_HF\_CALL\_ADDR\_TYPE\_INTERNATIONAL (C++ enumerator), 234  
 esp\_hf\_call\_addr\_type\_t (C++ enum), 234  
 ESP\_HF\_CALL\_ADDR\_TYPE\_UNKNOWN (C++ enumerator), 234  
 ESP\_HF\_CALL\_HELD\_STATUS\_HELD (C++ enumerator), 233  
 ESP\_HF\_CALL\_HELD\_STATUS\_HELD\_AND\_ACTIVE (C++ enumerator), 233  
 ESP\_HF\_CALL\_HELD\_STATUS\_NONE (C++ enumerator), 233  
 esp\_hf\_call\_held\_status\_t (C++ enum), 233  
 ESP\_HF\_CALL\_SETUP\_STATUS\_INCOMING (C++ enumerator), 232  
 ESP\_HF\_CALL\_SETUP\_STATUS\_NONE (C++ enumerator), 232  
 ESP\_HF\_CALL\_SETUP\_STATUS\_OUTGOING\_ALERTING (C++ enumerator), 233  
 ESP\_HF\_CALL\_SETUP\_STATUS\_OUTGOING\_DIALING (C++ enumerator), 232  
 esp\_hf\_call\_setup\_status\_t (C++ enum), 232  
 ESP\_HF\_CALL\_STATUS\_CALL\_IN\_PROGRESS (C++ enumerator), 232  
 ESP\_HF\_CALL\_STATUS\_NO\_CALLS (C++ enumerator), 232  
 esp\_hf\_call\_status\_t (C++ enum), 232  
 ESP\_HF\_CHLD\_TYPE\_HOLD\_ACC (C++ enumerator), 235

- ESP\_HF\_CHLD\_TYPE\_MERGE (C++ *enumerator*), 235
- ESP\_HF\_CHLD\_TYPE\_MERGE\_DETACH (C++ *enumerator*), 235
- ESP\_HF\_CHLD\_TYPE\_PRIV\_X (C++ *enumerator*), 235
- ESP\_HF\_CHLD\_TYPE\_REL (C++ *enumerator*), 235
- ESP\_HF\_CHLD\_TYPE\_REL\_ACC (C++ *enumerator*), 235
- ESP\_HF\_CHLD\_TYPE\_REL\_X (C++ *enumerator*), 235
- esp\_hf\_chld\_type\_t (C++ *enum*), 235
- esp\_hf\_client\_answer\_call (C++ *function*), 239
- ESP\_HF\_CLIENT\_AT\_RESPONSE\_EVT (C++ *enumerator*), 248
- ESP\_HF\_CLIENT\_AUDIO\_STATE\_CONNECTED (C++ *enumerator*), 247
- ESP\_HF\_CLIENT\_AUDIO\_STATE\_CONNECTED\_MSBC (C++ *enumerator*), 247
- ESP\_HF\_CLIENT\_AUDIO\_STATE\_CONNECTING (C++ *enumerator*), 247
- ESP\_HF\_CLIENT\_AUDIO\_STATE\_DISCONNECTED (C++ *enumerator*), 247
- ESP\_HF\_CLIENT\_AUDIO\_STATE\_EVT (C++ *enumerator*), 247
- esp\_hf\_client\_audio\_state\_t (C++ *enum*), 246
- ESP\_HF\_CLIENT\_BINP\_EVT (C++ *enumerator*), 248
- ESP\_HF\_CLIENT\_BSIR\_EVT (C++ *enumerator*), 248
- ESP\_HF\_CLIENT\_BTRH\_EVT (C++ *enumerator*), 247
- ESP\_HF\_CLIENT\_BVRA\_EVT (C++ *enumerator*), 247
- esp\_hf\_client\_cb\_event\_t (C++ *enum*), 247
- esp\_hf\_client\_cb\_param\_t (C++ *union*), 241
- esp\_hf\_client\_cb\_param\_t::at\_response (C++ *member*), 241
- esp\_hf\_client\_cb\_param\_t::audio\_stat (C++ *member*), 241
- esp\_hf\_client\_cb\_param\_t::battery\_level (C++ *member*), 241
- esp\_hf\_client\_cb\_param\_t::binp (C++ *member*), 242
- esp\_hf\_client\_cb\_param\_t::bsir (C++ *member*), 242
- esp\_hf\_client\_cb\_param\_t::btrh (C++ *member*), 241
- esp\_hf\_client\_cb\_param\_t::bvra (C++ *member*), 241
- esp\_hf\_client\_cb\_param\_t::call (C++ *member*), 241
- esp\_hf\_client\_cb\_param\_t::call\_held (C++ *member*), 241
- esp\_hf\_client\_cb\_param\_t::call\_setup (C++ *member*), 241
- esp\_hf\_client\_cb\_param\_t::ccwa (C++ *member*), 241
- esp\_hf\_client\_cb\_param\_t::clcc (C++ *member*), 241
- esp\_hf\_client\_cb\_param\_t::clip (C++ *member*), 241
- esp\_hf\_client\_cb\_param\_t::cnum (C++ *member*), 242
- esp\_hf\_client\_cb\_param\_t::conn\_stat (C++ *member*), 241
- esp\_hf\_client\_cb\_param\_t::cops (C++ *member*), 241
- esp\_hf\_client\_cb\_param\_t::hf\_client\_at\_response\_p (C++ *class*), 242
- esp\_hf\_client\_cb\_param\_t::hf\_client\_at\_response\_p (C++ *member*), 242
- esp\_hf\_client\_cb\_param\_t::hf\_client\_at\_response\_p (C++ *member*), 242
- esp\_hf\_client\_cb\_param\_t::hf\_client\_audio\_stat\_pa (C++ *class*), 242
- esp\_hf\_client\_cb\_param\_t::hf\_client\_audio\_stat\_pa (C++ *member*), 242
- esp\_hf\_client\_cb\_param\_t::hf\_client\_audio\_stat\_pa (C++ *member*), 242
- esp\_hf\_client\_cb\_param\_t::hf\_client\_battery\_level (C++ *class*), 242
- esp\_hf\_client\_cb\_param\_t::hf\_client\_battery\_level (C++ *member*), 242
- esp\_hf\_client\_cb\_param\_t::hf\_client\_binp\_param (C++ *class*), 242
- esp\_hf\_client\_cb\_param\_t::hf\_client\_binp\_param::n (C++ *member*), 242
- esp\_hf\_client\_cb\_param\_t::hf\_client\_bsirparam (C++ *class*), 242
- esp\_hf\_client\_cb\_param\_t::hf\_client\_bsirparam::st (C++ *member*), 242
- esp\_hf\_client\_cb\_param\_t::hf\_client\_btrh\_param (C++ *class*), 242
- esp\_hf\_client\_cb\_param\_t::hf\_client\_btrh\_param::s (C++ *member*), 243
- esp\_hf\_client\_cb\_param\_t::hf\_client\_bvra\_param (C++ *class*), 243
- esp\_hf\_client\_cb\_param\_t::hf\_client\_bvra\_param::v (C++ *member*), 243
- esp\_hf\_client\_cb\_param\_t::hf\_client\_call\_held\_ind (C++ *class*), 243
- esp\_hf\_client\_cb\_param\_t::hf\_client\_call\_held\_ind (C++ *member*), 243
- esp\_hf\_client\_cb\_param\_t::hf\_client\_call\_ind\_para (C++ *class*), 243
- esp\_hf\_client\_cb\_param\_t::hf\_client\_call\_ind\_para (C++ *member*), 243
- esp\_hf\_client\_cb\_param\_t::hf\_client\_call\_setup\_in (C++ *class*), 243
- esp\_hf\_client\_cb\_param\_t::hf\_client\_call\_setup\_in (C++ *member*), 243
- esp\_hf\_client\_cb\_param\_t::hf\_client\_ccwa\_param (C++ *class*), 243

esp\_hf\_client\_cb\_param\_t::hf\_client\_ccwa\_param\_t::service\_availability  
 (C++ member), 243  
 esp\_hf\_client\_cb\_param\_t::hf\_client\_client\_cb\_param\_t::signal\_strength  
 (C++ class), 243  
 esp\_hf\_client\_cb\_param\_t::hf\_client\_client\_cb\_param\_t::volume\_control  
 (C++ member), 244  
 esp\_hf\_client\_cb\_param\_t::hf\_client\_client\_cb\_t (C++ type), 246  
 esp\_hf\_client\_cb\_param\_t::hf\_client\_client\_param\_t::empty  
 (C++ member), 244  
 esp\_hf\_client\_cb\_param\_t::hf\_client\_client\_param\_t::number  
 (C++ member), 244  
 esp\_hf\_client\_cb\_param\_t::hf\_client\_client\_param\_t::us  
 (C++ member), 244  
 esp\_hf\_client\_cb\_param\_t::hf\_client\_client\_param\_t::x  
 (C++ class), 244  
 esp\_hf\_client\_cb\_param\_t::hf\_client\_client\_param\_t::y  
 (C++ member), 244  
 esp\_hf\_client\_cb\_param\_t::hf\_client\_cnum\_param\_t  
 (C++ class), 244  
 esp\_hf\_client\_cb\_param\_t::hf\_client\_cnum\_param\_t::number  
 (C++ member), 244  
 esp\_hf\_client\_cb\_param\_t::hf\_client\_cnum\_param\_t::type  
 (C++ member), 244  
 esp\_hf\_client\_cb\_param\_t::hf\_client\_connection\_param\_t::cind  
 (C++ class), 244  
 esp\_hf\_client\_cb\_param\_t::hf\_client\_connection\_param\_t::cind\_call  
 (C++ member), 244  
 esp\_hf\_client\_cb\_param\_t::hf\_client\_connection\_param\_t::cind\_call\_held  
 (C++ member), 244  
 esp\_hf\_client\_cb\_param\_t::hf\_client\_connection\_param\_t::cind\_call\_setup  
 (C++ member), 244  
 esp\_hf\_client\_cb\_param\_t::hf\_client\_connection\_param\_t::cind\_roaming  
 (C++ member), 244  
 esp\_hf\_client\_cb\_param\_t::hf\_client\_connection\_param\_t::cind\_voice  
 (C++ member), 244  
 esp\_hf\_client\_cb\_param\_t::hf\_client\_connection\_param\_t::cind\_signal  
 (C++ member), 244  
 esp\_hf\_client\_cb\_param\_t::hf\_client\_current\_operator\_param\_t::current\_operator  
 (C++ class), 244  
 esp\_hf\_client\_cb\_param\_t::hf\_client\_current\_operator\_param\_t::event  
 (C++ member), 245  
 esp\_hf\_client\_cb\_param\_t::hf\_client\_network\_param\_t::network\_param  
 (C++ class), 245  
 esp\_hf\_client\_cb\_param\_t::hf\_client\_network\_param\_t::network\_param  
 (C++ member), 245  
 esp\_hf\_client\_cb\_param\_t::hf\_client\_service\_availability\_param  
 (C++ class), 245  
 esp\_hf\_client\_cb\_param\_t::hf\_client\_service\_availability\_param::status  
 (C++ member), 245  
 esp\_hf\_client\_cb\_param\_t::hf\_client\_signal\_strength  
 (C++ class), 245  
 esp\_hf\_client\_cb\_param\_t::hf\_client\_signal\_strength::value  
 (C++ member), 245  
 esp\_hf\_client\_cb\_param\_t::hf\_client\_volume\_control  
 (C++ class), 245  
 esp\_hf\_client\_cb\_param\_t::hf\_client\_volume\_control::type  
 (C++ member), 245  
 esp\_hf\_client\_cb\_param\_t::hf\_client\_volume\_control::volume  
 (C++ member), 245  
 esp\_hf\_client\_cb\_param\_t::roaming  
 (C++ member), 241

esp\_hf\_client::number  
 (C++ member), 241  
 esp\_hf\_client::signal\_strength  
 (C++ member), 241  
 esp\_hf\_client::volume\_control  
 (C++ member), 241  
 ESP\_HF\_CLIENT\_CCWA\_EVT (C++ enumerator),  
 247  
 ESP\_HF\_CLIENT\_CHLD\_FEAT\_HOLD\_ACC (C 宏),  
 246  
 ESP\_HF\_CLIENT\_CHLD\_FEAT\_MERGE (C 宏), 246  
 ESP\_HF\_CLIENT\_CHLD\_FEAT\_MERGE\_DETACH  
 (C 宏), 246  
 ESP\_HF\_CLIENT\_CHLD\_FEAT\_PRIV\_X (C 宏),  
 246  
 ESP\_HF\_CLIENT\_CHLD\_FEAT\_REL (C 宏), 246  
 ESP\_HF\_CLIENT\_CHLD\_FEAT\_REL\_ACC (C 宏),  
 246  
 ESP\_HF\_CLIENT\_CHLD\_FEAT\_REL\_X (C 宏), 246  
 ESP\_HF\_CLIENT\_CIND\_BATTERY\_LEVEL\_EVT  
 (C++ enumerator), 247  
 ESP\_HF\_CLIENT\_CIND\_CALL\_EVT (C++ enumer-  
 ator), 247  
 ESP\_HF\_CLIENT\_CIND\_CALL\_HELD\_EVT (C++  
 enumerator), 247  
 ESP\_HF\_CLIENT\_CIND\_CALL\_SETUP\_EVT  
 (C++ enumerator), 247  
 ESP\_HF\_CLIENT\_CIND\_ROAMING\_STATUS\_EVT  
 (C++ enumerator), 247  
 ESP\_HF\_CLIENT\_CIND\_VOICE\_AVAILABILITY\_EVT  
 (C++ enumerator), 247  
 ESP\_HF\_CLIENT\_CIND\_SIGNAL\_STRENGTH\_EVT  
 (C++ enumerator), 247  
 ESP\_HF\_CLIENT\_CURRENT\_OPERATOR\_EVT (C++  
 enumerator),  
 248  
 ESP\_HF\_CLIENT\_CURRENT\_OPERATOR\_EVT\_HANDLER  
 (C++ enumerator),  
 247  
 ESP\_HF\_CLIENT\_NETWORK\_PARAM\_EVT (C++  
 enumerator),  
 248  
 esp\_hf\_client\_connect\_audio (C++ func-  
 tion), 238  
 ESP\_HF\_CLIENT\_CONNECTION\_STATE\_CONNECTED  
 (C++ enumerator), 246  
 ESP\_HF\_CLIENT\_CONNECTION\_STATE\_CONNECTING  
 (C++ enumerator), 246  
 ESP\_HF\_CLIENT\_CONNECTION\_STATE\_DISCONNECTED  
 (C++ enumerator), 246  
 ESP\_HF\_CLIENT\_CONNECTION\_STATE\_DISCONNECTING  
 (C++ enumerator), 246  
 ESP\_HF\_CLIENT\_CONNECTION\_STATE\_EVT  
 (C++ enumerator), 247  
 ESP\_HF\_CLIENT\_CONNECTION\_STATE\_SLC\_CONNECTED  
 (C++ enumerator), 246  
 esp\_hf\_client\_connection\_state\_t (C++  
 enum), 246  
 ESP\_HF\_CLIENT\_COPS\_CURRENT\_OPERATOR\_EVT

- (C++ enumerator), 247
- esp\_hf\_client\_deinit (C++ function), 237
- esp\_hf\_client\_dial (C++ function), 238
- esp\_hf\_client\_dial\_memory (C++ function), 238
- esp\_hf\_client\_disconnect (C++ function), 237
- esp\_hf\_client\_disconnect\_audio (C++ function), 238
- esp\_hf\_client\_in\_band\_ring\_state\_t (C++ enum), 247
- ESP\_HF\_CLIENT\_IN\_BAND\_RINGTONE\_NOT\_PROVIDED (C++ enumerator), 247
- ESP\_HF\_CLIENT\_IN\_BAND\_RINGTONE\_PROVIDED (C++ enumerator), 247
- esp\_hf\_client\_incoming\_data\_cb\_t (C++ type), 246
- esp\_hf\_client\_init (C++ function), 237
- esp\_hf\_client\_outgoing\_data\_cb\_t (C++ type), 246
- esp\_hf\_client\_outgoing\_data\_ready (C++ function), 240
- esp\_hf\_client\_pcm\_resample (C++ function), 240
- esp\_hf\_client\_pcm\_resample\_deinit (C++ function), 240
- esp\_hf\_client\_pcm\_resample\_init (C++ function), 240
- ESP\_HF\_CLIENT\_PEER\_FEAT\_3WAY (C 宏), 245
- ESP\_HF\_CLIENT\_PEER\_FEAT\_CODEC (C 宏), 246
- ESP\_HF\_CLIENT\_PEER\_FEAT\_ECC (C 宏), 245
- ESP\_HF\_CLIENT\_PEER\_FEAT\_ECNR (C 宏), 245
- ESP\_HF\_CLIENT\_PEER\_FEAT\_ECS (C 宏), 245
- ESP\_HF\_CLIENT\_PEER\_FEAT\_EXTERR (C 宏), 245
- ESP\_HF\_CLIENT\_PEER\_FEAT\_INBAND (C 宏), 245
- ESP\_HF\_CLIENT\_PEER\_FEAT\_REJECT (C 宏), 245
- ESP\_HF\_CLIENT\_PEER\_FEAT\_VREC (C 宏), 245
- ESP\_HF\_CLIENT\_PEER\_FEAT\_VTAG (C 宏), 245
- esp\_hf\_client\_query\_current\_calls (C++ function), 239
- esp\_hf\_client\_query\_current\_operator\_name (C++ function), 239
- esp\_hf\_client\_register\_callback (C++ function), 237
- esp\_hf\_client\_register\_data\_callback (C++ function), 240
- esp\_hf\_client\_reject\_call (C++ function), 239
- esp\_hf\_client\_request\_last\_voice\_tag\_number (C++ function), 240
- esp\_hf\_client\_retrieve\_subscriber\_info (C++ function), 240
- ESP\_HF\_CLIENT\_RING\_IND\_EVT (C++ enumerator), 248
- esp\_hf\_client\_send\_btrh\_cmd (C++ function), 239
- esp\_hf\_client\_send\_chld\_cmd (C++ function), 239
- esp\_hf\_client\_send\_dtmf (C++ function), 240
- esp\_hf\_client\_start\_voice\_recognition (C++ function), 238
- esp\_hf\_client\_stop\_voice\_recognition (C++ function), 238
- ESP\_HF\_CLIENT\_VOLUME\_CONTROL\_EVT (C++ enumerator), 248
- esp\_hf\_client\_volume\_update (C++ function), 238
- ESP\_HF\_CME\_AG\_FAILURE (C++ enumerator), 236
- ESP\_HF\_CME\_DIAL\_STRING\_TOO\_LONG (C++ enumerator), 236
- esp\_hf\_cme\_err\_t (C++ enum), 235
- ESP\_HF\_CME\_INCORRECT\_PASSWORD (C++ enumerator), 236
- ESP\_HF\_CME\_INVALID\_CHARACTERS\_IN\_DIAL\_STRING (C++ enumerator), 236
- ESP\_HF\_CME\_INVALID\_CHARACTERS\_IN\_TEXT\_STRING (C++ enumerator), 236
- ESP\_HF\_CME\_INVALID\_INDEX (C++ enumerator), 236
- ESP\_HF\_CME\_MEMEORY\_FAILURE (C++ enumerator), 236
- ESP\_HF\_CME\_MEMEORY\_FULL (C++ enumerator), 236
- ESP\_HF\_CME\_NETWORK\_NOT\_ALLOWED (C++ enumerator), 236
- ESP\_HF\_CME\_NETWORK\_TIMEOUT (C++ enumerator), 236
- ESP\_HF\_CME\_NO\_CONNECTION\_TO\_PHONE (C++ enumerator), 236
- ESP\_HF\_CME\_NO\_NETWORK\_SERVICE (C++ enumerator), 236
- ESP\_HF\_CME\_OPERATION\_NOT\_ALLOWED (C++ enumerator), 236
- ESP\_HF\_CME\_OPERATION\_NOT\_SUPPORTED (C++ enumerator), 236
- ESP\_HF\_CME\_PH\_SIM\_PIN\_REQUIRED (C++ enumerator), 236
- ESP\_HF\_CME\_SIM\_BUSY (C++ enumerator), 236
- ESP\_HF\_CME\_SIM\_FAILURE (C++ enumerator), 236
- ESP\_HF\_CME\_SIM\_NOT\_INSERTED (C++ enumerator), 236
- ESP\_HF\_CME\_SIM\_PIN2\_REQUIRED (C++ enumerator), 236
- ESP\_HF\_CME\_SIM\_PIN\_REQUIRED (C++ enumerator), 236
- ESP\_HF\_CME\_SIM\_PUK2\_REQUIRED (C++ enumerator), 236
- ESP\_HF\_CME\_SIM\_PUK\_REQUIRED (C++ enumerator), 236
- ESP\_HF\_CME\_TEXT\_STRING\_TOO\_LONG (C++ enumerator), 236
- ESP\_HF\_CURRENT\_CALL\_DIRECTION\_INCOMING

- (C++ *enumerator*), 233
- ESP\_HF\_CURRENT\_CALL\_DIRECTION\_OUTGOING (C++ *enumerator*), 233
- esp\_hf\_current\_call\_direction\_t (C++ *enum*), 233
- ESP\_HF\_CURRENT\_CALL\_MODE\_DATA (C++ *enumerator*), 234
- ESP\_HF\_CURRENT\_CALL\_MODE\_FAX (C++ *enumerator*), 234
- esp\_hf\_current\_call\_mode\_t (C++ *enum*), 234
- ESP\_HF\_CURRENT\_CALL\_MODE\_VOICE (C++ *enumerator*), 234
- ESP\_HF\_CURRENT\_CALL\_MPTY\_TYPE\_MULTI (C++ *enumerator*), 234
- ESP\_HF\_CURRENT\_CALL\_MPTY\_TYPE\_SINGLE (C++ *enumerator*), 234
- esp\_hf\_current\_call\_mpty\_type\_t (C++ *enum*), 233
- ESP\_HF\_CURRENT\_CALL\_STATUS\_ACTIVE (C++ *enumerator*), 233
- ESP\_HF\_CURRENT\_CALL\_STATUS\_ALERTING (C++ *enumerator*), 233
- ESP\_HF\_CURRENT\_CALL\_STATUS\_DIALING (C++ *enumerator*), 233
- ESP\_HF\_CURRENT\_CALL\_STATUS\_HELD (C++ *enumerator*), 233
- ESP\_HF\_CURRENT\_CALL\_STATUS\_HELD\_BY\_RESPONDENT (C++ *enumerator*), 233
- ESP\_HF\_CURRENT\_CALL\_STATUS\_INCOMING (C++ *enumerator*), 233
- esp\_hf\_current\_call\_status\_t (C++ *enum*), 233
- ESP\_HF\_CURRENT\_CALL\_STATUS\_WAITING (C++ *enumerator*), 233
- ESP\_HF\_ROAMING\_STATUS\_ACTIVE (C++ *enumerator*), 232
- ESP\_HF\_ROAMING\_STATUS\_INACTIVE (C++ *enumerator*), 232
- esp\_hf\_roaming\_status\_t (C++ *enum*), 232
- ESP\_HF\_SERVICE\_AVAILABILITY\_STATUS\_AVAILABLE (C++ *enumerator*), 233
- esp\_hf\_service\_availability\_status\_t (C++ *enum*), 233
- ESP\_HF\_SERVICE\_AVAILABILITY\_STATUS\_UNAVAILABLE (C++ *enumerator*), 233
- ESP\_HF\_SUBSCRIBER\_SERVICE\_TYPE\_FAX (C++ *enumerator*), 234
- esp\_hf\_subscriber\_service\_type\_t (C++ *enum*), 234
- ESP\_HF\_SUBSCRIBER\_SERVICE\_TYPE\_UNKNOWN (C++ *enumerator*), 234
- ESP\_HF\_SUBSCRIBER\_SERVICE\_TYPE\_VOICE (C++ *enumerator*), 234
- ESP\_HF\_VOLUME\_CONTROL\_TARGET\_MIC (C++ *enumerator*), 232
- ESP\_HF\_VOLUME\_CONTROL\_TARGET\_SPK (C++ *enumerator*), 232
- esp\_hf\_volume\_control\_target\_t (C++ *enum*), 232
- ESP\_HF\_VR\_STATE\_DISABLED (C++ *enumerator*), 235
- ESP\_HF\_VR\_STATE\_ENABLED (C++ *enumerator*), 235
- esp\_hf\_vr\_state\_t (C++ *enum*), 235
- esp\_himem\_alloc (C++ *function*), 1053
- esp\_himem\_alloc\_map\_range (C++ *function*), 1053
- ESP\_HIMEM\_BLKSIZE (C 宏), 1054
- esp\_himem\_free (C++ *function*), 1054
- esp\_himem\_free\_map\_range (C++ *function*), 1054
- esp\_himem\_get\_free\_size (C++ *function*), 1054
- esp\_himem\_get\_phys\_size (C++ *function*), 1054
- esp\_himem\_handle\_t (C++ *type*), 1055
- esp\_himem\_map (C++ *function*), 1053
- ESP\_HIMEM\_MAPFLAG\_RO (C 宏), 1055
- esp\_himem\_rangehandle\_t (C++ *type*), 1055
- esp\_himem\_reserved\_area\_size (C++ *function*), 1054
- esp\_himem\_unmap (C++ *function*), 1054
- esp\_http\_client\_add\_auth (C++ *function*), 778
- esp\_http\_client\_auth\_type\_t (C++ *enum*), 781
- esp\_http\_client\_cleanup (C++ *function*), 777
- esp\_http\_client\_close (C++ *function*), 777
- esp\_http\_client\_config\_t (C++ *class*), 778
- esp\_http\_client\_config\_t::auth\_type (C++ *member*), 779
- esp\_http\_client\_config\_t::buffer\_size (C++ *member*), 779
- esp\_http\_client\_config\_t::buffer\_size\_tx (C++ *member*), 779
- esp\_http\_client\_config\_t::cert\_pem (C++ *member*), 779
- esp\_http\_client\_config\_t::client\_cert\_pem (C++ *member*), 779
- esp\_http\_client\_config\_t::client\_key\_pem (C++ *member*), 779
- esp\_http\_client\_config\_t::disable\_auto\_redirect (C++ *member*), 779
- esp\_http\_client\_config\_t::event\_handler (C++ *member*), 779
- esp\_http\_client\_config\_t::host (C++ *member*), 778
- esp\_http\_client\_config\_t::is\_async (C++ *member*), 779
- esp\_http\_client\_config\_t::keep\_alive\_count (C++ *member*), 780
- esp\_http\_client\_config\_t::keep\_alive\_enable (C++ *member*), 779
- esp\_http\_client\_config\_t::keep\_alive\_idle (C++ *member*), 779

- esp\_http\_client\_config\_t::keep\_alive\_interval (C++ member), 779  
 esp\_http\_client\_config\_t::max\_redirect\_count (C++ member), 779  
 esp\_http\_client\_config\_t::method (C++ member), 779  
 esp\_http\_client\_config\_t::password (C++ member), 779  
 esp\_http\_client\_config\_t::path (C++ member), 779  
 esp\_http\_client\_config\_t::port (C++ member), 778  
 esp\_http\_client\_config\_t::query (C++ member), 779  
 esp\_http\_client\_config\_t::skip\_cert\_check (C++ member), 779  
 esp\_http\_client\_config\_t::timeout\_ms (C++ member), 779  
 esp\_http\_client\_config\_t::transport\_type (C++ member), 779  
 esp\_http\_client\_config\_t::url (C++ member), 778  
 esp\_http\_client\_config\_t::use\_global\_ca\_store (C++ member), 779  
 esp\_http\_client\_config\_t::user\_agent (C++ member), 779  
 esp\_http\_client\_config\_t::user\_data (C++ member), 779  
 esp\_http\_client\_config\_t::username (C++ member), 778  
 esp\_http\_client\_delete\_header (C++ function), 776  
 esp\_http\_client\_event (C++ class), 778  
 esp\_http\_client\_event::client (C++ member), 778  
 esp\_http\_client\_event::data (C++ member), 778  
 esp\_http\_client\_event::data\_len (C++ member), 778  
 esp\_http\_client\_event::event\_id (C++ member), 778  
 esp\_http\_client\_event::header\_key (C++ member), 778  
 esp\_http\_client\_event::header\_value (C++ member), 778  
 esp\_http\_client\_event::user\_data (C++ member), 778  
 esp\_http\_client\_event\_handle\_t (C++ type), 780  
 esp\_http\_client\_event\_id\_t (C++ enum), 780  
 esp\_http\_client\_event\_t (C++ type), 780  
 esp\_http\_client\_fetch\_headers (C++ function), 776  
 esp\_http\_client\_get\_content\_length (C++ function), 777  
 esp\_http\_client\_get\_header (C++ function), 775  
 esp\_http\_client\_get\_password (C++ function), 775  
 esp\_http\_client\_get\_post\_field (C++ function), 774  
 esp\_http\_client\_get\_status\_code (C++ function), 777  
 esp\_http\_client\_get\_transport\_type (C++ function), 777  
 esp\_http\_client\_get\_username (C++ function), 775  
 esp\_http\_client\_handle\_t (C++ type), 780  
 esp\_http\_client\_init (C++ function), 773  
 esp\_http\_client\_is\_chunked\_response (C++ function), 776  
 esp\_http\_client\_is\_complete\_data\_received (C++ function), 778  
 esp\_http\_client\_method\_t (C++ enum), 781  
 esp\_http\_client\_open (C++ function), 776  
 esp\_http\_client\_perform (C++ function), 773  
 esp\_http\_client\_read (C++ function), 776  
 esp\_http\_client\_set\_header (C++ function), 774  
 esp\_http\_client\_set\_method (C++ function), 776  
 esp\_http\_client\_set\_password (C++ function), 775  
 esp\_http\_client\_set\_post\_field (C++ function), 774  
 esp\_http\_client\_set\_redirection (C++ function), 777  
 esp\_http\_client\_set\_url (C++ function), 774  
 esp\_http\_client\_set\_username (C++ function), 775  
 esp\_http\_client\_transport\_t (C++ enum), 781  
 esp\_http\_client\_write (C++ function), 776  
 esp\_https\_ota (C++ function), 1122  
 esp\_https\_ota\_begin (C++ function), 1123  
 esp\_https\_ota\_config\_t (C++ class), 1124  
 esp\_https\_ota\_config\_t::http\_config (C++ member), 1124  
 esp\_https\_ota\_finish (C++ function), 1123  
 esp\_https\_ota\_get\_image\_len\_read (C++ function), 1124  
 esp\_https\_ota\_get\_img\_desc (C++ function), 1124  
 esp\_https\_ota\_handle\_t (C++ type), 1124  
 esp\_https\_ota\_is\_complete\_data\_received (C++ function), 1123  
 esp\_https\_ota\_perform (C++ function), 1123  
 ESP\_IDF\_VERSION (C 宏), 1140  
 ESP\_IDF\_VERSION\_MAJOR (C 宏), 1140  
 ESP\_IDF\_VERSION\_MINOR (C 宏), 1140  
 ESP\_IDF\_VERSION\_PATCH (C 宏), 1140  
 ESP\_IDF\_VERSION\_VAL (C 宏), 1140  
 ESP\_IMAGE\_FLASH\_SIZE\_16MB (C++ enumerator), 1133  
 ESP\_IMAGE\_FLASH\_SIZE\_1MB (C++ enumerator),



- 1132
- ESP\_IMAGE\_FLASH\_SIZE\_2MB (C++ enumerator), 1132
- ESP\_IMAGE\_FLASH\_SIZE\_4MB (C++ enumerator), 1133
- ESP\_IMAGE\_FLASH\_SIZE\_8MB (C++ enumerator), 1133
- ESP\_IMAGE\_FLASH\_SIZE\_MAX (C++ enumerator), 1133
- esp\_image\_flash\_size\_t (C++ enum), 1132
- ESP\_IMAGE\_HEADER\_MAGIC (C 宏), 1132
- esp\_image\_header\_t (C++ class), 1130
- esp\_image\_header\_t::chip\_id (C++ member), 1130
- esp\_image\_header\_t::entry\_addr (C++ member), 1130
- esp\_image\_header\_t::hash\_appended (C++ member), 1131
- esp\_image\_header\_t::magic (C++ member), 1130
- esp\_image\_header\_t::min\_chip\_rev (C++ member), 1131
- esp\_image\_header\_t::reserved (C++ member), 1131
- esp\_image\_header\_t::segment\_count (C++ member), 1130
- esp\_image\_header\_t::spi\_mode (C++ member), 1130
- esp\_image\_header\_t::spi\_pin\_drv (C++ member), 1130
- esp\_image\_header\_t::spi\_size (C++ member), 1130
- esp\_image\_header\_t::spi\_speed (C++ member), 1130
- esp\_image\_header\_t::wp\_pin (C++ member), 1130
- ESP\_IMAGE\_MAX\_SEGMENTS (C 宏), 1132
- esp\_image\_segment\_header\_t (C++ class), 1131
- esp\_image\_segment\_header\_t::data\_len (C++ member), 1131
- esp\_image\_segment\_header\_t::load\_addr (C++ member), 1131
- esp\_image\_spi\_freq\_t (C++ enum), 1132
- ESP\_IMAGE\_SPI\_MODE\_DIO (C++ enumerator), 1132
- ESP\_IMAGE\_SPI\_MODE\_DOUT (C++ enumerator), 1132
- ESP\_IMAGE\_SPI\_MODE\_FAST\_READ (C++ enumerator), 1132
- ESP\_IMAGE\_SPI\_MODE\_QIO (C++ enumerator), 1132
- ESP\_IMAGE\_SPI\_MODE\_QOUT (C++ enumerator), 1132
- ESP\_IMAGE\_SPI\_MODE\_SLOW\_READ (C++ enumerator), 1132
- esp\_image\_spi\_mode\_t (C++ enum), 1132
- ESP\_IMAGE\_SPI\_SPEED\_20M (C++ enumerator), 1132
- ESP\_IMAGE\_SPI\_SPEED\_26M (C++ enumerator), 1132
- ESP\_IMAGE\_SPI\_SPEED\_40M (C++ enumerator), 1132
- ESP\_IMAGE\_SPI\_SPEED\_80M (C++ enumerator), 1132
- esp\_intr\_wdt\_init (C++ function), 1061
- esp\_intr\_alloc (C++ function), 1057
- esp\_intr\_alloc\_intrstatus (C++ function), 1057
- ESP\_INTR\_DISABLE (C 宏), 1060
- esp\_intr\_disable (C++ function), 1058
- ESP\_INTR\_ENABLE (C 宏), 1060
- esp\_intr\_enable (C++ function), 1058
- ESP\_INTR\_FLAG\_EDGE (C 宏), 1059
- ESP\_INTR\_FLAG\_HIGH (C 宏), 1059
- ESP\_INTR\_FLAG\_INTRDISABLED (C 宏), 1059
- ESP\_INTR\_FLAG\_IRAM (C 宏), 1059
- ESP\_INTR\_FLAG\_LEVEL1 (C 宏), 1059
- ESP\_INTR\_FLAG\_LEVEL2 (C 宏), 1059
- ESP\_INTR\_FLAG\_LEVEL3 (C 宏), 1059
- ESP\_INTR\_FLAG\_LEVEL4 (C 宏), 1059
- ESP\_INTR\_FLAG\_LEVEL5 (C 宏), 1059
- ESP\_INTR\_FLAG\_LEVEL6 (C 宏), 1059
- ESP\_INTR\_FLAG\_LEVELMASK (C 宏), 1059
- ESP\_INTR\_FLAG\_LOWMED (C 宏), 1059
- ESP\_INTR\_FLAG\_NMI (C 宏), 1059
- ESP\_INTR\_FLAG\_SHARED (C 宏), 1059
- esp\_intr\_free (C++ function), 1058
- esp\_intr\_get\_cpu (C++ function), 1058
- esp\_intr\_get\_intno (C++ function), 1058
- esp\_intr\_mark\_shared (C++ function), 1056
- esp\_intr\_noniram\_disable (C++ function), 1059
- esp\_intr\_noniram\_enable (C++ function), 1059
- esp\_intr\_reserve (C++ function), 1057
- esp\_intr\_set\_in\_iram (C++ function), 1058
- ESP\_IO\_CAP\_IN (C 宏), 132
- ESP\_IO\_CAP\_IO (C 宏), 131
- ESP\_IO\_CAP\_KBDISP (C 宏), 132
- ESP\_IO\_CAP\_NONE (C 宏), 132
- ESP\_IO\_CAP\_OUT (C 宏), 131
- esp\_ipc\_call (C++ function), 1074
- esp\_ipc\_call\_blocking (C++ function), 1074
- ESP\_LE\_AUTH\_BOND (C 宏), 131
- ESP\_LE\_AUTH\_NO\_BOND (C 宏), 131
- ESP\_LE\_AUTH\_REQ\_BOND\_MITM (C 宏), 131
- ESP\_LE\_AUTH\_REQ\_MITM (C 宏), 131
- ESP\_LE\_AUTH\_REQ\_SC\_BOND (C 宏), 131
- ESP\_LE\_AUTH\_REQ\_SC\_MITM (C 宏), 131
- ESP\_LE\_AUTH\_REQ\_SC\_MITM\_BOND (C 宏), 131
- ESP\_LE\_AUTH\_REQ\_SC\_ONLY (C 宏), 131
- ESP\_LE\_KEY\_LCSRK (C 宏), 131
- ESP\_LE\_KEY\_LENC (C 宏), 131
- ESP\_LE\_KEY\_LID (C 宏), 131
- ESP\_LE\_KEY\_LLK (C 宏), 131

- ESP\_LE\_KEY\_NONE (C 宏), 131
- ESP\_LE\_KEY\_PCSRK (C 宏), 131
- ESP\_LE\_KEY\_PENC (C 宏), 131
- ESP\_LE\_KEY\_PID (C 宏), 131
- ESP\_LE\_KEY\_PLK (C 宏), 131
- esp\_light\_sleep\_start (C++ function), 1110
- ESP\_LINE\_ENDINGS\_CR (C++ enumerator), 921
- ESP\_LINE\_ENDINGS\_CRLF (C++ enumerator), 921
- ESP\_LINE\_ENDINGS\_LF (C++ enumerator), 921
- esp\_line\_endings\_t (C++ enum), 921
- esp\_link\_key (C++ type), 110
- esp\_local\_ctrl\_add\_property (C++ function), 827
- esp\_local\_ctrl\_config (C++ class), 830
- esp\_local\_ctrl\_config::handlers (C++ member), 830
- esp\_local\_ctrl\_config::max\_properties (C++ member), 830
- esp\_local\_ctrl\_config::transport (C++ member), 830
- esp\_local\_ctrl\_config::transport\_config (C++ member), 830
- esp\_local\_ctrl\_config\_t (C++ type), 830
- esp\_local\_ctrl\_get\_property (C++ function), 827
- esp\_local\_ctrl\_get\_transport\_ble (C++ function), 826
- esp\_local\_ctrl\_get\_transport\_httpd (C++ function), 827
- esp\_local\_ctrl\_handlers (C++ class), 829
- esp\_local\_ctrl\_handlers::get\_prop\_value (C++ member), 829
- esp\_local\_ctrl\_handlers::set\_prop\_value (C++ member), 829
- esp\_local\_ctrl\_handlers::usr\_ctx (C++ member), 829
- esp\_local\_ctrl\_handlers::usr\_ctx\_free\_fn (C++ member), 830
- esp\_local\_ctrl\_handlers\_t (C++ type), 830
- esp\_local\_ctrl\_prop (C++ class), 828
- esp\_local\_ctrl\_prop::ctx (C++ member), 828
- esp\_local\_ctrl\_prop::ctx\_free\_fn (C++ member), 828
- esp\_local\_ctrl\_prop::flags (C++ member), 828
- esp\_local\_ctrl\_prop::name (C++ member), 828
- esp\_local\_ctrl\_prop::size (C++ member), 828
- esp\_local\_ctrl\_prop::type (C++ member), 828
- esp\_local\_ctrl\_prop\_t (C++ type), 830
- esp\_local\_ctrl\_prop\_val (C++ class), 828
- esp\_local\_ctrl\_prop\_val::data (C++ member), 829
- esp\_local\_ctrl\_prop\_val::free\_fn (C++ member), 829
- esp\_local\_ctrl\_prop\_val::size (C++ member), 829
- esp\_local\_ctrl\_prop\_val\_t (C++ type), 830
- esp\_local\_ctrl\_remove\_property (C++ function), 827
- esp\_local\_ctrl\_set\_handler (C++ function), 827
- esp\_local\_ctrl\_start (C++ function), 827
- esp\_local\_ctrl\_stop (C++ function), 827
- ESP\_LOCAL\_CTRL\_TRANSPORT\_BLE (C 宏), 830
- esp\_local\_ctrl\_transport\_config\_ble\_t (C++ type), 830
- esp\_local\_ctrl\_transport\_config\_httpd\_t (C++ type), 830
- esp\_local\_ctrl\_transport\_config\_t (C++ union), 828
- esp\_local\_ctrl\_transport\_config\_t::ble (C++ member), 828
- esp\_local\_ctrl\_transport\_config\_t::httpd (C++ member), 828
- ESP\_LOCAL\_CTRL\_TRANSPORT\_HTTPD (C 宏), 830
- esp\_local\_ctrl\_transport\_t (C++ type), 830
- ESP\_LOG\_BUFFER\_CHAR (C 宏), 1081
- ESP\_LOG\_BUFFER\_CHAR\_LEVEL (C 宏), 1080
- ESP\_LOG\_BUFFER\_HEX (C 宏), 1081
- ESP\_LOG\_BUFFER\_HEX\_LEVEL (C 宏), 1080
- ESP\_LOG\_BUFFER\_HEXDUMP (C 宏), 1081
- ESP\_LOG\_DEBUG (C++ enumerator), 1082
- ESP\_LOG\_EARLY\_IMPL (C 宏), 1081
- esp\_log\_early\_timestamp (C++ function), 1080
- ESP\_LOG\_ERROR (C++ enumerator), 1082
- ESP\_LOG\_INFO (C++ enumerator), 1082
- ESP\_LOG\_LEVEL (C 宏), 1082
- ESP\_LOG\_LEVEL\_LOCAL (C 宏), 1082
- esp\_log\_level\_set (C++ function), 1079
- esp\_log\_level\_t (C++ enum), 1082
- ESP\_LOG\_NONE (C++ enumerator), 1082
- esp\_log\_set\_vprintf (C++ function), 1080
- esp\_log\_timestamp (C++ function), 1080
- ESP\_LOG\_VERBOSE (C++ enumerator), 1082
- ESP\_LOG\_WARN (C++ enumerator), 1082
- esp\_log\_write (C++ function), 1080
- esp\_log\_writev (C++ function), 1080
- ESP\_LOGD (C 宏), 1082
- ESP\_LOGE (C 宏), 1081
- ESP\_LOGI (C 宏), 1082
- ESP\_LOGV (C 宏), 1082
- ESP\_LOGW (C 宏), 1082
- ESP\_MAC\_BT (C++ enumerator), 1139
- ESP\_MAC\_ETH (C++ enumerator), 1139
- esp\_mac\_type\_t (C++ enum), 1138
- ESP\_MAC\_WIFI\_SOFTAP (C++ enumerator), 1138
- ESP\_MAC\_WIFI\_STA (C++ enumerator), 1138
- esp\_mesh\_allow\_root\_conflicts (C++ function), 517

- `esp_mesh_available_txupQ_num` (C++ function), 516
- `esp_mesh_connect` (C++ function), 520
- `esp_mesh_deinit` (C++ function), 509
- `esp_mesh_delete_group_id` (C++ function), 517
- `esp_mesh_disconnect` (C++ function), 520
- `esp_mesh_fix_root` (C++ function), 518
- `esp_mesh_flush_scan_result` (C++ function), 520
- `esp_mesh_flush_upstream_packets` (C++ function), 519
- `esp_mesh_get_ap_assoc_expire` (C++ function), 515
- `esp_mesh_get_ap_authmode` (C++ function), 514
- `esp_mesh_get_ap_connections` (C++ function), 514
- `esp_mesh_get_capacity_num` (C++ function), 518
- `esp_mesh_get_config` (C++ function), 512
- `esp_mesh_get_group_list` (C++ function), 517
- `esp_mesh_get_group_num` (C++ function), 517
- `esp_mesh_get_id` (C++ function), 512
- `esp_mesh_get_ie_crypto_key` (C++ function), 518
- `esp_mesh_get_layer` (C++ function), 514
- `esp_mesh_get_max_layer` (C++ function), 513
- `esp_mesh_get_parent_bssid` (C++ function), 514
- `esp_mesh_get_root_healing_delay` (C++ function), 518
- `esp_mesh_get_router` (C++ function), 512
- `esp_mesh_get_router_bssid` (C++ function), 520
- `esp_mesh_get_routing_table` (C++ function), 516
- `esp_mesh_get_routing_table_size` (C++ function), 516
- `esp_mesh_get_rx_pending` (C++ function), 516
- `esp_mesh_get_self_organized` (C++ function), 514
- `esp_mesh_get_subnet_nodes_list` (C++ function), 520
- `esp_mesh_get_subnet_nodes_num` (C++ function), 520
- `esp_mesh_get_total_node_num` (C++ function), 516
- `esp_mesh_get_tsf_time` (C++ function), 521
- `esp_mesh_get_tx_pending` (C++ function), 516
- `esp_mesh_get_type` (C++ function), 513
- `esp_mesh_get_vote_percentage` (C++ function), 515
- `esp_mesh_get_xon_qsize` (C++ function), 517
- `esp_mesh_init` (C++ function), 508
- `esp_mesh_is_my_group` (C++ function), 517
- `esp_mesh_is_root` (C++ function), 514
- `esp_mesh_is_root_conflicts_allowed` (C++ function), 517
- `esp_mesh_is_root_fixed` (C++ function), 519
- `esp_mesh_post_toDS_state` (C++ function), 516
- `esp_mesh_recv` (C++ function), 510
- `esp_mesh_recv_toDS` (C++ function), 511
- `esp_mesh_scan_get_ap_ie_len` (C++ function), 519
- `esp_mesh_scan_get_ap_record` (C++ function), 519
- `esp_mesh_send` (C++ function), 509
- `esp_mesh_send_block_time` (C++ function), 510
- `esp_mesh_set_ap_assoc_expire` (C++ function), 515
- `esp_mesh_set_ap_authmode` (C++ function), 513
- `esp_mesh_set_ap_connections` (C++ function), 514
- `esp_mesh_set_ap_password` (C++ function), 513
- `esp_mesh_set_capacity_num` (C++ function), 517
- `esp_mesh_set_config` (C++ function), 511
- `esp_mesh_set_group_id` (C++ function), 517
- `esp_mesh_set_id` (C++ function), 512
- `esp_mesh_set_ie_crypto_funcs` (C++ function), 518
- `esp_mesh_set_ie_crypto_key` (C++ function), 518
- `esp_mesh_set_max_layer` (C++ function), 513
- `esp_mesh_set_parent` (C++ function), 519
- `esp_mesh_set_root_healing_delay` (C++ function), 518
- `esp_mesh_set_router` (C++ function), 512
- `esp_mesh_set_self_organized` (C++ function), 514
- `esp_mesh_set_type` (C++ function), 513
- `esp_mesh_set_vote_percentage` (C++ function), 515
- `esp_mesh_set_xon_qsize` (C++ function), 516
- `esp_mesh_start` (C++ function), 509
- `esp_mesh_stop` (C++ function), 509
- `esp_mesh_switch_channel` (C++ function), 520
- `esp_mesh_waive_root` (C++ function), 515
- `esp_mqtt_client_config_t` (C++ class), 814
- `esp_mqtt_client_config_t::alpn_protos` (C++ member), 816
- `esp_mqtt_client_config_t::buffer_size` (C++ member), 815
- `esp_mqtt_client_config_t::cert_len` (C++ member), 815
- `esp_mqtt_client_config_t::cert_pem` (C++ member), 815
- `esp_mqtt_client_config_t::client_cert_len` (C++ member), 815
- `esp_mqtt_client_config_t::client_cert_pem` (C++ member), 815

---

esp\_mqtt\_client\_config\_t::client\_id (C++ member), 815  
 esp\_mqtt\_client\_config\_t::client\_key\_len (C++ member), 816  
 esp\_mqtt\_client\_config\_t::client\_key\_pass (C++ member), 815  
 esp\_mqtt\_client\_config\_t::clientkey\_password (C++ member), 816  
 esp\_mqtt\_client\_config\_t::clientkey\_password\_len (C++ member), 816  
 esp\_mqtt\_client\_config\_t::disable\_auto\_reconnect (C++ member), 815  
 esp\_mqtt\_client\_config\_t::disable\_clear\_session (C++ member), 815  
 esp\_mqtt\_client\_config\_t::disable\_heartbeat (C++ member), 816  
 esp\_mqtt\_client\_config\_t::ds\_data (C++ member), 816  
 esp\_mqtt\_client\_config\_t::event\_handle (C++ member), 814  
 esp\_mqtt\_client\_config\_t::event\_loop\_handle (C++ member), 814  
 esp\_mqtt\_client\_config\_t::host (C++ member), 814  
 esp\_mqtt\_client\_config\_t::keepalive (C++ member), 815  
 esp\_mqtt\_client\_config\_t::lwt\_msg (C++ member), 815  
 esp\_mqtt\_client\_config\_t::lwt\_msg\_len (C++ member), 815  
 esp\_mqtt\_client\_config\_t::lwt\_qos (C++ member), 815  
 esp\_mqtt\_client\_config\_t::lwt\_retain (C++ member), 815  
 esp\_mqtt\_client\_config\_t::lwt\_topic (C++ member), 815  
 esp\_mqtt\_client\_config\_t::network\_timeout\_ms (C++ member), 816  
 esp\_mqtt\_client\_config\_t::out\_buffer\_size (C++ member), 816  
 esp\_mqtt\_client\_config\_t::password (C++ member), 815  
 esp\_mqtt\_client\_config\_t::port (C++ member), 814  
 esp\_mqtt\_client\_config\_t::protocol\_ver (C++ member), 816  
 esp\_mqtt\_client\_config\_t::psk\_hint\_key (C++ member), 816  
 esp\_mqtt\_client\_config\_t::reconnect\_timeout\_ms (C++ member), 816  
 esp\_mqtt\_client\_config\_t::refresh\_connection (C++ member), 816  
 esp\_mqtt\_client\_config\_t::skip\_cert\_common\_name\_check (C++ member), 816  
 esp\_mqtt\_client\_config\_t::task\_prio (C++ member), 815  
 esp\_mqtt\_client\_config\_t::task\_stack (C++ member), 815  
 esp\_mqtt\_client\_config\_t::transport (C++ member), 816  
 esp\_mqtt\_client\_config\_t::uri (C++ member), 814  
 esp\_mqtt\_client\_config\_t::use\_global\_ca\_store (C++ member), 816  
 esp\_mqtt\_client\_config\_t::use\_secure\_element (C++ member), 816  
 esp\_mqtt\_client\_config\_t::user\_context (C++ member), 815  
 esp\_mqtt\_client\_config\_t::username (C++ member), 815  
 esp\_mqtt\_client\_destroy (C++ function), 812  
 esp\_mqtt\_client\_disconnect (C++ function), 811  
 esp\_mqtt\_client\_get\_outbox\_size (C++ function), 813  
 esp\_mqtt\_client\_handle\_t (C++ type), 817  
 esp\_mqtt\_client\_init (C++ function), 811  
 esp\_mqtt\_client\_publish (C++ function), 812  
 esp\_mqtt\_client\_reconnect (C++ function), 811  
 esp\_mqtt\_client\_register\_event (C++ function), 813  
 esp\_mqtt\_client\_set\_uri (C++ function), 811  
 esp\_mqtt\_client\_start (C++ function), 811  
 esp\_mqtt\_client\_stop (C++ function), 812  
 esp\_mqtt\_client\_subscribe (C++ function), 812  
 esp\_mqtt\_client\_unsubscribe (C++ function), 812  
 esp\_mqtt\_connect\_return\_code\_t (C++ enum), 818  
 esp\_mqtt\_error\_codes (C++ class), 813  
 esp\_mqtt\_error\_codes::connect\_return\_code (C++ member), 814  
 esp\_mqtt\_error\_codes::error\_type (C++ member), 813  
 esp\_mqtt\_error\_codes::esp\_tls\_cert\_verify\_flags (C++ member), 813  
 esp\_mqtt\_error\_codes::esp\_tls\_last\_esp\_err (C++ member), 813  
 esp\_mqtt\_error\_codes::esp\_tls\_stack\_err (C++ member), 813  
 esp\_mqtt\_error\_codes::esp\_transport\_sock\_errno (C++ member), 814  
 esp\_mqtt\_error\_codes\_t (C++ type), 817  
 esp\_mqtt\_event\_type\_t (C++ enum), 818  
 esp\_mqtt\_event\_handle\_t (C++ type), 817  
 esp\_mqtt\_event\_id\_t (C++ enum), 817  
 esp\_mqtt\_event\_t (C++ class), 814  
 esp\_mqtt\_event\_t::client (C++ member), 814  
 esp\_mqtt\_event\_t::current\_data\_offset (C++ member), 814  
 esp\_mqtt\_event\_t::data (C++ member), 814  
 esp\_mqtt\_event\_t::data\_len (C++ member), 814

- esp\_mqtt\_event\_t::error\_handle (C++ member), 814  
 esp\_mqtt\_event\_t::event\_id (C++ member), 814  
 esp\_mqtt\_event\_t::msg\_id (C++ member), 814  
 esp\_mqtt\_event\_t::session\_present (C++ member), 814  
 esp\_mqtt\_event\_t::topic (C++ member), 814  
 esp\_mqtt\_event\_t::topic\_len (C++ member), 814  
 esp\_mqtt\_event\_t::total\_data\_len (C++ member), 814  
 esp\_mqtt\_event\_t::user\_context (C++ member), 814  
 esp\_mqtt\_protocol\_ver\_t (C++ enum), 818  
 esp\_mqtt\_set\_config (C++ function), 813  
 esp\_mqtt\_transport\_t (C++ enum), 818  
 esp\_netif\_netstack\_buf\_free (C++ function), 553  
 esp\_netif\_netstack\_buf\_ref (C++ function), 553  
 esp\_nimble\_hci\_and\_controller\_deinit (C++ function), 250  
 esp\_nimble\_hci\_and\_controller\_init (C++ function), 249  
 esp\_nimble\_hci\_deinit (C++ function), 250  
 esp\_nimble\_hci\_init (C++ function), 249  
 esp\_now\_add\_peer (C++ function), 500  
 esp\_now\_deinit (C++ function), 499  
 esp\_now\_del\_peer (C++ function), 500  
 ESP\_NOW\_ETH\_ALEN (C 宏), 502  
 esp\_now\_fetch\_peer (C++ function), 500  
 esp\_now\_get\_peer (C++ function), 500  
 esp\_now\_get\_peer\_num (C++ function), 501  
 esp\_now\_get\_version (C++ function), 499  
 esp\_now\_init (C++ function), 499  
 esp\_now\_is\_peer\_exist (C++ function), 501  
 ESP\_NOW\_KEY\_LEN (C 宏), 502  
 ESP\_NOW\_MAX\_DATA\_LEN (C 宏), 502  
 ESP\_NOW\_MAX\_ENCRYPT\_PEER\_NUM (C 宏), 502  
 ESP\_NOW\_MAX\_TOTAL\_PEER\_NUM (C 宏), 502  
 esp\_now\_mod\_peer (C++ function), 500  
 esp\_now\_peer\_info (C++ class), 501  
 esp\_now\_peer\_info::channel (C++ member), 501  
 esp\_now\_peer\_info::encrypt (C++ member), 501  
 esp\_now\_peer\_info::ifidx (C++ member), 501  
 esp\_now\_peer\_info::lmk (C++ member), 501  
 esp\_now\_peer\_info::peer\_addr (C++ member), 501  
 esp\_now\_peer\_info::priv (C++ member), 501  
 esp\_now\_peer\_info\_t (C++ type), 502  
 esp\_now\_peer\_num (C++ class), 502  
 esp\_now\_peer\_num::encrypt\_num (C++ member), 502  
 esp\_now\_peer\_num::total\_num (C++ member), 502  
 esp\_now\_peer\_num\_t (C++ type), 502  
 esp\_now\_recv\_cb\_t (C++ type), 502  
 esp\_now\_register\_recv\_cb (C++ function), 499  
 esp\_now\_register\_send\_cb (C++ function), 499  
 esp\_now\_send (C++ function), 499  
 esp\_now\_send\_cb\_t (C++ type), 503  
 ESP\_NOW\_SEND\_FAIL (C++ enumerator), 503  
 esp\_now\_send\_status\_t (C++ enum), 503  
 ESP\_NOW\_SEND\_SUCCESS (C++ enumerator), 503  
 esp\_now\_set\_pmk (C++ function), 501  
 esp\_now\_unregister\_recv\_cb (C++ function), 499  
 esp\_now\_unregister\_send\_cb (C++ function), 499  
 ESP\_OK (C 宏), 1127  
 ESP\_OK\_EFUSE\_CNT (C 宏), 1072  
 esp\_ota\_begin (C++ function), 1118  
 esp\_ota\_check\_rollback\_is\_possible (C++ function), 1121  
 esp\_ota\_end (C++ function), 1119  
 esp\_ota\_erase\_last\_boot\_app\_partition (C++ function), 1121  
 esp\_ota\_get\_app\_description (C++ function), 1118  
 esp\_ota\_get\_app\_elf\_sha256 (C++ function), 1118  
 esp\_ota\_get\_boot\_partition (C++ function), 1119  
 esp\_ota\_get\_last\_invalid\_partition (C++ function), 1120  
 esp\_ota\_get\_next\_update\_partition (C++ function), 1120  
 esp\_ota\_get\_partition\_description (C++ function), 1120  
 esp\_ota\_get\_running\_partition (C++ function), 1119  
 esp\_ota\_get\_state\_partition (C++ function), 1120  
 esp\_ota\_handle\_t (C++ type), 1121  
 esp\_ota\_mark\_app\_invalid\_rollback\_and\_reboot (C++ function), 1120  
 esp\_ota\_mark\_app\_valid\_cancel\_rollback (C++ function), 1120  
 esp\_ota\_set\_boot\_partition (C++ function), 1119  
 esp\_ota\_write (C++ function), 1118  
 esp\_partition\_check\_identity (C++ function), 875  
 esp\_partition\_deregister\_external (C++ function), 875  
 esp\_partition\_erase\_range (C++ function), 874  
 esp\_partition\_find (C++ function), 872  
 esp\_partition\_find\_first (C++ function),

- 872
- `esp_partition_get` (C++ function), 872
- `esp_partition_get_sha256` (C++ function), 874
- `esp_partition_iterator_release` (C++ function), 873
- `esp_partition_iterator_t` (C++ type), 876
- `esp_partition_mmap` (C++ function), 874
- `esp_partition_next` (C++ function), 873
- `esp_partition_read` (C++ function), 873
- `esp_partition_register_external` (C++ function), 875
- `ESP_PARTITION_SUBTYPE_ANY` (C++ enumerator), 878
- `ESP_PARTITION_SUBTYPE_APP_FACTORY` (C++ enumerator), 876
- `ESP_PARTITION_SUBTYPE_APP_OTA_0` (C++ enumerator), 876
- `ESP_PARTITION_SUBTYPE_APP_OTA_1` (C++ enumerator), 876
- `ESP_PARTITION_SUBTYPE_APP_OTA_10` (C++ enumerator), 877
- `ESP_PARTITION_SUBTYPE_APP_OTA_11` (C++ enumerator), 877
- `ESP_PARTITION_SUBTYPE_APP_OTA_12` (C++ enumerator), 877
- `ESP_PARTITION_SUBTYPE_APP_OTA_13` (C++ enumerator), 877
- `ESP_PARTITION_SUBTYPE_APP_OTA_14` (C++ enumerator), 877
- `ESP_PARTITION_SUBTYPE_APP_OTA_15` (C++ enumerator), 877
- `ESP_PARTITION_SUBTYPE_APP_OTA_2` (C++ enumerator), 877
- `ESP_PARTITION_SUBTYPE_APP_OTA_3` (C++ enumerator), 877
- `ESP_PARTITION_SUBTYPE_APP_OTA_4` (C++ enumerator), 877
- `ESP_PARTITION_SUBTYPE_APP_OTA_5` (C++ enumerator), 877
- `ESP_PARTITION_SUBTYPE_APP_OTA_6` (C++ enumerator), 877
- `ESP_PARTITION_SUBTYPE_APP_OTA_7` (C++ enumerator), 877
- `ESP_PARTITION_SUBTYPE_APP_OTA_8` (C++ enumerator), 877
- `ESP_PARTITION_SUBTYPE_APP_OTA_9` (C++ enumerator), 877
- `ESP_PARTITION_SUBTYPE_APP_OTA_MAX` (C++ enumerator), 877
- `ESP_PARTITION_SUBTYPE_APP_OTA_MIN` (C++ enumerator), 876
- `ESP_PARTITION_SUBTYPE_APP_TEST` (C++ enumerator), 877
- `ESP_PARTITION_SUBTYPE_DATA_COREDUMP` (C++ enumerator), 877
- `ESP_PARTITION_SUBTYPE_DATA_EFUSE_EM` (C++ enumerator), 877
- `ESP_PARTITION_SUBTYPE_DATA_ESPHTTTPD` (C++ enumerator), 877
- `ESP_PARTITION_SUBTYPE_DATA_FAT` (C++ enumerator), 878
- `ESP_PARTITION_SUBTYPE_DATA_NVS` (C++ enumerator), 877
- `ESP_PARTITION_SUBTYPE_DATA_NVS_KEYS` (C++ enumerator), 877
- `ESP_PARTITION_SUBTYPE_DATA_OTA` (C++ enumerator), 877
- `ESP_PARTITION_SUBTYPE_DATA_PHY` (C++ enumerator), 877
- `ESP_PARTITION_SUBTYPE_DATA_SPIFFS` (C++ enumerator), 878
- `ESP_PARTITION_SUBTYPE_OTA` (C 宏), 876
- `esp_partition_subtype_t` (C++ enum), 876
- `esp_partition_t` (C++ class), 875
- `esp_partition_t::address` (C++ member), 876
- `esp_partition_t::encrypted` (C++ member), 876
- `esp_partition_t::flash_chip` (C++ member), 876
- `esp_partition_t::label` (C++ member), 876
- `esp_partition_t::size` (C++ member), 876
- `esp_partition_t::subtype` (C++ member), 876
- `esp_partition_t::type` (C++ member), 876
- `ESP_PARTITION_TYPE_APP` (C++ enumerator), 876
- `ESP_PARTITION_TYPE_DATA` (C++ enumerator), 876
- `esp_partition_type_t` (C++ enum), 876
- `esp_partition_verify` (C++ function), 873
- `esp_partition_write` (C++ function), 873
- `ESP_PD_DOMAIN_MAX` (C++ enumerator), 1111
- `ESP_PD_DOMAIN_RTC_FAST_MEM` (C++ enumerator), 1111
- `ESP_PD_DOMAIN_RTC_PERIPH` (C++ enumerator), 1111
- `ESP_PD_DOMAIN_RTC_SLOW_MEM` (C++ enumerator), 1111
- `ESP_PD_DOMAIN_XTAL` (C++ enumerator), 1111
- `ESP_PD_OPTION_AUTO` (C++ enumerator), 1112
- `ESP_PD_OPTION_OFF` (C++ enumerator), 1111
- `ESP_PD_OPTION_ON` (C++ enumerator), 1111
- `ESP_PM_APB_FREQ_MAX` (C++ enumerator), 1103
- `esp_pm_config_esp32_t` (C++ class), 1104
- `esp_pm_config_esp32_t::light_sleep_enable` (C++ member), 1104
- `esp_pm_config_esp32_t::max_freq_mhz` (C++ member), 1104
- `esp_pm_config_esp32_t::min_freq_mhz` (C++ member), 1104
- `esp_pm_configure` (C++ function), 1102
- `ESP_PM_CPU_FREQ_MAX` (C++ enumerator), 1103
- `esp_pm_dump_locks` (C++ function), 1103
- `esp_pm_lock_acquire` (C++ function), 1102

- esp\_pm\_lock\_create (C++ function), 1102  
 esp\_pm\_lock\_delete (C++ function), 1103  
 esp\_pm\_lock\_handle\_t (C++ type), 1103  
 esp\_pm\_lock\_release (C++ function), 1102  
 esp\_pm\_lock\_type\_t (C++ enum), 1103  
 ESP\_PM\_NO\_LIGHT\_SLEEP (C++ enumerator), 1104  
 esp\_power\_level\_t (C++ enum), 107  
 esp\_pthread\_cfg\_t (C++ class), 1126  
 esp\_pthread\_cfg\_t::inherit\_cfg (C++ member), 1126  
 esp\_pthread\_cfg\_t::pin\_to\_core (C++ member), 1126  
 esp\_pthread\_cfg\_t::prio (C++ member), 1126  
 esp\_pthread\_cfg\_t::stack\_size (C++ member), 1126  
 esp\_pthread\_cfg\_t::thread\_name (C++ member), 1126  
 esp\_pthread\_get\_cfg (C++ function), 1126  
 esp\_pthread\_get\_default\_config (C++ function), 1125  
 esp\_pthread\_set\_cfg (C++ function), 1125  
 ESP\_PWR\_LVL\_N0 (C++ enumerator), 108  
 ESP\_PWR\_LVL\_N11 (C++ enumerator), 108  
 ESP\_PWR\_LVL\_N12 (C++ enumerator), 108  
 ESP\_PWR\_LVL\_N14 (C++ enumerator), 108  
 ESP\_PWR\_LVL\_N2 (C++ enumerator), 108  
 ESP\_PWR\_LVL\_N3 (C++ enumerator), 108  
 ESP\_PWR\_LVL\_N5 (C++ enumerator), 108  
 ESP\_PWR\_LVL\_N6 (C++ enumerator), 108  
 ESP\_PWR\_LVL\_N8 (C++ enumerator), 108  
 ESP\_PWR\_LVL\_N9 (C++ enumerator), 108  
 ESP\_PWR\_LVL\_P1 (C++ enumerator), 108  
 ESP\_PWR\_LVL\_P3 (C++ enumerator), 108  
 ESP\_PWR\_LVL\_P4 (C++ enumerator), 108  
 ESP\_PWR\_LVL\_P6 (C++ enumerator), 108  
 ESP\_PWR\_LVL\_P7 (C++ enumerator), 108  
 ESP\_PWR\_LVL\_P9 (C++ enumerator), 108  
 esp\_random (C++ function), 1136  
 esp\_read\_mac (C++ function), 1137  
 esp\_register\_freertos\_idle\_hook (C++ function), 1032  
 esp\_register\_freertos\_idle\_hook\_for\_cpu (C++ function), 1031  
 esp\_register\_freertos\_tick\_hook (C++ function), 1032  
 esp\_register\_freertos\_tick\_hook\_for\_cpu (C++ function), 1032  
 esp\_register\_shutdown\_handler (C++ function), 1136  
 esp\_reset\_reason (C++ function), 1136  
 esp\_reset\_reason\_t (C++ enum), 1139  
 esp\_restart (C++ function), 1136  
 ESP\_RST\_BROWNOUT (C++ enumerator), 1139  
 ESP\_RST\_DEEPSLEEP (C++ enumerator), 1139  
 ESP\_RST\_EXT (C++ enumerator), 1139  
 ESP\_RST\_INT\_WDT (C++ enumerator), 1139  
 ESP\_RST\_PANIC (C++ enumerator), 1139  
 ESP\_RST\_POWERON (C++ enumerator), 1139  
 ESP\_RST\_SDIO (C++ enumerator), 1139  
 ESP\_RST\_SW (C++ enumerator), 1139  
 ESP\_RST\_TASK\_WDT (C++ enumerator), 1139  
 ESP\_RST\_UNKNOWN (C++ enumerator), 1139  
 ESP\_RST\_WDT (C++ enumerator), 1139  
 ESP\_SCO\_DATA\_PATH\_HCI (C++ enumerator), 108  
 ESP\_SCO\_DATA\_PATH\_PCM (C++ enumerator), 108  
 esp\_sco\_data\_path\_t (C++ enum), 108  
 esp\_service\_source\_t (C++ enum), 149  
 esp\_set\_deep\_sleep\_wake\_stub (C++ function), 1110  
 esp\_sleep\_disable\_wakeup\_source (C++ function), 1108  
 esp\_sleep\_enable\_ext0\_wakeup (C++ function), 1108  
 esp\_sleep\_enable\_ext1\_wakeup (C++ function), 1109  
 esp\_sleep\_enable\_gpio\_wakeup (C++ function), 1109  
 esp\_sleep\_enable\_timer\_wakeup (C++ function), 1108  
 esp\_sleep\_enable\_touchpad\_wakeup (C++ function), 1108  
 esp\_sleep\_enable\_uart\_wakeup (C++ function), 1109  
 esp\_sleep\_enable\_ulp\_wakeup (C++ function), 1108  
 esp\_sleep\_ext1\_wakeup\_mode\_t (C++ enum), 1111  
 esp\_sleep\_get\_ext1\_wakeup\_status (C++ function), 1109  
 esp\_sleep\_get\_touchpad\_wakeup\_status (C++ function), 1108  
 esp\_sleep\_get\_wakeup\_cause (C++ function), 1110  
 esp\_sleep\_pd\_config (C++ function), 1110  
 esp\_sleep\_pd\_domain\_t (C++ enum), 1111  
 esp\_sleep\_pd\_option\_t (C++ enum), 1111  
 esp\_sleep\_source\_t (C++ enum), 1112  
 ESP\_SLEEP\_WAKEUP\_ALL (C++ enumerator), 1112  
 esp\_sleep\_wakeup\_cause\_t (C++ type), 1111  
 ESP\_SLEEP\_WAKEUP\_EXT0 (C++ enumerator), 1112  
 ESP\_SLEEP\_WAKEUP\_EXT1 (C++ enumerator), 1112  
 ESP\_SLEEP\_WAKEUP\_GPIO (C++ enumerator), 1112  
 ESP\_SLEEP\_WAKEUP\_TIMER (C++ enumerator), 1112  
 ESP\_SLEEP\_WAKEUP\_TOUCHPAD (C++ enumerator), 1112  
 ESP\_SLEEP\_WAKEUP\_UART (C++ enumerator), 1112  
 ESP\_SLEEP\_WAKEUP\_ULP (C++ enumerator), 1112  
 ESP\_SLEEP\_WAKEUP\_UNDEFINED (C++ enumerator), 1112

- esp\_smartconfig\_fast\_mode (C++ function), 496  
 esp\_smartconfig\_get\_version (C++ function), 495  
 esp\_smartconfig\_set\_type (C++ function), 495  
 esp\_smartconfig\_start (C++ function), 495  
 esp\_smartconfig\_stop (C++ function), 495  
 esp\_spiffs\_format (C++ function), 932  
 esp\_spiffs\_info (C++ function), 932  
 esp\_spiffs\_mounted (C++ function), 931  
 ESP\_SPP\_BUSY (C++ enumerator), 230  
 esp\_spp\_cb\_event\_t (C++ enum), 231  
 esp\_spp\_cb\_param\_t (C++ union), 226  
 esp\_spp\_cb\_param\_t::cl\_init (C++ member), 227  
 esp\_spp\_cb\_param\_t::close (C++ member), 226  
 esp\_spp\_cb\_param\_t::cong (C++ member), 227  
 esp\_spp\_cb\_param\_t::data\_ind (C++ member), 227  
 esp\_spp\_cb\_param\_t::disc\_comp (C++ member), 226  
 esp\_spp\_cb\_param\_t::init (C++ member), 226  
 esp\_spp\_cb\_param\_t::open (C++ member), 226  
 esp\_spp\_cb\_param\_t::spp\_cl\_init\_evt\_param (C++ class), 227  
 esp\_spp\_cb\_param\_t::spp\_cl\_init\_evt\_param::handle (C++ member), 227  
 esp\_spp\_cb\_param\_t::spp\_cl\_init\_evt\_param::spp\_cl (C++ member), 227  
 esp\_spp\_cb\_param\_t::spp\_cl\_init\_evt\_param::status (C++ member), 227  
 esp\_spp\_cb\_param\_t::spp\_cl\_init\_evt\_param::spp\_close\_evt\_param (C++ member), 227  
 esp\_spp\_cb\_param\_t::spp\_cl\_init\_evt\_param::spp\_cong\_evt\_param (C++ member), 227  
 esp\_spp\_cb\_param\_t::spp\_cl\_init\_evt\_param::spp\_data\_ind\_evt\_param (C++ member), 228  
 esp\_spp\_cb\_param\_t::spp\_cl\_init\_evt\_param::spp\_disc\_comp (C++ member), 226  
 esp\_spp\_cb\_param\_t::spp\_cl\_init\_evt\_param::spp\_init\_evt\_param (C++ member), 228  
 esp\_spp\_cb\_param\_t::spp\_cl\_init\_evt\_param::spp\_open\_evt\_param (C++ member), 228  
 esp\_spp\_cb\_param\_t::spp\_cl\_init\_evt\_param::spp\_open\_evt\_param::fd (C++ member), 228  
 esp\_spp\_cb\_param\_t::spp\_cl\_init\_evt\_param::spp\_open\_evt\_param::handle (C++ member), 228  
 esp\_spp\_cb\_param\_t::spp\_cl\_init\_evt\_param::spp\_open\_evt\_param::rem\_bda (C++ member), 228  
 esp\_spp\_cb\_param\_t::spp\_cl\_init\_evt\_param::spp\_open\_evt\_param::status (C++ member), 228  
 esp\_spp\_cb\_param\_t::spp\_cl\_init\_evt\_param::spp\_srv\_open\_evt\_param (C++ class), 229  
 esp\_spp\_cb\_param\_t::spp\_cl\_init\_evt\_param::spp\_srv\_open\_evt\_param::fd (C++ member), 229  
 esp\_spp\_cb\_param\_t::spp\_cl\_init\_evt\_param::spp\_srv\_open\_evt\_param::handle (C++ member), 229  
 esp\_spp\_cb\_param\_t::spp\_cl\_init\_evt\_param::spp\_srv\_open\_evt\_param::new\_l (C++ member), 229  
 esp\_spp\_cb\_param\_t::spp\_cl\_init\_evt\_param::spp\_srv\_open\_evt\_param::rem\_b (C++ member), 229  
 esp\_spp\_cb\_param\_t::spp\_cl\_init\_evt\_param::spp\_srv\_open\_evt\_param::status (C++ member), 229  
 esp\_spp\_cb\_param\_t::spp\_close\_evt\_param (C++ class), 227  
 esp\_spp\_cb\_param\_t::spp\_close\_evt\_param::handle (C++ member), 227  
 esp\_spp\_cb\_param\_t::spp\_close\_evt\_param::status (C++ member), 227  
 esp\_spp\_cb\_param\_t::spp\_cong\_evt\_param (C++ class), 227  
 esp\_spp\_cb\_param\_t::spp\_cong\_evt\_param::handle (C++ member), 227  
 esp\_spp\_cb\_param\_t::spp\_cong\_evt\_param::status (C++ member), 227  
 esp\_spp\_cb\_param\_t::spp\_data\_ind\_evt\_param (C++ class), 228  
 esp\_spp\_cb\_param\_t::spp\_data\_ind\_evt\_param::data (C++ member), 228  
 esp\_spp\_cb\_param\_t::spp\_data\_ind\_evt\_param::handle (C++ member), 228  
 esp\_spp\_cb\_param\_t::spp\_data\_ind\_evt\_param::len (C++ member), 228  
 esp\_spp\_cb\_param\_t::spp\_data\_ind\_evt\_param::status (C++ member), 228  
 esp\_spp\_cb\_param\_t::spp\_discovery\_comp\_evt\_param (C++ class), 228  
 esp\_spp\_cb\_param\_t::spp\_discovery\_comp\_evt\_param::handle (C++ member), 228  
 esp\_spp\_cb\_param\_t::spp\_discovery\_comp\_evt\_param::status (C++ member), 228  
 esp\_spp\_cb\_param\_t::spp\_init\_evt\_param (C++ class), 228  
 esp\_spp\_cb\_param\_t::spp\_init\_evt\_param::status (C++ member), 228  
 esp\_spp\_cb\_param\_t::spp\_open\_evt\_param (C++ class), 228  
 esp\_spp\_cb\_param\_t::spp\_open\_evt\_param::fd (C++ member), 228  
 esp\_spp\_cb\_param\_t::spp\_open\_evt\_param::handle (C++ member), 228  
 esp\_spp\_cb\_param\_t::spp\_open\_evt\_param::rem\_bda (C++ member), 228  
 esp\_spp\_cb\_param\_t::spp\_open\_evt\_param::status (C++ member), 228  
 esp\_spp\_cb\_param\_t::spp\_srv\_open\_evt\_param (C++ class), 229  
 esp\_spp\_cb\_param\_t::spp\_srv\_open\_evt\_param::fd (C++ member), 229  
 esp\_spp\_cb\_param\_t::spp\_srv\_open\_evt\_param::handle (C++ member), 229  
 esp\_spp\_cb\_param\_t::spp\_srv\_open\_evt\_param::new\_l (C++ member), 229  
 esp\_spp\_cb\_param\_t::spp\_srv\_open\_evt\_param::rem\_b (C++ member), 229  
 esp\_spp\_cb\_param\_t::spp\_srv\_open\_evt\_param::status (C++ member), 229  
 esp\_spp\_cb\_param\_t::spp\_srv\_stop\_evt\_param (C++ class), 229  
 esp\_spp\_cb\_param\_t::spp\_srv\_stop\_evt\_param::scn (C++ member), 229  
 esp\_spp\_cb\_param\_t::spp\_srv\_stop\_evt\_param::status (C++ member), 229  
 esp\_spp\_cb\_param\_t::spp\_start\_evt\_param (C++ class), 229  
 esp\_spp\_cb\_param\_t::spp\_start\_evt\_param::handle (C++ member), 229  
 esp\_spp\_cb\_param\_t::spp\_start\_evt\_param::scn (C++ member), 229  
 esp\_spp\_cb\_param\_t::spp\_start\_evt\_param::sec\_id (C++ member), 229



- esp\_spp\_cb\_param\_t::spp\_start\_evt\_param (C++ member), 229  
 esp\_spp\_cb\_param\_t::spp\_start\_evt\_param (C++ member), 229  
 esp\_spp\_cb\_param\_t::spp\_uninit\_evt\_param (C++ class), 229  
 esp\_spp\_cb\_param\_t::spp\_uninit\_evt\_param (C++ member), 229  
 esp\_spp\_cb\_param\_t::spp\_write\_evt\_param (C++ class), 229  
 esp\_spp\_cb\_param\_t::spp\_write\_evt\_param (C++ member), 230  
 esp\_spp\_cb\_param\_t::spp\_write\_evt\_param (C++ member), 230  
 esp\_spp\_cb\_param\_t::spp\_write\_evt\_param (C++ member), 230  
 esp\_spp\_cb\_param\_t::spp\_write\_evt\_param (C++ member), 230  
 esp\_spp\_cb\_param\_t::srv\_open (C++ member), 226  
 esp\_spp\_cb\_param\_t::srv\_stop (C++ member), 227  
 esp\_spp\_cb\_param\_t::start (C++ member), 226  
 esp\_spp\_cb\_param\_t::uninit (C++ member), 226  
 esp\_spp\_cb\_param\_t::write (C++ member), 227  
 esp\_spp\_cb\_t (C++ type), 230  
 ESP\_SPP\_CL\_INIT\_EVT (C++ enumerator), 231  
 ESP\_SPP\_CLOSE\_EVT (C++ enumerator), 231  
 ESP\_SPP\_CONG\_EVT (C++ enumerator), 231  
 esp\_spp\_connect (C++ function), 225  
 ESP\_SPP\_DATA\_IND\_EVT (C++ enumerator), 231  
 esp\_spp\_deinit (C++ function), 224  
 esp\_spp\_disconnect (C++ function), 225  
 ESP\_SPP\_DISCOVERY\_COMP\_EVT (C++ enumerator), 231  
 ESP\_SPP\_FAILURE (C++ enumerator), 230  
 esp\_spp\_init (C++ function), 224  
 ESP\_SPP\_INIT\_EVT (C++ enumerator), 231  
 ESP\_SPP\_MAX\_MTU (C 宏), 230  
 ESP\_SPP\_MAX\_SCN (C 宏), 230  
 ESP\_SPP\_MODE\_CB (C++ enumerator), 231  
 esp\_spp\_mode\_t (C++ enum), 231  
 ESP\_SPP\_MODE\_VFS (C++ enumerator), 231  
 ESP\_SPP\_NEED\_DEINIT (C++ enumerator), 231  
 ESP\_SPP\_NEED\_INIT (C++ enumerator), 231  
 ESP\_SPP\_NO\_CONNECTION (C++ enumerator), 231  
 ESP\_SPP\_NO\_DATA (C++ enumerator), 231  
 ESP\_SPP\_NO\_RESOURCE (C++ enumerator), 231  
 ESP\_SPP\_NO\_SERVER (C++ enumerator), 231  
 ESP\_SPP\_OPEN\_EVT (C++ enumerator), 231  
 esp\_spp\_register\_callback (C++ function), 224  
 ESP\_SPP\_ROLE\_MASTER (C++ enumerator), 231  
 ESP\_SPP\_ROLE\_SLAVE (C++ enumerator), 231  
 esp\_spp\_role\_t (C++ enum), 231  
 ESP\_SPP\_SEC\_AUTHENTICATE (C 宏), 230  
 ESP\_SPP\_SEC\_AUTHORIZE (C 宏), 230  
 ESP\_SPP\_SEC\_ENCRYPT (C 宏), 230  
 ESP\_SPP\_SEC\_IN\_16\_DIGITS (C 宏), 230  
 ESP\_SPP\_SEC\_MITM (C 宏), 230  
 ESP\_SPP\_SEC\_MODE4\_LEVEL4 (C 宏), 230  
 ESP\_SPP\_SEC\_NONE (C 宏), 230  
 esp\_spp\_sec\_t (C++ type), 230  
 ESP\_SPP\_SRV\_OPEN\_EVT (C++ enumerator), 232  
 ESP\_SPP\_SRV\_STOP\_EVT (C++ enumerator), 232  
 esp\_spp\_start\_discovery (C++ function), 225  
 ESP\_SPP\_START\_EVT (C++ enumerator), 231  
 esp\_spp\_start\_srv (C++ function), 225  
 esp\_spp\_status\_t (C++ enum), 230  
 esp\_spp\_stop\_srv (C++ function), 225  
 esp\_spp\_stop\_srv\_scn (C++ function), 226  
 ESP\_SPP\_SUCCESS (C++ enumerator), 230  
 ESP\_SPP\_UNINIT\_EVT (C++ enumerator), 231  
 esp\_spp\_vfs\_register (C++ function), 226  
 esp\_spp\_write (C++ function), 226  
 ESP\_SPP\_WRITE\_EVT (C++ enumerator), 231  
 esp\_sysview\_flush (C++ function), 1099  
 esp\_sysview\_heap\_trace\_alloc (C++ function), 1099  
 esp\_sysview\_heap\_trace\_free (C++ function), 1099  
 esp\_sysview\_heap\_trace\_start (C++ function), 1099  
 esp\_sysview\_heap\_trace\_stop (C++ function), 1099  
 esp\_sysview\_vprintf (C++ function), 1099  
 esp\_task\_wdt\_add (C++ function), 1062  
 esp\_task\_wdt\_deinit (C++ function), 1062  
 esp\_task\_wdt\_delete (C++ function), 1062  
 esp\_task\_wdt\_init (C++ function), 1061  
 esp\_task\_wdt\_reset (C++ function), 1062  
 esp\_task\_wdt\_status (C++ function), 1063  
 esp\_timer\_cb\_t (C++ type), 1078  
 esp\_timer\_create (C++ function), 1076  
 esp\_timer\_create\_args\_t (C++ class), 1077  
 esp\_timer\_create\_args\_t::arg (C++ member), 1077  
 esp\_timer\_create\_args\_t::callback (C++ member), 1077  
 esp\_timer\_create\_args\_t::dispatch\_method (C++ member), 1077  
 esp\_timer\_create\_args\_t::name (C++ member), 1077  
 esp\_timer\_deinit (C++ function), 1076  
 esp\_timer\_delete (C++ function), 1076  
 esp\_timer\_dispatch\_t (C++ enum), 1078  
 esp\_timer\_dump (C++ function), 1077  
 esp\_timer\_get\_next\_alarm (C++ function), 1077  
 esp\_timer\_get\_time (C++ function), 1077  
 esp\_timer\_handle\_t (C++ type), 1078  
 esp\_timer\_init (C++ function), 1075  
 esp\_timer\_start\_once (C++ function), 1076

- esp\_timer\_start\_periodic (C++ function), 1076  
 esp\_timer\_stop (C++ function), 1076  
 ESP\_TIMER\_TASK (C++ enumerator), 1078  
 esp\_tls (C++ class), 768  
 esp\_tls::cacert (C++ member), 768  
 esp\_tls::cacert\_ptr (C++ member), 768  
 esp\_tls::clientcert (C++ member), 768  
 esp\_tls::clientkey (C++ member), 768  
 esp\_tls::conf (C++ member), 768  
 esp\_tls::conn\_state (C++ member), 768  
 esp\_tls::ctr\_drbg (C++ member), 768  
 esp\_tls::entropy (C++ member), 768  
 esp\_tls::error\_handle (C++ member), 769  
 esp\_tls::is\_tls (C++ member), 768  
 esp\_tls::read (C++ member), 768  
 esp\_tls::role (C++ member), 768  
 esp\_tls::rset (C++ member), 768  
 esp\_tls::server\_fd (C++ member), 768  
 esp\_tls::sockfd (C++ member), 768  
 esp\_tls::ssl (C++ member), 768  
 esp\_tls::write (C++ member), 768  
 esp\_tls::wset (C++ member), 768  
 esp\_tls\_cfg (C++ class), 767  
 esp\_tls\_cfg::alpn\_protos (C++ member), 767  
 esp\_tls\_cfg::cacert\_pem\_buf (C++ member), 767  
 esp\_tls\_cfg::cacert\_pem\_bytes (C++ member), 767  
 esp\_tls\_cfg::clientcert\_pem\_buf (C++ member), 767  
 esp\_tls\_cfg::clientcert\_pem\_bytes (C++ member), 767  
 esp\_tls\_cfg::clientkey\_password (C++ member), 767  
 esp\_tls\_cfg::clientkey\_password\_len (C++ member), 767  
 esp\_tls\_cfg::clientkey\_pem\_buf (C++ member), 767  
 esp\_tls\_cfg::clientkey\_pem\_bytes (C++ member), 767  
 esp\_tls\_cfg::common\_name (C++ member), 767  
 esp\_tls\_cfg::keep\_alive\_cfg (C++ member), 768  
 esp\_tls\_cfg::non\_block (C++ member), 767  
 esp\_tls\_cfg::skip\_common\_name (C++ member), 768  
 esp\_tls\_cfg::timeout\_ms (C++ member), 767  
 esp\_tls\_cfg::use\_global\_ca\_store (C++ member), 767  
 esp\_tls\_cfg\_t (C++ type), 770  
 ESP\_TLS\_CLIENT (C++ enumerator), 770  
 esp\_tls\_conn\_delete (C++ function), 765  
 esp\_tls\_conn\_http\_new (C++ function), 764  
 esp\_tls\_conn\_http\_new\_async (C++ function), 764  
 esp\_tls\_conn\_new (C++ function), 763  
 esp\_tls\_conn\_new\_async (C++ function), 764  
 esp\_tls\_conn\_new\_sync (C++ function), 764  
 esp\_tls\_conn\_read (C++ function), 765  
 esp\_tls\_conn\_state (C++ enum), 770  
 esp\_tls\_conn\_state\_t (C++ type), 770  
 esp\_tls\_conn\_write (C++ function), 765  
 ESP\_TLS\_CONNECTING (C++ enumerator), 770  
 ESP\_TLS\_DONE (C++ enumerator), 770  
 esp\_tls\_error\_handle\_t (C++ type), 770  
 ESP\_TLS\_FAIL (C++ enumerator), 770  
 esp\_tls\_free\_global\_ca\_store (C++ function), 766  
 esp\_tls\_get\_and\_clear\_last\_error (C++ function), 766  
 esp\_tls\_get\_bytes\_avail (C++ function), 765  
 esp\_tls\_get\_global\_ca\_store (C++ function), 766  
 ESP\_TLS\_HANDSHAKE (C++ enumerator), 770  
 ESP\_TLS\_INIT (C++ enumerator), 770  
 esp\_tls\_init (C++ function), 763  
 esp\_tls\_init\_global\_ca\_store (C++ function), 765  
 esp\_tls\_last\_error (C++ class), 766  
 esp\_tls\_last\_error::last\_error (C++ member), 766  
 esp\_tls\_last\_error::mbedtls\_error\_code (C++ member), 766  
 esp\_tls\_last\_error::mbedtls\_flags (C++ member), 766  
 esp\_tls\_last\_error\_t (C++ type), 770  
 esp\_tls\_role (C++ enum), 770  
 esp\_tls\_role\_t (C++ type), 770  
 ESP\_TLS\_SERVER (C++ enumerator), 770  
 esp\_tls\_set\_global\_ca\_store (C++ function), 765  
 esp\_tls\_t (C++ type), 770  
 esp\_unregister\_shutdown\_handler (C++ function), 1136  
 ESP\_UUID\_LEN\_128 (C 宏), 109  
 ESP\_UUID\_LEN\_16 (C 宏), 109  
 ESP\_UUID\_LEN\_32 (C 宏), 109  
 esp\_vendor\_ie\_cb\_t (C++ type), 477  
 esp\_vfs\_close (C++ function), 916  
 esp\_vfs\_dev\_uart\_port\_set\_rx\_line\_endings (C++ function), 920  
 esp\_vfs\_dev\_uart\_port\_set\_tx\_line\_endings (C++ function), 920  
 esp\_vfs\_dev\_uart\_register (C++ function), 919  
 esp\_vfs\_dev\_uart\_set\_rx\_line\_endings (C++ function), 920  
 esp\_vfs\_dev\_uart\_set\_tx\_line\_endings (C++ function), 920  
 esp\_vfs\_dev\_uart\_use\_driver (C++ function), 920  
 esp\_vfs\_dev\_uart\_use\_nonblocking (C++ function), 920

- esp\_vfs\_fat\_mount\_config\_t (C++ class), 923, 927  
 esp\_vfs\_fat\_mount\_config\_t::allocation\_size (C++ member), 923, 927  
 esp\_vfs\_fat\_mount\_config\_t::format\_if\_mount\_failed (C++ member), 923, 927  
 esp\_vfs\_fat\_mount\_config\_t::max\_files (C++ member), 923, 927  
 esp\_vfs\_fat\_rawflash\_mount (C++ function), 923  
 esp\_vfs\_fat\_rawflash\_unmount (C++ function), 924  
 esp\_vfs\_fat\_register (C++ function), 922  
 esp\_vfs\_fat\_sdmmc\_mount (C++ function), 922  
 esp\_vfs\_fat\_sdmmc\_unmount (C++ function), 923  
 esp\_vfs\_fat\_spiflash\_mount (C++ function), 926  
 esp\_vfs\_fat\_spiflash\_unmount (C++ function), 927  
 esp\_vfs\_fat\_unregister\_path (C++ function), 922  
 ESP\_VFS\_FLAG\_CONTEXT\_PTR (C 宏), 919  
 ESP\_VFS\_FLAG\_DEFAULT (C 宏), 919  
 esp\_vfs\_fstat (C++ function), 916  
 esp\_vfs\_id\_t (C++ type), 919  
 esp\_vfs\_link (C++ function), 916  
 esp\_vfs\_lseek (C++ function), 916  
 esp\_vfs\_open (C++ function), 916  
 ESP\_VFS\_PATH\_MAX (C 宏), 919  
 esp\_vfs\_poll (C++ function), 918  
 esp\_vfs\_pread (C++ function), 918  
 esp\_vfs\_pwrite (C++ function), 918  
 esp\_vfs\_read (C++ function), 916  
 esp\_vfs\_register (C++ function), 916  
 esp\_vfs\_register\_fd (C++ function), 917  
 esp\_vfs\_register\_fd\_range (C++ function), 916  
 esp\_vfs\_register\_with\_id (C++ function), 916  
 esp\_vfs\_rename (C++ function), 916  
 esp\_vfs\_select (C++ function), 917  
 esp\_vfs\_select\_sem\_t (C++ class), 918  
 esp\_vfs\_select\_sem\_t::is\_sem\_local (C++ member), 918  
 esp\_vfs\_select\_sem\_t::sem (C++ member), 918  
 esp\_vfs\_select\_triggered (C++ function), 917  
 esp\_vfs\_select\_triggered\_isr (C++ function), 917  
 esp\_vfs\_spiffs\_conf\_t (C++ class), 932  
 esp\_vfs\_spiffs\_conf\_t::base\_path (C++ member), 932  
 esp\_vfs\_spiffs\_conf\_t::format\_if\_mount\_failed (C++ member), 932  
 esp\_vfs\_spiffs\_conf\_t::max\_files (C++ member), 932  
 esp\_vfs\_spiffs\_conf\_t::partition\_label (C++ member), 932  
 esp\_vfs\_spiffs\_register (C++ function), 931  
 esp\_vfs\_spiffs\_unregister (C++ function), 931  
 esp\_vfs\_stat (C++ function), 916  
 esp\_vfs\_t (C++ class), 918  
 esp\_vfs\_t::end\_select (C++ member), 919  
 esp\_vfs\_t::flags (C++ member), 919  
 esp\_vfs\_t::get\_socket\_select\_semaphore (C++ member), 919  
 esp\_vfs\_t::socket\_select (C++ member), 919  
 esp\_vfs\_t::start\_select (C++ member), 919  
 esp\_vfs\_t::stop\_socket\_select (C++ member), 919  
 esp\_vfs\_t::stop\_socket\_select\_isr (C++ member), 919  
 esp\_vfs\_unlink (C++ function), 916  
 esp\_vfs\_unregister (C++ function), 917  
 esp\_vfs\_unregister\_fd (C++ function), 917  
 esp\_vfs\_utime (C++ function), 916  
 esp\_vfs\_write (C++ function), 916  
 esp\_vhci\_host\_callback (C++ class), 106  
 esp\_vhci\_host\_callback::notify\_host\_recv (C++ member), 106  
 esp\_vhci\_host\_callback::notify\_host\_send\_available (C++ member), 106  
 esp\_vhci\_host\_callback\_t (C++ type), 106  
 esp\_vhci\_host\_check\_send\_available (C++ function), 103  
 esp\_vhci\_host\_register\_callback (C++ function), 103  
 esp\_vhci\_host\_send\_packet (C++ function), 103  
 esp\_wake\_deep\_sleep (C++ function), 1110  
 esp\_websocket\_client\_config\_t (C++ class), 785  
 esp\_websocket\_client\_config\_t::buffer\_size (C++ member), 786  
 esp\_websocket\_client\_config\_t::cert\_pem (C++ member), 786  
 esp\_websocket\_client\_config\_t::disable\_auto\_reconnect (C++ member), 785  
 esp\_websocket\_client\_config\_t::host (C++ member), 785  
 esp\_websocket\_client\_config\_t::keep\_alive\_count (C++ member), 786  
 esp\_websocket\_client\_config\_t::keep\_alive\_enable (C++ member), 786  
 esp\_websocket\_client\_config\_t::keep\_alive\_idle (C++ member), 786  
 esp\_websocket\_client\_config\_t::keep\_alive\_interval (C++ member), 786  
 esp\_websocket\_client\_config\_t::password (C++ member), 785  
 esp\_websocket\_client\_config\_t::path (C++ member), 785

- esp\_websocket\_client\_config\_t::port (C++ member), 785  
 esp\_websocket\_client\_config\_t::subprotocol (C++ member), 786  
 esp\_websocket\_client\_config\_t::task\_priority (C++ member), 785  
 esp\_websocket\_client\_config\_t::task\_stack (C++ member), 785  
 esp\_websocket\_client\_config\_t::transport (C++ member), 786  
 esp\_websocket\_client\_config\_t::uri (C++ member), 785  
 esp\_websocket\_client\_config\_t::user\_content (C++ member), 785  
 esp\_websocket\_client\_config\_t::username (C++ member), 785  
 esp\_websocket\_client\_destroy (C++ function), 783  
 esp\_websocket\_client\_handle\_t (C++ type), 786  
 esp\_websocket\_client\_init (C++ function), 783  
 esp\_websocket\_client\_is\_connected (C++ function), 784  
 esp\_websocket\_client\_send (C++ function), 784  
 esp\_websocket\_client\_send\_bin (C++ function), 784  
 esp\_websocket\_client\_send\_text (C++ function), 784  
 esp\_websocket\_client\_set\_uri (C++ function), 783  
 esp\_websocket\_client\_start (C++ function), 783  
 esp\_websocket\_client\_stop (C++ function), 783  
 esp\_websocket\_event\_data\_t (C++ class), 785  
 esp\_websocket\_event\_data\_t::client (C++ member), 785  
 esp\_websocket\_event\_data\_t::data\_len (C++ member), 785  
 esp\_websocket\_event\_data\_t::data\_ptr (C++ member), 785  
 esp\_websocket\_event\_data\_t::op\_code (C++ member), 785  
 esp\_websocket\_event\_data\_t::payload\_len (C++ member), 785  
 esp\_websocket\_event\_data\_t::payload\_offset (C++ member), 785  
 esp\_websocket\_event\_data\_t::user\_context (C++ member), 785  
 esp\_websocket\_event\_id\_t (C++ enum), 786  
 esp\_websocket\_register\_events (C++ function), 784  
 esp\_websocket\_transport\_t (C++ enum), 786  
 esp\_wifi\_80211\_tx (C++ function), 472  
 esp\_wifi\_ap\_get\_sta\_aid (C++ function), 470  
 esp\_wifi\_ap\_get\_sta\_list (C++ function), 470  
 esp\_wifi\_clear\_fast\_connect (C++ function), 464  
 esp\_wifi\_connect (C++ function), 464  
 esp\_wifi\_death\_sta (C++ function), 464  
 esp\_wifi\_deinit (C++ function), 463  
 esp\_wifi\_disconnect (C++ function), 464  
 esp\_wifi\_get\_ant (C++ function), 473  
 esp\_wifi\_get\_ant\_gpio (C++ function), 473  
 esp\_wifi\_get\_bandwidth (C++ function), 466  
 esp\_wifi\_get\_channel (C++ function), 467  
 esp\_wifi\_get\_config (C++ function), 470  
 esp\_wifi\_get\_country (C++ function), 468  
 esp\_wifi\_get\_event\_mask (C++ function), 472  
 esp\_wifi\_get\_inactive\_time (C++ function), 474  
 esp\_wifi\_get\_mac (C++ function), 468  
 esp\_wifi\_get\_max\_tx\_power (C++ function), 471  
 esp\_wifi\_get\_mode (C++ function), 463  
 esp\_wifi\_get\_promiscuous (C++ function), 469  
 esp\_wifi\_get\_promiscuous\_ctrl\_filter (C++ function), 469  
 esp\_wifi\_get\_promiscuous\_filter (C++ function), 469  
 esp\_wifi\_get\_protocol (C++ function), 466  
 esp\_wifi\_get\_ps (C++ function), 466  
 esp\_wifi\_init (C++ function), 462  
 ESP\_WIFI\_MAX\_CONN\_NUM (C 宏), 487  
 esp\_wifi\_restore (C++ function), 463  
 esp\_wifi\_scan\_get\_ap\_num (C++ function), 465  
 esp\_wifi\_scan\_get\_ap\_records (C++ function), 465  
 esp\_wifi\_scan\_start (C++ function), 464  
 esp\_wifi\_scan\_stop (C++ function), 465  
 esp\_wifi\_set\_ant (C++ function), 473  
 esp\_wifi\_set\_ant\_gpio (C++ function), 473  
 esp\_wifi\_set\_bandwidth (C++ function), 466  
 esp\_wifi\_set\_channel (C++ function), 467  
 esp\_wifi\_set\_config (C++ function), 469  
 esp\_wifi\_set\_country (C++ function), 467  
 esp\_wifi\_set\_csi (C++ function), 473  
 esp\_wifi\_set\_csi\_config (C++ function), 472  
 esp\_wifi\_set\_csi\_rx\_cb (C++ function), 472  
 esp\_wifi\_set\_event\_mask (C++ function), 471  
 esp\_wifi\_set\_inactive\_time (C++ function), 473  
 esp\_wifi\_set\_mac (C++ function), 468  
 esp\_wifi\_set\_max\_tx\_power (C++ function), 471  
 esp\_wifi\_set\_mode (C++ function), 463  
 esp\_wifi\_set\_promiscuous (C++ function), 468  
 esp\_wifi\_set\_promiscuous\_ctrl\_filter (C++ function), 469

- esp\_wifi\_set\_promiscuous\_filter (C++ function), 469  
 esp\_wifi\_set\_promiscuous\_rx\_cb (C++ function), 468  
 esp\_wifi\_set\_protocol (C++ function), 466  
 esp\_wifi\_set\_ps (C++ function), 466  
 esp\_wifi\_set\_storage (C++ function), 470  
 esp\_wifi\_set\_vendor\_ie (C++ function), 470  
 esp\_wifi\_set\_vendor\_ie\_cb (C++ function), 471  
 esp\_wifi\_sta\_get\_ap\_info (C++ function), 465  
 esp\_wifi\_start (C++ function), 463  
 esp\_wifi\_status\_dump (C++ function), 474  
 esp\_wifi\_stop (C++ function), 463  
 eStandardSleep (C++ enumerator), 960  
 eSuspended (C++ enumerator), 959  
 eTaskGetState (C++ function), 944  
 eTaskState (C++ enum), 959  
 ETH\_CMD\_G\_MAC\_ADDR (C++ enumerator), 537  
 ETH\_CMD\_G\_PHY\_ADDR (C++ enumerator), 537  
 ETH\_CMD\_G\_SPEED (C++ enumerator), 537  
 ETH\_CMD\_S\_MAC\_ADDR (C++ enumerator), 537  
 ETH\_CMD\_S\_PHY\_ADDR (C++ enumerator), 537  
 ETH\_CMD\_S\_PROMISCUOUS (C++ enumerator), 537  
 ETH\_CRC\_LEN (C 宏), 536  
 ETH\_DEFAULT\_CONFIG (C 宏), 535  
 ETH\_DUPLEX\_FULL (C++ enumerator), 537  
 ETH\_DUPLEX\_HALF (C++ enumerator), 537  
 eth\_duplex\_t (C++ enum), 537  
 eth\_event\_t (C++ enum), 537  
 ETH\_HEADER\_LEN (C 宏), 536  
 ETH\_JUMBO\_FRAME\_PAYLOAD\_LEN (C 宏), 536  
 ETH\_LINK\_DOWN (C++ enumerator), 537  
 eth\_link\_t (C++ enum), 537  
 ETH\_LINK\_UP (C++ enumerator), 537  
 eth\_mac\_config\_t (C++ class), 541  
 eth\_mac\_config\_t::flags (C++ member), 541  
 eth\_mac\_config\_t::rx\_task\_prio (C++ member), 541  
 eth\_mac\_config\_t::rx\_task\_stack\_size (C++ member), 541  
 eth\_mac\_config\_t::smi\_mdc\_gpio\_num (C++ member), 541  
 eth\_mac\_config\_t::smi\_mdio\_gpio\_num (C++ member), 541  
 eth\_mac\_config\_t::sw\_reset\_timeout\_ms (C++ member), 541  
 ETH\_MAC\_DEFAULT\_CONFIG (C 宏), 541  
 ETH\_MAC\_FLAG\_PIN\_TO\_CORE (C 宏), 541  
 ETH\_MAC\_FLAG\_WORK\_WITH\_CACHE\_DISABLE (C 宏), 541  
 ETH\_MAX\_PACKET\_SIZE (C 宏), 536  
 ETH\_MAX\_PAYLOAD\_LEN (C 宏), 536  
 ETH\_MIN\_PACKET\_SIZE (C 宏), 536  
 ETH\_MIN\_PAYLOAD\_LEN (C 宏), 536  
 eth\_phy\_config\_t (C++ class), 544  
 eth\_phy\_config\_t::autonego\_timeout\_ms (C++ member), 544  
 eth\_phy\_config\_t::phy\_addr (C++ member), 544  
 eth\_phy\_config\_t::reset\_gpio\_num (C++ member), 544  
 eth\_phy\_config\_t::reset\_timeout\_ms (C++ member), 544  
 ETH\_PHY\_DEFAULT\_CONFIG (C 宏), 544  
 ETH\_SPEED\_100M (C++ enumerator), 537  
 ETH\_SPEED\_10M (C++ enumerator), 537  
 eth\_speed\_t (C++ enum), 537  
 ETH\_STATE\_DEINIT (C++ enumerator), 536  
 ETH\_STATE\_DUPLEX (C++ enumerator), 537  
 ETH\_STATE\_LINK (C++ enumerator), 536  
 ETH\_STATE\_LLINIT (C++ enumerator), 536  
 ETH\_STATE\_SPEED (C++ enumerator), 537  
 ETH\_VLAN\_TAG\_LEN (C 宏), 536  
 ETHERNET\_EVENT\_CONNECTED (C++ enumerator), 538  
 ETHERNET\_EVENT\_DISCONNECTED (C++ enumerator), 538  
 ETHERNET\_EVENT\_START (C++ enumerator), 537  
 ETHERNET\_EVENT\_STOP (C++ enumerator), 538  
 ETS\_INTERNAL\_INTR\_SOURCE\_OFF (C 宏), 1060  
 ETS\_INTERNAL\_PROFILING\_INTR\_SOURCE (C 宏), 1060  
 ETS\_INTERNAL\_SW0\_INTR\_SOURCE (C 宏), 1060  
 ETS\_INTERNAL\_SW1\_INTR\_SOURCE (C 宏), 1060  
 ETS\_INTERNAL\_TIMER0\_INTR\_SOURCE (C 宏), 1059  
 ETS\_INTERNAL\_TIMER1\_INTR\_SOURCE (C 宏), 1060  
 ETS\_INTERNAL\_TIMER2\_INTR\_SOURCE (C 宏), 1060  
 EventBits\_t (C++ type), 1016  
 EventGroupHandle\_t (C++ type), 1016
- ## F
- FAST\_PROV\_ACT\_ENTER (C++ enumerator), 299  
 FAST\_PROV\_ACT\_EXIT (C++ enumerator), 299  
 FAST\_PROV\_ACT\_MAX (C++ enumerator), 299  
 FAST\_PROV\_ACT\_NONE (C++ enumerator), 299  
 FAST\_PROV\_ACT\_SUSPEND (C++ enumerator), 299  
 ff\_diskio\_impl\_t (C++ class), 924  
 ff\_diskio\_impl\_t::init (C++ member), 924  
 ff\_diskio\_impl\_t::ioctl (C++ member), 925  
 ff\_diskio\_impl\_t::read (C++ member), 924  
 ff\_diskio\_impl\_t::status (C++ member), 924  
 ff\_diskio\_impl\_t::write (C++ member), 924  
 ff\_diskio\_register (C++ function), 924  
 ff\_diskio\_register\_raw\_partition (C++ function), 925  
 ff\_diskio\_register\_sdmmc (C++ function), 925  
 ff\_diskio\_register\_wl\_partition (C++ function), 925

filter\_cb\_t (C++ type), 732

## G

gpio\_config (C++ function), 587

gpio\_config\_t (C++ class), 592

gpio\_config\_t::intr\_type (C++ member), 592

gpio\_config\_t::mode (C++ member), 592

gpio\_config\_t::pin\_bit\_mask (C++ member), 592

gpio\_config\_t::pull\_down\_en (C++ member), 592

gpio\_config\_t::pull\_up\_en (C++ member), 592

gpio\_deep\_sleep\_hold\_dis (C++ function), 592

gpio\_deep\_sleep\_hold\_en (C++ function), 592

GPIO\_DRIVE\_CAP\_0 (C++ enumerator), 594

GPIO\_DRIVE\_CAP\_1 (C++ enumerator), 594

GPIO\_DRIVE\_CAP\_2 (C++ enumerator), 594

GPIO\_DRIVE\_CAP\_3 (C++ enumerator), 594

GPIO\_DRIVE\_CAP\_DEFAULT (C++ enumerator), 594

GPIO\_DRIVE\_CAP\_MAX (C++ enumerator), 594

gpio\_drive\_cap\_t (C++ enum), 594

GPIO\_FLOATING (C++ enumerator), 594

gpio\_get\_drive\_capability (C++ function), 591

gpio\_get\_level (C++ function), 588

gpio\_hold\_dis (C++ function), 591

gpio\_hold\_en (C++ function), 591

gpio\_install\_isr\_service (C++ function), 590

gpio\_int\_type\_t (C++ enum), 593

GPIO\_INTR\_ANYEDGE (C++ enumerator), 593

GPIO\_INTR\_DISABLE (C++ enumerator), 593

gpio\_intr\_disable (C++ function), 588

gpio\_intr\_enable (C++ function), 588

GPIO\_INTR\_HIGH\_LEVEL (C++ enumerator), 593

GPIO\_INTR\_LOW\_LEVEL (C++ enumerator), 593

GPIO\_INTR\_MAX (C++ enumerator), 593

GPIO\_INTR\_NEGEDGE (C++ enumerator), 593

GPIO\_INTR\_POSEDGE (C++ enumerator), 593

gpio\_iomux\_in (C++ function), 592

gpio\_iomux\_out (C++ function), 592

GPIO\_IS\_VALID\_GPIO (C 宏), 593

GPIO\_IS\_VALID\_OUTPUT\_GPIO (C 宏), 593

gpio\_isr\_handle\_t (C++ type), 593

gpio\_isr\_handler\_add (C++ function), 590

gpio\_isr\_handler\_remove (C++ function), 591

gpio\_isr\_register (C++ function), 589

gpio\_isr\_t (C++ type), 593

GPIO\_MODE\_DISABLE (C++ enumerator), 593

GPIO\_MODE\_INPUT (C++ enumerator), 593

GPIO\_MODE\_INPUT\_OUTPUT (C++ enumerator), 594

GPIO\_MODE\_INPUT\_OUTPUT\_OD (C++ enumerator), 594

GPIO\_MODE\_OUTPUT (C++ enumerator), 593

GPIO\_MODE\_OUTPUT\_OD (C++ enumerator), 594

gpio\_mode\_t (C++ enum), 593

GPIO\_NUM\_0 (C++ enumerator), 593

GPIO\_NUM\_1 (C++ enumerator), 593

GPIO\_NUM\_2 (C++ enumerator), 593

GPIO\_NUM\_NC (C++ enumerator), 593

gpio\_num\_t (C++ enum), 593

gpio\_pull\_mode\_t (C++ enum), 594

gpio\_pulldown\_dis (C++ function), 590

GPIO\_PULLDOWN\_DISABLE (C++ enumerator), 594

gpio\_pulldown\_en (C++ function), 590

GPIO\_PULLDOWN\_ENABLE (C++ enumerator), 594

GPIO\_PULLDOWN\_ONLY (C++ enumerator), 594

gpio\_pulldown\_t (C++ enum), 594

gpio\_pullup\_dis (C++ function), 589

GPIO\_PULLUP\_DISABLE (C++ enumerator), 594

gpio\_pullup\_en (C++ function), 589

GPIO\_PULLUP\_ENABLE (C++ enumerator), 594

GPIO\_PULLUP\_ONLY (C++ enumerator), 594

GPIO\_PULLUP\_PULLDOWN (C++ enumerator), 594

gpio\_pullup\_t (C++ enum), 594

gpio\_reset\_pin (C++ function), 587

GPIO\_SEL\_0 (C 宏), 592

GPIO\_SEL\_1 (C 宏), 592

GPIO\_SEL\_2 (C 宏), 592

gpio\_set\_direction (C++ function), 588

gpio\_set\_drive\_capability (C++ function), 591

gpio\_set\_intr\_type (C++ function), 587

gpio\_set\_level (C++ function), 588

gpio\_set\_pull\_mode (C++ function), 588

gpio\_uninstall\_isr\_service (C++ function), 590

gpio\_wakeup\_disable (C++ function), 589

gpio\_wakeup\_enable (C++ function), 589

## H

hall\_sensor\_read (C++ function), 563

heap\_caps\_add\_region (C++ function), 1039

heap\_caps\_add\_region\_with\_caps (C++ function), 1039

heap\_caps\_calloc (C++ function), 1035

heap\_caps\_calloc\_prefer (C++ function), 1037

heap\_caps\_check\_integrity (C++ function), 1036

heap\_caps\_check\_integrity\_addr (C++ function), 1036

heap\_caps\_check\_integrity\_all (C++ function), 1036

heap\_caps\_dump (C++ function), 1037

heap\_caps\_dump\_all (C++ function), 1038

heap\_caps\_enable\_nonos\_stack\_heaps (C++ function), 1039

heap\_caps\_free (C++ function), 1035

heap\_caps\_get\_free\_size (C++ function), 1035

- heap\_caps\_get\_info (C++ function), 1036  
 heap\_caps\_get\_largest\_free\_block (C++ function), 1036  
 heap\_caps\_get\_minimum\_free\_size (C++ function), 1035  
 heap\_caps\_init (C++ function), 1039  
 heap\_caps\_malloc (C++ function), 1035  
 heap\_caps\_malloc\_extmem\_enable (C++ function), 1037  
 heap\_caps\_malloc\_prefer (C++ function), 1037  
 heap\_caps\_print\_heap\_info (C++ function), 1036  
 heap\_caps\_realloc (C++ function), 1035  
 heap\_caps\_realloc\_prefer (C++ function), 1037  
 HEAP\_TRACE\_ALL (C++ enumerator), 1052  
 heap\_trace\_dump (C++ function), 1052  
 heap\_trace\_get (C++ function), 1052  
 heap\_trace\_get\_count (C++ function), 1051  
 heap\_trace\_init\_standalone (C++ function), 1050  
 heap\_trace\_init\_tohost (C++ function), 1051  
 HEAP\_TRACE\_LEAKS (C++ enumerator), 1052  
 heap\_trace\_mode\_t (C++ enum), 1052  
 heap\_trace\_record\_t (C++ class), 1052  
 heap\_trace\_record\_t::address (C++ member), 1052  
 heap\_trace\_record\_t::allocated\_by (C++ member), 1052  
 heap\_trace\_record\_t::ccount (C++ member), 1052  
 heap\_trace\_record\_t::freed\_by (C++ member), 1052  
 heap\_trace\_record\_t::size (C++ member), 1052  
 heap\_trace\_resume (C++ function), 1051  
 heap\_trace\_start (C++ function), 1051  
 heap\_trace\_stop (C++ function), 1051  
 HSPI\_HOST (C 宏), 701  
 HTTP\_AUTH\_TYPE\_BASIC (C++ enumerator), 781  
 HTTP\_AUTH\_TYPE\_DIGEST (C++ enumerator), 781  
 HTTP\_AUTH\_TYPE\_NONE (C++ enumerator), 781  
 HTTP\_EVENT\_DISCONNECTED (C++ enumerator), 781  
 HTTP\_EVENT\_ERROR (C++ enumerator), 780  
 http\_event\_handle\_cb (C++ type), 780  
 HTTP\_EVENT\_HEADER\_SENT (C++ enumerator), 780  
 HTTP\_EVENT\_HEADERS\_SENT (C++ enumerator), 780  
 HTTP\_EVENT\_ON\_CONNECTED (C++ enumerator), 780  
 HTTP\_EVENT\_ON\_DATA (C++ enumerator), 780  
 HTTP\_EVENT\_ON\_FINISH (C++ enumerator), 780  
 HTTP\_EVENT\_ON\_HEADER (C++ enumerator), 780  
 HTTP\_METHOD\_DELETE (C++ enumerator), 781  
 HTTP\_METHOD\_GET (C++ enumerator), 781  
 HTTP\_METHOD\_HEAD (C++ enumerator), 781  
 HTTP\_METHOD\_MAX (C++ enumerator), 781  
 HTTP\_METHOD\_NOTIFY (C++ enumerator), 781  
 HTTP\_METHOD\_OPTIONS (C++ enumerator), 781  
 HTTP\_METHOD\_PATCH (C++ enumerator), 781  
 HTTP\_METHOD\_POST (C++ enumerator), 781  
 HTTP\_METHOD\_PUT (C++ enumerator), 781  
 HTTP\_METHOD\_SUBSCRIBE (C++ enumerator), 781  
 HTTP\_METHOD\_UNSUBSCRIBE (C++ enumerator), 781  
 HTTP\_TRANSPORT\_OVER\_SSL (C++ enumerator), 781  
 HTTP\_TRANSPORT\_OVER\_TCP (C++ enumerator), 781  
 HTTP\_TRANSPORT\_UNKNOWN (C++ enumerator), 781  
 HTTPD\_200 (C 宏), 803  
 HTTPD\_204 (C 宏), 803  
 HTTPD\_207 (C 宏), 803  
 HTTPD\_400 (C 宏), 803  
 HTTPD\_400\_BAD\_REQUEST (C++ enumerator), 807  
 HTTPD\_404 (C 宏), 804  
 HTTPD\_404\_NOT\_FOUND (C++ enumerator), 807  
 HTTPD\_405\_METHOD\_NOT\_ALLOWED (C++ enumerator), 807  
 HTTPD\_408 (C 宏), 804  
 HTTPD\_408\_REQ\_TIMEOUT (C++ enumerator), 807  
 HTTPD\_411\_LENGTH\_REQUIRED (C++ enumerator), 807  
 HTTPD\_414\_URI\_TOO\_LONG (C++ enumerator), 807  
 HTTPD\_431\_REQ\_HDR\_FIELDS\_TOO\_LARGE (C++ enumerator), 807  
 HTTPD\_500 (C 宏), 804  
 HTTPD\_500\_INTERNAL\_SERVER\_ERROR (C++ enumerator), 807  
 HTTPD\_501\_METHOD\_NOT\_IMPLEMENTED (C++ enumerator), 807  
 HTTPD\_505\_VERSION\_NOT\_SUPPORTED (C++ enumerator), 807  
 httpd\_close\_func\_t (C++ type), 806  
 httpd\_config (C++ class), 801  
 httpd\_config::backlog\_conn (C++ member), 801  
 httpd\_config::close\_fn (C++ member), 802  
 httpd\_config::core\_id (C++ member), 801  
 httpd\_config::ctrl\_port (C++ member), 801  
 httpd\_config::global\_transport\_ctx (C++ member), 801  
 httpd\_config::global\_transport\_ctx\_free\_fn (C++ member), 802  
 httpd\_config::global\_user\_ctx (C++ member), 801  
 httpd\_config::global\_user\_ctx\_free\_fn (C++ member), 801  
 httpd\_config::lru\_purge\_enable (C++ member), 801  
 httpd\_config::max\_open\_sockets (C++

- member*), 801  
 httpd\_config::max\_resp\_headers (C++ *member*), 801  
 httpd\_config::max\_uri\_handlers (C++ *member*), 801  
 httpd\_config::open\_fn (C++ *member*), 802  
 httpd\_config::recv\_wait\_timeout (C++ *member*), 801  
 httpd\_config::send\_wait\_timeout (C++ *member*), 801  
 httpd\_config::server\_port (C++ *member*), 801  
 httpd\_config::stack\_size (C++ *member*), 801  
 httpd\_config::task\_priority (C++ *member*), 801  
 httpd\_config::uri\_match\_fn (C++ *member*), 802  
 httpd\_config\_t (C++ *type*), 806  
 HTTPD\_DEFAULT\_CONFIG (C 宏), 804  
 HTTPD\_ERR\_CODE\_MAX (C++ *enumerator*), 807  
 httpd\_err\_code\_t (C++ *enum*), 807  
 httpd\_err\_handler\_func\_t (C++ *type*), 805  
 httpd\_free\_ctx\_fn\_t (C++ *type*), 806  
 httpd\_get\_global\_transport\_ctx (C++ *function*), 800  
 httpd\_get\_global\_user\_ctx (C++ *function*), 800  
 httpd\_handle\_t (C++ *type*), 806  
 HTTPD\_MAX\_REQ\_HDR\_LEN (C 宏), 803  
 HTTPD\_MAX\_URI\_LEN (C 宏), 803  
 httpd\_method\_t (C++ *type*), 806  
 httpd\_open\_func\_t (C++ *type*), 806  
 httpd\_pending\_func\_t (C++ *type*), 805  
 httpd\_query\_key\_value (C++ *function*), 793  
 httpd\_queue\_work (C++ *function*), 799  
 httpd\_recv\_func\_t (C++ *type*), 805  
 httpd\_register\_err\_handler (C++ *function*), 798  
 httpd\_register\_uri\_handler (C++ *function*), 789  
 httpd\_req (C++ *class*), 802  
 httpd\_req::aux (C++ *member*), 802  
 httpd\_req::content\_len (C++ *member*), 802  
 httpd\_req::free\_ctx (C++ *member*), 803  
 httpd\_req::handle (C++ *member*), 802  
 httpd\_req::ignore\_sess\_ctx\_changes (C++ *member*), 803  
 httpd\_req::method (C++ *member*), 802  
 httpd\_req::sess\_ctx (C++ *member*), 802  
 httpd\_req::uri (C++ *member*), 802  
 httpd\_req::user\_ctx (C++ *member*), 802  
 httpd\_req\_get\_hdr\_value\_len (C++ *function*), 792  
 httpd\_req\_get\_hdr\_value\_str (C++ *function*), 792  
 httpd\_req\_get\_url\_query\_len (C++ *function*), 792  
 httpd\_req\_get\_url\_query\_str (C++ *function*), 793  
 httpd\_req\_recv (C++ *function*), 791  
 httpd\_req\_t (C++ *type*), 804  
 httpd\_req\_to\_sockfd (C++ *function*), 791  
 httpd\_resp\_send (C++ *function*), 794  
 httpd\_resp\_send\_404 (C++ *function*), 796  
 httpd\_resp\_send\_408 (C++ *function*), 797  
 httpd\_resp\_send\_500 (C++ *function*), 797  
 httpd\_resp\_send\_chunk (C++ *function*), 794  
 httpd\_resp\_send\_err (C++ *function*), 796  
 httpd\_resp\_sendstr (C++ *function*), 795  
 httpd\_resp\_sendstr\_chunk (C++ *function*), 795  
 httpd\_resp\_set\_hdr (C++ *function*), 796  
 httpd\_resp\_set\_status (C++ *function*), 795  
 httpd\_resp\_set\_type (C++ *function*), 795  
 HTTPD\_RESP\_USE\_STRLEN (C 宏), 804  
 httpd\_send (C++ *function*), 797  
 httpd\_send\_func\_t (C++ *type*), 804  
 httpd\_sess\_get\_ctx (C++ *function*), 799  
 httpd\_sess\_get\_transport\_ctx (C++ *function*), 800  
 httpd\_sess\_set\_ctx (C++ *function*), 799  
 httpd\_sess\_set\_pending\_override (C++ *function*), 791  
 httpd\_sess\_set\_recv\_override (C++ *function*), 790  
 httpd\_sess\_set\_send\_override (C++ *function*), 791  
 httpd\_sess\_set\_transport\_ctx (C++ *function*), 800  
 httpd\_sess\_trigger\_close (C++ *function*), 800  
 httpd\_sess\_update\_lru\_counter (C++ *function*), 800  
 HTTPD\_SOCK\_ERR\_FAIL (C 宏), 803  
 HTTPD\_SOCK\_ERR\_INVALID (C 宏), 803  
 HTTPD\_SOCK\_ERR\_TIMEOUT (C 宏), 803  
 httpd\_ssl\_config (C++ *class*), 808  
 httpd\_ssl\_config::cacert\_len (C++ *member*), 808  
 httpd\_ssl\_config::cacert\_pem (C++ *member*), 808  
 httpd\_ssl\_config::httpd (C++ *member*), 808  
 httpd\_ssl\_config::port\_insecure (C++ *member*), 808  
 httpd\_ssl\_config::port\_secure (C++ *member*), 808  
 httpd\_ssl\_config::prvtkey\_len (C++ *member*), 808  
 httpd\_ssl\_config::prvtkey\_pem (C++ *member*), 808  
 httpd\_ssl\_config::transport\_mode (C++ *member*), 808  
 HTTPD\_SSL\_CONFIG\_DEFAULT (C 宏), 809  
 httpd\_ssl\_config\_t (C++ *type*), 809  
 httpd\_ssl\_start (C++ *function*), 808



- [httpd\\_ssl\\_stop \(C++ function\), 808](#)  
[HTTPD\\_SSL\\_TRANSPORT\\_INSECURE \(C++ enumerator\), 809](#)  
[httpd\\_ssl\\_transport\\_mode\\_t \(C++ enum\), 809](#)  
[HTTPD\\_SSL\\_TRANSPORT\\_SECURE \(C++ enumerator\), 809](#)  
[httpd\\_start \(C++ function\), 798](#)  
[httpd\\_stop \(C++ function\), 799](#)  
[HTTPD\\_TYPE\\_JSON \(C 宏\), 804](#)  
[HTTPD\\_TYPE\\_OCTET \(C 宏\), 804](#)  
[HTTPD\\_TYPE\\_TEXT \(C 宏\), 804](#)  
[httpd\\_unregister\\_uri \(C++ function\), 790](#)  
[httpd\\_unregister\\_uri\\_handler \(C++ function\), 790](#)  
[httpd\\_uri \(C++ class\), 803](#)  
[httpd\\_uri::handler \(C++ member\), 803](#)  
[httpd\\_uri::method \(C++ member\), 803](#)  
[httpd\\_uri::uri \(C++ member\), 803](#)  
[httpd\\_uri::user\\_ctx \(C++ member\), 803](#)  
[httpd\\_uri\\_match\\_func\\_t \(C++ type\), 806](#)  
[httpd\\_uri\\_match\\_wildcard \(C++ function\), 793](#)  
[httpd\\_uri\\_t \(C++ type\), 804](#)  
[httpd\\_work\\_fn\\_t \(C++ type\), 806](#)  
[HttpStatus\\_Code \(C++ enum\), 781](#)  
[HttpStatus\\_Found \(C++ enumerator\), 782](#)  
[HttpStatus\\_MovedPermanently \(C++ enumerator\), 782](#)  
[HttpStatus\\_MultipleChoices \(C++ enumerator\), 782](#)  
[HttpStatus\\_Ok \(C++ enumerator\), 782](#)  
[HttpStatus\\_Unauthorized \(C++ enumerator\), 782](#)
- I**
- [i2c\\_ack\\_type\\_t \(C++ enum\), 610](#)  
[I2C\\_ADDR\\_BIT\\_10 \(C++ enumerator\), 609](#)  
[I2C\\_ADDR\\_BIT\\_7 \(C++ enumerator\), 609](#)  
[I2C\\_ADDR\\_BIT\\_MAX \(C++ enumerator\), 609](#)  
[i2c\\_addr\\_mode\\_t \(C++ enum\), 609](#)  
[I2C\\_APB\\_CLK\\_FREQ \(C 宏\), 608](#)  
[I2C\\_CMD\\_END \(C++ enumerator\), 609](#)  
[i2c\\_cmd\\_handle\\_t \(C++ type\), 608](#)  
[i2c\\_cmd\\_link\\_create \(C++ function\), 603](#)  
[i2c\\_cmd\\_link\\_delete \(C++ function\), 603](#)  
[I2C\\_CMD\\_READ \(C++ enumerator\), 609](#)  
[I2C\\_CMD\\_RESTART \(C++ enumerator\), 609](#)  
[I2C\\_CMD\\_STOP \(C++ enumerator\), 609](#)  
[I2C\\_CMD\\_WRITE \(C++ enumerator\), 609](#)  
[i2c\\_config\\_t \(C++ class\), 608](#)  
[i2c\\_config\\_t::addr\\_10bit\\_en \(C++ member\), 608](#)  
[i2c\\_config\\_t::clk\\_speed \(C++ member\), 608](#)  
[i2c\\_config\\_t::mode \(C++ member\), 608](#)  
[i2c\\_config\\_t::scl\\_io\\_num \(C++ member\), 608](#)  
[i2c\\_config\\_t::scl\\_pullup\\_en \(C++ member\), 608](#)  
[i2c\\_config\\_t::sda\\_io\\_num \(C++ member\), 608](#)  
[i2c\\_config\\_t::sda\\_pullup\\_en \(C++ member\), 608](#)  
[i2c\\_config\\_t::slave\\_addr \(C++ member\), 608](#)  
[I2C\\_DATA\\_MODE\\_LSB\\_FIRST \(C++ enumerator\), 609](#)  
[I2C\\_DATA\\_MODE\\_MAX \(C++ enumerator\), 609](#)  
[I2C\\_DATA\\_MODE\\_MSB\\_FIRST \(C++ enumerator\), 609](#)  
[i2c\\_driver\\_delete \(C++ function\), 602](#)  
[i2c\\_driver\\_install \(C++ function\), 601](#)  
[I2C\\_FIFO\\_LEN \(C 宏\), 608](#)  
[i2c\\_filter\\_disable \(C++ function\), 606](#)  
[i2c\\_filter\\_enable \(C++ function\), 606](#)  
[i2c\\_get\\_data\\_mode \(C++ function\), 608](#)  
[i2c\\_get\\_data\\_timing \(C++ function\), 607](#)  
[i2c\\_get\\_period \(C++ function\), 605](#)  
[i2c\\_get\\_start\\_timing \(C++ function\), 606](#)  
[i2c\\_get\\_stop\\_timing \(C++ function\), 607](#)  
[i2c\\_get\\_timeout \(C++ function\), 607](#)  
[i2c\\_isr\\_free \(C++ function\), 603](#)  
[i2c\\_isr\\_register \(C++ function\), 602](#)  
[I2C\\_MASTER\\_ACK \(C++ enumerator\), 610](#)  
[I2C\\_MASTER\\_ACK\\_MAX \(C++ enumerator\), 610](#)  
[i2c\\_master\\_cmd\\_begin \(C++ function\), 604](#)  
[I2C\\_MASTER\\_LAST\\_NACK \(C++ enumerator\), 610](#)  
[I2C\\_MASTER\\_NACK \(C++ enumerator\), 610](#)  
[I2C\\_MASTER\\_READ \(C++ enumerator\), 609](#)  
[i2c\\_master\\_read \(C++ function\), 604](#)  
[i2c\\_master\\_read\\_byte \(C++ function\), 604](#)  
[i2c\\_master\\_start \(C++ function\), 603](#)  
[i2c\\_master\\_stop \(C++ function\), 604](#)  
[I2C\\_MASTER\\_WRITE \(C++ enumerator\), 609](#)  
[i2c\\_master\\_write \(C++ function\), 604](#)  
[i2c\\_master\\_write\\_byte \(C++ function\), 603](#)  
[I2C\\_MODE\\_MASTER \(C++ enumerator\), 609](#)  
[I2C\\_MODE\\_MAX \(C++ enumerator\), 609](#)  
[I2C\\_MODE\\_SLAVE \(C++ enumerator\), 609](#)  
[i2c\\_mode\\_t \(C++ enum\), 609](#)  
[I2C\\_NUM\\_0 \(C++ enumerator\), 609](#)  
[I2C\\_NUM\\_1 \(C++ enumerator\), 609](#)  
[I2C\\_NUM\\_MAX \(C++ enumerator\), 609](#)  
[i2c\\_opmode\\_t \(C++ enum\), 609](#)  
[i2c\\_param\\_config \(C++ function\), 602](#)  
[i2c\\_port\\_t \(C++ enum\), 609](#)  
[i2c\\_reset\\_rx\\_fifo \(C++ function\), 602](#)  
[i2c\\_reset\\_tx\\_fifo \(C++ function\), 602](#)  
[i2c\\_rw\\_t \(C++ enum\), 609](#)  
[i2c\\_set\\_data\\_mode \(C++ function\), 607](#)  
[i2c\\_set\\_data\\_timing \(C++ function\), 607](#)  
[i2c\\_set\\_period \(C++ function\), 605](#)  
[i2c\\_set\\_pin \(C++ function\), 603](#)  
[i2c\\_set\\_start\\_timing \(C++ function\), 606](#)  
[i2c\\_set\\_stop\\_timing \(C++ function\), 606](#)

- i2c\_set\_timeout* (C++ function), 607  
*i2c\_slave\_read\_buffer* (C++ function), 605  
*i2c\_slave\_write\_buffer* (C++ function), 605  
*i2c\_trans\_mode\_t* (C++ enum), 609  
*i2s\_adc\_disable* (C++ function), 615  
*i2s\_adc\_enable* (C++ function), 615  
*I2S\_BITS\_PER\_SAMPLE\_16BIT* (C++ enumerator), 617  
*I2S\_BITS\_PER\_SAMPLE\_24BIT* (C++ enumerator), 617  
*I2S\_BITS\_PER\_SAMPLE\_32BIT* (C++ enumerator), 617  
*I2S\_BITS\_PER\_SAMPLE\_8BIT* (C++ enumerator), 617  
*i2s\_bits\_per\_sample\_t* (C++ enum), 617  
*I2S\_CHANNEL\_FMT\_ALL\_LEFT* (C++ enumerator), 617  
*I2S\_CHANNEL\_FMT\_ALL\_RIGHT* (C++ enumerator), 617  
*I2S\_CHANNEL\_FMT\_ONLY\_LEFT* (C++ enumerator), 617  
*I2S\_CHANNEL\_FMT\_ONLY\_RIGHT* (C++ enumerator), 617  
*I2S\_CHANNEL\_FMT\_RIGHT\_LEFT* (C++ enumerator), 617  
*i2s\_channel\_fmt\_t* (C++ enum), 617  
*I2S\_CHANNEL\_MONO* (C++ enumerator), 617  
*I2S\_CHANNEL\_STEREO* (C++ enumerator), 617  
*i2s\_channel\_t* (C++ enum), 617  
*I2S\_COMM\_FORMAT\_I2S* (C++ enumerator), 617  
*I2S\_COMM\_FORMAT\_I2S\_LSB* (C++ enumerator), 617  
*I2S\_COMM\_FORMAT\_I2S\_MSB* (C++ enumerator), 617  
*I2S\_COMM\_FORMAT\_PCM* (C++ enumerator), 617  
*I2S\_COMM\_FORMAT\_PCM\_LONG* (C++ enumerator), 617  
*I2S\_COMM\_FORMAT\_PCM\_SHORT* (C++ enumerator), 617  
*i2s\_comm\_format\_t* (C++ enum), 617  
*i2s\_config\_t* (C++ class), 615  
*i2s\_config\_t::bits\_per\_sample* (C++ member), 615  
*i2s\_config\_t::channel\_format* (C++ member), 615  
*i2s\_config\_t::communication\_format* (C++ member), 616  
*i2s\_config\_t::dma\_buf\_count* (C++ member), 616  
*i2s\_config\_t::dma\_buf\_len* (C++ member), 616  
*i2s\_config\_t::fixed\_mclk* (C++ member), 616  
*i2s\_config\_t::intr\_alloc\_flags* (C++ member), 616  
*i2s\_config\_t::mode* (C++ member), 615  
*i2s\_config\_t::sample\_rate* (C++ member), 615  
*i2s\_config\_t::tx\_desc\_auto\_clear* (C++ member), 616  
*i2s\_config\_t::use\_apll* (C++ member), 616  
*I2S\_DAC\_CHANNEL\_BOTH\_EN* (C++ enumerator), 619  
*I2S\_DAC\_CHANNEL\_DISABLE* (C++ enumerator), 618  
*I2S\_DAC\_CHANNEL\_LEFT\_EN* (C++ enumerator), 619  
*I2S\_DAC\_CHANNEL\_MAX* (C++ enumerator), 619  
*I2S\_DAC\_CHANNEL\_RIGHT\_EN* (C++ enumerator), 619  
*i2s\_dac\_mode\_t* (C++ enum), 618  
*i2s\_driver\_install* (C++ function), 612  
*i2s\_driver\_uninstall* (C++ function), 612  
*I2S\_EVENT\_DMA\_ERROR* (C++ enumerator), 618  
*I2S\_EVENT\_MAX* (C++ enumerator), 618  
*I2S\_EVENT\_RX\_DONE* (C++ enumerator), 618  
*i2s\_event\_t* (C++ class), 616  
*i2s\_event\_t::size* (C++ member), 616  
*i2s\_event\_t::type* (C++ member), 616  
*I2S\_EVENT\_TX\_DONE* (C++ enumerator), 618  
*i2s\_event\_type\_t* (C++ enum), 618  
*i2s\_get\_clk* (C++ function), 615  
*i2s\_isr\_handle\_t* (C++ type), 616  
*I2S\_MODE\_ADC\_BUILT\_IN* (C++ enumerator), 618  
*I2S\_MODE\_DAC\_BUILT\_IN* (C++ enumerator), 618  
*I2S\_MODE\_MASTER* (C++ enumerator), 618  
*I2S\_MODE\_PDM* (C++ enumerator), 618  
*I2S\_MODE\_RX* (C++ enumerator), 618  
*I2S\_MODE\_SLAVE* (C++ enumerator), 618  
*i2s\_mode\_t* (C++ enum), 618  
*I2S\_MODE\_TX* (C++ enumerator), 618  
*I2S\_NUM\_0* (C++ enumerator), 618  
*I2S\_NUM\_1* (C++ enumerator), 618  
*I2S\_NUM\_MAX* (C++ enumerator), 618  
*I2S\_PDM\_DSR\_16S* (C++ enumerator), 619  
*I2S\_PDM\_DSR\_8S* (C++ enumerator), 619  
*I2S\_PDM\_DSR\_MAX* (C++ enumerator), 619  
*i2s\_pdm\_dsr\_t* (C++ enum), 619  
*i2s\_pin\_config\_t* (C++ class), 616  
*i2s\_pin\_config\_t::bck\_io\_num* (C++ member), 616  
*i2s\_pin\_config\_t::data\_in\_num* (C++ member), 616  
*i2s\_pin\_config\_t::data\_out\_num* (C++ member), 616  
*i2s\_pin\_config\_t::ws\_io\_num* (C++ member), 616  
*I2S\_PIN\_NO\_CHANGE* (C 宏), 616  
*i2s\_port\_t* (C++ enum), 618  
*i2s\_read* (C++ function), 613  
*i2s\_set\_adc\_mode* (C++ function), 615  
*i2s\_set\_clk* (C++ function), 614  
*i2s\_set\_dac\_mode* (C++ function), 612  
*i2s\_set\_pdm\_rx\_down\_sample* (C++ function), 612  
*i2s\_set\_pin* (C++ function), 611

- i2s\_set\_sample\_rates* (C++ function), 614  
*i2s\_start* (C++ function), 614  
*i2s\_stop* (C++ function), 614  
*i2s\_write* (C++ function), 613  
*i2s\_write\_expand* (C++ function), 613  
*i2s\_zero\_dma\_buffer* (C++ function), 614  
I\_ADDI (C 宏), 1464  
I\_ADDR (C 宏), 1464  
I\_ANDI (C 宏), 1464  
I\_ANDR (C 宏), 1464  
I\_BGE (C 宏), 1463  
I\_BL (C 宏), 1463  
I\_BXFI (C 宏), 1464  
I\_BXFR (C 宏), 1464  
I\_BXI (C 宏), 1463  
I\_BXR (C 宏), 1463  
I\_BXZI (C 宏), 1463  
I\_BXZR (C 宏), 1463  
I\_DELAY (C 宏), 1462  
I\_END (C 宏), 1463  
I\_HALT (C 宏), 1462  
I\_LD (C 宏), 1463  
I\_LSHI (C 宏), 1464  
I\_LSHR (C 宏), 1464  
I\_MOVI (C 宏), 1464  
I\_MOVR (C 宏), 1464  
I\_ORI (C 宏), 1464  
I\_ORR (C 宏), 1464  
I\_RD\_REG (C 宏), 1463  
I\_RSHI (C 宏), 1464  
I\_RSHR (C 宏), 1464  
I\_ST (C 宏), 1463  
I\_SUBI (C 宏), 1464  
I\_SUBR (C 宏), 1464  
I\_WR\_REG (C 宏), 1463  
*intr\_handle\_data\_t* (C++ type), 1060  
*intr\_handle\_t* (C++ type), 1060  
*intr\_handler\_t* (C++ type), 1060  
IP\_EVENT\_AP\_STAIPASSIGNED (C++ enumerator), 555  
*ip\_event\_ap\_staipassigned\_t* (C++ class), 554  
*ip\_event\_ap\_staipassigned\_t::ip* (C++ member), 554  
IP\_EVENT\_ETH\_GOT\_IP (C++ enumerator), 556  
IP\_EVENT\_GOT\_IP6 (C++ enumerator), 556  
*ip\_event\_got\_ip6\_t* (C++ class), 553  
*ip\_event\_got\_ip6\_t::if\_index* (C++ member), 554  
*ip\_event\_got\_ip6\_t::ip6\_info* (C++ member), 554  
*ip\_event\_got\_ip\_t* (C++ class), 553  
*ip\_event\_got\_ip\_t::if\_index* (C++ member), 553  
*ip\_event\_got\_ip\_t::ip\_changed* (C++ member), 553  
*ip\_event\_got\_ip\_t::ip\_info* (C++ member), 553  
IP\_EVENT\_STA\_GOT\_IP (C++ enumerator), 555  
IP\_EVENT\_STA\_LOST\_IP (C++ enumerator), 555  
*ip\_event\_t* (C++ enum), 555  
**L**  
LEDC\_APB\_CLK (C++ enumerator), 630  
LEDC\_APB\_CLK\_HZ (C 宏), 629  
LEDC\_AUTO\_CLK (C++ enumerator), 629  
*ledc\_bind\_channel\_timer* (C++ function), 626  
LEDC\_CHANNEL\_0 (C++ enumerator), 630  
LEDC\_CHANNEL\_1 (C++ enumerator), 630  
LEDC\_CHANNEL\_2 (C++ enumerator), 630  
LEDC\_CHANNEL\_3 (C++ enumerator), 630  
LEDC\_CHANNEL\_4 (C++ enumerator), 630  
LEDC\_CHANNEL\_5 (C++ enumerator), 630  
LEDC\_CHANNEL\_6 (C++ enumerator), 630  
LEDC\_CHANNEL\_7 (C++ enumerator), 630  
*ledc\_channel\_config* (C++ function), 622  
*ledc\_channel\_config\_t* (C++ class), 628  
*ledc\_channel\_config\_t::channel* (C++ member), 628  
*ledc\_channel\_config\_t::duty* (C++ member), 628  
*ledc\_channel\_config\_t::gpio\_num* (C++ member), 628  
*ledc\_channel\_config\_t::hpoint* (C++ member), 628  
*ledc\_channel\_config\_t::intr\_type* (C++ member), 628  
*ledc\_channel\_config\_t::speed\_mode* (C++ member), 628  
*ledc\_channel\_config\_t::timer\_sel* (C++ member), 628  
LEDC\_CHANNEL\_MAX (C++ enumerator), 630  
*ledc\_channel\_t* (C++ enum), 630  
*ledc\_clk\_cfg\_t* (C++ enum), 629  
*ledc\_clk\_src\_t* (C++ enum), 630  
LEDC\_DUTY\_DIR\_DECREASE (C++ enumerator), 629  
LEDC\_DUTY\_DIR\_INCREASE (C++ enumerator), 629  
LEDC\_DUTY\_DIR\_MAX (C++ enumerator), 629  
*ledc\_duty\_direction\_t* (C++ enum), 629  
LEDC\_ERR\_DUTY (C 宏), 629  
LEDC\_ERR\_VAL (C 宏), 629  
*ledc\_fade\_func\_install* (C++ function), 626  
*ledc\_fade\_func\_uninstall* (C++ function), 627  
LEDC\_FADE\_MAX (C++ enumerator), 631  
*ledc\_fade\_mode\_t* (C++ enum), 631  
LEDC\_FADE\_NO\_WAIT (C++ enumerator), 631  
*ledc\_fade\_start* (C++ function), 627  
LEDC\_FADE\_WAIT\_DONE (C++ enumerator), 631  
*ledc\_get\_duty* (C++ function), 624  
*ledc\_get\_freq* (C++ function), 623  
*ledc\_get\_hpoint* (C++ function), 624  
LEDC\_HIGH\_SPEED\_MODE (C++ enumerator), 629  
LEDC\_INTR\_DISABLE (C++ enumerator), 629

- LEDC\_INTR\_FADE\_END (C++ enumerator), 629  
 ledc\_intr\_type\_t (C++ enum), 629  
 ledc\_isr\_handle\_t (C++ type), 629  
 ledc\_isr\_register (C++ function), 625  
 LEDC\_LOW\_SPEED\_MODE (C++ enumerator), 629  
 ledc\_mode\_t (C++ enum), 629  
 LEDC\_REF\_CLK\_HZ (C 宏), 629  
 LEDC\_REF\_TICK (C++ enumerator), 630  
 ledc\_set\_duty (C++ function), 624  
 ledc\_set\_duty\_and\_update (C++ function), 627  
 ledc\_set\_duty\_with\_hpoint (C++ function), 623  
 ledc\_set\_fade (C++ function), 624  
 ledc\_set\_fade\_step\_and\_start (C++ function), 627  
 ledc\_set\_fade\_time\_and\_start (C++ function), 627  
 ledc\_set\_fade\_with\_step (C++ function), 626  
 ledc\_set\_fade\_with\_time (C++ function), 626  
 ledc\_set\_freq (C++ function), 623  
 ledc\_set\_pin (C++ function), 623  
 LEDC\_SPEED\_MODE\_MAX (C++ enumerator), 629  
 ledc\_stop (C++ function), 623  
 LEDC\_TIMER\_0 (C++ enumerator), 630  
 LEDC\_TIMER\_1 (C++ enumerator), 630  
 LEDC\_TIMER\_10\_BIT (C++ enumerator), 631  
 LEDC\_TIMER\_11\_BIT (C++ enumerator), 631  
 LEDC\_TIMER\_12\_BIT (C++ enumerator), 631  
 LEDC\_TIMER\_13\_BIT (C++ enumerator), 631  
 LEDC\_TIMER\_14\_BIT (C++ enumerator), 631  
 LEDC\_TIMER\_15\_BIT (C++ enumerator), 631  
 LEDC\_TIMER\_16\_BIT (C++ enumerator), 631  
 LEDC\_TIMER\_17\_BIT (C++ enumerator), 631  
 LEDC\_TIMER\_18\_BIT (C++ enumerator), 631  
 LEDC\_TIMER\_19\_BIT (C++ enumerator), 631  
 LEDC\_TIMER\_1\_BIT (C++ enumerator), 630  
 LEDC\_TIMER\_2 (C++ enumerator), 630  
 LEDC\_TIMER\_20\_BIT (C++ enumerator), 631  
 LEDC\_TIMER\_2\_BIT (C++ enumerator), 630  
 LEDC\_TIMER\_3 (C++ enumerator), 630  
 LEDC\_TIMER\_3\_BIT (C++ enumerator), 630  
 LEDC\_TIMER\_4\_BIT (C++ enumerator), 631  
 LEDC\_TIMER\_5\_BIT (C++ enumerator), 631  
 LEDC\_TIMER\_6\_BIT (C++ enumerator), 631  
 LEDC\_TIMER\_7\_BIT (C++ enumerator), 631  
 LEDC\_TIMER\_8\_BIT (C++ enumerator), 631  
 LEDC\_TIMER\_9\_BIT (C++ enumerator), 631  
 LEDC\_TIMER\_BIT\_MAX (C++ enumerator), 631  
 ledc\_timer\_bit\_t (C++ enum), 630  
 ledc\_timer\_config (C++ function), 622  
 ledc\_timer\_config\_t (C++ class), 628  
 ledc\_timer\_config\_t::clk\_cfg (C++ member), 629  
 ledc\_timer\_config\_t::duty\_resolution (C++ member), 628  
 ledc\_timer\_config\_t::freq\_hz (C++ member), 628  
 ledc\_timer\_config\_t::speed\_mode (C++ member), 628  
 ledc\_timer\_config\_t::timer\_num (C++ member), 628  
 LEDC\_TIMER\_MAX (C++ enumerator), 630  
 ledc\_timer\_pause (C++ function), 625  
 ledc\_timer\_resume (C++ function), 625  
 ledc\_timer\_rst (C++ function), 625  
 ledc\_timer\_set (C++ function), 625  
 ledc\_timer\_t (C++ enum), 630  
 ledc\_update\_duty (C++ function), 622  
 LEDC\_USE\_APB\_CLK (C++ enumerator), 629  
 LEDC\_USE\_REF\_TICK (C++ enumerator), 629  
 LEDC\_USE\_RTC8M\_CLK (C++ enumerator), 629
- ## M
- M\_BGE (C 宏), 1465  
 M\_BL (C 宏), 1464  
 M\_BX (C 宏), 1465  
 M\_BXF (C 宏), 1465  
 M\_BXZ (C 宏), 1465  
 M\_LABEL (C 宏), 1464  
 MALLOC\_CAP\_32BIT (C 宏), 1038  
 MALLOC\_CAP\_8BIT (C 宏), 1038  
 MALLOC\_CAP\_DEFAULT (C 宏), 1038  
 MALLOC\_CAP\_DMA (C 宏), 1038  
 MALLOC\_CAP\_EXEC (C 宏), 1038  
 MALLOC\_CAP\_INTERNAL (C 宏), 1038  
 MALLOC\_CAP\_INVALID (C 宏), 1038  
 MALLOC\_CAP\_PID2 (C 宏), 1038  
 MALLOC\_CAP\_PID3 (C 宏), 1038  
 MALLOC\_CAP\_PID4 (C 宏), 1038  
 MALLOC\_CAP\_PID5 (C 宏), 1038  
 MALLOC\_CAP\_PID6 (C 宏), 1038  
 MALLOC\_CAP\_PID7 (C 宏), 1038  
 MALLOC\_CAP\_SPIRAM (C 宏), 1038  
 MAX\_BLE\_DEVNAME\_LEN (C 宏), 845  
 MAX\_FDS (C 宏), 919  
 MAX\_PASSPHRASE\_LEN (C 宏), 488  
 MAX\_SSID\_LEN (C 宏), 488  
 MAX\_WPS\_AP\_CRED (C 宏), 488  
 mbc\_master\_destroy (C++ function), 820  
 mbc\_master\_get\_cid\_info (C++ function), 822  
 mbc\_master\_get\_parameter (C++ function), 822  
 mbc\_master\_init (C++ function), 819  
 mbc\_master\_send\_request (C++ function), 821  
 mbc\_master\_set\_descriptor (C++ function), 821  
 mbc\_master\_set\_parameter (C++ function), 822  
 mbc\_master\_setup (C++ function), 819  
 mbc\_master\_start (C++ function), 820  
 mbc\_slave\_check\_event (C++ function), 821  
 mbc\_slave\_destroy (C++ function), 820  
 mbc\_slave\_get\_param\_info (C++ function), 821  
 mbc\_slave\_init (C++ function), 819

- mbc\_slave\_set\_descriptor (C++ function), 820  
 mbc\_slave\_setup (C++ function), 819  
 mbc\_slave\_start (C++ function), 820  
 MCPWM0A (C++ enumerator), 645  
 MCPWM0B (C++ enumerator), 645  
 MCPWM1A (C++ enumerator), 645  
 MCPWM1B (C++ enumerator), 645  
 MCPWM2A (C++ enumerator), 645  
 MCPWM2B (C++ enumerator), 645  
 mcpwm\_action\_on\_pwmxa\_t (C++ enum), 647  
 mcpwm\_action\_on\_pwmxb\_t (C++ enum), 648  
 MCPWM\_ACTIVE\_HIGH\_COMPLIMENT\_MODE (C++ enumerator), 648  
 MCPWM\_ACTIVE\_HIGH\_MODE (C++ enumerator), 648  
 MCPWM\_ACTIVE\_LOW\_COMPLIMENT\_MODE (C++ enumerator), 649  
 MCPWM\_ACTIVE\_LOW\_MODE (C++ enumerator), 648  
 MCPWM\_ACTIVE\_RED\_FED\_FROM\_PWMXA (C++ enumerator), 649  
 MCPWM\_ACTIVE\_RED\_FED\_FROM\_PWMXB (C++ enumerator), 649  
 MCPWM\_BYPASS\_FED (C++ enumerator), 648  
 MCPWM\_BYPASS\_RED (C++ enumerator), 648  
 MCPWM\_CAP\_0 (C++ enumerator), 645  
 MCPWM\_CAP\_1 (C++ enumerator), 645  
 MCPWM\_CAP\_2 (C++ enumerator), 645  
 mcpwm\_capture\_disable (C++ function), 642  
 mcpwm\_capture\_enable (C++ function), 642  
 mcpwm\_capture\_on\_edge\_t (C++ enum), 648  
 mcpwm\_capture\_signal\_get\_edge (C++ function), 642  
 mcpwm\_capture\_signal\_get\_value (C++ function), 642  
 mcpwm\_capture\_signal\_t (C++ enum), 648  
 mcpwm\_carrier\_config\_t (C++ class), 644  
 mcpwm\_carrier\_config\_t::carrier\_duty (C++ member), 644  
 mcpwm\_carrier\_config\_t::carrier\_ivt\_mode (C++ member), 645  
 mcpwm\_carrier\_config\_t::carrier\_os\_mode (C++ member), 644  
 mcpwm\_carrier\_config\_t::carrier\_period (C++ member), 644  
 mcpwm\_carrier\_config\_t::pulse\_width\_in\_os (C++ member), 644  
 mcpwm\_carrier\_disable (C++ function), 639  
 mcpwm\_carrier\_enable (C++ function), 639  
 mcpwm\_carrier\_init (C++ function), 639  
 mcpwm\_carrier\_oneshot\_mode\_disable (C++ function), 640  
 mcpwm\_carrier\_oneshot\_mode\_enable (C++ function), 640  
 mcpwm\_carrier\_os\_t (C++ enum), 647  
 MCPWM\_CARRIER\_OUT\_IVT\_DIS (C++ enumerator), 647  
 MCPWM\_CARRIER\_OUT\_IVT\_EN (C++ enumerator), 647  
 mcpwm\_carrier\_out\_ivt\_t (C++ enum), 647  
 mcpwm\_carrier\_output\_invert (C++ function), 640  
 mcpwm\_carrier\_set\_duty\_cycle (C++ function), 640  
 mcpwm\_carrier\_set\_period (C++ function), 640  
 mcpwm\_config\_t (C++ class), 644  
 mcpwm\_config\_t::cmpr\_a (C++ member), 644  
 mcpwm\_config\_t::cmpr\_b (C++ member), 644  
 mcpwm\_config\_t::counter\_mode (C++ member), 644  
 mcpwm\_config\_t::duty\_mode (C++ member), 644  
 mcpwm\_config\_t::frequency (C++ member), 644  
 MCPWM\_COUNTER\_MAX (C++ enumerator), 646  
 mcpwm\_counter\_type\_t (C++ enum), 646  
 mcpwm\_deadtime\_disable (C++ function), 641  
 mcpwm\_deadtime\_enable (C++ function), 640  
 MCPWM\_DEADTIME\_TYPE\_MAX (C++ enumerator), 649  
 mcpwm\_deadtime\_type\_t (C++ enum), 648  
 MCPWM\_DOWN\_COUNTER (C++ enumerator), 646  
 MCPWM\_DUTY\_MODE\_0 (C++ enumerator), 646  
 MCPWM\_DUTY\_MODE\_1 (C++ enumerator), 646  
 MCPWM\_DUTY\_MODE\_MAX (C++ enumerator), 646  
 mcpwm\_duty\_type\_t (C++ enum), 646  
 MCPWM\_FAULT\_0 (C++ enumerator), 645  
 MCPWM\_FAULT\_1 (C++ enumerator), 645  
 MCPWM\_FAULT\_2 (C++ enumerator), 645  
 mcpwm\_fault\_deinit (C++ function), 642  
 mcpwm\_fault\_init (C++ function), 641  
 mcpwm\_fault\_input\_level\_t (C++ enum), 647  
 mcpwm\_fault\_set\_cyc\_mode (C++ function), 641  
 mcpwm\_fault\_set\_oneshot\_mode (C++ function), 641  
 mcpwm\_fault\_signal\_t (C++ enum), 647  
 MCPWM\_FORCE\_MCPWMXA\_HIGH (C++ enumerator), 648  
 MCPWM\_FORCE\_MCPWMXA\_LOW (C++ enumerator), 648  
 MCPWM\_FORCE\_MCPWMXB\_HIGH (C++ enumerator), 648  
 MCPWM\_FORCE\_MCPWMXB\_LOW (C++ enumerator), 648  
 mcpwm\_get\_duty (C++ function), 638  
 mcpwm\_get\_frequency (C++ function), 638  
 mcpwm\_gpio\_init (C++ function), 637  
 MCPWM\_HIGH\_LEVEL\_TGR (C++ enumerator), 647  
 mcpwm\_init (C++ function), 637  
 mcpwm\_io\_signals\_t (C++ enum), 645  
 mcpwm\_isr\_register (C++ function), 643  
 MCPWM\_LOW\_LEVEL\_TGR (C++ enumerator), 647  
 MCPWM\_NEG\_EDGE (C++ enumerator), 648

- MCPWM\_NO\_CHANGE\_IN\_MCPWMXA (C++ enumerator), 647
- MCPWM\_NO\_CHANGE\_IN\_MCPWMXB (C++ enumerator), 648
- MCPWM\_ONESHOT\_MODE\_DIS (C++ enumerator), 647
- MCPWM\_ONESHOT\_MODE\_EN (C++ enumerator), 647
- mcpwm\_operator\_t (C++ enum), 646
- MCPWM\_OPR\_A (C++ enumerator), 646
- MCPWM\_OPR\_B (C++ enumerator), 646
- MCPWM\_OPR\_MAX (C++ enumerator), 646
- mcpwm\_pin\_config\_t (C++ class), 643
- mcpwm\_pin\_config\_t::mcpwm0a\_out\_num (C++ member), 643
- mcpwm\_pin\_config\_t::mcpwm0b\_out\_num (C++ member), 643
- mcpwm\_pin\_config\_t::mcpwm1a\_out\_num (C++ member), 643
- mcpwm\_pin\_config\_t::mcpwm1b\_out\_num (C++ member), 643
- mcpwm\_pin\_config\_t::mcpwm2a\_out\_num (C++ member), 643
- mcpwm\_pin\_config\_t::mcpwm2b\_out\_num (C++ member), 643
- mcpwm\_pin\_config\_t::mcpwm\_cap0\_in\_num (C++ member), 644
- mcpwm\_pin\_config\_t::mcpwm\_cap1\_in\_num (C++ member), 644
- mcpwm\_pin\_config\_t::mcpwm\_cap2\_in\_num (C++ member), 644
- mcpwm\_pin\_config\_t::mcpwm\_fault0\_in\_num (C++ member), 644
- mcpwm\_pin\_config\_t::mcpwm\_fault1\_in\_num (C++ member), 644
- mcpwm\_pin\_config\_t::mcpwm\_fault2\_in\_num (C++ member), 644
- mcpwm\_pin\_config\_t::mcpwm\_sync0\_in\_num (C++ member), 644
- mcpwm\_pin\_config\_t::mcpwm\_sync1\_in\_num (C++ member), 644
- mcpwm\_pin\_config\_t::mcpwm\_sync2\_in\_num (C++ member), 644
- MCPWM\_POS\_EDGE (C++ enumerator), 648
- MCPWM\_SELECT\_CAP0 (C++ enumerator), 648
- MCPWM\_SELECT\_CAP1 (C++ enumerator), 648
- MCPWM\_SELECT\_CAP2 (C++ enumerator), 648
- MCPWM\_SELECT\_F0 (C++ enumerator), 647
- MCPWM\_SELECT\_F1 (C++ enumerator), 647
- MCPWM\_SELECT\_F2 (C++ enumerator), 647
- MCPWM\_SELECT\_SYNC0 (C++ enumerator), 647
- MCPWM\_SELECT\_SYNC1 (C++ enumerator), 647
- MCPWM\_SELECT\_SYNC2 (C++ enumerator), 647
- mcpwm\_set\_duty (C++ function), 637
- mcpwm\_set\_duty\_in\_us (C++ function), 638
- mcpwm\_set\_duty\_type (C++ function), 638
- mcpwm\_set\_frequency (C++ function), 637
- mcpwm\_set\_pin (C++ function), 637
- mcpwm\_set\_signal\_high (C++ function), 638
- mcpwm\_set\_signal\_low (C++ function), 639
- mcpwm\_start (C++ function), 639
- mcpwm\_stop (C++ function), 639
- MCPWM\_SYNC\_0 (C++ enumerator), 645
- MCPWM\_SYNC\_1 (C++ enumerator), 645
- MCPWM\_SYNC\_2 (C++ enumerator), 645
- mcpwm\_sync\_disable (C++ function), 643
- mcpwm\_sync\_enable (C++ function), 643
- mcpwm\_sync\_signal\_t (C++ enum), 647
- MCPWM\_TIMER\_0 (C++ enumerator), 646
- MCPWM\_TIMER\_1 (C++ enumerator), 646
- MCPWM\_TIMER\_2 (C++ enumerator), 646
- MCPWM\_TIMER\_MAX (C++ enumerator), 646
- mcpwm\_timer\_t (C++ enum), 646
- MCPWM\_TOG\_MCPWMXA (C++ enumerator), 648
- MCPWM\_TOG\_MCPWMXB (C++ enumerator), 648
- MCPWM\_UNIT\_0 (C++ enumerator), 645
- MCPWM\_UNIT\_1 (C++ enumerator), 646
- MCPWM\_UNIT\_MAX (C++ enumerator), 646
- mcpwm\_unit\_t (C++ enum), 645
- MCPWM\_UP\_COUNTER (C++ enumerator), 646
- MCPWM\_UP\_DOWN\_COUNTER (C++ enumerator), 646
- mdns\_free (C++ function), 758
- mdns\_handle\_system\_event (C++ function), 761
- mdns\_hostname\_set (C++ function), 758
- mdns\_init (C++ function), 757
- mdns\_instance\_name\_set (C++ function), 758
- mdns\_ip\_addr\_s (C++ class), 762
- mdns\_ip\_addr\_s::addr (C++ member), 762
- mdns\_ip\_addr\_s::next (C++ member), 762
- mdns\_ip\_addr\_t (C++ type), 763
- MDNS\_IP\_PROTOCOL\_MAX (C++ enumerator), 763
- mdns\_ip\_protocol\_t (C++ enum), 763
- MDNS\_IP\_PROTOCOL\_V4 (C++ enumerator), 763
- MDNS\_IP\_PROTOCOL\_V6 (C++ enumerator), 763
- mdns\_query (C++ function), 760
- mdns\_query\_a (C++ function), 761
- mdns\_query\_aaaa (C++ function), 761
- mdns\_query\_ptr (C++ function), 760
- mdns\_query\_results\_free (C++ function), 760
- mdns\_query\_srv (C++ function), 760
- mdns\_query\_txt (C++ function), 761
- mdns\_result\_s (C++ class), 762
- mdns\_result\_s::addr (C++ member), 762
- mdns\_result\_s::hostname (C++ member), 762
- mdns\_result\_s::instance\_name (C++ member), 762
- mdns\_result\_s::ip\_protocol (C++ member), 762
- mdns\_result\_s::next (C++ member), 762
- mdns\_result\_s::port (C++ member), 762
- mdns\_result\_s::tcpip\_if (C++ member), 762
- mdns\_result\_s::txt (C++ member), 762
- mdns\_result\_s::txt\_count (C++ member), 762
- mdns\_result\_t (C++ type), 763
- mdns\_service\_add (C++ function), 758

- mdns\_service\_instance\_name\_set (C++ function), 758
- mdns\_service\_port\_set (C++ function), 759
- mdns\_service\_remove (C++ function), 758
- mdns\_service\_remove\_all (C++ function), 760
- mdns\_service\_txt\_item\_remove (C++ function), 759
- mdns\_service\_txt\_item\_set (C++ function), 759
- mdns\_service\_txt\_set (C++ function), 759
- mdns\_txt\_item\_t (C++ class), 761
- mdns\_txt\_item\_t::key (C++ member), 761
- mdns\_txt\_item\_t::value (C++ member), 761
- MDNS\_TYPE\_A (C 宏), 762
- MDNS\_TYPE\_AAAA (C 宏), 762
- MDNS\_TYPE\_ANY (C 宏), 762
- MDNS\_TYPE\_NSEC (C 宏), 762
- MDNS\_TYPE\_OPT (C 宏), 762
- MDNS\_TYPE\_PTR (C 宏), 762
- MDNS\_TYPE\_SRV (C 宏), 762
- MDNS\_TYPE\_TXT (C 宏), 762
- mesh\_addr\_t (C++ union), 521
- mesh\_addr\_t::addr (C++ member), 521
- mesh\_addr\_t::mip (C++ member), 521
- mesh\_ap\_cfg\_t (C++ class), 525
- mesh\_ap\_cfg\_t::max\_connection (C++ member), 525
- mesh\_ap\_cfg\_t::password (C++ member), 525
- MESH\_ASSOC\_FLAG\_NETWORK\_FREE (C 宏), 528
- MESH\_ASSOC\_FLAG\_ROOT\_FIXED (C 宏), 528
- MESH\_ASSOC\_FLAG\_ROOTS\_FOUND (C 宏), 528
- MESH\_ASSOC\_FLAG\_VOTE\_IN\_PROGRESS (C 宏), 528
- mesh\_cfg\_t (C++ class), 525
- mesh\_cfg\_t::allow\_channel\_switch (C++ member), 525
- mesh\_cfg\_t::channel (C++ member), 525
- mesh\_cfg\_t::crypto\_funcs (C++ member), 526
- mesh\_cfg\_t::mesh\_ap (C++ member), 526
- mesh\_cfg\_t::mesh\_id (C++ member), 526
- mesh\_cfg\_t::router (C++ member), 526
- MESH\_DATA\_DROP (C 宏), 528
- MESH\_DATA\_ENC (C 宏), 528
- MESH\_DATA\_FROMDS (C 宏), 528
- MESH\_DATA\_GROUP (C 宏), 528
- MESH\_DATA\_NONBLOCK (C 宏), 528
- MESH\_DATA\_P2P (C 宏), 528
- mesh\_data\_t (C++ class), 524
- mesh\_data\_t::data (C++ member), 525
- mesh\_data\_t::proto (C++ member), 525
- mesh\_data\_t::size (C++ member), 525
- mesh\_data\_t::tos (C++ member), 525
- MESH\_DATA\_TODS (C 宏), 528
- mesh\_disconnect\_reason\_t (C++ enum), 531
- MESH\_EVENT\_CHANNEL\_SWITCH (C++ enumerator), 529
- mesh\_event\_channel\_switch\_t (C++ class), 522
- mesh\_event\_channel\_switch\_t::channel (C++ member), 522
- MESH\_EVENT\_CHILD\_CONNECTED (C++ enumerator), 529
- mesh\_event\_child\_connected\_t (C++ type), 528
- MESH\_EVENT\_CHILD\_DISCONNECTED (C++ enumerator), 529
- mesh\_event\_child\_disconnected\_t (C++ type), 529
- mesh\_event\_connected\_t (C++ class), 522
- mesh\_event\_connected\_t::connected (C++ member), 522
- mesh\_event\_connected\_t::self\_layer (C++ member), 522
- mesh\_event\_disconnected\_t (C++ type), 528
- MESH\_EVENT\_FIND\_NETWORK (C++ enumerator), 530
- mesh\_event\_find\_network\_t (C++ class), 523
- mesh\_event\_find\_network\_t::channel (C++ member), 523
- mesh\_event\_find\_network\_t::router\_bssid (C++ member), 523
- mesh\_event\_id\_t (C++ enum), 529
- mesh\_event\_info\_t (C++ union), 521
- mesh\_event\_info\_t::channel\_switch (C++ member), 521
- mesh\_event\_info\_t::child\_connected (C++ member), 521
- mesh\_event\_info\_t::child\_disconnected (C++ member), 521
- mesh\_event\_info\_t::connected (C++ member), 521
- mesh\_event\_info\_t::disconnected (C++ member), 521
- mesh\_event\_info\_t::find\_network (C++ member), 522
- mesh\_event\_info\_t::got\_ip (C++ member), 521
- mesh\_event\_info\_t::layer\_change (C++ member), 521
- mesh\_event\_info\_t::network\_state (C++ member), 522
- mesh\_event\_info\_t::no\_parent (C++ member), 521
- mesh\_event\_info\_t::root\_addr (C++ member), 521
- mesh\_event\_info\_t::root\_conflict (C++ member), 522
- mesh\_event\_info\_t::root\_fixed (C++ member), 522
- mesh\_event\_info\_t::router\_switch (C++ member), 522
- mesh\_event\_info\_t::routing\_table (C++ member), 521
- mesh\_event\_info\_t::scan\_done (C++ mem-

- ber), 522
- mesh\_event\_info\_t::switch\_req (C++ member), 521
- mesh\_event\_info\_t::toDS\_state (C++ member), 521
- mesh\_event\_info\_t::vote\_started (C++ member), 521
- MESH\_EVENT\_LAYER\_CHANGE (C++ enumerator), 529
- mesh\_event\_layer\_change\_t (C++ class), 523
- mesh\_event\_layer\_change\_t::new\_layer (C++ member), 523
- MESH\_EVENT\_MAX (C++ enumerator), 530
- MESH\_EVENT\_NETWORK\_STATE (C++ enumerator), 530
- mesh\_event\_network\_state\_t (C++ class), 524
- mesh\_event\_network\_state\_t::is\_rootless (C++ member), 524
- MESH\_EVENT\_NO\_PARENT\_FOUND (C++ enumerator), 529
- mesh\_event\_no\_parent\_found\_t (C++ class), 523
- mesh\_event\_no\_parent\_found\_t::scan\_times (C++ member), 523
- MESH\_EVENT\_PARENT\_CONNECTED (C++ enumerator), 529
- MESH\_EVENT\_PARENT\_DISCONNECTED (C++ enumerator), 529
- MESH\_EVENT\_ROOT\_ADDRESS (C++ enumerator), 529
- mesh\_event\_root\_address\_t (C++ type), 528
- MESH\_EVENT\_ROOT\_ASKED\_YIELD (C++ enumerator), 529
- mesh\_event\_root\_conflict\_t (C++ class), 523
- mesh\_event\_root\_conflict\_t::addr (C++ member), 524
- mesh\_event\_root\_conflict\_t::capacity (C++ member), 524
- mesh\_event\_root\_conflict\_t::rssi (C++ member), 524
- MESH\_EVENT\_ROOT\_FIXED (C++ enumerator), 530
- mesh\_event\_root\_fixed\_t (C++ class), 524
- mesh\_event\_root\_fixed\_t::is\_fixed (C++ member), 524
- mesh\_event\_root\_got\_ip\_t (C++ type), 528
- MESH\_EVENT\_ROOT\_SWITCH\_ACK (C++ enumerator), 529
- MESH\_EVENT\_ROOT\_SWITCH\_REQ (C++ enumerator), 529
- mesh\_event\_root\_switch\_req\_t (C++ class), 523
- mesh\_event\_root\_switch\_req\_t::rc\_addr (C++ member), 523
- mesh\_event\_root\_switch\_req\_t::reason (C++ member), 523
- MESH\_EVENT\_ROUTER\_SWITCH (C++ enumerator), 530
- mesh\_event\_router\_switch\_t (C++ type), 529
- MESH\_EVENT\_ROUTING\_TABLE\_ADD (C++ enumerator), 529
- mesh\_event\_routing\_table\_change\_t (C++ class), 524
- mesh\_event\_routing\_table\_change\_t::rt\_size\_change (C++ member), 524
- mesh\_event\_routing\_table\_change\_t::rt\_size\_new (C++ member), 524
- MESH\_EVENT\_ROUTING\_TABLE\_REMOVE (C++ enumerator), 529
- MESH\_EVENT\_SCAN\_DONE (C++ enumerator), 530
- mesh\_event\_scan\_done\_t (C++ class), 524
- mesh\_event\_scan\_done\_t::number (C++ member), 524
- MESH\_EVENT\_STARTED (C++ enumerator), 529
- MESH\_EVENT\_STOP\_RECONNECTION (C++ enumerator), 530
- MESH\_EVENT\_STOPPED (C++ enumerator), 529
- MESH\_EVENT\_TODS\_STATE (C++ enumerator), 529
- mesh\_event\_toDS\_state\_t (C++ enum), 531
- MESH\_EVENT\_VOTE\_STARTED (C++ enumerator), 529
- mesh\_event\_vote\_started\_t (C++ class), 523
- mesh\_event\_vote\_started\_t::attempts (C++ member), 523
- mesh\_event\_vote\_started\_t::rc\_addr (C++ member), 523
- mesh\_event\_vote\_started\_t::reason (C++ member), 523
- MESH\_EVENT\_VOTE\_STOPPED (C++ enumerator), 529
- MESH\_IDLE (C++ enumerator), 530
- MESH\_INIT\_CONFIG\_DEFAULT (C 宏), 528
- MESH\_LEAF (C++ enumerator), 530
- MESH\_MPS (C 宏), 527
- MESH\_MTU (C 宏), 527
- MESH\_NODE (C++ enumerator), 530
- MESH\_OPT\_RECV\_DS\_ADDR (C 宏), 528
- MESH\_OPT\_SEND\_GROUP (C 宏), 528
- mesh\_opt\_t (C++ class), 524
- mesh\_opt\_t::len (C++ member), 524
- mesh\_opt\_t::type (C++ member), 524
- mesh\_opt\_t::val (C++ member), 524
- MESH\_PROTO\_BIN (C++ enumerator), 530
- MESH\_PROTO\_HTTP (C++ enumerator), 530
- MESH\_PROTO\_JSON (C++ enumerator), 530
- MESH\_PROTO\_MQTT (C++ enumerator), 530
- mesh\_proto\_t (C++ enum), 530
- mesh\_rc\_config\_t (C++ union), 522
- mesh\_rc\_config\_t::attempts (C++ member), 522
- mesh\_rc\_config\_t::rc\_addr (C++ member), 522
- MESH\_REASON\_CYCLIC (C++ enumerator), 531
- MESH\_REASON\_DIFF\_ID (C++ enumerator), 531



- MESH\_REASON\_EMPTY\_PASSWORD (C++ *enumerator*), 531
- MESH\_REASON\_IE\_UNKNOWN (C++ *enumerator*), 531
- MESH\_REASON\_LEAF (C++ *enumerator*), 531
- MESH\_REASON\_PARENT\_IDLE (C++ *enumerator*), 531
- MESH\_REASON\_PARENT\_STOPPED (C++ *enumerator*), 531
- MESH\_REASON\_PARENT\_UNENCRYPTED (C++ *enumerator*), 531
- MESH\_REASON\_PARENT\_WORSE (C++ *enumerator*), 531
- MESH\_REASON\_ROOTS (C++ *enumerator*), 531
- MESH\_REASON\_SCAN\_FAIL (C++ *enumerator*), 531
- MESH\_REASON\_WAIVE\_ROOT (C++ *enumerator*), 531
- MESH\_ROOT (C++ *enumerator*), 530
- MESH\_ROOT\_LAYER (C 宏), 527
- mesh\_router\_t (C++ *class*), 525
- mesh\_router\_t::allow\_router\_switch (C++ *member*), 525
- mesh\_router\_t::bssid (C++ *member*), 525
- mesh\_router\_t::password (C++ *member*), 525
- mesh\_router\_t::ssid (C++ *member*), 525
- mesh\_router\_t::ssid\_len (C++ *member*), 525
- mesh\_rx\_pending\_t (C++ *class*), 526
- mesh\_rx\_pending\_t::toDS (C++ *member*), 526
- mesh\_rx\_pending\_t::toSelf (C++ *member*), 526
- MESH\_TODS\_REACHABLE (C++ *enumerator*), 531
- MESH\_TODS\_UNREACHABLE (C++ *enumerator*), 531
- MESH\_TOS\_DEF (C++ *enumerator*), 531
- MESH\_TOS\_E2E (C++ *enumerator*), 531
- MESH\_TOS\_P2P (C++ *enumerator*), 530
- mesh\_tos\_t (C++ *enum*), 530
- mesh\_tx\_pending\_t (C++ *class*), 526
- mesh\_tx\_pending\_t::broadcast (C++ *member*), 526
- mesh\_tx\_pending\_t::mgmt (C++ *member*), 526
- mesh\_tx\_pending\_t::to\_child (C++ *member*), 526
- mesh\_tx\_pending\_t::to\_child\_p2p (C++ *member*), 526
- mesh\_tx\_pending\_t::to\_parent (C++ *member*), 526
- mesh\_tx\_pending\_t::to\_parent\_p2p (C++ *member*), 526
- mesh\_type\_t (C++ *enum*), 530
- MESH\_VOTE\_REASON\_CHILD\_INITIATED (C++ *enumerator*), 531
- MESH\_VOTE\_REASON\_ROOT\_INITIATED (C++ *enumerator*), 531
- mesh\_vote\_reason\_t (C++ *enum*), 531
- mesh\_vote\_t (C++ *class*), 526
- mesh\_vote\_t::config (C++ *member*), 526
- mesh\_vote\_t::is\_rc\_specified (C++ *member*), 526
- mesh\_vote\_t::percentage (C++ *member*), 526
- mip\_t (C++ *class*), 522
- mip\_t::ip4 (C++ *member*), 522
- mip\_t::port (C++ *member*), 522
- MQTT\_CONNECTION\_ACCEPTED (C++ *enumerator*), 818
- MQTT\_CONNECTION\_REFUSE\_BAD\_USERNAME (C++ *enumerator*), 818
- MQTT\_CONNECTION\_REFUSE\_ID\_REJECTED (C++ *enumerator*), 818
- MQTT\_CONNECTION\_REFUSE\_NOT\_AUTHORIZED (C++ *enumerator*), 818
- MQTT\_CONNECTION\_REFUSE\_PROTOCOL (C++ *enumerator*), 818
- MQTT\_CONNECTION\_REFUSE\_SERVER\_UNAVAILABLE (C++ *enumerator*), 818
- MQTT\_ERROR\_TYPE\_CONNECTION\_REFUSED (C++ *enumerator*), 818
- MQTT\_ERROR\_TYPE\_ESP\_TLS (C 宏), 816
- MQTT\_ERROR\_TYPE\_NONE (C++ *enumerator*), 818
- MQTT\_ERROR\_TYPE\_TCP\_TRANSPORT (C++ *enumerator*), 818
- MQTT\_EVENT\_ANY (C++ *enumerator*), 817
- MQTT\_EVENT\_BEFORE\_CONNECT (C++ *enumerator*), 818
- mqtt\_event\_callback\_t (C++ *type*), 817
- MQTT\_EVENT\_CONNECTED (C++ *enumerator*), 817
- MQTT\_EVENT\_DATA (C++ *enumerator*), 817
- MQTT\_EVENT\_DISCONNECTED (C++ *enumerator*), 817
- MQTT\_EVENT\_ERROR (C++ *enumerator*), 817
- MQTT\_EVENT\_PUBLISHED (C++ *enumerator*), 817
- MQTT\_EVENT\_SUBSCRIBED (C++ *enumerator*), 817
- MQTT\_EVENT\_UNSUBSCRIBED (C++ *enumerator*), 817
- MQTT\_PROTOCOL\_UNDEFINED (C++ *enumerator*), 818
- MQTT\_PROTOCOL\_V\_3\_1 (C++ *enumerator*), 818
- MQTT\_PROTOCOL\_V\_3\_1\_1 (C++ *enumerator*), 818
- MQTT\_TRANSPORT\_OVER\_SSL (C++ *enumerator*), 818
- MQTT\_TRANSPORT\_OVER\_TCP (C++ *enumerator*), 818
- MQTT\_TRANSPORT\_OVER\_WS (C++ *enumerator*), 818
- MQTT\_TRANSPORT\_OVER\_WSS (C++ *enumerator*), 818
- MQTT\_TRANSPORT\_UNKNOWN (C++ *enumerator*), 818
- multi\_heap\_check (C++ *function*), 1041
- multi\_heap\_dump (C++ *function*), 1041
- multi\_heap\_free (C++ *function*), 1040
- multi\_heap\_free\_size (C++ *function*), 1041
- multi\_heap\_get\_allocated\_size (C++ *function*), 1041
- multi\_heap\_get\_info (C++ *function*), 1042
- multi\_heap\_handle\_t (C++ *type*), 1042
- multi\_heap\_info\_t (C++ *class*), 1042

- multi\_heap\_info\_t::allocated\_blocks (C++ member), 1042  
 multi\_heap\_info\_t::free\_blocks (C++ member), 1042  
 multi\_heap\_info\_t::largest\_free\_block (C++ member), 1042  
 multi\_heap\_info\_t::minimum\_free\_bytes (C++ member), 1042  
 multi\_heap\_info\_t::total\_allocated\_bytes (C++ member), 1042  
 multi\_heap\_info\_t::total\_blocks (C++ member), 1042  
 multi\_heap\_info\_t::total\_free\_bytes (C++ member), 1042  
 multi\_heap\_malloc (C++ function), 1040  
 multi\_heap\_minimum\_free\_size (C++ function), 1042  
 multi\_heap\_realloc (C++ function), 1040  
 multi\_heap\_register (C++ function), 1041  
 multi\_heap\_set\_lock (C++ function), 1041
- ## N
- name\_uuid (C++ class), 844  
 name\_uuid::name (C++ member), 845  
 name\_uuid::uuid (C++ member), 845  
 nvs\_close (C++ function), 900  
 nvs\_commit (C++ function), 900  
 NVS\_DEFAULT\_PART\_NAME (C 宏), 904  
 nvs\_entry\_find (C++ function), 901  
 nvs\_entry\_info (C++ function), 902  
 nvs\_entry\_info\_t (C++ class), 902  
 nvs\_entry\_info\_t::key (C++ member), 902  
 nvs\_entry\_info\_t::namespace\_name (C++ member), 902  
 nvs\_entry\_info\_t::type (C++ member), 902  
 nvs\_entry\_next (C++ function), 902  
 nvs\_erase\_all (C++ function), 899  
 nvs\_erase\_key (C++ function), 899  
 nvs\_flash\_deinit (C++ function), 894  
 nvs\_flash\_deinit\_partition (C++ function), 894  
 nvs\_flash\_erase (C++ function), 894  
 nvs\_flash\_erase\_partition (C++ function), 894  
 nvs\_flash\_erase\_partition\_ptr (C++ function), 895  
 nvs\_flash\_generate\_keys (C++ function), 895  
 nvs\_flash\_init (C++ function), 893  
 nvs\_flash\_init\_partition (C++ function), 894  
 nvs\_flash\_init\_partition\_ptr (C++ function), 894  
 nvs\_flash\_read\_security\_cfg (C++ function), 896  
 nvs\_flash\_secure\_init (C++ function), 895  
 nvs\_flash\_secure\_init\_partition (C++ function), 895  
 nvs\_get\_blob (C++ function), 898  
 nvs\_get\_i16 (C++ function), 897  
 nvs\_get\_i32 (C++ function), 897  
 nvs\_get\_i64 (C++ function), 897  
 nvs\_get\_i8 (C++ function), 897  
 nvs\_get\_stats (C++ function), 900  
 nvs\_get\_str (C++ function), 897  
 nvs\_get\_u16 (C++ function), 897  
 nvs\_get\_u32 (C++ function), 897  
 nvs\_get\_u64 (C++ function), 897  
 nvs\_get\_u8 (C++ function), 897  
 nvs\_get\_used\_entry\_count (C++ function), 901  
 nvs\_handle (C++ type), 904  
 nvs\_handle\_t (C++ type), 904  
 nvs\_iterator\_t (C++ type), 904  
 NVS\_KEY\_SIZE (C 宏), 896  
 nvs\_open (C++ function), 898  
 nvs\_open\_from\_partition (C++ function), 898  
 nvs\_open\_mode (C++ type), 904  
 nvs\_open\_mode\_t (C++ enum), 904  
 NVS\_READONLY (C++ enumerator), 904  
 NVS\_READWRITE (C++ enumerator), 904  
 nvs\_release\_iterator (C++ function), 902  
 nvs\_sec\_cfg\_t (C++ class), 896  
 nvs\_sec\_cfg\_t::eky (C++ member), 896  
 nvs\_sec\_cfg\_t::tky (C++ member), 896  
 nvs\_set\_blob (C++ function), 899  
 nvs\_set\_i16 (C++ function), 897  
 nvs\_set\_i32 (C++ function), 897  
 nvs\_set\_i64 (C++ function), 897  
 nvs\_set\_i8 (C++ function), 896  
 nvs\_set\_str (C++ function), 897  
 nvs\_set\_u16 (C++ function), 897  
 nvs\_set\_u32 (C++ function), 897  
 nvs\_set\_u64 (C++ function), 897  
 nvs\_set\_u8 (C++ function), 897  
 nvs\_stats\_t (C++ class), 902  
 nvs\_stats\_t::free\_entries (C++ member), 902  
 nvs\_stats\_t::namespace\_count (C++ member), 902  
 nvs\_stats\_t::total\_entries (C++ member), 902  
 nvs\_stats\_t::used\_entries (C++ member), 902  
 NVS\_TYPE\_ANY (C++ enumerator), 905  
 NVS\_TYPE\_BLOB (C++ enumerator), 904  
 NVS\_TYPE\_I16 (C++ enumerator), 904  
 NVS\_TYPE\_I32 (C++ enumerator), 904  
 NVS\_TYPE\_I64 (C++ enumerator), 904  
 NVS\_TYPE\_I8 (C++ enumerator), 904  
 NVS\_TYPE\_STR (C++ enumerator), 904  
 nvs\_type\_t (C++ enum), 904  
 NVS\_TYPE\_U16 (C++ enumerator), 904  
 NVS\_TYPE\_U32 (C++ enumerator), 904  
 NVS\_TYPE\_U64 (C++ enumerator), 904  
 NVS\_TYPE\_U8 (C++ enumerator), 904

## O

OTA\_SIZE\_UNKNOWN (C 宏), 1121

## P

- PCNT\_CHANNEL\_0 (C++ enumerator), 656  
 PCNT\_CHANNEL\_1 (C++ enumerator), 656  
 PCNT\_CHANNEL\_MAX (C++ enumerator), 656  
 pcnt\_channel\_t (C++ enum), 656  
 pcnt\_config\_t (C++ class), 654  
 pcnt\_config\_t::channel (C++ member), 655  
 pcnt\_config\_t::counter\_h\_lim (C++ member), 655  
 pcnt\_config\_t::counter\_l\_lim (C++ member), 655  
 pcnt\_config\_t::ctrl\_gpio\_num (C++ member), 655  
 pcnt\_config\_t::hctrl\_mode (C++ member), 655  
 pcnt\_config\_t::lctrl\_mode (C++ member), 655  
 pcnt\_config\_t::neg\_mode (C++ member), 655  
 pcnt\_config\_t::pos\_mode (C++ member), 655  
 pcnt\_config\_t::pulse\_gpio\_num (C++ member), 655  
 pcnt\_config\_t::unit (C++ member), 655  
 PCNT\_COUNT\_DEC (C++ enumerator), 656  
 PCNT\_COUNT\_DIS (C++ enumerator), 656  
 PCNT\_COUNT\_INC (C++ enumerator), 656  
 PCNT\_COUNT\_MAX (C++ enumerator), 656  
 pcnt\_count\_mode\_t (C++ enum), 655  
 pcnt\_counter\_clear (C++ function), 651  
 pcnt\_counter\_pause (C++ function), 651  
 pcnt\_counter\_resume (C++ function), 651  
 pcnt\_ctrl\_mode\_t (C++ enum), 655  
 pcnt\_event\_disable (C++ function), 652  
 pcnt\_event\_enable (C++ function), 652  
 PCNT\_EVT\_H\_LIM (C++ enumerator), 656  
 PCNT\_EVT\_L\_LIM (C++ enumerator), 656  
 PCNT\_EVT\_MAX (C++ enumerator), 657  
 PCNT\_EVT\_THRES\_0 (C++ enumerator), 656  
 PCNT\_EVT\_THRES\_1 (C++ enumerator), 656  
 pcnt\_evt\_type\_t (C++ enum), 656  
 PCNT\_EVT\_ZERO (C++ enumerator), 657  
 pcnt\_filter\_disable (C++ function), 653  
 pcnt\_filter\_enable (C++ function), 653  
 pcnt\_get\_counter\_value (C++ function), 651  
 pcnt\_get\_event\_value (C++ function), 652  
 pcnt\_get\_filter\_value (C++ function), 653  
 pcnt\_intr\_disable (C++ function), 651  
 pcnt\_intr\_enable (C++ function), 651  
 pcnt\_isr\_handle\_t (C++ type), 655  
 pcnt\_isr\_handler\_add (C++ function), 654  
 pcnt\_isr\_handler\_remove (C++ function), 654  
 pcnt\_isr\_register (C++ function), 652  
 pcnt\_isr\_service\_install (C++ function), 654  
 pcnt\_isr\_service\_uninstall (C++ function), 654  
 PCNT\_MODE\_DISABLE (C++ enumerator), 655  
 PCNT\_MODE\_KEEP (C++ enumerator), 655  
 PCNT\_MODE\_MAX (C++ enumerator), 655  
 PCNT\_MODE\_REVERSE (C++ enumerator), 655  
 PCNT\_PIN\_NOT\_USED (C 宏), 655  
 pcnt\_set\_event\_value (C++ function), 652  
 pcnt\_set\_filter\_value (C++ function), 653  
 pcnt\_set\_mode (C++ function), 653  
 pcnt\_set\_pin (C++ function), 653  
 PCNT\_UNIT\_0 (C++ enumerator), 656  
 PCNT\_UNIT\_1 (C++ enumerator), 656  
 PCNT\_UNIT\_2 (C++ enumerator), 656  
 PCNT\_UNIT\_3 (C++ enumerator), 656  
 PCNT\_UNIT\_4 (C++ enumerator), 656  
 PCNT\_UNIT\_5 (C++ enumerator), 656  
 PCNT\_UNIT\_6 (C++ enumerator), 656  
 PCNT\_UNIT\_7 (C++ enumerator), 656  
 pcnt\_unit\_config (C++ function), 651  
 PCNT\_UNIT\_MAX (C++ enumerator), 656  
 pcnt\_unit\_t (C++ enum), 656  
 pcQueueGetName (C++ function), 966  
 pcTaskGetTaskName (C++ function), 948  
 pcTimerGetTimerName (C++ function), 999  
 PDM\_PCM\_CONV\_DISABLE (C++ enumerator), 618  
 PDM\_PCM\_CONV\_ENABLE (C++ enumerator), 618  
 pdm\_pcm\_conv\_t (C++ enum), 618  
 PDM\_SAMPLE\_RATE\_RATIO\_128 (C++ enumerator), 618  
 PDM\_SAMPLE\_RATE\_RATIO\_64 (C++ enumerator), 617  
 pdm\_sample\_rate\_ratio\_t (C++ enum), 617  
 PendedFunction\_t (C++ type), 1008  
 protocomm\_add\_endpoint (C++ function), 839  
 protocomm\_ble\_config (C++ class), 845  
 protocomm\_ble\_config::device\_name (C++ member), 845  
 protocomm\_ble\_config::nu\_lookup (C++ member), 845  
 protocomm\_ble\_config::nu\_lookup\_count (C++ member), 845  
 protocomm\_ble\_config::service\_uuid (C++ member), 845  
 protocomm\_ble\_config\_t (C++ type), 845  
 protocomm\_ble\_name\_uuid\_t (C++ type), 845  
 protocomm\_ble\_start (C++ function), 844  
 protocomm\_ble\_stop (C++ function), 844  
 protocomm\_close\_session (C++ function), 839  
 protocomm\_delete (C++ function), 838  
 protocomm\_http\_server\_config\_t (C++ class), 843  
 protocomm\_http\_server\_config\_t::port (C++ member), 843  
 protocomm\_http\_server\_config\_t::stack\_size (C++ member), 843  
 protocomm\_http\_server\_config\_t::task\_priority (C++ member), 843  
 protocomm\_httpd\_config\_data\_t (C++ union), 843

- protocomm\_httpd\_config\_data\_t::config (C++ member), 843  
 protocomm\_httpd\_config\_data\_t::handle (C++ member), 843  
 protocomm\_httpd\_config\_t (C++ class), 844  
 protocomm\_httpd\_config\_t::data (C++ member), 844  
 protocomm\_httpd\_config\_t::ext\_handle\_provided (C++ member), 844  
 PROTOCOMM\_HTTPD\_DEFAULT\_CONFIG (C 宏), 844  
 protocomm\_httpd\_start (C++ function), 843  
 protocomm\_httpd\_stop (C++ function), 843  
 protocomm\_new (C++ function), 838  
 protocomm\_open\_session (C++ function), 839  
 protocomm\_remove\_endpoint (C++ function), 839  
 protocomm\_req\_handle (C++ function), 840  
 protocomm\_req\_handler\_t (C++ type), 841  
 protocomm\_security (C++ class), 842  
 protocomm\_security::cleanup (C++ member), 842  
 protocomm\_security::close\_transport\_session (C++ member), 842  
 protocomm\_security::decrypt (C++ member), 842  
 protocomm\_security::encrypt (C++ member), 842  
 protocomm\_security::init (C++ member), 842  
 protocomm\_security::new\_transport\_session (C++ member), 842  
 protocomm\_security::security\_req\_handler (C++ member), 842  
 protocomm\_security::ver (C++ member), 842  
 protocomm\_security\_handle\_t (C++ type), 842  
 protocomm\_security\_pop (C++ class), 841  
 protocomm\_security\_pop::data (C++ member), 841  
 protocomm\_security\_pop::len (C++ member), 841  
 protocomm\_security\_pop\_t (C++ type), 842  
 protocomm\_security\_t (C++ type), 842  
 protocomm\_set\_security (C++ function), 840  
 protocomm\_set\_version (C++ function), 841  
 protocomm\_t (C++ type), 841  
 protocomm\_unset\_security (C++ function), 840  
 protocomm\_unset\_version (C++ function), 841  
 PROV\_DATA\_FLAGS\_FLAG (C 宏), 285  
 PROV\_DATA\_IV\_INDEX\_FLAG (C 宏), 285  
 PROV\_DATA\_NET\_IDX\_FLAG (C 宏), 285  
 PROXY\_FILTER\_BLACKLIST (C++ enumerator), 299  
 PROXY\_FILTER\_WHITELIST (C++ enumerator), 299  
 PTHREAD\_STACK\_MIN (C 宏), 1126  
 pvTaskGetThreadLocalStoragePointer (C++ function), 949  
 pvTimerGetTimerID (C++ function), 996  
 pxTaskGetStackStart (C++ function), 948
- ## Q
- QueueHandle\_t (C++ type), 979  
 QueueSetHandle\_t (C++ type), 979  
 QueueSetMemberHandle\_t (C++ type), 979
- ## R
- R0 (C 宏), 1462  
 R1 (C 宏), 1462  
 R2 (C 宏), 1462  
 R3 (C 宏), 1462  
 RINGBUF\_TYPE\_ALLOWSPLIT (C++ enumerator), 1030  
 RINGBUF\_TYPE\_BYTEBUF (C++ enumerator), 1030  
 RINGBUF\_TYPE\_MAX (C++ enumerator), 1031  
 RINGBUF\_TYPE\_NOSPLIT (C++ enumerator), 1030  
 RingbufferType\_t (C++ enum), 1030  
 RingbufHandle\_t (C++ type), 1030  
 RMT\_BASECLK\_APB (C++ enumerator), 671  
 RMT\_BASECLK\_MAX (C++ enumerator), 671  
 RMT\_BASECLK\_REF (C++ enumerator), 671  
 RMT\_CARRIER\_LEVEL\_HIGH (C++ enumerator), 672  
 RMT\_CARRIER\_LEVEL\_LOW (C++ enumerator), 672  
 RMT\_CARRIER\_LEVEL\_MAX (C++ enumerator), 672  
 rmt\_carrier\_level\_t (C++ enum), 672  
 RMT\_CHANNEL\_0 (C++ enumerator), 671  
 RMT\_CHANNEL\_1 (C++ enumerator), 671  
 RMT\_CHANNEL\_2 (C++ enumerator), 671  
 RMT\_CHANNEL\_3 (C++ enumerator), 671  
 RMT\_CHANNEL\_4 (C++ enumerator), 671  
 RMT\_CHANNEL\_5 (C++ enumerator), 671  
 RMT\_CHANNEL\_6 (C++ enumerator), 671  
 RMT\_CHANNEL\_7 (C++ enumerator), 671  
 RMT\_CHANNEL\_BUSY (C++ enumerator), 672  
 RMT\_CHANNEL\_IDLE (C++ enumerator), 672  
 RMT\_CHANNEL\_MAX (C++ enumerator), 671  
 rmt\_channel\_status\_result\_t (C++ class), 669  
 rmt\_channel\_status\_result\_t::status (C++ member), 669  
 rmt\_channel\_status\_t (C++ enum), 672  
 rmt\_channel\_t (C++ enum), 671  
 RMT\_CHANNEL\_UNINIT (C++ enumerator), 672  
 rmt\_clr\_intr\_enable\_mask (C++ function), 665  
 rmt\_config (C++ function), 666  
 rmt\_config\_t (C++ class), 669  
 rmt\_config\_t::channel (C++ member), 670  
 rmt\_config\_t::clk\_div (C++ member), 670  
 rmt\_config\_t::gpio\_num (C++ member), 670  
 rmt\_config\_t::mem\_block\_num (C++ member), 670  
 rmt\_config\_t::rmt\_mode (C++ member), 670

- `rmt_config_t::rx_config` (C++ member), 670  
`rmt_config_t::tx_config` (C++ member), 670  
`RMT_DATA_MODE_FIFO` (C++ enumerator), 671  
`RMT_DATA_MODE_MAX` (C++ enumerator), 671  
`RMT_DATA_MODE_MEM` (C++ enumerator), 671  
`rmt_data_mode_t` (C++ enum), 671  
`rmt_driver_install` (C++ function), 667  
`rmt_driver_uninstall` (C++ function), 667  
`rmt_fill_tx_items` (C++ function), 667  
`rmt_get_channel_status` (C++ function), 667  
`rmt_get_clk_div` (C++ function), 661  
`rmt_get_idle_level` (C++ function), 665  
`rmt_get_mem_block_num` (C++ function), 662  
`rmt_get_mem_pd` (C++ function), 662  
`rmt_get_memory_owner` (C++ function), 663  
`rmt_get_ringbuf_handle` (C++ function), 668  
`rmt_get_rx_idle_thresh` (C++ function), 661  
`rmt_get_source_clk` (C++ function), 664  
`rmt_get_status` (C++ function), 665  
`rmt_get_tx_loop_mode` (C++ function), 664  
`RMT_IDLE_LEVEL_HIGH` (C++ enumerator), 672  
`RMT_IDLE_LEVEL_LOW` (C++ enumerator), 672  
`RMT_IDLE_LEVEL_MAX` (C++ enumerator), 672  
`rmt_idle_level_t` (C++ enum), 671  
`rmt_isr_deregister` (C++ function), 666  
`rmt_isr_handle_t` (C++ type), 670  
`rmt_isr_register` (C++ function), 666  
`RMT_MEM_BLOCK_BYTE_NUM` (C 宏), 670  
`RMT_MEM_ITEM_NUM` (C 宏), 670  
`RMT_MEM_OWNER_MAX` (C++ enumerator), 671  
`RMT_MEM_OWNER_RX` (C++ enumerator), 671  
`rmt_mem_owner_t` (C++ enum), 671  
`RMT_MEM_OWNER_TX` (C++ enumerator), 671  
`rmt_memory_rw_rst` (C++ function), 663  
`RMT_MODE_MAX` (C++ enumerator), 671  
`RMT_MODE_RX` (C++ enumerator), 671  
`rmt_mode_t` (C++ enum), 671  
`RMT_MODE_TX` (C++ enumerator), 671  
`rmt_register_tx_end_callback` (C++ function), 668  
`rmt_rx_config_t` (C++ class), 669  
`rmt_rx_config_t::filter_en` (C++ member), 669  
`rmt_rx_config_t::filter_ticks_thresh` (C++ member), 669  
`rmt_rx_config_t::idle_threshold` (C++ member), 669  
`rmt_rx_start` (C++ function), 663  
`rmt_rx_stop` (C++ function), 663  
`rmt_set_clk_div` (C++ function), 661  
`rmt_set_err_intr_en` (C++ function), 665  
`rmt_set_idle_level` (C++ function), 665  
`rmt_set_intr_enable_mask` (C++ function), 665  
`rmt_set_mem_block_num` (C++ function), 661  
`rmt_set_mem_pd` (C++ function), 662  
`rmt_set_memory_owner` (C++ function), 663  
`rmt_set_pin` (C++ function), 666  
`rmt_set_rx_filter` (C++ function), 664  
`rmt_set_rx_idle_thresh` (C++ function), 661  
`rmt_set_rx_intr_en` (C++ function), 665  
`rmt_set_source_clk` (C++ function), 664  
`rmt_set_tx_carrier` (C++ function), 662  
`rmt_set_tx_intr_en` (C++ function), 665  
`rmt_set_tx_loop_mode` (C++ function), 664  
`rmt_set_tx_thr_intr_en` (C++ function), 666  
`rmt_source_clk_t` (C++ enum), 671  
`rmt_translator_init` (C++ function), 668  
`rmt_tx_config_t` (C++ class), 669  
`rmt_tx_config_t::carrier_duty_percent` (C++ member), 669  
`rmt_tx_config_t::carrier_en` (C++ member), 669  
`rmt_tx_config_t::carrier_freq_hz` (C++ member), 669  
`rmt_tx_config_t::carrier_level` (C++ member), 669  
`rmt_tx_config_t::idle_level` (C++ member), 669  
`rmt_tx_config_t::idle_output_en` (C++ member), 669  
`rmt_tx_config_t::loop_en` (C++ member), 669  
`rmt_tx_end_callback_t` (C++ class), 670  
`rmt_tx_end_callback_t::arg` (C++ member), 670  
`rmt_tx_end_callback_t::function` (C++ member), 670  
`rmt_tx_end_fn_t` (C++ type), 670  
`rmt_tx_start` (C++ function), 662  
`rmt_tx_stop` (C++ function), 663  
`rmt_wait_tx_done` (C++ function), 668  
`rmt_write_items` (C++ function), 667  
`rmt_write_sample` (C++ function), 668  
`ROLE_FAST_PROV` (C++ enumerator), 299  
`ROLE_NODE` (C++ enumerator), 299  
`ROLE_PROVISIONER` (C++ enumerator), 299  
`rtc_gpio_deinit` (C++ function), 595  
`rtc_gpio_force_hold_dis_all` (C++ function), 597  
`rtc_gpio_get_drive_capability` (C++ function), 597  
`rtc_gpio_get_level` (C++ function), 595  
`rtc_gpio_hold_dis` (C++ function), 596  
`rtc_gpio_hold_en` (C++ function), 596  
`rtc_gpio_init` (C++ function), 595  
`RTC_GPIO_IS_VALID_GPIO` (C 宏), 598  
`rtc_gpio_is_valid_gpio` (C++ function), 595  
`rtc_gpio_isolate` (C++ function), 597  
`RTC_GPIO_MODE_DISABLED` (C++ enumerator), 598  
`RTC_GPIO_MODE_INPUT_ONLY` (C++ enumerator), 598  
`RTC_GPIO_MODE_INPUT_OUTPUT` (C++ enumerator), 598

- RTC\_GPIO\_MODE\_OUTPUT\_ONLY (C++ *enumerator*), 598
- rtc\_gpio\_mode\_t (C++ *enum*), 598
- rtc\_gpio\_pulldown\_dis (C++ *function*), 596
- rtc\_gpio\_pulldown\_en (C++ *function*), 596
- rtc\_gpio\_pullup\_dis (C++ *function*), 596
- rtc\_gpio\_pullup\_en (C++ *function*), 595
- rtc\_gpio\_set\_direction (C++ *function*), 595
- rtc\_gpio\_set\_drive\_capability (C++ *function*), 597
- rtc\_gpio\_set\_level (C++ *function*), 595
- rtc\_gpio\_wakeup\_disable (C++ *function*), 597
- rtc\_gpio\_wakeup\_enable (C++ *function*), 597
- RTC\_SLOW\_MEM (C 宏), 1465
- ## S
- sample\_to\_rmt\_t (C++ *type*), 670
- SC\_EVENT\_FOUND\_CHANNEL (C++ *enumerator*), 497
- SC\_EVENT\_GOT\_SSID\_PSWD (C++ *enumerator*), 497
- SC\_EVENT\_SCAN\_DONE (C++ *enumerator*), 497
- SC\_EVENT\_SEND\_ACK\_DONE (C++ *enumerator*), 497
- SC\_TYPE\_AIRKISS (C++ *enumerator*), 497
- SC\_TYPE\_ESPTOUCH (C++ *enumerator*), 497
- SC\_TYPE\_ESPTOUCH\_AIRKISS (C++ *enumerator*), 497
- sdio\_event\_cb\_t (C++ *type*), 690
- sdio\_slave\_buf\_handle\_t (C++ *type*), 690
- sdio\_slave\_clear\_host\_int (C++ *function*), 689
- sdio\_slave\_config\_t (C++ *class*), 689
- sdio\_slave\_config\_t::event\_cb (C++ *member*), 689
- sdio\_slave\_config\_t::flags (C++ *member*), 689
- sdio\_slave\_config\_t::recv\_buffer\_size (C++ *member*), 689
- sdio\_slave\_config\_t::send\_queue\_size (C++ *member*), 689
- sdio\_slave\_config\_t::sending\_mode (C++ *member*), 689
- sdio\_slave\_config\_t::timing (C++ *member*), 689
- sdio\_slave\_deinit (C++ *function*), 686
- SDIO\_SLAVE\_FLAG\_DAT2\_DISABLED (C 宏), 689
- SDIO\_SLAVE\_FLAG\_HOST\_INTR\_DISABLED (C 宏), 689
- SDIO\_SLAVE\_FLAG\_INTERNAL\_PULLUP (C 宏), 690
- sdio\_slave\_get\_host\_intena (C++ *function*), 688
- SDIO\_SLAVE\_HOSTINT\_BIT0 (C++ *enumerator*), 690
- SDIO\_SLAVE\_HOSTINT\_BIT1 (C++ *enumerator*), 690
- SDIO\_SLAVE\_HOSTINT\_BIT2 (C++ *enumerator*), 690
- SDIO\_SLAVE\_HOSTINT\_BIT3 (C++ *enumerator*), 690
- SDIO\_SLAVE\_HOSTINT\_BIT4 (C++ *enumerator*), 690
- SDIO\_SLAVE\_HOSTINT\_BIT5 (C++ *enumerator*), 690
- SDIO\_SLAVE\_HOSTINT\_BIT6 (C++ *enumerator*), 690
- SDIO\_SLAVE\_HOSTINT\_BIT7 (C++ *enumerator*), 690
- SDIO\_SLAVE\_HOSTINT\_RECV\_OVF (C++ *enumerator*), 690
- SDIO\_SLAVE\_HOSTINT\_SEND\_NEW\_PACKET (C++ *enumerator*), 690
- SDIO\_SLAVE\_HOSTINT\_SEND\_UDF (C++ *enumerator*), 690
- sdio\_slave\_hostint\_t (C++ *enum*), 690
- sdio\_slave\_initialize (C++ *function*), 686
- sdio\_slave\_read\_reg (C++ *function*), 688
- sdio\_slave\_recv (C++ *function*), 687
- sdio\_slave\_recv\_get\_buf (C++ *function*), 687
- sdio\_slave\_recv\_load\_buf (C++ *function*), 687
- SDIO\_SLAVE\_RECV\_MAX\_BUFFER (C 宏), 689
- sdio\_slave\_recv\_register\_buf (C++ *function*), 687
- sdio\_slave\_recv\_unregister\_buf (C++ *function*), 687
- sdio\_slave\_reset (C++ *function*), 687
- sdio\_slave\_send\_get\_finished (C++ *function*), 688
- sdio\_slave\_send\_host\_int (C++ *function*), 688
- SDIO\_SLAVE\_SEND\_PACKET (C++ *enumerator*), 691
- sdio\_slave\_send\_queue (C++ *function*), 687
- SDIO\_SLAVE\_SEND\_STREAM (C++ *enumerator*), 691
- sdio\_slave\_sending\_mode\_t (C++ *enum*), 691
- sdio\_slave\_set\_host\_intena (C++ *function*), 688
- sdio\_slave\_start (C++ *function*), 686
- sdio\_slave\_stop (C++ *function*), 686
- SDIO\_SLAVE\_TIMING\_NSEND\_NSAMPLE (C++ *enumerator*), 690
- SDIO\_SLAVE\_TIMING\_NSEND\_PSAMPLE (C++ *enumerator*), 690
- SDIO\_SLAVE\_TIMING\_PSEND\_NSAMPLE (C++ *enumerator*), 690
- SDIO\_SLAVE\_TIMING\_PSEND\_PSAMPLE (C++ *enumerator*), 690
- sdio\_slave\_timing\_t (C++ *enum*), 690
- sdio\_slave\_transmit (C++ *function*), 688
- sdio\_slave\_wait\_int (C++ *function*), 689
- sdio\_slave\_write\_reg (C++ *function*), 688
- sdmmc\_card\_init (C++ *function*), 880

- sdmmc\_card\_print\_info (C++ function), 881
- sdmmc\_card\_t (C++ class), 886
- sdmmc\_card\_t::cid (C++ member), 886
- sdmmc\_card\_t::csd (C++ member), 886
- sdmmc\_card\_t::ext\_csd (C++ member), 887
- sdmmc\_card\_t::host (C++ member), 886
- sdmmc\_card\_t::is\_ddr (C++ member), 887
- sdmmc\_card\_t::is\_mem (C++ member), 887
- sdmmc\_card\_t::is\_mmc (C++ member), 887
- sdmmc\_card\_t::is\_sdio (C++ member), 887
- sdmmc\_card\_t::log\_bus\_width (C++ member), 887
- sdmmc\_card\_t::max\_freq\_khz (C++ member), 887
- sdmmc\_card\_t::num\_io\_functions (C++ member), 887
- sdmmc\_card\_t::ocr (C++ member), 886
- sdmmc\_card\_t::raw\_cid (C++ member), 886
- sdmmc\_card\_t::rca (C++ member), 887
- sdmmc\_card\_t::reserved (C++ member), 887
- sdmmc\_card\_t::scr (C++ member), 887
- sdmmc\_cid\_t (C++ class), 884
- sdmmc\_cid\_t::date (C++ member), 884
- sdmmc\_cid\_t::mfg\_id (C++ member), 884
- sdmmc\_cid\_t::name (C++ member), 884
- sdmmc\_cid\_t::oem\_id (C++ member), 884
- sdmmc\_cid\_t::revision (C++ member), 884
- sdmmc\_cid\_t::serial (C++ member), 884
- sdmmc\_command\_t (C++ class), 885
- sdmmc\_command\_t::arg (C++ member), 885
- sdmmc\_command\_t::blklen (C++ member), 885
- sdmmc\_command\_t::data (C++ member), 885
- sdmmc\_command\_t::datalen (C++ member), 885
- sdmmc\_command\_t::error (C++ member), 885
- sdmmc\_command\_t::flags (C++ member), 885
- sdmmc\_command\_t::opcode (C++ member), 885
- sdmmc\_command\_t::response (C++ member), 885
- sdmmc\_command\_t::timeout\_ms (C++ member), 885
- sdmmc\_csd\_t (C++ class), 884
- sdmmc\_csd\_t::capacity (C++ member), 884
- sdmmc\_csd\_t::card\_command\_class (C++ member), 884
- sdmmc\_csd\_t::csd\_ver (C++ member), 884
- sdmmc\_csd\_t::mmc\_ver (C++ member), 884
- sdmmc\_csd\_t::read\_block\_len (C++ member), 884
- sdmmc\_csd\_t::sector\_size (C++ member), 884
- sdmmc\_csd\_t::tr\_speed (C++ member), 884
- sdmmc\_ext\_csd\_t (C++ class), 885
- sdmmc\_ext\_csd\_t::power\_class (C++ member), 885
- SDMMC\_FREQ\_26M (C 宏), 887
- SDMMC\_FREQ\_52M (C 宏), 887
- SDMMC\_FREQ\_DEFAULT (C 宏), 887
- SDMMC\_FREQ\_HIGHSPEED (C 宏), 887
- SDMMC\_FREQ\_PROBING (C 宏), 887
- SDMMC\_HOST\_DEFAULT (C 宏), 676
- sdmmc\_host\_deinit (C++ function), 675
- sdmmc\_host\_do\_transaction (C++ function), 675
- SDMMC\_HOST\_FLAG\_1BIT (C 宏), 887
- SDMMC\_HOST\_FLAG\_4BIT (C 宏), 887
- SDMMC\_HOST\_FLAG\_8BIT (C 宏), 887
- SDMMC\_HOST\_FLAG\_DDR (C 宏), 887
- SDMMC\_HOST\_FLAG\_SPI (C 宏), 887
- sdmmc\_host\_get\_slot\_width (C++ function), 674
- sdmmc\_host\_init (C++ function), 674
- sdmmc\_host\_init\_slot (C++ function), 674
- sdmmc\_host\_io\_int\_enable (C++ function), 675
- sdmmc\_host\_io\_int\_wait (C++ function), 675
- sdmmc\_host\_pullup\_en (C++ function), 675
- sdmmc\_host\_set\_bus\_ddr\_mode (C++ function), 675
- sdmmc\_host\_set\_bus\_width (C++ function), 674
- sdmmc\_host\_set\_card\_clk (C++ function), 674
- SDMMC\_HOST\_SLOT\_0 (C 宏), 676
- SDMMC\_HOST\_SLOT\_1 (C 宏), 676
- sdmmc\_host\_t (C++ class), 885
- sdmmc\_host\_t::command\_timeout\_ms (C++ member), 886
- sdmmc\_host\_t::deinit (C++ member), 886
- sdmmc\_host\_t::do\_transaction (C++ member), 886
- sdmmc\_host\_t::flags (C++ member), 886
- sdmmc\_host\_t::get\_bus\_width (C++ member), 886
- sdmmc\_host\_t::init (C++ member), 886
- sdmmc\_host\_t::io\_int\_enable (C++ member), 886
- sdmmc\_host\_t::io\_int\_wait (C++ member), 886
- sdmmc\_host\_t::io\_voltage (C++ member), 886
- sdmmc\_host\_t::max\_freq\_khz (C++ member), 886
- sdmmc\_host\_t::set\_bus\_ddr\_mode (C++ member), 886
- sdmmc\_host\_t::set\_bus\_width (C++ member), 886
- sdmmc\_host\_t::set\_card\_clk (C++ member), 886
- sdmmc\_host\_t::slot (C++ member), 886
- sdmmc\_io\_enable\_int (C++ function), 883
- sdmmc\_io\_get\_cis\_data (C++ function), 883
- sdmmc\_io\_print\_cis\_info (C++ function), 883
- sdmmc\_io\_read\_blocks (C++ function), 882
- sdmmc\_io\_read\_byte (C++ function), 881
- sdmmc\_io\_read\_bytes (C++ function), 882
- sdmmc\_io\_wait\_int (C++ function), 883

- sdmmc\_io\_write\_blocks (C++ function), 882
- sdmmc\_io\_write\_byte (C++ function), 881
- sdmmc\_io\_write\_bytes (C++ function), 882
- sdmmc\_read\_sectors (C++ function), 881
- sdmmc\_response\_t (C++ type), 888
- sdmmc\_scr\_t (C++ class), 885
- sdmmc\_scr\_t::bus\_width (C++ member), 885
- sdmmc\_scr\_t::sd\_spec (C++ member), 885
- SDMMC\_SLOT\_CONFIG\_DEFAULT (C 宏), 676
- sdmmc\_slot\_config\_t (C++ class), 676
- sdmmc\_slot\_config\_t::flags (C++ member), 676
- sdmmc\_slot\_config\_t::gpio\_cd (C++ member), 676
- sdmmc\_slot\_config\_t::gpio\_wp (C++ member), 676
- sdmmc\_slot\_config\_t::width (C++ member), 676
- SDMMC\_SLOT\_FLAG\_INTERNAL\_PULLUP (C 宏), 676
- SDMMC\_SLOT\_NO\_CD (C 宏), 676
- SDMMC\_SLOT\_NO\_WP (C 宏), 676
- SDMMC\_SLOT\_WIDTH\_DEFAULT (C 宏), 676
- sdmmc\_switch\_func\_rsp\_t (C++ class), 885
- sdmmc\_switch\_func\_rsp\_t::data (C++ member), 885
- sdmmc\_write\_sectors (C++ function), 881
- SDSPI\_HOST\_DEFAULT (C 宏), 679
- sdspi\_host\_deinit (C++ function), 678
- sdspi\_host\_do\_transaction (C++ function), 677
- sdspi\_host\_init (C++ function), 677
- sdspi\_host\_init\_slot (C++ function), 677
- sdspi\_host\_io\_int\_enable (C++ function), 678
- sdspi\_host\_io\_int\_wait (C++ function), 678
- sdspi\_host\_set\_card\_clk (C++ function), 678
- SDSPI\_SLOT\_CONFIG\_DEFAULT (C 宏), 679
- sdspi\_slot\_config\_t (C++ class), 678
- sdspi\_slot\_config\_t::dma\_channel (C++ member), 679
- sdspi\_slot\_config\_t::gpio\_cd (C++ member), 678
- sdspi\_slot\_config\_t::gpio\_cs (C++ member), 678
- sdspi\_slot\_config\_t::gpio\_int (C++ member), 679
- sdspi\_slot\_config\_t::gpio\_miso (C++ member), 678
- sdspi\_slot\_config\_t::gpio\_mosi (C++ member), 678
- sdspi\_slot\_config\_t::gpio\_sck (C++ member), 678
- sdspi\_slot\_config\_t::gpio\_wp (C++ member), 679
- SDSPI\_SLOT\_NO\_CD (C 宏), 679
- SDSPI\_SLOT\_NO\_INT (C 宏), 679
- SDSPI\_SLOT\_NO\_WP (C 宏), 679
- SemaphoreHandle\_t (C++ type), 992
- semBINARY\_SEMAPHORE\_QUEUE\_LENGTH (C 宏), 980
- semGIVE\_BLOCK\_TIME (C 宏), 980
- semSEMAPHORE\_QUEUE\_ITEM\_LENGTH (C 宏), 980
- shutdown\_handler\_t (C++ type), 1138
- SIGMADELTA\_CHANNEL\_0 (C++ enumerator), 692
- SIGMADELTA\_CHANNEL\_1 (C++ enumerator), 693
- SIGMADELTA\_CHANNEL\_2 (C++ enumerator), 693
- SIGMADELTA\_CHANNEL\_3 (C++ enumerator), 693
- SIGMADELTA\_CHANNEL\_4 (C++ enumerator), 693
- SIGMADELTA\_CHANNEL\_5 (C++ enumerator), 693
- SIGMADELTA\_CHANNEL\_6 (C++ enumerator), 693
- SIGMADELTA\_CHANNEL\_7 (C++ enumerator), 693
- SIGMADELTA\_CHANNEL\_MAX (C++ enumerator), 693
- sigmadelta\_channel\_t (C++ enum), 692
- sigmadelta\_config (C++ function), 691
- sigmadelta\_config\_t (C++ class), 692
- sigmadelta\_config\_t::channel (C++ member), 692
- sigmadelta\_config\_t::sigmadelta\_duty (C++ member), 692
- sigmadelta\_config\_t::sigmadelta\_gpio (C++ member), 692
- sigmadelta\_config\_t::sigmadelta\_prescale (C++ member), 692
- sigmadelta\_set\_duty (C++ function), 692
- sigmadelta\_set\_pin (C++ function), 692
- sigmadelta\_set\_prescale (C++ function), 692
- slave\_transaction\_cb\_t (C++ type), 715
- smartconfig\_event\_got\_ssid\_pswd\_t (C++ class), 496
- smartconfig\_event\_got\_ssid\_pswd\_t::bssid (C++ member), 496
- smartconfig\_event\_got\_ssid\_pswd\_t::bssid\_set (C++ member), 496
- smartconfig\_event\_got\_ssid\_pswd\_t::cellphone\_ip (C++ member), 496
- smartconfig\_event\_got\_ssid\_pswd\_t::password (C++ member), 496
- smartconfig\_event\_got\_ssid\_pswd\_t::ssid (C++ member), 496
- smartconfig\_event\_got\_ssid\_pswd\_t::token (C++ member), 496
- smartconfig\_event\_got\_ssid\_pswd\_t::type (C++ member), 496
- smartconfig\_event\_t (C++ enum), 497
- SMARTCONFIG\_START\_CONFIG\_DEFAULT (C 宏), 496
- smartconfig\_start\_config\_t (C++ class), 496
- smartconfig\_start\_config\_t::enable\_log (C++ member), 496
- smartconfig\_type\_t (C++ enum), 497
- soc\_has\_cache\_lock\_bug (C++ function), 1072
- SPI1\_HOST (C++ enumerator), 701



- SPI2\_HOST (C++ enumerator), 701  
 SPI3\_HOST (C++ enumerator), 701  
 spi\_bus\_add\_device (C++ function), 703  
 spi\_bus\_add\_flash\_device (C++ function), 864  
 spi\_bus\_config\_t (C++ class), 702  
 spi\_bus\_config\_t::flags (C++ member), 702  
 spi\_bus\_config\_t::intr\_flags (C++ member), 702  
 spi\_bus\_config\_t::max\_transfer\_sz (C++ member), 702  
 spi\_bus\_config\_t::miso\_io\_num (C++ member), 702  
 spi\_bus\_config\_t::mosi\_io\_num (C++ member), 702  
 spi\_bus\_config\_t::quadhd\_io\_num (C++ member), 702  
 spi\_bus\_config\_t::quadwp\_io\_num (C++ member), 702  
 spi\_bus\_config\_t::sclk\_io\_num (C++ member), 702  
 spi\_bus\_free (C++ function), 701  
 spi\_bus\_initialize (C++ function), 701  
 spi\_bus\_remove\_device (C++ function), 704  
 spi\_bus\_remove\_flash\_device (C++ function), 865  
 spi\_cal\_clock (C++ function), 706  
 SPI\_DEVICE\_3WIRE (C 宏), 709  
 spi\_device\_acquire\_bus (C++ function), 705  
 SPI\_DEVICE\_BIT\_LSBFIRST (C 宏), 709  
 SPI\_DEVICE\_CLK\_AS\_CS (C 宏), 709  
 spi\_device\_get\_trans\_result (C++ function), 704  
 SPI\_DEVICE\_HALFDUPLEX (C 宏), 709  
 spi\_device\_handle\_t (C++ type), 710  
 spi\_device\_interface\_config\_t (C++ class), 706  
 spi\_device\_interface\_config\_t::address\_bits (C++ member), 707  
 spi\_device\_interface\_config\_t::clock\_speed\_hz (C++ member), 707  
 spi\_device\_interface\_config\_t::command\_bits (C++ member), 706  
 spi\_device\_interface\_config\_t::cs\_ena\_pos (C++ member), 707  
 spi\_device\_interface\_config\_t::cs\_ena\_pos\_inv (C++ member), 707  
 spi\_device\_interface\_config\_t::dummy\_bits (C++ member), 707  
 spi\_device\_interface\_config\_t::duty\_cycle\_pos (C++ member), 707  
 spi\_device\_interface\_config\_t::flags (C++ member), 707  
 spi\_device\_interface\_config\_t::input\_delay\_ns (C++ member), 707  
 spi\_device\_interface\_config\_t::mode (C++ member), 707  
 spi\_device\_interface\_config\_t::post\_cb (C++ member), 707  
 spi\_device\_interface\_config\_t::pre\_cb (C++ member), 707  
 spi\_device\_interface\_config\_t::queue\_size (C++ member), 707  
 spi\_device\_interface\_config\_t::spics\_io\_num (C++ member), 707  
 SPI\_DEVICE\_NO\_DUMMY (C 宏), 709  
 spi\_device\_polling\_end (C++ function), 705  
 spi\_device\_polling\_start (C++ function), 705  
 spi\_device\_polling\_transmit (C++ function), 705  
 SPI\_DEVICE\_POSITIVE\_CS (C 宏), 709  
 spi\_device\_queue\_trans (C++ function), 704  
 spi\_device\_release\_bus (C++ function), 706  
 SPI\_DEVICE\_RXBIT\_LSBFIRST (C 宏), 709  
 spi\_device\_transmit (C++ function), 704  
 SPI\_DEVICE\_TXBIT\_LSBFIRST (C 宏), 709  
 spi\_flash\_chip\_t (C++ type), 869  
 SPI\_FLASH\_DIO (C++ enumerator), 872  
 SPI\_FLASH\_DOUT (C++ enumerator), 872  
 SPI\_FLASH\_FASTRD (C++ enumerator), 872  
 spi\_flash\_host\_driver\_t (C++ class), 870  
 spi\_flash\_host\_driver\_t (C++ type), 871  
 spi\_flash\_host\_driver\_t::common\_command (C++ member), 870  
 spi\_flash\_host\_driver\_t::configure\_host\_io\_mode (C++ member), 871  
 spi\_flash\_host\_driver\_t::dev\_config (C++ member), 870  
 spi\_flash\_host\_driver\_t::driver\_data (C++ member), 870  
 spi\_flash\_host\_driver\_t::erase\_block (C++ member), 870  
 spi\_flash\_host\_driver\_t::erase\_chip (C++ member), 870  
 spi\_flash\_host\_driver\_t::erase\_sector (C++ member), 870  
 spi\_flash\_host\_driver\_t::flush\_cache (C++ member), 871  
 spi\_flash\_host\_driver\_t::host\_idle (C++ member), 871  
 spi\_flash\_host\_driver\_t::max\_read\_bytes (C++ member), 871  
 spi\_flash\_host\_driver\_t::max\_write\_bytes (C++ member), 871  
 spi\_flash\_host\_driver\_t::poll\_cmd\_done (C++ member), 871  
 spi\_flash\_host\_driver\_t::program\_page (C++ member), 870  
 spi\_flash\_host\_driver\_t::read (C++ member), 871  
 spi\_flash\_host\_driver\_t::read\_id (C++ member), 870  
 spi\_flash\_host\_driver\_t::read\_status (C++ member), 870  
 spi\_flash\_host\_driver\_t::set\_write\_protect

- (C++ member), 870
- `spi_flash_host_driver_t::supports_direct_access` (C++ member), 870
- `spi_flash_host_driver_t::supports_direct_access` (C++ member), 870
- `SPI_FLASH_QIO` (C++ enumerator), 872
- `SPI_FLASH_QOUT` (C++ enumerator), 872
- `SPI_FLASH_READ_MODE_MAX` (C++ enumerator), 872
- `SPI_FLASH_READ_MODE_MIN` (C 宏), 871
- `SPI_FLASH_SLOWRD` (C++ enumerator), 872
- `spi_flash_trans_t` (C++ class), 870
- `spi_flash_trans_t::command` (C++ member), 870
- `spi_flash_trans_t::miso_data` (C++ member), 870
- `spi_flash_trans_t::miso_len` (C++ member), 870
- `spi_flash_trans_t::mosi_data` (C++ member), 870
- `spi_flash_trans_t::mosi_len` (C++ member), 870
- `spi_get_actual_clock` (C++ function), 706
- `spi_get_freq_limit` (C++ function), 706
- `spi_get_timing` (C++ function), 706
- `SPI_HOST` (C 宏), 701
- `spi_host_device_t` (C++ enum), 701
- `SPI_MASTER_FREQ_10M` (C 宏), 708
- `SPI_MASTER_FREQ_11M` (C 宏), 709
- `SPI_MASTER_FREQ_13M` (C 宏), 709
- `SPI_MASTER_FREQ_16M` (C 宏), 709
- `SPI_MASTER_FREQ_20M` (C 宏), 709
- `SPI_MASTER_FREQ_26M` (C 宏), 709
- `SPI_MASTER_FREQ_40M` (C 宏), 709
- `SPI_MASTER_FREQ_80M` (C 宏), 709
- `SPI_MASTER_FREQ_8M` (C 宏), 708
- `SPI_MASTER_FREQ_9M` (C 宏), 708
- `SPI_MAX_DMA_LEN` (C 宏), 702
- `SPI_SLAVE_BIT_LSBFIRST` (C 宏), 715
- `spi_slave_free` (C++ function), 713
- `spi_slave_get_trans_result` (C++ function), 714
- `spi_slave_initialize` (C++ function), 713
- `spi_slave_interface_config_t` (C++ class), 714
- `spi_slave_interface_config_t::flags` (C++ member), 714
- `spi_slave_interface_config_t::mode` (C++ member), 714
- `spi_slave_interface_config_t::post_setup_cb` (C++ member), 714
- `spi_slave_interface_config_t::post_trans_cb` (C++ member), 715
- `spi_slave_interface_config_t::queue_size` (C++ member), 714
- `spi_slave_interface_config_t::spics_io` (C++ member), 714
- `spi_slave_queue_trans` (C++ function), 713
- `SPI_SLAVE_RXBIT_LSBFIRST` (C 宏), 715
- `spi_slave_transaction_t` (C++ class), 715
- `spi_slave_transaction_t` (C++ type), 715
- `spi_slave_transaction_t::length` (C++ member), 715
- `spi_slave_transaction_t::rx_buffer` (C++ member), 715
- `spi_slave_transaction_t::trans_len` (C++ member), 715
- `spi_slave_transaction_t::tx_buffer` (C++ member), 715
- `spi_slave_transaction_t::user` (C++ member), 715
- `spi_slave_transmit` (C++ function), 714
- `SPI_SLAVE_TXBIT_LSBFIRST` (C 宏), 715
- `SPI_SWAP_DATA_RX` (C 宏), 702
- `SPI_SWAP_DATA_TX` (C 宏), 702
- `SPI_TRANS_MODE_DIO` (C 宏), 709
- `SPI_TRANS_MODE_DIOQIO_ADDR` (C 宏), 709
- `SPI_TRANS_MODE_QIO` (C 宏), 709
- `SPI_TRANS_USE_RXDATA` (C 宏), 709
- `SPI_TRANS_USE_TXDATA` (C 宏), 709
- `SPI_TRANS_VARIABLE_ADDR` (C 宏), 710
- `SPI_TRANS_VARIABLE_CMD` (C 宏), 710
- `SPI_TRANS_VARIABLE_DUMMY` (C 宏), 710
- `spi_transaction_ext_t` (C++ class), 708
- `spi_transaction_ext_t::address_bits` (C++ member), 708
- `spi_transaction_ext_t::base` (C++ member), 708
- `spi_transaction_ext_t::command_bits` (C++ member), 708
- `spi_transaction_ext_t::dummy_bits` (C++ member), 708
- `spi_transaction_t` (C++ class), 707
- `spi_transaction_t` (C++ type), 710
- `spi_transaction_t::addr` (C++ member), 708
- `spi_transaction_t::cmd` (C++ member), 707
- `spi_transaction_t::flags` (C++ member), 707
- `spi_transaction_t::length` (C++ member), 708
- `spi_transaction_t::rx_buffer` (C++ member), 708
- `spi_transaction_t::rx_data` (C++ member), 708
- `spi_transaction_t::rxlength` (C++ member), 708
- `spi_transaction_t::tx_buffer` (C++ member), 708
- `spi_transaction_t::tx_data` (C++ member), 708
- `spi_transaction_t::user` (C++ member), 708
- `SPICOMMON_BUSFLAG_DUAL` (C 宏), 703
- `SPICOMMON_BUSFLAG_IOMUX_PINS` (C 宏), 703
- `SPICOMMON_BUSFLAG_MASTER` (C 宏), 703
- `SPICOMMON_BUSFLAG_MISO` (C 宏), 703
- `SPICOMMON_BUSFLAG_MOSI` (C 宏), 703

- SPICOMMON\_BUSFLAG\_NATIVE\_PINS (C 宏), 703
- SPICOMMON\_BUSFLAG\_QUAD (C 宏), 703
- SPICOMMON\_BUSFLAG\_SCLK (C 宏), 703
- SPICOMMON\_BUSFLAG\_SLAVE (C 宏), 703
- SPICOMMON\_BUSFLAG\_WPHD (C 宏), 703
- StaticRingbuffer\_t (C++ type), 1030
- system\_event\_ap\_probe\_req\_rx\_t (C++ type), 1093
- SYSTEM\_EVENT\_AP\_PROBEREQRECVED (C++ enumerator), 1094
- SYSTEM\_EVENT\_AP\_STA\_GOT\_IP6 (C 宏), 1093
- SYSTEM\_EVENT\_AP\_STACONNECTED (C++ enumerator), 1094
- system\_event\_ap\_staconnected\_t (C++ type), 1093
- SYSTEM\_EVENT\_AP\_STADISCONNECTED (C++ enumerator), 1094
- system\_event\_ap\_stadisconnected\_t (C++ type), 1093
- SYSTEM\_EVENT\_AP\_STAIPASSIGNED (C++ enumerator), 1094
- system\_event\_ap\_staipassigned\_t (C++ type), 1093
- SYSTEM\_EVENT\_AP\_START (C++ enumerator), 1094
- SYSTEM\_EVENT\_AP\_STOP (C++ enumerator), 1094
- system\_event\_cb\_t (C++ type), 1093
- SYSTEM\_EVENT\_ETH\_CONNECTED (C++ enumerator), 1095
- SYSTEM\_EVENT\_ETH\_DISCONNECTED (C++ enumerator), 1095
- SYSTEM\_EVENT\_ETH\_GOT\_IP (C++ enumerator), 1095
- SYSTEM\_EVENT\_ETH\_START (C++ enumerator), 1095
- SYSTEM\_EVENT\_ETH\_STOP (C++ enumerator), 1095
- SYSTEM\_EVENT\_GOT\_IP6 (C++ enumerator), 1094
- system\_event\_got\_ip6\_t (C++ type), 1093
- system\_event\_handler\_t (C++ type), 1093
- system\_event\_id\_t (C++ enum), 1094
- system\_event\_info\_t (C++ union), 1092
- system\_event\_info\_t::ap\_probereqrecv (C++ member), 1092
- system\_event\_info\_t::ap\_staipassigned (C++ member), 1092
- system\_event\_info\_t::auth\_change (C++ member), 1092
- system\_event\_info\_t::connected (C++ member), 1092
- system\_event\_info\_t::disconnected (C++ member), 1092
- system\_event\_info\_t::got\_ip (C++ member), 1092
- system\_event\_info\_t::got\_ip6 (C++ member), 1092
- system\_event\_info\_t::scan\_done (C++ member), 1092
- system\_event\_info\_t::sta\_connected (C++ member), 1092
- system\_event\_info\_t::sta\_disconnected (C++ member), 1092
- system\_event\_info\_t::sta\_er\_fail\_reason (C++ member), 1092
- system\_event\_info\_t::sta\_er\_pin (C++ member), 1092
- system\_event\_info\_t::sta\_er\_success (C++ member), 1092
- SYSTEM\_EVENT\_MAX (C++ enumerator), 1095
- SYSTEM\_EVENT\_SCAN\_DONE (C++ enumerator), 1094
- SYSTEM\_EVENT\_STA\_AUTHMODE\_CHANGE (C++ enumerator), 1094
- system\_event\_sta\_authmode\_change\_t (C++ type), 1093
- SYSTEM\_EVENT\_STA\_BEACON\_TIMEOUT (C++ enumerator), 1094
- SYSTEM\_EVENT\_STA\_CONNECTED (C++ enumerator), 1094
- system\_event\_sta\_connected\_t (C++ type), 1093
- SYSTEM\_EVENT\_STA\_DISCONNECTED (C++ enumerator), 1094
- system\_event\_sta\_disconnected\_t (C++ type), 1093
- SYSTEM\_EVENT\_STA\_GOT\_IP (C++ enumerator), 1094
- system\_event\_sta\_got\_ip\_t (C++ type), 1093
- SYSTEM\_EVENT\_STA\_LOST\_IP (C++ enumerator), 1094
- system\_event\_sta\_scan\_done\_t (C++ type), 1093
- SYSTEM\_EVENT\_STA\_START (C++ enumerator), 1094
- SYSTEM\_EVENT\_STA\_STOP (C++ enumerator), 1094
- SYSTEM\_EVENT\_STA\_WPS\_ER\_FAILED (C++ enumerator), 1094
- SYSTEM\_EVENT\_STA\_WPS\_ER\_PBC\_OVERLAP (C++ enumerator), 1094
- SYSTEM\_EVENT\_STA\_WPS\_ER\_PIN (C++ enumerator), 1094
- system\_event\_sta\_wps\_er\_pin\_t (C++ type), 1093
- SYSTEM\_EVENT\_STA\_WPS\_ER\_SUCCESS (C++ enumerator), 1094
- system\_event\_sta\_wps\_er\_success\_t (C++ type), 1093
- SYSTEM\_EVENT\_STA\_WPS\_ER\_TIMEOUT (C++ enumerator), 1094
- system\_event\_sta\_wps\_fail\_reason\_t (C++ type), 1093
- system\_event\_t (C++ class), 1092
- system\_event\_t::event\_id (C++ member), 1093
- system\_event\_t::event\_info (C++ member),

- 1093  
SYSTEM\_EVENT\_WIFI\_READY (C++ enumerator),  
1094
- ## T
- taskDISABLE\_INTERRUPTS (C 宏), 958  
taskENABLE\_INTERRUPTS (C 宏), 958  
taskENTER\_CRITICAL (C 宏), 958  
taskENTER\_CRITICAL\_ISR (C 宏), 958  
taskEXIT\_CRITICAL (C 宏), 958  
taskEXIT\_CRITICAL\_ISR (C 宏), 958  
TaskHandle\_t (C++ type), 959  
TaskHookFunction\_t (C++ type), 959  
taskSCHEDULER\_NOT\_STARTED (C 宏), 958  
taskSCHEDULER\_RUNNING (C 宏), 958  
taskSCHEDULER\_SUSPENDED (C 宏), 958  
TaskSnapshot\_t (C++ type), 959  
TaskStatus\_t (C++ type), 959  
taskYIELD (C 宏), 958  
tcpip\_adapter\_ap\_input (C++ function), 551  
tcpip\_adapter\_ap\_start (C++ function), 545  
tcpip\_adapter\_clear\_default\_eth\_handlers  
(C++ function), 552  
tcpip\_adapter\_clear\_default\_wifi\_handlers  
(C++ function), 553  
tcpip\_adapter\_create\_ip6\_linklocal  
(C++ function), 548  
TCPIP\_ADAPTER\_DHCP\_INIT (C++ enumerator),  
555  
tcpip\_adapter\_dhcp\_option\_id\_t (C++  
enum), 555  
tcpip\_adapter\_dhcp\_option\_mode\_t (C++  
enum), 555  
TCPIP\_ADAPTER\_DHCP\_STARTED (C++ enumera-  
tor), 555  
TCPIP\_ADAPTER\_DHCP\_STATUS\_MAX (C++ enu-  
merator), 555  
tcpip\_adapter\_dhcp\_status\_t (C++ enum),  
555  
TCPIP\_ADAPTER\_DHCP\_STOPPED (C++ enumera-  
tor), 555  
tcpip\_adapter\_dhcpc\_get\_status (C++  
function), 549  
tcpip\_adapter\_dhcpc\_option (C++ function),  
550  
tcpip\_adapter\_dhcpc\_start (C++ function),  
550  
tcpip\_adapter\_dhcpc\_stop (C++ function),  
550  
tcpip\_adapter\_dhcps\_get\_status (C++  
function), 549  
tcpip\_adapter\_dhcps\_option (C++ function),  
549  
tcpip\_adapter\_dhcps\_start (C++ function),  
549  
tcpip\_adapter\_dhcps\_stop (C++ function),  
549  
TCPIP\_ADAPTER\_DNS\_BACKUP (C++ enumerator),  
554  
TCPIP\_ADAPTER\_DNS\_FALLBACK (C++ enumera-  
tor), 555  
tcpip\_adapter\_dns\_info\_t (C++ class), 553  
tcpip\_adapter\_dns\_info\_t::ip (C++ mem-  
ber), 553  
TCPIP\_ADAPTER\_DNS\_MAIN (C++ enumerator),  
554  
TCPIP\_ADAPTER\_DNS\_MAX (C++ enumerator), 555  
tcpip\_adapter\_dns\_type\_t (C++ enum), 554  
TCPIP\_ADAPTER\_DOMAIN\_NAME\_SERVER (C++  
enumerator), 555  
tcpip\_adapter\_down (C++ function), 546  
tcpip\_adapter\_eth\_input (C++ function), 550  
tcpip\_adapter\_eth\_start (C++ function), 545  
tcpip\_adapter\_get\_dns\_info (C++ function),  
547  
tcpip\_adapter\_get\_esp\_if (C++ function),  
551  
tcpip\_adapter\_get\_hostname (C++ function),  
552  
tcpip\_adapter\_get\_ip6\_global (C++ func-  
tion), 548  
tcpip\_adapter\_get\_ip6\_linklocal (C++  
function), 548  
tcpip\_adapter\_get\_ip\_info (C++ function),  
546  
tcpip\_adapter\_get\_netif (C++ function), 552  
tcpip\_adapter\_get\_netif\_index (C++ func-  
tion), 553  
tcpip\_adapter\_get\_old\_ip\_info (C++ func-  
tion), 548  
tcpip\_adapter\_get\_sta\_list (C++ function),  
551  
TCPIP\_ADAPTER\_IF\_AP (C++ enumerator), 554  
TCPIP\_ADAPTER\_IF\_ETH (C++ enumerator), 554  
TCPIP\_ADAPTER\_IF\_MAX (C++ enumerator), 554  
TCPIP\_ADAPTER\_IF\_STA (C++ enumerator), 554  
tcpip\_adapter\_if\_t (C++ enum), 554  
TCPIP\_ADAPTER\_IF\_TEST (C++ enumerator), 554  
tcpip\_adapter\_init (C++ function), 545  
TCPIP\_ADAPTER\_IP\_ADDRESS\_LEASE\_TIME  
(C++ enumerator), 555  
TCPIP\_ADAPTER\_IP\_REQUEST\_RETRY\_TIME  
(C++ enumerator), 555  
tcpip\_adapter\_is\_netif\_up (C++ function),  
552  
TCPIP\_ADAPTER\_OP\_GET (C++ enumerator), 555  
TCPIP\_ADAPTER\_OP\_MAX (C++ enumerator), 555  
TCPIP\_ADAPTER\_OP\_SET (C++ enumerator), 555  
TCPIP\_ADAPTER\_OP\_START (C++ enumerator),  
555  
tcpip\_adapter\_option\_id\_t (C++ type), 554  
tcpip\_adapter\_option\_mode\_t (C++ type),  
554  
TCPIP\_ADAPTER\_REQUESTED\_IP\_ADDRESS  
(C++ enumerator), 555

- TCPIP\_ADAPTER\_ROUTER\_SOLICITATION\_ADDRESS (C++ enumerator), 555
- tcpip\_adapter\_set\_default\_eth\_handlers (C++ function), 552
- tcpip\_adapter\_set\_default\_wifi\_handlers (C++ function), 552
- tcpip\_adapter\_set\_dns\_info (C++ function), 547
- tcpip\_adapter\_set\_hostname (C++ function), 551
- tcpip\_adapter\_set\_ip\_info (C++ function), 546
- tcpip\_adapter\_set\_old\_ip\_info (C++ function), 548
- tcpip\_adapter\_sta\_input (C++ function), 551
- tcpip\_adapter\_sta\_start (C++ function), 545
- tcpip\_adapter\_stop (C++ function), 546
- tcpip\_adapter\_test\_start (C++ function), 552
- tcpip\_adapter\_up (C++ function), 546
- TCPIP\_HOSTNAME\_MAX\_SIZE (C 宏), 554
- TIMER\_0 (C++ enumerator), 721
- TIMER\_1 (C++ enumerator), 721
- TIMER\_ALARM\_DIS (C++ enumerator), 722
- TIMER\_ALARM\_EN (C++ enumerator), 722
- TIMER\_ALARM\_MAX (C++ enumerator), 722
- timer\_alarm\_t (C++ enum), 722
- TIMER\_AUTORELOAD\_DIS (C++ enumerator), 722
- TIMER\_AUTORELOAD\_EN (C++ enumerator), 722
- TIMER\_AUTORELOAD\_MAX (C++ enumerator), 722
- timer\_autoreload\_t (C++ enum), 722
- TIMER\_BASE\_CLK (C 宏), 721
- timer\_config\_t (C++ class), 720
- timer\_config\_t::alarm\_en (C++ member), 721
- timer\_config\_t::auto\_reload (C++ member), 721
- timer\_config\_t::counter\_dir (C++ member), 721
- timer\_config\_t::counter\_en (C++ member), 721
- timer\_config\_t::divider (C++ member), 721
- timer\_config\_t::intr\_type (C++ member), 721
- timer\_count\_dir\_t (C++ enum), 721
- TIMER\_COUNT\_DOWN (C++ enumerator), 721
- TIMER\_COUNT\_MAX (C++ enumerator), 722
- TIMER\_COUNT\_UP (C++ enumerator), 722
- timer\_disable\_intr (C++ function), 720
- timer\_enable\_intr (C++ function), 720
- timer\_get\_alarm\_value (C++ function), 719
- timer\_get\_config (C++ function), 720
- timer\_get\_counter\_time\_sec (C++ function), 717
- timer\_get\_counter\_value (C++ function), 717
- TIMER\_GROUP\_0 (C++ enumerator), 721
- TIMER\_GROUP\_1 (C++ enumerator), 721
- timer\_group\_intr\_disable (C++ function), 720
- timer\_group\_intr\_enable (C++ function), 720
- TIMER\_GROUP\_MAX (C++ enumerator), 721
- timer\_group\_t (C++ enum), 721
- timer\_idx\_t (C++ enum), 721
- timer\_init (C++ function), 719
- TIMER\_INTR\_LEVEL (C++ enumerator), 722
- TIMER\_INTR\_MAX (C++ enumerator), 722
- timer\_intr\_mode\_t (C++ enum), 722
- timer\_isr\_handle\_t (C++ type), 721
- timer\_isr\_register (C++ function), 719
- TIMER\_MAX (C++ enumerator), 721
- TIMER\_PAUSE (C++ enumerator), 722
- timer\_pause (C++ function), 718
- timer\_set\_alarm (C++ function), 719
- timer\_set\_alarm\_value (C++ function), 719
- timer\_set\_auto\_reload (C++ function), 718
- timer\_set\_counter\_mode (C++ function), 718
- timer\_set\_counter\_value (C++ function), 718
- timer\_set\_divider (C++ function), 718
- TIMER\_START (C++ enumerator), 722
- timer\_start (C++ function), 718
- timer\_start\_t (C++ enum), 722
- TimerCallbackFunction\_t (C++ type), 1008
- TimerHandle\_t (C++ type), 1008
- tls\_keep\_alive\_cfg (C++ class), 767
- tls\_keep\_alive\_cfg::keep\_alive\_count (C++ member), 767
- tls\_keep\_alive\_cfg::keep\_alive\_enable (C++ member), 767
- tls\_keep\_alive\_cfg::keep\_alive\_idle (C++ member), 767
- tls\_keep\_alive\_cfg::keep\_alive\_interval (C++ member), 767
- tls\_keep\_alive\_cfg\_t (C++ type), 770
- TlsDeleteCallbackFunction\_t (C++ type), 959
- tmrCOMMAND\_CHANGE\_PERIOD (C 宏), 999
- tmrCOMMAND\_CHANGE\_PERIOD\_FROM\_ISR (C 宏), 999
- tmrCOMMAND\_DELETE (C 宏), 999
- tmrCOMMAND\_EXECUTE\_CALLBACK (C 宏), 999
- tmrCOMMAND\_EXECUTE\_CALLBACK\_FROM\_ISR (C 宏), 999
- tmrCOMMAND\_RESET (C 宏), 999
- tmrCOMMAND\_RESET\_FROM\_ISR (C 宏), 999
- tmrCOMMAND\_START (C 宏), 999
- tmrCOMMAND\_START\_DONT\_TRACE (C 宏), 999
- tmrCOMMAND\_START\_FROM\_ISR (C 宏), 999
- tmrCOMMAND\_STOP (C 宏), 999
- tmrCOMMAND\_STOP\_FROM\_ISR (C 宏), 999
- tmrFIRST\_FROM\_ISR\_COMMAND (C 宏), 999
- touch\_cnt\_slope\_t (C++ enum), 733
- TOUCH\_FSM\_MODE\_DEFAULT (C 宏), 732
- TOUCH\_FSM\_MODE\_MAX (C++ enumerator), 734
- TOUCH\_FSM\_MODE\_SW (C++ enumerator), 734
- touch\_fsm\_mode\_t (C++ enum), 734
- TOUCH\_FSM\_MODE\_TIMER (C++ enumerator), 734

- `touch_high_volt_t` (C++ enum), 733
- `TOUCH_HVOLT_2V4` (C++ enumerator), 733
- `TOUCH_HVOLT_2V5` (C++ enumerator), 733
- `TOUCH_HVOLT_2V6` (C++ enumerator), 733
- `TOUCH_HVOLT_2V7` (C++ enumerator), 733
- `TOUCH_HVOLT_ATTEN_0V` (C++ enumerator), 733
- `TOUCH_HVOLT_ATTEN_0V5` (C++ enumerator), 733
- `TOUCH_HVOLT_ATTEN_1V` (C++ enumerator), 733
- `TOUCH_HVOLT_ATTEN_1V5` (C++ enumerator), 733
- `TOUCH_HVOLT_ATTEN_KEEP` (C++ enumerator), 733
- `TOUCH_HVOLT_ATTEN_MAX` (C++ enumerator), 733
- `TOUCH_HVOLT_KEEP` (C++ enumerator), 733
- `TOUCH_HVOLT_MAX` (C++ enumerator), 733
- `touch_isr_handle_t` (C++ type), 732
- `touch_low_volt_t` (C++ enum), 733
- `TOUCH_LVOLT_0V5` (C++ enumerator), 733
- `TOUCH_LVOLT_0V6` (C++ enumerator), 733
- `TOUCH_LVOLT_0V7` (C++ enumerator), 733
- `TOUCH_LVOLT_0V8` (C++ enumerator), 733
- `TOUCH_LVOLT_KEEP` (C++ enumerator), 733
- `TOUCH_LVOLT_MAX` (C++ enumerator), 733
- `TOUCH_PAD_BIT_MASK_MAX` (C 宏), 732
- `touch_pad_clear_group_mask` (C++ function), 730
- `touch_pad_clear_status` (C++ function), 730
- `touch_pad_config` (C++ function), 726
- `touch_pad_deinit` (C++ function), 725
- `touch_pad_filter_delete` (C++ function), 731
- `touch_pad_filter_start` (C++ function), 731
- `touch_pad_filter_stop` (C++ function), 731
- `touch_pad_get_cnt_mode` (C++ function), 728
- `touch_pad_get_filter_period` (C++ function), 731
- `touch_pad_get_fsm_mode` (C++ function), 728
- `touch_pad_get_group_mask` (C++ function), 730
- `touch_pad_get_meas_time` (C++ function), 727
- `touch_pad_get_status` (C++ function), 730
- `touch_pad_get_thresh` (C++ function), 729
- `touch_pad_get_trigger_mode` (C++ function), 729
- `touch_pad_get_trigger_source` (C++ function), 729
- `touch_pad_get_voltage` (C++ function), 728
- `touch_pad_get_wakeup_status` (C++ function), 731
- `TOUCH_PAD_GPIO0_CHANNEL` (C 宏), 735
- `TOUCH_PAD_GPIO12_CHANNEL` (C 宏), 735
- `TOUCH_PAD_GPIO13_CHANNEL` (C 宏), 735
- `TOUCH_PAD_GPIO14_CHANNEL` (C 宏), 735
- `TOUCH_PAD_GPIO15_CHANNEL` (C 宏), 735
- `TOUCH_PAD_GPIO27_CHANNEL` (C 宏), 735
- `TOUCH_PAD_GPIO2_CHANNEL` (C 宏), 735
- `TOUCH_PAD_GPIO32_CHANNEL` (C 宏), 735
- `TOUCH_PAD_GPIO33_CHANNEL` (C 宏), 735
- `TOUCH_PAD_GPIO4_CHANNEL` (C 宏), 735
- `touch_pad_init` (C++ function), 725
- `touch_pad_intr_disable` (C++ function), 730
- `touch_pad_intr_enable` (C++ function), 730
- `touch_pad_io_init` (C++ function), 728
- `touch_pad_isr_deregister` (C++ function), 727
- `touch_pad_isr_register` (C++ function), 727
- `TOUCH_PAD_MAX` (C++ enumerator), 732
- `TOUCH_PAD_MEASURE_CYCLE_DEFAULT` (C 宏), 732
- `TOUCH_PAD_MEASURE_WAIT_DEFAULT` (C 宏), 732
- `TOUCH_PAD_NUM0` (C++ enumerator), 732
- `TOUCH_PAD_NUM0_GPIO_NUM` (C 宏), 735
- `TOUCH_PAD_NUM1` (C++ enumerator), 732
- `TOUCH_PAD_NUM1_GPIO_NUM` (C 宏), 735
- `TOUCH_PAD_NUM2` (C++ enumerator), 732
- `TOUCH_PAD_NUM2_GPIO_NUM` (C 宏), 735
- `TOUCH_PAD_NUM3` (C++ enumerator), 732
- `TOUCH_PAD_NUM3_GPIO_NUM` (C 宏), 735
- `TOUCH_PAD_NUM4` (C++ enumerator), 732
- `TOUCH_PAD_NUM4_GPIO_NUM` (C 宏), 735
- `TOUCH_PAD_NUM5` (C++ enumerator), 732
- `TOUCH_PAD_NUM5_GPIO_NUM` (C 宏), 735
- `TOUCH_PAD_NUM6` (C++ enumerator), 732
- `TOUCH_PAD_NUM6_GPIO_NUM` (C 宏), 735
- `TOUCH_PAD_NUM7` (C++ enumerator), 732
- `TOUCH_PAD_NUM7_GPIO_NUM` (C 宏), 735
- `TOUCH_PAD_NUM8` (C++ enumerator), 732
- `TOUCH_PAD_NUM8_GPIO_NUM` (C 宏), 735
- `TOUCH_PAD_NUM9` (C++ enumerator), 732
- `TOUCH_PAD_NUM9_GPIO_NUM` (C 宏), 735
- `touch_pad_read` (C++ function), 726
- `touch_pad_read_filtered` (C++ function), 726
- `touch_pad_read_raw_data` (C++ function), 726
- `touch_pad_set_cnt_mode` (C++ function), 728
- `touch_pad_set_filter_period` (C++ function), 730
- `touch_pad_set_filter_read_cb` (C++ function), 726
- `touch_pad_set_fsm_mode` (C++ function), 728
- `touch_pad_set_group_mask` (C++ function), 730
- `touch_pad_set_meas_time` (C++ function), 727
- `touch_pad_set_thresh` (C++ function), 729
- `touch_pad_set_trigger_mode` (C++ function), 729
- `touch_pad_set_trigger_source` (C++ function), 729
- `touch_pad_set_voltage` (C++ function), 727
- `TOUCH_PAD_SLEEP_CYCLE_DEFAULT` (C 宏), 731
- `TOUCH_PAD_SLOPE_0` (C++ enumerator), 733
- `TOUCH_PAD_SLOPE_1` (C++ enumerator), 733
- `TOUCH_PAD_SLOPE_2` (C++ enumerator), 733
- `TOUCH_PAD_SLOPE_3` (C++ enumerator), 734
- `TOUCH_PAD_SLOPE_4` (C++ enumerator), 734
- `TOUCH_PAD_SLOPE_5` (C++ enumerator), 734
- `TOUCH_PAD_SLOPE_6` (C++ enumerator), 734
- `TOUCH_PAD_SLOPE_7` (C++ enumerator), 734

- TOUCH\_PAD\_SLOPE\_MAX (C++ enumerator), 734  
touch\_pad\_sw\_start (C++ function), 729  
touch\_pad\_t (C++ enum), 732  
TOUCH\_PAD\_TIE\_OPT\_HIGH (C++ enumerator), 734  
TOUCH\_PAD\_TIE\_OPT\_LOW (C++ enumerator), 734  
TOUCH\_PAD\_TIE\_OPT\_MAX (C++ enumerator), 734  
touch\_tie\_opt\_t (C++ enum), 734  
TOUCH\_TRIGGER\_ABOVE (C++ enumerator), 734  
TOUCH\_TRIGGER\_BELOW (C++ enumerator), 734  
TOUCH\_TRIGGER\_MAX (C++ enumerator), 734  
TOUCH\_TRIGGER\_MODE\_DEFAULT (C 宏), 732  
touch\_trigger\_mode\_t (C++ enum), 734  
TOUCH\_TRIGGER\_SOURCE\_BOTH (C++ enumerator), 734  
TOUCH\_TRIGGER\_SOURCE\_DEFAULT (C 宏), 732  
TOUCH\_TRIGGER\_SOURCE\_MAX (C++ enumerator), 734  
TOUCH\_TRIGGER\_SOURCE\_SET1 (C++ enumerator), 734  
touch\_trigger\_src\_t (C++ enum), 734  
touch\_volt\_atten\_t (C++ enum), 733  
transaction\_cb\_t (C++ type), 710  
tskIDLE\_PRIORITY (C 宏), 958  
tskKERNEL\_VERSION\_BUILD (C 宏), 958  
tskKERNEL\_VERSION\_MAJOR (C 宏), 958  
tskKERNEL\_VERSION\_MINOR (C 宏), 958  
tskKERNEL\_VERSION\_NUMBER (C 宏), 958  
tskNO\_AFFINITY (C 宏), 958
- ## U
- UART\_BITRATE\_MAX (C 宏), 751  
UART\_BREAK (C++ enumerator), 753  
UART\_BUFFER\_FULL (C++ enumerator), 753  
uart\_clear\_intr\_status (C++ function), 742  
uart\_config\_t (C++ class), 750  
uart\_config\_t::baud\_rate (C++ member), 750  
uart\_config\_t::data\_bits (C++ member), 750  
uart\_config\_t::flow\_ctrl (C++ member), 750  
uart\_config\_t::parity (C++ member), 750  
uart\_config\_t::rx\_flow\_ctrl\_thresh (C++ member), 750  
uart\_config\_t::stop\_bits (C++ member), 750  
uart\_config\_t::use\_ref\_tick (C++ member), 750  
UART\_CTS\_GPIO19\_DIRECT\_CHANNEL (C 宏), 754  
UART\_CTS\_GPIO6\_DIRECT\_CHANNEL (C 宏), 754  
UART\_CTS\_GPIO8\_DIRECT\_CHANNEL (C 宏), 754  
UART\_DATA (C++ enumerator), 753  
UART\_DATA\_5\_BITS (C++ enumerator), 752  
UART\_DATA\_6\_BITS (C++ enumerator), 752  
UART\_DATA\_7\_BITS (C++ enumerator), 752  
UART\_DATA\_8\_BITS (C++ enumerator), 752  
UART\_DATA\_BITS\_MAX (C++ enumerator), 752  
UART\_DATA\_BREAK (C++ enumerator), 753  
uart\_disable\_intr\_mask (C++ function), 743  
uart\_disable\_pattern\_det\_intr (C++ function), 747  
uart\_disable\_rx\_intr (C++ function), 743  
uart\_disable\_tx\_intr (C++ function), 743  
uart\_driver\_delete (C++ function), 745  
uart\_driver\_install (C++ function), 745  
uart\_enable\_intr\_mask (C++ function), 743  
uart\_enable\_pattern\_det\_intr (C++ function), 747  
uart\_enable\_rx\_intr (C++ function), 743  
uart\_enable\_tx\_intr (C++ function), 743  
UART\_EVENT\_MAX (C++ enumerator), 753  
uart\_event\_t (C++ class), 750  
uart\_event\_t::size (C++ member), 751  
uart\_event\_t::type (C++ member), 751  
uart\_event\_type\_t (C++ enum), 753  
UART\_FIFO\_LEN (C 宏), 751  
UART\_FIFO\_OVF (C++ enumerator), 753  
uart\_flush (C++ function), 747  
uart\_flush\_input (C++ function), 747  
UART\_FRAME\_ERR (C++ enumerator), 753  
uart\_get\_baudrate (C++ function), 742  
uart\_get\_buffered\_data\_len (C++ function), 747  
uart\_get\_collision\_flag (C++ function), 749  
uart\_get\_hw\_flow\_ctrl (C++ function), 742  
uart\_get\_parity (C++ function), 741  
uart\_get\_stop\_bits (C++ function), 741  
uart\_get\_wakeup\_threshold (C++ function), 749  
uart\_get\_word\_length (C++ function), 741  
UART\_GPIO10\_DIRECT\_CHANNEL (C 宏), 754  
UART\_GPIO11\_DIRECT\_CHANNEL (C 宏), 754  
UART\_GPIO16\_DIRECT\_CHANNEL (C 宏), 754  
UART\_GPIO17\_DIRECT\_CHANNEL (C 宏), 754  
UART\_GPIO19\_DIRECT\_CHANNEL (C 宏), 753  
UART\_GPIO1\_DIRECT\_CHANNEL (C 宏), 753  
UART\_GPIO22\_DIRECT\_CHANNEL (C 宏), 754  
UART\_GPIO3\_DIRECT\_CHANNEL (C 宏), 753  
UART\_GPIO6\_DIRECT\_CHANNEL (C 宏), 754  
UART\_GPIO7\_DIRECT\_CHANNEL (C 宏), 754  
UART\_GPIO8\_DIRECT\_CHANNEL (C 宏), 754  
UART\_GPIO9\_DIRECT\_CHANNEL (C 宏), 754  
uart\_hw\_flowcontrol\_t (C++ enum), 752  
UART\_HW\_FLOWCTRL\_CTS (C++ enumerator), 753  
UART\_HW\_FLOWCTRL\_CTS\_RTS (C++ enumerator), 753  
UART\_HW\_FLOWCTRL\_DISABLE (C++ enumerator), 752  
UART\_HW\_FLOWCTRL\_MAX (C++ enumerator), 753  
UART\_HW\_FLOWCTRL\_RTS (C++ enumerator), 752  
uart\_intr\_config (C++ function), 745  
uart\_intr\_config\_t (C++ class), 750  
uart\_intr\_config\_t::intr\_enable\_mask (C++ member), 750

- uart\_intr\_config\_t::rx\_timeout\_thresh (C++ member), 750
- uart\_intr\_config\_t::rxfifo\_full\_thresh (C++ member), 750
- uart\_intr\_config\_t::txfifo\_empty\_intr\_thresh (C++ member), 750
- UART\_INTR\_MASK (C 宏), 751
- UART\_INVERSE\_CTS (C 宏), 751
- UART\_INVERSE\_DISABLE (C 宏), 751
- UART\_INVERSE\_RTS (C 宏), 751
- UART\_INVERSE\_RXD (C 宏), 751
- UART\_INVERSE\_TXD (C 宏), 751
- uart\_is\_driver\_installed (C++ function), 740
- uart\_isr\_free (C++ function), 744
- uart\_isr\_handle\_t (C++ type), 751
- uart\_isr\_register (C++ function), 744
- UART\_LINE\_INV\_MASK (C 宏), 751
- UART\_MODE\_IRDA (C++ enumerator), 751
- UART\_MODE\_RS485\_APP\_CTRL (C++ enumerator), 751
- UART\_MODE\_RS485\_COLLISION\_DETECT (C++ enumerator), 751
- UART\_MODE\_RS485\_HALF\_DUPLEX (C++ enumerator), 751
- uart\_mode\_t (C++ enum), 751
- UART\_MODE\_UART (C++ enumerator), 751
- UART\_NUM\_0 (C++ enumerator), 752
- UART\_NUM\_0\_CTS\_DIRECT\_GPIO\_NUM (C 宏), 753
- UART\_NUM\_0\_RTS\_DIRECT\_GPIO\_NUM (C 宏), 754
- UART\_NUM\_0\_RXD\_DIRECT\_GPIO\_NUM (C 宏), 753
- UART\_NUM\_0\_TXD\_DIRECT\_GPIO\_NUM (C 宏), 753
- UART\_NUM\_1 (C++ enumerator), 752
- UART\_NUM\_1\_CTS\_DIRECT\_GPIO\_NUM (C 宏), 754
- UART\_NUM\_1\_RTS\_DIRECT\_GPIO\_NUM (C 宏), 754
- UART\_NUM\_1\_RXD\_DIRECT\_GPIO\_NUM (C 宏), 754
- UART\_NUM\_1\_TXD\_DIRECT\_GPIO\_NUM (C 宏), 754
- UART\_NUM\_2\_CTS\_DIRECT\_GPIO\_NUM (C 宏), 754
- UART\_NUM\_2\_RTS\_DIRECT\_GPIO\_NUM (C 宏), 754
- UART\_NUM\_2\_RXD\_DIRECT\_GPIO\_NUM (C 宏), 754
- UART\_NUM\_2\_TXD\_DIRECT\_GPIO\_NUM (C 宏), 754
- UART\_NUM\_MAX (C++ enumerator), 752
- uart\_param\_config (C++ function), 745
- UART\_PARITY\_DISABLE (C++ enumerator), 752
- UART\_PARITY\_ERR (C++ enumerator), 753
- UART\_PARITY\_EVEN (C++ enumerator), 752
- UART\_PARITY\_ODD (C++ enumerator), 752
- uart\_parity\_t (C++ enum), 752
- UART\_PATTERN\_DET (C++ enumerator), 753
- uart\_pattern\_get\_pos (C++ function), 748
- uart\_pattern\_pop\_pos (C++ function), 748
- uart\_pattern\_queue\_reset (C++ function), 748
- UART\_PIN\_NO\_CHANGE (C 宏), 751
- uart\_port\_t (C++ enum), 752
- uart\_read\_bytes (C++ function), 747
- UART\_RTS\_GPIO11\_DIRECT\_CHANNEL (C 宏), 754
- UART\_RTS\_GPIO22\_DIRECT\_CHANNEL (C 宏), 754
- UART\_RTS\_GPIO7\_DIRECT\_CHANNEL (C 宏), 754
- UART\_RXD\_GPIO16\_DIRECT\_CHANNEL (C 宏), 754
- UART\_RXD\_GPIO3\_DIRECT\_CHANNEL (C 宏), 754
- UART\_RXD\_GPIO9\_DIRECT\_CHANNEL (C 宏), 754
- uart\_set\_baudrate (C++ function), 741
- uart\_set\_dtr (C++ function), 744
- uart\_set\_hw\_flow\_ctrl (C++ function), 742
- uart\_set\_line\_inverse (C++ function), 742
- uart\_set\_mode (C++ function), 748
- uart\_set\_parity (C++ function), 741
- uart\_set\_pin (C++ function), 744
- uart\_set\_rts (C++ function), 744
- uart\_set\_rx\_timeout (C++ function), 749
- uart\_set\_stop\_bits (C++ function), 741
- uart\_set\_sw\_flow\_ctrl (C++ function), 742
- uart\_set\_tx\_idle\_num (C++ function), 745
- uart\_set\_wakeup\_threshold (C++ function), 749
- uart\_set\_word\_length (C++ function), 741
- UART\_STOP\_BITS\_1 (C++ enumerator), 752
- UART\_STOP\_BITS\_1\_5 (C++ enumerator), 752
- UART\_STOP\_BITS\_2 (C++ enumerator), 752
- UART\_STOP\_BITS\_MAX (C++ enumerator), 752
- uart\_stop\_bits\_t (C++ enum), 752
- uart\_tx\_chars (C++ function), 746
- UART\_TXD\_GPIO10\_DIRECT\_CHANNEL (C 宏), 754
- UART\_TXD\_GPIO17\_DIRECT\_CHANNEL (C 宏), 754
- UART\_TXD\_GPIO1\_DIRECT\_CHANNEL (C 宏), 754
- uart\_wait\_tx\_done (C++ function), 746
- uart\_word\_length\_t (C++ enum), 752
- uart\_write\_bytes (C++ function), 746
- uart\_write\_bytes\_with\_break (C++ function), 746
- UINT16 (C++ type), 296
- UINT32 (C++ type), 296
- UINT64 (C++ type), 296
- UINT8 (C++ type), 296
- ulp\_load\_binary (C++ function), 1467, 1471
- ulp\_process\_macros\_and\_load (C++ function), 1461
- ulp\_run (C++ function), 1462, 1468, 1471



ulp\_set\_wakeup\_period (C++ function), 1468, 1472  
 ulTaskNotifyTake (C++ function), 956  
 uxQueueMessagesWaiting (C++ function), 964  
 uxQueueMessagesWaitingFromISR (C++ function), 961  
 uxQueueSpacesAvailable (C++ function), 964  
 uxSemaphoreGetCount (C 宏), 992  
 uxTaskGetNumberOfTasks (C++ function), 948  
 uxTaskGetStackHighWaterMark (C++ function), 948  
 uxTaskGetSystemState (C++ function), 950  
 uxTaskPriorityGet (C++ function), 943  
 uxTaskPriorityGetFromISR (C++ function), 944  
**V**  
 vendor\_ie\_data\_t (C++ class), 481  
 vendor\_ie\_data\_t::element\_id (C++ member), 481  
 vendor\_ie\_data\_t::length (C++ member), 481  
 vendor\_ie\_data\_t::payload (C++ member), 482  
 vendor\_ie\_data\_t::vendor\_oui (C++ member), 481  
 vendor\_ie\_data\_t::vendor\_oui\_type (C++ member), 481  
 vEventGroupDelete (C++ function), 1014  
 vprintf\_like\_t (C++ type), 1082  
 vQueueAddToRegistry (C++ function), 965  
 vQueueDelete (C++ function), 964  
 vQueueUnregisterQueue (C++ function), 965  
 vRingbufferDelete (C++ function), 1028  
 vRingbufferGetInfo (C++ function), 1030  
 vRingbufferReturnItem (C++ function), 1028  
 vRingbufferReturnItemFromISR (C++ function), 1028  
 vSemaphoreDelete (C 宏), 992  
 VSPI\_HOST (C 宏), 701  
 vTaskDelay (C++ function), 942  
 vTaskDelayUntil (C++ function), 942  
 vTaskDelete (C++ function), 941  
 vTaskGetRunTimeStats (C++ function), 952  
 vTaskList (C++ function), 952  
 vTaskNotifyGiveFromISR (C++ function), 955  
 vTaskPrioritySet (C++ function), 944  
 vTaskResume (C++ function), 945  
 vTaskSetApplicationTaskTag (C++ function), 949  
 vTaskSetThreadLocalStoragePointer (C++ function), 949  
 vTaskSetThreadLocalStoragePointerAndDeleteCallback (C++ function), 949  
 vTaskSuspend (C++ function), 945  
 vTaskSuspendAll (C++ function), 946  
 vTimerSetTimerID (C++ function), 996

**W**

WEBSOCKET\_EVENT\_ANY (C++ enumerator), 786  
 WEBSOCKET\_EVENT\_CONNECTED (C++ enumerator), 786  
 WEBSOCKET\_EVENT\_DATA (C++ enumerator), 786  
 WEBSOCKET\_EVENT\_DISCONNECTED (C++ enumerator), 786  
 WEBSOCKET\_EVENT\_ERROR (C++ enumerator), 786  
 WEBSOCKET\_EVENT\_MAX (C++ enumerator), 786  
 WEBSOCKET\_TRANSPORT\_OVER\_SSL (C++ enumerator), 786  
 WEBSOCKET\_TRANSPORT\_OVER\_TCP (C++ enumerator), 786  
 WEBSOCKET\_TRANSPORT\_UNKNOWN (C++ enumerator), 786  
 wifi\_active\_scan\_time\_t (C++ class), 478  
 wifi\_active\_scan\_time\_t::max (C++ member), 478  
 wifi\_active\_scan\_time\_t::min (C++ member), 478  
 WIFI\_ALL\_CHANNEL\_SCAN (C++ enumerator), 491  
 WIFI\_AMPDU\_RX\_ENABLED (C 宏), 476  
 WIFI\_AMPDU\_TX\_ENABLED (C 宏), 476  
 WIFI\_ANT\_ANT0 (C++ enumerator), 490  
 WIFI\_ANT\_ANT1 (C++ enumerator), 490  
 wifi\_ant\_config\_t (C++ class), 484  
 wifi\_ant\_config\_t::enabled\_ant0 (C++ member), 485  
 wifi\_ant\_config\_t::enabled\_ant1 (C++ member), 485  
 wifi\_ant\_config\_t::rx\_ant\_default (C++ member), 484  
 wifi\_ant\_config\_t::rx\_ant\_mode (C++ member), 484  
 wifi\_ant\_config\_t::tx\_ant\_mode (C++ member), 485  
 wifi\_ant\_gpio\_config\_t (C++ class), 484  
 wifi\_ant\_gpio\_config\_t::gpio\_cfg (C++ member), 484  
 wifi\_ant\_gpio\_t (C++ class), 484  
 wifi\_ant\_gpio\_t::gpio\_num (C++ member), 484  
 wifi\_ant\_gpio\_t::gpio\_select (C++ member), 484  
 WIFI\_ANT\_MAX (C++ enumerator), 491  
 WIFI\_ANT\_MODE\_ANT0 (C++ enumerator), 492  
 WIFI\_ANT\_MODE\_ANT1 (C++ enumerator), 492  
 WIFI\_ANT\_MODE\_AUTO (C++ enumerator), 492  
 WIFI\_ANT\_MODE\_MAX (C++ enumerator), 492  
 wifi\_ant\_mode\_t (C++ enum), 492  
 wifi\_ant\_t (C++ enum), 490  
 wifi\_ap\_config\_t (C++ class), 480  
 wifi\_ap\_config\_t::authmode (C++ member), 480  
 wifi\_ap\_config\_t::beacon\_interval (C++ member), 480  
 wifi\_ap\_config\_t::channel (C++ member), 480

- wifi\_ap\_config\_t::max\_connection (C++ member), 480
- wifi\_ap\_config\_t::password (C++ member), 480
- wifi\_ap\_config\_t::ssid (C++ member), 480
- wifi\_ap\_config\_t::ssid\_hidden (C++ member), 480
- wifi\_ap\_config\_t::ssid\_len (C++ member), 480
- wifi\_ap\_record\_t (C++ class), 478
- wifi\_ap\_record\_t::ant (C++ member), 479
- wifi\_ap\_record\_t::authmode (C++ member), 479
- wifi\_ap\_record\_t::bssid (C++ member), 478
- wifi\_ap\_record\_t::country (C++ member), 479
- wifi\_ap\_record\_t::group\_cipher (C++ member), 479
- wifi\_ap\_record\_t::pairwise\_cipher (C++ member), 479
- wifi\_ap\_record\_t::phy\_11b (C++ member), 479
- wifi\_ap\_record\_t::phy\_11g (C++ member), 479
- wifi\_ap\_record\_t::phy\_11n (C++ member), 479
- wifi\_ap\_record\_t::phy\_lr (C++ member), 479
- wifi\_ap\_record\_t::primary (C++ member), 479
- wifi\_ap\_record\_t::reserved (C++ member), 479
- wifi\_ap\_record\_t::rssi (C++ member), 479
- wifi\_ap\_record\_t::second (C++ member), 479
- wifi\_ap\_record\_t::ssid (C++ member), 478
- wifi\_ap\_record\_t::wps (C++ member), 479
- WIFI\_AUTH\_MAX (C++ enumerator), 489
- wifi\_auth\_mode\_t (C++ enum), 488
- WIFI\_AUTH\_OPEN (C++ enumerator), 488
- WIFI\_AUTH\_WEP (C++ enumerator), 489
- WIFI\_AUTH\_WPA2\_ENTERPRISE (C++ enumerator), 489
- WIFI\_AUTH\_WPA2\_PSK (C++ enumerator), 489
- WIFI\_AUTH\_WPA2\_WPA3\_PSK (C++ enumerator), 489
- WIFI\_AUTH\_WPA3\_PSK (C++ enumerator), 489
- WIFI\_AUTH\_WPA\_PSK (C++ enumerator), 489
- WIFI\_AUTH\_WPA\_WPA2\_PSK (C++ enumerator), 489
- wifi\_bandwidth\_t (C++ enum), 491
- WIFI\_BW\_HT20 (C++ enumerator), 491
- WIFI\_BW\_HT40 (C++ enumerator), 491
- WIFI\_CACHE\_TX\_BUFFER\_NUM (C 宏), 476
- WIFI\_CIPHER\_TYPE\_AES\_CMAC128 (C++ enumerator), 490
- WIFI\_CIPHER\_TYPE\_CCMP (C++ enumerator), 490
- WIFI\_CIPHER\_TYPE\_NONE (C++ enumerator), 490
- wifi\_cipher\_type\_t (C++ enum), 490
- WIFI\_CIPHER\_TYPE\_TKIP (C++ enumerator), 490
- WIFI\_CIPHER\_TYPE\_TKIP\_CCMP (C++ enumerator), 490
- WIFI\_CIPHER\_TYPE\_UNKNOWN (C++ enumerator), 490
- WIFI\_CIPHER\_TYPE\_WEP104 (C++ enumerator), 490
- WIFI\_CIPHER\_TYPE\_WEP40 (C++ enumerator), 490
- wifi\_config\_t (C++ union), 477
- wifi\_config\_t::ap (C++ member), 477
- wifi\_config\_t::sta (C++ member), 477
- WIFI\_CONNECT\_AP\_BY\_SECURITY (C++ enumerator), 491
- WIFI\_CONNECT\_AP\_BY\_SIGNAL (C++ enumerator), 491
- WIFI\_COUNTRY\_POLICY\_AUTO (C++ enumerator), 488
- WIFI\_COUNTRY\_POLICY\_MANUAL (C++ enumerator), 488
- wifi\_country\_policy\_t (C++ enum), 488
- wifi\_country\_t (C++ class), 477
- wifi\_country\_t::cc (C++ member), 477
- wifi\_country\_t::max\_tx\_power (C++ member), 477
- wifi\_country\_t::nchan (C++ member), 477
- wifi\_country\_t::policy (C++ member), 478
- wifi\_country\_t::schan (C++ member), 477
- wifi\_csi\_cb\_t (C++ type), 477
- wifi\_csi\_config\_t (C++ class), 483
- wifi\_csi\_config\_t::channel\_filter\_en (C++ member), 484
- wifi\_csi\_config\_t::htlft\_en (C++ member), 483
- wifi\_csi\_config\_t::lltf\_en (C++ member), 483
- wifi\_csi\_config\_t::lft\_merge\_en (C++ member), 483
- wifi\_csi\_config\_t::manu\_scale (C++ member), 484
- wifi\_csi\_config\_t::shift (C++ member), 484
- wifi\_csi\_config\_t::stbc\_htlft2\_en (C++ member), 483
- WIFI\_CSI\_ENABLED (C 宏), 476
- wifi\_csi\_info\_t (C++ class), 484
- wifi\_csi\_info\_t::buf (C++ member), 484
- wifi\_csi\_info\_t::first\_word\_invalid (C++ member), 484
- wifi\_csi\_info\_t::len (C++ member), 484
- wifi\_csi\_info\_t::mac (C++ member), 484
- wifi\_csi\_info\_t::rx\_ctrl (C++ member), 484
- WIFI\_DEFAULT\_RX\_BA\_WIN (C 宏), 476
- WIFI\_DYNAMIC\_TX\_BUFFER\_NUM (C 宏), 476
- wifi\_err\_reason\_t (C++ enum), 489
- wifi\_event\_ap\_probe\_req\_rx\_t (C++ class),

- 486
- wifi\_event\_ap\_probe\_req\_rx\_t::mac (C++ member), 487
- wifi\_event\_ap\_probe\_req\_rx\_t::rssi (C++ member), 487
- WIFI\_EVENT\_AP\_PROBEREQRECVED (C++ enumerator), 494
- WIFI\_EVENT\_AP\_STACONNECTED (C++ enumerator), 494
- wifi\_event\_ap\_staconnected\_t (C++ class), 486
- wifi\_event\_ap\_staconnected\_t::aid (C++ member), 486
- wifi\_event\_ap\_staconnected\_t::mac (C++ member), 486
- WIFI\_EVENT\_AP\_STADISCONNECTED (C++ enumerator), 494
- wifi\_event\_ap\_stadisconnected\_t (C++ class), 486
- wifi\_event\_ap\_stadisconnected\_t::aid (C++ member), 486
- wifi\_event\_ap\_stadisconnected\_t::mac (C++ member), 486
- WIFI\_EVENT\_AP\_START (C++ enumerator), 494
- WIFI\_EVENT\_AP\_STOP (C++ enumerator), 494
- WIFI\_EVENT\_MASK\_ALL (C 宏), 488
- WIFI\_EVENT\_MASK\_AP\_PROBEREQRECVED (C 宏), 488
- WIFI\_EVENT\_MASK\_NONE (C 宏), 488
- WIFI\_EVENT\_SCAN\_DONE (C++ enumerator), 494
- WIFI\_EVENT\_STA\_AUTHMODE\_CHANGE (C++ enumerator), 494
- wifi\_event\_sta\_authmode\_change\_t (C++ class), 485
- wifi\_event\_sta\_authmode\_change\_t::new\_mode (C++ member), 486
- wifi\_event\_sta\_authmode\_change\_t::old\_mode (C++ member), 486
- WIFI\_EVENT\_STA\_BEACON\_TIMEOUT (C++ enumerator), 494
- WIFI\_EVENT\_STA\_CONNECTED (C++ enumerator), 494
- wifi\_event\_sta\_connected\_t (C++ class), 485
- wifi\_event\_sta\_connected\_t::authmode (C++ member), 485
- wifi\_event\_sta\_connected\_t::bssid (C++ member), 485
- wifi\_event\_sta\_connected\_t::channel (C++ member), 485
- wifi\_event\_sta\_connected\_t::ssid (C++ member), 485
- wifi\_event\_sta\_connected\_t::ssid\_len (C++ member), 485
- WIFI\_EVENT\_STA\_DISCONNECTED (C++ enumerator), 494
- wifi\_event\_sta\_disconnected\_t (C++ class), 485
- wifi\_event\_sta\_disconnected\_t::bssid (C++ member), 485
- wifi\_event\_sta\_disconnected\_t::reason (C++ member), 485
- wifi\_event\_sta\_disconnected\_t::ssid (C++ member), 485
- wifi\_event\_sta\_disconnected\_t::ssid\_len (C++ member), 485
- wifi\_event\_sta\_scan\_done\_t (C++ class), 485
- wifi\_event\_sta\_scan\_done\_t::number (C++ member), 485
- wifi\_event\_sta\_scan\_done\_t::scan\_id (C++ member), 485
- wifi\_event\_sta\_scan\_done\_t::status (C++ member), 485
- WIFI\_EVENT\_STA\_START (C++ enumerator), 494
- WIFI\_EVENT\_STA\_STOP (C++ enumerator), 494
- WIFI\_EVENT\_STA\_WPS\_ER\_FAILED (C++ enumerator), 494
- WIFI\_EVENT\_STA\_WPS\_ER\_PBC\_OVERLAP (C++ enumerator), 494
- WIFI\_EVENT\_STA\_WPS\_ER\_PIN (C++ enumerator), 494
- wifi\_event\_sta\_wps\_er\_pin\_t (C++ class), 486
- wifi\_event\_sta\_wps\_er\_pin\_t::pin\_code (C++ member), 486
- WIFI\_EVENT\_STA\_WPS\_ER\_SUCCESS (C++ enumerator), 494
- wifi\_event\_sta\_wps\_er\_success\_t (C++ class), 486
- wifi\_event\_sta\_wps\_er\_success\_t::ap\_cred (C++ member), 486
- wifi\_event\_sta\_wps\_er\_success\_t::ap\_cred\_cnt (C++ member), 486
- wifi\_event\_sta\_wps\_er\_success\_t::passphrase (C++ member), 486
- wifi\_event\_sta\_wps\_er\_success\_t::ssid (C++ member), 486
- WIFI\_EVENT\_STA\_WPS\_ER\_TIMEOUT (C++ enumerator), 494
- wifi\_event\_sta\_wps\_fail\_reason\_t (C++ enum), 495
- wifi\_event\_t (C++ enum), 494
- WIFI\_EVENT\_WIFI\_READY (C++ enumerator), 494
- WIFI\_FAST\_SCAN (C++ enumerator), 491
- WIFI\_IF\_AP (C 宏), 487
- WIFI\_IF\_STA (C 宏), 487
- WIFI\_INIT\_CONFIG\_DEFAULT (C 宏), 476
- WIFI\_INIT\_CONFIG\_MAGIC (C 宏), 476
- wifi\_init\_config\_t (C++ class), 474
- wifi\_init\_config\_t::ampdu\_rx\_enable (C++ member), 475
- wifi\_init\_config\_t::ampdu\_tx\_enable (C++ member), 475
- wifi\_init\_config\_t::beacon\_max\_len (C++ member), 475

- wifi\_init\_config\_t::cache\_tx\_buf\_num (C++ member), 475
- wifi\_init\_config\_t::csi\_enable (C++ member), 475
- wifi\_init\_config\_t::dynamic\_rx\_buf\_num (C++ member), 474
- wifi\_init\_config\_t::dynamic\_tx\_buf\_num (C++ member), 474
- wifi\_init\_config\_t::event\_handler (C++ member), 474
- wifi\_init\_config\_t::feature\_caps (C++ member), 475
- wifi\_init\_config\_t::magic (C++ member), 475
- wifi\_init\_config\_t::mgmt\_sbuf\_num (C++ member), 475
- wifi\_init\_config\_t::nano\_enable (C++ member), 475
- wifi\_init\_config\_t::nvs\_enable (C++ member), 475
- wifi\_init\_config\_t::osi\_funcs (C++ member), 474
- wifi\_init\_config\_t::rx\_ba\_win (C++ member), 475
- wifi\_init\_config\_t::static\_rx\_buf\_num (C++ member), 474
- wifi\_init\_config\_t::static\_tx\_buf\_num (C++ member), 474
- wifi\_init\_config\_t::tx\_buf\_type (C++ member), 474
- wifi\_init\_config\_t::wifi\_task\_core\_id (C++ member), 475
- wifi\_init\_config\_t::wpa\_crypto\_funcs (C++ member), 474
- wifi\_interface\_t (C++ type), 488
- WIFI\_MGMT\_SBUF\_NUM (C 宏), 476
- WIFI\_MODE\_AP (C++ enumerator), 488
- WIFI\_MODE\_APSTA (C++ enumerator), 488
- WIFI\_MODE\_MAX (C++ enumerator), 488
- WIFI\_MODE\_NULL (C++ enumerator), 488
- WIFI\_MODE\_STA (C++ enumerator), 488
- wifi\_mode\_t (C++ enum), 488
- WIFI\_NANO\_FORMAT\_ENABLED (C 宏), 476
- WIFI\_NVS\_ENABLED (C 宏), 476
- WIFI\_PHY\_RATE\_11M\_L (C++ enumerator), 492
- WIFI\_PHY\_RATE\_11M\_S (C++ enumerator), 492
- WIFI\_PHY\_RATE\_12M (C++ enumerator), 493
- WIFI\_PHY\_RATE\_18M (C++ enumerator), 493
- WIFI\_PHY\_RATE\_1M\_L (C++ enumerator), 492
- WIFI\_PHY\_RATE\_24M (C++ enumerator), 493
- WIFI\_PHY\_RATE\_2M\_L (C++ enumerator), 492
- WIFI\_PHY\_RATE\_2M\_S (C++ enumerator), 492
- WIFI\_PHY\_RATE\_36M (C++ enumerator), 493
- WIFI\_PHY\_RATE\_48M (C++ enumerator), 493
- WIFI\_PHY\_RATE\_54M (C++ enumerator), 493
- WIFI\_PHY\_RATE\_5M\_L (C++ enumerator), 492
- WIFI\_PHY\_RATE\_5M\_S (C++ enumerator), 492
- WIFI\_PHY\_RATE\_6M (C++ enumerator), 493
- WIFI\_PHY\_RATE\_9M (C++ enumerator), 493
- WIFI\_PHY\_RATE\_LORA\_250K (C++ enumerator), 494
- WIFI\_PHY\_RATE\_LORA\_500K (C++ enumerator), 494
- WIFI\_PHY\_RATE\_MAX (C++ enumerator), 494
- WIFI\_PHY\_RATE\_MCS0\_LGI (C++ enumerator), 493
- WIFI\_PHY\_RATE\_MCS0\_SGI (C++ enumerator), 493
- WIFI\_PHY\_RATE\_MCS1\_LGI (C++ enumerator), 493
- WIFI\_PHY\_RATE\_MCS1\_SGI (C++ enumerator), 493
- WIFI\_PHY\_RATE\_MCS2\_LGI (C++ enumerator), 493
- WIFI\_PHY\_RATE\_MCS2\_SGI (C++ enumerator), 493
- WIFI\_PHY\_RATE\_MCS3\_LGI (C++ enumerator), 493
- WIFI\_PHY\_RATE\_MCS3\_SGI (C++ enumerator), 493
- WIFI\_PHY\_RATE\_MCS4\_LGI (C++ enumerator), 493
- WIFI\_PHY\_RATE\_MCS4\_SGI (C++ enumerator), 493
- WIFI\_PHY\_RATE\_MCS5\_LGI (C++ enumerator), 493
- WIFI\_PHY\_RATE\_MCS5\_SGI (C++ enumerator), 493
- WIFI\_PHY\_RATE\_MCS6\_LGI (C++ enumerator), 493
- WIFI\_PHY\_RATE\_MCS6\_SGI (C++ enumerator), 493
- WIFI\_PHY\_RATE\_MCS7\_LGI (C++ enumerator), 493
- WIFI\_PHY\_RATE\_MCS7\_SGI (C++ enumerator), 494
- wifi\_phy\_rate\_t (C++ enum), 492
- WIFI\_PKT\_CTRL (C++ enumerator), 492
- WIFI\_PKT\_DATA (C++ enumerator), 492
- WIFI\_PKT\_MGMT (C++ enumerator), 492
- WIFI\_PKT\_MISC (C++ enumerator), 492
- wifi\_pkt\_rx\_ctrl\_t (C++ class), 482
- wifi\_pkt\_rx\_ctrl\_t::\_\_pad0\_\_ (C++ member), 482
- wifi\_pkt\_rx\_ctrl\_t::\_\_pad1\_\_ (C++ member), 482
- wifi\_pkt\_rx\_ctrl\_t::\_\_pad2\_\_ (C++ member), 482
- wifi\_pkt\_rx\_ctrl\_t::\_\_pad3\_\_ (C++ member), 482
- wifi\_pkt\_rx\_ctrl\_t::\_\_pad4\_\_ (C++ member), 483
- wifi\_pkt\_rx\_ctrl\_t::\_\_pad5\_\_ (C++ member), 483
- wifi\_pkt\_rx\_ctrl\_t::\_\_pad6\_\_ (C++ member), 483

- wifi\_pkt\_rx\_ctrl\_t::\_\_pad7\_\_ (C++ member), 483
- wifi\_pkt\_rx\_ctrl\_t::aggregation (C++ member), 482
- wifi\_pkt\_rx\_ctrl\_t::ampdu\_cnt (C++ member), 482
- wifi\_pkt\_rx\_ctrl\_t::ant (C++ member), 483
- wifi\_pkt\_rx\_ctrl\_t::channel (C++ member), 482
- wifi\_pkt\_rx\_ctrl\_t::cwb (C++ member), 482
- wifi\_pkt\_rx\_ctrl\_t::fec\_coding (C++ member), 482
- wifi\_pkt\_rx\_ctrl\_t::mcs (C++ member), 482
- wifi\_pkt\_rx\_ctrl\_t::noise\_floor (C++ member), 482
- wifi\_pkt\_rx\_ctrl\_t::not\_sounding (C++ member), 482
- wifi\_pkt\_rx\_ctrl\_t::rate (C++ member), 482
- wifi\_pkt\_rx\_ctrl\_t::rssi (C++ member), 482
- wifi\_pkt\_rx\_ctrl\_t::rx\_state (C++ member), 483
- wifi\_pkt\_rx\_ctrl\_t::secondary\_channel (C++ member), 482
- wifi\_pkt\_rx\_ctrl\_t::sgi (C++ member), 482
- wifi\_pkt\_rx\_ctrl\_t::sig\_len (C++ member), 483
- wifi\_pkt\_rx\_ctrl\_t::sig\_mode (C++ member), 482
- wifi\_pkt\_rx\_ctrl\_t::smoothing (C++ member), 482
- wifi\_pkt\_rx\_ctrl\_t::stbc (C++ member), 482
- wifi\_pkt\_rx\_ctrl\_t::timestamp (C++ member), 483
- wifi\_pmf\_config\_t (C++ class), 479
- wifi\_pmf\_config\_t::capable (C++ member), 479
- wifi\_pmf\_config\_t::required (C++ member), 479
- WIFI\_PROMIS\_CTRL\_FILTER\_MASK\_ACK (C 宏), 488
- WIFI\_PROMIS\_CTRL\_FILTER\_MASK\_ALL (C 宏), 487
- WIFI\_PROMIS\_CTRL\_FILTER\_MASK\_BA (C 宏), 487
- WIFI\_PROMIS\_CTRL\_FILTER\_MASK\_BAR (C 宏), 487
- WIFI\_PROMIS\_CTRL\_FILTER\_MASK\_CFEND (C 宏), 488
- WIFI\_PROMIS\_CTRL\_FILTER\_MASK\_CFENDACK (C 宏), 488
- WIFI\_PROMIS\_CTRL\_FILTER\_MASK\_CTS (C 宏), 487
- WIFI\_PROMIS\_CTRL\_FILTER\_MASK\_PSPOLL (C 宏), 487
- WIFI\_PROMIS\_CTRL\_FILTER\_MASK\_RTS (C 宏), 487
- WIFI\_PROMIS\_CTRL\_FILTER\_MASK\_WRAPPER (C 宏), 487
- WIFI\_PROMIS\_FILTER\_MASK\_ALL (C 宏), 487
- WIFI\_PROMIS\_FILTER\_MASK\_CTRL (C 宏), 487
- WIFI\_PROMIS\_FILTER\_MASK\_DATA (C 宏), 487
- WIFI\_PROMIS\_FILTER\_MASK\_DATA\_AMPDU (C 宏), 487
- WIFI\_PROMIS\_FILTER\_MASK\_DATA\_MPDU (C 宏), 487
- WIFI\_PROMIS\_FILTER\_MASK\_FCSFAIL (C 宏), 487
- WIFI\_PROMIS\_FILTER\_MASK\_MGMT (C 宏), 487
- WIFI\_PROMIS\_FILTER\_MASK\_MISC (C 宏), 487
- wifi\_promiscuous\_cb\_t (C++ type), 476
- wifi\_promiscuous\_filter\_t (C++ class), 483
- wifi\_promiscuous\_filter\_t::filter\_mask (C++ member), 483
- wifi\_promiscuous\_pkt\_t (C++ class), 483
- wifi\_promiscuous\_pkt\_t::payload (C++ member), 483
- wifi\_promiscuous\_pkt\_t::rx\_ctrl (C++ member), 483
- wifi\_promiscuous\_pkt\_type\_t (C++ enum), 492
- WIFI\_PROTOCOL\_11B (C 宏), 487
- WIFI\_PROTOCOL\_11G (C 宏), 487
- WIFI\_PROTOCOL\_11N (C 宏), 487
- WIFI\_PROTOCOL\_LR (C 宏), 487
- wifi\_prov\_cb\_event\_t (C++ enum), 856
- wifi\_prov\_cb\_func\_t (C++ type), 856
- wifi\_prov\_config\_data\_handler (C++ function), 858
- wifi\_prov\_config\_get\_data\_t (C++ class), 858
- wifi\_prov\_config\_get\_data\_t::conn\_info (C++ member), 859
- wifi\_prov\_config\_get\_data\_t::fail\_reason (C++ member), 859
- wifi\_prov\_config\_get\_data\_t::wifi\_state (C++ member), 859
- wifi\_prov\_config\_handlers (C++ class), 859
- wifi\_prov\_config\_handlers::apply\_config\_handler (C++ member), 859
- wifi\_prov\_config\_handlers::ctx (C++ member), 859
- wifi\_prov\_config\_handlers::get\_status\_handler (C++ member), 859
- wifi\_prov\_config\_handlers::set\_config\_handler (C++ member), 859
- wifi\_prov\_config\_handlers\_t (C++ type), 859
- wifi\_prov\_config\_set\_data\_t (C++ class), 859
- wifi\_prov\_config\_set\_data\_t::bssid (C++ member), 859
- wifi\_prov\_config\_set\_data\_t::channel (C++ member), 859

- wifi\_prov\_config\_set\_data\_t::password (C++ member), 859
- wifi\_prov\_config\_set\_data\_t::ssid (C++ member), 859
- WIFI\_PROV\_CRED\_FAIL (C++ enumerator), 856
- WIFI\_PROV\_CRED\_RECV (C++ enumerator), 856
- WIFI\_PROV\_CRED\_SUCCESS (C++ enumerator), 857
- wifi\_prov\_ctx\_t (C++ type), 859
- WIFI\_PROV\_DEINIT (C++ enumerator), 857
- WIFI\_PROV\_END (C++ enumerator), 857
- WIFI\_PROV\_EVENT\_HANDLER\_NONE (C 宏), 856
- wifi\_prov\_event\_handler\_t (C++ class), 855
- wifi\_prov\_event\_handler\_t::event\_cb (C++ member), 855
- wifi\_prov\_event\_handler\_t::user\_data (C++ member), 855
- WIFI\_PROV\_INIT (C++ enumerator), 856
- wifi\_prov\_mgr\_config\_t (C++ class), 855
- wifi\_prov\_mgr\_config\_t::app\_event\_handler (C++ member), 856
- wifi\_prov\_mgr\_config\_t::scheme (C++ member), 856
- wifi\_prov\_mgr\_config\_t::scheme\_event\_handler (C++ member), 856
- wifi\_prov\_mgr\_configure\_sta (C++ function), 854
- wifi\_prov\_mgr\_deinit (C++ function), 851
- wifi\_prov\_mgr\_disable\_auto\_stop (C++ function), 852
- wifi\_prov\_mgr\_endpoint\_create (C++ function), 853
- wifi\_prov\_mgr\_endpoint\_register (C++ function), 853
- wifi\_prov\_mgr\_endpoint\_unregister (C++ function), 854
- wifi\_prov\_mgr\_event\_handler (C++ function), 854
- wifi\_prov\_mgr\_get\_wifi\_disconnect\_reason (C++ function), 854
- wifi\_prov\_mgr\_get\_wifi\_state (C++ function), 854
- wifi\_prov\_mgr\_init (C++ function), 851
- wifi\_prov\_mgr\_is\_provisioned (C++ function), 851
- wifi\_prov\_mgr\_set\_app\_info (C++ function), 853
- wifi\_prov\_mgr\_start\_provisioning (C++ function), 851
- wifi\_prov\_mgr\_stop\_provisioning (C++ function), 852
- wifi\_prov\_mgr\_wait (C++ function), 852
- wifi\_prov\_scheme (C++ class), 855
- wifi\_prov\_scheme::delete\_config (C++ member), 855
- wifi\_prov\_scheme::new\_config (C++ member), 855
- wifi\_prov\_scheme::prov\_start (C++ member), 855
- wifi\_prov\_scheme::prov\_stop (C++ member), 855
- wifi\_prov\_scheme::set\_config\_endpoint (C++ member), 855
- wifi\_prov\_scheme::set\_config\_service (C++ member), 855
- wifi\_prov\_scheme::wifi\_mode (C++ member), 855
- wifi\_prov\_scheme\_ble\_event\_cb\_free\_ble (C++ function), 857
- wifi\_prov\_scheme\_ble\_event\_cb\_free\_bt (C++ function), 857
- wifi\_prov\_scheme\_ble\_event\_cb\_free\_btadm (C++ function), 857
- WIFI\_PROV\_SCHEME\_BLE\_EVENT\_HANDLER\_FREE\_BLE (C 宏), 857
- WIFI\_PROV\_SCHEME\_BLE\_EVENT\_HANDLER\_FREE\_BT (C 宏), 857
- WIFI\_PROV\_SCHEME\_BLE\_EVENT\_HANDLER\_FREE\_BTDM (C 宏), 857
- wifi\_prov\_scheme\_ble\_set\_service\_uuid (C++ function), 857
- wifi\_prov\_scheme\_softap\_set\_httpd\_handle (C++ function), 858
- wifi\_prov\_scheme\_t (C++ type), 856
- wifi\_prov\_security (C++ enum), 857
- WIFI\_PROV\_SECURITY\_0 (C++ enumerator), 857
- WIFI\_PROV\_SECURITY\_1 (C++ enumerator), 857
- wifi\_prov\_security\_t (C++ type), 856
- WIFI\_PROV\_STA\_AP\_NOT\_FOUND (C++ enumerator), 860
- WIFI\_PROV\_STA\_AUTH\_ERROR (C++ enumerator), 860
- wifi\_prov\_sta\_conn\_info\_t (C++ class), 858
- wifi\_prov\_sta\_conn\_info\_t::auth\_mode (C++ member), 858
- wifi\_prov\_sta\_conn\_info\_t::bssid (C++ member), 858
- wifi\_prov\_sta\_conn\_info\_t::channel (C++ member), 858
- wifi\_prov\_sta\_conn\_info\_t::ip\_addr (C++ member), 858
- wifi\_prov\_sta\_conn\_info\_t::ssid (C++ member), 858
- WIFI\_PROV\_STA\_CONNECTED (C++ enumerator), 860
- WIFI\_PROV\_STA\_CONNECTING (C++ enumerator), 860
- WIFI\_PROV\_STA\_DISCONNECTED (C++ enumerator), 860
- wifi\_prov\_sta\_fail\_reason\_t (C++ enum), 860
- wifi\_prov\_sta\_state\_t (C++ enum), 860
- WIFI\_PROV\_START (C++ enumerator), 856
- WIFI\_PS\_MAX\_MODEM (C++ enumerator), 491
- WIFI\_PS\_MIN\_MODEM (C++ enumerator), 491
- WIFI\_PS\_NONE (C++ enumerator), 491

- wifi\_ps\_type\_t (C++ enum), 491
- WIFI\_REASON\_4WAY\_HANDSHAKE\_TIMEOUT (C++ enumerator), 489
- WIFI\_REASON\_802\_1X\_AUTH\_FAILED (C++ enumerator), 489
- WIFI\_REASON\_AKMP\_INVALID (C++ enumerator), 489
- WIFI\_REASON\_AP\_TSF\_RESET (C++ enumerator), 490
- WIFI\_REASON\_ASSOC\_EXPIRE (C++ enumerator), 489
- WIFI\_REASON\_ASSOC\_FAIL (C++ enumerator), 490
- WIFI\_REASON\_ASSOC\_LEAVE (C++ enumerator), 489
- WIFI\_REASON\_ASSOC\_NOT\_AUTHED (C++ enumerator), 489
- WIFI\_REASON\_ASSOC\_TOOMANY (C++ enumerator), 489
- WIFI\_REASON\_AUTH\_EXPIRE (C++ enumerator), 489
- WIFI\_REASON\_AUTH\_FAIL (C++ enumerator), 490
- WIFI\_REASON\_AUTH\_LEAVE (C++ enumerator), 489
- WIFI\_REASON\_BEACON\_TIMEOUT (C++ enumerator), 490
- WIFI\_REASON\_CIPHER\_SUITE\_REJECTED (C++ enumerator), 489
- WIFI\_REASON\_CONNECTION\_FAIL (C++ enumerator), 490
- WIFI\_REASON\_DISASSOC\_PWRCAP\_BAD (C++ enumerator), 489
- WIFI\_REASON\_DISASSOC\_SUPCHAN\_BAD (C++ enumerator), 489
- WIFI\_REASON\_GROUP\_CIPHER\_INVALID (C++ enumerator), 489
- WIFI\_REASON\_GROUP\_KEY\_UPDATE\_TIMEOUT (C++ enumerator), 489
- WIFI\_REASON\_HANDSHAKE\_TIMEOUT (C++ enumerator), 490
- WIFI\_REASON\_IE\_IN\_4WAY\_DIFFERS (C++ enumerator), 489
- WIFI\_REASON\_IE\_INVALID (C++ enumerator), 489
- WIFI\_REASON\_INVALID\_PMKID (C++ enumerator), 489
- WIFI\_REASON\_INVALID\_RSN\_IE\_CAP (C++ enumerator), 489
- WIFI\_REASON\_MIC\_FAILURE (C++ enumerator), 489
- WIFI\_REASON\_NO\_AP\_FOUND (C++ enumerator), 490
- WIFI\_REASON\_NOT\_ASSOCED (C++ enumerator), 489
- WIFI\_REASON\_NOT\_AUTHED (C++ enumerator), 489
- WIFI\_REASON\_PAIRWISE\_CIPHER\_INVALID (C++ enumerator), 489
- WIFI\_REASON\_UNSPECIFIED (C++ enumerator), 489
- WIFI\_REASON\_UNSUPP\_RSN\_IE\_VERSION (C++ enumerator), 489
- wifi\_scan\_config\_t (C++ class), 478
- wifi\_scan\_config\_t::bssid (C++ member), 478
- wifi\_scan\_config\_t::channel (C++ member), 478
- wifi\_scan\_config\_t::scan\_time (C++ member), 478
- wifi\_scan\_config\_t::scan\_type (C++ member), 478
- wifi\_scan\_config\_t::show\_hidden (C++ member), 478
- wifi\_scan\_config\_t::ssid (C++ member), 478
- wifi\_scan\_method\_t (C++ enum), 491
- wifi\_scan\_threshold\_t (C++ class), 479
- wifi\_scan\_threshold\_t::authmode (C++ member), 479
- wifi\_scan\_threshold\_t::rssi (C++ member), 479
- wifi\_scan\_time\_t (C++ class), 478
- wifi\_scan\_time\_t::active (C++ member), 478
- wifi\_scan\_time\_t::passive (C++ member), 478
- WIFI\_SCAN\_TYPE\_ACTIVE (C++ enumerator), 490
- WIFI\_SCAN\_TYPE\_PASSIVE (C++ enumerator), 490
- wifi\_scan\_type\_t (C++ enum), 490
- WIFI\_SECOND\_CHAN\_ABOVE (C++ enumerator), 490
- WIFI\_SECOND\_CHAN\_BELOW (C++ enumerator), 490
- WIFI\_SECOND\_CHAN\_NONE (C++ enumerator), 490
- wifi\_second\_chan\_t (C++ enum), 490
- WIFI\_SOFTAP\_BEACON\_MAX\_LEN (C 宏), 476
- wifi\_sort\_method\_t (C++ enum), 491
- wifi\_sta\_config\_t (C++ class), 480
- wifi\_sta\_config\_t::bssid (C++ member), 480
- wifi\_sta\_config\_t::bssid\_set (C++ member), 480
- wifi\_sta\_config\_t::channel (C++ member), 480
- wifi\_sta\_config\_t::listen\_interval (C++ member), 480
- wifi\_sta\_config\_t::password (C++ member), 480
- wifi\_sta\_config\_t::pmf\_cfg (C++ member), 481
- wifi\_sta\_config\_t::scan\_method (C++ member), 480
- wifi\_sta\_config\_t::sort\_method (C++ member), 480
- wifi\_sta\_config\_t::ssid (C++ member), 480

wifi\_sta\_config\_t::threshold (C++ member), 481

wifi\_sta\_info\_t (C++ class), 481

wifi\_sta\_info\_t::mac (C++ member), 481

wifi\_sta\_info\_t::phy\_11b (C++ member), 481

wifi\_sta\_info\_t::phy\_11g (C++ member), 481

wifi\_sta\_info\_t::phy\_11n (C++ member), 481

wifi\_sta\_info\_t::phy\_1r (C++ member), 481

wifi\_sta\_info\_t::reserved (C++ member), 481

wifi\_sta\_info\_t::rssi (C++ member), 481

wifi\_sta\_list\_t (C++ class), 481

wifi\_sta\_list\_t::num (C++ member), 481

wifi\_sta\_list\_t::sta (C++ member), 481

WIFI\_STATIC\_TX\_BUFFER\_NUM (C 宏), 476

WIFI\_STATIS\_ALL (C 宏), 488

WIFI\_STATIS\_BUFFER (C 宏), 488

WIFI\_STATIS\_DIAG (C 宏), 488

WIFI\_STATIS\_HW (C 宏), 488

WIFI\_STATIS\_RXTX (C 宏), 488

WIFI\_STORAGE\_FLASH (C++ enumerator), 491

WIFI\_STORAGE\_RAM (C++ enumerator), 491

wifi\_storage\_t (C++ enum), 491

WIFI\_TASK\_CORE\_ID (C 宏), 476

WIFI\_VENDOR\_IE\_ELEMENT\_ID (C 宏), 487

wifi\_vendor\_ie\_id\_t (C++ enum), 491

wifi\_vendor\_ie\_type\_t (C++ enum), 491

WIFI\_VND\_IE\_ID\_0 (C++ enumerator), 492

WIFI\_VND\_IE\_ID\_1 (C++ enumerator), 492

WIFI\_VND\_IE\_TYPE\_ASSOC\_REQ (C++ enumerator), 491

WIFI\_VND\_IE\_TYPE\_ASSOC\_RESP (C++ enumerator), 491

WIFI\_VND\_IE\_TYPE\_BEACON (C++ enumerator), 491

WIFI\_VND\_IE\_TYPE\_PROBE\_REQ (C++ enumerator), 491

WIFI\_VND\_IE\_TYPE\_PROBE\_RESP (C++ enumerator), 491

wl\_erase\_range (C++ function), 927

wl\_handle\_t (C++ type), 929

WL\_INVALID\_HANDLE (C 宏), 928

wl\_mount (C++ function), 927

wl\_read (C++ function), 928

wl\_sector\_size (C++ function), 928

wl\_size (C++ function), 928

wl\_unmount (C++ function), 927

wl\_write (C++ function), 928

WPS\_FAIL\_REASON\_MAX (C++ enumerator), 495

WPS\_FAIL\_REASON\_NORMAL (C++ enumerator), 495

WPS\_FAIL\_REASON\_RECV\_M2D (C++ enumerator), 495

## X

xEventGroupClearBits (C++ function), 1011

xEventGroupClearBitsFromISR (C 宏), 1014

xEventGroupCreate (C++ function), 1008

xEventGroupCreateStatic (C++ function), 1009

xEventGroupGetBits (C 宏), 1016

xEventGroupGetBitsFromISR (C++ function), 1014

xEventGroupSetBits (C++ function), 1011

xEventGroupSetBitsFromISR (C 宏), 1015

xEventGroupSync (C++ function), 1012

xEventGroupWaitBits (C++ function), 1009

xQueueAddToSet (C++ function), 967

xQueueCreate (C 宏), 968

xQueueCreateSet (C++ function), 966

xQueueCreateStatic (C 宏), 968

xQueueGenericCreate (C++ function), 966

xQueueGenericCreateStatic (C++ function), 966

xQueueGenericReceive (C++ function), 962

xQueueGenericSend (C++ function), 961

xQueueGenericSendFromISR (C++ function), 960

xQueueGiveFromISR (C++ function), 961

xQueueIsQueueEmptyFromISR (C++ function), 961

xQueueIsQueueFullFromISR (C++ function), 961

xQueueOverwrite (C 宏), 972

xQueueOverwriteFromISR (C 宏), 977

xQueuePeek (C 宏), 973

xQueuePeekFromISR (C++ function), 962

xQueueReceive (C 宏), 975

xQueueReceiveFromISR (C++ function), 964

xQueueRemoveFromSet (C++ function), 967

xQueueReset (C 宏), 979

xQueueSelectFromSet (C++ function), 967

xQueueSelectFromSetFromISR (C++ function), 967

xQueueSend (C 宏), 971

xQueueSendFromISR (C 宏), 978

xQueueSendToBack (C 宏), 970

xQueueSendToBackFromISR (C 宏), 976

xQueueSendToFront (C 宏), 969

xQueueSendToFrontFromISR (C 宏), 976

xRingbufferAddToQueueSetRead (C++ function), 1029

xRingbufferCanRead (C++ function), 1029

xRingbufferCreate (C++ function), 1025


xRingbufferCreateNoSplit (C++ function), 1025

xRingbufferCreateStatic (C++ function), 1025

xRingbufferGetCurFreeSize (C++ function), 1029

xRingbufferGetMaxItemSize (C++ function), 1028



- [xRingbufferPrintInfo \(C++ function\), 1030](#)  
[xRingbufferReceive \(C++ function\), 1026](#)  
[xRingbufferReceiveFromISR \(C++ function\), 1027](#)  
[xRingbufferReceiveSplit \(C++ function\), 1027](#)  
[xRingbufferReceiveSplitFromISR \(C++ function\), 1027](#)  
[xRingbufferReceiveUpTo \(C++ function\), 1027](#)  
[xRingbufferReceiveUpToFromISR \(C++ function\), 1028](#)  
[xRingbufferRemoveFromQueueSetRead \(C++ function\), 1029](#)  
[xRingbufferSend \(C++ function\), 1025](#)  
[xRingbufferSendAcquire \(C++ function\), 1026](#)  
[xRingbufferSendComplete \(C++ function\), 1026](#)  
[xRingbufferSendFromISR \(C++ function\), 1025](#)  
[xSemaphoreCreateBinary \(C 宏\), 980](#)  
[xSemaphoreCreateBinaryStatic \(C 宏\), 980](#)  
[xSemaphoreCreateCounting \(C 宏\), 990](#)  
[xSemaphoreCreateCountingStatic \(C 宏\), 991](#)  
[xSemaphoreCreateMutex \(C 宏\), 987](#)  
[xSemaphoreCreateMutexStatic \(C 宏\), 987](#)  
[xSemaphoreCreateRecursiveMutex \(C 宏\), 988](#)  
[xSemaphoreCreateRecursiveMutexStatic \(C 宏\), 989](#)  
[xSemaphoreGetMutexHolder \(C 宏\), 992](#)  
[xSemaphoreGive \(C 宏\), 983](#)  
[xSemaphoreGiveFromISR \(C 宏\), 985](#)  
[xSemaphoreGiveRecursive \(C 宏\), 984](#)  
[xSemaphoreTake \(C 宏\), 981](#)  
[xSemaphoreTakeFromISR \(C 宏\), 986](#)  
[xSemaphoreTakeRecursive \(C 宏\), 982](#)  
[xSTATIC\\_RINGBUFFER \(C++ class\), 1030](#)  
[xTASK\\_SNAPSHOT \(C++ class\), 957](#)  
[xTASK\\_SNAPSHOT::pxEndOfStack \(C++ member\), 957](#)  
[xTASK\\_SNAPSHOT::pxTCB \(C++ member\), 957](#)  
[xTASK\\_SNAPSHOT::pxTopOfStack \(C++ member\), 957](#)  
[xTASK\\_STATUS \(C++ class\), 957](#)  
[xTASK\\_STATUS::eCurrentState \(C++ member\), 957](#)  
[xTASK\\_STATUS::pcTaskName \(C++ member\), 957](#)  
[xTASK\\_STATUS::pxStackBase \(C++ member\), 957](#)  
[xTASK\\_STATUS::ulRunTimeCounter \(C++ member\), 957](#)  
[xTASK\\_STATUS::usStackHighWaterMark \(C++ member\), 957](#)  
[xTASK\\_STATUS::uxBasePriority \(C++ member\), 957](#)  
[xTASK\\_STATUS::uxCurrentPriority \(C++ member\), 957](#)  
[xTASK\\_STATUS::xCoreID \(C++ member\), 957](#)  
[xTASK\\_STATUS::xHandle \(C++ member\), 957](#)  
[xTASK\\_STATUS::xTaskNumber \(C++ member\), 957](#)  
[xTaskCallApplicationTaskHook \(C++ function\), 950](#)  
[xTaskCreate \(C++ function\), 938](#)  
[xTaskCreatePinnedToCore \(C++ function\), 938](#)  
[xTaskCreateStatic \(C++ function\), 940](#)  
[xTaskCreateStaticPinnedToCore \(C++ function\), 939](#)  
[xTaskGetApplicationTaskTag \(C++ function\), 949](#)  
[xTaskGetIdleTaskHandle \(C++ function\), 950](#)  
[xTaskGetIdleTaskHandleForCPU \(C++ function\), 950](#)  
[xTaskGetTickCount \(C++ function\), 948](#)  
[xTaskGetTickCountFromISR \(C++ function\), 948](#)  
[xTaskNotify \(C++ function\), 953](#)  
[xTaskNotifyFromISR \(C++ function\), 953](#)  
[xTaskNotifyGive \(C 宏\), 958](#)  
[xTaskNotifyWait \(C++ function\), 954](#)  
[xTaskResumeAll \(C++ function\), 947](#)  
[xTaskResumeFromISR \(C++ function\), 946](#)  
[xTimerChangePeriod \(C 宏\), 1000](#)  
[xTimerChangePeriodFromISR \(C 宏\), 1006](#)  
[xTimerCreate \(C++ function\), 992](#)  
[xTimerCreateStatic \(C++ function\), 994](#)  
[xTimerDelete \(C 宏\), 1001](#)  
[xTimerGetExpiryTime \(C++ function\), 997](#)  
[xTimerGetPeriod \(C++ function\), 997](#)  
[xTimerGetTimerDaemonTaskHandle \(C++ function\), 997](#)  
[xTimerIsTimerActive \(C++ function\), 997](#)  
[xTimerPendFunctionCall \(C++ function\), 999](#)  
[xTimerPendFunctionCallFromISR \(C++ function\), 997](#)  
[xTimerReset \(C 宏\), 1002](#)  
[xTimerResetFromISR \(C 宏\), 1007](#)  
[xTimerStart \(C 宏\), 999](#)  
[xTimerStartFromISR \(C 宏\), 1004](#)  
[xTimerStop \(C 宏\), 1000](#)  
[xTimerStopFromISR \(C 宏\), 1005](#)
-  环境变量
  - [CONFIG\\_EFUSE\\_CUSTOM\\_TABLE, 1063](#)
  - [CONFIG\\_EFUSE\\_MAX\\_BLK\\_LEN, 1064](#)
  - [CONFIG\\_EFUSE\\_VIRTUAL, 1067](#)
  - [CONFIG\\_ESPTOOLPY\\_FLASHSIZE, 861](#)
  - [CONFIG\\_LOG\\_DEFAULT\\_LEVEL, 1078](#)
  - [CONFIG\\_LWIP\\_USE\\_ONLY\\_LWIP\\_SELECT, 914](#)
  - [CONFIG\\_SPIRAM\\_BANKSWITCH\\_ENABLE, 1053](#)
  - [CONFIG\\_SPIRAM\\_BANKSWITCH\\_RESERVE, 1053](#)

EFUSE\_CODE\_SCHEME\_SELECTOR, [1065](#)