



# **Audio Development Framework Handbook**

**Espressif Systems**

**Sep 14, 2024**



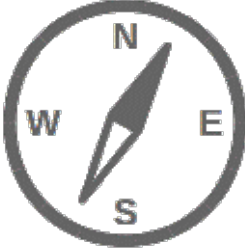


# CONTENTS

<b>1</b>	<b>Get Started</b>	<b>3</b>
1.1	Development Board Overview . . . . .	3
1.2	About ESP-ADF . . . . .	3
1.3	Quick Start . . . . .	4
1.4	Installation Step by Step . . . . .	5
1.5	Step 1. Set up ESP-IDF . . . . .	5
1.6	Step 2. Get ESP-ADF . . . . .	6
1.7	Step 3. Set up the environment . . . . .	6
1.8	Step 4. Start a Project . . . . .	7
1.9	Step 5. Connect Your Device . . . . .	8
1.10	Step 6. Configure . . . . .	8
1.11	Step 7. Build the Project . . . . .	10
1.12	Step 8. Flash onto the Device . . . . .	10
1.13	Step 9. Monitor . . . . .	11
1.14	VS Code Extension . . . . .	12
1.15	IDF Eclipse Plugin and Espressif IDE . . . . .	12
1.16	Update ESP-ADF . . . . .	14
<b>2</b>	<b>API Reference</b>	<b>15</b>
2.1	Audio Framework . . . . .	16
2.2	Audio Streams . . . . .	70
2.3	Playlist . . . . .	100
2.4	Codecs . . . . .	119
2.5	Audio Processing . . . . .	135
2.6	Services . . . . .	149
2.7	Speech Recognition . . . . .	197
2.8	Peripherals . . . . .	207
2.9	Abstraction Layer . . . . .	241
2.10	Configuration Options . . . . .	263
<b>3</b>	<b>Design Guide</b>	<b>327</b>
3.1	Project Design . . . . .	327
3.2	Design Considerations . . . . .	329
3.3	Software Design . . . . .	332
3.4	Development Boards . . . . .	334
3.5	Audio Samples . . . . .	420
<b>4</b>	<b>Resources</b>	<b>423</b>
<b>5</b>	<b>Copyrights and Licenses</b>	<b>425</b>

5.1 Software Copyrights . . . . .	425
<b>6 English-Chinese Glossary</b>	<b>427</b>
<b>7 About</b>	<b>437</b>
<b>Index</b>	<b>439</b>

This is the documentation for Espressif Audio Development Framework (ADF).

		
Get Started	API Reference	Design Guide



## GET STARTED

This document is intended to help users set up the software environment for the development of audio applications using hardware based on the ESP32 family of chips by Espressif. After that, a simple example will show you how to use ESP-ADF (Espressif Audio Development Framework).

### 1.1 Development Board Overview

For easier start with ESP-ADF, Espressif designed ESP32, ESP32-S2, and ESP32-S3 based development boards intended for audio applications. Click the links below to learn more about the available boards.

- [\*ESP32-LyraT\*](#)
- [\*ESP32-LyraT-Mini\*](#)
- [\*ESP32-LyraTD-MSK\*](#)
- [\*ESP32-S2-Kaluga-1\*](#)
- [\*ESP32-Korvo-DU1906\*](#)
- [\*ESP32-S3-Korvo-2\*](#)
- [\*ESP32-C3-Lyra\*](#)

If you do not have any of the above boards, you can still use ESP-ADF for the ESP32 and ESP32-S2 based audio applications. For this, your board needs to have a compatible audio codec or DSP chip; alternatively, you can develop a driver to support communication with your specific chip.

### 1.2 About ESP-ADF

The ESP-ADF is available as a set of [components](#) to extend the functionality already delivered by the [ESP-IDF](#) (Espressif IoT Development Framework).

To use ESP-ADF you need set up the ESP-IDF first, and this is described in the next section.

---

**Note:** ESP-ADF provides support for specific [ESP-IDF versions](#). If you have already set up another version, please switch to a supported ESP-IDF version, or you may not be able to compile ESP-ADF applications. In addition, the python version needs to be between 3.7 and 3.11.

---

### 1.3 Quick Start

This section provides quick steps to run a simple ADF sample project on an ESP device for experienced users. For beginners, please go through the complete steps from *Step 1. Set up ESP-IDF* to *Step 9. Monitor* to build a project.

---

**Note:** If you encounter issues in the following steps, you could refer to the complete steps from *Step 1. Set up ESP-IDF* to *Step 9. Monitor* or describe them in [GitHub Issues](#) or [ESP Forum](#).

---

#### 1.3.1 Linux and macOS

The operating environment below is on Linux Ubuntu 18.04 and above.

1. Download the full ESP-ADF repository from [GitHub](#) by running:

```
git clone --recursive https://github.com/espressif/esp-adf.git
```

For users located in China, it is faster to download from [Gitee](#):

```
git clone --recursive https://gitee.com/EspresifSystems/esp-adf.git
```

2. Configure the `$ESP-IDF` and `$ESP-ADF` compilation environment by running:

```
cd esp-adf
./install.sh
. ./export.sh
```

3. After completing the above environment variable configuration, you can compile the ADF sample project `$ADF_PATH/examples/get-started/play_mp3_control`. Switch to the project's directory, compile, and flash it onto your ESP device by running the following command. Then, you will see the serial port of the routine is printed.

```
cd $ADF_PATH/examples/get-started/play_mp3_control
idf.py build flash monitor
```

#### 1.3.2 Windows

1. Download the full ESP-ADF repository from [GitHub](#) by running:

```
git clone --recursive https://github.com/espressif/esp-adf.git
```

For users located in China, it is faster to download from [Gitee](#):

```
git clone --recursive https://gitee.com/EspresifSystems/esp-adf.git
```

2. Install the `$ESP-IDF` compilation environment in the command prompt window:

```
cd esp-adf
.\install.bat
```

Or first download the full ESP-IDF Windows Installer from [ESP-IDF Windows Installer](#) (Please download the [ESP-IDF versions](#) supported by ESP-ADF). And then turn off the antivirus software (Because it may prevent the installation as the software writes the Windows system regedit) and install the downloaded file. After the installation



is complete, open the ESP-IDF CMD shortcut icon on the desktop, the script will automatically help you download submodules, and set environment variables such as `IDF_PATH`.

3. Set the `ADF_PATH` by running the following commands:

```
.\export.bat  
echo %ADF_PATH%
```

4. If your `ADF_PATH` variable prints correctly, it's time to compile the ADF routines:

```
cd %ADF_PATH%\examples\get-started\play_mp3_control  
idf.py build flash monitor
```

## 1.4 Installation Step by Step

This is a detailed roadmap to walk you through the installation process.

### 1.4.1 Setting up Development Environment

- *Step 1. Set up ESP-IDF for Windows, Linux or Mac OS*
- *Step 2. Get ESP-ADF*
- *Step 3. Set up the environment*

### 1.4.2 Creating Your First Project

- *Step 4. Start a Project*
- *Step 5. Connect Your Device*
- *Step 6. Configure*
- *Step 7. Build the Project*
- *Step 8. Flash onto the Device*
- *Step 9. Monitor*

## 1.5 Step 1. Set up ESP-IDF

Configure your PC according to **Getting Started** section of **ESP-IDF Programming Guide**. Windows, Linux and Mac OS operating systems are supported. Please select and follow the guide specific to [ESP32](#) or [ESP32-S2](#) chip. The chip name is provided in the board name.

---

**Note:** This guide uses the directory `~/esp` on Linux and macOS or `%userprofile%\esp` on Windows as an installation folder for ESP-ADF. You can use any directory, but you will need to adjust paths for the commands accordingly. Keep in mind that ESP-ADF does not support spaces in paths.

---

To make the installation easier and less prone to errors, use the `~/esp` default directory for the installation.

If this is your first exposure to the [ESP-IDF](#), then it is recommended to get familiar with **hello\_world** or **blink** example first.

After getting familiar with ESP-IDF, decide on which ESP-IDF version to use for your application depending on the Espressif chip that you have and your project type. For this, consult [Versions](#) section of ESP-IDF Programming Guide.

Once you successfully build, upload, and run examples for your version of ESP-IDF, you can proceed to the next step.

## 1.6 Step 2. Get ESP-ADF

Now you can start installing audio-specific API / libraries provided in [ESP-ADF repository](#).

### 1.6.1 Windows

Open Command Prompt and run the following commands:

```
cd %userprofile%\esp
git clone --recursive https://github.com/espressif/esp-adf.git
```

### 1.6.2 Linux and macOS

Open Terminal, and run the following commands:

```
cd ~/esp
git clone --recursive https://github.com/espressif/esp-adf.git
```

## 1.7 Step 3. Set up the environment

Before being able to compile ESP-ADF projects, on each new session, ESP-IDF tools should be added to the PATH environment variable. To make the tools usable from the command line, some environment variables must be set. ESP-ADF provides a script which does that.

### 1.7.1 Windows

[ESP-IDF Tools Installer](#) for Windows creates an “ESP-IDF Command Prompt” shortcut in the Start Menu. This shortcut opens the Command Prompt and sets up all the required environment variables. You can open this shortcut and proceed to the next step.

Alternatively, if you want to use ESP-IDF in an existing Command Prompt window, you can run:

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

## 1.7.2 Linux and macOS

In the terminal where you have installed ESP-IDF, run:

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

Note the space between the leading dot and the path!

You can also create an alias for the export script to your `.profile` or `.bash_profile` script. This way you can set up the environment in a new terminal window by typing `get_idf`:

```
alias get_idf='. $HOME/esp/esp-adf/export.sh'
```

Note that it is not recommended to source `export.sh` from the profile script directly. Doing so activates IDF virtual environment in every terminal session (even in those where IDF is not needed), defeating the purpose of the virtual environment and likely affecting other software.

## 1.8 Step 4. Start a Project

After initial preparation you are ready to build the first audio application. The process has already been described in ESP-IDF documentation. Now we would like to discuss remaining key steps and show how the toolchain is able to access the ESP-ADF components by using the `ADF_PATH` variable.

To demonstrate how to build an application, we will use `get-started/play_mp3_control` project from `examples` directory in the ADF.

### 1.8.1 Windows

```
cd %userprofile%\esp  
xcopy /e /i %ADF_PATH%\examples\get-started\play_mp3_control play_mp3_control
```

### 1.8.2 Linux and macOS

```
cd ~/esp  
cp -r $ADF_PATH/examples/get-started/play_mp3_control .
```

There is a range of example projects in the `examples` directory in ESP-ADF. You can copy any project in the same way as presented above and run it.

It is also possible to build examples in-place, without copying them first.

---

**Important:** The ESP-IDF build system does not support spaces in the paths to either ESP-IDF or to projects.

---

## 1.9 Step 5. Connect Your Device

Connect the audio board to the PC, check under what serial port the board is visible and verify, if serial communication works as described in [ESP-IDF documentation](#).

---

**Note:** Keep the port name handy as you will need it in the next steps.

---

## 1.10 Step 6. Configure

Navigate to your `play_mp3_control` directory from *Step 4. Start a Project* and configure the project:

### 1.10.1 Windows

```
cd %userprofile%\esp\play_mp3_control
idf.py set-target esp32
idf.py menuconfig
```

### 1.10.2 Linux and macOS

```
cd ~/esp/play_mp3_control
idf.py set-target esp32
idf.py menuconfig
```

---

**Note:** If you are using an ESP32-S2 based board, then the second command above should be `idf.py set-target esp32s2`.

---

Setting the target with `idf.py set-target <target>` should be done once, after opening a new project. If the project contains some existing builds and configuration, they will be cleared and initialized. The target may be saved in environment variable to skip this step at all. See [Selecting the Target](#) in ESP-IDF Programming Guide for additional information.

If the previous steps have been done correctly, the following menu appears:

You are using this menu to set up your board type and other project specific variables, e.g. Wi-Fi network name and password, the processor speed, etc.

Select your board from the menu, press `S` to save configuration and then `Q` to exit.

---

**Note:** The colors of the menu could be different in your terminal. You can change the appearance with the option `--style`. Please run `idf.py menuconfig --help` for further information.

---

```

(Top)
      Espressif IoT Development Framework Configuration
SDK tool configuration --->
Build type --->
Application manager --->
Bootloader config --->
Security features --->
Serial flasher config --->
Partition Table --->
My Audio Board --->
Audio HAL --->
Recorder Engine Configuration --->
ESP Speech Recognition --->
Compiler options --->
Component config --->
Compatibility options --->

[Space/Enter] Toggle/enter  [ESC] Leave menu          [S] Save
[O] Load                    [?] Symbol info          [/] Jump to symbol
[F] Toggle show-help mode   [C] Toggle show-name mode [A] Toggle show-all mode
[Q] Quit (prompts for save) [D] Save minimal config (advanced)

```

Fig. 1: Project configuration - Home window

```

(Top) → Audio HAL → Audio board
      Espressif IoT Development Framework Configuration
( ) Custom audio board
(X) ESP32-Lyrat V4.3
( ) ESP32-Lyrat V4.2
( ) ESP32-LyraTD-MSC V2.1
( ) ESP32-LyraTD-MSC V2.2
( ) ESP32-Lyrat-Mini V1.1
( ) ESP32_KORVO_DU1906
( ) ESP32-S2-Kaluga-1 v1.2

[Space/Enter] Toggle/enter  [ESC] Leave menu          [S] Save
[O] Load                    [?] Symbol info          [/] Jump to symbol
[F] Toggle show-help mode   [C] Toggle show-name mode [A] Toggle show-all mode
[Q] Quit (prompts for save) [D] Save minimal config (advanced)

```

Fig. 2: Project configuration - Board selection

## 1.11 Step 7. Build the Project

Build the project by running:

```
idf.py build
```

This command will compile the application and all ESP-IDF and ESP-ADF components, then it will generate the boot-loader, partition table, and application binaries.

```
$ idf.py build
Executing action: all (aliases: build)
Running ninja in directory /path/to/esp/play_mp3_control/build
Executing "ninja all"...
[0/1] Re-running CMake...

... (more lines of build system output)

[1064/1064] Generating binary image from built executable
esptool.py v3.0-dev
Generated /path/to/esp/play_mp3_control/build/play_mp3_control.bin

Project build complete. To flash it, run this command:
/path/to/.espressif/python_env/idf4.2_py2.7_env/bin/python ../esp-idf/components/
↪esptool_py/esptool/esptool.py -p (PORT) -b 460800 --before default_reset --after_
↪hard_reset --chip esp32 write_flash --flash_mode dio --flash_size detect --flash_
↪freq 40m 0x1000 build/bootloader/bootloader.bin 0x8000 build/partition_table/
↪partition-table.bin 0x10000 build/play_mp3_control.bin
or run 'idf.py -p (PORT) flash'
```

If there are no errors, the build will finish by generating the firmware binary .bin file.

## 1.12 Step 8. Flash onto the Device

Flash the binaries that you just built onto your board by running:

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

Replace PORT with your board's serial port name from [Step 5. Connect Your Device](#).

You can also change the flasher baud rate by replacing BAUD with the baud rate you need. The default baud rate is 460800.

For more information on idf.py arguments, see [Using the Build System](#) in ESP-IDF Programming Guide.

---

**Note:** The option `flash` automatically builds and flashes the project, so running `idf.py build` is not necessary.

---

To upload the binaries, the board should be put into upload mode. To do so, hold down **Boot** button, momentarily press **Reset** button and release the **Boot** button. The upload mode may be initiated anytime during the application build, but no later than “Connecting” message is being displayed:

```
...
esptool.py v3.0-dev
```

(continues on next page)

(continued from previous page)

```
Serial port /dev/ttyUSB0
Connecting....._____.
```

Without the upload mode enabled, after showing several `....._____`, the connection will eventually time out.

Once build and upload is complete, you should see the following:

```
...
Leaving...
Hard resetting via RTS pin...
Executing action: monitor
Running idf_monitor in directory /path/to/esp/play_mp3_control
Executing "/path/to/.espressif/python_env/idf4.2_py2.7_env/bin/python /path/to/esp/
↳ esp-idf/tools/idf_monitor.py -p /dev/ttyUSB0 -b 115200 --toolchain-prefix xtensa-
↳ esp32-elf- /path/to/esp/play_mp3_control/build/play_mp3_control.elf -m '/path/to/.
↳ espressif/python_env/idf4.2_py2.7_env/bin/python' '/path/to/esp/esp-idf/tools/idf.py
↳ '..."
--- idf_monitor on /dev/ttyUSB0 115200 ---
--- Quit: Ctrl+] | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---
```

If there are no issues by the end of the flash process, the board will reboot and start up the “play\_mp3\_control” application.

## 1.13 Step 9. Monitor

At this point press the **Reset** button to start the application. Following several lines of start up log, the play\_mp3\_control application specific messages should be displayed:

```
...
I (397) PLAY_FLASH_MP3_CONTROL: [ 1 ] Start audio codec chip
I (427) PLAY_FLASH_MP3_CONTROL: [ 2 ] Create audio pipeline, add all elements to
↳ pipeline, and subscribe pipeline event
I (427) PLAY_FLASH_MP3_CONTROL: [2.1] Create mp3 decoder to decode mp3 file and set
↳ custom read callback
I (437) PLAY_FLASH_MP3_CONTROL: [2.2] Create i2s stream to write data to codec chip
I (467) PLAY_FLASH_MP3_CONTROL: [2.3] Register all elements to audio pipeline
I (467) PLAY_FLASH_MP3_CONTROL: [2.4] Link it together [mp3_music_read_cb]-->mp3_
↳ decoder-->i2s_stream-->[codec_chip]
I (477) PLAY_FLASH_MP3_CONTROL: [ 3 ] Set up event listener
I (477) PLAY_FLASH_MP3_CONTROL: [3.1] Listening event from all elements of pipeline
I (487) PLAY_FLASH_MP3_CONTROL: [ 4 ] Start audio_pipeline
I (507) PLAY_FLASH_MP3_CONTROL: [ * ] Receive music info from mp3 decoder, sample_
↳ rates=44100, bits=16, ch=2
I (7277) PLAY_FLASH_MP3_CONTROL: [ 5 ] Stop audio_pipeline
```

If there are no issues, besides the above log, you should hear a sound played for about 7 seconds by the speakers or headphones connected to your audio board. Reset the board to hear it again if required.

Now you are ready to try some other [examples](#), or go right to developing your own applications. Check how the [examples](#) are made aware of location of the ESP-ADF. Open the [get-started/play\\_mp3\\_control/Makefile](#) and you should see

```
include($ENV{ADF_PATH}/CMakeLists.txt)
include($ENV{IDF_PATH}/tools/cmake/project.cmake)
```

The first line contains `ADF_PATH` to point the toolchain to another file in ESP-ADF directory that provides configuration variables and path to ESP-ADF components reacquired by the toolchain. You need similar `Makefile` in your own applications developed with the ESP-ADF.

### 1.14 VS Code Extension

1. Follow [VS Code Extension Quick Installation Guide](#) to install ESP-IDF Visual Studio Code Extension. If the previous steps have been done correctly, the following toolbar appears:

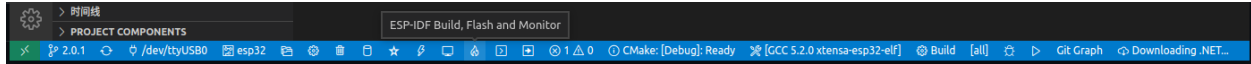


Fig. 3: VS Code Extension Toolbar

2. To install the ESP-ADF extension, open Command Palette and enter `install adf`. Then, a progress bar shows up in the lower right corner.

If you have cloned the ESP-ADF repository before, please enter `open settings(ui)` in Command Palette. Go to `User > Extensions > ESP_IDF` and manually set the ESP-ADF path in `idf.espAdfPath` or `idf.espAdfPathWin` (for Windows). You can also set the ESP-ADF path in `.vscode/settings.json`.

3. In Command Palette, enter `show examples project`, and then a window will be opened with a list of example projects.
4. Select an example, click `Create project using example XX`, and select the directory to save the current example.
5. On the toolbar at the bottom of VS Code, click the gear symbol `menuconfig` to configure the example and click the column symbol `Build` to build the example. See available [shortcut keys](#) for VS code extensions.
6. On the toolbar at the bottom of VS Code, click the plug-in symbol `Select Port` to configure the serial port and click the lightning symbol `Flash Device` to flash firmware. After the firmware is flashed successfully, click `Monitor Device` to start the monitor function. Or, you can also use the flame symbol to build, flash, and monitor the example at the same time.

### 1.15 IDF Eclipse Plugin and Espressif IDE

#### 1.15.1 Install and Set up Environment Variables

1. Follow [IDF Eclipse Plugin Quick Installation Guide](#) to install IDF Eclipse Plugin or download and install Espressif IDE from [Espressif IDE Download Link](#). If the previous steps have been done correctly, you can create, build and flash IDF project in the Eclipse environment.
2. To install ESP-ADF, follow section [Step 2. Get ESP-ADF](#).

3. To set `ADF_PATH` environment variable, open `Window > Preferences > C/C++ > Build > Environment` panel, click **Add** button and fill in `ADF_PATH`. After you complete the above steps, select `ADF_PATH` in `Environment variables` table and click **Edit** and **OK** button without changing any value (There is a bug in Eclipse CDT that is appending a null value before the path hence we need to click on edit and save it).

If this step does not work, you can delete `ADF_PATH` set in Eclipse and set `ADF_PATH` as system environment variable. For Windows, set environment variable in `Advanced System Setting` panel. For Linux and macOS, add `export ADF_PATH=your adf path` in file `/etc/profile`. However,





7. Click **Launch** button to flash the project.
8. Click **Open a Terminal** button and select **ESP-IDF Serial Monitor** to view serial output.

For more information about IDF Eclipse Plugin and Espressif IDE, please refer to [ESP-IDF Eclipse Plugin](#).

### 1.16 Update ESP-ADF

After some time of using ESP-ADF, you may want to update it to take advantage of new features or bug fixes. The simplest way to do so is by deleting existing `esp-adf` folder and cloning it again, which is same as when doing initial installation described in sections *Step 2. Get ESP-ADF*.

Another solution is to update only what has changed. This method is useful if you have a slow connection to the GitHub. To do the update run the following commands:

```
cd ~/esp/esp-adf
git pull
git submodule update --init --recursive
```

The `git pull` command is fetching and merging changes from ESP-ADF repository on GitHub. Then `git submodule update --init --recursive` is updating existing submodules or getting a fresh copy of new ones. On GitHub the submodules are represented as links to other repositories and require this additional command to get them onto your PC.

## API REFERENCE

This API provides a way to develop audio applications using *Elements* like *Codecs* (Decoders and Encoders), *Streams* or *Audio Processing* functions.

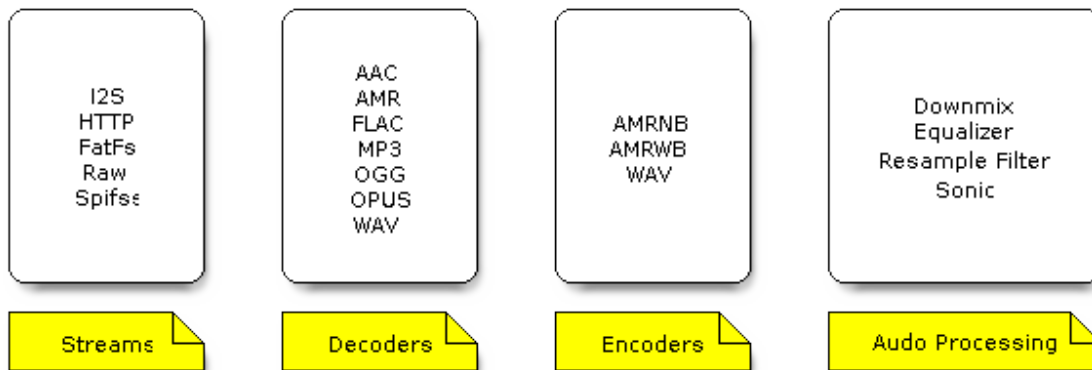


Fig. 1: **Elements** of the Audio Development Framework

The application is developed by combining the *Elements* into a *Pipeline*. A diagram below presents organization of two elements, MP3 decoder and I2S stream, in the Audio Pipeline, that has been used in [get-started/play\\_mp3\\_control](#) example.

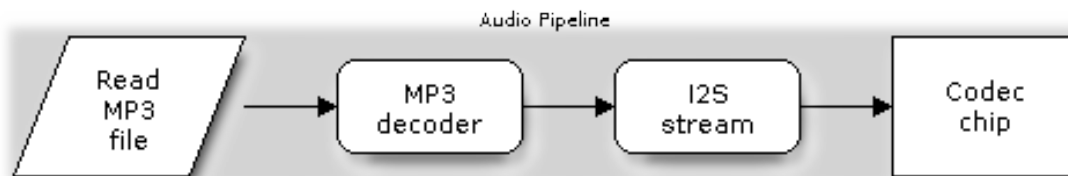


Fig. 2: Sample Organization of Elements in Audio Pipeline

The audio data is typically acquired using an input *Stream*, processed with *Codecs* and in some cases with *Audio Processing* functions, and finally output with another *Stream*. There is an *Event Interface* to facilitate communication of the application events. Interfacing with specific hardware is done using *Peripherals*.

See a table of contents below with links to description of all the above components.

## 2.1 Audio Framework

### 2.1.1 Audio Element

The basic building block for the application programmer developing with ADF is the `audio_element` object. Every decoder, encoder, filter, input stream, or output stream is in fact an Audio Element.

This API has been designed and then used to implement Audio Elements provided by ADF.

The general functionality of an Element is to take some data on input, processes it, and output to the next. Each Element is run as a separate task. To enable control on particular stages of the data lifecycle from the input, during processing and up to the output, the `audio_element` object provides possibility to trigger callbacks per stage. There are seven types of available callback functions: `open`, `seek`, `process`, `close`, `destroy`, `read` and `write`, and they are defined in `audio_element_cfg_t`. Particular Elements typically use a subset of all available callbacks. For instance the *MP3 Decoder* is using `open`, `process`, `close` and `destroy` callback functions.

The available Audio Element types intended for development with this API are listed in description of `audio_common.h` header file under `audio_element_type_t` enumerator.

### API Reference

#### Header File

- `components/audio_pipeline/include/audio_element.h`

#### Functions

`audio_element_handle_t` **audio\_element\_init** (`audio_element_cfg_t` \*config)

Initialize audio element with config.

**Parameters** `config` – The configuration

**Returns**

- `audio_element_handle_t` handle object
- `NULL`

`esp_err_t` **audio\_element\_deinit** (`audio_element_handle_t` el)

Destroy audio element handle object, stop, clear, delete all.

**Parameters** `el` – [in] The audio element handle

**Returns**

- `ESP_OK`
- `ESP_FAIL`

`esp_err_t` **audio\_element\_setdata** (`audio_element_handle_t` el, void \*data)

Set context data to element handle object. It can be retrieved by calling `audio_element_getdata`.

**Parameters**

- `el` – [in] The audio element handle
- `data` – The data pointer

**Returns**

- ESP\_OK
- ESP\_FAIL

void \***audio\_element\_getdata** (*audio\_element\_handle\_t* el)

Get context data from element handle object.

**Parameters** **e1** – [in] The audio element handle

**Returns** data pointer

esp\_err\_t **audio\_element\_set\_tag** (*audio\_element\_handle\_t* el, const char \*tag)

Set element tag name, or clear if tag = NULL.

**Parameters**

- **e1** – [in] The audio element handle
- **tag** – [in] The tag name pointer

**Returns**

- ESP\_OK
- ESP\_FAIL

char \***audio\_element\_get\_tag** (*audio\_element\_handle\_t* el)

Get element tag name.

**Parameters** **e1** – [in] The audio element handle

**Returns** Element tag name pointer

esp\_err\_t **audio\_element\_setinfo** (*audio\_element\_handle\_t* el, *audio\_element\_info\_t* \*info)

Set audio element information.

**Parameters**

- **e1** – [in] The audio element handle
- **info** – The information pointer

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **audio\_element\_getinfo** (*audio\_element\_handle\_t* el, *audio\_element\_info\_t* \*info)

Get audio element information.

**Parameters**

- **e1** – [in] The audio element handle
- **info** – The information pointer

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **audio\_element\_set\_uri** (*audio\_element\_handle\_t* el, const char \*uri)

Set audio element URI.

**Parameters**

- **e1** – [in] The audio element handle
- **uri** – [in] The uri pointer

### Returns

- ESP\_OK
- ESP\_FAIL

char \***audio\_element\_get\_uri** (*audio\_element\_handle\_t* el)

Get audio element URI.

**Parameters** **e1** – [in] The audio element handle

**Returns** URI pointer

esp\_err\_t **audio\_element\_run** (*audio\_element\_handle\_t* el)

Start Audio Element. With this function, audio\_element will start as freeRTOS task, and put the task into 'PAUSED' state. Note: Element does not actually start when this function returns.

**Parameters** **e1** – [in] The audio element handle

### Returns

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **audio\_element\_terminate** (*audio\_element\_handle\_t* el)

Terminate Audio Element. With this function, audio\_element will exit the task function. Note: this API only sends request. It does not actually terminate immediately when this function returns.

**Parameters** **e1** – [in] The audio element handle

### Returns

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **audio\_element\_terminate\_with\_ticks** (*audio\_element\_handle\_t* el, TickType\_t ticks\_to\_wait)

Terminate Audio Element with specific ticks for timeout. With this function, audio\_element will exit the task function. Note: this API only sends request. It does not actually terminate immediately when this function returns.

### Parameters

- **e1** – [in] The audio element handle
- **ticks\_to\_wait** – [in] The maximum amount of time to blocking

### Returns

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **audio\_element\_stop** (*audio\_element\_handle\_t* el)

Request stop of the Audio Element. After receiving the stop request, the element will ignore the actions being performed (read/write, wait for the ringbuffer ...) and close the task, reset the state variables. Note: this API only sends requests, Element does not actually stop when this function returns.

**Parameters** **e1** – [in] The audio element handle

### Returns

- ESP\_OK

- ESP\_FAIL

esp\_err\_t **audio\_element\_wait\_for\_stop** (*audio\_element\_handle\_t* el)

After the `audio_element_stop` function is called, the Element task will perform some abort procedures. This function will be blocked (Time is `DEFAULT_MAX_WAIT_TIME`) until Element Task has done and exit.

**Parameters** **e1** – [in] The audio element handle

**Returns**

- ESP\_OK, Success
- ESP\_FAIL, The state is not `AEL_STATE_RUNNING`
- ESP\_ERR\_TIMEOUT, Timeout

esp\_err\_t **audio\_element\_wait\_for\_stop\_ms** (*audio\_element\_handle\_t* el, TickType\_t ticks\_to\_wait)

After the `audio_element_stop` function is called, the Element task will perform some abort procedures. The maximum amount of time should block waiting for Element task has stopped.

**Parameters**

- **e1** – [in] The audio element handle
- **ticks\_to\_wait** – [in] The maximum amount of time to wait for stop

**Returns**

- ESP\_OK, Success
- ESP\_FAIL, The state is not `AEL_STATE_RUNNING`
- ESP\_ERR\_TIMEOUT, Timeout

esp\_err\_t **audio\_element\_pause** (*audio\_element\_handle\_t* el)

Request audio Element enter 'PAUSE' state. In this state, the task will wait for any event.

**Parameters** **e1** – [in] The audio element handle

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **audio\_element\_resume** (*audio\_element\_handle\_t* el, float wait\_for\_rb\_threshold, TickType\_t timeout)

Request audio Element enter 'RUNNING' state. In this state, the task listens to events and invokes the callback functions. At the same time it will wait until the `size/total_size` of the output ringbuffer is greater than or equal to `wait_for_rb_threshold`. If the timeout period has been exceeded and ringbuffer output has not yet reached `wait_for_rb_threshold` then the function will return.

**Parameters**

- **e1** – [in] The audio element handle
- **wait\_for\_rb\_threshold** – [in] The wait for rb threshold (0 .. 1)
- **timeout** – [in] The timeout

**Returns**

- ESP\_OK
- ESP\_FAIL

`esp_err_t audio_element_msg_set_listener` (*audio\_element\_handle\_t* el, *audio\_event\_iface\_handle\_t* listener)

This function will add a `listener` to listen to all events from audio element `el`. Any event from `el->external_event` will be send to the `listener`.

### Parameters

- `el` – The audio element handle
- `listener` – The event will be listen to

### Returns

- `ESP_OK`
- `ESP_FAIL`

`esp_err_t audio_element_set_event_callback` (*audio\_element\_handle\_t* el, *event\_cb\_func* cb\_func, void \*ctx)

This function will add a `callback` to be called from audio element `el`. Any event to caller will cause to call callback function.

### Parameters

- `el` – The audio element handle
- `cb_func` – The callback function
- `ctx` – Caller context

### Returns

- `ESP_OK`
- `ESP_FAIL`

`esp_err_t audio_element_msg_remove_listener` (*audio\_element\_handle\_t* el, *audio\_event\_iface\_handle\_t* listener)

Remove listener out of `el`. No new events will be sent to the listener.

### Parameters

- `el` – [in] The audio element handle
- `listener` – The listener

### Returns

- `ESP_OK`
- `ESP_FAIL`

`esp_err_t audio_element_set_input_ringbuf` (*audio\_element\_handle\_t* el, *ringbuf\_handle\_t* rb)

Set Element input ringbuffer.

### Parameters

- `el` – [in] The audio element handle
- `rb` – [in] The ringbuffer handle

### Returns

- `ESP_OK`
- `ESP_FAIL`



*ringbuf\_handle\_t* **audio\_element\_get\_input\_ringbuf** (*audio\_element\_handle\_t* el)

Get Element input ringbuffer.

**Parameters** **e1** – [in] The audio element handle

**Returns** *ringbuf\_handle\_t*

*esp\_err\_t* **audio\_element\_set\_output\_ringbuf** (*audio\_element\_handle\_t* el, *ringbuf\_handle\_t* rb)

Set Element output ringbuffer.

**Parameters**

- **e1** – [in] The audio element handle
- **rb** – [in] The ringbuffer handle

**Returns**

- ESP\_OK
- ESP\_FAIL

*ringbuf\_handle\_t* **audio\_element\_get\_output\_ringbuf** (*audio\_element\_handle\_t* el)

Get Element output ringbuffer.

**Parameters** **e1** – [in] The audio element handle

**Returns** *ringbuf\_handle\_t*

*audio\_element\_state\_t* **audio\_element\_get\_state** (*audio\_element\_handle\_t* el)

Get current Element state.

**Parameters** **e1** – [in] The audio element handle

**Returns** *audio\_element\_state\_t*

*esp\_err\_t* **audio\_element\_abort\_input\_ringbuf** (*audio\_element\_handle\_t* el)

If the element is requesting data from the input ringbuffer, this function forces it to abort.

**Parameters** **e1** – [in] The audio element handle

**Returns**

- ESP\_OK
- ESP\_FAIL

*esp\_err\_t* **audio\_element\_abort\_output\_ringbuf** (*audio\_element\_handle\_t* el)

If the element is waiting to write data to the ringbuffer output, this function forces it to abort.

**Parameters** **e1** – [in] The audio element handle

**Returns**

- ESP\_OK
- ESP\_FAIL

*esp\_err\_t* **audio\_element\_wait\_for\_buffer** (*audio\_element\_handle\_t* el, int size\_expect, TickType\_t timeout)

This function will wait until the sizeof the output ringbuffer is greater than or equal to *size\_expect*. If the timeout period has been exceeded and ringbuffer output has not yet reached *size\_expect* then the function will return *ESP\_FAIL*

**Parameters**

- **e1** – [in] The audio element handle
- **size\_expect** – [in] The size expect
- **timeout** – [in] The timeout

### Returns

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **audio\_element\_report\_status** (*audio\_element\_handle\_t* el, *audio\_element\_status\_t* status)

Element will sendout event (status) to event by this function.

### Parameters

- **e1** – [in] The audio element handle
- **status** – [in] The status

### Returns

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **audio\_element\_report\_info** (*audio\_element\_handle\_t* el)

Element will sendout event (information) to event by this function.

**Parameters** **e1** – [in] The audio element handle

### Returns

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **audio\_element\_report\_codec\_fmt** (*audio\_element\_handle\_t* el)

Element will sendout event (codec format) to event by this function.

**Parameters** **e1** – [in] The audio element handle

### Returns

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **audio\_element\_report\_pos** (*audio\_element\_handle\_t* el)

Element will sendout event with a duplicate information by this function.

**Parameters** **e1** – [in] The audio element handle

### Returns

- ESP\_OK
- ESP\_FAIL
- ESP\_ERR\_NO\_MEM

esp\_err\_t **audio\_element\_set\_input\_timeout** (*audio\_element\_handle\_t* el, TickType\_t timeout)

Set input read timeout (default is portMAX\_DELAY).

### Parameters

- **e1** – [in] The audio element handle

- **timeout** – [in] The timeout

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **audio\_element\_set\_output\_timeout** (*audio\_element\_handle\_t* el, TickType\_t timeout)

Set output read timeout (default is portMAX\_DELAY).

**Parameters**

- **e1** – [in] The audio element handle
- **timeout** – [in] The timeout

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **audio\_element\_reset\_input\_ringbuf** (*audio\_element\_handle\_t* el)

Reset inputbuffer.

**Parameters** **e1** – [in] The audio element handle

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **audio\_element\_finish\_state** (*audio\_element\_handle\_t* el)

Set element finish state.

**Parameters** **e1** – [in] The audio element handle

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **audio\_element\_change\_cmd** (*audio\_element\_handle\_t* el, *audio\_element\_msg\_cmd\_t* cmd)

Change element running state with specific command.

**Parameters**

- **e1** – [in] The audio element handle
- **cmd** – [in] Specific command from *audio\_element\_msg\_cmd\_t*

**Returns**

- ESP\_OK
- ESP\_FAIL
- ESP\_ERR\_INVALID\_ARG Element handle is null

esp\_err\_t **audio\_element\_reset\_output\_ringbuf** (*audio\_element\_handle\_t* el)

Reset outputbuffer.

**Parameters** **e1** – [in] The audio element handle

**Returns**

- ESP\_OK
- ESP\_FAIL

*audio\_element\_err\_t* **audio\_element\_input** (*audio\_element\_handle\_t* el, char \*buffer, int wanted\_size)

Call this function to provide Element input data. Depending on setup using ringbuffer or function callback, Element invokes read ringbuffer, or calls read callback function.

### Parameters

- **el** – [in] The audio element handle
- **buffer** – The buffer pointer
- **wanted\_size** – [in] The wanted size

### Returns

- > 0 number of bytes produced
- <=0 *audio\_element\_err\_t*

*audio\_element\_err\_t* **audio\_element\_output** (*audio\_element\_handle\_t* el, char \*buffer, int write\_size)

Call this function to sendout Element output data. Depending on setup using ringbuffer or function callback, Element will invoke write to ringbuffer, or call write callback function.

### Parameters

- **el** – [in] The audio element handle
- **buffer** – The buffer pointer
- **write\_size** – [in] The write size

### Returns

- > 0 number of bytes written
- <=0 *audio\_element\_err\_t*

*esp\_err\_t* **audio\_element\_set\_read\_cb** (*audio\_element\_handle\_t* el, *stream\_func* fn, void \*context)

This API allows the application to set a read callback for the first *audio\_element* in the pipeline for allowing the pipeline to interface with other systems. The callback is invoked every time the audio element requires data to be processed.

### Parameters

- **el** – [in] The audio element handle
- **fn** – [in] Callback read function. The callback function should return number of bytes read or -1 in case of error in reading. Note that the callback function may decide to block and that may block the entire pipeline.
- **context** – [in] An optional context which will be passed to callback function on every invocation

### Returns

- ESP\_OK
- ESP\_FAIL

*esp\_err\_t* **audio\_element\_set\_write\_cb** (*audio\_element\_handle\_t* el, *stream\_func* fn, void \*context)

This API allows the application to set a write callback for the last *audio\_element* in the pipeline for allowing the pipeline to interface with other systems. The callback is invoked every time the audio element has a processed data that needs to be passed forward.

**Parameters**

- **e1** – [in] The audio element
- **fn** – [in] Callback write function The callback function should return number of bytes written or -1 in case of error in writing. Note that the callback function may decide to block and that may block the entire pipeline.
- **context** – [in] An optional context which will be passed to callback function on every invocation

**Returns**

- ESP\_OK
- ESP\_FAIL

*stream\_func* **audio\_element\_get\_write\_cb** (*audio\_element\_handle\_t* el)

Get callback write function that register to the element.

**Parameters** **e1** – [in] The audio element

**Returns**

- Callback write function pointer
- NULL Failed

*stream\_func* **audio\_element\_get\_read\_cb** (*audio\_element\_handle\_t* el)

Get callback read function that register to the element.

**Parameters** **e1** – [in] The audio element

**Returns**

- Callback read function pointer
- NULL Failed

*QueueHandle\_t* **audio\_element\_get\_event\_queue** (*audio\_element\_handle\_t* el)

Get External queue of Emitter. We can read any event that has been send out of Element from this *QueueHandle\_t*.

**Parameters** **e1** – [in] The audio element handle

**Returns** *QueueHandle\_t*

*esp\_err\_t* **audio\_element\_set\_ringbuf\_done** (*audio\_element\_handle\_t* el)

Set inputbuffer and outputbuffer have finished.

**Parameters** **e1** – [in] The audio element handle

**Returns**

- ESP\_OK
- ESP\_FAIL

*esp\_err\_t* **audio\_element\_reset\_state** (*audio\_element\_handle\_t* el)

Enforce 'AEL\_STATE\_INIT' state.

**Parameters** **e1** – [in] The audio element handle

**Returns**

- ESP\_OK

- ESP\_FAIL

int **audio\_element\_get\_output\_ringbuf\_size** (*audio\_element\_handle\_t* el)

Get Element output ringbuffer size.

**Parameters** **e1** – [in] The audio element handle

**Returns**

- =0: Parameter NULL
- >0: Size of ringbuffer

esp\_err\_t **audio\_element\_set\_output\_ringbuf\_size** (*audio\_element\_handle\_t* el, int rb\_size)

Set Element output ringbuffer size.

**Parameters**

- **e1** – [in] The audio element handle
- **rb\_size** – [in] Size of the ringbuffer

**Returns**

- ESP\_OK
- ESP\_FAIL

int **audio\_element\_multi\_input** (*audio\_element\_handle\_t* el, char \*buffer, int wanted\_size, int index, TickType\_t ticks\_to\_wait)

Call this function to read data from multi input ringbuffer by given index.

**Parameters**

- **e1** – The audio element handle
- **buffer** – The buffer pointer
- **wanted\_size** – The wanted size
- **index** – The index of multi input ringbuffer, start from 0, should be less than NUMBER\_OF\_MULTI\_RINGBUF
- **ticks\_to\_wait** – Timeout of ringbuffer

**Returns**

- ESP\_ERR\_INVALID\_SIZE
- Others are same as the `rb_read`

int **audio\_element\_multi\_output** (*audio\_element\_handle\_t* el, char \*buffer, int wanted\_size, TickType\_t ticks\_to\_wait)

Call this function write data by multi output ringbuffer.

**Parameters**

- **e1** – [in] The audio element handle
- **buffer** – The buffer pointer
- **wanted\_size** – [in] The wanted size
- **ticks\_to\_wait** – Timeout of ringbuffer

**Returns**

- The return value is same as the `rb_write`

`esp_err_t audio_element_set_multi_input_ringbuf` (*audio\_element\_handle\_t* el, *ringbuf\_handle\_t* rb, int index)

Set multi input ringbuffer Element.

**Parameters**

- **e1** – [in] The audio element handle
- **rb** – [in] The ringbuffer handle
- **index** – [in] Index of multi ringbuffer, starts from 0, should be less than NUMBER\_OF\_MULTI\_RINGBUF

**Returns**

- ESP\_OK
- ESP\_FAIL

`esp_err_t audio_element_set_multi_output_ringbuf` (*audio\_element\_handle\_t* el, *ringbuf\_handle\_t* rb, int index)

Set multi output ringbuffer Element.

**Parameters**

- **e1** – [in] The audio element handle
- **rb** – [in] The ringbuffer handle
- **index** – [in] Index of multi ringbuffer, starts from 0, should be less than NUMBER\_OF\_MULTI\_RINGBUF

**Returns**

- ESP\_OK
- ESP\_ERR\_INVALID\_ARG

*ringbuf\_handle\_t* `audio_element_get_multi_input_ringbuf` (*audio\_element\_handle\_t* el, int index)

Get handle of multi input ringbuffer Element by index.

**Parameters**

- **e1** – [in] The audio element handle
- **index** – [in] Index of multi ringbuffer, starts from 0, should be less than NUMBER\_OF\_MULTI\_RINGBUF

**Returns**

- NULL Error
- Others *ringbuf\_handle\_t*

*ringbuf\_handle\_t* `audio_element_get_multi_output_ringbuf` (*audio\_element\_handle\_t* el, int index)

Get handle of multi output ringbuffer Element by index.

**Parameters**

- **e1** – [in] The audio element handle
- **index** – [in] Index of multi ringbuffer, starts from 0, should be less than NUMBER\_OF\_MULTI\_RINGBUF

**Returns**

- NULL Error

- Others `ringbuf_handle_t`

`esp_err_t audio_element_process_init` (*audio\_element\_handle\_t* el)

Provides a way to call element's `open`

**Parameters** `e1` – [in] The audio element handle

**Returns**

- `ESP_OK`
- `ESP_FAIL`

`esp_err_t audio_element_process_deinit` (*audio\_element\_handle\_t* el)

Provides a way to call element's `close`

**Parameters** `e1` – [in] The audio element handle

**Returns**

- `ESP_OK`
- `ESP_FAIL`

`esp_err_t audio_element_seek` (*audio\_element\_handle\_t* el, void \*in\_data, int in\_size, void \*out\_data, int \*out\_size)

Call element's `seek`

**Parameters**

- `e1` – [in] The audio element handle
- `in_data` – [in] A pointer to in data
- `in_size` – [in] The size of the `in_data`
- `out_data` – [out] A pointer to the out data
- `out_size` – [out] The size of the `out_data`

**Returns**

- `ESP_OK`
- `ESP_FAIL`
- `ESP_ERR_NOT_SUPPORTED`

`bool audio_element_is_stopping` (*audio\_element\_handle\_t* el)

Get Element stopping flag.

**Parameters** `e1` – [in] The audio element handle

**Returns** element's stopping flag

`esp_err_t audio_element_update_byte_pos` (*audio\_element\_handle\_t* el, int pos)

Update the byte position of element information.

**Parameters**

- `e1` – [in] The audio element handle
- `pos` – [in] The `byte_pos` accumulated by this value

**Returns**

- `ESP_OK`



- ESP\_FAIL

esp\_err\_t **audio\_element\_set\_byte\_pos** (*audio\_element\_handle\_t* el, int pos)

Set the byte position of element information.

**Parameters**

- **e1** – [in] The audio element handle
- **pos** – [in] This value is assigned to byte\_pos

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **audio\_element\_update\_total\_bytes** (*audio\_element\_handle\_t* el, int total\_bytes)

Update the total bytes of element information.

**Parameters**

- **e1** – [in] The audio element handle
- **total\_bytes** – [in] The total\_bytes accumulated by this value

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **audio\_element\_set\_total\_bytes** (*audio\_element\_handle\_t* el, int total\_bytes)

Set the total bytes of element information.

**Parameters**

- **e1** – [in] The audio element handle
- **total\_bytes** – [in] This value is assigned to total\_bytes

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **audio\_element\_set\_bps** (*audio\_element\_handle\_t* el, int bit\_rate)

Set the bps of element information.

**Parameters**

- **e1** – [in] The audio element handle
- **bit\_rate** – [in] This value is assigned to bps

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **audio\_element\_set\_codec\_fmt** (*audio\_element\_handle\_t* el, int format)

Set the codec format of element information.

**Parameters**

- **e1** – [in] The audio element handle

- **format** – [in] This value is assigned to `codec_fmt`

### Returns

- ESP\_OK
- ESP\_FAIL

`esp_err_t audio_element_set_music_info` (*audio\_element\_handle\_t* el, int sample\_rates, int channels, int bits)

Set the `sample_rate`, `channels`, `bits` of element information.

### Parameters

- **e1** – [in] The audio element handle
- **sample\_rates** – [in] `Sample_rates` of music information
- **channels** – [in] `Channels` of music information
- **bits** – [in] `Bits` of music information

### Returns

- ESP\_OK
- ESP\_FAIL

`esp_err_t audio_element_set_duration` (*audio\_element\_handle\_t* el, int duration)

Set the `duration` of element information.

### Parameters

- **e1** – [in] The audio element handle
- **duration** – [in] This value is assigned to `duration`

### Returns

- ESP\_OK
- ESP\_FAIL

`esp_err_t audio_element_set_reserve_user0` (*audio\_element\_handle\_t* el, int user\_data0)

Set the `user_data_0` of element information.

### Parameters

- **e1** – [in] The audio element handle
- **user\_data0** – [in] This value is assigned to `user_data_0`

### Returns

- ESP\_OK
- ESP\_FAIL

`esp_err_t audio_element_set_reserve_user1` (*audio\_element\_handle\_t* el, int user\_data1)

Set the `user_data_1` of element information.

### Parameters

- **e1** – [in] The audio element handle
- **user\_data1** – [in] This value is assigned to `user_data_1`

### Returns

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **audio\_element\_set\_reserve\_user2** (*audio\_element\_handle\_t* el, int user\_data2)

Set the user\_data\_2 of element information.

#### Parameters

- **e1** – [in] The audio element handle
- **user\_data2** – [in] This value is assigned to user\_data\_2

#### Returns

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **audio\_element\_set\_reserve\_user3** (*audio\_element\_handle\_t* el, int user\_data3)

Set the user\_data\_3 of element information.

#### Parameters

- **e1** – [in] The audio element handle
- **user\_data3** – [in] This value is assigned to user\_data\_3

#### Returns

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **audio\_element\_set\_reserve\_user4** (*audio\_element\_handle\_t* el, int user\_data4)

Set the user\_data\_4 of element information.

#### Parameters

- **e1** – [in] The audio element handle
- **user\_data4** – [in] This value is assigned to user\_data\_4

#### Returns

- ESP\_OK
- ESP\_FAIL

## Structures

struct **audio\_element\_reserve\_data\_t**

Audio Element user reserved data.

### Public Members

int **user\_data\_0**  
user data 0

int **user\_data\_1**  
user data 1

int **user\_data\_2**  
user data 2

int **user\_data\_3**  
user data 3

int **user\_data\_4**  
user data 4

struct **audio\_element\_info\_t**  
Audio Element informations.

### Public Members

int **sample\_rates**  
Sample rates in Hz

int **channels**  
Number of audio channel, mono is 1, stereo is 2

int **bits**  
Bit wide (8, 16, 24, 32 bits)

int **bps**  
Bit per second

int64\_t **byte\_pos**  
The current position (in bytes) being processed for an element

int64\_t **total\_bytes**  
The total bytes for an element

int **duration**  
The duration for an element (optional)

char \***uri**  
URI (optional)

`esp_codec_type_t` **codec\_fmt**

Music format (optional)

`audio_element_reserve_data_t` **reserve\_data**

This value is reserved for user use (optional)

struct **audio\_element\_cfg\_t**

Audio Element configurations. Each Element at startup will be a self-running task. These tasks will execute the callback open -> [loop: read -> process -> write] -> close. These callback functions are provided by the user corresponding to this configuration.

### Public Members

`el_io_func` **open**

Open callback function

`ctrl_func` **seek**

Seek callback function

`process_func` **process**

Process callback function

`el_io_func` **close**

Close callback function

`el_io_func` **destroy**

Destroy callback function

`stream_func` **read**

Read callback function

`stream_func` **write**

Write callback function

int **buffer\_len**

Buffer length use for an Element

int **task\_stack**

Element task stack

int **task\_prio**

Element task priority (based on freeRTOS priority)

int **task\_core**

Element task running in core (0 or 1)

`int out_rb_size`  
Output ringbuffer size

`void *data`  
User context

`const char *tag`  
Element tag

`bool stack_in_ext`  
Try to allocate stack in external memory

`int multi_in_rb_num`  
The number of multiple input ringbuffer

`int multi_out_rb_num`  
The number of multiple output ringbuffer

### Macros

`AUDIO_ELEMENT_INFO_DEFAULT()`

`DEFAULT_ELEMENT_RINGBUF_SIZE`

`DEFAULT_ELEMENT_BUFFER_LENGTH`

`DEFAULT_ELEMENT_STACK_SIZE`

`DEFAULT_ELEMENT_TASK_PRIO`

`DEFAULT_ELEMENT_TASK_CORE`

`DEFAULT_AUDIO_ELEMENT_CONFIG()`

### Type Definitions

`typedef struct audio_element *audio_element_handle_t`

`typedef esp_err_t (*el_io_func)(audio_element_handle_t self)`

`typedef audio_element_err_t (*process_func)(audio_element_handle_t self, char *el_buffer, int el_buf_len)`

```
typedef audio_element_err_t (*stream_func)(audio_element_handle_t self, char *buffer, int len, TickType_t ticks_to_wait, void *context)
```

```
typedef esp_err_t (*event_cb_func)(audio_element_handle_t el, audio_event_iface_msg_t *event, void *ctx)
```

```
typedef esp_err_t (*ctrl_func)(audio_element_handle_t self, void *in_data, int in_size, void *out_data, int *out_size)
```

## Enumerations

```
enum audio_element_err_t
```

*Values:*

enumerator **AEL\_IO\_OK**

enumerator **AEL\_IO\_FAIL**

enumerator **AEL\_IO\_DONE**

enumerator **AEL\_IO\_ABORT**

enumerator **AEL\_IO\_TIMEOUT**

enumerator **AEL\_PROCESS\_FAIL**

```
enum audio_element_state_t
```

Audio element state.

*Values:*

enumerator **AEL\_STATE\_NONE**

enumerator **AEL\_STATE\_INIT**

enumerator **AEL\_STATE\_INITIALIZING**

enumerator **AEL\_STATE\_RUNNING**

enumerator **AEL\_STATE\_PAUSED**

enumerator **AEL\_STATE\_STOPPED**

enumerator **AEL\_STATE\_FINISHED**

enumerator **AEL\_STATE\_ERROR**

enum **audio\_element\_msg\_cmd\_t**

Audio element action command, process on dispatcher

*Values:*

enumerator **AEL\_MSG\_CMD\_NONE**

enumerator **AEL\_MSG\_CMD\_FINISH**

enumerator **AEL\_MSG\_CMD\_STOP**

enumerator **AEL\_MSG\_CMD\_PAUSE**

enumerator **AEL\_MSG\_CMD\_RESUME**

enumerator **AEL\_MSG\_CMD\_DESTROY**

enumerator **AEL\_MSG\_CMD\_REPORT\_STATUS**

enumerator **AEL\_MSG\_CMD\_REPORT\_MUSIC\_INFO**

enumerator **AEL\_MSG\_CMD\_REPORT\_CODEC\_FMT**

enumerator **AEL\_MSG\_CMD\_REPORT\_POSITION**

enum **audio\_element\_status\_t**

Audio element status report

*Values:*

enumerator **AEL\_STATUS\_NONE**

enumerator **AEL\_STATUS\_ERROR\_OPEN**

enumerator **AEL\_STATUS\_ERROR\_INPUT**

enumerator **AEL\_STATUS\_ERROR\_PROCESS**

enumerator **AEL\_STATUS\_ERROR\_OUTPUT**

enumerator **AEL\_STATUS\_ERROR\_CLOSE**



enumerator **AEL\_STATUS\_ERROR\_TIMEOUT**

enumerator **AEL\_STATUS\_ERROR\_UNKNOWN**

enumerator **AEL\_STATUS\_INPUT\_DONE**

enumerator **AEL\_STATUS\_INPUT\_BUFFERING**

enumerator **AEL\_STATUS\_OUTPUT\_DONE**

enumerator **AEL\_STATUS\_OUTPUT\_BUFFERING**

enumerator **AEL\_STATUS\_STATE\_RUNNING**

enumerator **AEL\_STATUS\_STATE\_PAUSED**

enumerator **AEL\_STATUS\_STATE\_STOPPED**

enumerator **AEL\_STATUS\_STATE\_FINISHED**

enumerator **AEL\_STATUS\_MOUNTED**

enumerator **AEL\_STATUS\_UNMOUNTED**

## 2.1.2 Audio Pipeline

Dynamic combination of a group of linked *Elements* is done using the Audio Pipeline. You do not deal with the individual elements but with just one audio pipeline. Every element is connected by a ringbuffer. The Audio Pipeline also takes care of forwarding messages from the element tasks to an application.

A diagram below presents organization of three elements, HTTP reader stream, MP3 decoder and I2S writer stream, in the Audio Pipeline, that has been used in `player/pipeline_http_mp3` example.

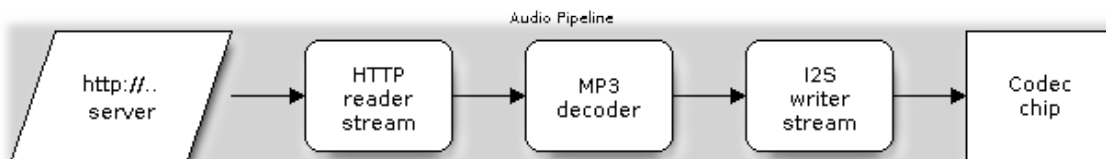


Fig. 3: Sample Organization of Elements in Audio Pipeline

### API Reference

#### Header File

- components/audio\_pipeline/include/audio\_pipeline.h

#### Functions

*audio\_pipeline\_handle\_t* **audio\_pipeline\_init** (*audio\_pipeline\_cfg\_t* \*config)

Initialize *audio\_pipeline\_handle\_t* object *audio\_pipeline* is responsible for controlling the audio data stream and connecting the audio elements with the ringbuffer. It will connect and start the audio element in order, responsible for retrieving the data from the previous element and passing it to the element after it. Also get events from each element, process events or pass it to a higher layer.

**Parameters** *config* – The configuration - *audio\_pipeline\_cfg\_t*

**Returns**

- *audio\_pipeline\_handle\_t* on success
- NULL when any errors

esp\_err\_t **audio\_pipeline\_deinit** (*audio\_pipeline\_handle\_t* pipeline)

This function removes all of the element's links in *audio\_pipeline*, cancels the registration of all events, invokes the destroy functions of the registered elements, and frees the memory allocated by the init function. Briefly, frees all memory.

**Parameters** *pipeline* – [in] The Audio Pipeline Handle

**Returns** ESP\_OK

esp\_err\_t **audio\_pipeline\_register** (*audio\_pipeline\_handle\_t* pipeline, *audio\_element\_handle\_t* el, const char \*name)

Registering an element for *audio\_pipeline*, each element can be registered multiple times, but *name* (as String) must be unique in *audio\_pipeline*, which is used to identify the element for link creation mentioned in the *audio\_pipeline\_link*

---

**Note:** Because of stop pipeline or pause pipeline depend much on register order. Please register element strictly in the following order: input element first, process middle, output element last.

---

**Parameters**

- **pipeline** – [in] The Audio Pipeline Handle
- **el** – [in] The Audio Element Handle
- **name** – [in] The name identifier of the *audio\_element* in this *audio\_pipeline*

**Returns**

- ESP\_OK on success
- ESP\_FAIL when any errors

esp\_err\_t **audio\_pipeline\_unregister** (*audio\_pipeline\_handle\_t* pipeline, *audio\_element\_handle\_t* el)

Unregister the audio\_element in audio\_pipeline, remove it from the list.

**Parameters**

- **pipeline** – [in] The Audio Pipeline Handle
- **el** – [in] The Audio Element Handle

**Returns**

- ESP\_OK on success
- ESP\_FAIL when any errors

esp\_err\_t **audio\_pipeline\_run** (*audio\_pipeline\_handle\_t* pipeline)

Start Audio Pipeline.

With this function audio\_pipeline will create tasks **for** all elements, that have been linked using the linking functions.

**Parameters** **pipeline** – [in] The Audio Pipeline Handle

**Returns**

- ESP\_OK on success
- ESP\_FAIL when any errors

esp\_err\_t **audio\_pipeline\_terminate** (*audio\_pipeline\_handle\_t* pipeline)

Stop Audio Pipeline.

With this function audio\_pipeline will destroy tasks of **all** elements, that have been linked using the linking functions.

**Parameters** **pipeline** – [in] The Audio Pipeline Handle

**Returns**

- ESP\_OK on success
- ESP\_FAIL when any errors

esp\_err\_t **audio\_pipeline\_terminate\_with\_ticks** (*audio\_pipeline\_handle\_t* pipeline, TickType\_t ticks\_to\_wait)

Stop Audio Pipeline with specific ticks for timeout.

With this function audio\_pipeline will destroy tasks of **all** elements, that have been linked using the linking functions.

**Parameters**

- **pipeline** – [in] The Audio Pipeline Handle
- **ticks\_to\_wait** – [in] The maximum amount of time to block wait for element destroy

### Returns

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **audio\_pipeline\_resume** (*audio\_pipeline\_handle\_t* pipeline)

This function will set all the elements to the RUNNING state and process the audio data as an inherent feature of audio\_pipeline.

**Parameters** **pipeline** – [in] The Audio Pipeline Handle

### Returns

- ESP\_OK on success
- ESP\_FAIL when any errors

esp\_err\_t **audio\_pipeline\_pause** (*audio\_pipeline\_handle\_t* pipeline)

This function will set all the elements to the PAUSED state. Everything remains the same except the data processing is stopped.

**Parameters** **pipeline** – [in] The Audio Pipeline Handle

### Returns

- ESP\_OK on success
- ESP\_FAIL when any errors

esp\_err\_t **audio\_pipeline\_stop** (*audio\_pipeline\_handle\_t* pipeline)

Stop all of the linked elements. Used with audio\_pipeline\_wait\_for\_stop to keep in sync. The link state of the elements in the pipeline is kept, events are still registered. The stopped audio\_pipeline restart by audio\_pipeline\_resume.

**Parameters** **pipeline** – [in] The Audio Pipeline Handle

### Returns

- ESP\_OK on success
- ESP\_FAIL when any errors

esp\_err\_t **audio\_pipeline\_wait\_for\_stop** (*audio\_pipeline\_handle\_t* pipeline)

The audio\_pipeline\_stop function sends requests to the elements and exits. But they need time to get rid of time-blocking tasks. This function will wait portMAX\_DELAY until all the Elements in the pipeline actually stop.

**Parameters** **pipeline** – [in] The Audio Pipeline Handle

### Returns

- ESP\_OK on success
- ESP\_FAIL when any errors

esp\_err\_t **audio\_pipeline\_wait\_for\_stop\_with\_ticks** (*audio\_pipeline\_handle\_t* pipeline, TickType\_t ticks\_to\_wait)

The audio\_pipeline\_stop function sends requests to the elements and exits. But they need time to get rid of time-blocking tasks. This function will wait ticks\_to\_wait until all the Elements in the pipeline actually stop.

### Parameters

- **pipeline** – [in] The Audio Pipeline Handle

- **ticks\_to\_wait** – [in] The maximum amount of time to block wait for stop

**Returns**

- ESP\_OK on success
- ESP\_FAIL when any errors

esp\_err\_t **audio\_pipeline\_link** (*audio\_pipeline\_handle\_t* pipeline, const char \*link\_tag[], int link\_num)

The audio\_element added to audio\_pipeline will be unconnected before it is called by this function. Based on element's name already registered by audio\_pipeline\_register, the path of the data will be linked in the order of the link\_tag. Element at index 0 is first, and index link\_num - 1 is final. As well as audio\_pipeline will subscribe all element's events.

**Parameters**

- **pipeline** – [in] The Audio Pipeline Handle
- **link\_tag** – Array of element name was registered by audio\_pipeline\_register
- **link\_num** – [in] Total number of elements of the link\_tag array

**Returns**

- ESP\_OK on success
- ESP\_FAIL when any errors

esp\_err\_t **audio\_pipeline\_unlink** (*audio\_pipeline\_handle\_t* pipeline)

Removes the connection of the elements, as well as unsubscribe events.

**Parameters** **pipeline** – [in] The Audio Pipeline Handle

**Returns**

- ESP\_OK on success
- ESP\_FAIL when any errors

*audio\_element\_handle\_t* **audio\_pipeline\_get\_el\_by\_tag** (*audio\_pipeline\_handle\_t* pipeline, const char \*tag)

Find un-kept element from registered pipeline by tag.

**Parameters**

- **pipeline** – [in] The Audio Pipeline Handle
- **tag** – [in] A char pointer

**Returns**

- NULL when any errors
- Others on success

*audio\_element\_handle\_t* **audio\_pipeline\_get\_el\_once** (*audio\_pipeline\_handle\_t* pipeline, const *audio\_element\_handle\_t* start\_el, const char \*tag)

Based on beginning element to find un-kept element from registered pipeline by tag.

**Parameters**

- **pipeline** – [in] The Audio Pipeline Handle
- **start\_el** – [in] Specific beginning element
- **tag** – [in] A char pointer

### Returns

- NULL when any errors
- Others on success

esp\_err\_t **audio\_pipeline\_remove\_listener** (*audio\_pipeline\_handle\_t* pipeline)

Remove event listener from this audio\_pipeline.

**Parameters** **pipeline** – [in] The Audio Pipeline Handle

### Returns

- ESP\_OK on success
- ESP\_FAIL when any errors

esp\_err\_t **audio\_pipeline\_set\_listener** (*audio\_pipeline\_handle\_t* pipeline, *audio\_event\_iface\_handle\_t* evt)

Set event listener for this audio\_pipeline, any event from this pipeline can be listened to by evt

### Parameters

- **pipeline** – [in] The Audio Pipeline Handle
- **evt** – [in] The Event Handle

### Returns

- ESP\_OK on success
- ESP\_FAIL when any errors

*audio\_event\_iface\_handle\_t* **audio\_pipeline\_get\_event\_iface** (*audio\_pipeline\_handle\_t* pipeline)

Get the event interface using by this pipeline.

**Parameters** **pipeline** – [in] The pipeline

**Returns** The Event Handle

esp\_err\_t **audio\_pipeline\_link\_insert** (*audio\_pipeline\_handle\_t* pipeline, bool first, *audio\_element\_handle\_t* prev, *ringbuf\_handle\_t* connect\_rb, *audio\_element\_handle\_t* next)

Insert the specific audio\_element to audio\_pipeline, previous element connects to the next element by ring buffer.

### Parameters

- **pipeline** – [in] The audio pipeline handle
- **first** – [in] Previous element is first input element, need to set true
- **prev** – [in] Previous element
- **connect\_rb** – [in] Connect ring buffer
- **next** – [in] Next element

### Returns

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **audio\_pipeline\_register\_more** (*audio\_pipeline\_handle\_t* pipeline, *audio\_element\_handle\_t* element\_1, ...)

Register a NULL-terminated list of elements to audio\_pipeline.

**Parameters**

- **pipeline** – [in] The audio pipeline handle
- **element\_1** – [in] The element to add to the audio\_pipeline.
- . . . – [in] Additional elements to add to the audio\_pipeline.

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **audio\_pipeline\_unregister\_more** (*audio\_pipeline\_handle\_t* pipeline, *audio\_element\_handle\_t* element\_1, ...)

Unregister a NULL-terminated list of elements to audio\_pipeline.

**Parameters**

- **pipeline** – [in] The audio pipeline handle
- **element\_1** – [in] The element to add to the audio\_pipeline.
- . . . – [in] Additional elements to add to the audio\_pipeline.

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **audio\_pipeline\_link\_more** (*audio\_pipeline\_handle\_t* pipeline, *audio\_element\_handle\_t* element\_1, ...)

Adds a NULL-terminated list of elements to audio\_pipeline.

**Parameters**

- **pipeline** – [in] The audio pipeline handle
- **element\_1** – [in] The element to add to the audio\_pipeline.
- . . . – [in] Additional elements to add to the audio\_pipeline.

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **audio\_pipeline\_listen\_more** (*audio\_pipeline\_handle\_t* pipeline, *audio\_element\_handle\_t* element\_1, ...)

Subscribe a NULL-terminated list of element's events to audio\_pipeline.

**Parameters**

- **pipeline** – [in] The audio pipeline handle
- **element\_1** – [in] The element event to subscribe to the audio\_pipeline.
- . . . – [in] Additional elements event to subscribe to the audio\_pipeline.

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **audio\_pipeline\_check\_items\_state** (*audio\_pipeline\_handle\_t* pipeline, *audio\_element\_handle\_t* dest\_el, *audio\_element\_status\_t* status)

Update the destination element state and check the all of linked elements state are same.

### Parameters

- **pipeline** – [in] The audio pipeline handle
- **dest\_el** – [in] Destination element
- **status** – [in] The new status

### Returns

- ESP\_OK All linked elements state are same.
- ESP\_FAIL All linked elements state are not same.

esp\_err\_t **audio\_pipeline\_reset\_items\_state** (*audio\_pipeline\_handle\_t* pipeline)

Reset pipeline element items state to AEL\_STATUS\_NONE

**Parameters** **pipeline** – [in] The Audio Pipeline Handle

### Returns

- ESP\_OK on success
- ESP\_FAIL when any errors

esp\_err\_t **audio\_pipeline\_reset\_ringbuffer** (*audio\_pipeline\_handle\_t* pipeline)

Reset pipeline element ringbuffer.

**Parameters** **pipeline** – [in] The Audio Pipeline Handle

### Returns

- ESP\_OK on success
- ESP\_FAIL when any errors

esp\_err\_t **audio\_pipeline\_reset\_elements** (*audio\_pipeline\_handle\_t* pipeline)

Reset Pipeline linked elements state.

**Parameters** **pipeline** – [in] The Audio Pipeline Handle

### Returns

- ESP\_OK on success
- ESP\_FAIL when any errors

esp\_err\_t **audio\_pipeline\_reset\_kept\_state** (*audio\_pipeline\_handle\_t* pipeline, *audio\_element\_handle\_t* el)

Reset the specific element kept state.

### Parameters

- **pipeline** – [in] The Audio Pipeline Handle
- **el** – [in] The Audio element Handle

### Returns

- ESP\_OK on success
- ESP\_FAIL when any errors



esp\_err\_t **audio\_pipeline\_breakup\_elements** (*audio\_pipeline\_handle\_t* pipeline, *audio\_element\_handle\_t* kept\_ctx\_el)

Break up all the linked elements of specific pipeline. The include and before kept\_ctx\_el working (AEL\_STATE\_RUNNING or AEL\_STATE\_PAUSED) elements and connected ringbuffer will be reserved.

---

**Note:** There is no element reserved when kept\_ctx\_el is NULL. This function will unsubscribe all element's events.

---

**Parameters**

- **pipeline** – [in] The audio pipeline handle
- **kept\_ctx\_el** – [in] Destination keep elements

**Returns**

- ESP\_OK All linked elements state are same.
- ESP\_ERR\_INVALID\_ARG Invalid parameters.

esp\_err\_t **audio\_pipeline\_relink** (*audio\_pipeline\_handle\_t* pipeline, const char \*link\_tag[], int link\_num)

Basing on element's name already registered by audio\_pipeline\_register, relink the pipeline following the order of names in the link\_tag.

---

**Note:** If the ringbuffer is not enough to connect the new pipeline will create new ringbuffer.

---

**Parameters**

- **pipeline** – [in] The Audio Pipeline Handle
- **link\_tag** – Array of elements name that was registered by audio\_pipeline\_register
- **link\_num** – [in] Total number of elements of the link\_tag array

**Returns**

- ESP\_OK All linked elements state are same.
- ESP\_FAIL Error.
- ESP\_ERR\_INVALID\_ARG Invalid parameters.

esp\_err\_t **audio\_pipeline\_relink\_more** (*audio\_pipeline\_handle\_t* pipeline, *audio\_element\_handle\_t* element\_1, ...)

Adds a NULL-terminated list of elements to audio\_pipeline.

---

**Note:** If the ringbuffer is not enough to connect the new pipeline will create new ringbuffer.

---

**Parameters**

- **pipeline** – [in] The Audio Pipeline Handle
- **element\_1** – [in] The element to add to the audio\_pipeline.
- ... – [in] Additional elements to add to the audio\_pipeline.

### Returns

- ESP\_OK All linked elements state are same.
- ESP\_FAIL Error.
- ESP\_ERR\_INVALID\_ARG Invalid parameters.

esp\_err\_t **audio\_pipeline\_change\_state** (*audio\_pipeline\_handle\_t* pipeline, *audio\_element\_state\_t* new\_state)

Set the pipeline state.

### Parameters

- **pipeline** – [in] The Audio Pipeline Handle
- **new\_state** – [in] The new state will be set

### Returns

- ESP\_OK All linked elements state are same.
- ESP\_FAIL Error.

## Structures

struct **audio\_pipeline\_cfg**  
Audio Pipeline configurations.

### Public Members

int **rb\_size**  
Audio Pipeline ringbuffer size

## Macros

**DEFAULT\_PIPELINE\_RINGBUF\_SIZE**  
**DEFAULT\_AUDIO\_PIPELINE\_CONFIG()**

## Type Definitions

typedef struct audio\_pipeline \***audio\_pipeline\_handle\_t**

typedef struct *audio\_pipeline\_cfg* **audio\_pipeline\_cfg\_t**  
Audio Pipeline configurations.

### 2.1.3 Event Interface

The ADF provides the Event Interface API to establish communication between Audio Elements in a pipeline. The API is built around FreeRTOS queue. It implements 'listeners' to watch for incoming messages and inform about them with a callback function.

#### Application Examples

Implementation of this API is demonstrated in couple of examples including `get-started/play_mp3_control`.

#### API Reference

##### Header File

- `components/audio_pipeline/include/audio_event_iface.h`

##### Functions

`audio_event_iface_handle_t` **audio\_event\_iface\_init** (`audio_event_iface_cfg_t` \*config)

Initialize audio event.

**Parameters** `config` – The configurations

**Returns**

- `ESP_OK`
- `ESP_FAIL`

`esp_err_t` **audio\_event\_iface\_destroy** (`audio_event_iface_handle_t` evt)

Cleanup event, it doesn't free evt pointer.

**Parameters** `evt` – The event

**Returns**

- `ESP_OK`
- `ESP_FAIL`

`esp_err_t` **audio\_event\_iface\_set\_listener** (`audio_event_iface_handle_t` evt, `audio_event_iface_handle_t` listener)

Add audio event `evt` to the listener, then we can listen `evt` event from `listen`

**Parameters**

- **listener** – The event can listen another event
- **evt** – The event to be added to

**Returns**

- `ESP_OK`
- `ESP_FAIL`

`esp_err_t audio_event_iface_remove_listener` (*audio\_event\_iface\_handle\_t* listener, *audio\_event\_iface\_handle\_t* evt)

Remove audio event `evt` from the listener.

### Parameters

- **listener** – The event listener
- **evt** – The event to be removed from

### Returns

- `ESP_OK`
- `ESP_FAIL`

`esp_err_t audio_event_iface_set_cmd_waiting_timeout` (*audio\_event\_iface\_handle\_t* evt, `TickType_t` wait\_time)

Set current queue wait time for the event.

### Parameters

- **evt** – The event
- **wait\_time** – [in] The wait time

### Returns

- `ESP_OK`
- `ESP_FAIL`

`esp_err_t audio_event_iface_waiting_cmd_msg` (*audio\_event\_iface\_handle\_t* evt)

Waiting internal queue message.

**Parameters** **evt** – The event

### Returns

- `ESP_OK`
- `ESP_FAIL`

`esp_err_t audio_event_iface_cmd` (*audio\_event\_iface\_handle\_t* evt, *audio\_event\_iface\_msg\_t* \*msg)

Trigger an event for internal queue with a message.

### Parameters

- **evt** – The event
- **msg** – The message

### Returns

- `ESP_OK`
- `ESP_FAIL`

`esp_err_t audio_event_iface_cmd_from_isr` (*audio\_event\_iface\_handle\_t* evt, *audio\_event\_iface\_msg\_t* \*msg)

It's same with `audio_event_iface_cmd`, but can send a message from ISR.

### Parameters

- **evt** – [in] The event
- **msg** – The message

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **audio\_event\_iface\_sendout** (*audio\_event\_iface\_handle\_t* evt, *audio\_event\_iface\_msg\_t* \*msg)

Trigger and event out with a message.

**Parameters**

- **evt** – The event
- **msg** – The message

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **audio\_event\_iface\_discard** (*audio\_event\_iface\_handle\_t* evt)

Discard all ongoing event message.

**Parameters** **evt** – The event

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **audio\_event\_iface\_listen** (*audio\_event\_iface\_handle\_t* evt, *audio\_event\_iface\_msg\_t* \*msg, TickType\_t wait\_time)

Listening and invoke callback function if there are any event are comming.

**Parameters**

- **evt** – The event
- **msg** – The message
- **wait\_time** – The wait time

**Returns**

- ESP\_OK
- ESP\_FAIL

QueueHandle\_t **audio\_event\_iface\_get\_queue\_handle** (*audio\_event\_iface\_handle\_t* evt)

Get External queue handle of Emmitter.

**Parameters** **evt** – [in] The external queue

**Returns** External QueueHandle\_t

esp\_err\_t **audio\_event\_iface\_read** (*audio\_event\_iface\_handle\_t* evt, *audio\_event\_iface\_msg\_t* \*msg, TickType\_t wait\_time)

Read the event from all the registered event emitters in the queue set of the interface.

**Parameters**

- **evt** – [in] The event interface
- **msg** – [out] The pointer to structure in which event is to be received
- **wait\_time** – [in] Timeout for receiving event

### Returns

- ESP\_OK On successful receiving of event
- ESP\_FAIL In case of a timeout or invalid parameter passed

QueueHandle\_t **audio\_event\_iface\_get\_msg\_queue\_handle** (*audio\_event\_iface\_handle\_t* evt)

Get Internal queue handle of Emmitter.

**Parameters** **evt** – [in] The Internal queue

**Returns** Internal QueueHandle\_t

esp\_err\_t **audio\_event\_iface\_set\_msg\_listener** (*audio\_event\_iface\_handle\_t* evt,  
*audio\_event\_iface\_handle\_t* listener)

Add audio internal event *evt* to the listener, then we can listen *evt* event from *listen*

### Parameters

- **listener** – The event can listen another event
- **evt** – The event to be added to

### Returns

- ESP\_OK
- ESP\_FAIL

## Structures

struct **audio\_event\_iface\_msg\_t**

Event message

### Public Members

int **cmd**

Command id

void \***data**

Data pointer

int **data\_len**

Data length

void \***source**

Source event

int **source\_type**

Source type (To know where it came from)

bool **need\_free\_data**

Need to free data pointer after the event has been processed

struct **audio\_event\_iface\_cfg\_t**

Event interface configurations

### Public Members

int **internal\_queue\_size**

It's optional, Queue size for event `internal_queue`

int **external\_queue\_size**

It's optional, Queue size for event `external_queue`

int **queue\_set\_size**

It's optional, QueueSet size for event `queue_set`

*on\_event\_iface\_func* **on\_cmd**

Function callback for listener when any event arrived

void \***context**

Context will pass to callback function

TickType\_t **wait\_time**

Timeout to check for event queue

int **type**

it will pass to *audio\_event\_iface\_msg\_t* `source_type` (To know where it came from)

### Macros

**DEFAULT\_AUDIO\_EVENT\_IFACE\_SIZE**

**AUDIO\_EVENT\_IFACE\_DEFAULT\_CFG** ()

### Type Definitions

typedef esp\_err\_t (\***on\_event\_iface\_func**)(*audio\_event\_iface\_msg\_t*\*, void\*)

typedef struct audio\_event\_iface \***audio\_event\_iface\_handle\_t**

### 2.1.4 Audio Common

Enumerations that define type of *Audio Elements*, type and format of *Codecs* and type of *Streams*.

#### API Reference

##### Header File

- components/audio\_pipeline/include/audio\_common.h

##### Macros

**ELEMENT\_SUB\_TYPE\_OFFSET**

**mem\_assert** (x)

##### Enumerations

enum **audio\_element\_type\_t**

*Values:*

enumerator **AUDIO\_ELEMENT\_TYPE\_UNKNOWN**

enumerator **AUDIO\_ELEMENT\_TYPE\_ELEMENT**

enumerator **AUDIO\_ELEMENT\_TYPE\_PLAYER**

enumerator **AUDIO\_ELEMENT\_TYPE\_SERVICE**

enumerator **AUDIO\_ELEMENT\_TYPE\_PERIPH**

enum **audio\_stream\_type\_t**

*Values:*

enumerator **AUDIO\_STREAM\_NONE**

enumerator **AUDIO\_STREAM\_READER**

enumerator **AUDIO\_STREAM\_WRITER**

enum **audio\_codec\_type\_t**

*Values:*



enumerator **AUDIO\_CODEC\_TYPE\_NONE**

enumerator **AUDIO\_CODEC\_TYPE\_DECODER**

enumerator **AUDIO\_CODEC\_TYPE\_ENCODER**

## 2.1.5 ESP Audio

This component provides several simple high level APIs. It is intended for quick implementation of audio applications based on typical interconnections of standardized audio elements.

### API Reference

#### Header File

- components/esp-adf-libs/esp\_audio/include/audio\_def.h

#### Structures

struct **esp\_audio\_state\_t**

esp\_audio status information parameters

#### Public Members

*esp\_audio\_status\_t* **status**

Status of esp\_audio

*audio\_err\_t* **err\_msg**

Status is AUDIO\_STATUS\_ERROR, err\_msg will be setup

*media\_source\_type\_t* **media\_src**

Media source type

#### Macros

**ESP\_ERR\_AUDIO\_BASE**

Starting number of ESP audio error codes

### Type Definitions

```
typedef void (*esp_audio_event_callback)(esp_audio_state_t *audio, void *ctx)
```

```
typedef esp_err_t (*audio_volume_set)(void *hd, int vol)
```

```
typedef esp_err_t (*audio_volume_get)(void *hd, int *vol)
```

### Enumerations

```
enum audio_err_t
```

*Values:*

```
enumerator ESP_ERR_AUDIO_NO_ERROR
```

```
enumerator ESP_ERR_AUDIO_FAIL
```

```
enumerator ESP_ERR_AUDIO_NO_INPUT_STREAM
```

```
enumerator ESP_ERR_AUDIO_NO_OUTPUT_STREAM
```

```
enumerator ESP_ERR_AUDIO_NO_CODEC
```

```
enumerator ESP_ERR_AUDIO_HAL_FAIL
```

```
enumerator ESP_ERR_AUDIO_MEMORY_LACK
```

```
enumerator ESP_ERR_AUDIO_INVALID_URI
```

```
enumerator ESP_ERR_AUDIO_INVALID_PATH
```

```
enumerator ESP_ERR_AUDIO_INVALID_PARAMETER
```

```
enumerator ESP_ERR_AUDIO_NOT_READY
```

```
enumerator ESP_ERR_AUDIO_NOT_SUPPORT
```

```
enumerator ESP_ERR_AUDIO_TIMEOUT
```

```
enumerator ESP_ERR_AUDIO_ALREADY_EXISTS
```

enumerator **ESP\_ERR\_AUDIO\_LINK\_FAIL**

enumerator **ESP\_ERR\_AUDIO\_UNKNOWN**

enumerator **ESP\_ERR\_AUDIO\_OUT\_OF\_RANGE**

enumerator **ESP\_ERR\_AUDIO\_STOP\_BY\_USER**

enumerator **ESP\_ERR\_AUDIO\_OPEN**

enumerator **ESP\_ERR\_AUDIO\_INPUT**

enumerator **ESP\_ERR\_AUDIO\_PROCESS**

enumerator **ESP\_ERR\_AUDIO\_OUTPUT**

enumerator **ESP\_ERR\_AUDIO\_CLOSE**

enum **esp\_audio\_status\_t**

*Values:*

enumerator **AUDIO\_STATUS\_UNKNOWN**

enumerator **AUDIO\_STATUS\_RUNNING**

enumerator **AUDIO\_STATUS\_PAUSED**

enumerator **AUDIO\_STATUS\_STOPPED**

enumerator **AUDIO\_STATUS\_FINISHED**

enumerator **AUDIO\_STATUS\_ERROR**

enum **audio\_termination\_type\_t**

*Values:*

enumerator **TERMINATION\_TYPE\_NOW**

Audio operation will be terminated immediately

enumerator **TERMINATION\_TYPE\_DONE**

Audio operation will be stopped when finished

enumerator **TERMINATION\_TYPE\_MAX**

enum **esp\_audio\_prefer\_t**

*Values:*

enumerator **ESP\_AUDIO\_PREFER\_MEM**

enumerator **ESP\_AUDIO\_PREFER\_SPEED**

enum **media\_source\_type\_t**

*Values:*

enumerator **MEDIA\_SRC\_TYPE\_NULL**

enumerator **MEDIA\_SRC\_TYPE\_MUSIC\_BASE**

enumerator **MEDIA\_SRC\_TYPE\_MUSIC\_SD**

enumerator **MEDIA\_SRC\_TYPE\_MUSIC\_HTTP**

enumerator **MEDIA\_SRC\_TYPE\_MUSIC\_FLASH**

enumerator **MEDIA\_SRC\_TYPE\_MUSIC\_A2DP**

enumerator **MEDIA\_SRC\_TYPE\_MUSIC\_DLNA**

enumerator **MEDIA\_SRC\_TYPE\_MUSIC\_RAW**

enumerator **MEDIA\_SRC\_TYPE\_MUSIC\_MAX**

enumerator **MEDIA\_SRC\_TYPE\_TONE\_BASE**

enumerator **MEDIA\_SRC\_TYPE\_TONE\_SD**

enumerator **MEDIA\_SRC\_TYPE\_TONE\_HTTP**

enumerator **MEDIA\_SRC\_TYPE\_TONE\_FLASH**

enumerator **MEDIA\_SRC\_TYPE\_TONE\_MAX**

enumerator **MEDIA\_SRC\_TYPE\_RESERVE\_BASE**

enumerator **MEDIA\_SRC\_TYPE\_RESERVE\_MAX**

## Header File

- components/esp-adf-libs/esp\_audio/include/esp\_audio.h

## Functions

*esp\_audio\_handle\_t* **esp\_audio\_create** (const *esp\_audio\_cfg\_t* \*cfg)

Create esp\_audio instance according to 'cfg' parameter.

This function create an esp\_audio instance, at the specified configuration.

**Parameters** **cfg** – [in] Provide esp\_audio initialization configuration

### Returns

- NULL: Error
- Others: esp\_audio instance fully certifying

*audio\_err\_t* **esp\_audio\_destroy** (*esp\_audio\_handle\_t* handle)

Specific esp\_audio instance will be destroyed.

**Parameters** **handle** – [in] The esp\_audio instance

### Returns

- ESP\_ERR\_AUDIO\_NO\_ERROR: on success
- ESP\_ERR\_AUDIO\_INVALID\_PARAMETER: no instance to free, call esp\_audio\_init first

*audio\_err\_t* **esp\_audio\_input\_stream\_add** (*esp\_audio\_handle\_t* handle, *audio\_element\_handle\_t* in\_stream)

Add audio input stream to specific esp\_audio instance.

### Parameters

- **handle** – [in] The esp\_audio instance
- **in\_stream** – [in] Audio stream instance

### Returns

- ESP\_ERR\_AUDIO\_NO\_ERROR: on success
- ESP\_ERR\_AUDIO\_INVALID\_PARAMETER: invalid arguments
- ESP\_ERR\_AUDIO\_MEMORY\_LACK: allocate memory fail

*audio\_err\_t* **esp\_audio\_output\_stream\_add** (*esp\_audio\_handle\_t* handle, *audio\_element\_handle\_t* out\_stream)

Add audio output stream to specific esp\_audio instance.

### Parameters

- **handle** – [in] The esp\_audio instance
- **out\_stream** – [in] The audio stream element instance

### Returns

- ESP\_ERR\_AUDIO\_NO\_ERROR: on success
- ESP\_ERR\_AUDIO\_INVALID\_PARAMETER: invalid arguments
- ESP\_ERR\_AUDIO\_MEMORY\_LACK: allocate memory fail

*audio\_err\_t* **esp\_audio\_codec\_lib\_add** (*esp\_audio\_handle\_t* handle, *audio\_codec\_type\_t* type, *audio\_element\_handle\_t* lib)

Add a new codec lib that can decode or encode a music file.

### Parameters

- **handle** – [in] The esp\_audio instance
- **type** – [in] The audio codec type(encoder or decoder)
- **lib** – [in] To provide audio stream element

### Returns

- ESP\_ERR\_AUDIO\_NO\_ERROR: on success
- ESP\_ERR\_AUDIO\_INVALID\_PARAMETER: invalid arguments
- ESP\_ERR\_AUDIO\_MEMORY\_LACK: allocate memory fail

*audio\_err\_t* **esp\_audio\_codec\_lib\_query** (*esp\_audio\_handle\_t* handle, *audio\_codec\_type\_t* type, const char \*extension)

Check if this kind of music extension is supported or not.

---

**Note:** This function just query the codec which has already add by esp\_audio\_codec\_lib\_add. The max length of extension is 6.

---

### Parameters

- **handle** – [in] The esp\_audio instance
- **type** – [in] The CODEC\_ENCODER or CODEC\_DECODER
- **extension** – [in] Such as “mp3”, “wav”, “aac”

### Returns

- ESP\_ERR\_AUDIO\_NO\_ERROR: supported
- ESP\_ERR\_AUDIO\_NOT\_SUPPORT: not support
- ESP\_ERR\_AUDIO\_INVALID\_PARAMETER: invalid arguments

*audio\_err\_t* **esp\_audio\_play** (*esp\_audio\_handle\_t* handle, *audio\_codec\_type\_t* type, const char \*uri, int pos)

Play the given uri.

The esp\_audio\_play have follow activity, setup inputstream, outputstream and codec by uri, start all of them. There is a rule that esp\_audio will select input stream, codec and output stream by URI field.

Rule of URI field are as follow.

- UF\_SCHEMA field of URI for choose input stream from existing streams. e.g:”http”,”file”
- UF\_PATH field of URI for choose codec from existing codecs. e.g:”/audio/mp3\_music.mp3”
- UF\_FRAGMENT field of URI for choose output stream from existing streams, output stream is I2S by default.
- UF\_USERINFO field of URI for specific sample rate and channels at encode mode.

The format “user:password” in the userinfo field, “user” is sample rate, “password” is channels.

Now esp\_audio\_play support follow URIs.

- ”https://dl.espressif.com/dl/audio/mp3\_music.mp3”

- "http://media-ice.musicradio.com/ClassicFMMP3"
- "file://sdcard/test.mp3"
- "iis://16000:2@from.pcm/rec.wav#file"
- "iis://16000:1@record.pcm/record.wav#raw"
- "aadp://44100:2@bt/sink/stream.pcm"
- "hfp://8000:1@bt/hfp/stream.pcm"

**Note:**

- The URI parse by `http_parser_parse_url`, any illegal string will be return `ESP_ERR_AUDIO_INVALID_URI`.
- If the `esp_decoder` codec is added to `handle`, then the `handle` of `esp_decoder` will be set as the default decoder, even if other decoders are added.
- Enabled `CONFIG_FATFS_API_ENCODING_UTF_8`, the URI can be support Chinese characters.
- Asynchronous interface
- The maximum of block time can be modify by `esp_audio_play_timeout_set`, default value is 25 seconds.

**Parameters**

- **handle** – The `esp_audio_handle_t` instance
- **uri** – Such as "file://sdcard/test.wav" or "http://iot.espressif.com/file/example.mp3". If NULL to be set, the uri setup by `esp_audio_setup` will used.
- **type** – Specific handle type decoder or encoder
- **pos** – Specific starting position by bytes

**Returns**

- `ESP_ERR_AUDIO_NO_ERROR`: on success
- `ESP_ERR_AUDIO_TIMEOUT`: timeout the play activity
- `ESP_ERR_AUDIO_NOT_SUPPORT`: Currently status is `AUDIO_STATUS_RUNNING`
- `ESP_ERR_AUDIO_INVALID_URI`: URI is illegal
- `ESP_ERR_AUDIO_INVALID_PARAMETER`: invalid arguments
- `ESP_ERR_AUDIO_STOP_BY_USER`: Exit without play due to `esp_audio_stop` has been called.

*audio\_err\_t* **esp\_audio\_sync\_play** (*esp\_audio\_handle\_t* handle, const char \*uri, int pos)

Play the given uri until music finished or error occurred.

**Note:**

- All features are same with `esp_audio_play`
- Synchronous interface
- Support decoder mode only

- No any events post during playing
- 

### Parameters

- **handle** – The `esp_audio_handle_t` instance
- **uri** – Such as “file://sdcard/test.wav” or “http://iot.espressif.com/file/example.mp3”,
- **pos** – Specific starting position by bytes

### Returns

- `ESP_ERR_AUDIO_NO_ERROR`: on success
- `ESP_ERR_AUDIO_TIMEOUT`: timeout the play activity
- `ESP_ERR_AUDIO_NOT_SUPPORT`: Currently status is `AUDIO_STATUS_RUNNING`
- `ESP_ERR_AUDIO_INVALID_URI`: URI is illegal
- `ESP_ERR_AUDIO_INVALID_PARAMETER`: invalid arguments

*audio\_err\_t* **esp\_audio\_stop** (*esp\_audio\_handle\_t* handle, *audio\_termination\_type\_t* type)

A synchronous interface for stop the `esp_audio`. The maximum of block time is 8000ms.

---

**Note:** 1. If user queue has been registered by `evt_que`, `AUDIO_STATUS_STOPPED` event for success or `AUDIO_STATUS_ERROR` event for error will be received.

- a. `TERMINATION_TYPE_DONE` only works with input stream which can't stopped by itself, e.g. raw read/write stream, others streams are no effect.
  - b. The synchronous interface is used to ensure that working pipeline is stopped.
- 

### Parameters

- **handle** – [in] The `esp_audio` instance
- **type** – [in] Stop immediately or done

### Returns

- `ESP_ERR_AUDIO_NO_ERROR`: on success
- `ESP_ERR_AUDIO_INVALID_PARAMETER`: invalid arguments
- `ESP_ERR_AUDIO_NOT_READY`: The status is not `AUDIO_STATUS_RUNNING` or `AUDIO_STATUS_PAUSED` or element has not created
- `ESP_ERR_AUDIO_TIMEOUT`: timeout(8000ms) the stop activity.

*audio\_err\_t* **esp\_audio\_pause** (*esp\_audio\_handle\_t* handle)

Pause the `esp_audio`.

---

**Note:** 1. Only support music and without live stream. If user queue has been registered by `evt_que`, `AUDIO_STATUS_PAUSED` event for success or `AUDIO_STATUS_ERROR` event for error will be received.

- a. The Paused music must be stoped by `esp_audio_stop` before new playing, otherwise got block on new play.
-



**Parameters** **handle** – [in] The esp\_audio instance

**Returns**

- ESP\_ERR\_AUDIO\_NO\_ERROR: on success
- ESP\_ERR\_AUDIO\_INVALID\_PARAMETER: invalid arguments
- ESP\_ERR\_AUDIO\_NOT\_READY: the status is not running
- ESP\_ERR\_AUDIO\_TIMEOUT: timeout the pause activity.

*audio\_err\_t* **esp\_audio\_resume** (*esp\_audio\_handle\_t* handle)

Resume the music paused.

---

**Note:** Only support music and without live stream. If user queue has been registered by evt\_que, AUDIO\_STATUS\_RUNNING event for success or AUDIO\_STATUS\_ERROR event for error will be received.

---

**Parameters** **handle** – [in] The esp\_audio instance

**Returns**

- ESP\_ERR\_AUDIO\_NO\_ERROR: on success
- ESP\_ERR\_AUDIO\_INVALID\_PARAMETER: invalid arguments
- ESP\_ERR\_AUDIO\_TIMEOUT: timeout the resume activity.

*audio\_err\_t* **esp\_audio\_eq\_gain\_set** (*esp\_audio\_handle\_t* handle, int band\_index, int nch, int eq\_gain)

Set the audio gain to be processed by the equalizer.

**Parameters**

- **handle** – [in] The esp\_audio instance
- **band\_index** – [in] The position of center frequencies of equalizer. The range of eq band index is [0 - 9].
- **nch** – [in] The number of channel. As for mono, the nch can only set to 1. As for dual, the nch can set to 1 and 2.
- **eq\_gain** – [in] The value of audio gain which in band\_index.

**Returns**

- ESP\_ERR\_AUDIO\_NO\_ERROR: on success
- ESP\_ERR\_AUDIO\_INVALID\_PARAMETER: invalid arguments

*audio\_err\_t* **esp\_audio\_eq\_gain\_get** (*esp\_audio\_handle\_t* handle, int band\_index, int nch, int \*eq\_gain)

Get the audio gain to be processed by the equalizer.

**Parameters**

- **handle** – [in] Audio element handle
- **band\_index** – [in] The position of center frequencies of equalizer. The range of eq band index is [0 - 9].
- **nch** – [in] The number of channel. As for mono, the nch can only set to 1. As for dual, the nch can set to 1 and 2.
- **eq\_gain** – [out] The pointer of the gain processed by equalizer

### Returns

- `ESP_ERR_AUDIO_NO_ERROR`: on success
- `ESP_ERR_AUDIO_INVALID_PARAMETER`: invalid arguments

*audio\_err\_t* **esp\_audio\_speed\_get** (*esp\_audio\_handle\_t* handle, *esp\_audio\_play\_speed\_t* \*speed\_index)

Getting esp\_audio play speed index, index value is from “esp\_audio\_speed\_t” enum.

### Parameters

- **handle** – [in] The esp\_audio instance
- **speed\_index** – [out] Current audio play speed index.

### Returns

- `ESP_ERR_AUDIO_NO_ERROR`: on success
- `ESP_ERR_AUDIO_INVALID_PARAMETER`: invalid arguments

*audio\_err\_t* **esp\_audio\_speed\_set** (*esp\_audio\_handle\_t* handle, *esp\_audio\_play\_speed\_t* speed\_index)

Use speed\_index which is from “esp\_audio\_speed\_t” enum to set esp\_audio play speed.

### Parameters

- **handle** – [in] The esp\_audio instance
- **speed\_index** – [in] Value from “esp\_audio\_speed\_t” enum.

### Returns

- `ESP_ERR_AUDIO_NO_ERROR`: on success
- `ESP_ERR_AUDIO_INVALID_PARAMETER`: invalid arguments

*audio\_err\_t* **esp\_audio\_speed\_idx\_to\_float** (*esp\_audio\_handle\_t* handle, *esp\_audio\_play\_speed\_t* speed\_index, float \*speed)

Use speed\_index which is from “esp\_audio\_speed\_t” enum to get esp\_audio play speed which is float type.

### Parameters

- **handle** – [in] The esp\_audio instance
- **speed\_index** – [in] Current audio play speed index.
- **speed** – [out] Current audio play speed.

### Returns

- `ESP_ERR_AUDIO_NO_ERROR`: on success
- `ESP_ERR_AUDIO_INVALID_PARAMETER`: invalid arguments

*audio\_err\_t* **esp\_audio\_vol\_set** (*esp\_audio\_handle\_t* handle, int vol)

Setting esp\_audio volume.

### Parameters

- **handle** – [in] The esp\_audio instance
- **vol** – [in] Specific volume will be set. 0-100 is legal. 0 will be mute.

### Returns

- `ESP_ERR_AUDIO_NO_ERROR`: on success
- `ESP_ERR_AUDIO_CTRL_HAL_FAIL`: error with hardware.

- ESP\_ERR\_AUDIO\_INVALID\_PARAMETER: invalid arguments

*audio\_err\_t* **esp\_audio\_vol\_get** (*esp\_audio\_handle\_t* handle, int \*vol)

Get esp\_audio volume.

#### Parameters

- **handle** – [in] The esp\_audio instance
- **vol** – [out] A pointer to int that indicates esp\_audio volume.

#### Returns

- ESP\_ERR\_AUDIO\_NO\_ERROR: on success
- ESP\_ERR\_AUDIO\_CTRL\_HAL\_FAIL: error with hardware.
- ESP\_ERR\_AUDIO\_INVALID\_PARAMETER: invalid arguments

*audio\_err\_t* **esp\_audio\_state\_get** (*esp\_audio\_handle\_t* handle, *esp\_audio\_state\_t* \*state)

Get esp\_audio status.

#### Parameters

- **handle** – [in] The esp\_audio instance
- **state** – [out] A pointer to *esp\_audio\_state\_t* that indicates esp\_audio status.

#### Returns

- ESP\_ERR\_AUDIO\_NO\_ERROR: on success
- ESP\_ERR\_AUDIO\_INVALID\_PARAMETER: no esp\_audio instance or esp\_audio does not playing

*audio\_err\_t* **esp\_audio\_pos\_get** (*esp\_audio\_handle\_t* handle, int \*pos)

Get the position in bytes of currently played music.

---

**Note:** This function works only with decoding music.

---

#### Parameters

- **handle** – [in] The esp\_audio instance
- **pos** – [out] A pointer to int that indicates esp\_audio decoding position.

#### Returns

- ESP\_ERR\_AUDIO\_NO\_ERROR: on success
- ESP\_ERR\_AUDIO\_INVALID\_PARAMETER: no esp\_audio instance
- ESP\_ERR\_AUDIO\_NOT\_READY: no codec element

*audio\_err\_t* **esp\_audio\_time\_get** (*esp\_audio\_handle\_t* handle, int \*time)

Get the position in microseconds of currently played music.

---

**Note:** This function works only with decoding music.

---

#### Parameters

- **handle** – [in] The esp\_audio instance
- **time** – [out] A pointer to int that indicates esp\_audio decoding position.

### Returns

- ESP\_ERR\_AUDIO\_NO\_ERROR: on success
- ESP\_ERR\_AUDIO\_INVALID\_PARAMETER: no esp\_audio instance
- ESP\_ERR\_AUDIO\_NOT\_READY: no out stream

*audio\_err\_t* **esp\_audio\_setup** (*esp\_audio\_handle\_t* handle, *esp\_audio\_setup\_t* \*sets)

Choose the in\_stream, codec and out\_stream definitely, and set uri.

---

**Note:** This function provide a manual way to select in/out stream and codec, should be called before the esp\_audio\_play, then ignore the esp\_audio\_play URI parameter only one time.

---

### Parameters

- **handle** – [in] The esp\_audio instance
- **sets** – [in] A pointer to *esp\_audio\_setup\_t*.

### Returns

- ESP\_ERR\_AUDIO\_NO\_ERROR: on success
- ESP\_ERR\_AUDIO\_INVALID\_PARAMETER: no esp\_audio instance
- ESP\_ERR\_AUDIO\_MEMORY\_LACK: allocate memory fail

*audio\_err\_t* **esp\_audio\_media\_type\_set** (*esp\_audio\_handle\_t* handle, *media\_source\_type\_t* type)

*audio\_err\_t* **esp\_audio\_music\_info\_get** (*esp\_audio\_handle\_t* handle, *esp\_audio\_music\_info\_t* \*info)

*audio\_err\_t* **esp\_audio\_info\_get** (*esp\_audio\_handle\_t* handle, *esp\_audio\_info\_t* \*info)

*audio\_err\_t* **esp\_audio\_info\_set** (*esp\_audio\_handle\_t* handle, *esp\_audio\_info\_t* \*info)

*audio\_err\_t* **esp\_audio\_callback\_set** (*esp\_audio\_handle\_t* handle, *esp\_audio\_event\_callback* cb, void \*cb\_ctx)

*audio\_err\_t* **esp\_audio\_seek** (*esp\_audio\_handle\_t* handle, int seek\_time\_sec)

Seek the position in second of currently played music.

---

**Note:** This function works only with decoding music.

---

### Parameters

- **handle** – [in] The esp\_audio instance
- **seek\_time\_sec** – [out] A pointer to int that indicates esp\_audio decoding position.

### Returns

- ESP\_ERR\_AUDIO\_NO\_ERROR: on success
- ESP\_ERR\_AUDIO\_FAIL: codec or allocation fail

- `ESP_ERR_AUDIO_TIMEOUT`: timeout for sync the element status
- `ESP_ERR_AUDIO_INVALID_PARAMETER`: no `esp_audio` instance
- `ESP_ERR_AUDIO_NOT_SUPPORT`: codec has finished
- `ESP_ERR_AUDIO_OUT_OF_RANGE`: the `seek_time_ms` is out of the range
- `ESP_ERR_AUDIO_NOT_READY`: the status is neither running nor paused

*audio\_err\_t* **esp\_audio\_duration\_get** (*esp\_audio\_handle\_t* handle, int \*duration)

Get the duration in microseconds of playing music.

---

**Note:** This function works only with decoding music.

---

#### Parameters

- **handle** – [in] The `esp_audio` instance
- **duration** – [out] A pointer to int that indicates decoding total time.

#### Returns

- `ESP_ERR_AUDIO_NO_ERROR`: on success
- `ESP_ERR_AUDIO_INVALID_PARAMETER`: no `esp_audio` instance
- `ESP_ERR_AUDIO_NOT_READY`: no codec element or no in element

*audio\_err\_t* **esp\_audio\_play\_timeout\_set** (*esp\_audio\_handle\_t* handle, int time\_ms)

Setting the maximum amount of time to waiting for `esp_audio_play` only.

#### Parameters

- **handle** – [in] The `esp_audio` instance
- **time\_ms** – [in] The maximum amount of time

#### Returns

- `ESP_ERR_AUDIO_NO_ERROR`: on success
- `ESP_ERR_AUDIO_INVALID_PARAMETER`: invalid arguments

*audio\_err\_t* **esp\_audio\_prefer\_type\_get** (*esp\_audio\_handle\_t* handle, *esp\_audio\_prefer\_t* \*type)

Get the type of `esp_audio_prefer_t`

#### Parameters

- **handle** – [in] The `esp_audio` instance
- **type** – [out] A pointer to `esp_audio_prefer_t`

#### Returns

- `ESP_ERR_AUDIO_NO_ERROR`: on success
- `ESP_ERR_AUDIO_INVALID_PARAMETER`: no `esp_audio` instance

*audio\_err\_t* **esp\_audio\_event\_que\_set** (*esp\_audio\_handle\_t* handle, *QueueHandle\_t* que)

Set event queue to notify the `esp_audio` status.

#### Parameters

- **handle** – [in] The esp\_audio instance
- **que** – [out] A pointer to QueueHandle\_t

### Returns

- ESP\_ERR\_AUDIO\_NO\_ERROR: on success
- ESP\_ERR\_AUDIO\_INVALID\_PARAMETER: no esp\_audio instance

## Structures

struct **esp\_audio\_cfg\_t**  
esp\_audio configuration parameters

### Public Members

int **in\_stream\_buf\_size**  
Input buffer size

int **out\_stream\_buf\_size**  
Output buffer size

int **resample\_rate**  
Destination sample rate, 0: disable resample; others: 44.1K, 48K, 32K, 16K, 8K has supported It should be make sure same with I2S stream `sample_rate`

int **component\_select**  
The select of audio forge component. eg. To choose equalizer and ALC together, please enter ESP\_AUDIO\_COMPONENT\_SELECT\_ALC | ESP\_AUDIO\_COMPONENT\_SELECT\_EQUALIZER.

QueueHandle\_t **evt\_que**  
For received esp\_audio events (optional)

*esp\_audio\_event\_callback* **cb\_func**  
esp\_audio events callback (optional)

void \***cb\_ctx**  
esp\_audio callback context (optional)

*esp\_audio\_prefer\_t* **prefer\_type**  
esp\_audio works on sepcific type, default memory is preferred.

- ESP\_AUDIO\_PREFER\_MEM mode stopped the previous linked elements before the new pipeline starting, except out stream element.
- ESP\_AUDIO\_PREFER\_SPEED mode kept the previous linked elements before the new pipeline starting, except out stream element.

void **\*vol\_handle**  
Volume change instance

*audio\_volume\_set* **vol\_set**  
Set volume callback

*audio\_volume\_get* **vol\_get**  
Get volume callback

int **task\_prio**  
esp\_audio task priority

int **task\_stack**  
Size of esp\_audio task stack

struct **esp\_audio\_setup\_t**  
esp\_audio setup parameters by manual

### Public Members

*audio\_codec\_type\_t* **set\_type**  
Set codec type

int **set\_sample\_rate**  
Set music sample rate

int **set\_channel**  
Set music channels

int **set\_pos**  
Set starting position

int **set\_time**  
Set starting position of the microseconds time (optional)

char **\*set\_uri**  
Set URI

char **\*set\_in\_stream**  
Tag of in\_stream

char **\*set\_codec**  
Tag of the codec

char \***set\_out\_stream**

Tag of out\_stream

struct **esp\_audio\_info\_t**

esp\_audio information

### Public Members

*audio\_element\_info\_t* **codec\_info**

Codec information

*audio\_element\_handle\_t* **in\_el**

Handle of the in stream

*audio\_element\_handle\_t* **out\_el**

Handle of the out stream

*audio\_element\_handle\_t* **codec\_el**

Handle of the codec

*audio\_element\_handle\_t* **filter\_el**

Handle of the filter

*esp\_audio\_state\_t* **st**

The state of esp\_audio

int **time\_pos**

Position of the microseconds time

float **audio\_speed**

Play speed of audio

int64\_t **in\_stream\_total\_size**

Total size of in stream

struct **esp\_audio\_music\_info\_t**

The music informations.



## Public Members

int **sample\_rates**

Sample rates in Hz

int **channels**

Number of audio channel, mono is 1, stereo is 2

int **bits**

Bit wide (8, 16, 24, 32 bits)

int **bps**

Bit per second

esp\_codec\_type\_t **codec\_fmt**

Music format

## Macros

**ESP\_AUDIO\_COMPONENT\_SELECT\_DEFAULT**

Default selected

**ESP\_AUDIO\_COMPONENT\_SELECT\_ALC**

ALC selected

**ESP\_AUDIO\_COMPONENT\_SELECT\_EQUALIZER**

Equalizer selected

**DEFAULT\_ESP\_AUDIO\_CONFIG()**

## Type Definitions

typedef void \***esp\_audio\_handle\_t**

## Enumerations

enum **esp\_audio\_play\_speed\_t**

esp\_audio play speed

*Values:*

enumerator **ESP\_AUDIO\_PLAY\_SPEED\_UNKNOW**

enumerator `ESP_AUDIO_PLAY_SPEED_0_50`

enumerator `ESP_AUDIO_PLAY_SPEED_0_75`

enumerator `ESP_AUDIO_PLAY_SPEED_1_00`

enumerator `ESP_AUDIO_PLAY_SPEED_1_25`

enumerator `ESP_AUDIO_PLAY_SPEED_1_50`

enumerator `ESP_AUDIO_PLAY_SPEED_1_75`

enumerator `ESP_AUDIO_PLAY_SPEED_2_00`

enumerator `ESP_AUDIO_PLAY_SPEED_MAX`

## 2.2 Audio Streams

The Audio Stream refers to an *Audio Element* that is responsible for acquiring of audio data and then sending the data out after processing.

The following stream types are supported:

- *Algorithm Stream*
- *FatFs Stream*
- *HTTP Stream*
- *I2S Stream*
- *PWM Stream*
- *Raw Stream*
- *SPIFFS Stream*
- *TCP Client Stream*
- *Tone Stream*
- *Flash-Embedding Stream*
- *TTS Stream*

Each stream is initialized with a structure as an input, and the returned `audio_element_handle_t` handle is used to call the functions in `audio_element.h`. Most streams have two types, `AUDIO_STREAM_READER` (reader) and `AUDIO_STREAM_WRITER` (writer). For example, to set the I2S stream type, use `i2s_stream_init()` and `i2s_stream_cfg_t`.

See description below for the API details.

## 2.2.1 Algorithm Stream

The algorithm stream integrates front-end algorithms such as acoustic echo cancellation (AEC), automatic gain control (AGC), and noise suppression (NS) to process the received audio. It is often used in audio preprocessing scenarios, including VoIP, speech recognition, and keyword wake-up. The stream calls `esp-sr` and thus occupies large memory. The stream only supports the `AUDIO_STREAM_READER` type.

### Application Example

- `advanced_examples/algorithm/main/algorithm_examples.c`
- `protocols/voip/main/voip_app.c`

### Header File

- `components/audio_stream/include/algorithm_stream.h`

### Functions

*audio\_element\_handle\_t* **algo\_stream\_init** (*algorithm\_stream\_cfg\_t* \*config)

Initialize algorithm stream.

**Parameters** `config` – The algorithm Stream configuration

**Returns** The audio element handle

*audio\_element\_err\_t* **algo\_stream\_set\_delay** (*audio\_element\_handle\_t* el, *ringbuf\_handle\_t* ringbuf, int delay\_ms)

Set playback signal or recording signal delay when use type2.

---

**Note:** The AEC internal buffering mechanism requires that the recording signal is delayed by around 0 - 10 ms compared to the corresponding reference (playback) signal.

---

#### Parameters

- `el` – Handle of element
- `ringbuf` – Handle of ringbuf
- `delay_ms` – The delay between playback and recording in ms This `delay_ms` can be debugged by yourself, you can set the configuration `debug_input` to true, then get the original input data (left channel is the signal captured from the microphone, right channel is the signal played to the speaker), and check the delay with an audio analysis tool.

#### Returns

- `ESP_OK`
- `ESP_FAIL`
- `ESP_ERR_INVALID_ARG`

`esp_err_t algorithm_mono_fix` (`uint8_t *sbuff`, `uint32_t len`)

Fix I2S mono noise issue.

---

**Note:** This API only for ESP32 with I2S 16bits

---

### Parameters

- **sbuff** – I2S data buffer
- **len** – I2S data len

**Returns** ESP\_OK

## Structures

struct **algorithm\_stream\_cfg\_t**

Algorithm stream configurations.

### Public Members

*algorithm\_stream\_input\_type\_t* **input\_type**

Input type of stream

int **task\_stack**

Task stack size

int **task\_prio**

Task peroid

int **task\_core**

The core that task to be created

int **out\_rb\_size**

Size of output ringbuffer

bool **stack\_in\_ext**

Try to allocate stack in external memory

int **rec\_linear\_factor**

The linear amplication factor of record signal

int **ref\_linear\_factor**

The linear amplication factor of reference signal

bool **debug\_input**

debug algorithm input data

bool **swap\_ch**  
Swap left and right channels

int8\_t **algo\_mask**  
Choose algorithm to use

int **sample\_rate**  
The sampling rate of the input PCM (in Hz)

int **mic\_ch**  
MIC channel num

int **agc\_gain**  
AGC gain(dB) for voice communication

bool **aec\_low\_cost**  
AEC uses less cpu and ram resources, but has poor suppression of nonlinear distortion

char \***partition\_label**  
Partition label which stored the model data

## Macros

**ALGORITHM\_STREAM\_PINNED\_TO\_CORE**

**ALGORITHM\_STREAM\_TASK\_PERIOD**

**ALGORITHM\_STREAM\_RINGBUFFER\_SIZE**

**ALGORITHM\_STREAM\_TASK\_STACK\_SIZE**

**ALGORITHM\_STREAM\_DEFAULT\_SAMPLE\_RATE\_HZ**

**ALGORITHM\_STREAM\_DEFAULT\_SAMPLE\_BIT**

**ALGORITHM\_STREAM\_DEFAULT\_MIC\_CHANNELS**

**ALGORITHM\_STREAM\_DEFAULT\_AGC\_GAIN\_DB**

**ALGORITHM\_STREAM\_DEFAULT\_MASK**

**ALGORITHM\_STREAM\_CFG\_DEFAULT()**

### Enumerations

enum **algorithm\_stream\_input\_type\_t**

Two types of algorithm stream input method.

*Values:*

enumerator **ALGORITHM\_STREAM\_INPUT\_TYPE1**

Type 1 is default used by mini-board, the reference signal and the recording signal are respectively read in from the left channel and the right channel of the same I2S

enumerator **ALGORITHM\_STREAM\_INPUT\_TYPE2**

As the simple diagram above shows, when type2 is chosen, the recording signal and reference signal should be input by users.

enum **algorithm\_stream\_mask\_t**

Choose the algorithm to be used.

*Values:*

enumerator **ALGORITHM\_STREAM\_USE\_AEC**

Use AEC

enumerator **ALGORITHM\_STREAM\_USE\_AGC**

Use AGC

enumerator **ALGORITHM\_STREAM\_USE\_NS**

Use NS

enumerator **ALGORITHM\_STREAM\_USE\_VAD**

Use VAD

### 2.2.2 FatFs Stream

The FatFs stream reads and writes data from FatFs. It has two types: “reader” and “writer”. The type is defined by *audio\_stream\_type\_t*.

#### Application Example

- Reader example: `player/pipeline_play_sdcard_music`
- Writer example: `recorder/pipeline_recording_to_sdcard`

## Header File

- components/audio\_stream/include/fatfs\_stream.h

## Functions

*audio\_element\_handle\_t* **fatfs\_stream\_init** (*fatfs\_stream\_cfg\_t* \*config)

Create a handle to an Audio Element to stream data from FatFs to another Element or get data from other elements written to FatFs, depending on the configuration the stream type, either AUDIO\_STREAM\_READER or AUDIO\_STREAM\_WRITER.

**Parameters** **config** – The configuration

**Returns** The Audio Element handle

## Structures

struct **fatfs\_stream\_cfg\_t**

FATFS Stream configurations, if any entry is zero then the configuration will be set to default values.

### Public Members

*audio\_stream\_type\_t* **type**

Stream type

int **buf\_sz**

Audio Element Buffer size

int **out\_rb\_size**

Size of output ringbuffer

int **task\_stack**

Task stack size

int **task\_core**

Task running in core (0 or 1)

int **task\_prio**

Task priority (based on freeRTOS priority)

bool **ext\_stack**

Allocate stack on extern ram

bool **write\_header**

Choose to write amrnb/amrwb header in fatfs whether or not (true or false, true means choose to write amrnb header)

### Macros

`FATFS_STREAM_BUF_SIZE`

`FATFS_STREAM_TASK_STACK`

`FATFS_STREAM_TASK_CORE`

`FATFS_STREAM_TASK_PRIO`

`FATFS_STREAM_RINGBUFFER_SIZE`

`FATFS_STREAM_CFG_DEFAULT()`

### 2.2.3 HTTP Stream

The HTTP stream obtains and sends data through `esp_http_client()`. The stream has two types: “reader” and “writer”, and the type is defined by `audio_stream_type_t`. `AUDIO_STREAM_READER` supports HTTP, HTTPS, HTTP Live Stream, and other protocols. Make sure the network is connected before using the stream.

#### Application Example

- Reader example
  - `player/pipeline_living_stream`
  - `player/pipeline_http_mp3`
- Writer example
  - `recorder/pipeline_raw_http`

#### Header File

- `components/audio_stream/include/http_stream.h`

#### Functions

`audio_element_handle_t http_stream_init (http_stream_cfg_t *config)`

Create a handle to an Audio Element to stream data from HTTP to another Element or get data from other elements sent to HTTP, depending on the configuration the stream type, either `AUDIO_STREAM_READER` or `AUDIO_STREAM_WRITER`.

**Parameters** `config` – The configuration

**Returns** The Audio Element handle



`esp_err_t http_stream_next_track (audio_element_handle_t el)`

Connect to next track in the playlist.

This function can be used in event\_handler of http\_stream.  
User can call this function to connect to next track in playlist when he/  
→she gets `HTTP\_STREAM\_FINISH\_TRACK` event

**Parameters** `e1` – The http\_stream element handle

**Returns**

- ESP\_OK on success
- ESP\_FAIL on errors

`esp_err_t http_stream_restart (audio_element_handle_t el)`

`esp_err_t http_stream_fetch_again (audio_element_handle_t el)`

Try to fetch the tracks again.

If this **is** live stream we will need to keep fetching URIs.

**Parameters** `e1` – The http\_stream element handle

**Returns**

- ESP\_OK on success
- ESP\_ERR\_NOT\_SUPPORTED if playlist is finished

`esp_err_t http_stream_set_server_cert (audio_element_handle_t el, const char *cert)`

Set SSL server certification.

---

**Note:** EM format as string, if the client requires to verify server

---

**Parameters**

- `e1` – The http\_stream element handle
- `cert` – server certification

**Returns**

- ESP\_OK on success

### Structures

struct **http\_stream\_event\_msg\_t**

Stream event message.

#### Public Members

*http\_stream\_event\_id\_t* **event\_id**

Event ID

void \***http\_client**

Reference to HTTP Client using by this HTTP Stream

void \***buffer**

Reference to Buffer using by the Audio Element

int **buffer\_len**

Length of buffer

void \***user\_data**

User data context, from *http\_stream\_cfg\_t*

*audio\_element\_handle\_t* **el**

Audio element context

struct **http\_stream\_cfg\_t**

HTTP Stream configurations Default value will be used if any entry is zero.

#### Public Members

*audio\_stream\_type\_t* **type**

Type of stream

int **out\_rb\_size**

Size of output ringbuffer

int **task\_stack**

Task stack size

int **task\_core**

Task running in core (0 or 1)

int **task\_prio**

Task priority (based on freeRTOS priority)

bool **stack\_in\_ext**

Try to allocate stack in external memory

*http\_stream\_event\_handle\_t* **event\_handle**

The hook function for HTTP Stream

void **\*user\_data**

User data context

bool **auto\_connect\_next\_track**

connect next track without open/close

bool **enable\_playlist\_parser**

Enable playlist parser

int **multi\_out\_num**

The number of multiple output

const char **\*cert\_pem**

SSL server certification, PEM format as string, if the client requires to verify server

esp\_err\_t (**\*cert\_bundle\_attach**)(void \*conf)

Function pointer to esp\_cert\_bundle\_attach. Enables the use of certification bundle for server verification, must be enabled in menuconfig

int **request\_size**

Request data size each time from http\_client Defaults use DEFAULT\_ELEMENT\_BUFFER\_LENGTH if set to 0 Need care this setting if audio frame size is small and want low latency playback

int **request\_range\_size**

Range size setting for header Range: bytes=start-end Request full range of resource if set to 0 Range size bigger than request size is recommended

const char **\*user\_agent**

The User Agent string to send with HTTP requests

## Macros

**HTTP\_STREAM\_TASK\_STACK**

**HTTP\_STREAM\_TASK\_CORE**

**HTTP\_STREAM\_TASK\_PRIO**

**HTTP\_STREAM\_RINGBUFFER\_SIZE**

**HTTP\_STREAM\_CFG\_DEFAULT** ( )

### Type Definitions

```
typedef int (*http_stream_event_handle_t)(http_stream_event_msg_t *msg)
```

### Enumerations

enum **http\_stream\_event\_id\_t**

HTTP Stream hook type.

*Values:*

enumerator **HTTP\_STREAM\_PRE\_REQUEST**

The event handler will be called before HTTP Client making the connection to the server

enumerator **HTTP\_STREAM\_ON\_REQUEST**

The event handler will be called when HTTP Client is requesting data, If the function return the value (-1: ESP\_FAIL), HTTP Client will be stopped If the function return the value > 0, HTTP Stream will ignore the post\_field If the function return the value = 0, HTTP Stream continue send data from post\_field (if any)

enumerator **HTTP\_STREAM\_ON\_RESPONSE**

The event handler will be called when HTTP Client is receiving data If the function return the value (-1: ESP\_FAIL), HTTP Client will be stopped If the function return the value > 0, HTTP Stream will ignore the read function If the function return the value = 0, HTTP Stream continue read data from HTTP Server

enumerator **HTTP\_STREAM\_POST\_REQUEST**

The event handler will be called after HTTP Client send header and body to the server, before fetching the headers

enumerator **HTTP\_STREAM\_FINISH\_REQUEST**

The event handler will be called after HTTP Client fetch the header and ready to read HTTP body

enumerator **HTTP\_STREAM\_RESOLVE\_ALL\_TRACKS**

enumerator **HTTP\_STREAM\_FINISH\_TRACK**

enumerator **HTTP\_STREAM\_FINISH\_PLAYLIST**

## 2.2.4 I2S Stream

The I2S stream receives and transmits audio data through the chip's I2S, PDM, ADC, and DAC interfaces. To use the ADC and DAC functions, the chip needs to define `SOC_I2S_SUPPORTS_ADC_DAC`. The stream integrates automatic level control (ALC) to adjust volume, multi-channel output, and sending audio data with extended bit width. The relevant control bits are defined in `i2s_stream_cfg_t`.

### Application Example

- Reader example: `recorder/pipeline_wav_amr_sdcard`
- Writer example: `get-started/play_mp3_control`

### Header File

- `components/audio_stream/include/i2s_stream.h`

### Functions

`audio_element_handle_t i2s_stream_init (i2s_stream_cfg_t *config)`

Create a handle to an Audio Element to stream data from I2S to another Element or get data from other elements sent to I2S, depending on the configuration of stream type is `AUDIO_STREAM_READER` or `AUDIO_STREAM_WRITER`.

---

**Note:** If I2S stream is enabled with built-in DAC mode, please don't use `I2S_NUM_1`. The built-in DAC functions are only supported on I2S0 for the current ESP32 chip.

---

**Parameters** `config` – The configuration

**Returns** The Audio Element handle

`esp_err_t i2s_stream_set_channel_type (i2s_stream_cfg_t *config, i2s_channel_type_t type)`

Set I2S stream channel format type.

---

**Note:** : This function only updates `i2s_stream_cfg_t`, so it needs to be called before `i2s_stream_init`.

---

#### Parameters

- `config` – [in] The I2S stream configuration
- `type` – [in] I2S channel format type

#### Returns

- `ESP_OK`
- `ESP_ERR_INVALID_ARG`

`esp_err_t i2s_stream_set_clk` (*audio\_element\_handle\_t* i2s\_stream, int rate, int bits, int ch)

Setup clock for I2S Stream, this function is only used with handle created by `i2s_stream_init`

### Parameters

- **i2s\_stream** – [in] The i2s element handle
- **rate** – [in] Clock rate (in Hz)
- **bits** – [in] Audio bit width (8, 16, 24, 32)
- **ch** – [in] Number of Audio channels (1: Mono, 2: Stereo). But when set to tdm mode, ch is slot mask.(ex: I2S\_TDM\_SLOT0 | I2S\_TDM\_SLOT1 | I2S\_TDM\_SLOT2 | I2S\_TDM\_SLOT3)

### Returns

- ESP\_OK
- ESP\_FAIL

`esp_err_t i2s_alc_volume_set` (*audio\_element\_handle\_t* i2s\_stream, int volume)

Set the volume of input audio stream with ALC. Positive value indicates an increase in volume, negative value indicates a decrease in volume, 0 indicates the volume level remains unchanged.

### Parameters

- **i2s\_stream** – [in] The i2s element handle
- **volume** – [in] The gain of input audio stream:
  - Supported range [-64, 63], unit: dB

### Returns

- ESP\_OK
- ESP\_FAIL

`esp_err_t i2s_alc_volume_get` (*audio\_element\_handle\_t* i2s\_stream, int \*volume)

Get volume of stream.

### Parameters

- **i2s\_stream** – [in] The i2s element handle
- **volume** – [in] The volume of stream

### Returns

- ESP\_OK
- ESP\_FAIL

`esp_err_t i2s_stream_sync_delay` (*audio\_element\_handle\_t* i2s\_stream, int delay\_ms)

Set sync delay of stream.

### Parameters

- **i2s\_stream** – [in] The i2s element handle
- **delay\_ms** – [in] The delay of stream

### Returns

- ESP\_OK
- ESP\_FAIL

## Structures

struct **i2s\_stream\_cfg\_t**

I2S Stream configurations Default value will be used if any entry is zero.

### Public Members

*audio\_stream\_type\_t* **type**

Type of stream

*i2s\_comm\_mode\_t* **transmit\_mode**

I2S transmit mode

*i2s\_chan\_config\_t* **chan\_cfg**

I2S controller channel configuration

*i2s\_std\_config\_t* **std\_cfg**

I2S standard mode major configuration that including clock/slot/gpio configuration

*i2s\_data\_bit\_width\_t* **expand\_src\_bits**

The source bits per sample when data expand

bool **use\_alc**

It is a flag for ALC. If use ALC, the value is true. Or the value is false

int **volume**

The volume of audio input data will be set.

int **out\_rb\_size**

Size of output ringbuffer

int **task\_stack**

Task stack size

int **task\_core**

Task running in core (0 or 1)

int **task\_prio**

Task priority (based on freeRTOS priority)

bool **stack\_in\_ext**

Try to allocate stack in external memory

int **multi\_out\_num**

The number of multiple output

bool **uninstall\_drv**

whether uninstall the i2s driver when stream destroyed

bool **need\_expand**

whether to expand i2s data

int **buffer\_len**

Buffer length use for an Element. Note: when 'bits\_per\_sample' is 24 bit, the buffer length must be a multiple of 3. The recommended value is 3600

### Macros

**I2S\_STREAM\_TASK\_STACK**

**I2S\_STREAM\_BUF\_SIZE**

**I2S\_STREAM\_TASK\_PRIO**

**I2S\_STREAM\_TASK\_CORE**

**I2S\_STREAM\_RINGBUFFER\_SIZE**

**I2S\_STREAM\_CFG\_DEFAULT** ()

**I2S\_STREAM\_CFG\_DEFAULT\_WITH\_PARA** (port, rate, bits, stream\_type)

**I2S\_STD\_PHILIPS\_SLOT\_DEFAULT\_ADF\_CONFIG** (bits\_per\_sample, mono\_or\_stereo)

**I2S\_STREAM\_CFG\_DEFAULT\_WITH\_TYLE\_AND\_CH** (port, rate, bits, stream\_type, channel)

### Enumerations

enum **i2s\_channel\_type\_t**

*Values:*

enumerator **I2S\_CHANNEL\_TYPE\_RIGHT\_LEFT**

Separated left and right channel

enumerator **I2S\_CHANNEL\_TYPE\_ALL\_RIGHT**

Load right channel data in both two channels

enumerator **I2S\_CHANNEL\_TYPE\_ALL\_LEFT**

Load left channel data in both two channels



enumerator **I2S\_CHANNEL\_TYPE\_ONLY\_RIGHT**

Only load data in right channel (mono mode)

enumerator **I2S\_CHANNEL\_TYPE\_ONLY\_LEFT**

Only load data in left channel (mono mode)

## 2.2.5 PWM Stream

In some cost-sensitive scenarios, the audio signal is not converted by the DAC but is modulated by the PWM (pulse width modulation) and then implemented by a filter circuit. The PWM stream modulates the audio signal with the chip's PWM and sends out the processed audio. It only has the `AUDIO_STREAM_WRITER` type. Note that the digital-to-analog conversion by PWM has a lower signal-to-noise ratio.

### Application Example

- Writer example: `player/pipeline_play_mp3_with_dac_or_pwm`

### Header File

- `components/audio_stream/include/pwm_stream.h`

### Functions

`audio_element_handle_t pwm_stream_init (pwm_stream_cfg_t *config)`

Initialize PWM stream Only support `AUDIO_STREAM_READER` type.

**Parameters** `config` – The PWM Stream configuration

**Returns** The audio element handle

`esp_err_t pwm_stream_set_clk (audio_element_handle_t pwm_stream, int rate, int bits, int ch)`

Setup clock for PWM Stream, this function is only used with handle created by `pwm_stream_init`

**Parameters**

- `pwm_stream` – [in] The pwm element handle
- `rate` – [in] Clock rate (in Hz)
- `bits` – [in] Audio bit width (16, 32)
- `ch` – [in] Number of Audio channels (1: Mono, 2: Stereo)

**Returns**

- `ESP_OK`
- `ESP_FAIL`

### Structures

struct **audio\_pwm\_config\_t**

PWM audio configurations.

#### Public Members

timer\_group\_t **tg\_num**

timer group number (0 - 1)

timer\_idx\_t **timer\_num**

timer number (0 - 1)

int **gpio\_num\_left**

the LEDC output gpio\_num, Left channel

int **gpio\_num\_right**

the LEDC output gpio\_num, Right channel

ledc\_channel\_t **ledc\_channel\_left**

LEDC channel (0 - 7), Corresponding to left channel

ledc\_channel\_t **ledc\_channel\_right**

LEDC channel (0 - 7), Corresponding to right channel

ledc\_timer\_t **ledc\_timer\_sel**

Select the timer source of channel (0 - 3)

ledc\_timer\_bit\_t **duty\_resolution**

ledc pwm bits

uint32\_t **data\_len**

ringbuffer size

struct **pwm\_stream\_cfg\_t**

PWM Stream configurations Default value will be used if any entry is zero.

## Public Members

*audio\_stream\_type\_t* **type**

Type of stream

*audio\_pwm\_config\_t* **pwm\_config**

driver configurations

int **out\_rb\_size**

Size of output ringbuffer

int **task\_stack**

Task stack size

int **task\_core**

Task running in core (0 or 1)

int **task\_prio**

Task priority (based on freeRTOS priority)

int **buffer\_len**

pwm\_stream buffer length

bool **ext\_stack**

Allocate stack on extern ram

## Macros

**PWM\_STREAM\_GPIO\_NUM\_LEFT**

**PWM\_STREAM\_GPIO\_NUM\_RIGHT**

**PWM\_STREAM\_TASK\_STACK**

**PWM\_STREAM\_BUF\_SIZE**

**PWM\_STREAM\_TASK\_PRIO**

**PWM\_STREAM\_TASK\_CORE**

**PWM\_STREAM\_RINGBUFFER\_SIZE**

**PWM\_CONFIG\_RINGBUFFER\_SIZE**

**PWM\_STREAM\_CFG\_DEFAULT** ()

### 2.2.6 Raw Stream

The raw stream is used to obtain the output data of the previous element of the connection or to provide the data for the next element of the connection. It does not create a thread. For `AUDIO_STREAM_READER`, the connection is [i2s] -> [filter] -> [raw] or [i2s] -> [codec-amr] -> [raw]. For `AUDIO_STREAM_WRITER`, the connection is [raw] -> [codec-mp3]->[i2s].

#### Application Example

- Reader example: `protocols/voip`
- Writer example: `advanced_examples/downmix_pipeline`

#### Header File

- `components/audio_stream/include/raw_stream.h`

#### Functions

`audio_element_handle_t raw_stream_init (raw_stream_cfg_t *cfg)`

Initialize RAW stream.

**Parameters** `cfg` – The RAW Stream configuration

**Returns** The audio element handle

`int raw_stream_read (audio_element_handle_t pipeline, char *buffer, int buf_size)`

Read data from Stream.

**Parameters**

- `pipeline` – The audio pipeline handle
- `buffer` – The buffer
- `buf_size` – Maximum number of bytes to be read.

**Returns** Number of bytes actually read.

`int raw_stream_write (audio_element_handle_t pipeline, char *buffer, int buf_size)`

Write data to Stream.

**Parameters**

- `pipeline` – The audio pipeline handle
- `buffer` – The buffer
- `buf_size` – Number of bytes to write

**Returns** Number of bytes written

## Structures

struct **raw\_stream\_cfg\_t**

Raw stream provides APIs to obtain the pipeline data without output stream or fill the pipeline data without input stream. The stream has two types / modes, reader and writer:

- AUDIO\_STREAM\_READER, e.g. [i2s]->[filter]->[raw],[i2s]->[codec-amr]->[raw]
- AUDIO\_STREAM\_WRITER, e.g. [raw]->[codec-mp3]->[i2s] Raw Stream configurations

## Public Members

*audio\_stream\_type\_t* **type**

Type of stream

int **out\_rb\_size**

Size of output ringbuffer

## Macros

**RAW\_STREAM\_RINGBUFFER\_SIZE**

**RAW\_STREAM\_CFG\_DEFAULT** ()

## 2.2.7 SPIFFS Stream

The SPIFFS stream reads and writes audio data from or into SPIFFS.

### Application Example

- `player/pipeline_spiffs_mp3`

### Header File

- `components/audio_stream/include/spiffs_stream.h`

### Functions

*audio\_element\_handle\_t* **spiffs\_stream\_init** (*spiffs\_stream\_cfg\_t* \*config)

Create a handle to an Audio Element to stream data from SPIFFS to another Element or get data from other elements written to SPIFFS, depending on the configuration the stream type, either AUDIO\_STREAM\_READER or AUDIO\_STREAM\_WRITER.

**Parameters** **config** – The configuration

**Returns** The Audio Element handle

### Structures

struct **spiffs\_stream\_cfg\_t**

SPIFFS Stream configuration, if any entry is zero then the configuration will be set to default values.

#### Public Members

*audio\_stream\_type\_t* **type**

Stream type

int **buf\_sz**

Audio Element Buffer size

int **out\_rb\_size**

Size of output ringbuffer

int **task\_stack**

Task stack size

int **task\_core**

Task running in core (0 or 1)

int **task\_prio**

Task priority (based on freeRTOS priority)

bool **write\_header**

Choose to write amrnb/armwb header in spiffs whether or not (true or false, true means choose to write amrnb header)

## Macros

`SPIFFS_STREAM_BUF_SIZE`

`SPIFFS_STREAM_TASK_STACK`

`SPIFFS_STREAM_TASK_CORE`

`SPIFFS_STREAM_TASK_PRIO`

`SPIFFS_STREAM_RINGBUFFER_SIZE`

`SPIFFS_STREAM_CFG_DEFAULT()`

## 2.2.8 TCP Client Stream

The TCP client stream reads and writes server data over TCP.

### Application Example

- `get-started/pipeline_tcp_client`

### Header File

- `components/audio_stream/include/tcp_client_stream.h`

### Functions

*audio\_element\_handle\_t* `tcp_stream_init` (*tcp\_stream\_cfg\_t* \*config)

Initialize a TCP stream to/from an audio element. This function creates a TCP stream to/from an audio element depending on the stream type configuration (e.g., `AUDIO_STREAM_READER` or `AUDIO_STREAM_WRITER`). The handle of the audio element is returned.

**Parameters** `config` – The configuration

**Returns** The audio element handle

### Structures

struct `tcp_stream_event_msg`

TCP Stream message configuration.

### Public Members

void \***source**

Element handle

void \***data**

Data of input/output

int **data\_len**

Data length of input/output

esp\_transport\_handle\_t **sock\_fd**

handle of socket

struct **tcp\_stream\_cfg\_t**

TCP Stream configuration, if any entry is zero then the configuration will be set to default values.

### Public Members

*audio\_stream\_type\_t* **type**

Type of stream

int **timeout\_ms**

time timeout for read/write

int **port**

TCP port>

char \***host**

TCP host>

int **task\_stack**

Task stack size

int **task\_core**

Task running in core (0 or 1)

int **task\_prio**

Task priority (based on freeRTOS priority)

bool **ext\_stack**

Allocate stack on extern ram

*tcp\_stream\_event\_handle\_cb* **event\_handler**

TCP stream event callback



void \***event\_ctx**  
User context

## Macros

**TCP\_STREAM\_DEFAULT\_PORT**  
TCP stream parameters.

**TCP\_STREAM\_TASK\_STACK**

**TCP\_STREAM\_BUF\_SIZE**

**TCP\_STREAM\_TASK\_PRIO**

**TCP\_STREAM\_TASK\_CORE**

**TCP\_SERVER\_DEFAULT\_RESPONSE\_LENGTH**

**TCP\_STREAM\_CFG\_DEFAULT** ()

## Type Definitions

typedef struct *tcp\_stream\_event\_msg* **tcp\_stream\_event\_msg\_t**  
TCP Stream message configuration.

typedef esp\_err\_t (\***tcp\_stream\_event\_handle\_cb**)(*tcp\_stream\_event\_msg\_t* \*msg, *tcp\_stream\_status\_t* state, void \*event\_ctx)

## Enumerations

enum **tcp\_stream\_status\_t**

*Values:*

enumerator **TCP\_STREAM\_STATE\_NONE**

enumerator **TCP\_STREAM\_STATE\_CONNECTED**

### 2.2.9 Tone Stream

The tone stream reads the data generated by `tools/audio_tone/mk_audio_tone.py`. It only supports the `AUDIO_STREAM_READER` type.

#### Application Example

- `player/pipeline_flash_tone`

#### Header File

- `components/audio_stream/include/tone_stream.h`

#### Functions

*audio\_element\_handle\_t* **tone\_stream\_init** (*tone\_stream\_cfg\_t* \*config)

Create an Audio Element handle to stream data from flash to another Element, only support `AUDIO_STREAM_READER` type.

**Parameters** `config` – The configuration

**Returns** The Audio Element handle

#### Structures

struct **tone\_stream\_cfg\_t**

TONE Stream configurations, if any entry is zero then the configuration will be set to default values.

#### Public Members

*audio\_stream\_type\_t* **type**

Stream type

int **buf\_sz**

Audio Element Buffer size

int **out\_rb\_size**

Size of output ringbuffer

int **task\_stack**

Task stack size

int **task\_core**

Task running in core (0 or 1)

int **task\_prio**

Task priority (based on freeRTOS priority)

const char \***label**

Label of tone stored in flash. The default value is `flash_tone`

bool **extern\_stack**

Task stack allocate on the extern ram

bool **use\_delegate**

Read tone partition with `esp_delegate`. If task stack is on extern ram, this MUST be TRUE

## Macros

**TONE\_STREAM\_BUF\_SIZE**

**TONE\_STREAM\_TASK\_STACK**

**TONE\_STREAM\_TASK\_CORE**

**TONE\_STREAM\_TASK\_PRIO**

**TONE\_STREAM\_RINGBUFFER\_SIZE**

**TONE\_STREAM\_EXT\_STACK**

**TONE\_STREAM\_USE\_DELEGATE**

**TONE\_STREAM\_CFG\_DEFAULT()**

### 2.2.10 Flash-Embedding Stream

The flash-embedding stream reads the data generated by `tools/audio_tone/mk_embed_flash.py`. It only supports the `AUDIO_STREAM_READER` type.

#### Application Example

- `player/pipeline_embed_flash_tone`

### Header File

- components/audio\_stream/include/embed\_flash\_stream.h

### Functions

*audio\_element\_handle\_t* **embed\_flash\_stream\_init** (*embed\_flash\_stream\_cfg\_t* \*config)

Create an Audio Element handle to stream data from flash to another Element, only support AUDIO\_STREAM\_READER type.

**Parameters** **config** – The configuration

**Returns** The Audio Element handle

esp\_err\_t **embed\_flash\_stream\_set\_context** (*audio\_element\_handle\_t* embed\_stream, const *embed\_item\_info\_t* \*context, int max\_num)

Set the embed flash context.

This function mainly provides information about embed flash data

#### Parameters

- **embed\_stream** – [in] The embed flash element handle
- **context** – [in] The embed flash context
- **max\_num** – [in] The number of embed flash context

#### Returns

- ESP\_OK
- ESP\_FAIL

### Structures

struct **embed\_flash\_stream\_cfg\_t**

Flash-embedding stream configurations, if any entry is zero then the configuration will be set to default values.

#### Public Members

int **buf\_sz**

Audio Element Buffer size

int **out\_rb\_size**

Size of output ringbuffer

int **task\_stack**

Task stack size

int **task\_core**

Task running in core (0 or 1)

int **task\_prio**

Task priority (based on freeRTOS priority)

bool **extern\_stack**

At present, task stack can only be placed on SRAM, so it should always be set to `false`

struct **embed\_item\_info**

Embed tone information in flash.

### Public Members

const uint8\_t \***address**

The corresponding address in flash

int **size**

Size of corresponding data

### Macros

**EMBED\_FLASH\_STREAM\_BUF\_SIZE**

**EMBED\_FLASH\_STREAM\_TASK\_STACK**

**EMBED\_FLASH\_STREAM\_TASK\_CORE**

**EMBED\_FLASH\_STREAM\_TASK\_PRIO**

**EMBED\_FLASH\_STREAM\_RINGBUFFER\_SIZE**

**EMBED\_FLASH\_STREAM\_EXT\_STACK**

**EMBED\_FLASH\_STREAM\_CFG\_DEFAULT** ( )

## Type Definitions

typedef struct *embed\_item\_info* **embed\_item\_info\_t**  
Embed tone information in flash.

### 2.2.11 TTS Stream

The tex-to-speech stream (TTS stream) obtains the `esp_tts_voice` data of `esp-sr`. It only supports the `AUDIO_STREAM_READER` type.

#### Application Example

- Reader example: `player/pipeline_tts_stream`

#### Header File

- `components/audio_stream/include/tts_stream.h`

#### Functions

*audio\_element\_handle\_t* **tts\_stream\_init** (*tts\_stream\_cfg\_t* \*config)

Create a handle to an Audio Element to stream data from TTS to another Element, the stream type only support `AUDIO_STREAM_READER` for now.

**Parameters** `config` – The configuration

**Returns** The Audio Element handle

`esp_err_t` **tts\_stream\_set\_strings** (*audio\_element\_handle\_t* el, const char \*strings)

Set tts stream strings.

**Parameters**

- `el` – [in] The audio element handle
- `strings` – [in] The string pointer

**Returns**

- `ESP_OK`
- `ESP_FAIL`

`esp_err_t` **tts\_stream\_set\_speed** (*audio\_element\_handle\_t* el, *tts\_voice\_speed\_t* speed)

Setting tts stream voice speed.

**Parameters**

- `el` – [in] The `esp_audio` instance
- `speed` – [in] Speed will be set. 0-5 is legal. 0 is the slowest speed.

**Returns**

- `ESP_OK`
- `ESP_FAIL`

esp\_err\_t **tts\_stream\_get\_speed** (*audio\_element\_handle\_t* el, *tts\_voice\_speed\_t* \*speed)

Get tts stream voice speed.

#### Parameters

- **e1** – [in] The esp\_audio instance
- **speed** – [in] Return tts stream Speed will be [0,5]

#### Returns

- ESP\_OK
- ESP\_FAIL

## Structures

struct **tts\_stream\_cfg\_t**

TTS Stream configurations, if any entry is zero then the configuration will be set to default values.

### Public Members

*audio\_stream\_type\_t* **type**

Stream type

int **buf\_sz**

Audio Element Buffer size

int **out\_rb\_size**

Size of output ringbuffer

int **task\_stack**

Task stack size

int **task\_core**

Task running in core (0 or 1)

int **task\_prio**

Task priority (based on freeRTOS priority)

bool **ext\_stack**

Allocate stack on extern ram

### Macros

**TTS\_STREAM\_BUF\_SIZE**

**TTS\_STREAM\_TASK\_STACK**

**TTS\_STREAM\_TASK\_CORE**

**TTS\_STREAM\_TASK\_PRIO**

**TTS\_STREAM\_RINGBUFFER\_SIZE**

**TTS\_STREAM\_CFG\_DEFAULT** ()

### Enumerations

enum **tts\_voice\_speed\_t**

*Values:*

enumerator **TTS\_VOICE\_SPEED\_0**

enumerator **TTS\_VOICE\_SPEED\_1**

enumerator **TTS\_VOICE\_SPEED\_2**

enumerator **TTS\_VOICE\_SPEED\_3**

enumerator **TTS\_VOICE\_SPEED\_4**

enumerator **TTS\_VOICE\_SPEED\_5**

enumerator **TTS\_VOICE\_SPEED\_MAX**

## 2.3 Playlist

A playlist is a list of audio files that can be played back either sequentially or in a specified order.

The `sdcard_scan()` function in the `playlist/include/sdcard_scan.h` scans the audio files in a microSD card and generate a playlist of files. You can specify file depth and filter out file types when scanning. The playlist instances can be saved to a variety of storage media. The following are the supported storage media:

- *Saving to MicroSD Card*
- *Saving to DRAM*



- *Saving to NVS Partition in Flash*
- *Saving to DATA\_UNDEFINED Partition in Flash*

After scanning the audio files, you can use the `playlist_operator_handle_t` handle to call the corresponding functions to create, save, print playlists, and obtain the path corresponding to the audio ID. Currently, most of the storage media mentioned in this document support the above functions.

See the description below for the API details.

### 2.3.1 Scanning MicroSD Card

The `sdcard_scan()` function can scan audio files in a specified path and generate playlists. It supports the scanning of files at a specified depth and filtering of file types. Then, the playlist can be saved to the specified storage medium using a callback function.

#### Application Example

- `player/pipeline_sdcard_mp3_control`
- `cli`

#### Header File

- `components/playlist/include/sdcard_scan.h`

#### Functions

`esp_err_t sdcard_scan(sdcard_scan_cb_t cb, const char *path, int depth, const char *file_extension[], int filter_num, void *user_data)`

Scan files in SD card and use callback function to save files that meet filtering conditions.

---

**Note:** example `sdcard_scan(callback, "/sdcard", 5, const char *[]{"mp3", "aac"}, 2, user_data)`; Scan 5 levels folder in sdcard and save mp3 files and aac files.

---

#### Parameters

- **cb** – The callback function
- **path** – The path to be scanned
- **depth** – The depth of file scanning // .e.g. if you only want to save files in “/test”, depth = 0. // if you want to save files in “/test/scan\_test/”, depth = 1
- **file\_extension** – File extension of files that are supposed to be saved // .e.g. `const char *[]{"mp3", "aac"}`
- **filter\_num** – Number of filters
- **user\_data** – The data to be used by callback function

#### Returns

- `ESP_OK` success
- `ESP_FAIL` failed

### Type Definitions

```
typedef void (*sdcard_scan_cb_t)(void *user_data, char *url)
```

### 2.3.2 Saving Playlist

#### Saving to MicroSD Card

The playlist can be stored in the microSD card. Functions, such as those to save and display the playlist, can be called through the `playlist_operator_handle_t` handle.

#### Application Example

- `player/pipeline_sdcard_mp3_control`
- `cli`

#### Header File

- `components/playlist/include/sdcard_list.h`

#### Functions

`esp_err_t sdcard_list_create (playlist_operator_handle_t *handle)`

Create a playlist in sdcard by list id.

**Parameters** `handle` – [out] The playlist handle from application layer

**Returns**

- `ESP_OK` success
- `ESP_FAIL` failed

`esp_err_t sdcard_list_show (playlist_operator_handle_t handle)`

Show all the URLs in sdcard playlist.

**Parameters** `handle` – Playlist handle

**Returns**

- `ESP_OK` success
- `ESP_FAIL` failed

`esp_err_t sdcard_list_next (playlist_operator_handle_t handle, int step, char **url_buff)`

The following URLs in sdcard playlist.

**Parameters**

- `handle` – Playlist handle
- `step` – The offset of URL from current URL
- `url_buff` – [out] A second rank pointer to get a address of URL

**Returns**

- ESP\_OK success
- ESP\_FAIL failed

esp\_err\_t **sdcard\_list\_prev** (*playlist\_operator\_handle\_t* handle, int step, char \*\*url\_buff)

The previous URLs in sdcard playlist.

**Parameters**

- **handle** – Playlist handle
- **step** – The offset of URL from current URL
- **url\_buff** – [out] A second rank pointer to get a address of URL

**Returns**

- ESP\_OK success
- ESP\_FAIL failed

bool **sdcard\_list\_exist** (*playlist\_operator\_handle\_t* handle, const char \*url)

Judge whether the url exists in sdcard playlist.

**Parameters**

- **handle** – Playlist handle
- **url** – The url to be checked

**Returns**

- true existence
- false Non-existent

esp\_err\_t **sdcard\_list\_reset** (*playlist\_operator\_handle\_t* handle)

Reset sdcard playlist.

**Parameters** **handle** – Playlist handle

**Returns**

- ESP\_OK success
- ESP\_FAIL failed

esp\_err\_t **sdcard\_list\_current** (*playlist\_operator\_handle\_t* handle, char \*\*url\_buff)

Get current URL in sdcard playlist.

**Parameters**

- **handle** – Playlist handle
- **url\_buff** – [out] A second rank pointer to get a address of URL

**Returns**

- ESP\_OK success
- ESP\_FAIL failed

esp\_err\_t **sdcard\_list\_choose** (*playlist\_operator\_handle\_t* handle, int url\_id, char \*\*url\_buff)

Choose a url by url id.

**Parameters**

- **handle** – Playlist handle
- **url\_id** – The id of url in sdc card list
- **url\_buff** – [out] A second rank pointer to get a address of URL

### Returns

- ESP\_OK success
- ESP\_FAIL failed

int **sdc card\_list\_get\_url\_num** (*playlist\_operator\_handle\_t* handle)

Get URLs number in sdc card playlist.

**Parameters** **handle** – Playlist handle

### Returns

- URLs number in sdc card playlist
- ESP\_FAIL Fail to get number of urls

int **sdc card\_list\_get\_url\_id** (*playlist\_operator\_handle\_t* handle)

Get current url id in the sdc card playlist.

**Parameters** **handle** – Playlist handle

### Returns

- Current url id in partition playlist
- ESP\_FAIL Fail to get url id

esp\_err\_t **sdc card\_list\_destroy** (*playlist\_operator\_handle\_t* handle)

Destroy sdc card playlist.

**Parameters** **handle** – Playlist handle

### Returns

- ESP\_OK success
- ESP\_FAIL failed

esp\_err\_t **sdc card\_list\_save** (*playlist\_operator\_handle\_t* handle, const char \*url)

Save URL to sdc card playlist.

### Parameters

- **handle** – Playlist handle
- **url** – URL to be saved

### Returns

- ESP\_OK success
- ESP\_FAIL failed

## Saving to DRAM

The playlist can be stored in DRAM. Functions, such as those to save and display the playlist, can be called through the `playlist_operator_handle_t` handle.

### Header File

- `components/playlist/include/dram_list.h`

### Functions

`esp_err_t dram_list_create` (*playlist\_operator\_handle\_t* \*handle)

Create a playlist in dram.

**Parameters** `handle` – [out] The playlist handle from application layer

#### Returns

- `ESP_OK` success
- `ESP_FAIL` failed

`esp_err_t dram_list_save` (*playlist\_operator\_handle\_t* handle, const char \*url)

Save URL to dram playlist.

#### Parameters

- `handle` – Playlist handle
- `url` – URL to be saved

#### Returns

- `ESP_OK` success
- `ESP_FAIL` failed

`esp_err_t dram_list_next` (*playlist\_operator\_handle\_t* handle, int step, char \*\*url\_buff)

The following URLs in dram playlist.

#### Parameters

- `handle` – Playlist handle
- `step` – The offset of URL from current URL
- `url_buff` – [out] A second rank pointer to get a address of URL

#### Returns

- `ESP_OK` success
- `ESP_FAIL` failed

`esp_err_t dram_list_prev` (*playlist\_operator\_handle\_t* handle, int step, char \*\*url\_buff)

The previous URLs in dram playlist.

#### Parameters

- `handle` – Playlist handle
- `step` – The offset of URL from current URL

- **url\_buff** – [out] A second rank pointer to get a address of URL

### Returns

- ESP\_OK success
- ESP\_FAIL failed

bool **dram\_list\_exist** (*playlist\_operator\_handle\_t* handle, const char \*url)

Judge whether the url exists in dram playlist.

### Parameters

- **handle** – Playlist handle
- **url** – The url to be checked

### Returns

- true existence
- false Non-existent

esp\_err\_t **dram\_list\_reset** (*playlist\_operator\_handle\_t* handle)

Reset dram playlist.

**Parameters** **handle** – Playlist handle

### Returns

- ESP\_OK success
- ESP\_FAIL failed

esp\_err\_t **dram\_list\_current** (*playlist\_operator\_handle\_t* handle, char \*\*url\_buff)

The current URL in current playlist.

### Parameters

- **handle** – Playlist handle
- **url\_buff** – [out] A second rank pointer to get a address of URL

### Returns

- ESP\_OK success
- ESP\_FAIL failed

esp\_err\_t **dram\_list\_choose** (*playlist\_operator\_handle\_t* handle, int url\_id, char \*\*url\_buff)

Choose a url by url id.

### Parameters

- **handle** – Playlist handle
- **url\_id** – The id of url in dram list
- **url\_buff** – [out] A second rank pointer to get a address of URL

### Returns

- ESP\_OK success
- ESP\_FAIL failed

int **dram\_list\_get\_url\_num** (*playlist\_operator\_handle\_t* handle)

Get URLs number in the dram playlist.

**Parameters** **handle** – Playlist handle

**Returns**

- URLs number in dram playlist
- ESP\_FAIL Fail to get number of urls

int **dram\_list\_get\_url\_id** (*playlist\_operator\_handle\_t* handle)

Get current url id in the dram playlist.

**Parameters** **handle** – Playlist handle

**Returns**

- Current url id in dram playlist
- ESP\_FAIL Fail to get url id

esp\_err\_t **dram\_list\_show** (*playlist\_operator\_handle\_t* handle)

Show all the URLs in the dram playlist.

**Parameters** **handle** – Playlist handle

**Returns**

- ESP\_OK success
- ESP\_FAIL failed

esp\_err\_t **dram\_list\_remove\_by\_url** (*playlist\_operator\_handle\_t* handle, const char \*url)

Remove corresponding url in dram list.

**Parameters**

- **handle** – Playlist handle
- **url** – The url to be removed

**Returns**

- ESP\_OK success
- ESP\_FAIL failed

esp\_err\_t **dram\_list\_remove\_by\_url\_id** (*playlist\_operator\_handle\_t* handle, uint16\_t url\_id)

Remove url by id.

**Parameters**

- **handle** – Playlist handle
- **url\_id** – The url id to be removed

**Returns**

- ESP\_OK success
- ESP\_FAIL failed

esp\_err\_t **dram\_list\_destroy** (*playlist\_operator\_handle\_t* handle)

Destroy the dram playlist.

**Parameters** **handle** – Playlist handle

### Returns

- ESP\_OK success
- ESP\_FAIL failed

## Saving to NVS Partition in Flash

The playlist can be stored in the [NVS partition](#) in flash. Functions, such as those to save and display the playlist, can be called through the `playlist_operator_handle_t` handle.

### Header File

- `components/playlist/include/flash_list.h`

### Functions

`esp_err_t flash_list_create(playlist_operator_handle_t *handle)`

Create a playlist in nvs flash.

**Parameters** `handle` – [out] Playlist handle

#### Returns

- ESP\_OK success
- ESP\_FAIL failed

`esp_err_t flash_list_save(playlist_operator_handle_t handle, const char *url)`

Save URL to nvs flash list.

#### Parameters

- `handle` – Playlist handle
- `url` – URL to be saved

#### Returns

- ESP\_OK success
- ESP\_FAIL failed

`esp_err_t flash_list_show(playlist_operator_handle_t handle)`

Show all the URLs in nvs flash list.

**Parameters** `handle` – Playlist handle

#### Returns

- ESP\_OK success
- ESP\_FAIL failed

`esp_err_t flash_list_next(playlist_operator_handle_t handle, int step, char **url_buff)`

The following URLs in nvs flash playlist.

#### Parameters

- `handle` – Playlist handle
- `step` – The offset of URL from current URL



- **url\_buff** – [out] A second rank pointer to get a address of URL

**Returns**

- ESP\_OK success
- ESP\_FAIL failed

esp\_err\_t **flash\_list\_prev** (*playlist\_operator\_handle\_t* handle, int step, char \*\*url\_buff)

The previous URLs in nvs flash playlist.

**Parameters**

- **handle** – Playlist handle
- **step** – The offset of URL from current URL
- **url\_buff** – [out] A second rank pointer to get a address of URL

**Returns**

- ESP\_OK success
- ESP\_FAIL failed

esp\_err\_t **flash\_list\_current** (*playlist\_operator\_handle\_t* handle, char \*\*url\_buff)

The current URL in nvs flash playlist.

**Parameters**

- **handle** – Playlist handle
- **url\_buff** – [out] A second rank pointer to get a address of URL

**Returns**

- ESP\_OK success
- ESP\_FAIL failed

bool **flash\_list\_exist** (*playlist\_operator\_handle\_t* handle, const char \*url)

Judge whether the url exists in flash playlist.

**Parameters**

- **handle** – Playlist handle
- **url** – The url to be checked

**Returns**

- true existence
- false Non-existent

esp\_err\_t **flash\_list\_reset** (*playlist\_operator\_handle\_t* handle)

Reset flash playlist.

**Parameters** **handle** – Playlist handle

**Returns**

- ESP\_OK success
- ESP\_FAIL failed

`esp_err_t flash_list_choose` (*playlist\_operator\_handle\_t* handle, int url\_id, char \*\*url\_buff)

Choose a url by url id.

### Parameters

- **handle** – Playlist handle
- **url\_id** – The id of url in flash list
- **url\_buff** – [out] A second rank pointer to get a address of URL

### Returns

- ESP\_OK success
- ESP\_FAIL failed

`int flash_list_get_url_num` (*playlist\_operator\_handle\_t* handle)

Get URLs number in the flash playlist.

**Parameters** **handle** – Playlist handle

### Returns

- URLs number in flash playlist
- ESP\_FAIL Fail to get number of urls

`int flash_list_get_url_id` (*playlist\_operator\_handle\_t* handle)

Get current url id in the flash playlist.

**Parameters** **handle** – Playlist handle

### Returns

- Curernt url id in flash playlist
- ESP\_FAIL Fail to get url id

`esp_err_t flash_list_destroy` (*playlist\_operator\_handle\_t* handle)

Destroy the nvs flash playlist.

**Parameters** **handle** – Playlist handle

### Returns

- ESP\_OK success
- ESP\_FAIL failed

## Saving to DATA\_UNDEFINED Partition in Flash

The playlist can be stored in the DATA\_UNDEFINED partition (see [Partition Tables](#) for details) in flash. Functions, such as those to save and display the playlist, can be called through the `playlist_operator_handle_t` handle. Please add the two partitions whose subtypes are 0x06 and 0x07 to the flash partition table first.

## Header File

- components/playlist/include/partition\_list.h

## Functions

esp\_err\_t **partition\_list\_create** (*playlist\_operator\_handle\_t* \*handle)

Create a playlist in flash partition by list id.

---

**Note:** Please add 2 partitions to partition table whose subtype are 0x06 and 0x07 first

---

**Parameters** **handle** – [out] The playlist handle from application layer

### Returns

- ESP\_OK success
- ESP\_FAIL failed

esp\_err\_t **partition\_list\_save** (*playlist\_operator\_handle\_t* handle, const char \*url)

Save URL to partition playlist.

### Parameters

- **handle** – Playlist handle
- **url** – URL to be saved

### Returns

- ESP\_OK success
- ESP\_FAIL failed

esp\_err\_t **partition\_list\_next** (*playlist\_operator\_handle\_t* handle, int step, char \*\*url\_buff)

The following URLs in partition playlist.

### Parameters

- **handle** – Playlist handle
- **step** – The offset of URL from current URL
- **url\_buff** – [out] A second rank pointer to get a address of URL

### Returns

- ESP\_OK success
- ESP\_FAIL failed

esp\_err\_t **partition\_list\_prev** (*playlist\_operator\_handle\_t* handle, int step, char \*\*url\_buff)

The previous URLs in partition playlist.

### Parameters

- **handle** – Playlist handle
- **step** – The offset of URL from current URL
- **url\_buff** – [out] A second rank pointer to get a address of URL

### Returns

- ESP\_OK success
- ESP\_FAIL failed

bool **partition\_list\_exist** (*playlist\_operator\_handle\_t* handle, const char \*url)

Judge whether the url exists in partition playlist.

### Parameters

- **handle** – Playlist handle
- **url** – The url to be checked

### Returns

- true existence
- false Non-existent

esp\_err\_t **partition\_list\_reset** (*playlist\_operator\_handle\_t* handle)

Reset partition playlist.

**Parameters** **handle** – Playlist handle

### Returns

- ESP\_OK success
- ESP\_FAIL failed

esp\_err\_t **partition\_list\_current** (*playlist\_operator\_handle\_t* handle, char \*\*url\_buff)

Get current URL in the partition playlist.

### Parameters

- **handle** – Playlist handle
- **url\_buff** – [out] A second rank pointer to get a address of URL

### Returns

- ESP\_OK success
- ESP\_FAIL failed

esp\_err\_t **partition\_list\_choose** (*playlist\_operator\_handle\_t* handle, int url\_id, char \*\*url\_buff)

Choose a url by url id.

### Parameters

- **handle** – Playlist handle
- **url\_id** – The id of url in partition list
- **url\_buff** – [out] A second rank pointer to get a address of URL

### Returns

- ESP\_OK success
- ESP\_FAIL failed

int **partition\_list\_get\_url\_num** (*playlist\_operator\_handle\_t* handle)

Get URLs number in the partition playlist.

**Parameters** **handle** – Playlist handle

**Returns**

- URLs number in partition playlist
- ESP\_FAIL Fail to get number of urls

int **partition\_list\_get\_url\_id** (*playlist\_operator\_handle\_t* handle)

Get cureernt url id in the partition playlist.

**Parameters** **handle** – Playlist handle

**Returns**

- Current url id in partition playlist
- ESP\_FAIL Fail to get url id

esp\_err\_t **partition\_list\_show** (*playlist\_operator\_handle\_t* handle)

Show all the URLs in the partition playlist.

**Parameters** **handle** – Playlist handle

**Returns**

- ESP\_OK success
- ESP\_FAIL failed

esp\_err\_t **partition\_list\_destroy** (*playlist\_operator\_handle\_t* handle)

Destroy the partition playlist.

**Parameters** **handle** – Playlist handle

**Returns**

- ESP\_OK success
- ESP\_FAIL failed

### 2.3.3 Playlist Manager

Playlist Manager manages the above playlists and can add multiple playlist instances to the `playlist_handle_t` handle.

#### Header File

- `components/playlist/include/playlist.h`

#### Functions

*playlist\_handle\_t* **playlist\_create** (void)

Create a playlist manager handle.

**Returns**

- playlist handle success
- NULL failed

`esp_err_t playlist_add` (*playlist\_handle\_t* handle, *playlist\_operator\_handle\_t* list\_handle, `uint8_t` list\_id)

Create a playlist manager and add playlist handle to it.

---

**Note:** The partition playlist can only be added once, or it will be overwritten by the newest partition playlist

---

---

**Note:** Different lists must use different IDs, because even if they are in different handles, `list_id` is the only indicator that distinguishes them.

---

### Parameters

- **handle** – Playlist manager handle
- **list\_handle** – The playlist handle to be added
- **list\_id** – The playlist id to be registered

### Returns

- `ESP_OK` success
- `ESP_FAIL` failed

`esp_err_t playlist_checkout_by_id` (*playlist\_handle\_t* handle, `uint8_t` id)

Playlist checkout by list id.

### Parameters

- **handle** – Playlist handle
- **id** – Specified list id

### Returns

- `ESP_OK` success
- `ESP_FAIL` failed

`int playlist_get_list_num` (*playlist\_handle\_t* handle)

Get number of playlists in the handle.

**Parameters** **handle** – Playlist handle

### Returns

- success Number of playlists in handle
- failed -1

*playlist\_type\_t* `playlist_get_current_list_type` (*playlist\_handle\_t* handle)

Get current playlist type.

**Parameters** **handle** – Playlist handle

### Returns

- success Type of current playlist
- failed -1

int **playlist\_get\_current\_list\_id** (*playlist\_handle\_t* handle)

Get current playlist id.

**Parameters** **handle** – Playlist handle

**Returns**

- success Current playlist id
- failed -1

esp\_err\_t **playlist\_get\_current\_list\_url** (*playlist\_handle\_t* handle, char \*\*url\_buff)

Get current URL in current playlist.

**Parameters**

- **handle** – Playlist handle
- **url\_buff** – [out] A second rank pointer to get a address of URL

**Returns**

- ESP\_OK success
- ESP\_FAIL failed

int **playlist\_get\_current\_list\_url\_num** (*playlist\_handle\_t* handle)

Get number of URLs in current playlist.

**Parameters** **handle** – Playlist handle

**Returns** Number of URLS in current playsit

int **playlist\_get\_current\_list\_url\_id** (*playlist\_handle\_t* handle)

Get current url id in current playlist.

**Parameters** **handle** – Playlist handle

**Returns** Current url's id in current playsit

esp\_err\_t **playlist\_save** (*playlist\_handle\_t* handle, const char \*url)

Save a URL to the current playlist.

**Parameters**

- **handle** – Playlist handle
- **url** – The URL to be saved ot sdcad

**Returns**

- ESP\_OK success
- ESP\_FAIL failed

esp\_err\_t **playlist\_next** (*playlist\_handle\_t* handle, int step, char \*\*url\_buff)

Next URI in current playlist.

**Parameters**

- **handle** – Playlist handle
- **step** – Next steps from current position
- **url\_buff** – [out] A second rank pointer to get a address of URL

**Returns**

- ESP\_OK success
- ESP\_FAIL failed

esp\_err\_t **playlist\_prev** (*playlist\_handle\_t* handle, int step, char \*\*url\_buff)

Previous URL in current playlist.

### Parameters

- **handle** – Playlist handle
- **step** – Previous steps from current position
- **url\_buff** – [out] A second rank pointer to get a address of URL

### Returns

- ESP\_OK success
- ESP\_FAIL failed

esp\_err\_t **playlist\_choose** (*playlist\_handle\_t* handle, int url\_id, char \*\*url\_buff)

Choose a url by url id.

### Parameters

- **handle** – Playlist handle
- **url\_id** – The id of url in current list
- **url\_buff** – [out] A second rank pointer to get a address of URL

### Returns

- ESP\_OK success
- ESP\_FAIL failed

esp\_err\_t **playlist\_show** (*playlist\_handle\_t* handle)

Show URLs in current playlist.

**Parameters** **handle** – Playlist handle

### Returns

- ESP\_OK success
- ESP\_FAIL failed

esp\_err\_t **playlist\_reset** (*playlist\_handle\_t* handle)

Reset current playlist.

**Parameters** **handle** – Playlist handle

### Returns

- ESP\_OK success
- ESP\_FAIL failed

esp\_err\_t **playlist\_remove\_by\_url** (*playlist\_handle\_t* handle, const char \*url)

Remove corresponding url.

### Parameters

- **handle** – Playlist handle
- **url** – The url to be removed



**Returns**

- ESP\_OK success
- ESP\_FAIL failed

esp\_err\_t **playlist\_remove\_by\_url\_id** (*playlist\_handle\_t* handle, uint16\_t url\_id)

Remove url by url id.

**Parameters**

- **handle** – Playlist handle
- **url\_id** – The id of url to be removed

**Returns**

- ESP\_OK success
- ESP\_FAIL failed

bool **playlist\_exist** (*playlist\_handle\_t* handle, const char \*url)

Judge whether the url exists in current playlist.

**Parameters**

- **handle** – Playlist handle
- **url** – The url to be checked

**Returns**

- true existence
- false Non-existent

esp\_err\_t **playlist\_destroy** (*playlist\_handle\_t* handle)

Destroy all playlists in the handle.

**Parameters** **handle** – Playlist handle

**Returns**

- ESP\_OK success
- ESP\_FAIL failed

**Structures**

struct **playlist\_operation\_t**

All types of Playlists' operation.

### Public Members

`esp_err_t (*show)(void *playlist)`

Show all the URLs in playlist

`esp_err_t (*save)(void *playlist, const char *url)`

Save URLs to playlist

`esp_err_t (*next)(void *playlist, int step, char **url_buff)`

Get next URL in playlist

`esp_err_t (*prev)(void *playlist, int step, char **url_buff)`

Get previous URL in playlist

`esp_err_t (*reset)(void *playlist)`

Reset the playlist

`esp_err_t (*choose)(void *playlist, int url_id, char **url_buff)`

Get url by url id

`esp_err_t (*current)(void *playlist, char **url_buff)`

Get current URL in playlist

`esp_err_t (*destroy)(void *playlist)`

Destroy playlist

`bool (*exist)(void *playlist, const char *url)`

Judge whether the url exists

`int (*get_url_num)(void *playlist)`

Get number of URLs in current playlist

`int (*get_url_id)(void *playlist)`

Get current url id in playlist

*playlist\_type\_t* **type**

Type of playlist

`esp_err_t (*remove_by_url)(void *playlist, const char *url)`

Remove the corresponding url

`esp_err_t (*remove_by_id)(void *playlist, uint16_t url_id)`

Remove url by id

struct **playlist\_operator\_t**

Information of playlist manager node.

## Public Members

void \***playlist**

Specific playlist's pointer

esp\_err\_t (\***get\_operation**)(*playlist\_operation\_t* \*operation)

Function pointer to get playlists' handle

## Type Definitions

```
typedef playlist_operator_t *playlist_operator_handle_t
```

```
typedef struct playlist_handle *playlist_handle_t
```

## Enumerations

```
enum playlist_type_t
```

Type of playlist.

*Values:*

```
enumerator PLAYLIST_UNKNOWN
```

Unknown type

```
enumerator PLAYLIST_SDCARD
```

Playlist in sdcard

```
enumerator PLAYLIST_FLASH
```

Playlist in nvs

```
enumerator PLAYLIST_DRAM
```

Playlist in ram

```
enumerator PLAYLIST_PARTITION
```

Playlist in partition

## 2.4 Codecs

### 2.4.1 AAC Decoder

Decode an audio data stream provided in AAC format.

### API Reference

#### Header File

- `components/esp-adf-libs/esp_codec/include/codec/aac_decoder.h`

#### Functions

*audio\_element\_handle\_t* **aac\_decoder\_init** (*aac\_decoder\_cfg\_t* \*config)

Create an Audio Element handle to decode incoming AAC data.

**Parameters** `config` – The configuration

**Returns** The audio element handle

#### Structures

struct **aac\_decoder\_cfg\_t**

AAC Decoder configuration.

#### Public Members

int **out\_rb\_size**

Size of output ringbuffer

int **task\_stack**

Task stack size

int **task\_core**

CPU core number (0 or 1) where decoder task is running

int **task\_prio**

Task priority (based on freeRTOS priority)

bool **stack\_in\_ext**

Try to allocate stack in external memory

bool **plus\_enable**

Dynamically enable HE-AAC (v1 v2) decoding

## Macros

`AAC_DECODER_TASK_STACK_SIZE`

`AAC_DECODER_TASK_CORE`

`AAC_DECODER_TASK_PRIO`

`AAC_DECODER_RINGBUFFER_SIZE`

`DEFAULT_AAC_DECODER_CONFIG()`

## 2.4.2 AMR Decoder and Encoder

Decode and encode an audio data stream from / to AMR format. Encoders cover both AMR-NB and AMR-WB formats.

### Application Examples

Implementation of this API is demonstrated in the following examples:

- `player/pipeline_play_sdcard_music`
- `recorder/pipeline_wav_amr_sdcard`

### API Reference - Decoder

#### Header File

- `components/esp-adf-libs/esp_codec/include/codec/amr_decoder.h`

#### Functions

`audio_element_handle_t amr_decoder_init (amr_decoder_cfg_t *config)`

Create an Audio Element handle to decode incoming AMR data.

**Parameters** `config` – The configuration

**Returns** The audio element handle

#### Structures

struct `amr_decoder_cfg_t`

AMR Decoder configuration.

### Public Members

- int **out\_rb\_size**  
Size of output ringbuffer
- int **task\_stack**  
Task stack size
- int **task\_core**  
CPU core number (0 or 1) where decoder task in running
- int **task\_prio**  
Task priority (based on freeRTOS priority)
- bool **stack\_in\_ext**  
Try to allocate stack in external memory

### Macros

**AMR\_DECODER\_TASK\_STACK\_SIZE**

**AMR\_DECODER\_TASK\_CORE**

**AMR\_DECODER\_TASK\_PRIO**

**AMR\_DECODER\_RINGBUFFER\_SIZE**

**DEFAULT\_AMR\_DECODER\_CONFIG** ()

### API Reference - AMR-NB Encoder

#### Header File

- `components/esp-adf-libs/esp_codec/include/codecs/amrnb_encoder.h`

#### Functions

esp\_err\_t **amrnb\_encoder\_set\_bitrate** (*audio\_element\_handle\_t* self, *amrnb\_encoder\_bitrate\_t* bitrate\_mode)

Set AMRNB encoder bitrate.

#### Parameters

- **self** – Audio element handle
- **bitrate\_mode** – Bitrate choose, value from `amrnb_encoder_bitrate_t`

**Returns**

ESP\_OK ESP\_FAIL

*audio\_element\_handle\_t* **amrnb\_encoder\_init** (*amrnb\_encoder\_cfg\_t* \*config)

Create an Audio Element handle to encode incoming AMRNB data.

**Parameters** **config** – The configuration**Returns** The audio element handle**Structures**struct **amrnb\_encoder\_cfg\_t**

AMRNB Encoder configurations.

**Public Members**int **out\_rb\_size**

Size of output ringbuffer

int **task\_stack**

Task stack size

int **task\_core**

Task running in core (0 or 1)

int **task\_prio**

Task priority (based on freeRTOS priority)

*amrnb\_encoder\_bitrate\_t* **bitrate\_mode**

AMRNB Encoder bitrate choose

bool **contain\_amrnb\_header**

Choose to contain amrnb header in amrnb encoder whether or not (true or false, true means choose to contain amrnb header)

bool **stack\_in\_ext**

Try to allocate stack in external memory

### Macros

**AMRNB\_ENCODER\_TASK\_STACK**

**AMRNB\_ENCODER\_TASK\_CORE**

**AMRNB\_ENCODER\_TASK\_PRIO**

**AMRNB\_ENCODER\_RINGBUFFER\_SIZE**

**DEFAULT\_AMRNB\_ENCODER\_CONFIG()**

### Enumerations

enum **amrnb\_encoder\_bitrate\_t**

Enum of AMRNB Encoder bitrate choose.

*Values:*

enumerator **AMRNB\_ENC\_BITRATE\_UNKNOW**

Invalid mode

enumerator **AMRNB\_ENC\_BITRATE\_MR475**

enumerator **AMRNB\_ENC\_BITRATE\_MR515**

enumerator **AMRNB\_ENC\_BITRATE\_MR59**

enumerator **AMRNB\_ENC\_BITRATE\_MR67**

enumerator **AMRNB\_ENC\_BITRATE\_MR74**

enumerator **AMRNB\_ENC\_BITRATE\_MR795**

enumerator **AMRNB\_ENC\_BITRATE\_MR102**

enumerator **AMRNB\_ENC\_BITRATE\_MR122**

enumerator **AMRNB\_ENC\_BITRATE\_MRDTX**

enumerator **AMRNB\_ENC\_BITRATE\_N\_MODES**



## API Reference - AMR-WB Encoder

### Header File

- components/esp-adf-libs/esp\_codec/include/codec/amrwb\_encoder.h

### Functions

esp\_err\_t **amrwb\_encoder\_set\_bitrate** (*audio\_element\_handle\_t* self, *amrwb\_encoder\_bitrate\_t* bitrate\_mode)

Set AMRWB encoder bitrate.

#### Parameters

- **self** – Audio element handle
- **bitrate\_mode** – Bitrate choose, value from *amrwb\_encoder\_bitrate\_t*

#### Returns

ESP\_OK ESP\_FAIL

*audio\_element\_handle\_t* **amrwb\_encoder\_init** (*amrwb\_encoder\_cfg\_t* \*config)

Create an Audio Element handle to encode incoming amrwb data.

**Parameters** **config** – The configuration

**Returns** The audio element handle

### Structures

struct **amrwb\_encoder\_cfg\_t**

AMRWB Encoder configurations.

#### Public Members

int **out\_rb\_size**

Size of output ringbuffer

int **task\_stack**

Task stack size

int **task\_core**

Task running in core (0 or 1)

int **task\_prio**

Task priority (based on freeRTOS priority)

*amrwb\_encoder\_bitrate\_t* **bitrate\_mode**

AMRWB Encoder bitrate choose

bool **contain\_amrwb\_header**

Choose to contain amrwb header in amrwb encoder whether or not (true or false, true means choose to contain amrwb header)

bool **stack\_in\_ext**

Try to allocate stack in external memory

### Macros

**AMRWB\_ENCODER\_TASK\_STACK**

**AMRWB\_ENCODER\_TASK\_CORE**

**AMRWB\_ENCODER\_TASK\_PRIO**

**AMRWB\_ENCODER\_RINGBUFFER\_SIZE**

**DEFAULT\_AMRWB\_ENCODER\_CONFIG()**

### Enumerations

enum **amrwb\_encoder\_bitrate\_t**

Enum of AMRWB Encoder bitrate choose.

*Values:*

enumerator **AMRWB\_ENC\_BITRATE\_MDNONE**

Invalid mode

enumerator **AMRWB\_ENC\_BITRATE\_MD66**

6.60kbps

enumerator **AMRWB\_ENC\_BITRATE\_MD885**

8.85kbps

enumerator **AMRWB\_ENC\_BITRATE\_MD1265**

12.65kbps

enumerator **AMRWB\_ENC\_BITRATE\_MD1425**

14.25kbps

enumerator **AMRWB\_ENC\_BITRATE\_MD1585**

15.85bps

enumerator **AMRWB\_ENC\_BITRATE\_MD1825**  
18.25bps

enumerator **AMRWB\_ENC\_BITRATE\_MD1985**  
19.85kbps

enumerator **AMRWB\_ENC\_BITRATE\_MD2305**  
23.05kbps

enumerator **AMRWB\_ENC\_BITRATE\_MD2385**  
23.85kbps>

enumerator **AMRWB\_ENC\_BITRATE\_N\_MODES**  
Invalid mode

### 2.4.3 FLAC Decoder

Decode an audio data stream provided in FLAC format.

#### API Reference

##### Header File

- `components/esp-adf-libs/esp_codec/include/codecs/flac_decoder.h`

##### Functions

*audio\_element\_handle\_t* **flac\_decoder\_init** (*flac\_decoder\_cfg\_t* \*config)

Create an Audio Element handle to decode incoming FLAC data.

**Parameters** `config` – The configuration

**Returns** The audio element handle

##### Structures

struct **flac\_decoder\_cfg\_t**

FLAC Decoder configuration.

### Public Members

int **out\_rb\_size**

Size of output ringbuffer

int **task\_stack**

Task stack size

int **task\_core**

CPU core number (0 or 1) where decoder task in running

int **task\_prio**

Task priority (based on freeRTOS priority)

bool **stack\_in\_ext**

Try to allocate stack in external memory

### Macros

**FLAC\_DECODER\_TASK\_STACK\_SIZE**

**FLAC\_DECODER\_TASK\_CORE**

**FLAC\_DECODER\_TASK\_PRIO**

**FLAC\_DECODER\_RINGBUFFER\_SIZE**

**DEFAULT\_FLAC\_DECODER\_CONFIG** ()

## 2.4.4 MP3 Decoder

Decode an audio data stream provided in MP3 format.

### Application Examples

Implementation of this API is demonstrated in the following examples:

- `get-started/play_mp3_control`
- `player/pipeline_play_sdcard_music`
- `player/pipeline_play_mp3_with_dac_or_pwm`

## API Reference

### Header File

- components/esp-adf-libs/esp\_codec/include/codec/mp3\_decoder.h

### Functions

*audio\_element\_handle\_t* **mp3\_decoder\_init** (*mp3\_decoder\_cfg\_t* \*config)

Create an Audio Element handle to decode incoming MP3 data.

**Parameters** *config* – The configuration

**Returns** The audio element handle

const *esp\_id3\_info\_t* \***mp3\_decoder\_get\_id3\_info** (*audio\_element\_handle\_t* self)

Get ID3 information.

**Parameters** *self* – The audio element handle

**Returns** *esp\_id3\_info\_t*: success NULL: ID3 is not exist or memory allocation failed.

### Structures

struct **mp3\_decoder\_cfg\_t**

Mp3 Decoder configuration.

#### Public Members

int **out\_rb\_size**

Size of output ringbuffer

int **task\_stack**

Task stack size

int **task\_core**

CPU core number (0 or 1) where decoder task in running

int **task\_prio**

Task priority (based on freeRTOS priority)

bool **stack\_in\_ext**

Try to allocate stack in external memory

bool **id3\_parse\_enable**

True: parse ID3. False: Don't parse ID3

### Macros

`MP3_DECODER_TASK_STACK_SIZE`

`MP3_DECODER_TASK_CORE`

`MP3_DECODER_TASK_PRIO`

`MP3_DECODER_RINGBUFFER_SIZE`

`DEFAULT_MP3_DECODER_CONFIG()`

### 2.4.5 OGG Decoder

Decode an audio data stream provided in OGG format.

#### API Reference

##### Header File

- `components/esp-adf-libs/esp_codec/include/codec/ogg_decoder.h`

##### Functions

*audio\_element\_handle\_t* `ogg_decoder_init` (*ogg\_decoder\_cfg\_t* \*config)

Create an Audio Element handle to decode incoming OGG data.

**Parameters** `config` – The configuration

**Returns** The audio element handle

##### Structures

struct `ogg_decoder_cfg_t`

OGG Decoder configuration.

##### Public Members

int `out_rb_size`

Size of output ringbuffer

int `task_stack`

Task stack size

int **task\_core**  
CPU core number (0 or 1) where decoder task in running

int **task\_prio**  
Task priority (based on freeRTOS priority)

bool **stack\_in\_ext**  
Try to allocate stack in external memory

## Macros

**OGG\_DECODER\_TASK\_STACK\_SIZE**

**OGG\_DECODER\_TASK\_CORE**

**OGG\_DECODER\_TASK\_PRIO**

**OGG\_DECODER\_RINGBUFFER\_SIZE**

**DEFAULT\_OGG\_DECODER\_CONFIG()**

## 2.4.6 OPUS Decoder

Decode an audio data stream provided in OPUS format.

### API Reference

#### Header File

- [components/esp-adf-libs/esp\\_codec/include/codec/opus\\_decoder.h](#)

#### Functions

*audio\_element\_handle\_t* **decoder\_opus\_init** (*opus\_decoder\_cfg\_t* \*config)

Create an Audio Element handle to decode incoming OPUS data.

**Parameters** **config** – The configuration

**Returns** The audio element handle

### Structures

struct **opus\_decoder\_cfg\_t**  
OPUS Decoder configuration.

#### Public Members

int **out\_rb\_size**  
Size of output ringbuffer

int **task\_stack**  
Task stack size

int **task\_core**  
CPU core number (0 or 1) where decoder task is running

int **task\_prio**  
Task priority (based on freeRTOS priority)

bool **stack\_in\_ext**  
Try to allocate stack in external memory

### Macros

**OPUS\_DECODER\_TASK\_STACK\_SIZE**

**OPUS\_DECODER\_TASK\_CORE**

**OPUS\_DECODER\_TASK\_PRIO**

**OPUS\_DECODER\_RINGBUFFER\_SIZE**

**DEFAULT\_OPUS\_DECODER\_CONFIG** ( )

## 2.4.7 WAV Decoder and Encoder

Decode and encode an audio data stream from / to WAV format.



## Application Examples

Implementation of this API is demonstrated in the following examples:

- `player/pipeline_play_sdcard_music`
- `recorder/pipeline_wav_amr_sdcard`

## API Reference - Decoder

### Header File

- `components/esp-adf-libs/esp_codec/include/codec/wav_decoder.h`

### Functions

*audio\_element\_handle\_t* **wav\_decoder\_init** (*wav\_decoder\_cfg\_t* \*config)

Create an Audio Element handle to decode incoming WAV data.

**Parameters** `config` – The configuration

**Returns** The audio element handle

### Structures

struct **wav\_decoder\_cfg\_t**

brief WAV Decoder configurations

#### Public Members

int **out\_rb\_size**

Size of output ringbuffer

int **task\_stack**

Task stack size

int **task\_core**

Task running in core (0 or 1)

int **task\_prio**

Task priority (based on freeRTOS priority)

bool **stack\_in\_ext**

Try to allocate stack in external memory

### Macros

`WAV_DECODER_TASK_STACK`

`WAV_DECODER_TASK_CORE`

`WAV_DECODER_TASK_PRIO`

`WAV_DECODER_RINGBUFFER_SIZE`

`DEFAULT_WAV_DECODER_CONFIG()`

### API Reference - Encoder

#### Header File

- `components/esp-adf-libs/esp_codec/include/codec/wav_encoder.h`

#### Functions

*audio\_element\_handle\_t* **wav\_encoder\_init** (*wav\_encoder\_cfg\_t* \*config)

Create a handle to an Audio Element to encode incoming data using WAV format.

**Parameters** `config` – The configuration

**Returns** The audio element handle

#### Structures

struct **wav\_encoder\_cfg\_t**

WAV Encoder configurations.

#### Public Members

int **out\_rb\_size**

Size of output ringbuffer

int **task\_stack**

Task stack size

int **task\_core**

Task running in core (0 or 1)

int **task\_prio**  
Task priority (based on freeRTOS priority)

bool **stack\_in\_ext**  
Try to allocate stack in external memory

## Macros

**WAV\_ENCODER\_TASK\_STACK**

**WAV\_ENCODER\_TASK\_CORE**

**WAV\_ENCODER\_TASK\_PRIO**

**WAV\_ENCODER\_RINGBUFFER\_SIZE**

**DEFAULT\_WAV\_ENCODER\_CONFIG()**

## 2.5 Audio Processing

There are couple of options implemented in the ESP-ADF to modify contents of an audio stream:

- Combine contents of two audio streams using *Downmix*
- Apply ten band *Equalizer*
- Change audio sampling frequency and convert between single and two channel with *Resample Filter*
- Modify pitch and speed of the stream using *Sonic*

Please refer to description of respective APIs below.

### 2.5.1 Downmix

This API is intended for mixing of two audio files (streams), defined as the base audio file and the newcomer audio file, into one output audio file.

The newcomer audio file will be downmixed into the base audio file with individual gains applied to each file.

The number of channel(s) of the output audio file will be the same with that of the base audio file. The number of channel(s) of the newcomer audio file will also be changed to the same with the base audio file, if it is different from that of the base audio file.

The downmix process has 3 states:

- Bypass Downmixing – Only the base audio file will be processed;
- Switch on Downmixing – The base audio file and the target audio file will first enter the transition period, during which the gains of these two files will be changed from the original level to the target level; then enter the stable period, sharing a same target gain;

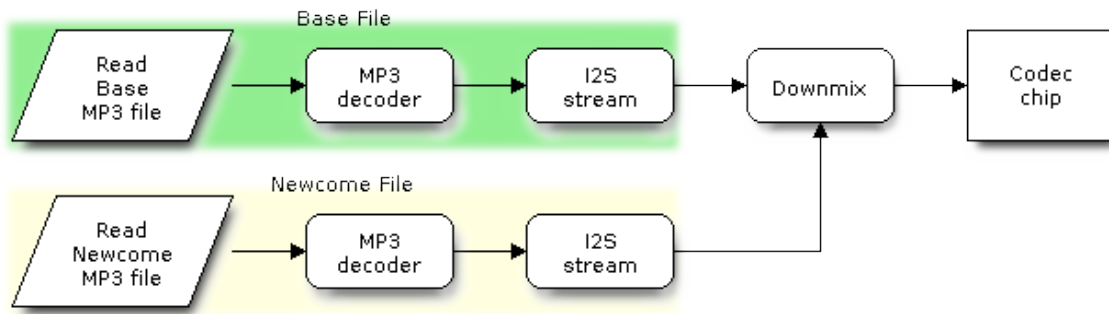


Fig. 4: Illustration of Downmixing Process

- Switch off Downmixing – The base audio file and the target audio file will first enter the transition period, during which the gains of these two files will be changed back to their original levels; then enter the stable period, with their original gains, respectively. After that, the downmix process enters the bypass state.

Note that, the sample rates of the base audio file and the newcome audio file must be the same, otherwise an error occurs.

## Application Example

Implementation of this API is demonstrated in [advanced\\_examples/downmix\\_pipeline](#) example.

## API Reference

### Header File

- [components/esp-adf-libs/esp\\_codec/include/codec/downmix.h](#)

### Functions

void **downmix\_set\_input\_rb\_timeout** (*audio\_element\_handle\_t* self, int ticks\_to\_wait, int index)

Sets the downmix timeout.

#### Parameters

- **self** – Audio element handle
- **ticks\_to\_wait** – Input ringbuffer timeout
- **index** – The index of multi input ringbuffer

void **downmix\_set\_input\_rb** (*audio\_element\_handle\_t* self, *ringbuf\_handle\_t* rb, int index)

Sets the downmix input ringbuffer. refer to `ringbuf.h`

#### Parameters

- **self** – Audio element handle
- **rb** – Handle of ringbuffer
- **index** – The index of multi input ringbuffer.

`esp_err_t downmix_set_output_type` (*audio\_element\_handle\_t* self, `esp_downmix_output_type_t` output\_type)

Set channel mode of output data. Only supported mono and dual.

#### Parameters

- **self** – Audio element handle
- **output\_type** – Down-mixer output type.

#### Returns

- `ESP_OK` on success
- `ESP_ERR_INVALID_ARG` invalid arguments

`esp_err_t downmix_set_work_mode` (*audio\_element\_handle\_t* self, `esp_downmix_work_mode_t` mode)

Sets BYPASS, ON or OFF status of down-mixer.

#### Parameters

- **self** – Audio element handle
- **mode** – Down-mixer work mode.

#### Returns

- `ESP_OK` on success
- `ESP_ERR_INVALID_ARG` invalid arguments

`esp_err_t downmix_set_out_ctx_info` (*audio\_element\_handle\_t* self, `esp_downmix_out_ctx_type_t` out\_ctx)

Passes content of per channel output stream by down-mixer.

#### Parameters

- **self** – Audio element handle
- **out\_ctx** – Content of output stream.

#### Returns

- `ESP_OK` on success
- `ESP_ERR_INVALID_ARG` invalid arguments

`esp_err_t downmix_set_source_stream_info` (*audio\_element\_handle\_t* self, int rate, int ch, int index)

Sets the sample rate and the number of channels of input stream to be processed.

#### Parameters

- **self** – Audio element handle
- **rate** – Sample rate of the input stream
- **ch** – Channel number of the input stream. Only supported mono and dual.
- **index** – The index of input stream. The index must be in [0, `SOURCE_NUM_MAX` - 1] range.

#### Returns

- `ESP_OK` on success
- `ESP_ERR_INVALID_ARG` invalid arguments

`esp_err_t downmix_set_gain_info (audio_element_handle_t self, float *gain, int index)`

Sets the audio gain to be processed.

### Parameters

- **self** – Audio element handle
- **gain** – The gain array of downmix which the array size is 2 and the gain data range is [-100, 100], unit: dB.
- **index** – The index of input stream. The index must be in [0, SOURCE\_NUM\_MAX - 1] range.

### Returns

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG invalid arguments

`esp_err_t downmix_set_transit_time_info (audio_element_handle_t self, int transit_time, int index)`

Sets the audio `transit_time` to be processed.

### Parameters

- **self** – Audio element handle
- **transit\_time** – The reset value of `transit_time`
- **index** – The index of input stream. The index must be in [0, SOURCE\_NUM\_MAX - 1] range

### Returns

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG invalid arguments

`esp_err_t source_info_init (audio_element_handle_t self, esp_downmix_input_info_t *source_info)`

Initializes information of the source streams for downmixing.

### Parameters

- **self** – Audio element handle
- **source\_info** – The information array of source streams

### Returns

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG invalid arguments

`audio_element_handle_t downmix_init (downmix_cfg_t *config)`

Initializes the Audio Element handle for downmixing.

**Parameters** `config` – the configuration

**Returns** The audio element handler

## Structures

struct **downmix\_cfg\_t**

Downmix configuration.

### Public Members

esp\_downmix\_info\_t **downmix\_info**

Downmix information

int **max\_sample**

The number of samples per downmix processing

int **out\_rb\_size**

Size of ring buffer

int **task\_stack**

Size of task stack

int **task\_core**

Task running in core...

int **task\_prio**

Task priority (based on the FreeRTOS priority)

bool **stack\_in\_ext**

Try to allocate stack in external memory

## Macros

**DOWNMIX\_TASK\_STACK**

**DOWNMIX\_TASK\_CORE**

**DOWNMIX\_TASK\_PRIO**

**DOWNMIX\_RINGBUFFER\_SIZE**

**DM\_BUF\_SIZE**

**DEFAULT\_DOWNMIX\_CONFIG** ()

### 2.5.2 Equalizer

Provided in this API equalizer supports:

- fixed number of ten (10) bands;
- four sample rates: 11025 Hz, 22050 Hz, 44100 Hz and 48000 Hz.

The center frequencies of bands are shown in table below.

Band Index	0	1	2	3	4	5	6	7	8	9
Frequency	31 Hz	62 Hz	125 Hz	250 Hz	500 Hz	1 kHz	2 kHz	4 kHz	8 kHz	16 kHz

Default gain of each band is -13 dB. To set the gains of all bands use structure `equalizer_cfg`. To set the gain of individual band use function `equalizer_set_gain_info()`.

#### Application Example

Implementation of this API is demonstrated in the `audio_processing/pipeline_equalizer` example.

#### API Reference

##### Header File

- `components/esp-adf-libs/esp_codec/include/codec/equalizer.h`

##### Functions

`esp_err_t equalizer_set_info` (`audio_element_handle_t` self, int rate, int ch)

Set the audio sample rate and the number of channels to be processed by the equalizer.

##### Parameters

- **self** – Audio element handle
- **rate** – Audio sample rate
- **ch** – Audio channel

##### Returns

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG invalid arguments

`esp_err_t equalizer_set_gain_info` (`audio_element_handle_t` self, int index, int value\_gain, bool is\_channels\_gain\_equal)

Set the audio gain to be processed by the equalizer.

##### Parameters

- **self** – Audio element handle
- **index** – The position of center frequencies of equalizer. If channel is mono, the index range is [0, 9]; If channel is stereo and `is_channels_gain_equal` is true, the index range is [0, 9]; If channel is stereo and `is_channels_gain_equal` is false, the index range is [0, 19];





### Public Members

int **samplerate**

Audio sample rate. Supported samplerate: 11025, 22050, 44100, 48000, unit: Hz

int **channel**

Number of audio channels. Supported channel: mono, stereo

int **\*set\_gain**

Equalizer gain

int **out\_rb\_size**

Size of output ring buffer

int **task\_stack**

Task stack size

int **task\_core**

Task running in core...

int **task\_prio**

Task priority

bool **stack\_in\_ext**

Try to allocate stack in external memory

### Macros

**EQUALIZER\_TASK\_STACK**

**EQUALIZER\_TASK\_CORE**

**EQUALIZER\_TASK\_PRIO**

**EQUALIZER\_RINGBUFFER\_SIZE**

**DEFAULT\_EQUALIZER\_CONFIG()**



### API Reference

#### Header File

- `components/esp-adf-libs/esp_codec/include/codec/filter_resample.h`

#### Functions

`esp_err_t rsp_filter_set_src_info` (*audio\_element\_handle\_t* self, int src\_rate, int src\_ch)

Set the source audio sample rate and the number of channels to be processed by the resample. If need change bits or not ensure source data infomation, please use `rsp_filter_change_src_info` to instead this function.

##### Parameters

- **self** – Audio element handle
- **src\_rate** – The sample rate of stream data
- **src\_ch** – The number channels of stream data

##### Returns

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG invalid arguments

`esp_err_t rsp_filter_change_src_info` (*audio\_element\_handle\_t* self, int src\_rate, int src\_ch, int src\_bit)

Set the source audio sample rate, the number of channels and bits per sample to be processed by the resample.

##### Parameters

- **self** – Audio element handle
- **src\_rate** – The sample rate of stream data
- **src\_ch** – The number channels of stream data
- **src\_bit** – The bit per sample of stream data

##### Returns

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG invalid arguments

*audio\_element\_handle\_t* `rsp_filter_init` (*rsp\_filter\_cfg\_t* \*config)

Create an Audio Element handle to resample incoming data.

Depending on configuration, there are upsampling, downsampling, as well as converting data between mono and dual.

- If the `esp_resample_mode_t` is `RESAMPLE_DECODE_MODE`, `src_rate` and `src_ch` will be fetched from `audio_element_getinfo`.
- If the `esp_resample_mode_t` is `RESAMPLE_ENCODE_MODE`, `src_rate`, `src_ch`, `dest_rate` and `dest_ch` must be configured.

**Parameters** `config` – The configuration

**Returns** The audio element handler

## Structures

struct **rsp\_filter\_cfg\_t**

Resample Filter Configuration.

### Public Members

int **src\_rate**

The sampling rate of the source PCM file (in Hz)

int **src\_ch**

The number of channel(s) of the source PCM file (Mono=1, Dual=2)

int **dest\_rate**

The sampling rate of the destination PCM file (in Hz)

int **dest\_bits**

The bit for sample of the destination PCM data. Currently, supported bit width :16 bits.

int **dest\_ch**

The number of channel(s) of the destination PCM file (Mono=1, Dual=2)

int **src\_bits**

The bit for sample of the source PCM data. Currently, supported bit width :8bits 16 bits 24bits 32bits.

esp\_resample\_mode\_t **mode**

The resampling mode (the encoding mode or the decoding mode). For decoding mode, input PCM length is constant; for encoding mode, output PCM length is constant.

int **max\_indata\_bytes**

The maximum buffer size of the input PCM (in bytes)

int **out\_len\_bytes**

The buffer length of the output stream data. This parameter must be configured in encoding mode.

esp\_resample\_type\_t **type**

The resampling type (Automatic, Upsampling and Downsampling)

int **complexity**

Indicates the complexity of the resampling. This parameter is only valid when a FIR filter is used. Range:[1, 5]; 1 indicates the lowest complexity, which means the accuracy is the lowest and the speed is the fastest; Meanwhile, 5 indicates the highest complexity, which means the accuracy is the highest and the speed is the slowest.If user set complexity less than 1, complexity can be set 1. If user set complexity more than 5, complexity can be set 5.

int **down\_ch\_idx**

Indicates the channel that is selected (the right channel or the left channel). This parameter is only valid when the complexity parameter is set to 0 and the number of channel(s) of the input file has changed from dual to mono.

esp\_rsp\_prefer\_type\_t **prefer\_flag**

The select flag about lesser CPU usage or lower INRAM usage, refer to esp\_resample.h

int **out\_rb\_size**

Output ringbuffer size

int **task\_stack**

Task stack size

int **task\_core**

Task running on core

int **task\_prio**

Task priority

bool **stack\_in\_ext**

Try to allocate stack in external memory

### Macros

**RSP\_FILTER\_BUFFER\_BYTE**

**RSP\_FILTER\_TASK\_STACK**

**RSP\_FILTER\_TASK\_CORE**

**RSP\_FILTER\_TASK\_PRIO**

**RSP\_FILTER\_RINGBUFFER\_SIZE**

**DEFAULT\_RESAMPLE\_FILTER\_CONFIG()**

## 2.5.4 Sonic

The Sonic component acts as a multidimensional filter that lets you adjust audio parameters of a WAV stream. This functionality may be useful to e.g. increase playback speed of an audio recording by a user selectable rate.

The following parameters can be adjusted:

- speed
- pitch
- interpolation type

The adjustments of the first two parameters are represented by *float* values that provide the rate of adjustment. For example, to increase the speed of an audio sample by 2 times, call `sonic_set_pitch_and_speed_info(el, 1.0, 2.0)`. To keep the speed as it is, call `sonic_set_pitch_and_speed_info(el, 1.0, 1.0)`.

For the *interpolation type* you may select either faster but less accurate linear interpolation, or slower but more accurate FIR interpolation.

### Application Example

Implementation of this API is demonstrated in `audio_processing/pipeline_sonic` example.

### API Reference

#### Header File

- `components/esp-adf-libs/esp_codec/include/codec/audio_sonic.h`

#### Functions

`esp_err_t sonic_set_info(audio_element_handle_t self, int rate, int ch)`

Sets the audio sample rate and the number of channels to be processed by the sonic.

##### Parameters

- **self** – Audio element handle
- **rate** – The sample rate of stream data
- **ch** – The channel numbers of stream data

##### Returns

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG invalid arguments

`esp_err_t sonic_set_pitch_and_speed_info(audio_element_handle_t self, float pitch, float speed)`

Sets the audio pitch and speed to be processed by the sonic.

##### Parameters

- **self** – Audio element handle
- **pitch** – Scale factor of pitch of audio file. 0 means the original pitch. The range is [0.2 4.0].
- **speed** – Scale factor of speed of audio file. 0 means the original speed. The range is [0.1 8.0].

### Returns

- ESP\_OK on success
- ESP\_ERR\_INVALID\_ARG invalid arguments

*audio\_element\_handle\_t* **sonic\_init** (*sonic\_cfg\_t* \*config)

Creates an Audio Element handle for sonic.

**Parameters** **config** – The sonic configuration

**Returns** The audio element handler

## Structures

struct **sonic\_info\_t**

Information on audio file and configuration parameters required by sonic to process the file.

### Public Members

int **samplerate**

Audio file sample rate (in Hz)

int **channel**

Number of audio file channels (Mono=1, Dual=2)

int **resample\_linear\_interpolate**

Flag of using simple linear interpolation. 1 indicates using simple linear interpolation. 0 indicates not using simple linear interpolation.

float **pitch**

Scale factor of pitch of audio file. If the value of 'pitch' is 0.3, the pitch of audio file processed by sonic is lower than the original. If the value of 'pitch' is 1.3, the pitch of audio file processed by sonic is 30% higher than the original.

float **speed**

Scale factor of speed of audio file. If the value of 'speed' is 0.3, the speed of audio file processed by sonic is 70% slower than the original. If the value of 'speed' is 1.3, the speed of audio file processed by sonic is 30% faster than the original.

struct **sonic\_cfg\_t**

Sonic configuration.



## Public Members

*sonic\_info\_t* **sonic\_info**

Information of sonic

int **out\_rb\_size**

Size of output ring buffer

int **task\_stack**

Task stack size

int **task\_core**

Task running in core

int **task\_prio**

Task priority

bool **stack\_in\_ext**

Try to allocate stack in external memory

## Macros

**SONIC\_SET\_VALUE\_FOR\_INITIALIZATION**

**SONIC\_TASK\_STACK**

**SONIC\_TASK\_CORE**

**SONIC\_TASK\_PRIO**

**SONIC\_RINGBUFFER\_SIZE**

**DEFAULT\_SONIC\_CONFIG** ()

## 2.6 Services

A service is a software implementation of specific product functions, such as input keys, network configuration management, and battery check. Each service is also an abstraction of hardware. For example, the input key service supports ADC keys and GPIO keys. Services can be reused on different products, and high-level APIs and events allow easy and convenient product development.

For details please refer to descriptions listed below.

### 2.6.1 Bluetooth Service

The Bluetooth service is dedicated to interface with Bluetooth devices and provides support for the following protocols:

- HFP (Hands-Free Profile): remotely controlling the mobile phone by the Hands-Free device and the voice connections between them
- A2DP (Advanced Audio Distribution Profile): implementing streaming of multimedia audio using a Bluetooth connection
- AVRCP (Audio Video Remote Control Profile): used together with A2DP for remote control of devices such as headphones, car audio systems, or speakers

#### Application Example

Implementation of this API is demonstrated in the following example:

- `player/pipeline_bt_sink`

#### Header File

- `components/bluetooth_service/include/bluetooth_service.h`

#### Functions

`esp_err_t bluetooth_service_start (bluetooth_service_cfg_t *config)`

Initialize and start the Bluetooth service. This function can only be called for one time, and `bluetooth_service_destroy` must be called after use.

**Parameters** `config` – The configuration

**Returns**

- `ESP_OK`
- `ESP_FAIL`

`audio_element_handle_t bluetooth_service_create_stream ()`

Create Bluetooth stream, it is valid when Bluetooth service has started. The returned audio stream compatible with existing audio streams and can be used with the Audio Pipeline.

**Returns** The Audio Element handle

`esp_periph_handle_t bluetooth_service_create_periph ()`

Create Bluetooth peripheral, it is valid when Bluetooth service has started. The returned bluetooth peripheral compatible with existing peripherals and can be used with the ESP Peripherals.

**Returns** The Peripheral handle

`esp_err_t periph_bluetooth_play (esp_periph_handle_t periph)`

Send the AVRCP passthrough command (PLAY) to the Bluetooth device.

**Parameters** `periph` – [in] The periph

**Returns**

- `ESP_OK`

- ESP\_FAIL

esp\_err\_t **periph\_bluetooth\_pause** (*esp\_periph\_handle\_t* periph)

Send the AVRC passthrough command (PAUSE) to the Bluetooth device.

**Parameters** **periph** – [in] The periph

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **periph\_bluetooth\_stop** (*esp\_periph\_handle\_t* periph)

Send the AVRC passthrough command (STOP) to the Bluetooth device.

**Parameters** **periph** – [in] The periph

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **periph\_bluetooth\_next** (*esp\_periph\_handle\_t* periph)

Send the AVRC passthrough command (NEXT) to the Bluetooth device.

**Parameters** **periph** – [in] The periph

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **periph\_bluetooth\_prev** (*esp\_periph\_handle\_t* periph)

Send the AVRC passthrough command (PREV) to the Bluetooth device.

**Parameters** **periph** – [in] The periph

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **periph\_bluetooth\_rewind** (*esp\_periph\_handle\_t* periph)

Send the AVRC passthrough command (REWIND) to the Bluetooth device.

**Parameters** **periph** – [in] The periph

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **periph\_bluetooth\_fast\_forward** (*esp\_periph\_handle\_t* periph)

Send the AVRC passthrough command (FAST FORWARD) to the Bluetooth device.

**Parameters** **periph** – [in] The periph

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **periph\_bluetooth\_discover** (*esp\_periph\_handle\_t* periph)

Start device discovery.

**Parameters** **periph** – [in] The periph

**Returns**

- ESP\_OK : Succeed
- ESP\_ERR\_INVALID\_STATE: if bluetooth stack is not yet enabled
- ESP\_ERR\_INVALID\_ARG: if invalid parameters are provided
- ESP\_FAIL: others

esp\_err\_t **periph\_bluetooth\_cancel\_discover** (*esp\_periph\_handle\_t* periph)

Cancel device discovery.

**Parameters** **periph** – [in] The periph

**Returns**

- ESP\_OK : Succeed
- ESP\_ERR\_INVALID\_STATE: if bluetooth stack is not yet enabled
- ESP\_FAIL: others

esp\_err\_t **periph\_bluetooth\_connect** (*esp\_periph\_handle\_t* periph, *bluetooth\_addr\_t* remote\_bda)

Connect remote Device.

**Parameters**

- **periph** – [in] The periph
- **remote\_bda** – [in] remote Bluetooth device address

**Returns**

- ESP\_OK : Succeed
- ESP\_ERR\_INVALID\_STATE: if bluetooth stack is not yet enabled
- ESP\_FAIL: others

esp\_err\_t **bluetooth\_service\_destroy** ()

Destroy and cleanup bluetooth service, this function must be called after destroying the Bluetooth Stream and Bluetooth Peripheral created by `bluetooth_service_create_stream` and `bluetooth_service_create_periph`

**Returns**

- ESP\_OK
- ESP\_FAIL

int **periph\_bluetooth\_get\_a2dp\_sample\_rate** ()

Get a2dp sample rate.

**Returns**

- sample rate

## Structures

struct **bluetooth\_service\_user\_cb\_t**

brief Bluetooth service user callback

### Public Members

esp\_a2d\_cb\_t **user\_a2d\_cb**

callback for a2dp

esp\_a2d\_sink\_data\_cb\_t **user\_a2d\_sink\_data\_cb**

callback for a2dp sink data

esp\_a2d\_source\_data\_cb\_t **user\_a2d\_source\_data\_cb**

callback for a2dp source data

esp\_avrc\_ct\_cb\_t **user\_avrc\_ct\_cb**

callback for avrc ct

struct **bluetooth\_service\_cfg\_t**

brief Bluetooth service configuration

### Public Members

const char \***device\_name**

Bluetooth local device name

const char \***remote\_name**

Bluetooth remote device name

*bluetooth\_service\_mode\_t* **mode**

Bluetooth working mode

*bluetooth\_service\_user\_cb\_t* **user\_callback**

Bluetooth user callback

### Macros

**ESP\_A2DP\_SAMPLE\_RATE**

**BLUETOOTH\_ADDR\_LEN**

brief Bluetooth address length

### Type Definitions

typedef uint8\_t **bluetooth\_addr\_t**[BLUETOOTH\_ADDR\_LEN]

brief Bluetooth device address

### Enumerations

enum **bluetooth\_service\_mode\_t**

brief Bluetooth service working mode

*Values:*

enumerator **BLUETOOTH\_A2DP\_SINK**

A2DP Bluetooth sink audio, ESP32 will receive audio data from other bluetooth devices

enumerator **BLUETOOTH\_A2DP\_SOURCE**

A2DP Bluetooth source audio, ESP32 can send audio data to other bluetooth devices

### Header File

- components/bluetooth\_service/include/bt\_keycontrol.h

### Header File

- components/bluetooth\_service/include/hfp\_stream.h

### Header File

- components/bluetooth\_service/include/a2dp\_stream.h

## 2.6.2 Input Key Service

The input key service provides GPIO input interrupts and ADC key functions in the form of events. For what common key functions are defined by the events for audio products, see *input\_key\_user\_id\_t*.

### Application Example

Implementation of this API is demonstrated in the following example:

- checks/check\_board\_buttons
- player/pipeline\_sdcard\_mp3\_control
- protocols/voip

### Header File

- components/input\_key\_service/include/input\_key\_service.h

### Functions

*periph\_service\_handle\_t* **input\_key\_service\_create** (*input\_key\_service\_cfg\_t* \*input\_key\_config)

Initialize and start the input key service.

**Parameters** *input\_key\_config* – Configuration of input key service

**Returns** NULL failed others input key service handle

*periph\_service\_state\_t* **get\_input\_key\_service\_state** (*periph\_service\_handle\_t* input\_handle)

Get the state of input key service.

**Parameters** *input\_handle* – The handle of input key service

**Returns** state of input key service

esp\_err\_t **input\_key\_service\_add\_key** (*periph\_service\_handle\_t* input\_key\_handle, *input\_key\_service\_info\_t* \*input\_key\_info, int add\_key\_num)

Add input key's information to service list.

**Parameters**

- *input\_key\_handle* – handle of service
- *input\_key\_info* – input key's information
- *add\_key\_num* – number of keys

**Returns** ESP\_OK success ESP\_FAIL failed

### Structures

struct **input\_key\_service\_info\_t**  
input key's information

#### Public Members

*esp\_periph\_id\_t* **type**  
ID of peripherals

int **user\_id**  
The key's user id

int **act\_id**  
The key's action id

struct **input\_key\_service\_cfg\_t**  
input key's configuration

#### Public Members

*periph\_service\_config\_t* **based\_cfg**  
Peripheral service configuration

*esp\_periph\_set\_handle\_t* **handle**  
Peripheral set handle

### Macros

**INPUT\_KEY\_SERVICE\_TASK\_STACK\_SIZE**

**INPUT\_KEY\_SERVICE\_TASK\_PRIORITY**

**INPUT\_KEY\_SERVICE\_TASK\_ON\_CORE**

**INPUT\_KEY\_SERVICE\_DEFAULT\_CONFIG()**



## Enumerations

enum **input\_key\_service\_action\_id\_t**

input key action id

*Values:*

enumerator **INPUT\_KEY\_SERVICE\_ACTION\_UNKNOWN**

unknown action id

enumerator **INPUT\_KEY\_SERVICE\_ACTION\_CLICK**

click action id

enumerator **INPUT\_KEY\_SERVICE\_ACTION\_CLICK\_RELEASE**

click release action id

enumerator **INPUT\_KEY\_SERVICE\_ACTION\_PRESS**

long press action id

enumerator **INPUT\_KEY\_SERVICE\_ACTION\_PRESS\_RELEASE**

long press release id

## Header File

- components/input\_key\_service/include/input\_key\_com\_user\_id.h

## Enumerations

enum **input\_key\_user\_id\_t**

input key user user-defined id

*Values:*

enumerator **INPUT\_KEY\_USER\_ID\_UNKNOWN**

unknown user id

enumerator **INPUT\_KEY\_USER\_ID\_REC**

user id for recording

enumerator **INPUT\_KEY\_USER\_ID\_SET**

user id for settings

enumerator **INPUT\_KEY\_USER\_ID\_PLAY**

user id for playing

enumerator **INPUT\_KEY\_USER\_ID\_MODE**  
user id for mode

enumerator **INPUT\_KEY\_USER\_ID\_VOLDOWN**  
user id for volume down

enumerator **INPUT\_KEY\_USER\_ID\_VOLUP**  
user id for volume up

enumerator **INPUT\_KEY\_USER\_ID\_MUTE**  
user id for mute

enumerator **INPUT\_KEY\_USER\_ID\_CAPTURE**  
user id for capture photo

enumerator **INPUT\_KEY\_USER\_ID\_MSG**  
user id for message

enumerator **INPUT\_KEY\_USER\_ID\_BATTERY\_CHARGING**  
user id for battery charging

enumerator **INPUT\_KEY\_USER\_ID\_WAKEUP**  
user id for GPIO wakeup

enumerator **INPUT\_KEY\_USER\_ID\_COLOR**  
user id for color

enumerator **INPUT\_KEY\_USER\_ID\_MAX**

### 2.6.3 Wi-Fi Service

The Wi-Fi service can configure and manage the network. SmartConfig, BluFi, and AirKiss are supported for network configuration.

#### Application Example

Implementation of this API is demonstrated in the following example:

- [dueros](#)

## Header File

- components/wifi\_service/include/wifi\_service.h

## Functions

*periph\_service\_handle\_t* **wifi\_service\_create** (*wifi\_service\_config\_t* \*config)

esp\_err\_t **wifi\_service\_destroy** (*periph\_service\_handle\_t* handle)

esp\_err\_t **wifi\_service\_register\_setting\_handle** (*periph\_service\_handle\_t* handle, *esp\_wifi\_setting\_handle\_t* setting, int \*out\_index)

esp\_err\_t **wifi\_service\_setting\_start** (*periph\_service\_handle\_t* handle, int index)

esp\_err\_t **wifi\_service\_update\_sta\_info** (*periph\_service\_handle\_t* handle, *wifi\_config\_t* \*wifi\_conf)

esp\_err\_t **wifi\_service\_setting\_stop** (*periph\_service\_handle\_t* handle, int index)

esp\_err\_t **wifi\_service\_connect** (*periph\_service\_handle\_t* handle)

esp\_err\_t **wifi\_service\_disconnect** (*periph\_service\_handle\_t* handle)

esp\_err\_t **wifi\_service\_set\_sta\_info** (*periph\_service\_handle\_t* handle, *wifi\_config\_t* \*info)

*periph\_service\_state\_t* **wifi\_service\_state\_get** (*periph\_service\_handle\_t* handle)

*wifi\_service\_disconnect\_reason\_t* **wifi\_service\_disconnect\_reason\_get** (*periph\_service\_handle\_t* handle)

esp\_err\_t **wifi\_service\_erase\_ssid\_manager\_info** (*periph\_service\_handle\_t* handle)

esp\_err\_t **wifi\_service\_get\_last\_ssid\_cfg** (*periph\_service\_handle\_t* handle, *wifi\_config\_t* \*wifi\_cfg)

## Structures

struct **wifi\_service\_config\_t**

WiFi service configurations.

### Public Members

int **task\_stack**

>0 Service task stack; =0 with out task created

int **task\_prio**

Service task priority (based on freeRTOS priority)

int **task\_core**

Service task running in core (0 or 1)

bool **extern\_stack**

Task stack allocate on the extern ram

*periph\_service\_cb* **evt\_cb**

Service callback function

void \***cb\_ctx**

Callback context

char \***user\_data**

User data

int **setting\_timeout\_s**

Timeout of setting WiFi

int **max\_retry\_time**

Maximum times of reconnection

int **max\_prov\_retry\_time**

Maximum times of reconnection after wifi provision

uint8\_t **max\_ssid\_num**

Maximum ssid that can be stored

## Macros

**WIFI\_SERVICE\_DEFAULT\_CONFIG()**

## Enumerations

enum **wifi\_service\_event\_t**

WiFi STA service status.

*Values:*

enumerator **WIFI\_SERV\_EVENT\_UNKNOWN**

enumerator **WIFI\_SERV\_EVENT\_CONNECTING**

enumerator **WIFI\_SERV\_EVENT\_CONNECTED**

enumerator **WIFI\_SERV\_EVENT\_DISCONNECTED**

enumerator **WIFI\_SERV\_EVENT\_SETTING\_TIMEOUT**

enumerator **WIFI\_SERV\_EVENT\_SETTING\_FAILED**

enumerator **WIFI\_SERV\_EVENT\_SETTING\_FINISHED**

enum **wifi\_service\_disconnect\_reason\_t**

WiFi STA disconnection reasons.

*Values:*

enumerator **WIFI\_SERV\_STA\_UNKNOWN**

enumerator **WIFI\_SERV\_STA\_COM\_ERROR**

enumerator **WIFI\_SERV\_STA\_AUTH\_ERROR**

enumerator **WIFI\_SERV\_STA\_AP\_NOT\_FOUND**

enumerator **WIFI\_SERV\_STA\_BY\_USER**

enumerator **WIFI\_SERV\_STA\_SET\_INFO**

## Header File

- components/wifi\_service/include/esp\_wifi\_setting.h

## Functions

*esp\_wifi\_setting\_handle\_t* **esp\_wifi\_setting\_create** (const char \*tag)

brief Create wifi setting handle instance

**Parameters** **tag** – Tag of the wifi setting handle

**Returns**

- NULL, Fail
- Others, Success

**esp\_err\_t** **esp\_wifi\_setting\_destroy** (*esp\_wifi\_setting\_handle\_t* handle)

brief Destroy wifi setting handle instance

**Parameters** **handle** – The wifi setting handle instance

**Returns**

- ESP\_OK
- ESP\_FAIL

`esp_err_t esp_wifi_setting_register_function` (*esp\_wifi\_setting\_handle\_t* handle, *wifi\_setting\_func* start, *wifi\_setting\_func* stop, *wifi\_setting\_tearardown\_func* teardown)

Register the wifi setting execute functions.

### Parameters

- **handle** – The wifi setting handle instance
- **start** – The start wifi setting
- **stop** – The stop
- **teardown** – The destroy

### Returns

- ESP\_OK
- ESP\_FAIL

`esp_err_t esp_wifi_setting_register_notify_handle` (*esp\_wifi\_setting\_handle\_t* handle, void \*on\_handle)

Register the notify execute handle.

### Parameters

- **handle** – The peripheral handle
- **on\_handle** – The notify execute handle

### Returns

- ESP\_OK
- ESP\_FAIL

`esp_err_t esp_wifi_setting_info_notify` (*esp\_wifi\_setting\_handle\_t* handle, *wifi\_config\_t* \*info)

Call this to notify the *wifi\_config\_t* to on\_handle

### Parameters

- **handle** – The wifi setting handle instance
- **info** – The *wifi\_config\_t*

### Returns

- ESP\_OK
- ESP\_FAIL

`esp_err_t esp_wifi_setting_set_data` (*esp\_wifi\_setting\_handle\_t* handle, void \*data)

Set the user data.

---

**Note:** Make sure the `data` lifetime is sufficient, this function does not copy all data, it only stores the data pointer

---

### Parameters

- **handle** – [in] The wifi setting handle instance
- **data** – The user data

### Returns

- ESP\_OK
- ESP\_FAIL

void **\*esp\_wifi\_setting\_get\_data** (*esp\_wifi\_setting\_handle\_t* handle)

Get the user data stored on handle

**Parameters** **handle** – [in] The wifi setting handle instance

**Returns** user data pointer

esp\_err\_t **esp\_wifi\_setting\_start** (*esp\_wifi\_setting\_handle\_t* handle)

Call this to execute `start` function of wifi setting instance.

**Parameters** **handle** – The wifi setting handle instance

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **esp\_wifi\_setting\_stop** (*esp\_wifi\_setting\_handle\_t* handle)

Call this to execute `stop` function of wifi setting instance.

**Parameters** **handle** – The wifi setting handle instance

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **esp\_wifi\_setting\_teardown** (*esp\_wifi\_setting\_handle\_t* handle, *wifi\_config\_t* \*info)

Call this to execute `teardown` function of wifi setting instance.

**Parameters**

- **handle** – The wifi setting handle instance
- **info** – The `wifi_config_t`

**Returns**

- ESP\_OK
- ESP\_FAIL

## Type Definitions

```
typedef struct esp_wifi_setting *esp_wifi_setting_handle_t
```

```
typedef esp_err_t (*wifi_setting_func)(esp_wifi_setting_handle_t handle)
```

```
typedef esp_err_t (*wifi_setting_teardown_func)(esp_wifi_setting_handle_t handle, wifi_config_t *info)
```

### Header File

- components/wifi\_service/include/smart\_config.h

### Functions

*esp\_wifi\_setting\_handle\_t* **smart\_config\_create** (*smart\_config\_info\_t* \*info)

brief Create smartconfig setting handle instance

**Parameters** **info** – Configuration of the smartconfig

**Returns**

- NULL, Fail
- Others, Success

### Structures

struct **smart\_config\_info\_t**  
esp smartconfig configuration

#### Public Members

smartconfig\_type\_t **type**  
Type of smartconfig

### Macros

**SMART\_CONFIG\_INFO\_DEFAULT** ()

### Header File

- components/wifi\_service/include/blufi\_config.h

### Functions

*esp\_wifi\_setting\_handle\_t* **blufi\_config\_create** (void \*info)

Create blufi setting handle instance.

**Parameters** **info** – [in] A pointer to void

**Returns**

- NULL, Fail
- Others, Success



`esp_err_t blufi_set_sta_connected_flag` (*esp\_wifi\_setting\_handle\_t* handle, bool flag)

Set flag to judge whether the station has connected to the AP.

#### Parameters

- **handle** – [in] Wifi setting handle
- **flag** – [in] bool type of station connection state

#### Returns

- NULL, Fail
- Others, Success

`esp_err_t blufi_set_customized_data` (*esp\_wifi\_setting\_handle\_t* handle, char \*data, int data\_len)

Set customized data to be sent after configurate wifi successfully.

#### Parameters

- **handle** – [in] Wifi setting handle
- **data** – [in] Customized data
- **data\_len** – [in] Customized data length

#### Returns

- ESP\_FAIL, Fail
- ESP\_OK, Success

`esp_err_t blufi_send_customized_data` (*esp\_wifi\_setting\_handle\_t* handle)

Send customized data that be set before.

**Parameters** **handle** – [in] Wifi setting handle

#### Returns

- ESP\_FAIL, Fail
- ESP\_OK, Success

## Header File

- components/wifi\_service/include/airkiss\_config.h

## Functions

*esp\_wifi\_setting\_handle\_t* **airkiss\_config\_create** (*airkiss\_config\_info\_t* \*info)

brief Create airkiss setting handle instance

**Parameters** **info** – Configuration of the airkiss

#### Returns

- NULL, Fail
- Others, Success

### Structures

struct **airkiss\_lan\_pack\_param\_t**  
airkiss lan data pack

#### Public Members

void \***appid**  
APP identifier data

void \***deviceid**  
Device identifier data

struct **airkiss\_config\_info\_t**  
airkiss configurations

#### Public Members

*airkiss\_lan\_pack\_param\_t* **lan\_pack**  
User lan pack parameter

bool **ssdp\_notify\_enable**  
Notify enable flag

char \***aes\_key**  
Airkiss aes key data

### Macros

**AIRKISS\_CONFIG\_INFO\_DEFAULT** ()

### Header File

- [components/wifi\\_service/include/wifi\\_ssid\\_manager.h](#)

## Functions

```

wifi_ssid_manager_handle_t wifi_ssid_manager_create (uint8_t max_ssid_num)

esp_err_t wifi_ssid_manager_get_latest_config (wifi_ssid_manager_handle_t handle, wifi_config_t
                                             *config)

esp_err_t wifi_ssid_manager_save (wifi_ssid_manager_handle_t handle, const char *ssid, const char *pwd)

esp_err_t wifi_ssid_manager_get_best_config (wifi_ssid_manager_handle_t handle, wifi_config_t
                                             *config)

int wifi_ssid_manager_get_ssid_num (wifi_ssid_manager_handle_t handle)

esp_err_t wifi_ssid_manager_list_show (wifi_ssid_manager_handle_t handle)

esp_err_t wifi_ssid_manager_erase_all (wifi_ssid_manager_handle_t handle)

esp_err_t wifi_ssid_manager_destroy (wifi_ssid_manager_handle_t handle)

```

## Type Definitions

```
typedef struct wifi_ssid_manager *wifi_ssid_manager_handle_t
```

## 2.6.4 OTA Service

OTA service can upgrade firmware over the air (OTA). The firmware can be fetched from local files or the network.

### Application Example

Implementation of this API is demonstrated in the following example:

- ota

### Header File

- components/ota\_service/include/ota\_service.h

## Functions

```
periph_service_handle_t ota_service_create (ota_service_config_t *config)
```

Create the OTA service instance.

**Parameters** **config** – configuration of the OTA service

### Returns

- NULL: Failed
- Others: Success

`esp_err_t ota_service_set_upgrade_param` (*periph\_service\_handle\_t* handle, *ota\_upgrade\_ops\_t* \*list, int list\_len)

Configure the upgrade parameter This function is not thread safe.

This function will set the parameter table to ota service, and the ota service will upgrade the partitions defined in the table one by one,

### Parameters

- **handle** – [in] pointer to ‘periph\_service\_handle\_t’ structure
- **list** – [in] pointer to ‘ota\_upgrade\_ops\_t’ structure
- **list\_len** – [in] length of the ‘list’

### Returns

- ESP\_OK: Success
- Others: Failed

## Structures

struct **ota\_service\_config\_t**

The OTA service configuration.

### Public Members

int **task\_stack**

>0 Service task stack; =0 with out task created

int **task\_prio**

Service task priority (based on freeRTOS priority)

int **task\_core**

Service task running in core (0 or 1)

*periph\_service\_cb* **evt\_cb**

Service callback function

void \***cb\_ctx**

Callback context

struct **ota\_node\_attr\_t**

The OTA node attributions.

**Public Members**

esp\_partition\_type\_t **type**

Partition type

char \***label**

Partition label

char \***uri**

The upgrade URL

char \***cert\_pem**

SSL server certification, PEM format as string, if the client requires to verify server

struct **ota\_upgrade\_ops\_t**

The upgrade operation.

**Public Members**

*ota\_node\_attr\_t* **node**

The OTA node

*ota\_service\_err\_reason\_t* (\***prepare**)(void \*\*handle, *ota\_node\_attr\_t* \*node)

Functions ready for upgrade

*ota\_service\_err\_reason\_t* (\***need\_upgrade**)(void \*handle, *ota\_node\_attr\_t* \*node)

Detect whether an upgrade is required

*ota\_service\_err\_reason\_t* (\***execute\_upgrade**)(void \*handle, *ota\_node\_attr\_t* \*node)

For execute upgrade

*ota\_service\_err\_reason\_t* (\***finished\_check**)(void \*handle, *ota\_node\_attr\_t* \*node,

*ota\_service\_err\_reason\_t* result)

Check result of upgrade

bool **reboot\_flag**

Reboot or not after upgrade

bool **break\_after\_fail**

Abort upgrade when got failed

struct **ota\_result\_t**

The result of the OTA upgrade.

### Public Members

`uint8_t id`

The result ID

`ota_service_err_reason_t result`

The error reason

### Macros

`OTA_SERVICE_ERR_REASON_BASE`

`OTA_SERVICE_DEFAULT_CONFIG()`

### Enumerations

enum `ota_service_event_type_t`

The OTA service event type.

*Values:*

enumerator `OTA_SERV_EVENT_TYPE_RESULT`

enumerator `OTA_SERV_EVENT_TYPE_FINISH`

enum `ota_service_err_reason_t`

The OTA service error reasons.

*Values:*

enumerator `OTA_SERV_ERR_REASON_UNKNOWN`

enumerator `OTA_SERV_ERR_REASON_SUCCESS`

enumerator `OTA_SERV_ERR_REASON_NULL_POINTER`

enumerator `OTA_SERV_ERR_REASON_URL_PARSE_FAIL`

enumerator `OTA_SERV_ERR_REASON_ERROR_VERSION`

enumerator `OTA_SERV_ERR_REASON_NO_HIGHER_VERSION`

enumerator `OTA_SERV_ERR_REASON_ERROR_MAGIC_WORD`

enumerator `OTA_SERV_ERR_REASON_ERROR_PROJECT_NAME`

enumerator `OTA_SERV_ERR_REASON_FILE_NOT_FOUND`

enumerator `OTA_SERV_ERR_REASON_PARTITION_NOT_FOUND`

enumerator `OTA_SERV_ERR_REASON_PARTITION_WT_FAIL`

enumerator `OTA_SERV_ERR_REASON_PARTITION_RD_FAIL`

enumerator `OTA_SERV_ERR_REASON_STREAM_INIT_FAIL`

enumerator `OTA_SERV_ERR_REASON_STREAM_RD_FAIL`

enumerator `OTA_SERV_ERR_REASON_GET_NEW_APP_DESC_FAIL`

## Header File

- `components/ota_service/include/ota_proc_default.h`

## Functions

void `ota_app_get_default_proc` (*ota\_upgrade\_ops\_t* \*ops)

get the default process of app partition upgrade

**Parameters** `ops` – [in] pointer to *ota\_upgrade\_ops\_t* structure

void `ota_data_get_default_proc` (*ota\_upgrade\_ops\_t* \*ops)

get the default process of data partition upgrade

**Parameters** `ops` – [in] pointer to *ota\_upgrade\_ops\_t* structure

*ota\_service\_err\_reason\_t* `ota_data_image_stream_read` (void \*handle, char \*buf, int wanted\_size)

read from the stream of upgrading

### Parameters

- `handle` – [in] pointer to upgrade handle
- `buf` – [in] pointer to receive buffer
- `wanted_size` – [in] bytes to read

### Returns

- `OTA_SERV_ERR_REASON_SUCCESS`: Success
- Others: Failed

*ota\_service\_err\_reason\_t* **ota\_data\_partition\_write** (void \*handle, char \*buf, int size)

write to the data partition under upgrading

### Parameters

- **handle** – [in] pointer to upgrade handle
- **buf** – [in] pointer to data buffer
- **size** – [in] bytes to write

### Returns

- OTA\_SERV\_ERR\_REASON\_SUCCESS: Success
- Others: Failed

void **ota\_data\_partition\_erase\_mark** (void \*handle)

Indicates that the ota partition has been erased. By default, this part of flash will be erased during ota. If the behavior of erasing is called in application, this API needs to be called.

**Parameters** **handle** – [in] pointer to upgrade handle

int **ota\_get\_version\_number** (char \*version)

Convert string of version to integer. The version should be (V0.0.0 - V255.255.255)

**Parameters** **version** – [in] pointer to the string of version

### Returns

- -1: Failed
- Others: version number

## Header File

- `components/ota_service/include/esp_fs_ota.h`

## Functions

esp\_err\_t **esp\_fs\_ota** (*esp\_fs\_ota\_config\_t* \*ota\_config)

Upgrade the firmware from filesystem.

---

**Note:** This API handles the entire OTA operation, so if this API is being used then no other APIs from `esp_fs_ota` component should be called. If more information and control is needed during the FS OTA process, then one can use `esp_fs_ota_begin` and subsequent APIs. If this API returns successfully, `esp_restart()` must be called to boot from the new firmware image.

---

**Parameters** **ota\_config** – [in] pointer to *esp\_fs\_ota\_config\_t* structure.

### Returns

- ESP\_OK: OTA data updated, next reboot will use specified partition.
- ESP\_FAIL: For generic failure.
- ESP\_ERR\_INVALID\_ARG: Invalid argument
- ESP\_ERR\_OTA\_VALIDATE\_FAILED: Invalid app image



- `ESP_ERR_NO_MEM`: Cannot allocate memory for OTA operation.
- `ESP_ERR_FLASH_OP_TIMEOUT` or `ESP_ERR_FLASH_OP_FAIL`: Flash write failed.
- For other return codes, refer OTA documentation in esp-idf's app\_update component.

`esp_err_t esp_fs_ota_begin` (*esp\_fs\_ota\_config\_t* \*ota\_config, *esp\_fs\_ota\_handle\_t* \*handle)

Start FS OTA Firmware upgrade.

This function initializes ESP FS OTA context and open the firmware file. This function must be invoked first. If this function returns successfully, then `esp_fs_ota_perform` should be called to continue with the OTA process and there should be a call to `esp_fs_ota_finish` on completion of OTA operation or on failure in subsequent operations.

#### Parameters

- **ota\_config** – [in] pointer to *esp\_fs\_ota\_config\_t* structure
- **handle** – [out] pointer to an allocated data of type *esp\_fs\_ota\_handle\_t* which will be initialised in this function

#### Returns

- `ESP_OK`: FS OTA Firmware upgrade context initialised and file opened successful
- `ESP_FAIL`: For generic failure.
- `ESP_ERR_INVALID_ARG`: Invalid argument (missing/incorrect config, etc.)
- For other return codes, refer documentation in app\_update component and `esp_http_client` component in esp-idf.

`esp_err_t esp_fs_ota_perform` (*esp\_fs\_ota\_handle\_t* fs\_ota\_handle)

Read image data from file stream and write it to OTA partition.

This function reads image data from file stream and writes it to OTA partition. This function must be called only if `esp_fs_ota_begin()` returns successfully. This function must be called in a loop since it returns after every file read operation thus giving you the flexibility to stop OTA operation midway.

**Parameters** **fs\_ota\_handle** – [in] pointer to *esp\_fs\_ota\_handle\_t* structure

#### Returns

- `ESP_ERR_FS_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.

`esp_err_t esp_fs_ota_finish` (*esp\_fs\_ota\_handle\_t* fs\_ota\_handle)

Clean-up FS OTA Firmware upgrade and close the file stream.

This function closes the file stream and frees the ESP FS 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

---

**Parameters** `fs_ota_handle` – [in] pointer to `esp_fs_ota_handle_t` structure

**Returns**

- `ESP_OK`: Clean-up successful
- `ESP_ERR_INVALID_STATE`
- `ESP_ERR_INVALID_ARG`: Invalid argument
- `ESP_ERR_OTA_VALIDATE_FAILED`: Invalid app image

`esp_err_t esp_fs_ota_get_img_desc` (*esp\_fs\_ota\_handle\_t* fs\_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_fs_ota_begin()` and before `esp_fs_ota_perform()`. Calling this API is not mandatory.

---

**Parameters**

- `fs_ota_handle` – [in] pointer to `esp_fs_ota_handle_t` structure
- `new_app_info` – [out] pointer to an allocated `esp_app_desc_t` structure

**Returns**

- `ESP_ERR_INVALID_ARG`: Invalid arguments
- `ESP_FAIL`: Failed to read image descriptor
- `ESP_OK`: Successfully read image descriptor

`int esp_fs_ota_get_image_len_read` (*esp\_fs\_ota\_handle\_t* fs\_ota\_handle)

## Structures

struct `esp_fs_ota_config_t`

ESP FS OTA configuration.

**Public Members**

const char \*`path`

file path

const int `buffer_size`

Size of buffer

## Macros

**ESP\_ERR\_FS\_OTA\_BASE**

**ESP\_ERR\_FS\_OTA\_IN\_PROGRESS**

## Type Definitions

```
typedef void *esp_fs_ota_handle_t
```

## 2.6.5 DuerOS Service

The DuerOS service allows voice interaction with DuerOS.

### Application Example

Implementation of this API is demonstrated in the following example:

- dueros

### Header File

- components/dueros\_service/include/dueros\_service.h

### Functions

*audio\_service\_handle\_t* **dueros\_service\_create** ()

Create the dueros service.

#### Returns

- NULL, Fail
- Others, Success

*service\_state\_t* **dueros\_service\_state\_get** ()

Get dueros service state.

**Returns** The state of service

esp\_err\_t **dueros\_voice\_upload** (*audio\_service\_handle\_t* handle, void \*buf, int len)

Upload voice to backend server.

#### Parameters

- **handle** – dueros service handle
- **buf** – Data buffer
- **len** – Size of buffer

**Returns** ESP\_OK ESP\_FAIL

esp\_err\_t **dueros\_voice\_cancel** (*audio\_service\_handle\_t* handle)

Cancel the current session.

**Parameters** **handle** – dueros service handle

**Returns** ESP\_OK ESP\_FAIL

esp\_err\_t **dueros\_start\_wifi\_cfg** (*audio\_service\_handle\_t* handle, *duer\_wifi\_cfg\_t* \*cfg)

Start the wifi configure process.

**Parameters**

- **handle** – Dueros service handle
- **cfg** – Configuration

**Returns** ESP\_OK ESP\_FAIL

esp\_err\_t **dueros\_stop\_wifi\_cfg** (*audio\_service\_handle\_t* handle)

Stop the wifi configure process.

**Parameters** **handle** – Dueros service handle

**Returns** ESP\_OK ESP\_FAIL

esp\_err\_t **dueros\_wifi\_status\_report** (*audio\_service\_handle\_t* handle, *dueros\_wifi\_st\_t* \*st)

Report the wifi status to dipb.

**Parameters**

- **handle** – Dueros service handle
- **st** – WiFi status and error code

**Returns** ESP\_OK ESP\_FAIL

## Structures

struct **dueros\_wifi\_st\_t**

Status of WiFi connection.

## Public Members

int **status**

Please refer to: enum *duer\_dipb\_client\_status\_e*

int **err**

Please refer to: enum *duer\_wifi\_connect\_error\_code\_e*

## 2.6.6 Display Service

The display service defines enumeration values for some common display patterns in *display\_pattern\_t*, with which you can set the corresponding patterns of LEDs or LED bars.

The currently supported LED drivers are AW2013, WS2812, and IS31x.

### Application Example

Implementation of this API is demonstrated in the following example:

- `checks/check_display_led`

### Header File

- `components/display_service/include/display_service.h`

### Functions

*display\_service\_handle\_t* **display\_service\_create** (*display\_service\_config\_t* \*cfg)

esp\_err\_t **display\_service\_set\_pattern** (void \*handle, int disp\_pattern, int value)

esp\_err\_t **display\_destroy** (*display\_service\_handle\_t* handle)

### Structures

struct **display\_service\_config\_t**

Display service configurations.

#### Public Members

*periph\_service\_config\_t* **based\_cfg**

Peripheral service configuration

void \***instance**

Sub-instance

### Type Definitions

```
typedef struct display_service_impl *display_service_handle_t
```

### Enumerations

```
enum display_pattern_t
```

*Values:*

```
enumerator DISPLAY_PATTERN_UNKNOWN
```

```
enumerator DISPLAY_PATTERN_WIFI_SETTING
```

```
enumerator DISPLAY_PATTERN_WIFI_CONNECTTING
```

```
enumerator DISPLAY_PATTERN_WIFI_CONNECTED
```

```
enumerator DISPLAY_PATTERN_WIFI_DISCONNECTED
```

```
enumerator DISPLAY_PATTERN_WIFI_SETTING_FINISHED
```

```
enumerator DISPLAY_PATTERN_BT_CONNECTTING
```

```
enumerator DISPLAY_PATTERN_BT_CONNECTED
```

```
enumerator DISPLAY_PATTERN_BT_DISCONNECTED
```

```
enumerator DISPLAY_PATTERN_RECORDING_START
```

```
enumerator DISPLAY_PATTERN_RECORDING_STOP
```

```
enumerator DISPLAY_PATTERN_RECOGNITION_START
```

```
enumerator DISPLAY_PATTERN_RECOGNITION_STOP
```

```
enumerator DISPLAY_PATTERN_WAKEUP_ON
```

```
enumerator DISPLAY_PATTERN_WAKEUP_FINISHED
```

```
enumerator DISPLAY_PATTERN_MUSIC_ON
```

enumerator `DISPLAY_PATTERN_MUSIC_FINISHED`

enumerator `DISPLAY_PATTERN_VOLUME`

enumerator `DISPLAY_PATTERN_MUTE_ON`

enumerator `DISPLAY_PATTERN_MUTE_OFF`

enumerator `DISPLAY_PATTERN_TURN_ON`

enumerator `DISPLAY_PATTERN_TURN_OFF`

enumerator `DISPLAY_PATTERN_BATTERY_LOW`

enumerator `DISPLAY_PATTERN_BATTERY_CHARGING`

enumerator `DISPLAY_PATTERN_BATTERY_FULL`

enumerator `DISPLAY_PATTERN_POWERON_INIT`

enumerator `DISPLAY_PATTERN_WIFI_NO_CFG`

enumerator `DISPLAY_PATTERN_SPEECH_BEGIN`

enumerator `DISPLAY_PATTERN_SPEECH_OVER`

enumerator `DISPLAY_PATTERN_MAX`

## Header File

- `components/display_service/led_bar/include/led_bar_aw2013.h`

## Functions

void `aw2013_led_bar_task` (void \*parameters)

*esp\_periph\_handle\_t* `led_bar_aw2013_init` (void)

Initialize led bar instance.

### Returns

- NULL Error
- others Success

esp\_err\_t **led\_bar\_aw2013\_pattern** (void \*handle, int pat, int value)

Set led bar display pattern.

### Parameters

- **handle** – led bar instance
- **pat** – display pattern
- **value** – value of pattern

### Returns

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **led\_bar\_aw2013\_set\_blink\_time** (void \*handle, uint8\_t time, int period)

Set blinking period and times.

### Parameters

- **handle** – led bar instance
- **time** – times of blink
- **period** – period of blink

### Returns

- ESP\_OK
- ESP\_FAIL

void **led\_bar\_aw2013\_deinit** (*esp\_periph\_handle\_t* handle)

Destroy esp\_periph\_handle\_t instance.

**Parameters** **handle** – led bar instance

## Header File

- [components/display\\_service/led\\_bar/include/led\\_bar\\_is31x.h](#)

## Functions

*esp\_periph\_handle\_t* **led\_bar\_is31x\_init** ()

Initialize esp\_periph\_handle\_t instance.

### Returns

- NULL, Fail
- Others, Success

esp\_err\_t **led\_bar\_is31x\_pattern** (void \*handle, int pat, int value)

Set led bar display pattern.

### Parameters

- **handle** – led bar instance
- **pat** – display pattern



- **value** – value of pattern

#### Returns

- ESP\_OK
- ESP\_FAIL

void **led\_bar\_is31x\_deinit** (*esp\_periph\_handle\_t* handle)

Destroy *esp\_periph\_handle\_t* instance.

**Parameters** **handle** – led bar instance

#### Header File

- components/display\_service/led\_bar/include/led\_bar\_ws2812.h

#### Functions

*led\_bar\_ws2812\_handle\_t* **led\_bar\_ws2812\_init** (gpio\_num\_t gpio\_num, int led\_num)

Initialize *led\_bar\_ws2812\_handle\_t* instance.

#### Parameters

- **gpio\_num** – The GPIO number of ws2812
- **led\_num** – The number of all ws2812

#### Returns

- *led\_bar\_ws2812\_handle\_t*

esp\_err\_t **led\_bar\_ws2812\_pattern** (void \*handle, int pat, int value)

Set ws2812 pattern.

#### Parameters

- **handle** – ws2812 indicator instance
- **pat** – display pattern
- **value** – value of pattern

#### Returns

- ESP\_OK, success
- Others, fail

esp\_err\_t **led\_bar\_ws2812\_deinit** (*led\_bar\_ws2812\_handle\_t* handle)

Destroy *led\_bar\_ws2812\_handle\_t* instance.

**Parameters** **handle** – ws2812 indicator instance

#### Returns

#### Returns

- ESP\_OK, success
- Others, fail

### Type Definitions

```
typedef struct led_bar_ws2812_impl *led_bar_ws2812_handle_t
```

### Header File

- components/display\_service/led\_indicator/include/led\_indicator.h

### Functions

*led\_indicator\_handle\_t* **led\_indicator\_init** (gpio\_num\_t num)

Initialize led\_indicator\_handle\_t instance.

**Parameters** num – led gpio number

**Returns**

- NULL, Fail
- Others, Success

esp\_err\_t **led\_indicator\_pattern** (void \*handle, int pat, int value)

Set led indicator display pattern.

**Parameters**

- **handle** – led indicator instance
- **pat** – display pattern
- **value** – value of pattern

**Returns**

- ESP\_OK
- ESP\_FAIL

void **led\_indicator\_deinit** (*led\_indicator\_handle\_t* handle)

Destroy led\_indicator\_handle\_t instance.

**Parameters** handle – led indicator instance

### Type Definitions

```
typedef struct led_indicator_impl *led_indicator_handle_t
```

## 2.6.7 Battery Service

The battery service can monitor and manage battery voltage. You can access battery voltage and the events defined by *battery\_service\_event\_t* through callback functions.

### Application Example

Implementation of this API is demonstrated in the following example:

- `system/battery`

### Header File

- `components/battery_service/include/battery_service.h`

### Functions

*periph\_service\_handle\_t* **battery\_service\_create** (*battery\_service\_config\_t* \*config)

Create the battery service instance.

**Parameters** `config` – configuration of the battery service

#### Returns

- NULL: Failed
- Others: Success

*esp\_err\_t* **battery\_service\_vol\_report\_switch** (*periph\_service\_handle\_t* handle, bool on\_off)

Start or stop the battery voltage report.

#### Parameters

- **handle** – [in] pointer to ‘periph\_service\_handle\_t’ structure
- **on\_off** – [in] ‘true’ to start, ‘false’ to stop

#### Returns

- ESP\_OK: Success
- ESP\_ERR\_INVALID\_ARG handle is NULL

*esp\_err\_t* **battery\_service\_set\_vol\_report\_freq** (*periph\_service\_handle\_t* handle, int freq)

Set voltage report frequency.

#### Parameters

- **handle** – [in] pointer to ‘periph\_service\_handle\_t’ structure
- **freq** – [in] voltage report frequency

#### Returns

- ESP\_OK: Success
- ESP\_ERR\_INVALID\_ARG handle is NULL

### Structures

struct **battery\_service\_config\_t**

Battery service configure.

#### Public Members

int **task\_stack**

>0 Service task stack; =0 with out task created

int **task\_prio**

Service task priority (based on freeRTOS priority)

int **task\_core**

Service task running in core (0 or 1)

bool **extern\_stack**

Task stack allocate on the extern ram

*periph\_service\_cb* **evt\_cb**

Service callback function

void \***cb\_ctx**

Callback context

*vol\_monitor\_handle\_t* **vol\_monitor**

Battery monitor

void \***charger\_monitor**

Charger monitor. Not supported yet

### Macros

**BATTERY\_SERVICE\_DEFAULT\_CONFIG** ( )

### Enumerations

enum **battery\_service\_event\_t**

Battery service event.

*Values:*

enumerator **BAT\_SERV\_EVENT\_UNKNOWN**

enumerator `BAT_SERV_EVENT_VOL_REPORT`

enumerator `BAT_SERV_EVENT_BAT_FULL`

enumerator `BAT_SERV_EVENT_BAT_LOW`

enumerator `BAT_SERV_EVENT_CHARGING_BEGIN`

enumerator `BAT_SERV_EVENT_CHARGING_STOP`

## Header File

- `components/battery_service/monitors/include/voltage_monitor.h`

## Functions

`vol_monitor_handle_t` **vol\_monitor\_create** (`vol_monitor_param_t` \*config)

Create the voltage monitor instance.

**Parameters** `config` – [in] pointer to ‘`vol_monitor_param_t`’ structure

### Returns

- NULL: Failed
- Others: Success

`esp_err_t` **vol\_monitor\_destroy** (`vol_monitor_handle_t` handle)

Destroy the voltage monitor.

**Parameters** `handle` – [in] pointer to ‘`vol_monitor_handle_t`’ structure

### Returns

- ESP\_OK: Success
- Others: Failed

`esp_err_t` **vol\_monitor\_set\_event\_cb** (`vol_monitor_handle_t` handle, `vol_monitor_event_cb` event\_cb, void \*user\_ctx)

Set the event callback.

### Parameters

- **handle** – [in] pointer to ‘`vol_monitor_handle_t`’ structure
- **event\_cb** – [in] callback used to handle the events of voltage monitor
- **user\_ctx** – [in] user’s data

### Returns

- ESP\_OK: Success
- Others: Failed

esp\_err\_t **vol\_monitor\_start\_freq\_report** (*vol\_monitor\_handle\_t* handle)

Start the voltage report with the configured frequency.

**Parameters** **handle** – [in] pointer to ‘vol\_monitor\_handle\_t’ structure

**Returns**

- ESP\_OK: Success
- Others: Failed

esp\_err\_t **vol\_monitor\_stop\_freq\_report** (*vol\_monitor\_handle\_t* handle)

Stop the voltage frequency report.

**Parameters** **handle** – [in] pointer to ‘vol\_monitor\_handle\_t’ structure

**Returns**

- ESP\_OK: Success
- Others: Failed

esp\_err\_t **vol\_monitor\_set\_report\_freq** (*vol\_monitor\_handle\_t* handle, int freq)

Set the voltage report frequency.

**Parameters**

- **handle** – [in] pointer to ‘vol\_monitor\_handle\_t’ structure
- **freq** – [in] voltage report frequency

**Returns**

- ESP\_OK: Success
- Others: Failed

## Structures

struct **vol\_monitor\_param\_t**

Battery adc configure.

### Public Members

bool (\***init**)(void\*)

Voltage read init

bool (\***deinit**)(void\*)

Voltage read deinit

int (\***vol\_get**)(void\*)

Voltage read interface

void \***user\_data**

Parameters for callbacks

int **read\_freq**

Voltage read frequency, unit: s

int **report\_freq**

Voltage report frequency, voltage will be report with a interval calculate by  $\frac{read\_freq}{report\_freq}$

int **vol\_full\_threshold**

Voltage threshold to report, unit: mV

int **vol\_low\_threshold**

Voltage threshold to report, unit: mV

## Type Definitions

typedef void **\*vol\_monitor\_handle\_t**

voltage monitor handle

typedef void (**\*vol\_monitor\_event\_cb**)(int msg\_id, void \*data, void \*user\_ctx)

callback define

## Enumerations

enum **vol\_monitor\_event\_t**

Battery adc configure.

*Values:*

enumerator **VOL\_MONITOR\_EVENT\_FREQ\_REPORT**

enumerator **VOL\_MONITOR\_EVENT\_BAT\_FULL**

enumerator **VOL\_MONITOR\_EVENT\_BAT\_LOW**

## 2.6.8 Core Dump Upload Service

To investigate crashes in some sold devices, the backtrace is needed for analysis. The core dump upload service transmits the backtrace stored in the device partition over HTTP. To enable this feature, select `ESP_COREDUMP_ENABLE_TO_FLASH`.

### Application Example

Implementation of this API is demonstrated in the following example:

- `system/coredump`

### Header File

- `components/coredump_upload_service/include/coredump_upload_service.h`

### Functions

bool `coredump_need_upload()`

This function will check the reset code and determine whether to upload the coredump.

#### Returns

- true: last reboot is a abnormal reset.
- false

esp\_err\_t `coredump_upload(periph_service_handle_t handle, char *url)`

Upload the core dump image to the url. This function will block the current task until the upload process finished.

#### Parameters

- **handle** – [in] the ‘periph\_service\_handle\_t’
- **url** – [in] server addr

#### Returns

- ESP\_OK
- ESP\_FAIL

*periph\_service\_handle\_t* `coredump_upload_service_create(coredump_upload_service_config_t *config)`

Create the core dump upload service instance.

**Parameters** `config` – configuration of the OTA service

#### Returns

- NULL: Failed
- Others: Success

### Structures

struct `coredump_upload_service_config_t`

coredump service configuration parameters



## Public Members

int **task\_stack**

>0 Service task stack; =0 with out task created

int **task\_prio**

Service task priority (based on freeRTOS priority)

int **task\_core**

Service task running in core (0 or 1)

*periph\_service\_cb* **evt\_cb**

Service callback function

void \***cb\_ctx**

Callback context

bool (**\*do\_post**)(char \*url, uint8\_t \*data, size\_t len)

POST interface, users can override this to customize the http client. if left NULL, the service will use the default one

## Macros

**COREDUMP\_UPLOAD\_SERVICE\_DEFAULT\_CONFIG** ( )

## 2.6.9 Header File

- components/esp\_dispatcher/include/periph\_service.h

## 2.6.10 Functions

*periph\_service\_handle\_t* **periph\_service\_create** (*periph\_service\_config\_t* \*config)

brief Create peripheral service instance

**Parameters** **config** – [**in**] Configuration of the peripheral service instance

**Returns**

- NULL, Fail
- Others, Success

esp\_err\_t **periph\_service\_destroy** (*periph\_service\_handle\_t* handle)

brief Destroy peripheral service instance

**Parameters** **handle** – [**in**] The peripheral service instance

**Returns**

- ESP\_OK

- ESP\_FAIL
- ESP\_ERR\_INVALID\_ARG

esp\_err\_t **periph\_service\_start** (*periph\_service\_handle\_t* handle)

brief Start the specific peripheral service

**Parameters** **handle** – [in] The peripheral service instance

**Returns**

- ESP\_OK
- ESP\_FAIL
- ESP\_ERR\_INVALID\_ARG

esp\_err\_t **periph\_service\_stop** (*periph\_service\_handle\_t* handle)

brief Stop the specific peripheral service

**Parameters** **handle** – [in] The peripheral service instance

**Returns**

- ESP\_OK
- ESP\_FAIL
- ESP\_ERR\_INVALID\_ARG

esp\_err\_t **periph\_service\_set\_callback** (*periph\_service\_handle\_t* handle, *periph\_service\_cb* cb, void \*ctx)

brief Set the specific peripheral service callback function

**Parameters**

- **handle** – [in] The peripheral service instance
- **cb** – [in] A pointer to service\_callback
- **ctx** – [in] A pointer to user context

**Returns**

- ESP\_OK
- ESP\_FAIL
- ESP\_ERR\_INVALID\_ARG

esp\_err\_t **periph\_service\_callback** (*periph\_service\_handle\_t* handle, *periph\_service\_event\_t* \*evt)

brief Called peripheral service by configurations

**Parameters**

- **handle** – [in] The peripheral service instance
- **evt** – [in] A pointer to *periph\_service\_event\_t*

**Returns**

- ESP\_OK
- ESP\_FAIL
- ESP\_ERR\_INVALID\_ARG

esp\_err\_t **periph\_service\_set\_data** (*periph\_service\_handle\_t* handle, void \*data)

brief Set user data to specific peripheral service instance

**Parameters**

- **handle** – [in] The peripheral service instance
- **data** – [in] A pointer to user data

**Returns**

- ESP\_OK
- ESP\_FAIL
- ESP\_ERR\_INVALID\_ARG

void \***periph\_service\_get\_data** (*periph\_service\_handle\_t* handle)

brief Get user data by specific peripheral service instance

**Parameters** **handle** – [in] The peripheral service instance

**Returns** A pointer to user data

esp\_err\_t **periph\_service\_ioctl** (*periph\_service\_handle\_t* handle, void \*ioctl\_handle, int cmd, int value)

brief In/out control by peripheral service instance

**Parameters**

- **handle** – [in] The peripheral service instance
- **ioctl\_handle** – [in] Sub-instance handle
- **cmd** – [in] Command of value
- **value** – [in] Data of the command

**Returns**

- ESP\_OK
- ESP\_FAIL
- ESP\_ERR\_INVALID\_ARG

## 2.6.11 Structures

struct **periph\_service\_event\_t**

Peripheral service event informations.

**Public Members**

int **type**

Type of event

void \***source**

Event source

void **\*data**  
Event data

int **len**  
Length of data

struct **periph\_service\_config\_t**  
Peripheral service configurations.

### Public Members

int **task\_stack**  
>0 Service task stack; =0 with out task created

int **task\_prio**  
Service task priority (based on freeRTOS priority)

int **task\_core**  
Service task running in core (0 or 1)

TaskFunction\_t **task\_func**  
Service task function

bool **extern\_stack**  
Task stack allocate on the extern ram

*periph\_service\_ctrl* **service\_start**  
Start function

*periph\_service\_ctrl* **service\_stop**  
Stop function

*periph\_service\_ctrl* **service\_destroy**  
Destroy function

*periph\_service\_io* **service\_ioctl**  
In out control function

char **\*service\_name**  
Name of peripheral service

void **\*user\_data**  
User data

## 2.6.12 Type Definitions

```
typedef struct periph_service_impl *periph_service_handle_t
```

```
typedef esp_err_t (*periph_service_ctrl)(periph_service_handle_t handle)
```

```
typedef esp_err_t (*periph_service_io)(void *ioctl_handle, int cmd, int value)
```

```
typedef esp_err_t (*periph_service_cb)(periph_service_handle_t handle, periph_service_event_t *evt, void *ctx)
```

## 2.6.13 Enumerations

```
enum periph_service_state_t
```

Peripheral service state.

*Values:*

```
enumerator PERIPH_SERVICE_STATE_UNKNOWN
```

```
enumerator PERIPH_SERVICE_STATE_IDLE
```

```
enumerator PERIPH_SERVICE_STATE_RUNNING
```

```
enumerator PERIPH_SERVICE_STATE_STOPPED
```

## 2.6.14 Header File

- `components/esp_dispatcher/include/audio_service.h`

## 2.6.15 Functions

```
audio_service_handle_t audio_service_create(audio_service_config_t *config)
```

brief Create audio service instance

**Parameters** `config` – [in] Configuration of the audio service instance

**Returns**

- NULL, Fail
- Others, Success

```
esp_err_t audio_service_destroy(audio_service_handle_t handle)
```

brief Destroy audio service instance

**Parameters** `handle` – [in] The audio service instance

**Returns**

- ESP\_OK
- ESP\_FAIL
- ESP\_ERR\_INVALID\_ARG

esp\_err\_t **audio\_service\_start** (*audio\_service\_handle\_t* handle)

brief Start the specific audio service

**Parameters** **handle** – [in] The audio service instance

**Returns**

- ESP\_OK
- ESP\_FAIL
- ESP\_ERR\_INVALID\_ARG

esp\_err\_t **audio\_service\_stop** (*audio\_service\_handle\_t* handle)

brief Stop the specific audio service

**Parameters** **handle** – [in] The audio service instance

**Returns**

- ESP\_OK
- ESP\_FAIL
- ESP\_ERR\_INVALID\_ARG

esp\_err\_t **audio\_service\_set\_callback** (*audio\_service\_handle\_t* handle, *service\_callback* cb, void \*ctx)

brief Set the specific audio service callback function.

**Parameters**

- **handle** – [in] The audio service instance
- **cb** – [in] A pointer to *service\_callback*
- **ctx** – [in] A pointer to user context

**Returns**

- ESP\_OK
- ESP\_FAIL
- ESP\_ERR\_INVALID\_ARG

esp\_err\_t **audio\_service\_callback** (*audio\_service\_handle\_t* handle, *service\_event\_t* \*evt)

brief Called audio service by configurations

**Parameters**

- **handle** – [in] The audio service instance
- **evt** – [in] A pointer to *service\_event\_t*

**Returns**

- ESP\_OK
- ESP\_FAIL
- ESP\_ERR\_INVALID\_ARG

esp\_err\_t **audio\_service\_connect** (*audio\_service\_handle\_t* handle)

brief Connect the specific audio service

**Parameters** **handle** – [in] The audio service instance

**Returns**

- ESP\_OK
- ESP\_FAIL
- ESP\_ERR\_INVALID\_ARG

esp\_err\_t **audio\_service\_disconnect** (*audio\_service\_handle\_t* handle)

brief Disconnect the specific audio service

**Parameters** **handle** – [in] The audio service instance

**Returns**

- ESP\_OK
- ESP\_FAIL
- ESP\_ERR\_INVALID\_ARG

esp\_err\_t **audio\_service\_set\_data** (*audio\_service\_handle\_t* handle, void \*data)

brief Set user data to specific audio service instance

**Parameters**

- **handle** – [in] The audio service instance
- **data** – [in] A pointer to user data

**Returns**

- ESP\_OK
- ESP\_FAIL
- ESP\_ERR\_INVALID\_ARG

void \***audio\_service\_get\_data** (*audio\_service\_handle\_t* handle)

brief Get user data by specific audio service instance

**Parameters** **handle** – [in] The audio service instance

**Returns** A pointer to user data

## 2.6.16 Structures

struct **service\_event\_t**

Audio service event informations.

### Public Members

int **type**  
Type of event

void \***source**  
Event source

void \***data**  
Event data

int **len**  
Length of data

struct **audio\_service\_config\_t**  
Audio service configurations.

### Public Members

int **task\_stack**  
>0 Service task stack; =0 with out task created

int **task\_prio**  
Service task priority (based on freeRTOS priority)

int **task\_core**  
Service task running in core (0 or 1)

TaskFunction\_t **task\_func**  
A pointer to TaskFunction\_t for service task function

*service\_ctrl* **service\_start**  
Start function

*service\_ctrl* **service\_stop**  
Stop function

*service\_ctrl* **service\_connect**  
Connect function

*service\_ctrl* **service\_disconnect**  
Disconnect function

*service\_ctrl* **service\_destroy**  
Destroy function



```
const char *service_name
```

Name of audio service

```
void *user_data
```

User context

## 2.6.17 Type Definitions

```
typedef struct audio_service_impl *audio_service_handle_t
```

```
typedef esp_err_t (*service_ctrl)(audio_service_handle_t handle)
```

```
typedef esp_err_t (*service_callback)(audio_service_handle_t handle, service_event_t *evt, void *ctx)
```

## 2.6.18 Enumerations

```
enum service_state_t
```

Audio service state.

*Values:*

```
enumerator SERVICE_STATE_UNKNOWN
```

```
enumerator SERVICE_STATE_IDLE
```

```
enumerator SERVICE_STATE_CONNECTING
```

```
enumerator SERVICE_STATE_CONNECTED
```

```
enumerator SERVICE_STATE_RUNNING
```

```
enumerator SERVICE_STATE_STOPPED
```

## 2.7 Speech Recognition

ESP-ADF offers a comprehensive range of speech recognition processing functions, such as front-end speech processing, TTS, voice wake-up, and command word recognition. ESP-ADF's Element-based *audio recorder* integrates speech recognition and audio signal processing into an event-driven High-level API. This approach enables users to maintain flexibility in their configurations while making it easier to use.

## 2.7.1 Voice Activity Detection

Voice activity detection (VAD) is a technique used in speech processing to detect the presence (or absence) of human speech. Detection of somebody speaking may be used to activate some processes, e.g. automatically switch on voice recording. It may be also used to deactivate processes, e.g. stop coding and transmission of silence packets to save on computation and network bandwidth.

Provided in this section API implements VAD functionality together with couple of options to configure sensitivity of speech detection, set sample rate or duration of audio samples.

### Application Example

Implementation of the voice activity detection API is demonstrated in [speech\\_recognition/vad](#) example.

### API Reference

For the latest API reference please refer to [Espressif Speech recognition repository](#).

## 2.7.2 WakeNet Interface

Setting up the speech recognition application to detect a wakeup word may be done using series of Audio Elements linked into a pipeline shown below.

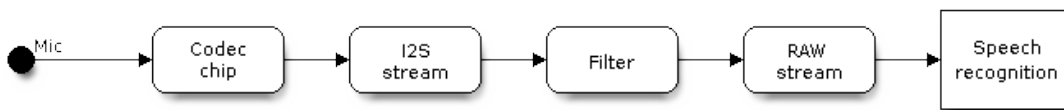


Fig. 5: Sample Speech Recognition Pipeline

Configuration and use of particular elements is demonstrated in several [examples](#) linked to elsewhere in this documentation. What may need clarification is use of the **Filter** and the **RAW stream**. The filter is used to adjust the sample rate of the I2S stream to match the sample rate of the speech recognition model. The RAW stream is the way to feed the audio input to the model.

The above introduction is the primary guidance. ESP-ADF offers users a more flexible and convenient module, namely the *audio recorder*, which is strongly recommended for use.

### API Reference

For the latest speech recognition API reference, please refer to [ESP-SR Speech Recognition Framework](#).

### 2.7.3 Audio Recorder

The Audio Recorder API is a set of functions to facilitate voice recording. It combines two important functions, namely Audio Front End (AFE) and audio encoding. This allows users to customize AFE's Voice Activity Detection (VAD), Automatic Gain Control (AGC), and Acoustic Echo Cancellation (AEC) settings. The encoding function is used by users to establish the encoding audio element, which supports various formats such as AAC, AMR-NB, AMR-WB, ADPCM, WAV, OPUS, and G711. The `audio_rec_evt_t` event makes it easy for users to interact with the Audio Recorder software.

The data path of the Audio recorder is presented in the diagram below.

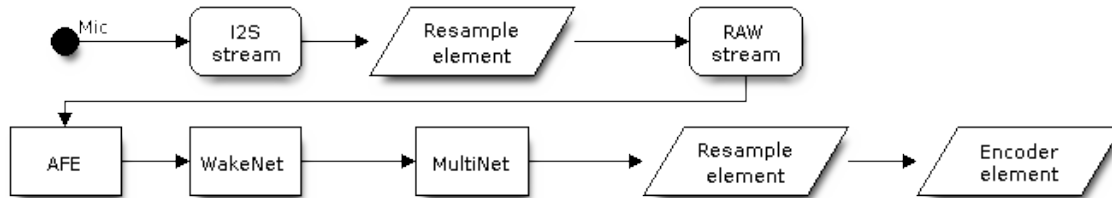


Fig. 6: Audio recorder data path

The area represented by the parallelogram is configurable by the user according to their needs, such as sampling frequency, whether to encode, and encoding format.

#### Application Example

The [speech\\_recognition/wwe/](#) example demonstrates how to initialize the speech recognition model, determine the number of samples and the sample rate of voice data to feed to the model, detect the wake-up word and command words, and encode voice to specific audio format.

#### API Reference

##### Header File

- `components/audio_recorder/include/audio_recorder.h`

##### Functions

`audio_rec_handle_t audio_recorder_create (audio_rec_cfg_t *cfg)`

Initialize and start up audio recorder.

**Parameters** `cfg` – Configuration of audio recorder

**Returns** NULL failed Others audio recorder handle

`esp_err_t audio_recorder_trigger_start (audio_rec_handle_t handle)`

Start recording by force.

---

**Note:** If there need to read from recorder without wake word detected or read from recorder while the wake word detection is disabled, this interface can be use to force start the recorder process.

---

**Parameters** **handle** – Audio recorder handle

**Returns** ESP\_OK ESP\_FAIL

esp\_err\_t **audio\_recorder\_trigger\_stop** (*audio\_rec\_handle\_t* handle)

Stop recording by force.

---

**Note:** No matter the recorder process is triggered by wake word detected or triggered by `audio_recorder_trigger_start`, this function can be used to force stop the recorder. And if the VAD detection is disabled, this must be invoked to stop recording after `audio_recorder_trigger_start`.

---

**Parameters** **handle** – Audio recorder handle

**Returns** ESP\_OK ESP\_FAIL

esp\_err\_t **audio\_recorder\_wakenet\_enable** (*audio\_rec\_handle\_t* handle, bool enable)

Enable or suspend wake word detection.

**Parameters**

- **handle** – Audio recorder handle
- **enable** – true: enable wake word detection false: disable wake word detection

**Returns** ESP\_OK ESP\_FAIL

esp\_err\_t **audio\_recorder\_multinet\_enable** (*audio\_rec\_handle\_t* handle, bool enable)

Enable or suspend speech command recognition.

**Parameters**

- **handle** – Audio recorder handle
- **enable** – true: enable speech command recognition false: disable speech command recognition

**Returns** ESP\_OK ESP\_FAIL

esp\_err\_t **audio\_recorder\_vad\_check\_enable** (*audio\_rec\_handle\_t* handle, bool enable)

Enable or suspend voice duration check.

**Parameters**

- **handle** – Audio recorder handle
- **enable** – true: enable voice duration check false: disable voice duration check

**Returns** ESP\_OK ESP\_FAIL

int **audio\_recorder\_data\_read** (*audio\_rec\_handle\_t* handle, void \*buffer, int length, TickType\_t ticks)

Read data from audio recorder.

**Parameters**

- **handle** – Audio recorder handle
- **buffer** – Buffer to save data
- **length** – Size of buffer
- **ticks** – Timeout for reading

**Returns** Length of data actually read ESP\_ERR\_INVALID\_ARG

esp\_err\_t **audio\_recorder\_destroy** (*audio\_rec\_handle\_t* handle)

Destroy audio recorder and recycle all resource.

**Parameters** **handle** – Audio recorder handle

**Returns** ESP\_OK ESP\_FAIL

bool **audio\_recorder\_get\_wakeup\_state** (*audio\_rec\_handle\_t* handle)

Get the wake up state of audio recorder.

**Parameters** **handle** – Audio recorder handle

**Returns** true false

## Structures

struct **audio\_rec\_evt\_t**

Recorder event.

## Public Types

enum [**anonymous**]

Audio recorder event type.

*Values:*

enumerator **AUDIO\_REC\_WAKEUP\_START**

Wakeup start

enumerator **AUDIO\_REC\_WAKEUP\_END**

Wakeup stop

enumerator **AUDIO\_REC\_VAD\_START**

Vad start

enumerator **AUDIO\_REC\_VAD\_END**

Vad stop

enumerator **AUDIO\_REC\_COMMAND\_DECT**

Form 0 is the id of the voice commands detected by Multinet

### Public Members

enum *audio\_rec\_evt\_t*::[anonymous] **type**

Audio recorder event type.

Event type

void \***event\_data**

Event data: For AUDIO\_REC\_WAKEUP\_START, event data is *recorder\_sr\_wakeup\_result\_t*  
For AUDIO\_REC\_COMMAND\_DECT or higher, event data is *recorder\_sr\_mn\_result\_t* For other events, event data is NULL

size\_t **data\_len**

Length of event data

struct **audio\_rec\_cfg\_t**

Audio recorder configuration.

### Public Members

int **pinned\_core**

Audio recorder task pinned to core

int **task\_prio**

Audio recorder task priority

int **task\_size**

Audio recorder task stack size

*rec\_event\_cb\_t* **event\_cb**

Event callback function, event type as *audio\_rec\_evt\_t* shown above

void \***user\_data**

Pointer to user data (optional)

*recorder\_data\_read\_t* **read**

Data callback function used feed data to audio recorder

void \***sr\_handle**

SR handle

*recorder\_sr\_iface\_t* \***sr\_iface**

SR interface

int **wakeup\_time**

Unit:ms. The duration that the wakeup state remains when VAD is not triggered

int **vad\_start**

Unit:ms. Consecutive speech frame will be judged to vad start

int **vad\_off**

Unit:ms. When the silence time exceeds this value, it is determined as AUDIO\_REC\_VAD\_END state

int **wakeup\_end**

Unit:ms. When the silence time after AUDIO\_REC\_VAD\_END state exceeds this value, it is determined as AUDIO\_REC\_WAKEUP\_END

void \***encoder\_handle**

Encoder handle

recorder\_encoder\_iface\_t \***encoder\_iface**

Encoder interface

## Macros

**AUDIO\_REC\_DEF\_TASK\_SZ**

Stack size of recorder task

**AUDIO\_REC\_DEF\_TASK\_PRIO**

Priority of recoder task

**AUDIO\_REC\_DEF\_TASK\_CORE**

Pinned to core

**AUDIO\_REC\_DEF\_WAKEUP\_TM**

Default wake up time (ms)

**AUDIO\_REC\_DEF\_WAKEEND\_TM**

Duration after vad off (ms)

**AUDIO\_REC\_VAD\_START\_SPEECH\_MS**

Consecutive speech frame will be judged to vad start (ms)

**AUDIO\_REC\_DEF\_VAD\_OFF\_TM**

Default vad off time (ms)

**AUDIO\_RECORDER\_DEFAULT\_CFG()**

### Type Definitions

typedef esp\_err\_t (**\*rec\_event\_cb\_t**)(*audio\_rec\_evt\_t* \*event, void \*user\_data)

Event Notification.

typedef struct \_\_audio\_recorder **\*audio\_rec\_handle\_t**

Audio recorder handle.

### Header File

- `components/audio_recorder/include/recorder_sr.h`

### Functions

*recorder\_sr\_handle\_t* **recorder\_sr\_create** (*recorder\_sr\_cfg\_t* \*cfg, recorder\_sr\_iface\_t \*\*iface)

Initialize sr processor, and the sr is disabled as default.

#### Parameters

- **cfg** – Configuration of sr
- **iface** – User interface provide by recorder sr

**Returns** NULL failed Others SR handle

esp\_err\_t **recorder\_sr\_destroy** (*recorder\_sr\_handle\_t* handle)

Destroy SR processor and recycle all resource.

**Parameters** **handle** – SR processor handle

**Returns** ESP\_OK ESP\_FAIL

esp\_err\_t **recorder\_sr\_reset\_speech\_cmd** (*recorder\_sr\_handle\_t* handle, char \*command\_str, char \*err\_phrase\_id)

Reset the speech commands.

#### Parameters

- **handle** – SR processor handle
- **command\_str** – String of the commands. more details on #2reset-api-on-the-fly
- **err\_phrase\_id** – error string output

**Returns** ESP\_OK ESP\_FAIL



## Structures

struct **recorder\_sr\_cfg\_t**

SR processor configuration.

---

**Note:** Since the detection of command words requires a clear starting point, the moment the wake word is detected is taken as the default start of detection. Therefore, if the wake word detection is disabled, the detection will use the `vad_state` detected by `esp-sr` as the start of detection. However, due to the fluctuation of this `vad_state`, the effectiveness of command word detection will be limited.

---

### Public Members

afe\_config\_t **afe\_cfg**

Configuration of AFE

int8\_t **input\_order**[DAT\_CH\_MAX]

Channel order of the input data

bool **multinet\_init**

Enable of speech command recognition

int **feed\_task\_core**

Core id of feed task

int **feed\_task\_prio**

Priority of feed task

int **feed\_task\_stack**

Stack size of feed task

int **fetch\_task\_core**

Core id of fetch task

int **fetch\_task\_prio**

Priority of fetch task

int **fetch\_task\_stack**

Stack size of fetch task

int **rb\_size**

Ringbuffer size of recorder sr

char \***partition\_label**

Partition label which stored the model data

char **\*mn\_language**

Command language for multinet to load

char **\*wn\_wakeword**

Wake Word for WakeNet to load. This is useful when multiple Wake Words are selected in sdkconfig. Setting this to NULL will use the first found model.

### Macros

**FEED\_TASK\_STACK\_SZ**

**FETCH\_TASK\_STACK\_SZ**

**FEED\_TASK\_PRIO**

**FETCH\_TASK\_PRIO**

**FEED\_TASK\_PINNED\_CORE**

**FETCH\_TASK\_PINNED\_CORE**

**SR\_OUTPUT\_RB\_SIZE**

**INPUT\_ORDER\_DEFAULT** ()

**DEFAULT\_RECORDER\_SR\_CFG** ()

### Type Definitions

typedef void **\*recorder\_sr\_handle\_t**

SR processor handle.

### Header File

- [components/audio\\_recorder/include/recorder\\_encoder.h](#)

## Functions

*recorder\_encoder\_handle\_t* **recorder\_encoder\_create** (*recorder\_encoder\_cfg\_t* \*cfg,  
recorder\_encoder\_iface\_t \*\*iface)

Initialize encoder processor, and the encoder is disabled as default.

### Parameters

- **cfg** – Configuration of encoder
- **iface** – User interface provide by recorder encoder

**Returns** NULL failed Others encoder handle

esp\_err\_t **recorder\_encoder\_destroy** (*recorder\_encoder\_handle\_t* handle)

Destroy encoder processor and recycle all resource.

**Parameters** **handle** – Encoder processor handle

**Returns** ESP\_OK ESP\_FAIL

## Structures

struct **recorder\_encoder\_cfg\_t**  
recorder encoder configuration parameters

### Public Members

*audio\_element\_handle\_t* **resample**  
Handle of resample

*audio\_element\_handle\_t* **encoder**  
Handle of encoder

## Type Definitions

typedef void \***recorder\_encoder\_handle\_t**  
encoder handle

## 2.8 Peripherals

There are several peripherals available in the ESP-ADF, ranging from buttons and LEDs to SD Card or Wi-Fi. The peripherals are implemented using common API that is then expanded with peripheral specific functionality. The following description covers common functionality.

### 2.8.1 ESP Peripherals

This library simplifies the management of peripherals, by pooling and monitoring in a single task, adding basic functions to send and receive events. And it also provides APIs to easily integrate new peripherals.

---

**Note:** Note that if you do not intend to integrate new peripherals into `esp_peripherals`, you are only interested in simple api `esp_periph_init`, `esp_periph_start`, `esp_periph_stop` and `esp_periph_destroy`. If you want to integrate new peripherals, please refer to *Periph Button* source code

---

#### Examples

Please refer to `player/pipeline_http_mp3/main/play_http_mp3_example.c`.

#### API Reference

##### Header File

- `components/esp_peripherals/include/esp_peripherals.h`

##### Functions

*esp\_periph\_set\_handle\_t* **esp\_periph\_set\_init** (*esp\_periph\_config\_t* \*config)

Initialize `esp_periph_set` sets, create empty peripherals list. Call this function before starting any peripherals (with `esp_periph_start`). This call will initialize the data needed for `esp_peripherals` to work, but does not actually create the task. The `event_handle` is optional if you want to receive events from this callback function. The `esp_peripherals` task will send all events out to `event_iface`, can be listen by `event_iface` by `esp_periph_get_event_iface`. The `user_context` will sent `esp_periph_event_handle_t` as `*context` parameter.

**Parameters** `config` – [in] The configurations

**Returns** The peripheral sets instance

`esp_err_t` **esp\_periph\_set\_destroy** (*esp\_periph\_set\_handle\_t* periph\_set\_handle)

This function will stop and kill the monitor task, calling all destroy callback functions of the peripheral (so you do not need to destroy the peripheral object manually). It will also remove all memory allocated to the peripherals list, so you need to call the `esp_periph_set_init` function again if you want to use it.

**Parameters** `periph_set_handle` – The `esp_periph_set_handle_t` instance

**Returns**

- `ESP_OK`
- `ESP_FAIL`

`esp_err_t` **esp\_periph\_set\_stop\_all** (*esp\_periph\_set\_handle\_t* periph\_set\_handle)

Stop monitoring all peripherals, the peripheral state is still kept. This function only temporary disables the peripheral.

**Parameters** `periph_set_handle` – The `esp_periph_set_handle_t` instance

**Returns**

- ESP\_OK
- ESP\_FAIL

*esp\_periph\_handle\_t* **esp\_periph\_set\_get\_by\_id** (*esp\_periph\_set\_handle\_t* periph\_set\_handle, int periph\_id)

Get the peripheral handle by Peripheral ID.

**Parameters**

- **periph\_set\_handle** – The *esp\_periph\_set\_handle\_t* instance
- **periph\_id** – [in] as *esp\_periph\_id\_t*, or any ID you use when calling *esp\_periph\_create*

**Returns** The *esp\_periph\_handle\_t*

*audio\_event\_iface\_handle\_t* **esp\_periph\_set\_get\_event\_iface** (*esp\_periph\_set\_handle\_t* periph\_set\_handle)

Return the event\_iface used by this esp\_peripherals.

**Parameters** **periph\_set\_handle** – The *esp\_periph\_set\_handle\_t* instance

**Returns** The audio event iface handle

*esp\_err\_t* **esp\_periph\_set\_register\_callback** (*esp\_periph\_set\_handle\_t* periph\_set\_handle, *esp\_periph\_event\_handle\_t* cb, void \*user\_context)

Register peripheral sets event callback function.

**Parameters**

- **periph\_set\_handle** – The *esp\_periph\_set\_handle\_t* instance
- **cb** – The event handle callback function
- **user\_context** – The user context pointer

**Returns**

- ESP\_OK
- ESP\_FAIL

*QueueHandle\_t* **esp\_periph\_set\_get\_queue** (*esp\_periph\_set\_handle\_t* periph\_set\_handle)

Peripheral is using event\_iface to control the event, all events are send out to event\_iface queue. This function will be useful in case we want to read events directly from the event\_iface queue.

**Parameters** **periph\_set\_handle** – The *esp\_periph\_set\_handle\_t* instance

**Returns** The queue handle

*esp\_err\_t* **esp\_periph\_set\_list\_init** (*esp\_periph\_set\_handle\_t* periph\_set\_handle)

Call this function to initialize all the listed peripherals.

---

**Note:** Work with no task peripheral set only

---

**Parameters** **periph\_set\_handle** – The *esp\_periph\_set\_handle\_t* instance

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **esp\_periph\_set\_list\_run** (*esp\_periph\_set\_handle\_t* periph\_set\_handle, *audio\_event\_iface\_msg\_t* msg)

Call this function to run all the listed peripherals.

---

**Note:** Work with no task peripheral set only

---

### Parameters

- **periph\_set\_handle** – The *esp\_periph\_set\_handle\_t* instance
- **msg** – The *audio\_event\_iface\_msg\_t* handle message

### Returns

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **esp\_periph\_set\_list\_destroy** (*esp\_periph\_set\_handle\_t* periph\_set\_handle)

Call this function to destroy all the listed peripherals.

---

**Note:** Work with no task peripheral set only

---

**Parameters** **periph\_set\_handle** – The *esp\_periph\_set\_handle\_t* instance

### Returns

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **esp\_periph\_remove\_from\_set** (*esp\_periph\_set\_handle\_t* periph\_set\_handle, *esp\_periph\_handle\_t* periph)

Call this function to remove periph from periph\_set.

### Parameters

- **periph\_set\_handle** – The *esp\_periph\_set\_handle\_t* instance
- **periph** – The *esp\_periph\_handle\_t* instance

### Returns

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **esp\_periph\_alloc\_periph\_id** (*esp\_periph\_set\_handle\_t* periph\_set\_handle, int \*periph\_id)

Allocated a peripheral ID from the periph\_set.

---

**Note:** This function can apply for a peripheral ID to realize a customized peripheral module. The requested peripheral ID will not be included in *esp\_periph\_id\_t*

---

### Parameters

- **periph\_set\_handle** – [in] The *esp\_periph\_set\_handle\_t* instance

- **periph\_id** – [out] The peripheral identifier

#### Returns

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **esp\_periph\_set\_change\_waiting\_time** (*esp\_periph\_set\_handle\_t* periph\_set\_handle, int time\_ms)

Call this function to change periph\_set waiting time.

#### Parameters

- **periph\_set\_handle** – The esp\_periph\_set\_handle\_t instance
- **time\_ms** – The waiting time

#### Returns

- ESP\_OK
- ESP\_FAIL

*esp\_periph\_handle\_t* **esp\_periph\_create** (int periph\_id, const char \*tag)

Call this function to initialize a new peripheral.

#### Parameters

- **periph\_id** – [in] The periph identifier
- **tag** – [in] The tag name, we named it easy to get in debug logs

**Returns** The peripheral handle

esp\_err\_t **esp\_periph\_set\_function** (*esp\_periph\_handle\_t* periph, *esp\_periph\_func* init, *esp\_periph\_run\_func* run, *esp\_periph\_func* destroy)

Each peripheral has a cycle of sequential operations from initialization, execution of commands to destroying the peripheral. These operations are represented by functions passed as call parameters to this function.

#### Parameters

- **periph** – [in] The periph
- **init** – [in] The initialize
- **run** – [in] The run
- **destroy** – [in] The destroy

#### Returns

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **esp\_periph\_start** (*esp\_periph\_set\_handle\_t* periph\_set\_handle, *esp\_periph\_handle\_t* periph)

Add the peripheral to peripherals list, enable and start monitor task (if task stack size > 0)

---

**Note:** This peripheral must be first created by calling `esp_periph_create`

---

#### Parameters

- **periph\_set\_handle** – [in] The esp\_periph\_set\_handle\_t instance

- **periph** – [in] The peripheral instance

### Returns

- ESP\_OK on success
- ESP\_FAIL when any errors

esp\_err\_t **esp\_periph\_stop** (*esp\_periph\_handle\_t* periph)

Stop monitoring the peripheral, the peripheral state is still kept. This function only temporarily disables the peripheral.

**Parameters** **periph** – [in] The periph

### Returns

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **esp\_periph\_send\_cmd** (*esp\_periph\_handle\_t* periph, int cmd, void \*data, int data\_len)

When this function is called, the command is passed to the event\_iface command queue, and the `esp_periph_run_func` of this peripheral will be executed in the main peripheral task. This function can be called from any task, basically it only sends a queue to the main peripheral task.

### Parameters

- **periph** – [in] The periph
- **cmd** – [in] The command
- **data** – The data
- **data\_len** – [in] The data length

### Returns

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **esp\_periph\_send\_cmd\_from\_isr** (*esp\_periph\_handle\_t* periph, int cmd, void \*data, int data\_len)

Similar to `esp_periph_send_cmd`, but it can be called in the hardware interrupt handle.

### Parameters

- **periph** – [in] The periph
- **cmd** – [in] The command
- **data** – The data
- **data\_len** – [in] The data length

### Returns

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **esp\_periph\_send\_event** (*esp\_periph\_handle\_t* periph, int event\_id, void \*data, int data\_len)

In addition to sending an event via `event_iface`, this function will dispatch the `event_handle` callback if the `event_handle` callback is provided at `esp_periph_init`

### Parameters

- **periph** – [in] The peripheral



- **event\_id** – [in] The event identifier
- **data** – The data
- **data\_len** – [in] The data length

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **esp\_periph\_start\_timer** (*esp\_periph\_handle\_t* periph, TickType\_t interval\_tick, *timer\_callback* callback)

Each peripheral can initialize a timer, which is by default NULL. When this function is called, the timer for the peripheral is created and it invokes the callback function every interval tick.

**Note:**

- You do not need to stop or destroy the timer, when the `esp_periph_destroy` function is called, it will stop and destroy all
- This timer using FreeRTOS Timer, with `autoreload = true`

**Parameters**

- **periph** – [in] The peripheral
- **interval\_tick** – [in] The interval tick
- **callback** – [in] The callback

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **esp\_periph\_stop\_timer** (*esp\_periph\_handle\_t* periph)

Stop peripheral timer.

**Parameters** **periph** – [in] The peripheral

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **esp\_periph\_set\_data** (*esp\_periph\_handle\_t* periph, void \*data)

Set the user data.

---

**Note:** Make sure the `data` lifetime is sufficient, this function does not copy all data, it only stores the data pointer

---

**Parameters**

- **periph** – [in] The peripheral
- **data** – The data

**Returns**

- ESP\_OK
- ESP\_FAIL

void \***esp\_periph\_get\_data** (*esp\_periph\_handle\_t* periph)

Get the user data stored in the peripheral.

**Parameters** **periph** – [in] The peripheral

**Returns** Peripheral data pointer

*esp\_periph\_state\_t* **esp\_periph\_get\_state** (*esp\_periph\_handle\_t* periph)

Get the current state of peripheral.

**Parameters** **periph** – [in] The handle of peripheral

**Returns** The peripheral working state

*esp\_periph\_id\_t* **esp\_periph\_get\_id** (*esp\_periph\_handle\_t* periph)

Get Peripheral identifier.

**Parameters** **periph** – [in] The peripheral

**Returns** The peripheral identifier

esp\_err\_t **esp\_periph\_set\_id** (*esp\_periph\_handle\_t* periph, *esp\_periph\_id\_t* periph\_id)

Set Peripheral identifier.

**Parameters**

- **periph** – [in] The peripheral
- **periph\_id** – [in] The peripheral identifier

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **esp\_periph\_init** (*esp\_periph\_handle\_t* periph)

Call this to execute `init` function of peripheral instance.

**Parameters** **periph** – The peripheral handle

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **esp\_periph\_run** (*esp\_periph\_handle\_t* periph)

Call this to execute `run` function of peripheral instance.

**Parameters** **periph** – The peripheral handle

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **esp\_periph\_destroy** (*esp\_periph\_handle\_t* periph)

Call this to execute `destroy` function of peripheral instance.

**Parameters** **periph** – The peripheral handle

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **esp\_periph\_register\_on\_events** (*esp\_periph\_handle\_t* periph, *esp\_periph\_event\_t* \*evts)

Register peripheral on event handle.

**Parameters**

- **periph** – The peripheral handle
- **evts** – The esp\_periph\_event\_t handle

**Returns**

- ESP\_OK
- ESP\_FAIL

**Structures**

struct **esp\_periph\_config\_t**

Common peripherals configurations.

**Public Members**

int **task\_stack**

>0 Service task stack size; =0 without task created

int **task\_prio**

Service task priority (based on freeRTOS priority)

int **task\_core**

Service task running in core (0 or 1)

bool **extern\_stack**

Service task stack allocate on extern ram

struct **esp\_periph\_event**

peripheral events

### Public Members

`void *user_ctx`  
peripheral context data

`esp_periph_event_handle_t cb`  
peripheral callback function

`audio_event_iface_handle_t iface`  
peripheral event

### Macros

`DEFAULT_ESP_PERIPH_STACK_SIZE`

`DEFAULT_ESP_PERIPH_TASK_PRIO`

`DEFAULT_ESP_PERIPH_TASK_CORE`

`DEFAULT_ESP_PERIPH_SET_CONFIG()`

`periph_tick_get`

### Type Definitions

`typedef struct esp_periph_sets *esp_periph_set_handle_t`

`typedef struct esp_periph *esp_periph_handle_t`

`typedef esp_err_t (*esp_periph_func)(esp_periph_handle_t periph)`

`typedef esp_err_t (*esp_periph_run_func)(esp_periph_handle_t periph, audio_event_iface_msg_t *msg)`

`typedef esp_err_t (*esp_periph_event_handle_t)(audio_event_iface_msg_t *event, void *context)`

`typedef void (*timer_callback)(xTimerHandle tmr)`

`typedef struct esp_periph_event esp_periph_event_t`  
peripheral events

## Enumerations

enum **esp\_periph\_id\_t**

Peripheral Identify, this must be unique for each peripheral added to the peripherals list.

*Values:*

enumerator **PERIPH\_ID\_BUTTON**

enumerator **PERIPH\_ID\_TOUCH**

enumerator **PERIPH\_ID\_SDCARD**

enumerator **PERIPH\_ID\_WIFI**

enumerator **PERIPH\_ID\_FLASH**

enumerator **PERIPH\_ID\_AUXIN**

enumerator **PERIPH\_ID\_ADC**

enumerator **PERIPH\_ID\_CONSOLE**

enumerator **PERIPH\_ID\_BLUETOOTH**

enumerator **PERIPH\_ID\_LED**

enumerator **PERIPH\_ID\_SPIFFS**

enumerator **PERIPH\_ID\_ADC\_BTN**

enumerator **PERIPH\_ID\_IS31FL3216**

enumerator **PERIPH\_ID\_GPIO\_ISR**

enumerator **PERIPH\_ID\_WS2812**

enumerator **PERIPH\_ID\_AW2013**

enumerator **PERIPH\_ID\_LCD**

enumerator **PERIPH\_ID\_CUSTOM\_BASE**

enum **esp\_periph\_state\_t**

Peripheral working state.

*Values:*

enumerator **PERIPH\_STATE\_NULL**

enumerator **PERIPH\_STATE\_INIT**

enumerator **PERIPH\_STATE\_RUNNING**

enumerator **PERIPH\_STATE\_PAUSE**

enumerator **PERIPH\_STATE\_STOPPING**

enumerator **PERIPH\_STATE\_ERROR**

enumerator **PERIPH\_STATE\_STATUS\_MAX**

The peripheral specific functionality is available by calling dedicated functions described below. Some peripherals are available on both *ESP32-LyraT* and *ESP32-LyraTD-MSC* development boards, some on a specific board only. The following table provides all implemented peripherals broken down by development board.

### 2.8.2 Wi-Fi Peripheral

The Wi-Fi Peripheral is used to configure Wi-Fi connections, provide APIs to control Wi-Fi connection configuration, as well as monitor the status of Wi-Fi networks.

#### Application Example

Implementation of this API is demonstrated in `player/pipeline_http_mp3` example.

#### API Reference

##### Header File

- `components/esp_peripherals/include/periph_wifi.h`

## Functions

*esp\_periph\_handle\_t* **periph\_wifi\_init** (*periph\_wifi\_cfg\_t* \*config)

Create the wifi peripheral handle for esp\_peripherals.

---

**Note:** The handle was created by this function automatically destroy when *esp\_periph\_destroy* is called

---

**Parameters** *config* – The configuration

**Returns** The esp peripheral handle

*esp\_err\_t* **periph\_wifi\_wait\_for\_connected** (*esp\_periph\_handle\_t* periph, TickType\_t tick\_to\_wait)

This function will block current thread (in *tick\_to\_wait* tick) and wait until ESP32 connected to the Wi-Fi network, and got ip.

**Parameters**

- **periph** – [in] The periph
- **tick\_to\_wait** – [in] The tick to wait

**Returns**

- ESP\_OK
- ESP\_FAIL

*periph\_wifi\_state\_t* **periph\_wifi\_is\_connected** (*esp\_periph\_handle\_t* periph)

Check the Wi-Fi connection status.

**Parameters** *periph* – [in] The periph

**Returns** Wi-Fi network status

*esp\_err\_t* **esp\_wifi\_set\_listen\_interval** (*esp\_periph\_handle\_t* periph, int interval)

Set Wi-Fi listen interval for ESP32 station to receive beacon.

**Parameters**

- **periph** – [in] The wifi periph
- **interval** – [in] listen interval. units: AP beacon intervals(see BcnInt, default: 100ms)

**Returns**

- ESP\_OK
- ESP\_FAIL

*esp\_err\_t* **periph\_wifi\_config\_start** (*esp\_periph\_handle\_t* periph, *periph\_wifi\_config\_mode\_t* mode)

Start Wi-Fi network setup in mode

**Parameters**

- **periph** – [in] The periph
- **mode** – [in] The mode

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **periph\_wifi\_config\_wait\_done** (*esp\_periph\_handle\_t* periph, TickType\_t tick\_to\_wait)

Wait for Wi-Fi setup done.

### Parameters

- **periph** – [in] The periph
- **tick\_to\_wait** – [in] The tick to wait

### Returns

- ESP\_OK
- ESP\_FAIL

## Structures

struct **periph\_wpa2\_enterprise\_cfg\_t**

The WPA2 enterprise peripheral configuration.

### Public Members

bool **disable\_wpa2\_e**

Disable wpa2 enterprise

int **eap\_method**

TLS: 0, PEAP: 1, TTLS: 2

char \***ca\_pem\_start**

binary wpa2 ca pem start

char \***ca\_pem\_end**

binary wpa2 ca pem end

char \***wpa2\_e\_cert\_start**

binary wpa2 cert start

char \***wpa2\_e\_cert\_end**

binary wpa2 cert end

char \***wpa2\_e\_key\_start**

binary wpa2 key start

char \***wpa2\_e\_key\_end**

binary wpa2 key end

const char \***eap\_id**

Identity in phase 1 of EAP procedure



const char \***eap\_username**  
Username for EAP method (PEAP and TTLS)

const char \***eap\_password**  
Password for EAP method (PEAP and TTLS)

struct **periph\_wifi\_cfg\_t**  
The Wi-Fi peripheral configuration.

### Public Members

bool **disable\_auto\_reconnect**  
Disable Wi-Fi auto reconnect

int **reconnect\_timeout\_ms**  
The reconnect timeout after disconnected from Wi-Fi network

wifi\_config\_t **wifi\_config**  
Wifi configure

*periph\_wpa2\_enterprise\_cfg\_t* **wpa2\_e\_cfg**  
wpa2 enterprise config

### Enumerations

enum **periph\_wifi\_state\_t**  
Peripheral Wi-Fi event id.

*Values:*

enumerator **PERIPH\_WIFI\_UNCHANGE**

enumerator **PERIPH\_WIFI\_CONNECTING**

enumerator **PERIPH\_WIFI\_CONNECTED**

enumerator **PERIPH\_WIFI\_DISCONNECTED**

enumerator **PERIPH\_WIFI\_SETTING**

enumerator **PERIPH\_WIFI\_CONFIG\_DONE**

enumerator **PERIPH\_WIFI\_CONFIG\_ERROR**

enumerator **PERIPH\_WIFI\_ERROR**

enum **periph\_wifi\_config\_mode\_t**

Wi-Fi setup mode type.

*Values:*

enumerator **WIFI\_CONFIG\_ESPTOUCH**

Using smartconfig with ESPTOUCH protocol

enumerator **WIFI\_CONFIG\_AIRKISS**

Using smartconfig with AIRKISS protocol

enumerator **WIFI\_CONFIG\_ESPTOUCH\_AIRKISS**

Using smartconfig with ESPTOUCH\_AIRKISS protocol

enumerator **WIFI\_CONFIG\_WPS**

Using WPS (not support)

enumerator **WIFI\_CONFIG\_BLUEFI**

Using BLUEFI

enumerator **WIFI\_CONFIG\_WEB**

Using HTTP Server (not support)

### 2.8.3 SD Card Peripheral

If your board has a SD card connected, use this API to initialize, mount and unmount the card, see functions `periph_sdcard_init()`, `periph_sdcard_mount()` and `periph_sdcard_unmount()`. The data reading / writing is implemented in a separate API described in *FatFs Stream*.

#### Application Examples

Implementation of this API is demonstrated in couple of examples:

- `player/pipeline_play_sdcard_music`
- `player/pipeline_sdcard_mp3_control`
- `recorder/pipeline_recording_to_sdcard`
- `recorder/pipeline_wav_amr_sdcard`

## API Reference

### Header File

- components/esp\_peripherals/include/periph\_sdcard.h

### Functions

*esp\_periph\_handle\_t* **periph\_sdcard\_init** (*periph\_sdcard\_cfg\_t* \*sdcard\_config)

Create the sdcard peripheral handle for esp\_peripherals.

---

**Note:** The handle was created by this function automatically destroy when `esp_periph_destroy` is called

---

**Parameters** **sdcard\_config** – The sdcard configuration

**Returns** The esp peripheral handle

bool **periph\_sdcard\_is\_mounted** (*esp\_periph\_handle\_t* periph)

Check the sdcard is mounted or not.

**Parameters** **periph** – [in] The periph

**Returns** SDCARD mounted state

### Structures

struct **periph\_sdcard\_cfg\_t**

The SD Card Peripheral configuration.

#### Public Members

int **card\_detect\_pin**

Card detect gpio number

const char \***root**

Base path for vfs

*periph\_sdcard\_mode\_t* **mode**

card mode

### Enumerations

enum **periph\_sdcard\_event\_id\_t**

Peripheral sdcard event id.

*Values:*

enumerator **SDCARD\_STATUS\_UNKNOWN**

No event

enumerator **SDCARD\_STATUS\_CARD\_DETECT\_CHANGE**

Detect changes in the card\_detect pin

enumerator **SDCARD\_STATUS\_MOUNTED**

SDCARD mounted successfully

enumerator **SDCARD\_STATUS\_UNMOUNTED**

SDCARD unmounted successfully

enumerator **SDCARD\_STATUS\_MOUNT\_ERROR**

SDCARD mount error

enumerator **SDCARD\_STATUS\_UNMOUNT\_ERROR**

SDCARD unmount error

enum **periph\_sdcard\_mode\_t**

SD card mode, SPI, 1-line SD mode, 4-line SD mode.

*Values:*

enumerator **SD\_MODE\_SPI**

sd\_card SPI

enumerator **SD\_MODE\_1\_LINE**

sd\_card 1-line SD mode

enumerator **SD\_MODE\_4\_LINE**

sd\_card 4-line SD mode

enumerator **SD\_MODE\_8\_LINE**

sd\_card 8-line SD mode

enumerator **SD\_MODE\_MAX**

## 2.8.4 Spiffs Peripheral

Use this API to initialize, mount and unmount spiffs partition, see functions `periph_spiffs_init()`, `periph_spiffs_mount()` and `periph_spiffs_unmount()`. The data reading / writing is implemented in a separate API described in *SPIFFS Stream*.

### Application Example

Implementation of this API is demonstrated in `audio_processing/pipeline_spiffs_amr_resample` example.

### API Reference

#### Header File

- `components/esp_peripherals/include/periph_spiffs.h`

#### Functions

`esp_periph_handle_t` **periph\_spiffs\_init** (`periph_spiffs_cfg_t` \*spiffs\_config)

Create the spiffs peripheral handle for esp\_peripherals.

---

**Note:** The handle created by this function will be automatically destroyed when `esp_periph_destroy` is called

---

**Parameters** `spiffs_config` – The spiffs configuration

**Returns** The esp peripheral handle

`bool` **periph\_spiffs\_is\_mounted** (`esp_periph_handle_t` periph)

Check if the SPIFFS is mounted or not.

**Parameters** `periph` – [in] The periph

**Returns** SPIFFS mounted state

#### Structures

`struct` **periph\_spiffs\_cfg\_t**

The SPIFFS Peripheral configuration.

### Public Members

const char **\*root**

Base path for vfs

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 number of 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.

### Enumerations

enum **periph\_spiffs\_event\_id\_t**

Peripheral spiffs event id.

*Values:*

enumerator **SPIFFS\_STATUS\_UNKNOWN**

No event

enumerator **SPIFFS\_STATUS\_MOUNTED**

SPIFFS mounted successfully

enumerator **SPIFFS\_STATUS\_UNMOUNTED**

SPIFFS unmounted successfully

enumerator **SPIFFS\_STATUS\_MOUNT\_ERROR**

SPIFFS mount error

enumerator **SPIFFS\_STATUS\_UNMOUNT\_ERROR**

SPIFFS unmount error

## 2.8.5 Console Peripheral

Console Peripheral is used to control the Audio application from the terminal screen. It provides 2 ways to execute command, one sends an event to `esp_peripherals` (for a command without parameters), another calls a callback function (need parameters). If there is a callback function, no event will be sent.

Please make sure that the lifetime of `periph_console_cmd_t` must be ensured during console operation, `periph_console_init()` only reference, does not make a copy.

## Code example

Please refer to `cli/main/console_example.c`.

## API Reference

### Header File

- `components/esp_peripherals/include/periph_console.h`

### Functions

*esp\_periph\_handle\_t* **periph\_console\_init** (*periph\_console\_cfg\_t* \*config)

Initialize Console Peripheral.

**Parameters** `config` – The configuration

**Returns** The esp peripheral handle

### Structures

struct **periph\_console\_cmd\_t**

Command structure.

#### Public Members

const char \***cmd**

Name of command, must be unique

int **id**

Command ID will be sent together when the command is matched

const char \***help**

Explanation of the command

*console\_cmd\_callback\_t* **func**

Function callback for the command

struct **periph\_console\_cfg\_t**

Console Peripheral configuration.

### Public Members

int **command\_num**

Total number of commands

const *periph\_console\_cmd\_t* \***commands**

Pointer to array of commands

int **task\_stack**

Console task stack, using default if the value is zero

int **task\_prio**

Console task priority (based on freeRTOS priority), using default if the value is zero

int **buffer\_size**

Size of console input buffer

const char \***prompt\_string**

Console prompt string, using default CONSOLE\_PROMPT\_STRING if the pointer is NULL

### Macros

**CONSOLE\_DEFAULT\_TASK\_PRIO**

**CONSOLE\_DEFAULT\_TASK\_STACK**

**CONSOLE\_DEFAULT\_BUFFER\_SIZE**

**CONSOLE\_DEFAULT\_PROMPT\_STRING**

### Type Definitions

typedef esp\_err\_t (\***console\_cmd\_callback\_t**)(*esp\_periph\_handle\_t* periph, int argc, char \*argv[])

## 2.8.6 Touch Peripheral

Initialize ESP32 touchpad peripheral and retrieve information from the touch sensors.



## Application Example

Implementation of this API is demonstrated in [get-started/play\\_mp3\\_control](#) example.

## API Reference

### Header File

- [components/esp\\_peripherals/include/periph\\_touch.h](#)

### Functions

*esp\_periph\_handle\_t* **periph\_touch\_init** (*periph\_touch\_cfg\_t* \*config)

Create the touch peripheral handle for esp\_peripherals.

---

**Note:** The handle was created by this function automatically destroy when `esp_periph_destroy` is called

---

**Parameters** `config` – The configuration

**Returns** The esp peripheral handle

### Structures

struct **periph\_touch\_cfg\_t**

The Touch peripheral configuration.

#### Public Members

int **touch\_mask**

Touch pad mask using for this Touch peripheral, ex: TOUCH\_PAD\_SEL0 | TOUCH\_PAD\_SEL1

int **tap\_threshold\_percent**

Tap threshold percent, Tap event will be determined if the percentage value is less than the non-touch value

int **long\_tap\_time\_ms**

Long tap duration in milliseconds, default is 2000ms, PERIPH\_TOUCH\_LONG\_TAP will be occurred if TAP and time hold longer than this value

### Enumerations

enum **esp\_touch\_pad\_sel\_t**

Touch pad selection.

*Values:*

enumerator **TOUCH\_PAD\_SEL0**

enumerator **TOUCH\_PAD\_SEL1**

enumerator **TOUCH\_PAD\_SEL2**

enumerator **TOUCH\_PAD\_SEL3**

enumerator **TOUCH\_PAD\_SEL4**

enumerator **TOUCH\_PAD\_SEL5**

enumerator **TOUCH\_PAD\_SEL6**

enumerator **TOUCH\_PAD\_SEL7**

enumerator **TOUCH\_PAD\_SEL8**

enumerator **TOUCH\_PAD\_SEL9**

enum **periph\_touch\_event\_id\_t**

Peripheral touch event id.

*Values:*

enumerator **PERIPH\_TOUCH\_UNCHANGE**

No event

enumerator **PERIPH\_TOUCH\_TAP**

When touch pad is tapped

enumerator **PERIPH\_TOUCH\_RELEASE**

When touch pad is released after tap

enumerator **PERIPH\_TOUCH\_LONG\_TAP**

When touch pad is tapped and held after `long_tap_time_ms` time

enumerator **PERIPH\_TOUCH\_LONG\_RELEASE**

When touch pad is released after long tap

## 2.8.7 Button Peripheral

To control application flow you may use buttons connected and read through the ESP32 GPIOs. This API provides functions to initialize specific GPIOs and obtain information on button events such as when it has been pressed, when released, when pressed for a long time and released after long press. To get information on a particular event, establish a callback function with `button_dev_add_tap_cb()` or `button_dev_add_press_cb()`.

### Application Example

Implementation of this API is demonstrated in `cloud_services/google_translate_device` example.

### API Reference

#### Header File

- `components/esp_peripherals/include/periph_button.h`

#### Functions

`esp_periph_handle_t periph_button_init (periph_button_cfg_t *but_cfg)`

Create the button peripheral handle for esp\_peripherals.

---

**Note:** The handle was created by this function automatically destroy when `esp_periph_destroy` is called

---

**Parameters** `but_cfg` – The but configuration

**Returns** The esp peripheral handle

#### Structures

struct `periph_button_cfg_t`

The Button peripheral configuration.

#### Public Members

`uint64_t gpio_mask`

GPIO Mask using for this Button peripheral, it is `BIT(GPIO_NUM)`, ex: `GPIO_SEL_36 | GPIO_SEL_36`

`int long_press_time_ms`

Long press duration in milliseconds, default is 2000ms

### Enumerations

enum `periph_button_event_id_t`

Peripheral button event id.

*Values:*

enumerator `PERIPH_BUTTON_UNCHANGE`

No event

enumerator `PERIPH_BUTTON_PRESSED`

When button is pressed

enumerator `PERIPH_BUTTON_RELEASE`

When button is released

enumerator `PERIPH_BUTTON_LONG_PRESSED`

When button is pressed and kept for more than `long_press_time_ms`

enumerator `PERIPH_BUTTON_LONG_RELEASE`

When button is released and event `PERIPH_BUTTON_LONG_PRESSED` happened

### 2.8.8 LED Peripheral

Blink or fade a LED connected to a GPIO with configurable On and Off times.

#### Application Examples

Implementation of this API is demonstrated in couple of examples:

- `/cloud_services/google_translate_device`
- `/dueros`

#### API Reference

##### Header File

- `components/esp_peripherals/include/periph_led.h`

## Functions

*esp\_periph\_handle\_t* **periph\_led\_init** (*periph\_led\_cfg\_t* \*config)

Create the LED peripheral handle for esp\_peripherals.

---

**Note:** The handle was created by this function automatically destroy when *esp\_periph\_destroy* is called

---

**Parameters** **config** – The configuration

**Returns** The esp peripheral handle

*esp\_err\_t* **periph\_led\_blink** (*esp\_periph\_handle\_t* periph, int gpio\_num, int time\_on\_ms, int time\_off\_ms, bool fade, int loop, *periph\_led\_idle\_level\_t* level)

Blink LED Peripheral, this function will automatically configure the *gpio\_num* to control the LED, with *time\_on\_ms* as the time (in milliseconds) switch from OFF to ON (or ON if fade is disabled), and *time\_off\_ms* as the time (in milliseconds) switch from ON to OFF (or OFF if fade is disabled). When switching from ON -> OFF and vice versa, the loop decreases once, and will turn off the effect when the loop is 0. With a loop value less than 0, the LED effect will loop endlessly. PERIPH\_LED\_BLINK\_FINISH events will be sent at each end of loop.

### Parameters

- **periph** – [in] The LED periph
- **gpio\_num** – [in] The gpio number
- **time\_on\_ms** – [in] The time on milliseconds
- **time\_off\_ms** – [in] The time off milliseconds
- **fade** – [in] Fading enabled
- **loop** – [in] Loop
- **level** – [in] idle level

### Returns

- ESP\_OK
- ESP\_FAIL

*esp\_err\_t* **periph\_led\_stop** (*esp\_periph\_handle\_t* periph, int gpio\_num)

Stop Blink the LED.

### Parameters

- **periph** – [in] The periph
- **gpio\_num** – [in] The gpio number

### Returns

- ESP\_OK
- ESP\_FAIL

### Structures

struct **periph\_led\_cfg\_t**

The LED peripheral configuration.

#### Public Members

ledc\_mode\_t **led\_speed\_mode**

LEDC speed speed\_mode, high-speed mode or low-speed mode

ledc\_timer\_bit\_t **led\_duty\_resolution**

LEDC channel duty resolution

ledc\_timer\_t **led\_timer\_num**

Select the timer source of channel (0 - 3)

uint32\_t **led\_freq\_hz**

LEDC timer frequency (Hz)

int **gpio\_num**

Optional, < 0 invalid gpio number

### Enumerations

enum **periph\_led\_event\_id\_t**

Peripheral LED event id.

*Values:*

enumerator **PERIPH\_LED\_UNCHANGE**

No event

enumerator **PERIPH\_LED\_BLINK\_FINISH**

When LED blink is finished

enum **periph\_led\_idle\_level\_t**

Peripheral LED idle output level.

*Values:*

enumerator **PERIPH\_LED\_IDLE\_LEVEL\_LOW**

Low level output

enumerator **PERIPH\_LED\_IDLE\_LEVEL\_HIGH**

High level output

## 2.8.9 ADC Button Peripheral

Read status of buttons connected to an ADC input using a resistor ladder. Configuration provides for more than a single ADC input to read several sets of buttons. For an example hardware implementation please refer to schematic of [ESP32-LyraTD-MSC V2.2 Upper Board \(PDF\)](#).

### Application Examples

Implementation of this API is demonstrated in the following example:

- [checks/check\\_board\\_buttons](#)

### API Reference

#### Header File

- [components/esp\\_peripherals/include/periph\\_adc\\_button.h](#)

#### Functions

*esp\_periph\_handle\_t* **periph\_adc\_button\_init** (*periph\_adc\_button\_cfg\_t* \*btn\_cfg)

Create the button peripheral handle for esp\_peripherals.

---

**Note:** The handle created by this function is automatically destroyed when `esp_periph_destroy` is called.

---

**Parameters** `btn_cfg` – The button configuration.

**Returns** The esp peripheral handle.

#### Structures

struct **periph\_adc\_button\_cfg\_t**

The configuration of ADC Button.

#### Public Members

`adc_arr_t` \***arr**

An array with configuration of buttons

int **arr\_size**

The array size

`adc_btn_task_cfg_t` **task\_cfg**

Adc button task configuration

### Macros

`ADC_BUTTON_STACK_SIZE`

`ADC_BUTTON_TASK_PRIORITY`

`ADC_BUTTON_TASK_CORE_ID`

`PERIPH_ADC_BUTTON_DEFAULT_CONFIG()`

`ADC_DEFAULT_ARR()`

ESP32 ADC1 channels and GPIO table  
ADC1\_CHANNEL\_0 - GPIO36 ADC1\_CHANNEL\_1 - GPIO37  
ADC1\_CHANNEL\_2 - GPIO38 ADC1\_CHANNEL\_3 - GPIO39 ADC1\_CHANNEL\_4 - GPIO32  
ADC1\_CHANNEL\_5 - GPIO33 ADC1\_CHANNEL\_6 - GPIO34 ADC1\_CHANNEL\_7 - GPIO35

### Enumerations

enum `periph_adc_button_event_id_t`

*Values:*

enumerator `PERIPH_ADC_BUTTON_IDLE`

enumerator `PERIPH_ADC_BUTTON_PRESSED`

enumerator `PERIPH_ADC_BUTTON_RELEASE`

enumerator `PERIPH_ADC_BUTTON_LONG_PRESSED`

enumerator `PERIPH_ADC_BUTTON_LONG_RELEASE`

## 2.8.10 LED Controller Peripheral

This peripheral is applicable to IS31F13216 chip that is a light LED controller with an audio modulation mode. It can store data of 8 Frames with internal RAM to play small animations automatically. You can also use it to control a number of LEDs connected to GPIOs. If you want to use the IS31F13216, see functions `periph_is31f13216_init()`, `periph_is31f13216_set_blink_pattern()`, `periph_is31f13216_set_duty()`, `periph_is31f13216_set_state()`.



## Application Examples

Implementation of this API is demonstrated in `checks/check_display_led` example.

## API Reference

### Header File

- `components/esp_peripherals/include/periph_is31f3216.h`

### Functions

`esp_periph_handle_t` **periph\_is31f3216\_init** (`periph_is31f3216_cfg_t` \*is31f3216\_config)

Initialize the is31f3216.

**Parameters** `is31f3216_config` –

**Returns**

- `ESP_OK` Success
- `ESP_FAIL` Fail

`esp_err_t` **periph\_is31f3216\_set\_state** (`esp_periph_handle_t` periph, `periph_is31f3216_state_t` state)

Set the state of all the channels.

**Parameters**

- **periph** – The is31f3216 handle
- **state** – The state of all channels

**Returns**

- `ESP_OK` Success
- `ESP_FAIL` Fail

`esp_err_t` **periph\_is31f3216\_set\_blink\_pattern** (`esp_periph_handle_t` periph, `uint16_t` blink\_pattern)

Set the current enable channels.

**Parameters**

- **periph** – The is31f3216 handle
- **blink\_pattern** – The bit pattern of enabled channels

**Returns**

- `ESP_OK` Success
- `ESP_FAIL` Fail

`esp_err_t` **periph\_is31f3216\_set\_duty** (`esp_periph_handle_t` periph, `uint8_t` index, `uint8_t` value)

Set the duty of the channel.

**Parameters**

- **periph** – The is31f3216 handle
- **index** – The channel number

- **value** – The value of the channel's duty to be set

### Returns

- ESP\_OK Success
- ESP\_FAIL Fail

esp\_err\_t **periph\_is31f13216\_set\_duty\_step** (*esp\_periph\_handle\_t* periph, uint8\_t step)

Set the duty step of flash.

### Parameters

- **periph** – The is31f13216 handle
- **step** – The step of flash

### Returns

- ESP\_OK Success
- ESP\_FAIL Fail

esp\_err\_t **periph\_is31f13216\_set\_interval** (*esp\_periph\_handle\_t* periph, uint16\_t interval\_ms)

Set the interval time.

### Parameters

- **periph** – The is31f13216 handle
- **interval\_ms** – Time of interval

### Returns

- ESP\_OK Success
- ESP\_FAIL Fail

esp\_err\_t **periph\_is31f13216\_set\_shift\_mode** (*esp\_periph\_handle\_t* periph, *periph\_is31\_shift\_mode\_t* mode)

Set the shift mode.

### Parameters

- **periph** – The is31f13216 handle
- **mode** – Mode of *periph\_is31\_shift\_mode\_t*

### Returns

- ESP\_OK Success
- ESP\_FAIL Fail

esp\_err\_t **periph\_is31f13216\_set\_light\_on\_num** (*esp\_periph\_handle\_t* periph, uint16\_t light\_on\_num, uint16\_t max\_light\_num)

Set the light on numbers.

### Parameters

- **periph** – The is31f13216 handle
- **light\_on\_num** – Enabled led number
- **max\_light\_num** – Maximum led number

### Returns

- ESP\_OK Success
- ESP\_FAIL Fail

esp\_err\_t **periph\_is31fl3216\_set\_act\_time** (*esp\_periph\_handle\_t* periph, uint16\_t act\_ms)

Set the action time.

#### Parameters

- **periph** – The is31fl3216 handle
- **act\_ms** – Action time, unit is millisecond, 0 is infinite

#### Returns

- ESP\_OK Success
- ESP\_FAIL Fail

## Structures

struct **periph\_is31fl3216\_cfg\_t**

The configuration of is31fl3216.

#### Public Members

uint32\_t **duty**[IS31FL3216\_CH\_NUM]

An array of the is31fl3216's duty

uint16\_t **is31fl3216\_pattern**

Current enable channel

*periph\_is31fl3216\_state\_t* **state**

The state of all the channels

## Macros

**IS31FL3216\_CH\_NUM**

**BLUE\_LED\_MAX\_NUM**

## Enumerations

enum `periph_is31fl3216_state_t`

*Values:*

enumerator `IS31FL3216_STATE_UNKNOWN`

enumerator `IS31FL3216_STATE_OFF`

enumerator `IS31FL3216_STATE_ON`

enumerator `IS31FL3216_STATE_FLASH`

enumerator `IS31FL3216_STATE_BY_AUDIO`

enumerator `IS31FL3216_STATE_SHIFT`

enum `periph_is31_shift_mode_t`

*Values:*

enumerator `PERIPH_IS31_SHIFT_MODE_UNKNOWN`

enumerator `PERIPH_IS31_SHIFT_MODE_ACC`  
accumulation mode

enumerator `PERIPH_IS31_SHIFT_MODE_SINGLE`

Name of Peripheral	ESP32-LyraT	ESP32-LyraTD-MSK
<i>Wi-Fi</i>	✓	✓
<i>SD Card</i>	✓	✓
<i>Spiffs</i>	✓	✓
<i>Console</i>	✓	✓
<i>Touch</i>	✓	
<i>Button</i>	✓	
<i>LED</i>	✓	
<i>ADC Button</i>		✓
<i>LED Controller</i>		✓

## 2.9 Abstraction Layer

### 2.9.1 Ring Buffer

Ringbuffer is designed in addition to use as a data buffer, also used to connect Audio Elements. Each Element that requests data from the Ringbuffer will block the task until the data is available. Or block the task when writing data and the Buffer is full. Of course, we can stop this block at any time.



Fig. 7: Ring Buffer used in Audio Pipeline

### Application Example

In most of ESP-ADF examples connecting of Elements with Ringbuffers is done “behind the scenes” by a function `audio_pipeline_link()`. To see this operation exposed check `player/pipeline_sdcard_mp3_control` example.

### API Reference

#### Header File

- `components/audio_pipeline/include/ringbuf.h`

#### Functions

`ringbuf_handle_t rb_create` (int block\_size, int n\_blocks)

Create ringbuffer with total size = block\_size \* n\_blocks.

#### Parameters

- **block\_size** – [in] Size of each block
- **n\_blocks** – [in] Number of blocks

**Returns** ringbuf\_handle\_t

`esp_err_t rb_destroy` (ringbuf\_handle\_t rb)

Cleanup and free all memory created by ringbuf\_handle\_t.

**Parameters** **rb** – [in] The Ringbuffer handle

#### Returns

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **rb\_abort** (*ringbuf\_handle\_t* rb)

Abort waiting until there is space for reading or writing of the ringbuffer.

**Parameters** **rb** – [in] The Ringbuffer handle

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **rb\_reset** (*ringbuf\_handle\_t* rb)

Reset ringbuffer, clear all values as initial state.

**Parameters** **rb** – [in] The Ringbuffer handle

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **rb\_reset\_is\_done\_write** (*ringbuf\_handle\_t* rb)

Reset is\_done\_write flag.

**Parameters** **rb** – [in] The Ringbuffer handle

**Returns**

- ESP\_OK
- ESP\_FAIL

int **rb\_bytes\_available** (*ringbuf\_handle\_t* rb)

Get total bytes available of Ringbuffer.

**Parameters** **rb** – [in] The Ringbuffer handle

**Returns** total bytes available

int **rb\_bytes\_filled** (*ringbuf\_handle\_t* rb)

Get the number of bytes that have filled the ringbuffer.

**Parameters** **rb** – [in] The Ringbuffer handle

**Returns** The number of bytes that have filled the ringbuffer

int **rb\_get\_size** (*ringbuf\_handle\_t* rb)

Get total size of Ringbuffer (in bytes)

**Parameters** **rb** – [in] The Ringbuffer handle

**Returns** total size of Ringbuffer

int **rb\_read** (*ringbuf\_handle\_t* rb, char \*buf, int len, TickType\_t ticks\_to\_wait)

Read from Ringbuffer to *buf* with *len* and wait *ticks\_to\_wait* ticks until enough bytes to read if the ringbuffer bytes available is less than *len*. If *buf* argument provided is NULL, then ringbuffer do pseudo reads by simply advancing pointers.

**Parameters**

- **rb** – [in] The Ringbuffer handle
- **buf** – The buffer pointer to read out data
- **len** – [in] The length request

- **ticks\_to\_wait** – [in] The ticks to wait

**Returns** Number of bytes read

int **rb\_write** (*ringbuf\_handle\_t* rb, char \*buf, int len, TickType\_t ticks\_to\_wait)

Write to Ringbuffer from buf with len and wait tick\_to\_wait ticks until enough space to write if the ringbuffer space available is less than len

**Parameters**

- **rb** – [in] The Ringbuffer handle
- **buf** – The buffer
- **len** – [in] The length
- **ticks\_to\_wait** – [in] The ticks to wait

**Returns** Number of bytes written

esp\_err\_t **rb\_done\_write** (*ringbuf\_handle\_t* rb)

Set status of writing to ringbuffer is done.

**Parameters** **rb** – [in] The Ringbuffer handle

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **rb\_unblock\_reader** (*ringbuf\_handle\_t* rb)

Unblock from rb\_read.

**Parameters** **rb** – [in] The Ringbuffer handle

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **rb\_set\_reader\_holder** (*ringbuf\_handle\_t* rb, void \*holder)

Set the owner of the 'rb\_read'.

**Parameters**

- **rb** – [in] The Ringbuffer handle
- **holder** – [in] The owner of the 'rb\_read'

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **rb\_get\_reader\_holder** (*ringbuf\_handle\_t* rb, void \*\*holder)

Get the owner of the 'rb\_read'.

**Parameters**

- **rb** – [in] The Ringbuffer handle
- **holder** – [out] The owner of the 'rb\_read'

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **rb\_set\_writer\_holder** (*ringbuf\_handle\_t* rb, void \*holder)

Set the owner of the 'rb\_write'.

### Parameters

- **rb** – [in] The Ringbuffer handle
- **holder** – [in] The owner of the 'rb\_write'

### Returns

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **rb\_get\_writer\_holder** (*ringbuf\_handle\_t* rb, void \*\*holder)

Get the owner of the 'rb\_write'.

### Parameters

- **rb** – [in] The Ringbuffer handle
- **holder** – [out] The owner of the 'rb\_write'

### Returns

- ESP\_OK
- ESP\_FAIL

## Macros

**RB\_OK**

**RB\_FAIL**

**RB\_DONE**

**RB\_ABORT**

**RB\_TIMEOUT**

## Type Definitions

```
typedef struct ringbuf *ringbuf_handle_t
```



## 2.9.2 Audio HAL

Abstraction layer for audio board hardware, serves as an interface between the user application and the hardware driver for specific audio board like *ESP32 LyraT*.

The API provides data structures to configure sampling rates of ADC and DAC signal conversion, data bit widths, I2C stream parameters, and selection of signal channels connected to ADC and DAC. It also contains several specific functions to e.g. initialize the audio board, `audio_hal_init()`, control the volume, `audio_hal_get_volume()` and `audio_hal_set_volume()`.

### API Reference

#### Header File

- `components/audio_hal/include/audio_hal.h`

#### Functions

`audio_hal_handle_t audio_hal_init (audio_hal_codec_config_t *audio_hal_conf, audio_hal_func_t *audio_hal_func)`

Initialize media codec driver.

---

**Note:** If selected codec has already been installed, it'll return the `audio_hal` handle.

---

#### Parameters

- **audio\_hal\_conf** – Configure structure `audio_hal_config_t`
- **audio\_hal\_func** – Structure containing functions used to operate audio the codec chip

**Returns** `int`, 0&#8212;success, others&#8212;fail

`esp_err_t audio_hal_deinit (audio_hal_handle_t audio_hal)`

Uninitialize media codec driver.

**Parameters** **audio\_hal** – reference function pointer for selected audio codec

**Returns** `int`, 0&#8212;success, others&#8212;fail

`esp_err_t audio_hal_ctrl_codec (audio_hal_handle_t audio_hal, audio_hal_codec_mode_t mode, audio_hal_ctrl_t audio_hal_ctrl)`

Start/stop codec driver.

#### Parameters

- **audio\_hal** – reference function pointer for selected audio codec
- **mode** – select media hal codec mode either encode/decode/or both to start from `audio_hal_codec_mode_t`
- **audio\_hal\_ctrl** – select start stop state for specific mode

**Returns** `int`, 0&#8212;success, others&#8212;fail

esp\_err\_t **audio\_hal\_codec\_iface\_config** (*audio\_hal\_handle\_t* audio\_hal, *audio\_hal\_codec\_mode\_t* mode, *audio\_hal\_codec\_i2s\_iface\_t* \*iface)

Set codec I2S interface samples rate & bit width and format either I2S or PCM/DSP.

### Parameters

- **audio\_hal** – reference function pointer for selected audio codec
- **mode** – select media hal codec mode either encode/decode/or both to start from audio\_hal\_codec\_mode\_t
- **iface** – I2S sample rate (ex: 16000, 44100), I2S bit width (16, 24, 32), I2s format (I2S, PCM, DSP).

### Returns

- 0 Success
- -1 Error

esp\_err\_t **audio\_hal\_set\_mute** (*audio\_hal\_handle\_t* audio\_hal, bool mute)

Set voice mute. Enables or disables DAC mute of a codec.

---

**Note:** `audio_hal_get_volume` will still give a non-zero number in mute state. It will be set to that number when speaker is unmuted.

---

### Parameters

- **audio\_hal** – reference function pointer for selected audio codec
- **mute** – true/false. If true speaker will be muted and if false speaker will be unmuted.

**Returns** int, 0&#8212;success, others&#8212;fail

esp\_err\_t **audio\_hal\_set\_volume** (*audio\_hal\_handle\_t* audio\_hal, int volume)

Set voice volume.

---

**Note:** if volume is 0, mute is enabled, range is 0-100.

---

### Parameters

- **audio\_hal** – reference function pointer for selected audio codec
- **volume** – value of volume in percent(%)

**Returns** int, 0&#8212;success, others&#8212;fail

esp\_err\_t **audio\_hal\_get\_volume** (*audio\_hal\_handle\_t* audio\_hal, int \*volume)

get voice volume.

---

**Note:** if volume is 0, mute is enabled, range is 0-100.

---

### Parameters

- **audio\_hal** – reference function pointer for selected audio codec

- **volume** – value of volume in percent returned(%)

**Returns** int, 0&#8212;success, others&#8212;fail

esp\_err\_t **audio\_hal\_enable\_pa** (*audio\_hal\_handle\_t* audio\_hal, bool enable)

Enables or disables PA.

#### Parameters

- **audio\_hal** – reference function pointer for selected audio codec
- **enable** – true/false.

**Returns** int, 0&#8212;success, others&#8212;fail

## Structures

struct **audio\_hal\_codec\_i2s\_iface\_t**

I2s interface configuration for audio codec chip.

#### Public Members

*audio\_hal\_iface\_mode\_t* **mode**

audio codec chip mode

*audio\_hal\_iface\_format\_t* **fmt**

I2S interface format

*audio\_hal\_iface\_samples\_t* **samples**

I2S interface samples per second

*audio\_hal\_iface\_bits\_t* **bits**

i2s interface number of bits per sample

struct **audio\_hal\_codec\_config\_t**

Configure media hal for initialization of audio codec chip.

#### Public Members

*audio\_hal\_adc\_input\_t* **adc\_input**

set adc channel

*audio\_hal\_dac\_output\_t* **dac\_output**

set dac channel

*audio\_hal\_codec\_mode\_t* **codec\_mode**

select codec mode: adc, dac or both

*audio\_hal\_codec\_i2s\_iface\_t* **i2s\_iface**

set I2S interface configuration

struct **audio\_hal**

Configuration of functions and variables used to operate audio codec chip.

### Public Members

esp\_err\_t (\***audio\_codec\_initialize**)(*audio\_hal\_codec\_config\_t* \*codec\_cfg)

initialize codec

esp\_err\_t (\***audio\_codec\_deinitialize**)(void)

deinitialize codec

esp\_err\_t (\***audio\_codec\_ctrl**)(*audio\_hal\_codec\_mode\_t* mode, *audio\_hal\_ctrl\_t* ctrl\_state)

control codec mode and state

esp\_err\_t (\***audio\_codec\_config\_iface**)(*audio\_hal\_codec\_mode\_t* mode, *audio\_hal\_codec\_i2s\_iface\_t* \*iface)

configure i2s interface

esp\_err\_t (\***audio\_codec\_set\_mute**)(bool mute)

set codec mute

esp\_err\_t (\***audio\_codec\_set\_volume**)(int volume)

set codec volume

esp\_err\_t (\***audio\_codec\_get\_volume**)(int \*volume)

get codec volume

esp\_err\_t (\***audio\_codec\_enable\_pa**)(bool enable)

enable pa

xSemaphoreHandle **audio\_hal\_lock**

semaphore of codec

void \***handle**

handle of audio codec

## Macros

**AUDIO\_HAL\_VOL\_DEFAULT**

## Type Definitions

typedef struct *audio\_hal* \***audio\_hal\_handle\_t**

typedef struct *audio\_hal* **audio\_hal\_func\_t**

Configuration of functions and variables used to operate audio codec chip.

## Enumerations

enum **audio\_hal\_codec\_mode\_t**

Select media hal codec mode.

*Values:*

enumerator **AUDIO\_HAL\_CODEC\_MODE\_ENCODE**

select adc

enumerator **AUDIO\_HAL\_CODEC\_MODE\_DECODE**

select dac

enumerator **AUDIO\_HAL\_CODEC\_MODE\_BOTH**

select both adc and dac

enumerator **AUDIO\_HAL\_CODEC\_MODE\_LINE\_IN**

set adc channel

enum **audio\_hal\_adc\_input\_t**

Select adc channel for input mic signal.

*Values:*

enumerator **AUDIO\_HAL\_ADC\_INPUT\_LINE1**

mic input to adc channel 1

enumerator **AUDIO\_HAL\_ADC\_INPUT\_LINE2**

mic input to adc channel 2

enumerator **AUDIO\_HAL\_ADC\_INPUT\_ALL**

mic input to both channels of adc

enumerator **AUDIO\_HAL\_ADC\_INPUT\_DIFFERENCE**  
mic input to adc difference channel

enum **audio\_hal\_dac\_output\_t**

Select channel for dac output.

*Values:*

enumerator **AUDIO\_HAL\_DAC\_OUTPUT\_LINE1**  
dac output signal to channel 1

enumerator **AUDIO\_HAL\_DAC\_OUTPUT\_LINE2**  
dac output signal to channel 2

enumerator **AUDIO\_HAL\_DAC\_OUTPUT\_ALL**  
dac output signal to both channels

enum **audio\_hal\_ctrl\_t**

Select operating mode i.e. start or stop for audio codec chip.

*Values:*

enumerator **AUDIO\_HAL\_CTRL\_STOP**  
set stop mode

enumerator **AUDIO\_HAL\_CTRL\_START**  
set start mode

enum **audio\_hal\_iface\_mode\_t**

Select I2S interface operating mode i.e. master or slave for audio codec chip.

*Values:*

enumerator **AUDIO\_HAL\_MODE\_SLAVE**  
set slave mode

enumerator **AUDIO\_HAL\_MODE\_MASTER**  
set master mode

enum **audio\_hal\_iface\_samples\_t**

Select I2S interface samples per second.

*Values:*

enumerator **AUDIO\_HAL\_08K\_SAMPLES**  
set to 8k samples per second

enumerator **AUDIO\_HAL\_11K\_SAMPLES**

set to 11.025k samples per second

enumerator **AUDIO\_HAL\_16K\_SAMPLES**

set to 16k samples in per second

enumerator **AUDIO\_HAL\_22K\_SAMPLES**

set to 22.050k samples per second

enumerator **AUDIO\_HAL\_24K\_SAMPLES**

set to 24k samples in per second

enumerator **AUDIO\_HAL\_32K\_SAMPLES**

set to 32k samples in per second

enumerator **AUDIO\_HAL\_44K\_SAMPLES**

set to 44.1k samples per second

enumerator **AUDIO\_HAL\_48K\_SAMPLES**

set to 48k samples per second

enum **audio\_hal\_iface\_bits\_t**

Select I2S interface number of bits per sample.

*Values:*

enumerator **AUDIO\_HAL\_BIT\_LENGTH\_16BITS**

set 16 bits per sample

enumerator **AUDIO\_HAL\_BIT\_LENGTH\_24BITS**

set 24 bits per sample

enumerator **AUDIO\_HAL\_BIT\_LENGTH\_32BITS**

set 32 bits per sample

enum **audio\_hal\_iface\_format\_t**

Select I2S interface format for audio codec chip.

*Values:*

enumerator **AUDIO\_HAL\_I2S\_NORMAL**

set normal I2S format

enumerator **AUDIO\_HAL\_I2S\_LEFT**

set all left format

enumerator **AUDIO\_HAL\_I2S\_RIGHT**  
set all right format

enumerator **AUDIO\_HAL\_I2S\_DSP**  
set dsp/pcm format

### 2.9.3 ES8388 Driver

Driver for **ES8388** codec chip used in *ESP32 LyraT* audio board.

#### API Reference

##### Header File

- components/audio\_hal/driver/es8388/es8388.h

##### Functions

esp\_err\_t **es8388\_init** (*audio\_hal\_codec\_config\_t* \*cfg)  
Initialize ES8388 codec chip.

**Parameters** **cfg** – configuration of ES8388

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **es8388\_deinit** (void)  
Deinitialize ES8388 codec chip.

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **es8388\_config\_fmt** (es\_module\_t mod, es\_i2s\_fmt\_t cfg)  
Configure ES8388 I2S format.

**Parameters**

- **mod** – set ADC or DAC or both
- **cfg** – ES8388 I2S format

**Returns**

- ESP\_OK
- ESP\_FAIL



esp\_err\_t **es8388\_i2s\_config\_clock** (es\_i2s\_clock\_t cfg)

Configure I2s clock in MSATER mode.

**Parameters** **cfg** – set bits clock and WS clock

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **es8388\_set\_bits\_per\_sample** (es\_module\_t mode, es\_bits\_length\_t bit\_per\_sample)

Configure ES8388 data sample bits.

**Parameters**

- **mode** – set ADC or DAC or both
- **bit\_per\_sample** – bit number of per sample

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **es8388\_start** (es\_module\_t mode)

Start ES8388 codec chip.

**Parameters** **mode** – set ADC or DAC or both

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **es8388\_stop** (es\_module\_t mode)

Stop ES8388 codec chip.

**Parameters** **mode** – set ADC or DAC or both

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **es8388\_set\_voice\_volume** (int volume)

Set voice volume.

**Parameters** **volume** – voice volume (0~100)

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **es8388\_get\_voice\_volume** (int \*volume)

Get voice volume.

**Parameters** **\*volume** – [out] voice volume (0~100)

**Returns**

- ESP\_OK

- ESP\_FAIL

esp\_err\_t **es8388\_set\_voice\_mute** (bool enable)

Configure ES8388 DAC mute or not. Basically you can use this function to mute the output or unmute.

**Parameters** **enable** – enable(1) or disable(0)

**Returns**

- ESP\_FAIL Parameter error
- ESP\_OK Success

esp\_err\_t **es8388\_get\_voice\_mute** (void)

Get ES8388 DAC mute status.

**Returns**

- ESP\_FAIL Parameter error
- ESP\_OK Success

esp\_err\_t **es8388\_set\_mic\_gain** (es\_mic\_gain\_t gain)

Set ES8388 mic gain.

**Parameters** **gain** – db of mic gain

**Returns**

- ESP\_FAIL Parameter error
- ESP\_OK Success

esp\_err\_t **es8388\_config\_adc\_input** (es\_adc\_input\_t input)

Set ES8388 adc input mode.

**Parameters** **input** – adc input mode

**Returns**

- ESP\_FAIL Parameter error
- ESP\_OK Success

esp\_err\_t **es8388\_config\_dac\_output** (es\_dac\_output\_t output)

Set ES8388 dac output mode.

**Parameters** **output** – dac output mode

**Returns**

- ESP\_FAIL Parameter error
- ESP\_OK Success

esp\_err\_t **es8388\_write\_reg** (uint8\_t reg\_add, uint8\_t data)

Write ES8388 register.

**Parameters**

- **reg\_add** – address of register
- **data** – data of register

**Returns**

- ESP\_FAIL Parameter error

- ESP\_OK Success

void **es8388\_read\_all** (void)

Print all ES8388 registers.

esp\_err\_t **es8388\_config\_i2s** (*audio\_hal\_codec\_mode\_t* mode, *audio\_hal\_codec\_i2s\_iface\_t* \*iface)

Configure ES8388 codec mode and I2S interface.

#### Parameters

- **mode** – codec mode
- **iface** – I2S config

#### Returns

- ESP\_FAIL Parameter error
- ESP\_OK Success

esp\_err\_t **es8388\_ctrl\_state** (*audio\_hal\_codec\_mode\_t* mode, *audio\_hal\_ctrl\_t* ctrl\_state)

Control ES8388 codec chip.

#### Parameters

- **mode** – codec mode
- **ctrl\_state** – start or stop decode or encode progress

#### Returns

- ESP\_FAIL Parameter error
- ESP\_OK Success

esp\_err\_t **es8388\_pa\_power** (bool enable)

Set ES8388 PA power.

**Parameters** **enable** – true for enable PA power, false for disable PA power

#### Returns

- ESP\_ERR\_INVALID\_ARG
- ESP\_OK

## Macros

**ES8388\_ADDR**

0x22:CE=1;0x20:CE=0

**ES8388\_CONTROL1**

**ES8388\_CONTROL2**

**ES8388\_CHIPPOWER**

**ES8388\_ADCPOWER**

ES8388\_DACPOWER

ES8388\_CHIPLOPOW1

ES8388\_CHIPLOPOW2

ES8388\_ANAVOLMANAG

ES8388\_MASTERMODE

ES8388\_ADCCONTROL1

ES8388\_ADCCONTROL2

ES8388\_ADCCONTROL3

ES8388\_ADCCONTROL4

ES8388\_ADCCONTROL5

ES8388\_ADCCONTROL6

ES8388\_ADCCONTROL7

ES8388\_ADCCONTROL8

ES8388\_ADCCONTROL9

ES8388\_ADCCONTROL10

ES8388\_ADCCONTROL11

ES8388\_ADCCONTROL12

ES8388\_ADCCONTROL13

ES8388\_ADCCONTROL14

ES8388\_DACCONTROL1

ES8388\_DACCONTROL2

ES8388\_DACCONTROL3

ES8388\_DACCONTROL4

ES8388\_DACCONTROL5

ES8388\_DACCONTROL6

ES8388\_DACCONTROL7

ES8388\_DACCONTROL8

ES8388\_DACCONTROL9

ES8388\_DACCONTROL10

ES8388\_DACCONTROL11

ES8388\_DACCONTROL12

ES8388\_DACCONTROL13

ES8388\_DACCONTROL14

ES8388\_DACCONTROL15

ES8388\_DACCONTROL16

ES8388\_DACCONTROL17

ES8388\_DACCONTROL18

ES8388\_DACCONTROL19

ES8388\_DACCONTROL20

ES8388\_DACCONTROL21

ES8388\_DACCONTROL22

ES8388\_DACCONTROL23

ES8388\_DACCONTROL24

ES8388\_DACCONTROL25

ES8388\_DACCONTROL26

ES8388\_DACCONTROL27

ES8388\_DACCONTROL28

ES8388\_DACCONTROL29

ES8388\_DACCONTROL30

### 2.9.4 ES8374 Driver

Driver for ES8374 codec chip.

#### API Reference

##### Header File

- components/audio\_hal/driver/es8374/es8374.h

##### Functions

esp\_err\_t **es8374\_codec\_init** (*audio\_hal\_codec\_config\_t* \*cfg)

Initialize ES8374 codec chip.

**Parameters** **cfg** – configuration of ES8374

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **es8374\_codec\_deinit** (void)

Deinitialize ES8374 codec chip.

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **es8374\_config\_fmt** (es\_module\_t mode, es\_i2s\_fmt\_t fmt)

Configure ES8374 I2S format.

**Parameters**

- **mode** – Set ADC or DAC or both

- **fmt** – ES8374 I2S format

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **es8374\_i2s\_config\_clock** (es\_i2s\_clock\_t cfg)

Configure I2S clock in MSATER mode.

**Parameters** **cfg** – Set bits clock and WS clock

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **es8374\_set\_bits\_per\_sample** (es\_module\_t mode, es\_bits\_length\_t bit\_per\_sample)

Configure ES8374 data sample bits.

**Parameters**

- **mode** – Set ADC or DAC or both
- **bit\_per\_sample** – Bit number of per sample

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **es8374\_start** (es\_module\_t mode)

Start ES8374 codec chip.

**Parameters** **mode** – Set ADC or DAC or both

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **es8374\_stop** (es\_module\_t mode)

Stop ES8374 codec chip.

**Parameters** **mode** – Set ADC or DAC or both

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **es8374\_codec\_set\_voice\_volume** (int volume)

Set voice volume.

**Parameters** **volume** – Voice volume (0~100)

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **es8374\_codec\_get\_voice\_volume** (int \*volume)

Get voice volume.

**Parameters** \*volume – [out] Voice volume (0~100)

**Returns**

- ESP\_OK
- ESP\_FAIL

esp\_err\_t **es8374\_set\_voice\_mute** (bool enable)

Mute or unmute ES8374 DAC. Basically you can use this function to mute or unmute the output.

**Parameters** enable – [inout] Enable Mute(1) or Unmute(0)

**Returns**

- ESP\_FAIL Parameter error
- ESP\_OK Success

esp\_err\_t **es8374\_get\_voice\_mute** (void)

Get ES8374 DAC mute status.

**Returns**

- ESP\_FAIL
- ESP\_OK

esp\_err\_t **es8374\_set\_mic\_gain** (es\_mic\_gain\_t gain)

Set ES8374 mic gain.

**Parameters** gain – [inout] gain db of mic gain

**Returns**

- ESP\_FAIL Parameter error
- ESP\_OK Success

esp\_err\_t **es8374\_config\_adc\_input** (es\_adc\_input\_t input)

Set ES8374 ADC input mode.

**Parameters** input – [in] ADC input mode

**Returns**

- ESP\_FAIL Parameter error
- ESP\_OK Success

esp\_err\_t **es8374\_config\_dac\_output** (es\_dac\_output\_t output)

Set ES8374 DAC output mode.

**Parameters** output – [in] DAC output mode

**Returns**

- ESP\_FAIL Parameter error
- ESP\_OK Success



esp\_err\_t **es8374\_write\_reg** (uint8\_t reg\_add, uint8\_t data)

Write ES8374 register.

**Parameters**

- **reg\_add** – Address of register
- **data** – Data of register

**Returns**

- ESP\_FAIL Parameter error
- ESP\_OK Success

void **es8374\_read\_all** (void)

Print all ES8374 registers.

esp\_err\_t **es8374\_codec\_config\_i2s** (*audio\_hal\_codec\_mode\_t* mode, *audio\_hal\_codec\_i2s\_iface\_t* \*iface)

Configure ES8374 codec mode and I2S interface.

**Parameters**

- **mode** – Codec mode
- **iface** – I2S config

**Returns**

- ESP\_FAIL Parameter error
- ESP\_OK Success

esp\_err\_t **es8374\_codec\_ctrl\_state** (*audio\_hal\_codec\_mode\_t* mode, *audio\_hal\_ctrl\_t* ctrl\_state)

Control ES8374 codec chip.

**Parameters**

- **mode** – Codec mode
- **ctrl\_state** – Start or stop decode or encode progress

**Returns**

- ESP\_FAIL Parameter error
- ESP\_OK Success

esp\_err\_t **es8374\_pa\_power** (bool enable)

Set ES8374 PA power.

**Parameters** **enable** – True for enable PA power, false for disable PA power

**Returns**

- ESP\_ERR\_INVALID\_ARG
- ESP\_OK

### Macros

**ES8374\_ADDR**

## 2.9.5 ZL38063 Driver

Driver for *ZL38063* codec chip used in *ESP32-LyraTD-MSC* audio board.

### API Reference

#### Header File

- `components/audio_hal/driver/zl38063/zl38063.h`

#### Functions

`esp_err_t zl38063_codec_init (audio_hal_codec_config_t *cfg)`

Initialize ZL38063 chip.

**Parameters** `cfg` – configuration of ZL38063

**Returns**

- `ESP_OK`
- `ESP_FAIL`

`esp_err_t zl38063_codec_deinit (void)`

Deinitialize ZL38063 chip.

**Returns**

- `ESP_OK`
- `ESP_FAIL`

`esp_err_t zl38063_codec_ctrl_state (audio_hal_codec_mode_t mode, audio_hal_ctrl_t ctrl_state)`

Control ZL38063 chip.

The functions `zl38063_ctrl_state` and `zl38063_config_i2s` are not used by this driver. They are kept here to maintain the uniformity and convenience of the interface of the ADF project. These settings for `zl38063` are burned in firmware and configuration files. Default `i2s` configuration: 48000Hz, 16bit, Left-Right channels. Use resampling to be compatible with different file types.

**Parameters**

- `mode` – codec mode
- `ctrl_state` – start or stop decode or encode progress

**Returns**

- `ESP_FAIL` Parameter error
- `ESP_OK` Success

`esp_err_t z138063_codec_config_i2s` (*audio\_hal\_codec\_mode\_t* mode, *audio\_hal\_codec\_i2s\_iface\_t* \*iface)

Configure ZL38063 codec mode and I2S interface.

**Parameters**

- **mode** – codec mode
- **iface** – I2S config

**Returns**

- `ESP_FAIL` Parameter error
- `ESP_OK` Success

`esp_err_t z138063_codec_set_voice_mute` (bool mute)

Mute or unmute the codec

**Parameters** **mute** – true, false

**Returns**

- `ESP_OK`
- `ESP_FAIL`

`esp_err_t z138063_codec_set_voice_volume` (int volume)

Set voice volume.

**Parameters** **volume** – voice volume (0~100)

**Returns**

- `ESP_OK`
- `ESP_FAIL`

`esp_err_t z138063_codec_get_voice_volume` (int \*volume)

Get voice volume.

**Parameters** **\*volume** – [out] voice volume (0~100)

**Returns**

- `ESP_OK`
- `ESP_FAIL`

## 2.10 Configuration Options

Compile-time configuration options specific to ESP-ADF.

## 2.10.1 ADF Library Configuration

### **MEDIA\_PROTOCOL\_LIB\_ENABLE**

Enable Media Protocol Library

*Found in: ADF Library Configuration*

### **MEDIA\_LIB\_MEM\_AUTO\_TRACE**

Support trace memory automatically after media\_lib\_sal init

*Found in: ADF Library Configuration*

### **MEDIA\_LIB\_MEM\_TRACE\_DEPTH**

Memory trace stack depth

*Found in: ADF Library Configuration*

Set memory trace depth

### **MEDIA\_LIB\_MEM\_TRACE\_NUM**

Memory trace number

*Found in: ADF Library Configuration*

Set memory trace number

### **MEDIA\_LIB\_MEM\_TRACE\_MODULE**

Trace for module memory usage

*Found in: ADF Library Configuration*

### **MEDIA\_LIB\_MEM\_TRACE\_LEAKAGE**

Trace for memory leakage

*Found in: ADF Library Configuration*

### **MEDIA\_LIB\_MEM\_TRACE\_SAVE\_HISTORY**

Trace to save memory history

*Found in: ADF Library Configuration*

## **MEDIA\_LIB\_MEM\_SAVE\_CACHE\_SIZE**

Cache buffer size to store save history

*Found in: ADF Library Configuration*

Set cache size for memory history

## **MEDIA\_LIB\_MEM\_TRACE\_SAVE\_PATH**

Memory trace save path

*Found in: ADF Library Configuration*

Set memory trace save path

## **2.10.2 DuerOS Service**

### **DUEROS\_GEN\_PROFILE**

Generate the nvs partition which include the profile of dueros

*Found in: Component config > DuerOS Service*

### **DUEROS\_FLASH\_PROFILE**

Flash the generated nvs partition

*Found in: Component config > DuerOS Service*

### **DUEROS\_DEVICE\_NAME**

Duer device name

*Found in: Component config > DuerOS Service*

## **2.10.3 ESP Speech Recognition**

### **MODEL\_DATA\_PATH**

model data path

*Found in: ESP Speech Recognition*

#### **Available options:**

- MODEL\_IN\_FLASH
- MODEL\_IN\_SDCARD

### USE\_AFE

use afe

*Found in: ESP Speech Recognition*

### AFE\_INTERFACE\_SEL

Afe interface

*Found in: ESP Speech Recognition*

Select the afe interface to be used.

**Available options:**

- AFE\_INTERFACE\_V1

### USE\_NSNET

use nsnet

*Found in: ESP Speech Recognition*

### SR\_NSN\_MODEL\_LOAD

Select deep noise suppression

*Found in: ESP Speech Recognition*

Select the deep noise suppression to be loaded.

**Available options:**

- SR\_NSN\_NONE
- SR\_NSN\_NSNET1
- SR\_NSN\_NSNET2

### USE\_WAKENET

use wakenet

*Found in: ESP Speech Recognition*

### SR\_WN\_MODEL\_LOAD

Select wake words

*Found in: ESP Speech Recognition*

Select the Wake Words to be loaded.

**Available options:**

- SR\_WN\_WN5\_HILEXIN
- SR\_WN\_WN5X3\_HILEXIN

- SR\_WN\_WN5\_NIHAOXIAOZHI
- SR\_WN\_WN5X3\_NIHAOXIAOZHI
- SR\_WN\_WN5X3\_NIHAOXIAOXIN
- SR\_WN\_WN8\_ALEXA
- SR\_WN\_WN9\_HILEXIN
- SR\_WN\_WN9\_XIAOAITONGXUE
- SR\_WN\_WN9\_ALEXA
- SR\_WN\_WN9\_HIESP
- SR\_WN\_WN9\_HIMFIVE
- SR\_WN\_WN9\_NIHAOXIAOZHI\_TTS
- SR\_WN\_WN9\_JARVIS\_TTS
- SR\_WN\_WN9\_COMPUTER\_TTS
- SR\_WN\_WN9\_HEYWILLOW\_TTS
- SR\_WN\_WN9\_SOPHIA\_TTS
- SR\_WN\_WN9\_NIHAOXIAOXIN\_TTS
- SR\_WN\_WN9\_XIAOMEITONGXUE\_TTS
- SR\_WN\_WN9\_HIXIAOXING\_TTS
- SR\_WN\_WN9\_MYCROFT\_TTS
- SR\_WN\_WN9\_HEYPRINTER\_TTS
- SR\_WN\_WN9\_XIAOLONGXIAOLONG\_TTS
- SR\_WN\_WN9\_MIAOMIAOTONGXUE\_TTS
- SR\_WN\_WN9\_HIJOY\_TTS
- SR\_WN\_WN9\_HILILI\_TTS
- SR\_WN\_WN9\_HITELLY\_TTS
- SR\_WN\_WN9\_HEYWANDA\_TTS
- SR\_WN\_WN9\_HIMIAOMIAO\_TTS
- SR\_WN\_WN9\_XIAOBINXIAOBIN\_TTS
- SR\_WN\_WN9\_ASTROLABE\_TTS
- SR\_WN\_WN9\_CUSTOMWORD
- SR\_WN\_LOAD\_MULIT\_WORD

## Load Multiple Wake Words

### SR\_WN\_WN9\_HILEXIN\_MULTI

Hi,你好 (wn9\_hilexin)

*Found in: ESP Speech Recognition > Load Multiple Wake Words*

### SR\_WN\_WN9\_XIAOAITONGXUE\_MULTI

你好你好 (wn9\_xiaoitongxue)

*Found in: ESP Speech Recognition > Load Multiple Wake Words*

### SR\_WN\_WN9\_NIHAOXIAOZHI\_TTS\_MULTI

你好你好 (wn9\_nihaoxiaozhi\_tts)

*Found in: ESP Speech Recognition > Load Multiple Wake Words*

### SR\_WN\_WN9\_ALEXA\_MULTI

Alexa (wn9\_alexa)

*Found in: ESP Speech Recognition > Load Multiple Wake Words*

### SR\_WN\_WN9\_HIESP\_MULTI

Hi,ESP (wn9\_hiesp)

*Found in: ESP Speech Recognition > Load Multiple Wake Words*

### SR\_WN\_WN9\_JARVIS\_TTS\_MULTI

Jarvis (wn9\_jarvis\_tts)

*Found in: ESP Speech Recognition > Load Multiple Wake Words*

### SR\_WN\_WN9\_COMPUTER\_TTS\_MULTI

computer (wn9\_computer\_tts)

*Found in: ESP Speech Recognition > Load Multiple Wake Words*



**SR\_WN\_WN9\_HEYWILLOW\_TTS\_MULTI**

Hey,Willow (wn9\_heywillow\_tts)

*Found in: ESP Speech Recognition > Load Multiple Wake Words*

**SR\_WN\_WN9\_SOPHIA\_TTS\_MULTI**

Sophia (wn9\_sophia\_tts)

*Found in: ESP Speech Recognition > Load Multiple Wake Words*

**SR\_WN\_WN9\_NIHAOXIAOXIN\_TTS\_MULTI**

你好你好 (wn9\_nihaoxiaoxin\_tts)

*Found in: ESP Speech Recognition > Load Multiple Wake Words*

**SR\_WN\_WN9\_XIAOMEITONGXUE\_TTS\_MULTI**

好好学习 (wn9\_xiaomeitongxue\_tts)

*Found in: ESP Speech Recognition > Load Multiple Wake Words*

**SR\_WN\_WN9\_HEYPRINTER\_TTS\_MULTI**

Hey,Printer (wn9\_heyprinter\_tts)

*Found in: ESP Speech Recognition > Load Multiple Wake Words*

**SR\_WN\_WN9\_XIAOLONGXIAOLONG\_TTS\_MULTI**

龙龙龙龙 (wn9\_xiaolongxiaolong\_tts)

*Found in: ESP Speech Recognition > Load Multiple Wake Words*

**SR\_WN\_WN9\_MIAOMIAOTONGXUE\_TTS\_MULTI**

好好好好 (wn9\_miaomiaotongxue\_tts)

*Found in: ESP Speech Recognition > Load Multiple Wake Words*

**SR\_WN\_WN9\_HEYWANDA\_TTS\_MULTI**

Hey,Wanda (wn9\_heywanda\_tts)

*Found in: ESP Speech Recognition > Load Multiple Wake Words*

### SR\_WN\_WN9\_HIMIAOMIAO\_TTS\_MULTI

Hi,?? (wn9\_himiaomiao\_tts)

*Found in: ESP Speech Recognition > Load Multiple Wake Words*

### SR\_WN\_WN9\_MYCROFT\_TTS\_MULTI

Mycroft (wn9\_mycroft\_tts)

*Found in: ESP Speech Recognition > Load Multiple Wake Words*

### SR\_WN\_WN9\_HIJOY\_TTS\_MULTI

Hi,Joy (wn9\_hijoy\_tts)

*Found in: ESP Speech Recognition > Load Multiple Wake Words*

### SR\_WN\_WN9\_HILILI\_TTS\_MULTI

Hi,Lily/Hi,?? (wn9\_hilili\_tts)

*Found in: ESP Speech Recognition > Load Multiple Wake Words*

### SR\_WN\_WN9\_HITELLY\_TTS\_MULTI

Hi,Telly/Hi,?? (wn9\_hitelly\_tts)

*Found in: ESP Speech Recognition > Load Multiple Wake Words*

### SR\_WN\_WN9\_XIAOBINXIAOBIN\_TTS\_MULTI

????/???? (wn9\_xiaobinxiaobin\_tts)

*Found in: ESP Speech Recognition > Load Multiple Wake Words*

### SR\_WN\_WN9\_ASTROLABE\_TTS\_MULTI

Astrolabe (wn9\_astrolabe\_tts)

*Found in: ESP Speech Recognition > Load Multiple Wake Words*

### USE\_MULTINET

use multinet

*Found in: ESP Speech Recognition*

## CHINESE\_SR\_MN\_MODEL\_SEL

Chinese Speech Commands Model

*Found in: ESP Speech Recognition*

Select the Wake Word Engine to be used.

**Available options:**

- SR\_MN\_CN\_NONE
- SR\_MN\_CN\_MULTINET2\_SINGLE\_RECOGNITION
- SR\_MN\_CN\_MULTINET5\_RECOGNITION\_QUANT8
- SR\_MN\_CN\_MULTINET6\_QUANT
- SR\_MN\_CN\_MULTINET6\_AC\_QUANT
- SR\_MN\_CN\_MULTINET7\_QUANT
- SR\_MN\_CN\_MULTINET7\_AC\_QUANT

## ENGLISH\_SR\_MN\_MODEL\_SEL

English Speech Commands Model

*Found in: ESP Speech Recognition*

Select the Wake Word Engine to be used.

**Available options:**

- SR\_MN\_EN\_NONE
- SR\_MN\_EN\_MULTINET5\_SINGLE\_RECOGNITION\_QUANT8
- SR\_MN\_EN\_MULTINET6\_QUANT
- SR\_MN\_EN\_MULTINET7\_QUANT

## Add Chinese speech commands

### CN\_SPEECH\_COMMAND\_ID0

ID0

*Found in: ESP Speech Recognition > Add Chinese speech commands*

### CN\_SPEECH\_COMMAND\_ID1

ID1

*Found in: ESP Speech Recognition > Add Chinese speech commands*

### **CN\_SPEECH\_COMMAND\_ID2**

ID2

*Found in: ESP Speech Recognition > Add Chinese speech commands*

### **CN\_SPEECH\_COMMAND\_ID3**

ID3

*Found in: ESP Speech Recognition > Add Chinese speech commands*

### **CN\_SPEECH\_COMMAND\_ID4**

ID4

*Found in: ESP Speech Recognition > Add Chinese speech commands*

### **CN\_SPEECH\_COMMAND\_ID5**

ID5

*Found in: ESP Speech Recognition > Add Chinese speech commands*

### **CN\_SPEECH\_COMMAND\_ID6**

ID6

*Found in: ESP Speech Recognition > Add Chinese speech commands*

### **CN\_SPEECH\_COMMAND\_ID7**

ID7

*Found in: ESP Speech Recognition > Add Chinese speech commands*

### **CN\_SPEECH\_COMMAND\_ID8**

ID8

*Found in: ESP Speech Recognition > Add Chinese speech commands*

### **CN\_SPEECH\_COMMAND\_ID9**

ID9

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID10**

ID10

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID11**

ID11

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID12**

ID12

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID13**

ID13

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID14**

ID14

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID15**

ID15

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID16**

ID16

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID17**

ID17

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID18**

ID18

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID19**

ID19

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID20**

ID20

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID21**

ID21

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID22**

ID22

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID23**

ID23

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID24**

ID24

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID25**

ID25

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID26**

ID26

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID27**

ID27

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID28**

ID28

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID29**

ID29

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID30**

ID30

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID31**

ID31

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID32**

ID32

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID33**

ID33

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID34**

ID34

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID35**

ID35

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID36**

ID36

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID37**

ID37

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID38**

ID38

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID39**

ID39

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID40**

ID40

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID41**

ID41

*Found in: ESP Speech Recognition > Add Chinese speech commands*



#### **CN\_SPEECH\_COMMAND\_ID42**

ID42

*Found in: ESP Speech Recognition > Add Chinese speech commands*

#### **CN\_SPEECH\_COMMAND\_ID43**

ID43

*Found in: ESP Speech Recognition > Add Chinese speech commands*

#### **CN\_SPEECH\_COMMAND\_ID44**

ID44

*Found in: ESP Speech Recognition > Add Chinese speech commands*

#### **CN\_SPEECH\_COMMAND\_ID45**

ID45

*Found in: ESP Speech Recognition > Add Chinese speech commands*

#### **CN\_SPEECH\_COMMAND\_ID46**

ID46

*Found in: ESP Speech Recognition > Add Chinese speech commands*

#### **CN\_SPEECH\_COMMAND\_ID47**

ID47

*Found in: ESP Speech Recognition > Add Chinese speech commands*

#### **CN\_SPEECH\_COMMAND\_ID48**

ID48

*Found in: ESP Speech Recognition > Add Chinese speech commands*

#### **CN\_SPEECH\_COMMAND\_ID49**

ID49

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID50**

ID50

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID51**

ID51

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID52**

ID52

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID53**

ID53

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID54**

ID54

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID55**

ID55

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID56**

ID56

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID57**

ID57

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID58**

ID58

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID59**

ID59

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID60**

ID60

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID61**

ID61

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID62**

ID62

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID63**

ID63

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID64**

ID64

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID65**

ID65

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID66**

ID66

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID67**

ID67

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID68**

ID68

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID69**

ID69

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID70**

ID70

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID71**

ID71

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID72**

ID72

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID73**

ID73

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID74**

ID74

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID75**

ID75

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID76**

ID76

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID77**

ID77

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID78**

ID78

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID79**

ID79

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID80**

ID80

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID81**

ID81

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID82**

ID82

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID83**

ID83

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID84**

ID84

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID85**

ID85

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID86**

ID86

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID87**

ID87

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID88**

ID88

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID89**

ID89

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID90**

ID90

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID91**

ID91

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID92**

ID92

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID93**

ID93

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID94**

ID94

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID95**

ID95

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID96**

ID96

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID97**

ID97

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID98**

ID98

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID99**

ID99

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID100**

ID100

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID101**

ID101

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID102**

ID102

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID103**

ID103

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID104**

ID104

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID105**

ID105

*Found in: ESP Speech Recognition > Add Chinese speech commands*



**CN\_SPEECH\_COMMAND\_ID106**

ID106

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID107**

ID107

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID108**

ID108

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID109**

ID109

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID110**

ID110

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID111**

ID111

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID112**

ID112

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID113**

ID113

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID114**

ID114

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID115**

ID115

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID116**

ID116

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID117**

ID117

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID118**

ID118

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID119**

ID119

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID120**

ID120

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID121**

ID121

*Found in: ESP Speech Recognition > Add Chinese speech commands*

### **CN\_SPEECH\_COMMAND\_ID122**

ID122

*Found in: ESP Speech Recognition > Add Chinese speech commands*

### **CN\_SPEECH\_COMMAND\_ID123**

ID123

*Found in: ESP Speech Recognition > Add Chinese speech commands*

### **CN\_SPEECH\_COMMAND\_ID124**

ID124

*Found in: ESP Speech Recognition > Add Chinese speech commands*

### **CN\_SPEECH\_COMMAND\_ID125**

ID125

*Found in: ESP Speech Recognition > Add Chinese speech commands*

### **CN\_SPEECH\_COMMAND\_ID126**

ID126

*Found in: ESP Speech Recognition > Add Chinese speech commands*

### **CN\_SPEECH\_COMMAND\_ID127**

ID127

*Found in: ESP Speech Recognition > Add Chinese speech commands*

### **CN\_SPEECH\_COMMAND\_ID128**

ID128

*Found in: ESP Speech Recognition > Add Chinese speech commands*

### **CN\_SPEECH\_COMMAND\_ID129**

ID129

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID130**

ID130

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID131**

ID131

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID132**

ID132

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID133**

ID133

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID134**

ID134

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID135**

ID135

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID136**

ID136

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID137**

ID137

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID138**

ID138

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID139**

ID139

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID140**

ID140

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID141**

ID141

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID142**

ID142

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID143**

ID143

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID144**

ID144

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID145**

ID145

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID146**

ID146

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID147**

ID147

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID148**

ID148

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID149**

ID149

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID150**

ID150

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID151**

ID151

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID152**

ID152

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID153**

ID153

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID154**

ID154

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID155**

ID155

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID156**

ID156

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID157**

ID157

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID158**

ID158

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID159**

ID159

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID160**

ID160

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID161**

ID161

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID162**

ID162

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID163**

ID163

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID164**

ID164

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID165**

ID165

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID166**

ID166

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID167**

ID167

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID168**

ID168

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID169**

ID169

*Found in: ESP Speech Recognition > Add Chinese speech commands*



**CN\_SPEECH\_COMMAND\_ID170**

ID170

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID171**

ID171

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID172**

ID172

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID173**

ID173

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID174**

ID174

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID175**

ID175

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID176**

ID176

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID177**

ID177

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID178**

ID178

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID179**

ID179

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID180**

ID180

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID181**

ID181

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID182**

ID182

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID183**

ID183

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID184**

ID184

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID185**

ID185

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID186**

ID186

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID187**

ID187

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID188**

ID188

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID189**

ID189

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID190**

ID190

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID191**

ID191

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID192**

ID192

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID193**

ID193

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID194**

ID194

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID195**

ID195

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID196**

ID196

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID197**

ID197

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID198**

ID198

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**CN\_SPEECH\_COMMAND\_ID199**

ID199

*Found in: ESP Speech Recognition > Add Chinese speech commands*

**Add English speech commands**

**EN\_SPEECH\_COMMAND\_ID0**

ID0

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID1**

ID1

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID2**

ID2

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID3**

ID3

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID4**

ID4

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID5**

ID5

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID6**

ID6

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID7**

ID7

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID8**

ID8

*Found in: ESP Speech Recognition > Add English speech commands*

## EN\_SPEECH\_COMMAND\_ID9

ID9

*Found in: ESP Speech Recognition > Add English speech commands*

## EN\_SPEECH\_COMMAND\_ID10

ID10

*Found in: ESP Speech Recognition > Add English speech commands*

## EN\_SPEECH\_COMMAND\_ID11

ID11

*Found in: ESP Speech Recognition > Add English speech commands*

## EN\_SPEECH\_COMMAND\_ID12

ID12

*Found in: ESP Speech Recognition > Add English speech commands*

## EN\_SPEECH\_COMMAND\_ID13

ID13

*Found in: ESP Speech Recognition > Add English speech commands*

## EN\_SPEECH\_COMMAND\_ID14

ID14

*Found in: ESP Speech Recognition > Add English speech commands*

## EN\_SPEECH\_COMMAND\_ID15

ID15

*Found in: ESP Speech Recognition > Add English speech commands*

## EN\_SPEECH\_COMMAND\_ID16

ID16

*Found in: ESP Speech Recognition > Add English speech commands*

### **EN\_SPEECH\_COMMAND\_ID17**

ID17

*Found in: ESP Speech Recognition > Add English speech commands*

### **EN\_SPEECH\_COMMAND\_ID18**

ID18

*Found in: ESP Speech Recognition > Add English speech commands*

### **EN\_SPEECH\_COMMAND\_ID19**

ID19

*Found in: ESP Speech Recognition > Add English speech commands*

### **EN\_SPEECH\_COMMAND\_ID20**

ID20

*Found in: ESP Speech Recognition > Add English speech commands*

### **EN\_SPEECH\_COMMAND\_ID21**

ID21

*Found in: ESP Speech Recognition > Add English speech commands*

### **EN\_SPEECH\_COMMAND\_ID22**

ID22

*Found in: ESP Speech Recognition > Add English speech commands*

### **EN\_SPEECH\_COMMAND\_ID23**

ID23

*Found in: ESP Speech Recognition > Add English speech commands*

### **EN\_SPEECH\_COMMAND\_ID24**

ID24

*Found in: ESP Speech Recognition > Add English speech commands*

### **EN\_SPEECH\_COMMAND\_ID25**

ID25

*Found in: ESP Speech Recognition > Add English speech commands*

### **EN\_SPEECH\_COMMAND\_ID26**

ID26

*Found in: ESP Speech Recognition > Add English speech commands*

### **EN\_SPEECH\_COMMAND\_ID27**

ID27

*Found in: ESP Speech Recognition > Add English speech commands*

### **EN\_SPEECH\_COMMAND\_ID28**

ID28

*Found in: ESP Speech Recognition > Add English speech commands*

### **EN\_SPEECH\_COMMAND\_ID29**

ID29

*Found in: ESP Speech Recognition > Add English speech commands*

### **EN\_SPEECH\_COMMAND\_ID30**

ID30

*Found in: ESP Speech Recognition > Add English speech commands*

### **EN\_SPEECH\_COMMAND\_ID31**

ID31

*Found in: ESP Speech Recognition > Add English speech commands*

### **EN\_SPEECH\_COMMAND\_ID32**

ID32

*Found in: ESP Speech Recognition > Add English speech commands*



### **EN\_SPEECH\_COMMAND\_ID33**

ID33

*Found in: ESP Speech Recognition > Add English speech commands*

### **EN\_SPEECH\_COMMAND\_ID34**

ID34

*Found in: ESP Speech Recognition > Add English speech commands*

### **EN\_SPEECH\_COMMAND\_ID35**

ID35

*Found in: ESP Speech Recognition > Add English speech commands*

### **EN\_SPEECH\_COMMAND\_ID36**

ID36

*Found in: ESP Speech Recognition > Add English speech commands*

### **EN\_SPEECH\_COMMAND\_ID37**

ID37

*Found in: ESP Speech Recognition > Add English speech commands*

### **EN\_SPEECH\_COMMAND\_ID38**

ID38

*Found in: ESP Speech Recognition > Add English speech commands*

### **EN\_SPEECH\_COMMAND\_ID39**

ID39

*Found in: ESP Speech Recognition > Add English speech commands*

### **EN\_SPEECH\_COMMAND\_ID40**

ID40

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID41**

ID41

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID42**

ID42

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID43**

ID43

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID44**

ID44

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID45**

ID45

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID46**

ID46

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID47**

ID47

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID48**

ID48

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID49**

ID49

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID50**

ID50

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID51**

ID51

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID52**

ID52

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID53**

ID53

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID54**

ID54

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID55**

ID55

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID56**

ID56

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID57**

ID57

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID58**

ID58

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID59**

ID59

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID60**

ID60

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID61**

ID61

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID62**

ID62

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID63**

ID63

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID64**

ID64

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID65**

ID65

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID66**

ID66

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID67**

ID67

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID68**

ID68

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID69**

ID69

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID70**

ID70

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID71**

ID71

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID72**

ID72

*Found in: ESP Speech Recognition > Add English speech commands*

### EN\_SPEECH\_COMMAND\_ID73

ID73

*Found in: ESP Speech Recognition > Add English speech commands*

### EN\_SPEECH\_COMMAND\_ID74

ID74

*Found in: ESP Speech Recognition > Add English speech commands*

### EN\_SPEECH\_COMMAND\_ID75

ID75

*Found in: ESP Speech Recognition > Add English speech commands*

### EN\_SPEECH\_COMMAND\_ID76

ID76

*Found in: ESP Speech Recognition > Add English speech commands*

### EN\_SPEECH\_COMMAND\_ID77

ID77

*Found in: ESP Speech Recognition > Add English speech commands*

### EN\_SPEECH\_COMMAND\_ID78

ID78

*Found in: ESP Speech Recognition > Add English speech commands*

### EN\_SPEECH\_COMMAND\_ID79

ID79

*Found in: ESP Speech Recognition > Add English speech commands*

### EN\_SPEECH\_COMMAND\_ID80

ID80

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID81**

ID81

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID82**

ID82

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID83**

ID83

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID84**

ID84

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID85**

ID85

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID86**

ID86

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID87**

ID87

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID88**

ID88

*Found in: ESP Speech Recognition > Add English speech commands*

### EN\_SPEECH\_COMMAND\_ID89

ID89

*Found in: ESP Speech Recognition > Add English speech commands*

### EN\_SPEECH\_COMMAND\_ID90

ID90

*Found in: ESP Speech Recognition > Add English speech commands*

### EN\_SPEECH\_COMMAND\_ID91

ID91

*Found in: ESP Speech Recognition > Add English speech commands*

### EN\_SPEECH\_COMMAND\_ID92

ID92

*Found in: ESP Speech Recognition > Add English speech commands*

### EN\_SPEECH\_COMMAND\_ID93

ID93

*Found in: ESP Speech Recognition > Add English speech commands*

### EN\_SPEECH\_COMMAND\_ID94

ID94

*Found in: ESP Speech Recognition > Add English speech commands*

### EN\_SPEECH\_COMMAND\_ID95

ID95

*Found in: ESP Speech Recognition > Add English speech commands*

### EN\_SPEECH\_COMMAND\_ID96

ID96

*Found in: ESP Speech Recognition > Add English speech commands*



### **EN\_SPEECH\_COMMAND\_ID97**

ID97

*Found in: ESP Speech Recognition > Add English speech commands*

### **EN\_SPEECH\_COMMAND\_ID98**

ID98

*Found in: ESP Speech Recognition > Add English speech commands*

### **EN\_SPEECH\_COMMAND\_ID99**

ID99

*Found in: ESP Speech Recognition > Add English speech commands*

### **EN\_SPEECH\_COMMAND\_ID100**

ID100

*Found in: ESP Speech Recognition > Add English speech commands*

### **EN\_SPEECH\_COMMAND\_ID101**

ID101

*Found in: ESP Speech Recognition > Add English speech commands*

### **EN\_SPEECH\_COMMAND\_ID102**

ID102

*Found in: ESP Speech Recognition > Add English speech commands*

### **EN\_SPEECH\_COMMAND\_ID103**

ID103

*Found in: ESP Speech Recognition > Add English speech commands*

### **EN\_SPEECH\_COMMAND\_ID104**

ID104

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID105**

ID105

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID106**

ID106

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID107**

ID107

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID108**

ID108

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID109**

ID109

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID110**

ID110

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID111**

ID111

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID112**

ID112

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID113**

ID113

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID114**

ID114

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID115**

ID115

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID116**

ID116

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID117**

ID117

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID118**

ID118

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID119**

ID119

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID120**

ID120

*Found in: ESP Speech Recognition > Add English speech commands*

### **EN\_SPEECH\_COMMAND\_ID121**

ID121

*Found in: ESP Speech Recognition > Add English speech commands*

### **EN\_SPEECH\_COMMAND\_ID122**

ID122

*Found in: ESP Speech Recognition > Add English speech commands*

### **EN\_SPEECH\_COMMAND\_ID123**

ID123

*Found in: ESP Speech Recognition > Add English speech commands*

### **EN\_SPEECH\_COMMAND\_ID124**

ID124

*Found in: ESP Speech Recognition > Add English speech commands*

### **EN\_SPEECH\_COMMAND\_ID125**

ID125

*Found in: ESP Speech Recognition > Add English speech commands*

### **EN\_SPEECH\_COMMAND\_ID126**

ID126

*Found in: ESP Speech Recognition > Add English speech commands*

### **EN\_SPEECH\_COMMAND\_ID127**

ID127

*Found in: ESP Speech Recognition > Add English speech commands*

### **EN\_SPEECH\_COMMAND\_ID128**

ID128

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID129**

ID129

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID130**

ID130

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID131**

ID131

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID132**

ID132

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID133**

ID133

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID134**

ID134

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID135**

ID135

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID136**

ID136

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID137**

ID137

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID138**

ID138

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID139**

ID139

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID140**

ID140

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID141**

ID141

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID142**

ID142

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID143**

ID143

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID144**

ID144

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID145**

ID145

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID146**

ID146

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID147**

ID147

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID148**

ID148

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID149**

ID149

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID150**

ID150

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID151**

ID151

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID152**

ID152

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID153**

ID153

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID154**

ID154

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID155**

ID155

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID156**

ID156

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID157**

ID157

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID158**

ID158

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID159**

ID159

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID160**

ID160

*Found in: ESP Speech Recognition > Add English speech commands*



**EN\_SPEECH\_COMMAND\_ID161**

ID161

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID162**

ID162

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID163**

ID163

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID164**

ID164

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID165**

ID165

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID166**

ID166

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID167**

ID167

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID168**

ID168

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID169**

ID169

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID170**

ID170

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID171**

ID171

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID172**

ID172

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID173**

ID173

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID174**

ID174

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID175**

ID175

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID176**

ID176

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID177**

ID177

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID178**

ID178

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID179**

ID179

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID180**

ID180

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID181**

ID181

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID182**

ID182

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID183**

ID183

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID184**

ID184

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID185**

ID185

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID186**

ID186

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID187**

ID187

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID188**

ID188

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID189**

ID189

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID190**

ID190

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID191**

ID191

*Found in: ESP Speech Recognition > Add English speech commands*

**EN\_SPEECH\_COMMAND\_ID192**

ID192

*Found in: ESP Speech Recognition > Add English speech commands*

### **EN\_SPEECH\_COMMAND\_ID193**

ID193

*Found in: ESP Speech Recognition > Add English speech commands*

### **EN\_SPEECH\_COMMAND\_ID194**

ID194

*Found in: ESP Speech Recognition > Add English speech commands*

### **EN\_SPEECH\_COMMAND\_ID195**

ID195

*Found in: ESP Speech Recognition > Add English speech commands*

### **EN\_SPEECH\_COMMAND\_ID196**

ID196

*Found in: ESP Speech Recognition > Add English speech commands*

### **EN\_SPEECH\_COMMAND\_ID197**

ID197

*Found in: ESP Speech Recognition > Add English speech commands*

### **EN\_SPEECH\_COMMAND\_ID198**

ID198

*Found in: ESP Speech Recognition > Add English speech commands*

### **EN\_SPEECH\_COMMAND\_ID199**

ID199

*Found in: ESP Speech Recognition > Add English speech commands*

## **2.10.4 Audio Codec Device Configuration**

### **CODEC\_I2C\_BACKWARD\_COMPATIBLE**

Enable backward compatibility for the I2C driver (force use of the old i2c\_driver above v5.3)

*Found in: Component config > Audio Codec Device Configuration*

Enable this option for backward compatibility with the old I2C driver

### **CODEC\_ES8311\_SUPPORT**

Support ES8311 Codec Chip

*Found in: Component config > Audio Codec Device Configuration*

Enable this option to support codec ES8311.

### **CODEC\_ES7210\_SUPPORT**

Support ES7210 Codec Chip

*Found in: Component config > Audio Codec Device Configuration*

Enable this option to support codec ES7210.

### **CODEC\_ES7243\_SUPPORT**

Support ES7243 Codec Chip

*Found in: Component config > Audio Codec Device Configuration*

Enable this option to support codec ES7243.

### **CODEC\_ES7243E\_SUPPORT**

Support ES7243E Codec Chip

*Found in: Component config > Audio Codec Device Configuration*

Enable this option to support codec ES7243E.

### **CODEC\_ES8156\_SUPPORT**

Support ES8156 Codec Chip

*Found in: Component config > Audio Codec Device Configuration*

Enable this option to support codec ES8156.

### **CODEC\_AW88298\_SUPPORT**

Support AW88298 Codec Chip

*Found in: Component config > Audio Codec Device Configuration*

Enable this option to support codec AW88298.

### **CODEC\_ES8374\_SUPPORT**

Support ES8374 Codec Chip

*Found in: Component config > Audio Codec Device Configuration*

Enable this option to support codec ES8374.

### **CODEC\_ES8388\_SUPPORT**

Support ES8388 Codec Chip

*Found in: Component config > Audio Codec Device Configuration*

Enable this option to support codec ES8388.

### **CODEC\_TAS5805M\_SUPPORT**

Support TAS5805M Codec Chip

*Found in: Component config > Audio Codec Device Configuration*

Enable this option to support codec TAS5805M.

### **CODEC\_ZL38063\_SUPPORT**

Support ZL38063 Codec Chip

*Found in: Component config > Audio Codec Device Configuration*

Enable this option to support codec ZL38063. ZL38063 firmware only support xtensa, don't enable for RISC-V IC.

## **2.10.5 ADF Features**

### **ESP\_DISPATCHER\_DELEGATE\_TASK\_CORE**

Delegation task core

*Found in: ADF Features*

Pinned delegate task to core 0 or core 1.

### **ESP\_DISPATCHER\_DELEGATE\_TASK\_PRIO**

Delegate task's prio

*Found in: ADF Features*

The delegate task's prio.

### ESP\_DISPATCHER\_DELEGATE\_STACK\_SIZE

Delegate task's stack size

*Found in: ADF Features*

The delegate task's stack is located in DRAM, modify this size to make sure all the needed operation can be run success in the it.

## 2.10.6 Audio HAL

### AUDIO\_BOARD

Audio board

*Found in: Audio HAL*

Select an audio board to use with the ESP-ADF

#### Available options:

- AUDIO\_BOARD\_CUSTOM
- ESP\_LYRAT\_V4\_3\_BOARD
- ESP\_LYRAT\_V4\_2\_BOARD
- ESP\_LYRATD\_MSC\_V2\_1\_BOARD
- ESP\_LYRATD\_MSC\_V2\_2\_BOARD
- ESP\_LYRAT\_MINI\_V1\_1\_BOARD
- ESP32\_KORVO\_DU1906\_BOARD
- ESP32\_S2\_KALUGA\_1\_V1\_2\_BOARD
- ESP32\_S3\_KORVO2\_V3\_BOARD
- ESP32\_S3\_KORVO2L\_V1\_BOARD
- ESP32\_S3\_BOX\_LITE\_BOARD
- ESP32\_S3\_BOX\_BOARD
- ESP32\_S3\_BOX\_3\_BOARD
- ESP32\_C3\_LYRA\_V2\_BOARD
- ESP32\_C6\_DEVKIT\_BOARD
- ESP32\_P4\_FUNCTION\_EV\_BOARD



## ESP32\_KORVO\_DU1906\_DAC

ESP32 KORVO DU1906 Board DAC chip

*Found in: Audio HAL*

Select DAC chip to use on ESP32\_KORVO\_DU1906 board

**Available options:**

- ESP32\_KORVO\_DU1906\_DAC\_TAS5805M
- ESP32\_KORVO\_DU1906\_DAC\_ES7148

## ESP32\_KORVO\_DU1906\_ADC

ESP32 KORVO DU1906 Board ADC chip

*Found in: Audio HAL*

Select ADC chip to use on ESP32\_KORVO\_DU1906 board

**Available options:**

- ESP32\_KORVO\_DU1906\_ADC\_ES7243



## DESIGN GUIDE

The ESP32 is a powerful chip well positioned as a MCU of the audio projects. This section is intended to provide guidance on process of designing an audio project with the ESP32 inside.

### 3.1 Project Design

When designing a project with ability to process audio signals or audio data we typically consider a subset of the following components:

#### Input:

- **Analog signal input** to connect e.g. a microphone
- **Storage media**, e.g. microSD card with audio files to read
- **Wi-Fi interface** to obtain an audio data stream from the internet
- **Bluetooth interface** to obtain an audio data stream from e.g. a Bluetooth headset
- **I2S interface** to obtain audio data stream from a codec chip
- **Ethernet interface** to obtain an audio data stream from the internet
- An internal **chip's flash memory** with some audio samples to play
- **User Interface** e.g. buttons or some other means to provide user input

#### Output:

- **Analog signal output** to connect headphones or speakers
- **Storage media**, e.g. microSD card to write some audio files, e.g. with recording
- **Wi-Fi interface** to send out an audio data stream to the internet
- **Bluetooth interface** to stream audio data to e.g. a Bluetooth headset
- **I2S interface** to stream some data to a codec chip
- **Ethernet interface** to stream an audio data stream to the internet
- An internal **chip's flash memory** to store some audio recording
- **User Interface** e.g. a display, LEDs or some means of **haptic** feedback

#### Main Processing Unit:

A microcontroller or a computer with processing power to read the data from the input, process (e.g. encode / decode) and send to the output.

### 3.1.1 Project Options

The ESP chips (including ESP32, ESP32-S2, ESP32-S3) have all the above features or are able to support them (e.g. can drive Ethernet PHY). Considering the ESP chip cost is low, and availability of ESP-ADF software development platform, we are able to develop an audio project with minimum additional components at very low price.

Depending on the application, required functionality and performance, we may consider two project groups.

- **Minimum** - having minimum additional components, assuming using on board I2S, or PDM interface as well as DAC, if no high quality audio on the output is required.
- **Typical** - with an external codec chip and a power amplifier, for high quality output audio and multiple input / output options.

There may be several variation between the above projects, by adding or removing features/components. Below are a couple of examples.

### 3.1.2 Project Minimum

With several peripherals on an ESP chip, I2S or PDM or DAC interfaces can be used to implement a minimum project. With the digital microphones, we could input voice signals and build a command voice control project minimum that could communicate with a cloud service.

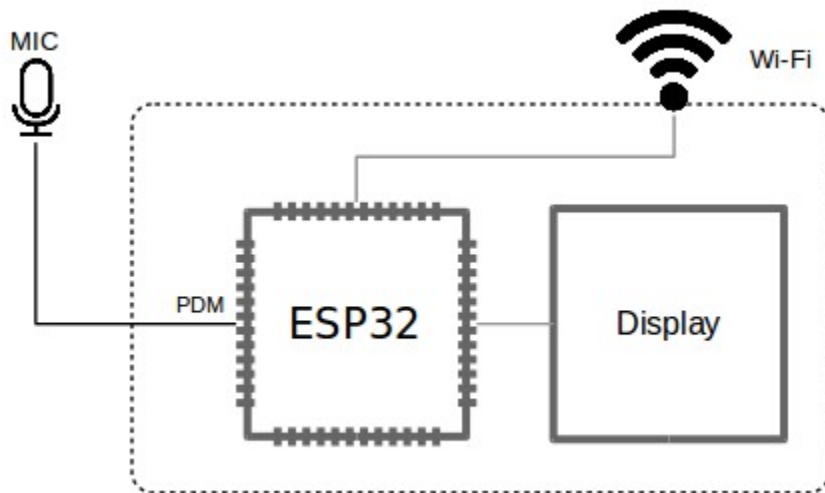


Fig. 1: Audio Project Example - Send Voice Commands to Cloud Service

With two on board DACs, if 8-bit width on the output is satisfactory, we may implement another project minimum - a device to play an internet connected radio.

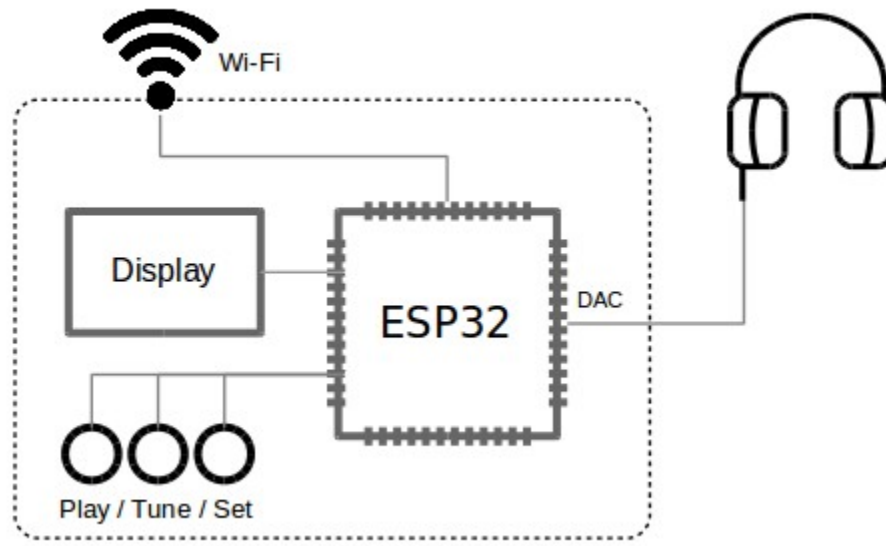


Fig. 2: Audio Project Example - Internet Connected Radio Player

### 3.1.3 Typical Project

When looking for better audio quality and more interfacing options we would use an external I2S codec to do all the analog input and output signal processing. The codec chip, depending on type, may provide additional functionality like audio input signal preamplifier, headphone output amplifier, multiple analog input and outputs, sound effects, etc. The I2S is considered as the industry standard for interfacing with audio codec chips, or in general for a high speed, continuous transfer of the audio data. To optimize performance of audio data processing, an additional memory may be required. For such cases, please consider using [ESP32-WROVER-E](#) that provides 8 MB PSRAM on a single module together with the ESP32 chip.

The ESP-ADF is designed primarily to support projects with a codec chip. The [ESP32 LyraT](#) board is an example of such a project. The software interfacing with the board is done by Audio HAL and a driver. The codec chip used on the ESP32 LyraT is [ES8388](#). Boards with a different codec chip may be supported by providing a different driver.

## 3.2 Design Considerations

Depending on the audio data format, that may be lossless, lossy or compressed, e.g. WAV, MP3 or FLAC and the quality expressed in sampling rate and bitrate, the project will require different resources: memory, storage space, input / output throughput and the processing power. The resources will also depend on the project type and features discussed in [Project Design](#).

This section describes capacity and performance of ESP32 system resources that should be considered when designing an audio project to meet required data format, audio quality and functionality.

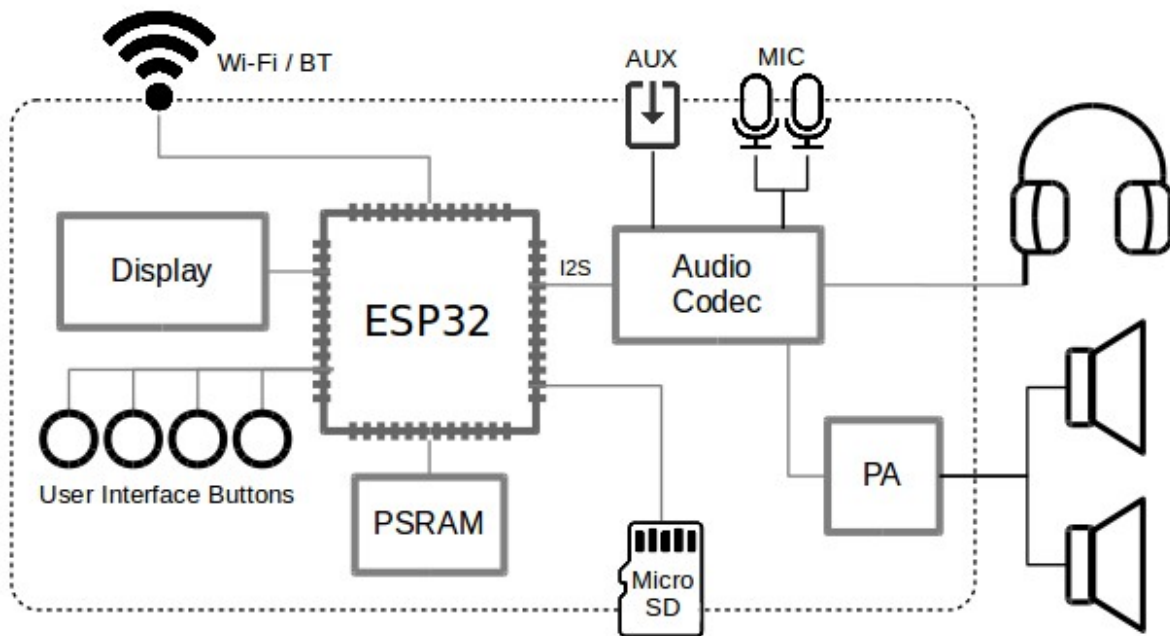


Fig. 3: Typical Audio Project Example

### 3.2.1 Memory

The spare internal Data-RAM is about 290kB with “hello\_world” example. For audio system this may be insufficient, and therefore the ESP32 incorporates the ability to use up to 4MB of external SPI RAM (i.e. PSRAM) memory. The external memory is incorporated in the memory map and is, within certain restrictions, usable in the same way internal Data-RAM is.

Refer to [External SPI-connected RAM](#) section in IDF documentation for details, especially pay attention to its [Restrictions](#) section which is very important.

To be able to use the PSRAM, if installed on your board, it should be enabled in menuconfig under *Component config > ESP32-specific > SPI RAM config*. The option `CONFIG_SPIRAM_CACHE_WORKAROUND`, set by default in the same menu, should be kept enabled.

---

**Note:** Bluetooth and Wi-Fi can not coexist without PSRAM because it will not leave enough memory for an audio application.

---

## Optimization of Internal RAM and Use of PSRAM

Internal RAM is more valuable asset since there are some restrictions on PSRAM. Here are some tips for optimizing internal RAM.

- If PSRAM is in use, set all the static buffer to minimum value in *Component config* > *Wi-Fi*; if PSRAM is not used then dynamic buffer should be selected to save memory. Refer to [Wi-Fi Buffer Usage](#) section in IDF documentation for details.
- If PSRAM and BT are used, then `CONFIG_BT_ALLOCATION_FROM_SPIRAM_FIRST` and `CONFIG_BT_BLE_DYNAMIC_ENV_MEMORY` should be set as “yes” under *Component config* > *Bluetooth* > *Bluedroid Enable*, to allocate more of 40kB memory to PSRAM
- If PSRAM and Wi-Fi are used, then `CONFIG_WIFI_LWIP_ALLOCATION_FROM_SPIRAM_FIRST` should be set as “yes” under *Component config* > *ESP32-specific* > *SPI RAM config*, to allocate some memory to PSRAM
- Set `CONFIG_WL_SECTOR_SIZE` as 512 in *Component config* > *Wear Levelling*

**Note:** The smaller the size of sector be, the slower the Write / Read speed will be, and vice versa, but only 512 and 4096 are supported.

- Call `char *buf = heap_caps_malloc(1024 * 10, MALLOC_CAP_SPIRAM | MALLOC_CAP_8BIT)` instead of `malloc(1024 * 10)` to use PSRAM, and call `char *buf = heap_caps_malloc(512, MALLOC_CAP_INTERNAL | MALLOC_CAP_8BIT)` to use internal RAM.
- Not relying on `malloc()` to automatically allocate PSRAM allows to make a full control of the memory. By avoiding the use of the internal RAM by other `malloc()` calls, you can reserve more memory for high-efficiency usage and task stack since PSRAM cannot be used as task stack memory.
- The task stack will always be allocated at internal RAM. On the other hand you can use of the `xTaskCreateStatic()` function that allows to create tasks with stack on PSRAM (see options in PSRAM and FreeRTOS menuconfig), but pay attention to its help information.

**Important:** Don't use ROM code in `xTaskCreateStatic` task. The ROM code itself is linked in `components/esp32/ld/esp32.rom.ld`. However, you also need to consider other pieces of code that *call* ROM functions, as well as the code that is not recompiled against the `CONFIG_SPIRAM_CACHE_WORKAROUND` patch, like the Wi-Fi and Bluetooth libraries. In general, we advise using this only in threads that do not call any IDF libraries (including `libc`), doing only calculations and using FreeRTOS primitives to talk to other threads.

## Memory Usage by Component Overview

Below is a table that contains ESP-ADF components and their memory usage. Choose the components needed and find out how much internal RAM is left. The table is divided into two parts, when PSRAM is used or not. If PSRAM (external RAM) is in use, then some of the memory will be allocated at PSRAM automatically.

The initial spare internal RAM is 290kB.

Component	Internal RAM Required	
	PSRAM not used	With PSRAM
Wi-Fi <sup>1</sup>	50kB+	50kB+
Bluetooth	140kB (50kB if only BLE needed)	95kB (50kB if only BLE needed)
Flash Card <sup>2</sup>	12kB+	12kB+
I2S <sup>3</sup>	Configurable, 8kB for reference	Configurable, 8kB for reference
RingBuffer <sup>4</sup>	Configurable, 30kB for reference	0kB, all moved into PSRAM

### Notes to the table above

1. According to the Wi-Fi menuconfig each Tx and Rx buffer occupies 1.6kB internal RAM. The value of 50kB RAM is assuming use of 5 Rx static buffers and 6 Tx static buffers. If PSRAM is not in use, then the “Type of WiFi Tx Buffer” option should be set as *DYNAMIC* in order to save RAM, in this case, the RAM usage will be far less than 50kB, but programmer should keep at least 50kB available for the Wi-Fi to be able to transmit the data. **[Internal RAM only]**
2. Taking ESP32-LyraT V4.3 as an example, depending on value of *SDCARD\_OPEN\_FILE\_NUM\_MAX* in `audio_board/lyrat_v4_3/board_def.h`, that is then used in `sdcard_mount()` function, the RAM needed will increase with a greater number of maximum open files. 12kB is the RAM needed with 5 max files and 512 bytes *CONFIG\_WL\_SECTOR\_SIZE*. **[Internal RAM only]**
3. Depending on configuration settings of the I2S stream, refer to `audio_stream/include/i2s_stream.h` and `audio_stream/i2s_stream.c`. **[Internal RAM only]**
4. Depending on configuration setting of the Ringbuffer, refer to *DEFAULT\_PIPELINE\_RINGBUF\_SIZE* in `audio_pipeline/include/audio_pipeline.h` or user setting, if the buffer is created with e.g. `rb_create()`.

## 3.2.2 System Settings

The following settings are recommended to achieve a high Wi-Fi performance in an audio project.

---

**Note:** Use ESP32 modules and boards from reputable vendors that put attention to product design, component selection and product testing. This is to have confidence of receiving well designed boards with calibrated RF.

---

- Set these following options in menuconfig.
  - Flash SPI mode as QIO
  - Flash SPI speed as 80MHz
  - CPU frequency as 240MHz
  - Set *Default receive window size* as 5 times greater than *Maximum Segment Size* in *Component config* > *LWIP* > *TCP*
- If external antenna is used, then set *PHY\_RF\_CAL\_PARTIAL* as *PHY\_RF\_CAL\_FULL* in “`esp-idf/components/esp32/phy_init.c`”

## 3.3 Software Design

Espressif audio framework project.

### 3.3.1 Features

1. All of Streams and Codecs based on audio element.
2. All events based on queue.
3. Audio pipeline supports dynamic combination.
4. Audio pipeline supports multiple elements.
5. Pipeline Support functionality plug-in.



6. Audio common peripherals support work in the one task.
7. Support post-event mechanism in peripherals.
8. Support high level audio play API based on element and audio pipeline.
9. Audio high level interface supports dynamic adding of codec library.
10. Audio high level interface supports dynamic adding of input and output stream.
11. ESP audio supports multiple audio pipelines.

### 3.3.2 Design Components

Five basic components are - Audio Element, Audio Event, Audio Pipeline, ESP peripherals, ESP audio

#### Audio Element

##### Example

Please refer to `audio_stream/fatfs_stream.c` for an example of using an audio element.

#### Audio Event

##### Example

Please refer to `player/pipeline_http_mp3/main/play_http_mp3_example.c` for an example of using an audio event.

#### Audio Pipeline

##### Example

Please refer to `player/pipeline_play_sdcard_music/main/play_sdcard_music_example.c` for an example of linking elements into an audio pipeline.

#### Audio Peripheral

##### Example

```
ESP_LOGI(TAG, "[ 3 ] Start and wait for Wi-Fi network");
esp_periph_config_t periph_cfg = DEFAULT_ESP_PERIPH_SET_CONFIG();
esp_periph_set_handle_t set = esp_periph_set_init(&periph_cfg);
periph_wifi_cfg_t wifi_cfg = {
    .wifi_config.sta.ssid = CONFIG_WIFI_SSID,
    .wifi_config.sta.password = CONFIG_WIFI_PASSWORD,
};
esp_periph_handle_t wifi_handle = periph_wifi_init(&wifi_cfg);
esp_periph_start(set, wifi_handle);
periph_wifi_wait_for_connected(wifi_handle, portMAX_DELAY);
```

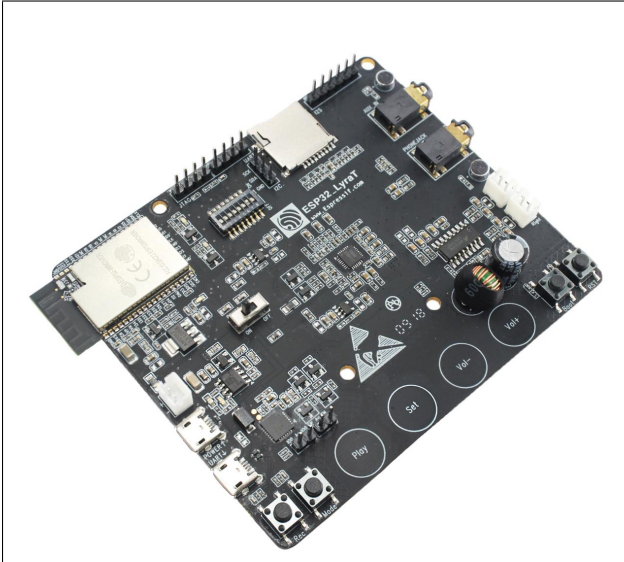
### Audio Player

#### Example

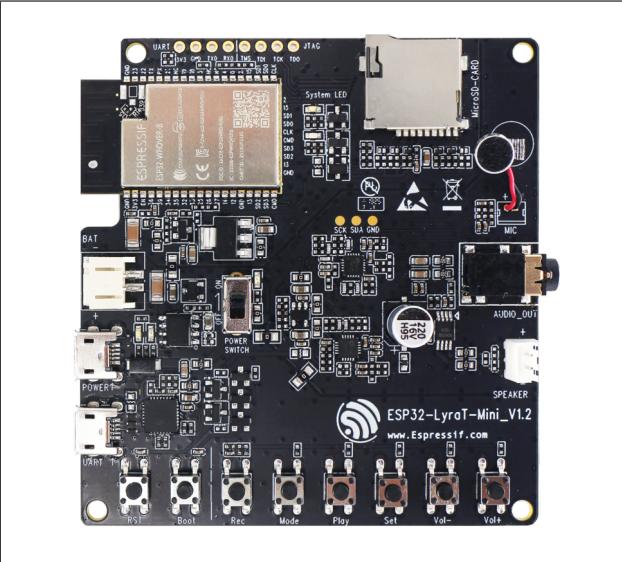
Please refer to [cli](#) for an example of initializing `esp_audio` as an audio player.

## 3.4 Development Boards

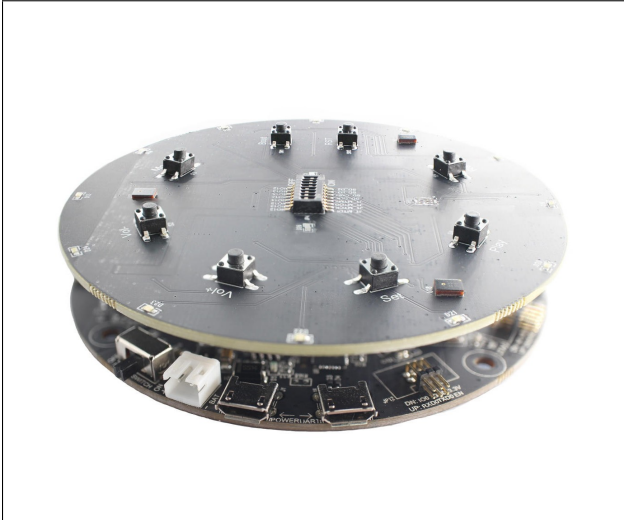
Below are getting started guides and hardware details of audio development boards designed by Espressif.



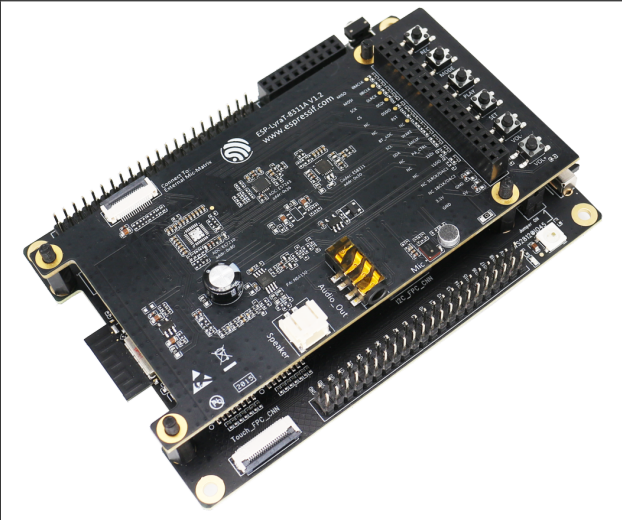
Getting Started with ESP32-LyraT  
Hardware Reference of ESP32-LyraT



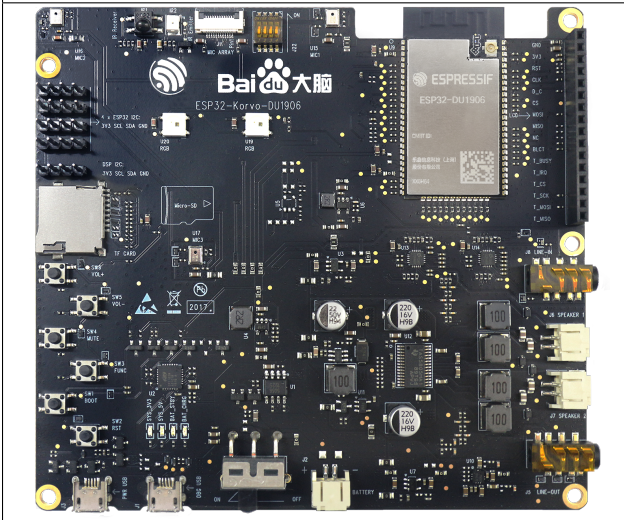
Getting Started with ESP32-LyraT-Mini  
Hardware Reference of ESP32-LyraT-Mini



Getting Started with ESP32-LyraTD-MS-C



Getting Started with ESP32-S2-Kaluga-1-Kit



Getting Started with ESP32-Korvo-DU1906



Getting Started with ESP32-S3-Korvo-2

3.4. Development Boards

### 3.4.1 ESP32-LyraT-Mini V1.2 Getting Started Guide

This guide provides users with functional descriptions, configuration options for ESP32-LyraT-Mini V1.2 audio development board, as well as how to get started with the ESP32-LyraT board.

The ESP32-LyraT is a hardware platform designed for the dual-core ESP32 audio applications, e.g., Wi-Fi or BT audio speakers, speech-based remote controllers, connected smart-home appliances with one or more audio functionality, etc.

The ESP32-LyraT-Mini is a mono audio board. If you are looking for a stereo audio board, check [ESP32-LyraT V4.3 Getting Started Guide](#).

#### What You Need

- [ESP32-LyraT-Mini V1.2 board](#)
- Speaker or headphones with a 3.5 mm jack. If you use a speaker, it is recommended to choose one no more than 3 watts, and JST PH 2.0 2-Pin plugs are needed. In case you do not have this type of plug it is also fine to use Dupont female jumper wires during development.
- Two Micro-USB 2.0 cables, Type A to Micro B
- PC loaded with Windows, Linux or Mac OS

Optional components

- Micro SD-card
- Li-ion Battery

If you like to start using this board right now, go directly to section [Start Application Development](#).

#### Overview

The ESP32-LyraT-Mini V1.2 is an audio development board produced by [Espressif](#) built around ESP32. It is intended for audio applications, by providing hardware for audio processing and additional RAM on top of what is already on-board of the ESP32 chip. The specific hardware includes:

- **ESP32-WROVER-E module**
- **Audio codec chip**
- **ADC chip**
- **Microphone** on board
- **Audio output**
- **1 x 3-watt speaker** output
- **MicroSD card** slot (1 line)
- **Eight keys**
- **Two system LEDs**
- **JTAG** and **UART** test points
- Integrated **USB-UART Bridge Chip**
- Li-ion **Battery-Charge Management**

The block diagram below presents main components of the ESP32-LyraT-Mini and interconnections between components.

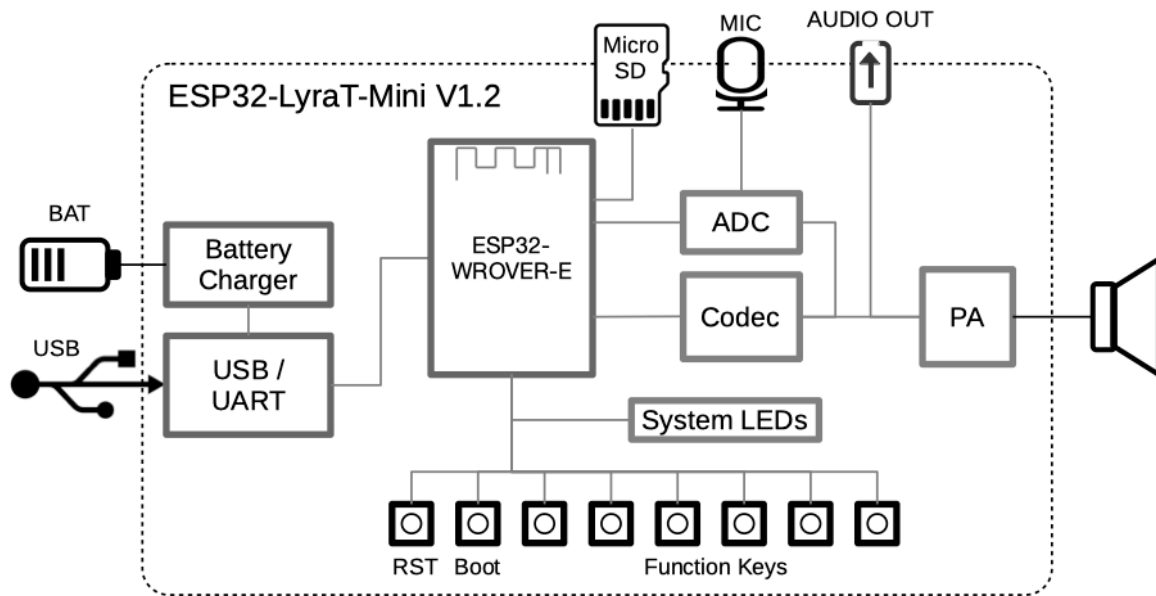


Fig. 4: ESP32-LyraT-Mini Block Diagram

## Components

The following list and figure describe key components, interfaces and controls of the ESP32-LyraT-Mini used in this guide. For detailed technical documentation of this board, please refer to [ESP32-LyraT-Mini V1.2 Hardware Reference](#) and [ESP32-LyraT-Mini V1.2 schematic](#) (PDF). The list below provides description starting from the picture's top right corner and going clockwise.

**Audio Codec Chip** The audio codec chip, [ES8311](#), is a low power mono audio codec. It consists of 1-channel ADC, 1-channel DAC, low noise pre-amplifier, headphone driver, digital sound effects, analog mixing and gain functions. It is interfaced with **ESP32-WROVER-E Module** over I2S and I2C buses to provide audio processing in hardware independently from the audio application.

**Audio Output** Output socket to connect headphones with a 3.5 mm stereo jack. (Please note that the board outputs a mono signal)

**Speaker Output** Output socket to connect a speaker. The 4-ohm and 3-watt speaker is recommended. The pins have a 2.00 mm / 0.08" pitch.

**USB-UART Port** Functions as the communication interface between a PC and the ESP32.

**USB Power Port** Provides the power supply for the board.

**Standby / Charging LEDs** The **Standby** green LED indicates that power has been applied to the **USB Power Port**. The **Charging** red LED indicates that a battery connected to the **Battery Socket** is being charged.

**Power On Switch** Power on/off knob: toggling it to the top powers the board on; toggling it to the down powers the board off.

**Power On LED** Red LED indicating that **Power On Switch** is turned on.

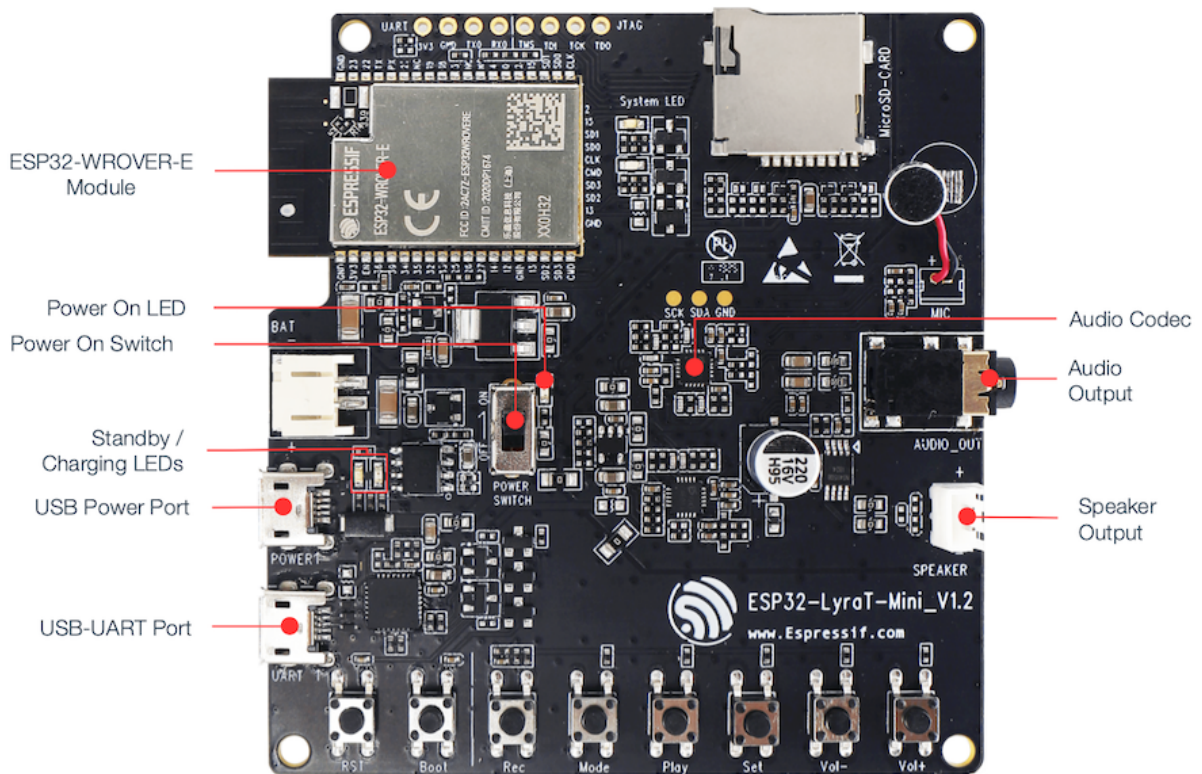


Fig. 5: ESP32 LyraT-Mini V1.2 Board Layout Overview

**ESP32-WROVER-E Module** The ESP32-WROVER-E module contains ESP32 chip to provide Wi-Fi / Bluetooth connectivity and data processing power as well as integrates 4 MB external SPI flash and an additional 8 MB PSRAM for flexible data storage.

## Start Application Development

Before powering up the ESP32-LyraT-Mini, please make sure that the board has been received in good condition with no obvious signs of damage.

### Initial Setup

Prepare the board for loading of the first sample application:

1. Connect speaker to the **Speaker Output**. Connecting headphones to the **Audio Output** is an option.
2. Plug in the Micro-USB cables to the PC and to **both USB ports** of the ESP32-LyraT-Mini.
3. The **Standby LED** (green) should turn on. Assuming that a battery is not connected, the **Charging LED** (red) will blink every couple of seconds.
4. Toggle top the **Power On Switch**.
5. The red **Power On LED** should turn on.

If this is what you see on the LEDs, the board should be ready for application upload. Now prepare the PC by loading and configuring development tools what is discussed in the next section.

### Develop Applications

Once the board is initially set up and checked, you can start preparing the development tools. The Section *Installation Step by Step* will walk you through the following steps:

- **Set up ESP-IDF** to get a common development framework for the ESP32 (and ESP32-S2) chips in C language;
- **Get ESP-ADF** to install the API specific to audio applications;
- **Set up env** to make the framework aware of the audio specific API;
- **Start a Project** that will provide a sample audio application for the board;
- **Connect Your Device** to prepare the application for loading;
- **Build the Project** to finally run the application and play some music.

### Revision History

- Changed the integrated module to ESP32-WROVER-E from ESP32-WROVER-B.

### Other Boards from LyraT Family

- *ESP32-LyraT V4.3 Getting Started Guide*
- *ESP32-LyraTD-MSC V2.2 Getting Started Guide*

### Related Documents

- *ESP32-LyraT-Mini V1.2 schematic (PDF)*
- *ESP32-LyraT-Mini V1.2 Board Dimensions (PDF)*
- *ESP32-LyraT-Mini V1.2 Hardware Reference*
- *ESP32 Datasheet (PDF)*
- *ESP32-WROVER-E Datasheet (PDF)*

### 3.4.2 ESP32-LyraT-Mini V1.2 Hardware Reference

This guide provides functional descriptions and configuration options for ESP32-LyraT-Mini V1.2 audio development board. As an introduction to functionality and using the LyraT, please see *ESP32-LyraT-Mini V1.2 Getting Started Guide*.

#### In this Section

- *Overview*
- *Functional Description*
- *Allocation of ESP32 Pins to Test Points*
  - *JTAG Test Point*
  - *UART Test Point*
- *MicroSD Card*
- *GPIO Allocation Summary*
- *Notes on Power Distribution*
  - *Power Supply over USB and from Battery*
  - *Independent Audio and Digital Power Supply*
- *Selecting of the Audio Output*
- *Related Documents*



## Overview

ESP32-LyraT is a hardware platform designed for the dual-core ESP32 audio applications, e.g., Wi-Fi or BT audio speakers, speech-based remote controllers, connected smart-home appliances with one or more audio functionality, etc.

The block diagram below presents main components of ESP32-LyraT-Mini.

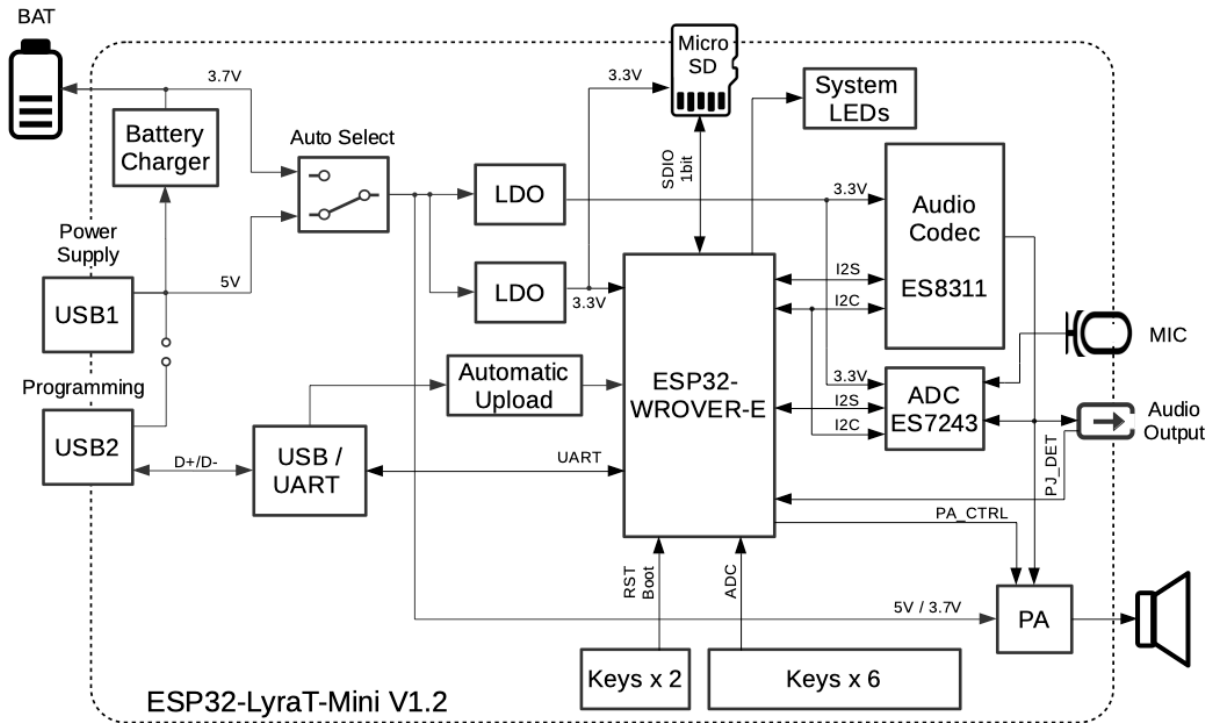


Fig. 6: ESP32-LyraT-Mini V1.2 Electrical Block Diagram

## Functional Description

The following list and figure describe key components, interfaces, and controls of the ESP32-LyraT-Mini board. The list provides description starting from the picture's top right corner and going clockwise.

**MicroSD Slot** The development board supports a MicroSD card in SPI/1-bit modes, and can store or play audio files in the MicroSD card. See [MicroSD Card](#) for pinout details.

**Microphone** On-board microphone connected to AINRP/AINRP of the **Audio ADC Chip**.

**System LEDs** Two general purpose LEDs (green and red) controlled by **ESP32-WROVER-E Module** to indicate certain operation states of the audio application using dedicated API.

**Audio Codec** The audio codec chip, **ES8311**, is a low power mono audio codec. It consists of 1-channel ADC, 1-channel DAC, low noise pre-amplifier, headphone driver, digital sound effects, analog mixing, and gain functions. It is interfaced with **ESP32-WROVER-E Module** over I2S and I2C buses to provide audio processing in hardware independently from the audio application.

**Audio Output** Output socket to connect headphones with a 3.5 mm stereo jack. One of the socket's terminals is wired to ESP32 to provide jack insertion detection.

**ADC** The audio codec chip, **ES7243**, is a low power multi-bit delta-sigma audio ADC and DAC. In this board this chip is used as the microphone interface.

**PA** A power amplifier used to amplify the audio signal from the **Audio Codec Chip** for driving the speaker.

**Speaker Output** Output socket to connect a speaker. The 4-ohm and 3-watt speaker is recommended. The pins have a 2.00 mm / 0.08" pitch.

**Function Press Keys** Six press keys labeled **Rec**, **Mode**, **Play**, **Set**, **Vol-**, and **Vol+**. They are routed to **ESP32-WROVER-E Module** and intended for development and testing of a UI for audio applications using dedicated API.

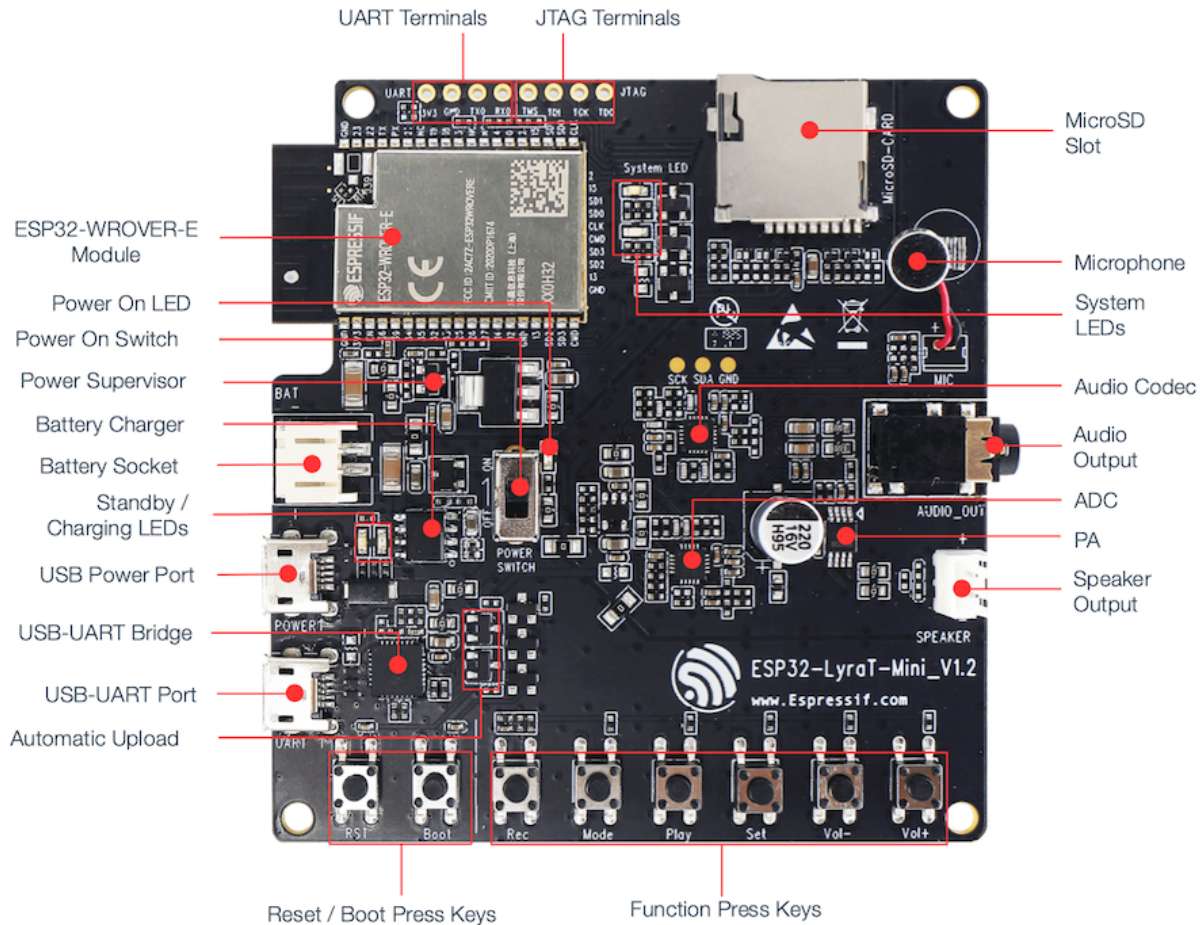


Fig. 7: ESP32 LyraT-Mini V1.2 Board Layout

**Boot/Reset Press Keys** **Boot**: holding down the **Boot** button and momentarily pressing the **Reset** button initiates the firmware upload mode. Then user can upload firmware through the serial port. **Reset**: pressing this button alone resets the system.

**Automatic Upload** A simple two transistor circuit to put ESP32 into firmware upload mode depending on the status of UART DTR and RTS signals. The signals are controlled by an external application to upload the firmware over the USB-UART interface.

**USB-UART Port** Functions as the communication interface between a PC and the ESP32 module.

**USB-UART Bridge** A single chip USB-UART bridge CP2102N provides up to 3 Mbps transfers rates.

**USB Power Port** Provides the power supply for the board.

**Standby/Charging LEDs** The **Standby** green LED indicates that power has been applied to the **USB Power Port**. The **Charging** red LED indicates that a battery connected to the **Battery Socket** is being charged.

**Battery Socket** Two-pin socket to connect a single cell Li-ion battery. The pins have a 2.00 mm / 0.08" pitch. The battery serves as an alternative power supply to the **USB Power Port** for charging the board. Make sure to use a Li-ion battery that has protection circuit and fuse. The recommended specifications of the battery: capacity > 1000 mAh, output voltage 3.7 V, input voltage 4.2 V – 5 V. Please verify if polarity on the battery plug matches polarity of the socket as marked on the board's soldermask besides the socket.

**Battery Charger** Constant current and constant voltage linear charger for single cell lithium-ion batteries AP5056. Used for charging of a battery connected to the **Battery Socket** over the **USB Power Port**.

**Power Supervisor** Provides EN signal to enable ESP32 once power supply voltage stabilizes.

**Power On Switch** Power on/off knob: toggling it to the top powers the board on; toggling it to the down powers the board off.

---

**Note:** The **Power On Switch** does not affect / disconnect the Li-ion battery charging. More information, you can refer to [ESP32-LyraT-Mini V1.2 schematic \(PDF\)](#).

---

**Power On LED** Red LED indicating that **Power On Switch** is turned on.

**ESP32-WROVER-E Module** The ESP32-WROVER-E module contains ESP32 chip to provide Wi-Fi / Bluetooth connectivity and data processing power as well as integrates 4 MB external SPI flash and an additional 8 MB PSRAM for flexible data storage.

**UART Termininals** Serial port: provides access to the serial TX/RX signals between **ESP32-WROVER-E Module** and **USB-UART Bridge Chip**. See [UART Test Point](#) for pinout details.

**JTAG Termininals** Provides access to the **JTAG** interface of **ESP32-WROVER-E Module**. It may be used for debugging, application upload, as well as implementing several other functions, e.g., [Application Level Tracing](#). See [JTAG Test Point](#) for pinout details.

### Allocation of ESP32 Pins to Test Points

This section describes allocation of test points available on the ESP32-LyraT-Mini board.

The test points are bare through hole solder pads and have standard 2.54 mm / 0.1 inch pitch. User may need to populate them with pin headers or sockets for easy connection of external hardware.

#### JTAG Test Point

.	ESP32 Pin	JTAG Signal
1	MTDO / GPIO15	TDO
2	MTCK / GPIO13	TCK
3	MTDI / GPIO12	TDI
4	MTMS / GPIO14	TMS

## UART Test Point

.	ESP32 Pin	Pin Description
1	RXD0	RX
2	TXD0	TX
3	GND	GND
4	n/a	3.3 V

## MicroSD Card

Implemented on this board MicoSD card interface operates in SPI/1-bit mode. The board is able to support SPI/4-b it mode after populating couple of additional components on locations reserved on the PCB. See [ESP32-LyraT-Mini V1.2 schematic \(PDF\)](#) for additional information. Not populated components are marked (*NC*) on the schematic.

.	ESP32 Pin	MicroSD Signal
1	MTDI / GPIO12	–
2	MTCK / GPIO13	–
3	MTDO / GPIO15	CMD
4	MTMS / GPIO14	CLK
5	GPIO2	DATA0
6	GPIO4	–
7	GPIO34	CD

## GPIO Allocation Summary

The table below provides allocation of GPIOs exposed on terminals of **ESP32-WROVER-E Module** to control specific components or functions of the board.

Pin <sup>1</sup>	Pin Name	ES8311	ES7243	Keys	MicroSD	Other
3	EN			EN_KEY		
4	S_VP		I2S_DATA			
5	S_VN			REC, MODE, PLAY, SET, VOL-, VOL+		
6	IO34				CD	
7	IO35	I2S0_ASDOUT				
8	IO32		I2S1_SCLK			
9	IO33		I2S1_LRCK			
10	IO25	I2S0_LRCK				
11	IO26	I2S0_DSDIN				
12	IO27					Blue_LED
13	IO14				CLK	
14	IO12				NC (DATA2)	
16	IO13				NC (DATA3)	
17	SD2					
18	SD3					
19	CMD					
20	CLK					
21	SD0					
22	SD1					

continues on next page

Table 1 – continued from previous page

Pin <sup>1</sup>	Pin Name	ES8311	ES7243	Keys	MicroSD	Other
23	IO15				CMD	
24	IO2			IO2_KEY	DATA0	
25	IO0	I2S0_MCLK	I2S1_MCLK	IO0_KEY		
26	IO4				NC (DATA1)	
27	NC (IO16)					
28	NC (IO17)					
29	IO5	I2S0_SCLK				
30	IO18	I2C_SDA	I2C_SDA			
31	IO19					PJ_DET <sup>2</sup>
33	IO21					PA_CTRL <sup>3</sup>
34	RXD0					RXD0 <sup>4</sup>
35	TXD0					TXD0 <sup>4</sup>
36	IO22					Green_LED
37	IO23	I2C_SCK	I2C_SCL			

1. **Pin** - ESP32-WROVER-E module pin number, GND and power supply pins are not listed
2. **PJ\_DET** - phone jack insertion detect signal
3. **PA\_CTRL** - NS4150 power amplifier chip control signal
4. **RXD0, TXD0** - serial communication signals connected to TXD and RXD pins of CP2102N USB-UART bridge
5. **NC** - not connected

### Notes on Power Distribution

The ESP32-LyraT-Mini board provides some basic features to isolate noise from digital components by providing separate power distribution for audio and digital subsystems.

### Power Supply over USB and from Battery

There are two ways to power the development board: 5 V USB Power Port or 3.7 V optional battery. The optional battery is preferable for applications where a cleaner power supply is required.

### Independent Audio and Digital Power Supply

The board features independent power supplies to the audio components and the ESP32 module. This should reduce noise in the audio signal from digital components and improve overall performance of the components.

### Selecting of the Audio Output

The board provides a mono audio output signal on pins OUTN and OUTP of the ES8311 codec chip. The signal is routed to two outputs:

- Power amplifier (PA) to drive an external speaker
- Phone jack socket to drive external headphones

The board design assumes that selection between one of these outputs is implemented in software, as opposed to using traditional mechanical contacts in a phone jack socket, that would disconnect the speaker once a headphone jack is inserted.

## Power System:

USB<->UART :

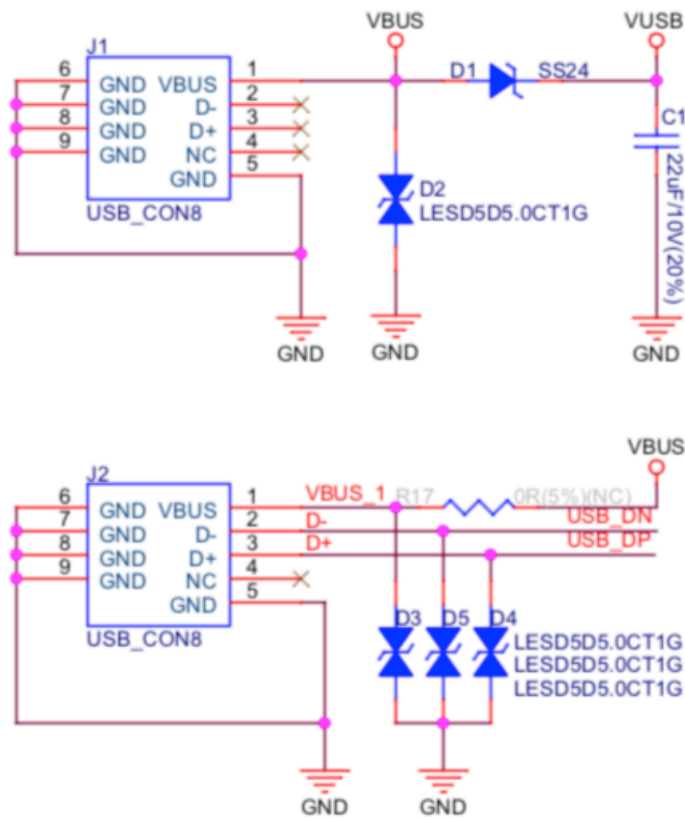


Fig. 8: ESP32-LyraT-Mini V1.2 - Dedicated USB Power Supply Socket

### Charge Circuit:

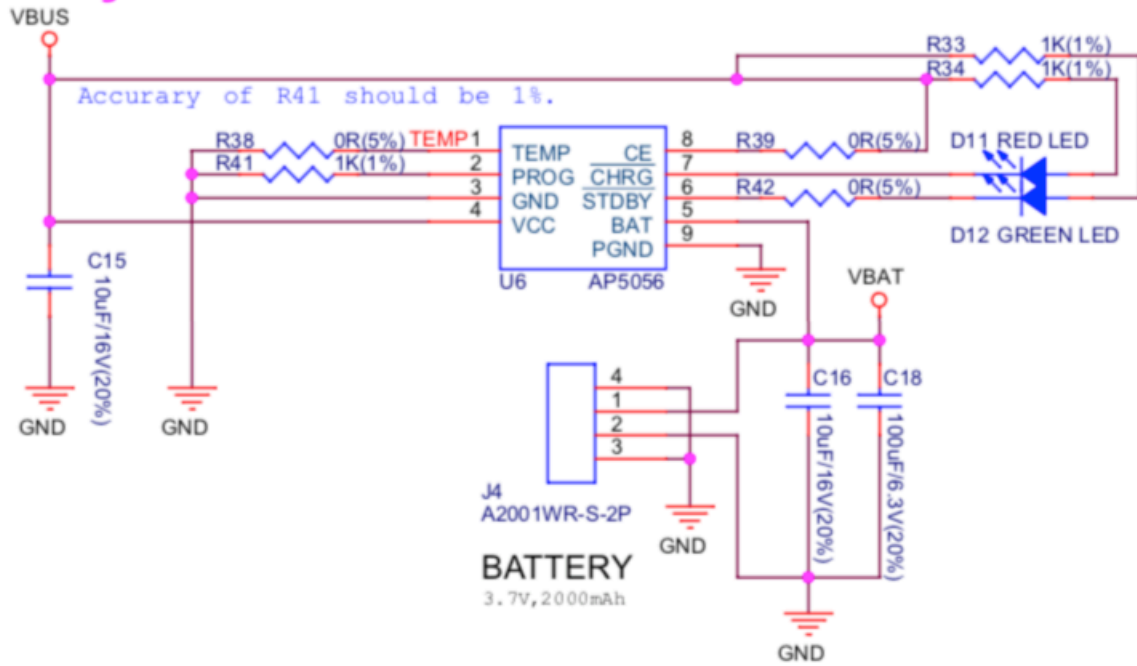


Fig. 9: ESP32-LyraT-Mini V1.2 - Power Supply from a Battery

### Module Power Supply:

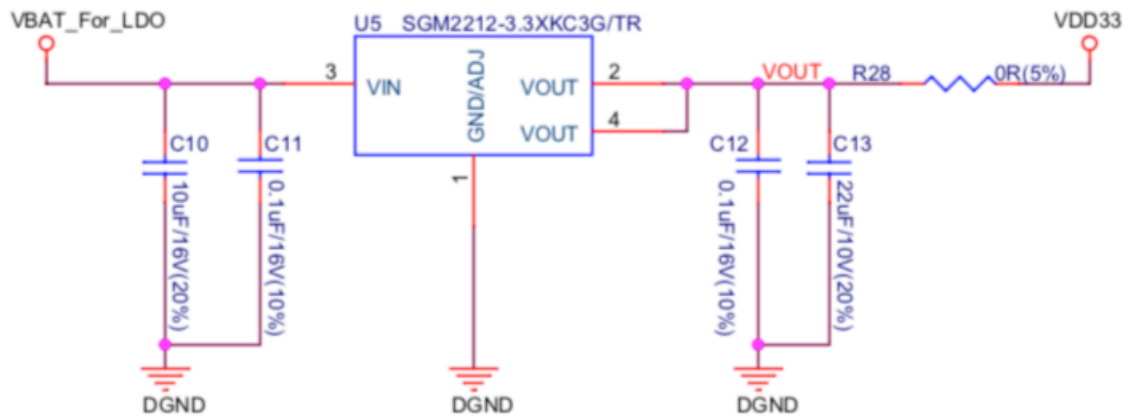


Fig. 10: ESP32-LyraT-Mini V1.2 - Digital Power Supply

Audio Power Supply:

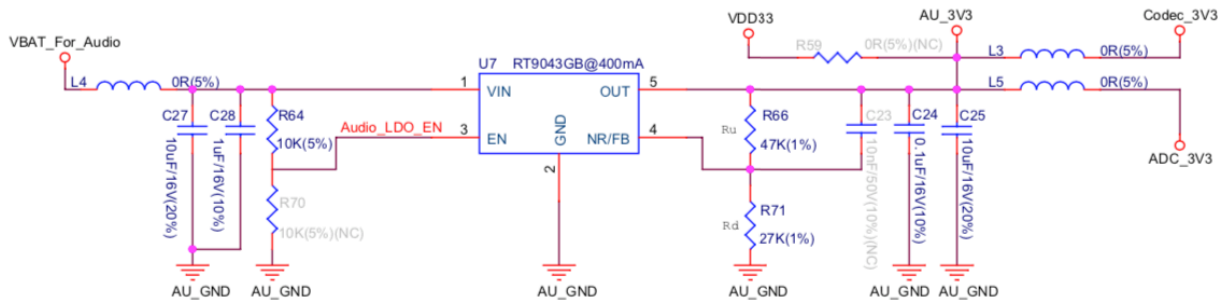


Fig. 11: ESP32-LyraT-Mini V1.2 - Audio Power Supply

Two digital IO signals are provided to implement selection between the speaker and the headphones:

- **PJ\_DET** - digital input signal to detect when a headphone jack is inserted,
- **PA\_CTRL** - digital output signal to enable or disable the amplifier IC.

The application running on ESP32 may then enable or disable the PA with **PA\_CTRL** basing on status of **PJ\_DET**. Please see *GPIO Allocation Summary* for specific GPIO numbers allocated to these signals.

Related Documents

- [ESP32-LyraT-Mini V1.2 Schematic \(PDF\)](#)
- [ESP32-LyraT-Mini V1.2 Board Dimensions \(PDF\)](#)
- [ESP32-LyraT-Mini V1.2 Getting Started Guide](#)
- [ESP32 Datasheet \(PDF\)](#)
- [ESP32-WROVER-E Datasheet \(PDF\)](#)

3.4.3 ESP32-LyraT V4.3 Getting Started Guide

This guide provides users with functional descriptions, configuration options for ESP32-LyraT V4.3 audio development board, as well as how to get started with the ESP32-LyraT board. Check section *Other Versions of LyraT*, if you have different version of this board.

The ESP32-LyraT is a hardware platform designed for the dual-core ESP32 audio applications, e.g., Wi-Fi or Bluetooth audio speakers, speech-based remote controllers, connected smart-home appliances with one or more audio functionality, etc.

The ESP32-LyraT is a stereo audio board. If you are looking for a mono audio board, intended for lower end applications, check *ESP32-LyraT-Mini V1.2 Getting Started Guide*.



## What You Need

- 1 × *ESP32 LyraT V4.3 board*
- 2 × Speaker or headphones with a 3.5 mm jack. If you use a speaker, it is recommended to choose one no more than 3 watts, and JST PH 2.0 2-Pin plugs are needed. In case you do not have this type of plug it is also fine to use Dupont female jumper wires during development.
- 2 × Micro-USB 2.0 cables, Type A to Micro B
- 1 × PC loaded with Windows, Linux or Mac OS

If you like to start using this board right now, go directly to section *Start Application Development*.

## Overview

The ESP32-LyraT V4.3 is an audio development board produced by [Espressif](#) built around ESP32. It is intended for audio applications, by providing hardware for audio processing and additional RAM on top of what is already onboard of the ESP32 chip. The specific hardware includes:

- ESP32-WROVER-E Module
- Audio Codec Chip
- Dual Microphones on board
- Headphone output
- 2 × 3-watt Speaker output
- Dual Auxiliary Input
- MicroSD Card slot (1 line or 4 lines)
- Six buttons (2 physical buttons and 4 touch buttons)
- JTAG header
- Integrated USB-UART Bridge Chip
- Li-ion Battery-Charge Management

The block diagram below presents main components of the ESP32-LyraT and interconnections between components.

## Components

The following list and figure describe key components, interfaces and controls of the ESP32-LyraT used in this guide. This covers just what is needed now. For detailed technical documentation of this board, please refer to *ESP32-LyraT V4.3 Hardware Reference* and *ESP32 LyraT V4.3 schematic (PDF)*.

**ESP32-WROVER-E Module** The ESP32-WROVER-E module contains ESP32 chip to provide Wi-Fi / Bluetooth connectivity and data processing power as well as integrates 4 MB external SPI flash and an additional 8 MB PSRAM for flexible data storage.

**Headphone Output** Output socket to connect headphones with a 3.5 mm stereo jack.

---

**Note:** The socket may be used with mobile phone headsets and is compatible with OMPT standard headsets only. It does not work with CTIA headsets. Please refer to [Phone connector \(audio\)](#) on Wikipedia.

---

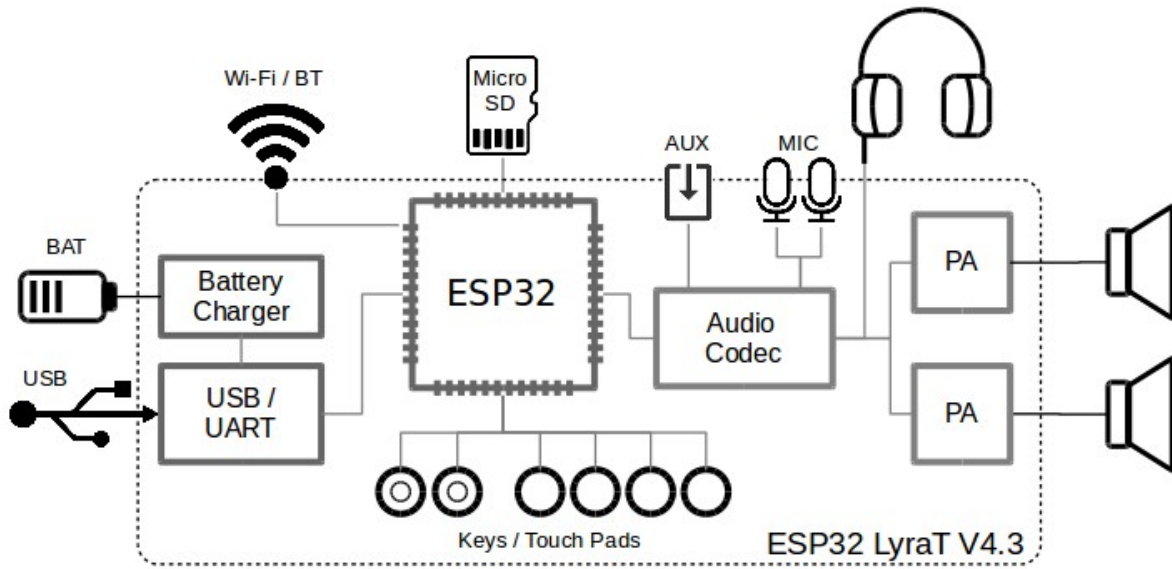


Fig. 12: ESP32-LyraT Block Diagram

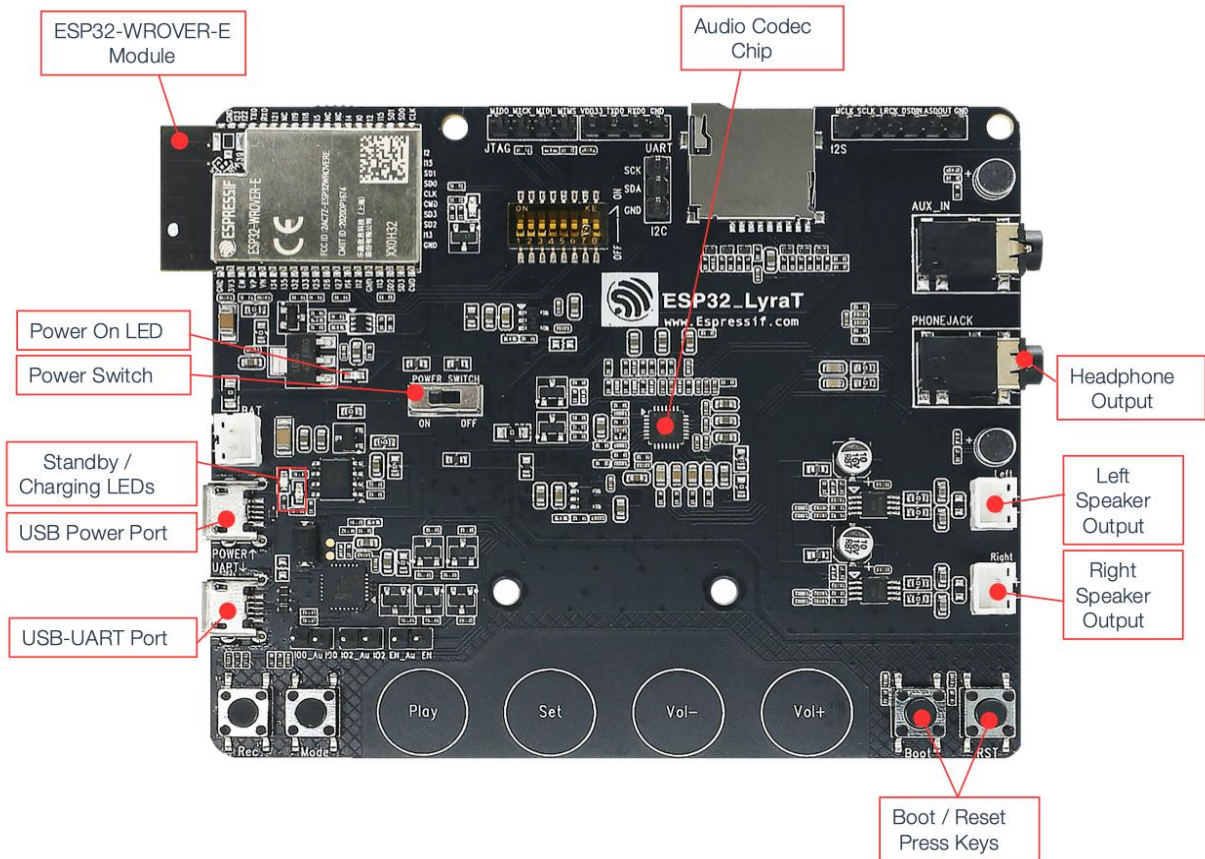


Fig. 13: ESP32-LyraT V4.3 Board Layout Overview

**Left Speaker Output** Output socket to connect a speaker. The 4-ohm and 3-watt speaker is recommended. The pins have a 2.00 mm / 0.08" pitch.

**Right Speaker Output** Output socket to connect a speaker. The 4-ohm and 3-watt speaker is recommended. The pins have a 2.00 mm / 0.08" pitch.

**Boot/Reset Press Keys** **Boot**: holding down the **Boot** button and momentarily pressing the **Reset** button initiates the firmware upload mode. Then user can upload firmware through the serial port. **Reset**: pressing this button alone resets the system.

**Audio Codec Chip** The Audio Codec Chip, [ES8388](#), is a low power stereo audio codec with a headphone amplifier. It consists of 2-channel ADC, 2-channel DAC, microphone amplifier, headphone amplifier, digital sound effects, analog mixing and gain functions. It is interfaced with **ESP32-WROVER-E Module** over I2S and I2S buses to provide audio processing in hardware independently from the audio application.

**USB-UART Port** Functions as the communication interface between a PC and the **ESP32-WROVER-E Module**.

**USB Power Port** Provides the power supply for the board.

**Standby / Charging LEDs** The **Standby** green LED indicates that power has been applied to the **Micro USB Port**. The **Charging** red LED indicates that a battery connected to the **Battery Socket** is being charged.

**Power Switch** Power on/off knob: toggling it to the left powers the board on; toggling it to the right powers the board off.

**Power On LED** Red LED indicating that **Power On Switch** is turned on.

## Start Application Development

Before powering up the ESP32-LyraT, please make sure that the board has been received in good condition with no obvious signs of damage.

### Initial Setup

Prepare the board for loading of the first sample application:

1. Connect speakers to the **Right** and **Left Speaker Output**. Connecting headphones to the **Headphone Output** is an option.
2. Plug in the Micro-USB cables to the PC and to **both USB ports** of the ESP32 LyraT.
3. The **Standby LED** (green) should turn on. Assuming that a battery is not connected, the **Charging LED** (red) will blink every couple of seconds.
4. Toggle left the **Power On Switch**.
5. The red **Power On LED** should turn on.

If this is what you see on the LEDs, the board should be ready for application upload. Now prepare the PC by loading and configuring development tools what is discussed in the next section.

### Develop Applications

Once the board is initially set up and checked, you can start preparing the development tools. The Section *Installation Step by Step* will walk you through the following steps:

- **Set up ESP-IDF** to get a common development framework for the ESP32 (and ESP32-S2) chips in C language;
- **Get ESP-ADF** to install the API specific to audio applications;
- **Set up env** to make the framework aware of the audio specific API;
- **Start a Project** that will provide a sample audio application for the board;
- **Connect Your Device** to prepare the application for loading;
- **Build the Project** to finally run the application and play some music.

### Summary of Key Changes from LyraT V4.2

- Changed the integrated module to ESP32-WROVER-E from ESP32-WROVER.
- Removed Red LED indicator light.
- Introduced headphone jack insert detection.
- Replaced single Power Amplifier (PA) chip with two separate chips.
- Updated power management design of several circuits: Battery Charging, ESP32, MicroSD, Codec Chip and PA.
- Updated electrical implementation design of several circuits: UART, Codec Chip, Left and Right Microphones, AUX Input, Headphone Output, MicroSD, Push Buttons and Automatic Upload.

### Other Versions of LyraT

- *ESP32-LyraT V4.2 Getting Started Guide*
- *ESP32-LyraT V4 Getting Started Guide*

### Other Boards from LyraT Family

- *ESP32-LyraT-Mini V1.2 Getting Started Guide*
- *ESP32-LyraTD-MSD V2.2 Getting Started Guide*

### Related Documents

- *ESP32-LyraT V4.3 Hardware Reference*
- ESP32 LyraT V4.3 schematic (PDF)
- ESP32-LyraT V4.3 Component Layout (PDF)
- ESP32 Datasheet (PDF)
- ESP32-WROVER-E Datasheet (PDF)

### 3.4.4 ESP32-LyraT V4.3 Hardware Reference

This guide provides functional descriptions, configuration options for ESP32-LyraT V4.3 audio development board. As an introduction to functionality and using the LyraT, please see *ESP32-LyraT V4.3 Getting Started Guide*. Check section *Other Versions of LyraT* if you have different version of the board.

#### In this Section

- *Overview*
- *Functional Description*
  - *Hardware Setup Options*
    - \* *Enable MicroSD Card in 1-wire Mode*
    - \* *Enable MicroSD Card in 4-wire Mode*
    - \* *Enable JTAG*
    - \* *Using Automatic Upload*
  - *Allocation of ESP32 Pins*
  - *Pinout of Extension Headers*
    - \* *UART Header / JP2*
    - \* *I2S Header / JP4*
    - \* *I2C Header / JP5*
    - \* *JTAG Header / JP7*
  - *Notes of Power Distribution*
    - \* *Power Supply Separation*
    - \* *Three Dedicated LDOs*
    - \* *Separate Power Feed for the PAs*
  - *Selecting of the Audio Output*
- *Other Versions of LyraT*
- *Related Documents*

#### Overview

The ESP32-LyraT development board is a hardware platform designed for the dual-core ESP32 audio applications, e.g., Wi-Fi or BT audio speakers, speech-based remote controllers, smart-home appliances with audio functionality(ies), etc.

The block diagram below presents main components of the ESP32-LyraT.

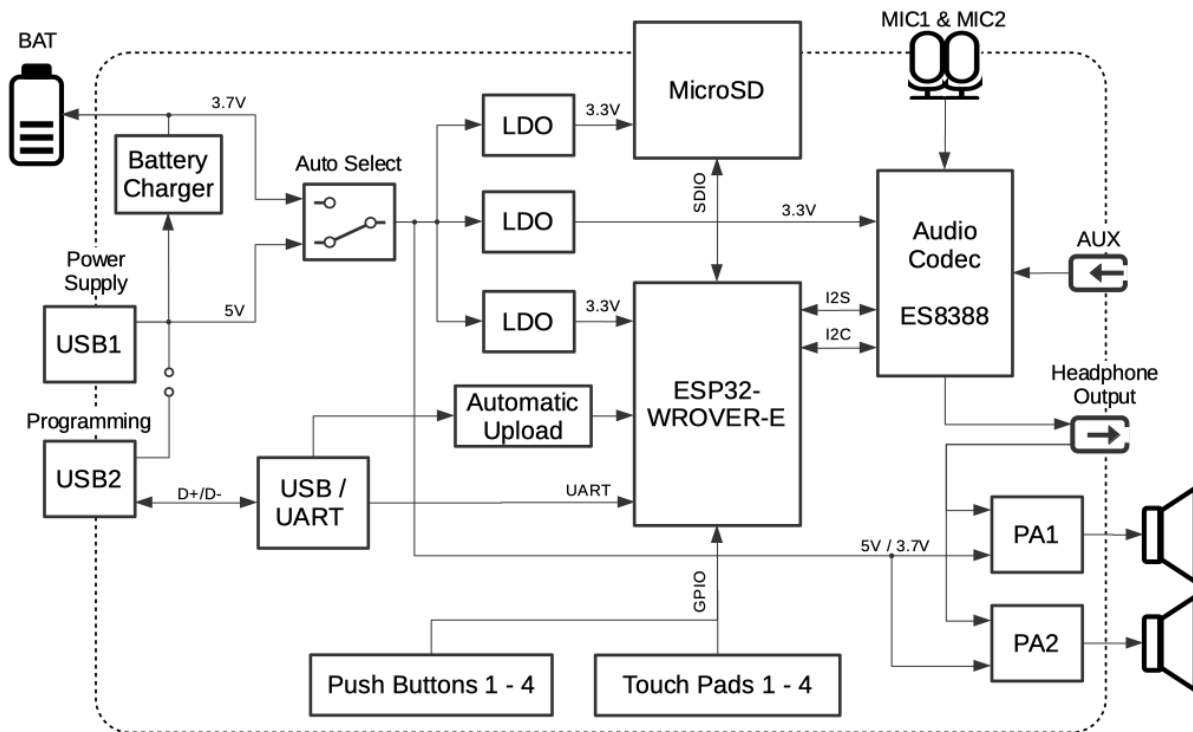


Fig. 14: ESP32-LyraT V4.3 Electrical Block Diagram

## Functional Description

The following list and figure describe key components, interfaces and controls of the ESP32-LyraT board.

**ESP32-WROVER-E Module** The ESP32-WROVER-E module contains ESP32 chip to provide Wi-Fi / Bluetooth connectivity and data processing power as well as integrates 4 MB external SPI flash and an additional 8 MB PSRAM for flexible data storage.

**Green LED** A general purpose LED controlled by the **ESP32-WROVER-E Module** to indicate certain operation states of the audio application using dedicated API.

**Function DIP Switch** Used to configure function of GPIO12 to GPIO15 pins that are shared between devices, primarily between **JTAG Header** and **MicroSD Card**. By default, the **MicroSD Card** is enabled with all switches in *OFF* position. To enable the **JTAG Header** instead, switches in positions 3, 4, 5 and 6 should be put *ON*. If **JTAG** is not used and **MicroSD Card** is operated in the one-line mode, then GPIO12 and GPIO13 may be assigned to other functions. Please refer to [ESP32 LyraT V4.3 schematic](#) for more details.

**JTAG Header** Provides access to the **JTAG** interface of **ESP32-WROVER-E Module**. It may be used for debugging, application upload, as well as implementing several other functions, e.g., [Application Level Tracing](#). See [JTAG Header / JP7](#) for pinout details. Before using **JTAG** signals to the header, **Function DIP Switch** should be enabled. Please note that when **JTAG** is in operation, **MicroSD Card** cannot be used and should be disconnected because some of **JTAG** signals are shared by both devices.

**UART Header** Serial port: provides access to the serial TX/RX signals between **ESP32-WROVER-E Module** and **USB-UART Bridge Chip**.

**I2C Header** Provides access to the I2C interface. Both **ESP32-WROVER-E Module** and **Audio Codec Chip** are connected to this interface. See [I2C Header / JP5](#) for pinout details.

**MicroSD Slot** The development board supports a MicroSD card in SPI/1-bit/4-bit modes, and can store or play audio files in the MicroSD card. Note that **JTAG** cannot be used and should be disconnected by setting **Function DIP Switch** when **MicroSD Card** is in operation, because some of signals are shared by both devices.

**I2S Header** Provides access to the I2S interface. Both **ESP32-WROVER-E Module** and **Audio Codec Chip** are connected to this interface. See *I2S Header / JP4* for pinout details.

**Left Microphone** Onboard microphone connected to IN1 of the **Audio Codec Chip**.

**AUX Input** Auxiliary input socket connected to IN2 (left and right channel) of the **Audio Codec Chip**. Use a 3.5 mm stereo jack to connect to this socket.

**Headphone Output** Output socket to connect headphones with a 3.5 mm stereo jack.

**Note:** The socket may be used with mobile phone headsets and is compatible with OMPT standard headsets only. It does work with CTIA headsets. Please refer to [Phone connector \(audio\)](#) on Wikipedia.

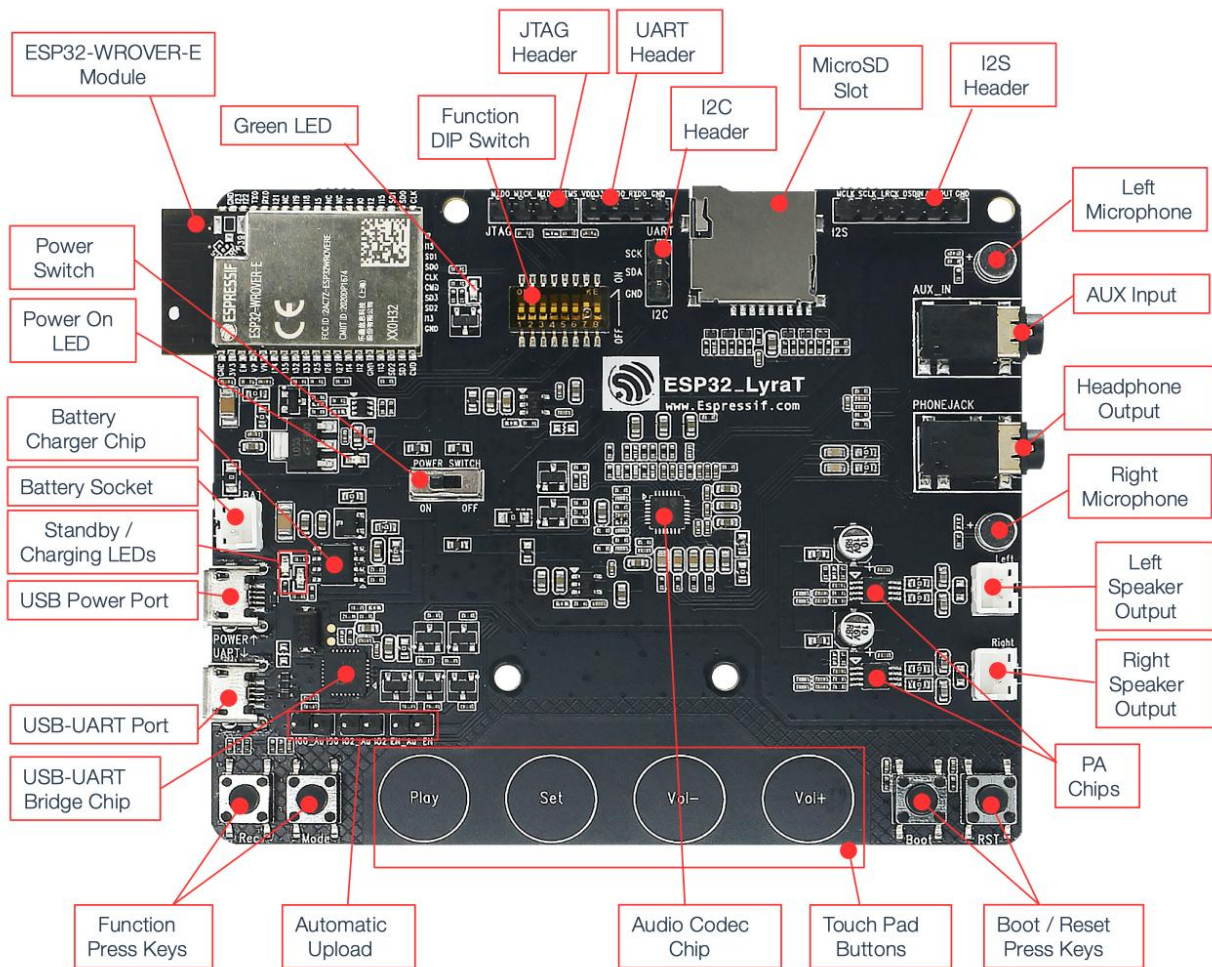


Fig. 15: ESP32-LyraT V4.3 Board Layout

**Right Microphone** Onboard microphone connected to IN1 of the **Audio Codec Chip**.

**Left Speaker Output** Output socket to connect a speaker. The 4-ohm and 3-watt speaker is recommended. The pins have a 2.00 mm / 0.08" pitch.

**Right Speaker Output** Output socket to connect a speaker. The 4-ohm and 3-watt speaker is recommended. The pins have a 2.00 mm / 0.08" pitch.

**PA Chip** A power amplifier used to amplify stereo audio signal from the **Audio Codec Chip** for driving two speakers.

**Boot/Reset Press Keys** Boot button: holding down the **Boot** button and momentarily pressing the **Reset** button to initiate the firmware download mode. Then you can download firmware through the serial port. Reset button: pressing this button alone resets the system.

**Touch Pad Buttons** Four touch pads labeled *Play*, *Sel*, *Vol+* and *Vol-*. They are routed to **ESP32-WROVER-E Module** and intended for development and testing of a UI for audio applications using dedicated API.

**Audio Codec Chip** The Audio Codec Chip, **ES8388**, is a low power stereo audio codec with a headphone amplifier. It consists of 2-channel ADC, 2-channel DAC, microphone amplifier, headphone amplifier, digital sound effects, analog mixing and gain functions. It is interfaced with **ESP32-WROVER-E Module** over I2S and I2S buses to provide audio processing in hardware independently from the audio application.

**Automatic Upload** Install three jumpers on this header to enable automatic loading of application to the ESP32. Install all jumpers together on all three headers. Remove all jumpers after upload is complete.

**Function Press Keys** Two key labeled *Rec* and *Mode*. They are routed to **ESP32-WROVER-E Module** and intended for developing and testing a UI for audio applications using dedicated API.

**USB-UART Bridge Chip** A single chip USB-UART bridge provides up to 1 Mbps transfers rate.

**USB-UART Port** Functions as the communication interface between a PC and the ESP32 module.

**USB Power Port** Provides the power supply for the board.

**Standby / Charging LEDs** The **Standby** green LED indicates that power has been applied to the **Micro USB Port**. The **Charging** red LED indicates that a battery connected to the **Battery Socket** is being charged.

**Battery Socket** Two pins socket to connect a single cell Li-ion battery.

---

**Note:** Please verify if polarity on the battery plug matches polarity of the socket as marked on the board's soldermask besides the socket.

---

**Battery Charger Chip** Constant current & constant voltage linear charger for single cell lithium-ion batteries AP5056. Used for charging of a battery connected to the **Battery Socket** over the **Micro USB Port**.

**Power On LED** Red LED indicating that **Power On Switch** is turned on.

---

**Note:** The **Power On Switch** does not affect / disconnect the Li-ion battery charging.

---

**Power Switch** Power on/off knob: toggling it to the left powers the board on; toggling it to the right powers the board off.

## Hardware Setup Options

There are a couple of options to change the hardware configuration of the ESP32-LyraT board. The options are selectable with the **Function DIP Switch**.



### Enable MicroSD Card in 1-wire Mode

DIP SW	Position
1	OFF
2	OFF
3	OFF
4	OFF
5	OFF
6	OFF
7	OFF <sup>1</sup>
8	n/a

1. **AUX Input** detection may be enabled by toggling the DIP SW 7 *ON*. Note that the **AUX Input** signal pin should not be plugged in when the system powers up. Otherwise the ESP32 may not be able to boot correctly.

In this mode:

- **JTAG** functionality is not available
- *Vol-* touch button is available for use with the API

### Enable MicroSD Card in 4-wire Mode

DIP SW	Position
1	ON
2	ON
3	OFF
4	OFF
5	OFF
6	OFF
7	OFF
8	n/a

In this mode:

- **JTAG** functionality is not available
- *Vol-* touch button is not available for use with the API
- **AUX Input** detection from the API is not available

### Enable JTAG

DIP SW	Position
1	OFF
2	OFF
3	ON
4	ON
5	ON
6	ON
7	ON
8	n/a

In this mode:

- **MicroSD Card** functionality is not available, remove the card from the slot
- *Vol-* touch button is not available for use with the API
- **AUX Input** detection from the API is not available

### Using Automatic Upload

Entering of the ESP32 into upload mode may be done in two ways:

- Manually by pressing both **Boot** and **RST** keys and then releasing first **RST** and then **Boot** key.
- Automatically by software performing the upload. The software is using **DTR** and **RTS** signals of the serial interface to control states of **EN**, **IO0** and **IO2** pins of the ESP32. This functionality is enabled by installing jumpers in three headers **JP23**, **JP24** and **JP25**. For details see [ESP32 LyraT V4.3 schematic](#). Remove all jumpers after upload is complete.

### Allocation of ESP32 Pins

Several pins ESP32 module are allocated to the on board hardware. Some of them, like GPIO0 or GPIO2, have multiple functions. Please refer to the table below or [ESP32 LyraT V4.3 schematic](#) for specific details.

GPIO Pin	Type	Function Definition
SENSOR_VP	I	Audio <b>Rec</b> (PB)
SENSOR_VN	I	Audio <b>Mode</b> (PB)
IO32	I/O	Audio <b>Set</b> (TP)
IO33	I/O	Audio <b>Play</b> (TP)
IO27	I/O	Audio <b>Vol+</b> (TP)
IO13	I/O	JTAG <b>MTCK</b> , MicroSD <b>D3</b> , Audio <b>Vol-</b> (TP)
IO14	I/O	JTAG <b>MTMS</b> , MicroSD <b>CLK</b>
IO12	I/O	JTAG <b>MTDI</b> , MicroSD <b>D2</b> , Aux signal <b>detect</b>
IO15	I/O	JTAG <b>MTDO</b> , MicroSD <b>CMD</b>
IO2	I/O	Automatic Upload, MicroSD <b>D0</b>
IO4	I/O	MicroSD <b>D1</b>
IO34	I	MicroSD insert <b>detect</b>
IO0	I/O	Automatic Upload, I2S <b>MCLK</b>
IO5	I/O	I2S <b>SCLK</b>
IO25	I/O	I2S <b>LRCK</b>
IO26	I/O	I2S <b>DSDIN</b>
IO35	I	I2S <b>ASDOUT</b>
IO19	I/O	Headphone jack insert <b>detect</b>
IO22	I/O	Green LED indicator
IO21	I/O	PA Enable output
IO18	I/O	I2C <b>SDA</b>
IO23	I/O	I2C <b>SCL</b>

- (TP) - touch pad
- (PB) - push button

## Pinout of Extension Headers

There are several pin headers available to connect external components, check the state of particular signal bus or debug operation of ESP32. Note that some signals are shared, see section *Allocation of ESP32 Pins* for details.

### UART Header / JP2

	Header Pin
1	3.3V
2	TX
3	RX
4	GND

### I2S Header / JP4

	I2C Header Pin	ESP32 Pin
1	MCLK	GPIO
2	SCLK	GPIO5
1	LRCK	GPIO25
2	DSDIN	GPIO26
3	ASDOUT	GPIO35
3	GND	GND

### I2C Header / JP5

	I2C Header Pin	ESP32 Pin
1	SCL	GPIO23
2	SDA	GPIO18
3	GND	GND

### JTAG Header / JP7

	ESP32 Pin	JTAG Signal
1	MTDO / GPIO15	TDO
2	MTCK / GPIO13	TCK
3	MTDI / GPIO12	TDI
4	MTMS / GPIO14	TMS

## Notes of Power Distribution

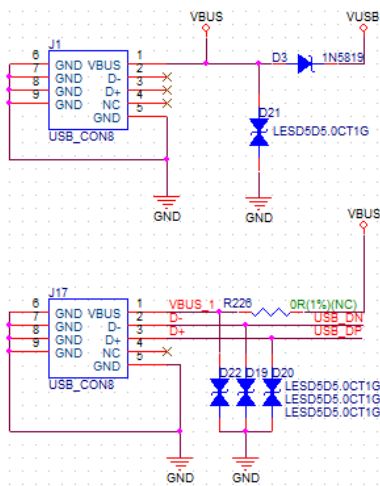
The board features quite extensive power distribution system. It provides independent power supplies to all critical components. This should reduce noise in the audio signal from digital components and improve overall performance of the components.

## Power Supply Separation

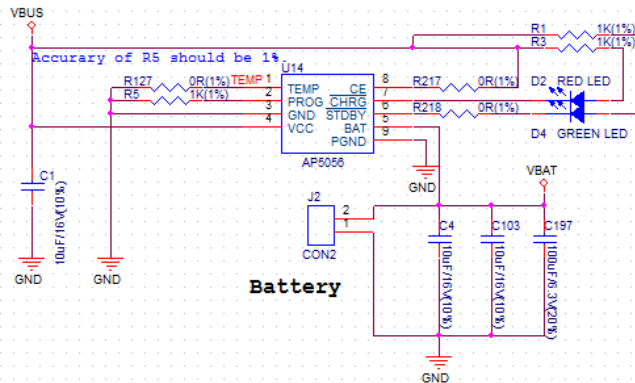
The main power supply is 5V and provided by a USB. The secondary power supply is 3.7V and provided by an optional battery. The USB power itself is fed with a dedicated cable, separate from a USB cable used for an application upload. To further reduce noise from the USB, the battery may be used instead of the USB.

### Power System:

USB<->UART:



Charge Circuit:



Power Switch:

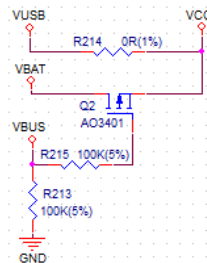


Fig. 16: ESP32 LyraT V4.3 - Power Supply Separation

### Three Dedicated LDOs

#### ESP32 Module

To provide enough current for the ESP32, the development board adopts LD1117S33CTR LDO capable to supply the maximum output current of 800mA.

#### Module Power Supply:

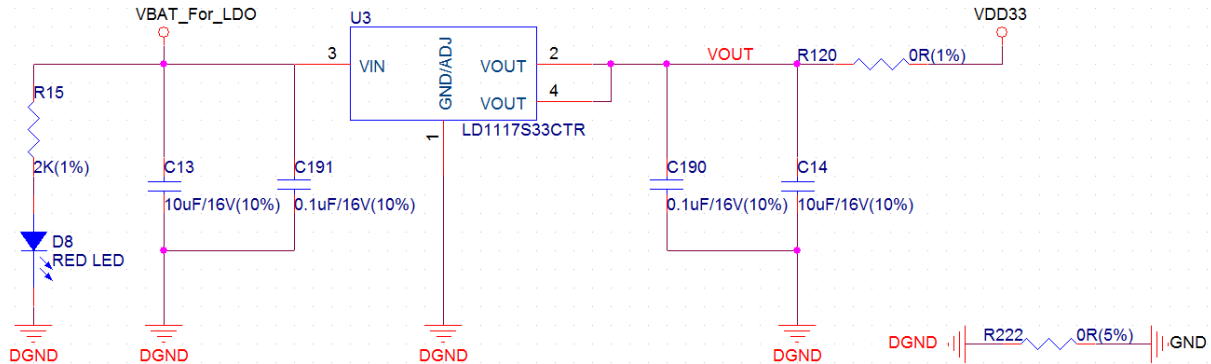


Fig. 17: ESP32 Lyrat V4.3 - Dedicated LDO for the ESP32 Module

#### MicroSD Card and Audio Codec

Two separate LDOs are provided for the MicroSD Card and the Audio Codec. Both circuits have similar design that includes an inductor and double decoupling capacitors on both the input and output of the LDO.

#### SDIO Power Supply:

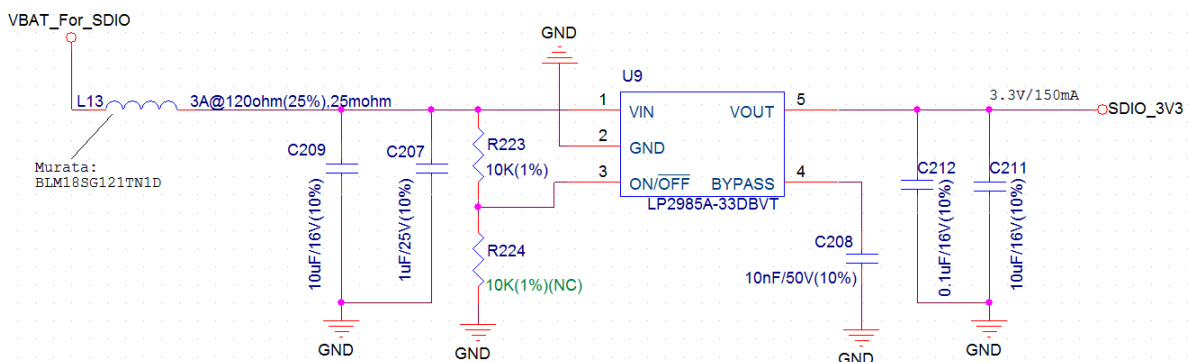


Fig. 18: ESP32 Lyrat V4.3 - Dedicated LDO for the MicroSD Card

### Separate Power Feed for the PAs

The audio amplifier unit features two NS4150 that require a large power supply for driving external speakers with the maximum output power of 3W. The power is supplied directly to both PAs from the battery or the USB. The development board adds a set of LC circuits at the front of the PA power supply, where L uses 1.5A magnetic beads and C uses 10uF aluminum electrolytic capacitors, to effectively filter out power crosstalk.

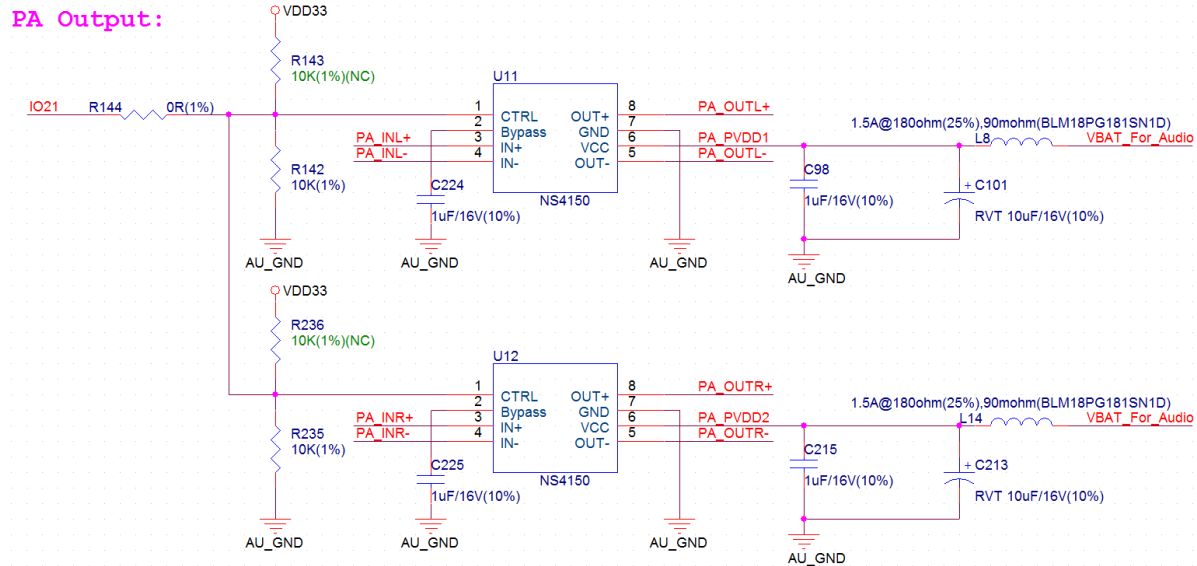


Fig. 19: ESP32 LyraT V4.3 - Power Supply for the PAs

### Selecting of the Audio Output

The development board uses two mono Class D amplifier ICs, model number NS4150 with maximum output power of 3W and operating voltage from 3.0V to 5.25V.

The audio input source is the digital-to-analog converter (DAC) output of the ES8388. Audio output supports two external speakers.

An optional audio output is a pair of headphones feed from the same DACs as the amplifier ICs.

To switch between using headphones and speakers, the board provides a digital input signal to detect when a headphone jack is inserted and a digital output signal to enable or disable the amplifier ICs. In other words selection between speakers and headphones is under software control instead of using mechanical contacts that would disconnect speakers once a headphone jack is inserted.

## Other Versions of LyraT

- [ESP32-LyraT V4.2 Getting Started Guide](#)
- [ESP32-LyraT V4 Getting Started Guide](#)

## Related Documents

- [ESP32 LyraT V4.3 schematic \(PDF\)](#)
- [ESP32-LyraT V4.3 Getting Started Guide](#)
- [ESP32 Datasheet \(PDF\)](#)
- [ESP32-WROVER-E Datasheet \(PDF\)](#)
- [JTAG Debugging](#)

### 3.4.5 ESP32-LyraT V4.2 Getting Started Guide

This guide provides users with functional descriptions, configuration options for ESP32-LyraT V4.2 audio development board, as well as how to get started with the ESP32-LyraT board.

The ESP32-LyraT development board is a hardware platform designed for the dual-core ESP32 audio applications, e.g., Wi-Fi or BT audio speakers, speech-based remote controllers, smart-home appliances with audio functionality(ies), etc.

If you like to start using this board right now, go directly to section [Start Application Development](#).

#### What You Need

- 1 × [ESP32 LyraT V4.2 board](#)
- 2 × Speaker or headphones with a 3.5 mm jack. If you use a speaker, it is recommended to choose one no more than 3 watts, and JST PH 2.0 2-Pin plugs are needed. In case you do not have this type of plug it is also fine to use Dupont female jumper wires during development.
- 2 x Micro-USB 2.0 cables, Type A to Micro B
- 1 × PC loaded with Windows, Linux or Mac OS

#### Overview

The ESP32-LyraT V4.2 is an audio development board produced by [Espressif](#) built around ESP32. It is intended for audio applications, by providing hardware for audio processing and additional RAM on top of what is already onboard of the ESP32 chip. The specific hardware includes:

- **ESP32-WROVER Module\***
- **Audio Codec Chip**
- Dual **Microphones** on board
- **Headphone** input
- **2 x 3-watt Speaker** output
- Dual **Auxiliary Input**
- **MicroSD Card** slot (1 line or 4 lines)

- **Six buttons** (2 physical buttons and 4 touch buttons)
- **JTAG** header
- Integrated **USB-UART Bridge Chip**
- Li-ion **Battery-Charge Management**

The superscript \* indicates that the product is in EOL status. The same applies below.

The block diagram below presents main components of the ESP32-LyraT and interconnections between components.

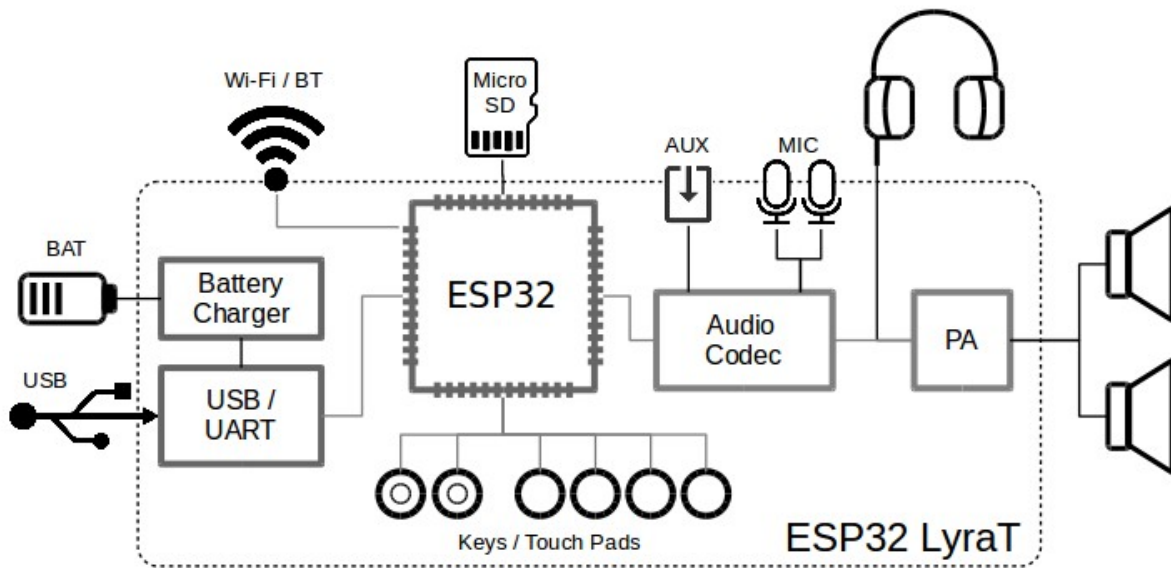


Fig. 20: ESP32-LyraT Block Diagram

## Functional Description

The following list and figure describe key components, interfaces and controls of the ESP32-LyraT board.

**ESP32-WROVER Module\*** The ESP32-WROVER module\* contains ESP32 chip to provide Wi-Fi / BT connectivity and data processing power as well as integrates 32 Mbit SPI flash and 32 Mbit PSRAM for flexible data storage.

**Green and Red LEDs** Two general purpose LEDs controlled by **ESP32-WROVER Module\*** to indicate certain operation states of the audio application using dedicated API.

**Function DIP Switch** Used to configure function of GPIO12 to GPIO15 pins that are shared between devices, primarily between **JTAG Header** and **MicroSD Card**. By default, the **MicroSD Card** is enabled with all switches in *OFF* position. To enable the **JTAG Header** instead, switches in positions 3, 4, 5 and 6 should be put *ON*. If **JTAG** is not used and **MicroSD Card** is operated in the one-line mode, then GPIO12 and GPIO13 may be assigned to other functions. Please refer to [ESP32 LyraT V4.2 schematic](#) for more details.

**JTAG Header** Provides access to the **JTAG** interface of **ESP32-WROVER Module\***. It may be used for debugging, application upload, as well as implementing several other functions, e.g., [Application Level Tracing](#). See [JTAG Header / JP7](#) for pinout details. Before using **JTAG** signals to the header, **Function DIP Switch** should be enabled. Please note that when **JTAG** is in operation, **MicroSD Card** cannot be used and should be disconnected because some of **JTAG** signals are shared by both devices.



**UART Header** Serial port: provides access to the serial TX/RX signals between **ESP32-WROVER Module\*** and **USB-UART Bridge Chip**.

**I2C Header** Provides access to the I2C interface. Both **ESP32-WROVER Module\*** and **Audio Codec Chip** are connected to this interface. See *I2C Header / JP5* for pinout details.

**MicroSD Card** The development board supports a MicroSD card in SPI/1-bit/4-bit modes, and can store or play audio files in the MicroSD card. See *MicroSD Card / J5* for pinout details. Note that **JTAG** cannot be used and should be disconnected by setting **Function DIP Switch** when **MicroSD Card** is in operation, because some of signals are shared by both devices.

**I2S Header** Provides access to the I2S interface. Both **ESP32-WROVER Module\*** and **Audio Codec Chip** are connected to this interface. See *I2S Header / JP4* for pinout details.

**Left Microphone** Onboard microphone connected to IN1 of the **Audio Codec Chip**.

**AUX Input** Auxiliary input socket connected to IN2 (left and right channel) of the **Audio Codec Chip**. Use a 3.5 mm stereo jack to connect to this socket.

**Headphone Output** Output socket to connect headphones with a 3.5 mm stereo jack.

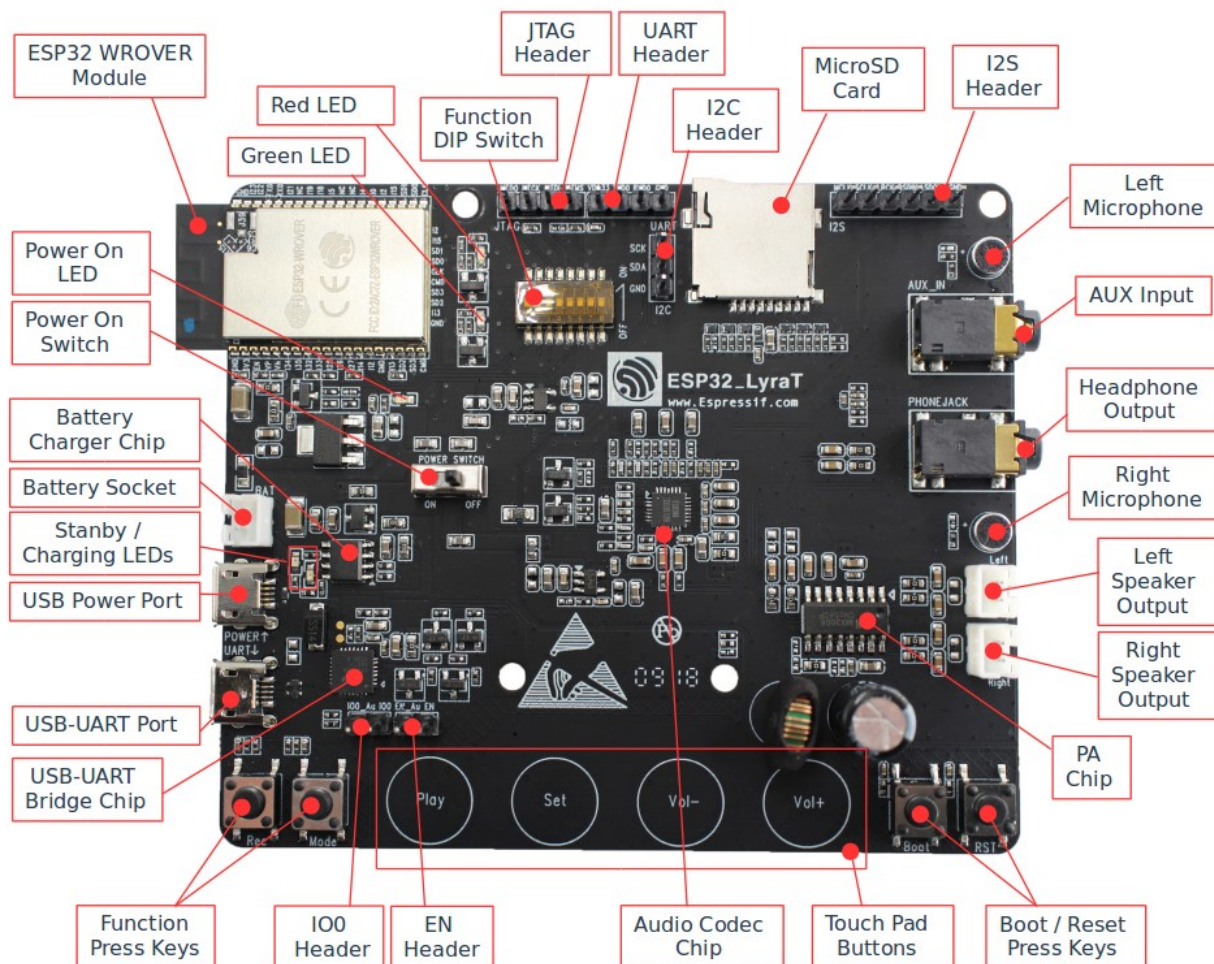


Fig. 21: ESP32-LyraT V4.2 Board Layout

**Right Microphone** Onboard microphone connected to IN1 of the **Audio Codec Chip**.

**Left Speaker Output** Output socket to connect a speaker. The 4-ohm and 3-watt speaker is recommended. The pins have a 2.00 mm / 0.08" pitch.

**Right Speaker Output** Output socket to connect a speaker. The 4-ohm and 3-watt speaker is recommended. The pins have a 2.00 mm / 0.08" pitch.

**PA Chip** A power amplifier used to amplify stereo audio signal from the **Audio Codec Chip** for driving two speakers.

**Boot/Reset Press Keys** **Boot**: holding down the **Boot** button and momentarily pressing the **Reset** button initiates the firmware upload mode. Then user can upload firmware through the serial port. **Reset**: pressing this button alone resets the system.

**Touch Pad Buttons** Four touch pads labeled *Play*, *Sel*, *Vol+* and *Vol-*. They are routed to **ESP32-WROVER Module\*** and intended for development and testing of a UI for audio applications using dedicated API.

**Audio Codec Chip** The Audio Codec Chip, **ES8388**, is a low power stereo audio codec with a headphone amplifier. It consists of 2-channel ADC, 2-channel DAC, microphone amplifier, headphone amplifier, digital sound effects, analog mixing and gain functions. It is interfaced with **ESP32-WROVER Module\*** over I2S and I2S buses to provide audio processing in hardware independently from the audio application.

**EN Header** Install a jumper on this header to enable automatic loading of application to the ESP32. Install or remove jumpers together on both IO0 and EN headers.

**IO0 Header** Install a jumper on this header to enable automatic loading of application to the ESP32. Install or remove jumpers together on both IO0 and EN headers.

**Function Press Keys** Two key labeled *Rec* and *Mode*. They are routed to **ESP32-WROVER Module\*** and intended for developing and testing a UI for audio applications using dedicated API.

**USB-UART Bridge Chip** A single chip USB-UART bridge provides up to 1 Mbps transfers rate.

**USB-UART Port** Functions as the communication interface between a PC and the ESP32 module.

**USB Power Port** Provides the power supply for the board.

**Standby / Charging LEDs** The **Standby** green LED indicates that power has been applied to the **Micro USB Port**. The **Charging** red LED indicates that a battery connected to the **Battery Socket** is being charged.

**Battery Charger Chip** Constant current & constant voltage linear charger for single cell lithium-ion batteries AP5056. Used for charging of a battery connected to the **Battery Socket** over the **Micro USB Port**.

**Power On Switch** Power on/off knob: toggling it to the left powers the board on; toggling it to the right powers the board off.

**Battery Socket** Two pins socket to connect a single cell Li-ion battery.

**Power On LED** Red LED indicating that **Power On Switch** is turned on.

---

**Note:** The **Power On Switch** does not affect / disconnect the Li-ion battery charging.

---

## Hardware Setup Options

There are a couple of options to change the hardware configuration of the ESP32-LyraT board. The options are selectable with the **Function DIP Switch**.

### Enable MicroSD Card in 1-wire Mode

DIP SW	Position
1	OFF
2	OFF
3	OFF
4	OFF
5	OFF
6	OFF
7	OFF <sup>1</sup>
8	n/a

1. **AUX Input** detection may be enabled by toggling the DIP SW 7 *ON*

In this mode:

- **JTAG** functionality is not available
- *Vol-* touch button is available for use with the API

### Enable MicroSD Card in 4-wire Mode

DIP SW	Position
1	ON
2	ON
3	OFF
4	OFF
5	OFF
6	OFF
7	OFF
8	n/a

In this mode:

- **JTAG** functionality is not available
- *Vol-* touch button is not available for use with the API
- **AUX Input** detection from the API is not available

### Enable JTAG

DIP SW	Position
1	OFF
2	OFF
3	ON
4	ON
5	ON
6	ON
7	ON
8	n/a

In this mode:

- **MicroSD Card** functionality is not available, remove the card from the slot
- *Vol-* touch button is not available for use with the API
- **AUX Input** detection from the API is not available

### Allocation of ESP32 Pins

Several pins / terminals of ESP32 modules are allocated to the on board hardware. Some of them, like GPIO0 or GPIO2, have multiple functions. Please refer to the tables below or [ESP32 LyraT V4.2 schematic](#) for specific details.

### Red / Green LEDs

	ESP32 Pin	LED Color
1	GPIO19	Red LED
2	GPIO22	Green LED

### Touch Pads

	ESP32 Pin	Touch Pad Function
1	GPIO33	Play
2	GPIO32	Set
3	GPIO13	Vol- <sup>1</sup>
4	GPIO27	Vol+

1. *Vol-* function is not available if **JTAG** is used. It is also not available for the **MicroSD Card** configured to operate in 4-wire mode.

**MicroSD Card / J5**

	ESP32 Pin	MicroSD Signal
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

**UART Header / JP2**

	Header Pin
1	3.3V
2	TX
3	RX
4	GND

**EN and IO0 Headers / JP23 and J24**

	ESP32 Pin	Header Pin
1	n/a	EN_Auto
2	EN	EN

	ESP32 Pin	Header Pin
1	n/a	IO0_Auto
2	GPIO0	IO0

**I2S Header / JP4**

	I2C Header Pin	ESP32 Pin
1	MCLK	GPIO
2	SCLK	GPIO5
1	LRCK	GPIO25
2	DSDIN	GPIO26
3	ASDOUT	GPIO35
3	GND	GND

### I2C Header / JP5

	I2C Header Pin	ESP32 Pin
1	SCL	GPIO23
2	SDA	GPIO18
3	GND	GND

### JTAG Header / JP7

	ESP32 Pin	JTAG Signal
1	MTDO / GPIO15	TDO
2	MTCK / GPIO13	TCK
3	MTDI / GPIO12	TDI
4	MTMS / GPIO14	TMS

### Function DIP Switch / JP8

	Switch OFF	Switch ON
1	GPIO12 not allocated	MicroSD Card 4-wire
2	Touch <i>Vol-</i> enabled	MicroSD Card 4-wire
3	MicroSD Card	JTAG
4	MicroSD Card	JTAG
5	MicroSD Card	JTAG
6	MicroSD Card	JTAG
7	MicroSD Card 4-wire	AUX IN detect <sup>1</sup>
8	not used	not used

1. The **AUX Input** signal pin should not be plugged in when the system powers up. Otherwise the ESP32 may not be able to boot correctly.

## Start Application Development

Before powering up the ESP32-LyraT, please make sure that the board has been received in good condition with no obvious signs of damage.

### Initial Setup

Prepare the board for loading of the first sample application:

1. Install jumpers on **IO0** and **EN** headers to enable automatic application upload. If there are no jumpers then upload may be triggered using **Boot** / **RST** buttons.
2. Connect speakers to the **Right** and **Left Speaker Output**. Connecting headphones to the **Headphone Output** is an option.
3. Plug in the Micro-USB cables to the PC and to **both USB ports** of the ESP32 LyraT.
4. The **Standby LED** (green) should turn on. Assuming that a battery is not connected, the **Charging LED** will blink every couple of seconds.

5. Toggle left the **Power On Switch**.
6. The red **Power On LED** should turn on.

If this is what you see on the LEDs, the board should be ready for application upload. Now prepare the PC by loading and configuring development tools what is discussed in the next section.

## Develop Applications

Once the board is initially set up and checked, you can start preparing the development tools. The Section *Installation Step by Step* will walk you through the following steps:

- **Set up ESP-IDF** to get a common development framework for the ESP32 (and ESP32-S2) chips in C language;
- **Get ESP-ADF** to install the API specific to audio applications;
- **Set up env** to make the framework aware of the audio specific API;
- **Start a Project** that will provide a sample audio application for the board;
- **Connect Your Device** to prepare the application for loading;
- **Build the Project** to finally run the application and play some music.

## Related Documents

- [ESP32 LyraT V4.2 schematic \(PDF\)](#)
- [ESP32 Datasheet \(PDF\)](#)
- [JTAG Debugging](#)
- [ESP32-LyraT V4 Getting Started Guide](#)

### 3.4.6 ESP32-LyraT V4 Getting Started Guide

This guide provide users with functional descriptions, configuration options for ESP32-LyraT V4 audio development board, as well as how to get started with ESP32-LyraT board.

The ESP32-LyraT development board is a hardware platform specifically designed for the dual-core ESP32 audio applications, e.g., Wi-Fi or BT audio speakers, speech-based remote controllers, smart-home appliances with audio functionality(ies), etc.

If you like to start using this board right now, go directly to section *Start Application Development*.

## What You Need

- 1 × *ESP32-LyraT V4 board*
- 2 × Speaker or headphones with a 3.5 mm jack. If you use a speaker, it is recommended to choose one no more than 3 watts, and JST PH 2.0 2-Pin plugs are needed. In case you do not have this type of plug it is also fine to use Dupont female jumper wires during development.
- 1 × Micro USB 2.0 Cable, Type A to Micro B
- 1 × PC loaded with Windows, Linux or Mac OS

Overview

The ESP32-LyraT V4 is an audio development board produced by [Espressif](#) built around ESP32. It is intended for audio applications, by providing hardware for audio processing and additional RAM on top of what is already onboard of the ESP32 chip. The specific hardware includes:

- **ESP32-WROVER Module\***
- **Audio Codec Chip**
- Dual **Microphones** on board
- **Headphone** input
- **2 x 3 Watt Speaker** output
- Dual **Auxiliary Input**
- **MicroSD Card** slot (1 line or 4 lines)
- **6 buttons** (2 physical buttons and 4 touch buttons)
- **JTAG** header
- Integrated **USB-UART Bridge Chip**
- Li-ion **Battery-Charge Management**

The superscript \* indicates that the product is in EOL status. The same applies below.

The block diagram below presents main components of the ESP32-LyraT and interconnections between components.

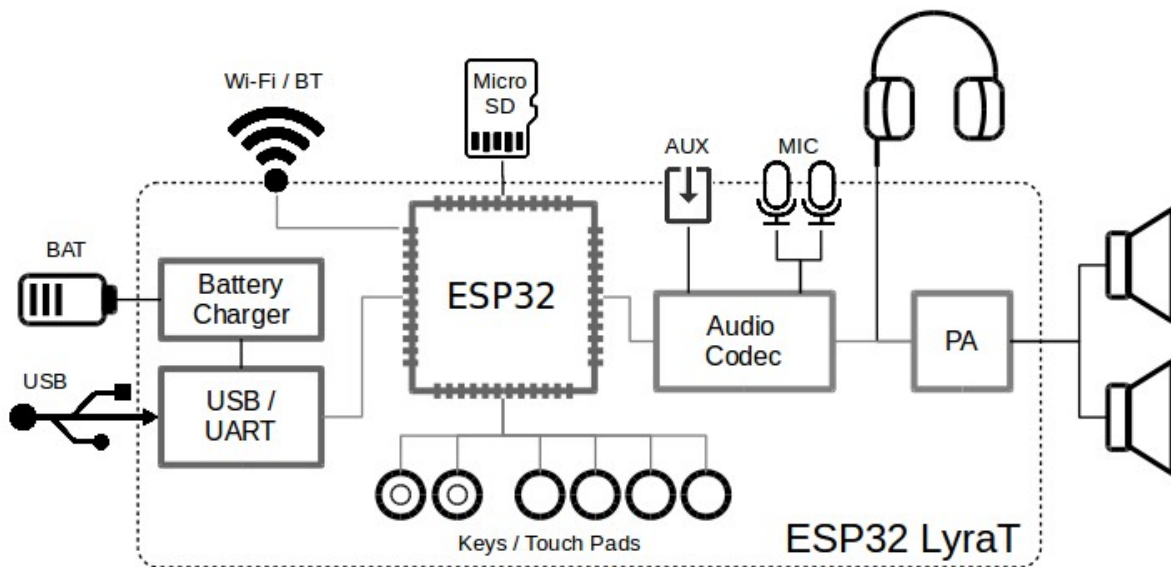


Fig. 22: ESP32-LyraT block diagram



## Functional Description

The following list and figure below describe key components, interfaces and controls of the ESP32-LyraT board.

**ESP32-WROVER Module\*** The ESP32-WROVER\* module contains ESP32 chip to provide Wi-Fi / BT connectivity and data processing power as well as integrates 32 Mbit SPI flash and 32 Mbit PSRAM for flexible data storage.

**Green and Red LEDs** Two general purpose LEDs controlled by **ESP32-WROVER Module\*** to indicate certain operation states of the audio application using dedicated API.

**Function DIP Switch** Used to configure function of GPIO12 to GPIO15 pins that are shared between devices, primarily between **JTAG Header** and **MicroSD Card**. By default **MicroSD Card** is enabled with all switches in *OFF* position. To enable **JTAG Header** instead, switches in positions 3, 4, 5 and 6 should be put *ON*. If **JTAG** is not used and **MicroSD Card** is operated in one-line mode, then GPIO12 and GPIO13 may be assigned to other functions. Please refer to [ESP32 LyraT V4 schematic](#) for more details.

**JTAG Header** Provides access to the **JTAG** interface of **ESP32-WROVER Module\***. May be used for debugging, application upload, as well as implementing several other functions, e.g., [Application Level Tracing](#). See [JTAG Header / JP7](#) for pinout details. Before using **JTAG** signals to the header, **Function DIP Switch** should be enabled. Please note that when **JTAG** is in operation, **MicroSD Card** cannot be used and should be disconnected because some of JTAG signals are shared by both devices.

**UART Header** Serial port provides access to the serial TX/RX signals between **ESP32-WROVER Module\*** and **USB-UART Bridge Chip**.

**I2C Header** Provides access to the I2C interface. Both **ESP32-WROVER Module\*** and **Audio Codec Chip** are connected to this interface. See [I2C Header / JP5](#) for pinout details.

**MicroSD Card** The development board supports a MicroSD card in SPI/1-bit/4-bit modes, and can store or play audio files in the MicroSD card. See [MicroSD Card / J5](#) for pinout details. Note that **JTAG** cannot be used and should be disconnected by setting **Function DIP Switch** when **MicroSD Card** is in operation, because some of the signals are shared by both devices.

**I2S Header** Provides access to the I2S interface. Both **ESP32-WROVER Module\*** and **Audio Codec Chip** are connected to this interface. See [I2S Header / JP4](#) for pinout details.

**Left Microphone** Onboard microphone connected to IN1 of the **Audio Codec Chip**.

**AUX Input** Auxiliary input socket connected to IN2 (left and right channels) of the **Audio Codec Chip**. Use a 3.5 mm stereo jack to connect to this socket.

**Headphone Output** Output socket to connect headphones with a 3.5 mm stereo jack.

**Right Microphone** Onboard microphone connected to IN1 of the **Audio Codec Chip**.

**Left Speaker Output** Output socket to connect a speaker. The 4-ohm and 3-watt speaker is recommended. The pins have a 2.00 mm / 0.08" pitch.

**Right Speaker Output** Output socket to connect a speaker. The 4-ohm and 3-watt speaker is recommended. The pins have a 2.00 mm / 0.08" pitch.

**PA Chip** A power amplifier used to amplify stereo audio signal from the **Audio Codec Chip** for driving two speakers.

**Boot/Reset Press Keys** Boot: holding down the **Boot** button and momentarily pressing the **Reset** button initiates the firmware upload mode. Then user can upload firmware through the serial port. Reset: pressing this button alone resets the system.

**Touch Pad Buttons** Four touch pads labeled *Play*, *Sel*, *Vol+* and *Vol-*. They are routed to **ESP32-WROVER Module\*** and intended for development and testing of a UI for audio applications using dedicated API.

**Audio Codec Chip** The Audio Codec Chip, **ES8388**, is a low-power stereo audio codec with headphone amplifier. It consists of 2-channel ADC, 2-channel DAC, microphone amplifier, headphone amplifier, digital sound effects,

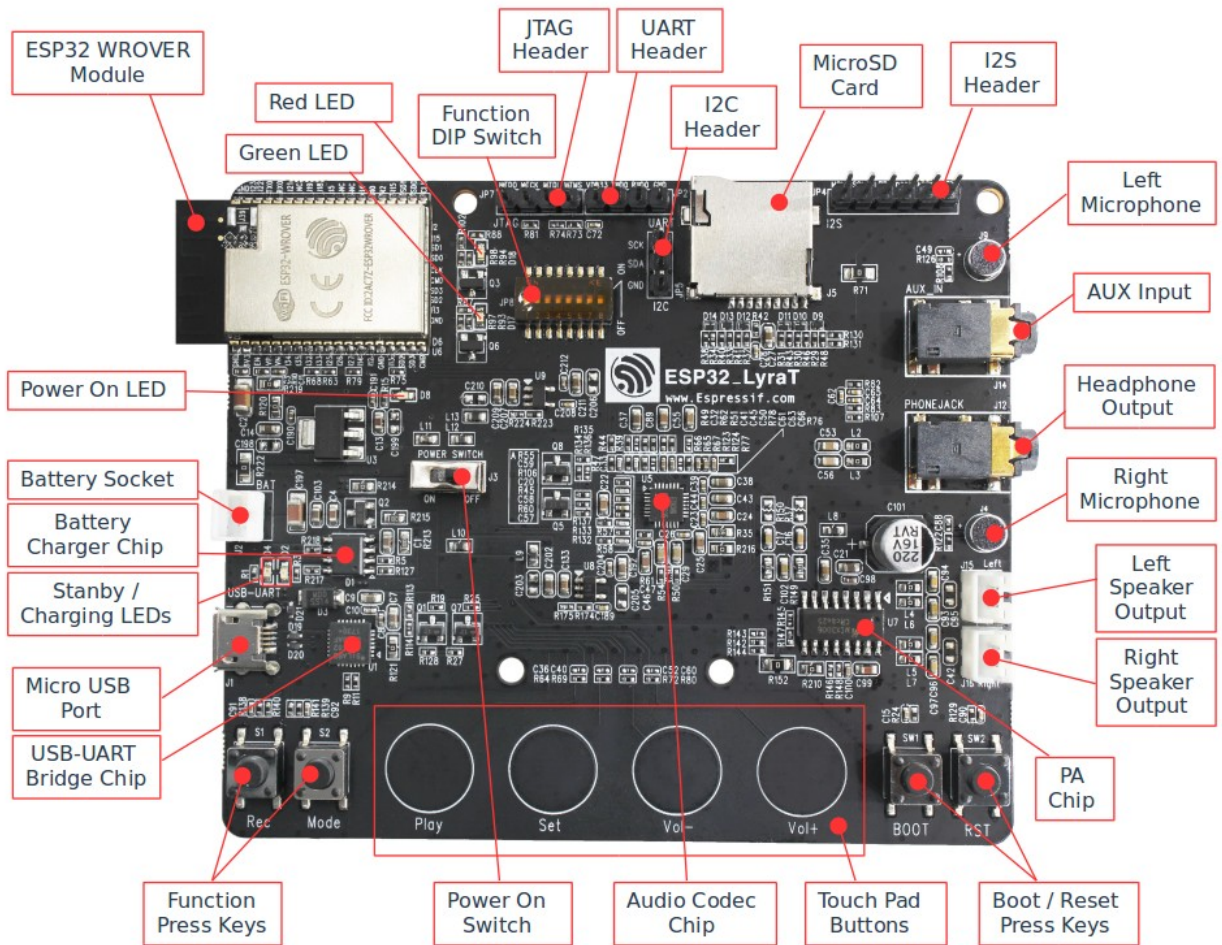


Fig. 23: ESP32 Lyrat V4 board layout

analog mixing and gain functions. It is interfaced with **ESP32-WROVER Module**<sup>\*</sup> over I2S and I2S buses to provide audio processing in hardware independently from the audio application.

**Function Press Keys** Two key labeled *Rec* and *Mode*. They are routed to **ESP32-WROVER Module**<sup>\*</sup> and intended for developing and testing a UI for audio applications using dedicated API.

**USB-UART Bridge Chip** A single chip USB-UART bridge provides up to 1 Mbps transfer rate.

**Micro USB Port** USB interface. It functions as the power supply for the board and the communication interface between a PC and the ESP32 module.

**Standby / Charging LEDs** The **Standby** green LED indicates that power has been applied to the **Micro USB Port**. The **Charging** red LED indicates that a battery connected to the **Battery Socket** is being charged.

**Battery Charger Chip** Constant current & constant voltage linear charger for single cell lithium-ion batteries AP5056. Used for charging of a battery connected to the **Battery Socket** over the **Micro USB Port**.

**Power On Switch** Power on/off knob: toggling it to the left powers the board on; toggling it to the right powers the board off.

**Battery Socket** Two pins socket to connect a single cell Li-ion battery.

**Power On LED** Red LED indicating that **Power On Switch** is turned on.

---

**Note:** The **Power On Switch** does not affect / disconnect the Li-ion battery charging.

---

## Hardware Setup Options

There are couple of options to change the hardware configuration of the ESP32-LyraT board. The options are selectable with the **Function DIP Switch**.

### Enable MicroSD Card in 1-wire Mode

DIP SW	Position
1	OFF
2	OFF
3	OFF
4	OFF
5	OFF
6	OFF
7	OFF <sup>1</sup>
8	n/a

1. **AUX Input** detection may be enabled by toggling the DIP SW 7 *ON*

In this mode:

- **JTAG** functionality is not available
- *Vol-* touch button is available for use with the API

### Enable MicroSD Card in 4-wire Mode

DIP SW	Position
1	ON
2	ON
3	OFF
4	OFF
5	OFF
6	OFF
7	OFF
8	n/a

In this mode:

- **JTAG** functionality is not available
- *Vol-* touch button is not available for use with the API
- **AUX Input** detection from the API is not available

### Enable JTAG

DIP SW	Position
1	OFF
2	OFF
3	ON
4	ON
5	ON
6	ON
7	ON
8	n/a

In this mode:

- **MicroSD Card** functionality is not available, remove the card from the slot
- *Vol-* touch button is not available for use with the API
- **AUX Input** detection from the API is not available

### Allocation of ESP32 Pins

Several pins / terminals of ESP32 modules are allocated to the onboard hardware. Some of them, like GPIO0 or GPIO2, have multiple functions. Please refer to tables below or [ESP32 Lyrat V4 schematic](#) for specific details.

### Red / Green LEDs

	ESP32 Pin	LED Color
1	GPIO19	Red LED
2	GPIO22	Green LED

### Touch Pads

	ESP32 Pin	Touch Pad Function
1	GPIO33	Play
2	GPIO32	Set
3	GPIO13	Vol- <sup>1</sup>
4	GPIO27	Vol+

1. Vol- function is not available if **JTAG** is used. It is also not available for the **MicroSD Card** configured to operate in 4-wire mode.

### MicroSD Card / J5

	ESP32 Pin	MicroSD Signal
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

### UART Header / JP2

	Header Pin
1	3.3V
2	TX
3	RX
4	GND

### I2S Header / JP4

	I2C Header Pin	ESP32 Pin
1	MCLK	GPIO
2	SCLK	GPIO5
1	LRCK	GPIO25
2	DSDIN	GPIO26
3	ASDOUT	GPIO35
3	GND	GND

### I2C Header / JP5

	I2C Header Pin	ESP32 Pin
1	SCL	GPIO23
2	SDA	GPIO18
3	GND	GND

### JTAG Header / JP7

	ESP32 Pin	JTAG Signal
1	MTDO / GPIO15	TDO
2	MTCK / GPIO13	TCK
3	MTDI / GPIO12	TDI
4	MTMS / GPIO14	TMS

### Function DIP Switch / JP8

	Switch OFF	Switch ON
1	GPIO12 not allocated	MicroSD Card 4-wire
2	Touch <i>Vol-</i> enabled	MicroSD Card 4-wire
3	MicroSD Card	JTAG
4	MicroSD Card	JTAG
5	MicroSD Card	JTAG
6	MicroSD Card	JTAG
7	MicroSD Card 4-wire	AUX IN detect <sup>1</sup>
8	not used	not used

1. The **AUX Input** signal pin should not be plugged in when the system powers up. Otherwise the ESP32 may not be able to boot correctly.

## Start Application Development

Before powering up the ESP32-LyraT, please make sure that the board has been received in good condition with no obvious signs of damage.

### Initial Setup

Prepare the board for loading of the first sample application:

1. Connect speakers to the **Right** and **Left Speaker Output**. Optionally connect headphones to the **Headphone Output**.
2. Plug in the Micro-USB cable to the PC and to the **Micro USB Port** of the ESP32-LyraT.
3. The **Standby LED** (green) should turn on. Assuming that a battery is not connected, the **Charging LED** will momentarily blink every couple of seconds.
4. Toggle left the **Power On Switch**.

5. The red **Power On LED** should turn on.

If this is what you see on the LEDs, the board should be ready for application upload. Now prepare the PC by loading and configuring development tools what is discussed in the next section.

## Develop Applications

Once the board is initially set up and checked, you can start preparing the development tools. The Section *Installation Step by Step* will walk you through the following steps:

- **Set up ESP-IDF** to get a common development framework for the ESP32 (and ESP32-S2) chips in C language;
- **Get ESP-ADF** to install the API specific to audio applications;
- **Set up env** to make the framework aware of the audio specific API;
- **Start a Project** that will provide a sample audio application for the board;
- **Connect Your Device** to prepare the application for loading;
- **Build the Project** to finally run the application and play some music.

## Related Documents

- [ESP32 LyraT V4 schematic \(PDF\)](#)
- [ESP32 Datasheet \(PDF\)](#)
- [JTAG Debugging](#)

### 3.4.7 ESP32-LyraTD-MSC V2.2 Getting Started Guide

This guide provides users with functional descriptions, configuration options for ESP32-LyraTD-MSC V2.2 audio development board, as well as how to get started with the ESP32-LyraTD-MSC board.

The ESP32-LyraTD-MSC is a hardware platform designed for smart speakers and AI applications. It supports Acoustic Echo Cancellation (AEC), Automatic Speech Recognition (ASR), Wake-up Interrupt and Voice Interaction.

#### What You Need

- 1 × *ESP32-LyraTD-MSC V2.2 board*
- 2 x Speaker or headphones with a 3.5 mm jack. If you use a speaker, it is recommended to choose one no more than 3 watts, and JST PH 2.0 2-Pin plugs are needed. In case you do not have this type of plug it is also fine to use Dupont female jumper wires during development.
- 2 x Micro-USB 2.0 cables, Type A to Micro B
- 1 × PC loaded with Windows, Linux or Mac OS

If you like to start using this board right now, go directly to section *Start Application Development*.

### Overview

The ESP32-LyraTD-MSC V2.2 is an audio development board produced by [Espressif](#) built around ESP32. It is intended for smart speakers and AI applications, by providing hardware for digital signal processing, microphone array and additional RAM on top of what is already onboard of the ESP32 chip.

This audio development board consists of two parts: the upper board (B), which provides a three-microphone array, function keys and LED lights; and the lower board (A), which integrates ESP32-WROVER-E, a MicroSemi Digital Signal Processing (DSP) chip, and a power management module.



Fig. 24: ESP32-LyraTD-MSC Side View

The specific hardware includes:

- ESP32-WROVER-E Module
- DSP (Digital Signal Processing) chip
- Three digital Microphones that support far-field voice pick-up
- 2 x 3-watt Speaker output
- Headphone output
- MicroSD Card slot (1 line or 4 lines)
- Individually controlled Twelve LEDs distributed in a circle on the board's edge
- Six Function Buttons that may be assigned user functions
- Several interface ports: I2S, I2C, SPI and JTAG
- Integrated USB-UART Bridge Chip
- Li-ion Battery-Charge Management

The block diagram below presents main components of the ESP32-LyraTD-MSC and interconnections between components.



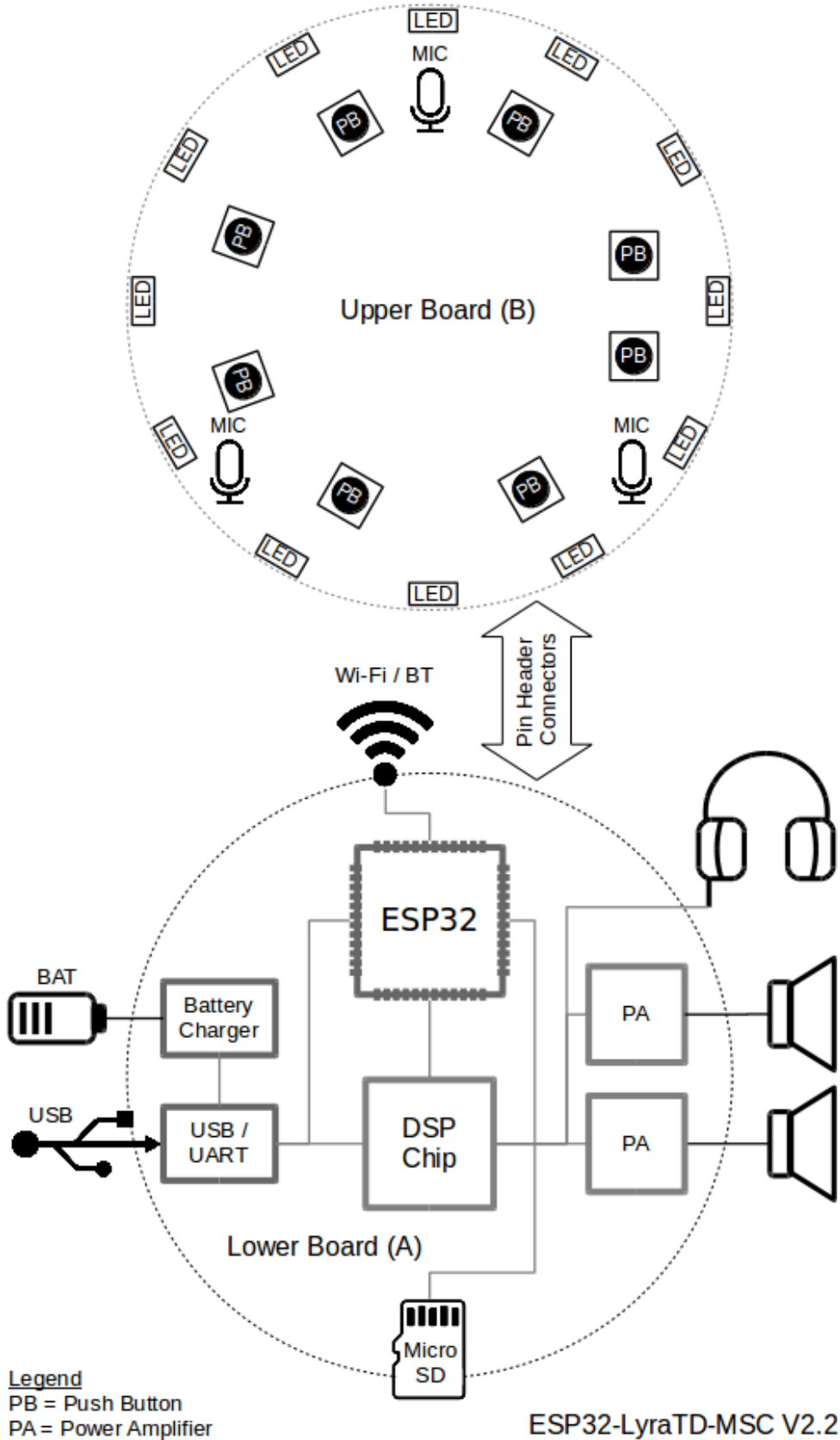


Fig. 25: ESP32-LyraTD-MSC Block Diagram

## Components

The following list and figure describe key components, interfaces and controls of the ESP32-LyraTD-MSC used in this guide. This covers just what is needed now. For additional details please refer to schematics provided in *Related Documents*.

**ESP32-WROVER-E Module** The ESP32-WROVER-E module contains ESP32 chip to provide Wi-Fi / Bluetooth connectivity and data processing power as well as integrates 4 MB external SPI flash and an additional 8 MB PSRAM for flexible data storage.

**DSP Chip** The Digital Signal Processing chip [ZL38063](#) is used for Automatic Speech Recognition (ASR) applications. It captures audio data from an external microphone array and outputs audio signals through its Digital-to-Analog-Converter (DAC) port.

**Headphone Output** Output socket to connect headphones with a 3.5 mm stereo jack.

---

**Note:** The socket may be used with mobile phone headsets and is compatible with OMPT standard headsets only. It does not work with CTIA headsets. Please refer to [Phone connector \(audio\)](#) on Wikipedia.

---

**Left Speaker Output** Output socket to connect a speaker. The 4-ohm and 3-watt speaker is recommended. The pins have a 2.00 mm / 0.08” pitch.

**Right Speaker Output** Output socket to connect a speaker. The 4-ohm and 3-watt speaker is recommended. The pins have a 2.00 mm / 0.08” pitch.

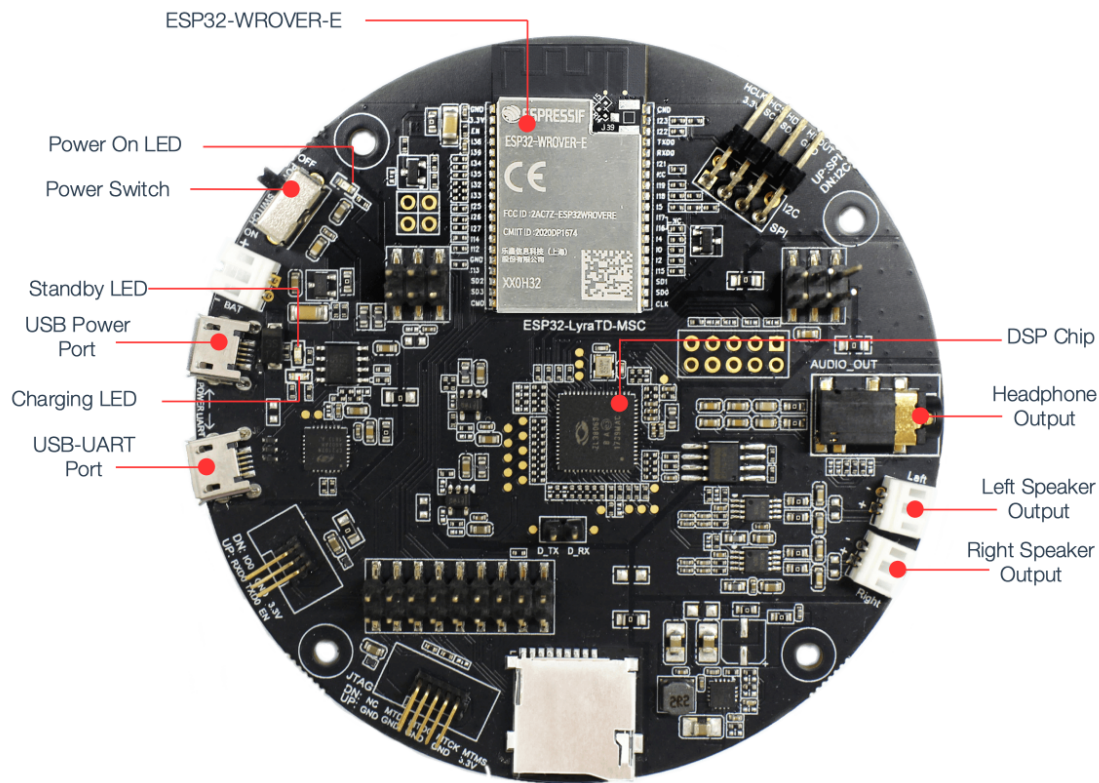


Fig. 26: ESP32-LyraTD-MSC V2.2 Lower Board (A) Components

**USB-UART Port** Functions as the communication interface between a PC and the ESP32-WROVER-E module.

**USB Power Port** Provides the power supply for the board.

**Standby / Charging LEDs** The **Standby** green LED indicates that power has been applied to the **Micro USB Port**.  
The **Charging** red LED indicates that a battery connected to the **Battery Socket** is being charged.

**Power Switch** Power on/off knob: toggling it right powers the board on; otherwise powers the board off.

**Power On LED** Red LED indicating that **Power Switch** is turned on.

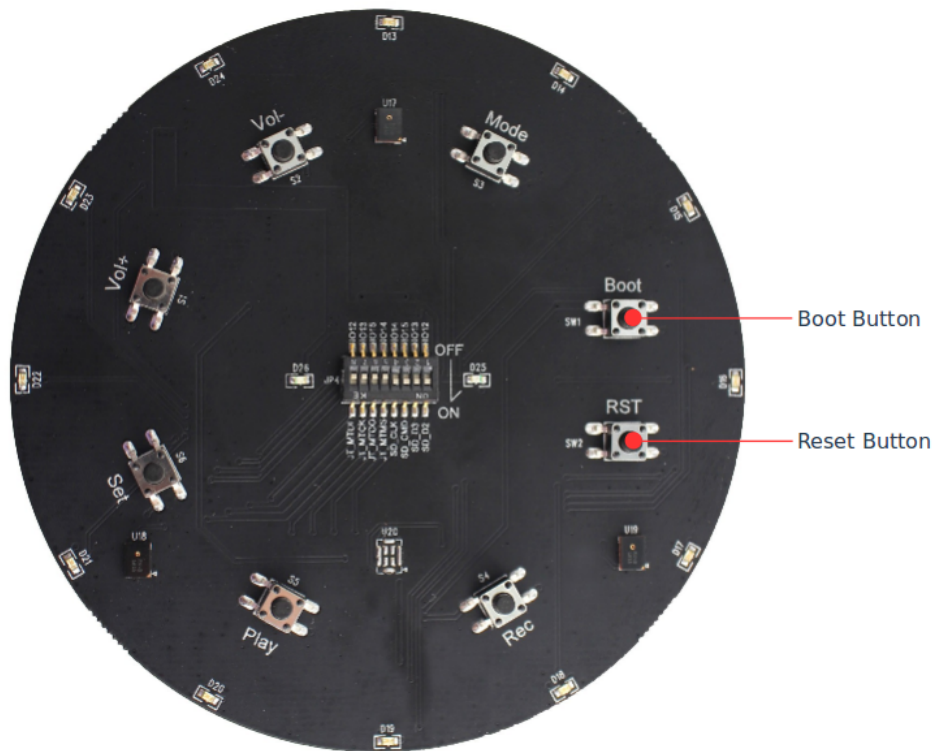


Fig. 27: ESP32-LyraTD-MSC V2.2 Upper Board (B) Components

**Boot/Reset Buttons** Boot: holding down the **Boot** button and momentarily pressing the **Reset** button initiates the firmware upload mode. Then user can upload firmware through the serial port.

Reset: pressing this button alone resets the system.

### Start Application Development

Before powering up the ESP32-LyraTD-MSC, please make sure that the board has been received in good condition with no obvious signs of damage. Both the lower A and the upper B board of the ESP32-LyraTD-MSC should be firmly connected together.

### Initial Setup

Prepare the board for loading of the first sample application:

1. Connect speakers to the **Right** and **Left Speaker Output**. Connecting headphones to the **Headphone Output** is an option.
2. Plug in the Micro-USB cables to the PC and to **both USB ports** of the ESP32-LyraTD-MS-C.
3. The **Standby LED** (green) should turn on. Assuming that a battery is not connected, the **Charging LED** (red) will blink every couple of seconds.
4. Toggle right the **Power Switch**.
5. The red **Power On LED** should turn on.

If this is what you see on the LEDs, the board should be ready for application upload. Now prepare the PC by loading and configuring development tools what is discussed in the next section.

### Develop Applications

Once the board is initially set up and checked, you can start preparing the development tools. The Section *Installation Step by Step* will walk you through the following steps:

- **Set up ESP-IDF** to get a common development framework for the ESP32 (and ESP32-S2) chips in C language;
- **Get ESP-ADF** to install the API specific to audio applications;
- **Set up env** to make the framework aware of the audio specific API;
- **Start a Project** that will provide a sample audio application for the board;
- **Connect Your Device** to prepare the application for loading;
- **Build the Project** to finally run the application and play some music.

### Revision History

- Changed the integrated module to ESP32-WROVER-E from ESP32-WROVER-B.

### Other Boards from LyraT Family

- *ESP32-LyraT V4.3 Getting Started Guide*
- *ESP32-LyraT-Mini V1.2 Getting Started Guide*

### Related Documents

- ESP32-LyraTD-MS-C V2.2 Schematic Lower Board (A) (PDF)
- ESP32-LyraTD-MS-C V2.2 Schematic Upper Board (B) (PDF)
- ESP32 Datasheet (PDF)
- ESP32-WROVER-E Datasheet (PDF)

### 3.4.8 ESP32-Korvo-DU1906

This user guide provides information on ESP32-Korvo-DU1906.

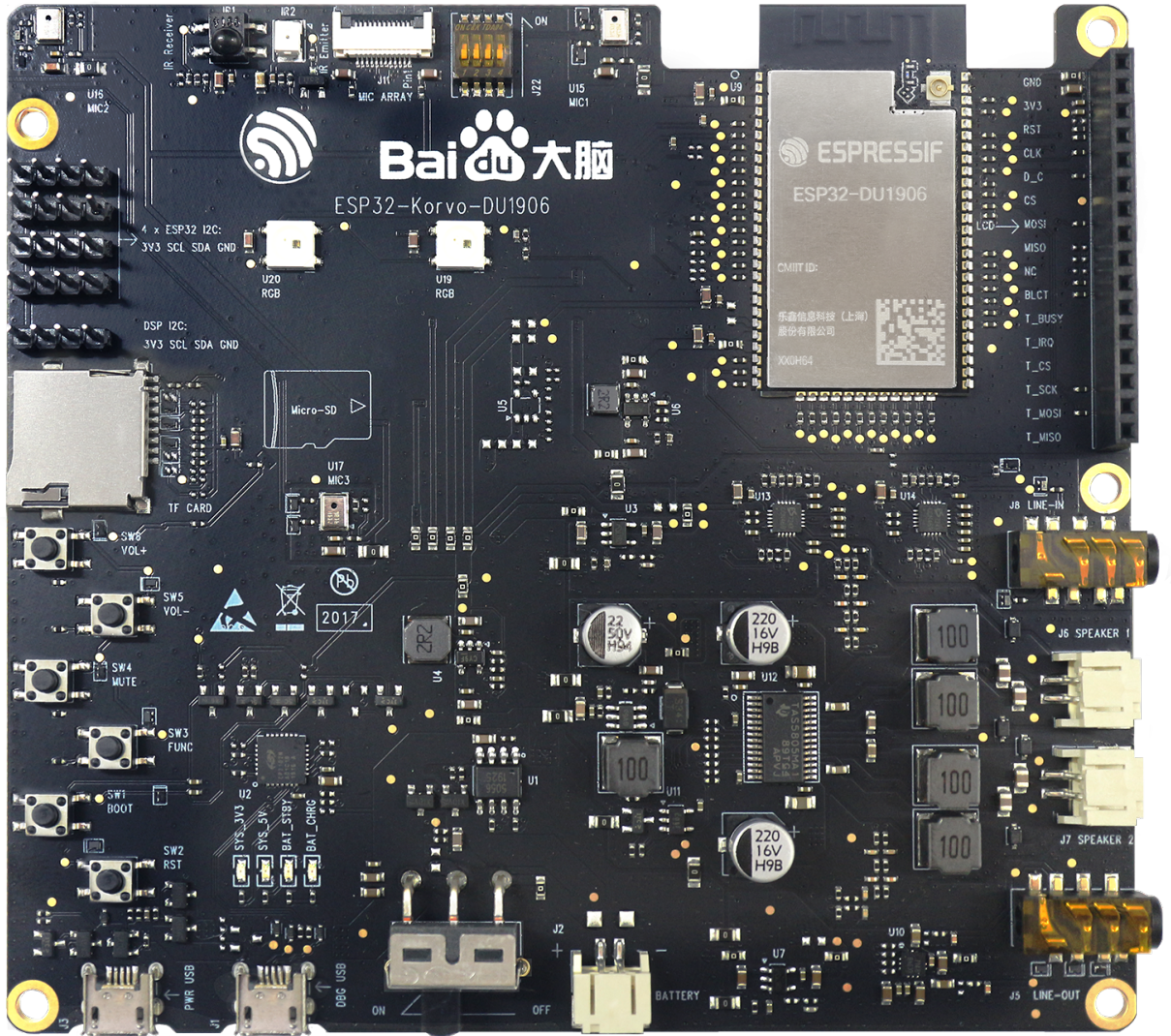


Fig. 28: ESP32-Korvo-DU1906 (click to enlarge)

The document consists of the following major sections:

- *Getting Started*: Provides an overview of the ESP32-Korvo-DU1906 and hardware/software setup instructions to get started.
- *Start Application Development*: Provides more detailed information about the ESP32-Korvo-DU1906's application development process.
- *Related Documents*: Gives links to related documentaiton.

### Getting Started

The core component of ESP32-Korvo-DU1906 includes an ESP32-DU1906 Bluetooth/Wi-Fi audio module, which is able to realize noise reduction, acoustic echo cancellation (AEC), beam formation and detection. ESP32-Korvo-DU1906 integrates power management, Bluetooth/Wi-Fi audio module, Coder-Decoder (CODEC), power amplifier, and etc., supports various functions such as:

- ADC
- Microphone array
- SD card
- Functional buttons
- Indicator lights
- Battery constant-current/constant-voltage linear power management chip
- USB-to-UART conversion
- LCD connector

### What You Need

- 1 x PC loaded with Windows, Mac OS and Linux (Linux Operating System is recommended)
- 1 x ESP32-Korvo-DU1906
- 2 x Micro USB cables
- 2 x Speaker (4 Ohm, 2.5 W)

### Overview

The biggest advantage of this development board is the audio chip – ESP32-DU1906, the core processing module, is an powerful AI module integrating Wi-Fi+Bluetooth+Bluetooth Low Energy RF and voice/speech signal processing functions, which can be used in various fields. By providing the leading end-to-end audio solutions, high-efficient integrated AI service capabilities, and an on-device AIOT platform which integrates ends and devices, this board is able to largely reduce the threshold for AI access.

DU1906 is a voice processing flagship chip launched by Baidu. This chip has a highly integrated algorithm, which can solve the industrial needs of real-time processing of far-field array signals, and high-precision voice wake-up and real-time monitoring with ultra-low error occurs simultaneously on this single one chip.

The block diagram below presents main components of the ESP32-Korvo-DU1906 and interconnections between components.

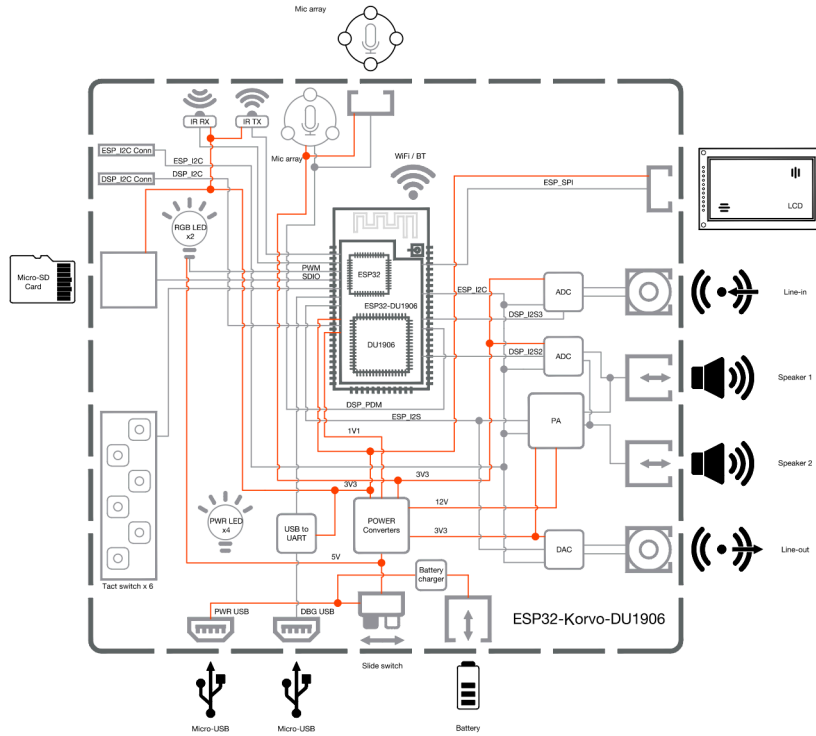


Fig. 29: ESP32-Korvo-DU1906 Block Diagram (click to enlarge)

### Description of Components

The following list and figure describe key components, interfaces and controls of the ESP32-Korvo-DU1906 used in this guide. This covers just what is needed now. For additional details please refer to schematics provided in Related Documents.

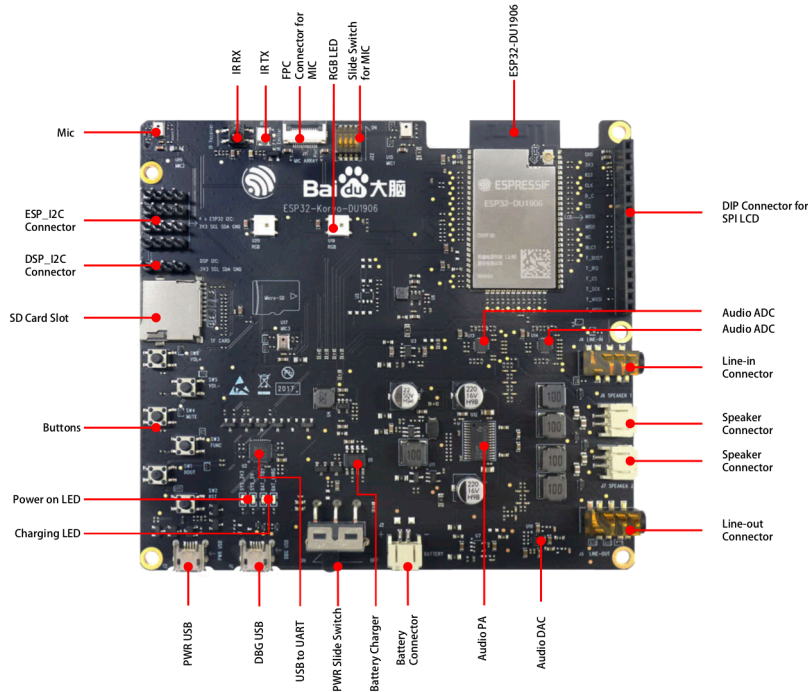


Fig. 30: ESP32-Korvo-DU1906 Components (click to enlarge)

Key Component	Description
ESP32-DU1906	This is a powerful, general-purpose, Wi-Fi/Bluetooth audio communication module, targeting a wide variety of applications ranging from low-power sensor networks to the most demanding tasks, such as voice encoding/decoding, music streaming and running voice assistant client SDK.
DIP Connector for SPI LCD	ESP32-Korvo-DU1906 has a 2.54 mm pitch connector to connect SPI LCD.
Audio ADC (Audio Analog-to-Digital Converter)	ESP32-Korvo-DU1906 includes two ES7243 high-efficiency ADCs, with one for the collection of Audio PA outputs, and another for the collection of Line-in outputs. Both ADCs can be used for AEC.
Line-in/out Connector (Earphone Jacks)	The two earphone jacks are used to connect to Line-out outputs of Audio DAC and Line-in inputs of Audio ADC respectively. When a device is plugged in the Line-out earphone jack of Audio DAC, the Audio PA on the ESP32-Korvo-DU1906 will be turned off.
Speaker Connector	Output sockets to connect two 4-ohm speakers to provide stereo sound via digital Audio PA.
Audio DAC (Audio Digital-to-Analog Converter)	ES7148 Stereo DAC is able to convert digital signals into analog audio outputs.
Audio PA (Digital Audio Power Amplifier)	TAS5805M is a high-efficiency stereo closed-loop D-type amplifier with low power dissipation and rich sound. It can convert audio digital signals into high-power analog audio outputs and transmit them to external speakers for playback. When the Line-out earphone jack of the audio DAC plugged into the device, the Digital Audio PA on the ESP32-Korvo-DU1906 will be turned off.
Battery Connector	Connect a battery.
Battery Charger	AP5056, a constant-current/constant-voltage linear power management chip, can be used for charging management to a single lithium-ion battery.
PWR Slide Switch	Power switch for the board, turn on/off the power supply.
USB to UART	CP2102N supports USB-to-UART conversion for easy download and debugging of software.
DBG USB (Debugging USB)	USB communication between PC and ESP32-DU1906 module.
PWR USB (Power supply USB)	Provide power supply for the whole system. It is recommended that it is connected to an at least 5 V / 2 A power adapter for sufficient current supply.
Charging LEDs	Indicating battery state. When a battery is connected, BAT_CHRG LED will turn red (indicating the battery is charging), then BAT_STBY LED will turn green (indicating the battery is fully charged). If no battery is connected, both LEDs will be off.



## Start Application Development

Before powering up the ESP32-Korvo-DU1906, please make sure that the board has been received in good condition with no obvious signs of damage.

### Initial Setup

Prepare the board for loading of the first sample application:

1. Connect 4-ohm speakers to the two **Speaker Connectors**. Connecting earphones to the **Line-out Connector** is an option.
2. Plug in the Micro-USB cables to the PC and to both **USB connectors** of the ESP32-Korvo-DU1906.
3. Assuming that a battery is connected, the **Charging LED** (red) will keep the lights on.
4. Toggle left the **PWR Slide Switch**.
5. The red **Power On LED** should turn on.

If this is what you see on the LEDs, the board should be ready for application upload. Now prepare the PC by loading and configuring development tools what is discussed in the next section.

### Develop Applications

Once the board is initially set up and checked, you can start preparing the development tools. The Section *Development Boards* will walk you through the following steps:

- **Set up ESP-IDF** to get a common development framework for the ESP32 (and ESP32-S2) chips in C language;
- **Get ESP-ADF** to install the API specific to audio applications;
- **Set up env** to make the framework aware of the audio specific API;
- **Start a Project** that will provide a sample audio application for the board;
- **Connect Your Device** to prepare the application for loading;
- **Build the Project** to finally run the application and play some music.

### Other Related Boards

- *ESP32-LyraT V4.3 Getting Started Guide*
- *ESP32-LyraT-Mini V1.2 Getting Started Guide*
- *ESP32-LyraTD-MSC V2.2 Getting Started Guide*

### Contents and Packaging

#### Retail orders

If you order one or several samples, each board will come in a plastic package or other package chosen by the retailer.

For retail orders, please go to <https://www.espressif.com/zh-hans/products/devkits/esp32-korvo-du1906>.

#### Related Documents

- [ESP32-Korvo-DU1906 Schematic \(PDF\)](#)
- [ESP32 Datasheet \(PDF\)](#)

### 3.4.9 ESP32-S3-Korvo-2 V3.0

This user guide will help you get started with ESP32-S3-Korvo-2 V3.0 and will also provide more in-depth information.

The ESP32-S3-Korvo-2 is a multimedia development board based on the ESP32-S3 chip. It is equipped with a two-microphone array which is suitable for voice recognition and near/far-field voice wake-up applications. The board integrates multiple peripherals such as LCD, camera, and microSD card. It also supports JPEG video stream processing. With all of its outstanding features, the board is an ideal choice for the development of low-cost and low-power network-connected audio and video products.

This board mainly consists of the following parts:

- Main board: ESP32-S3-Korvo-2
- LCD extension board: [ESP32-S3-Korvo-2-LCD](#)
- Camera

This document is mostly dedicated to the main board. For detailed information on other parts, click the links above.

The document consists of the following sections:

- *Getting started*: Overview of the board and hardware/software setup instructions to get started.
- *Hardware Reference*: More detailed information about the board's hardware.
- *Hardware Revision Details*: Hardware revision history, known issues, and links to user guides for previous versions (if any) of the board.
- *Related Documents*: Links to related documentation.

#### Getting Started

This section provides a brief introduction of ESP32-S3-Korvo-2 V3.0, instructions on how to do the initial hardware setup and how to flash firmware onto it.



Fig. 31: ESP32-S3-Korvo-2 V3.0 with ESP32-S3-WROOM-1 module

Description of Components

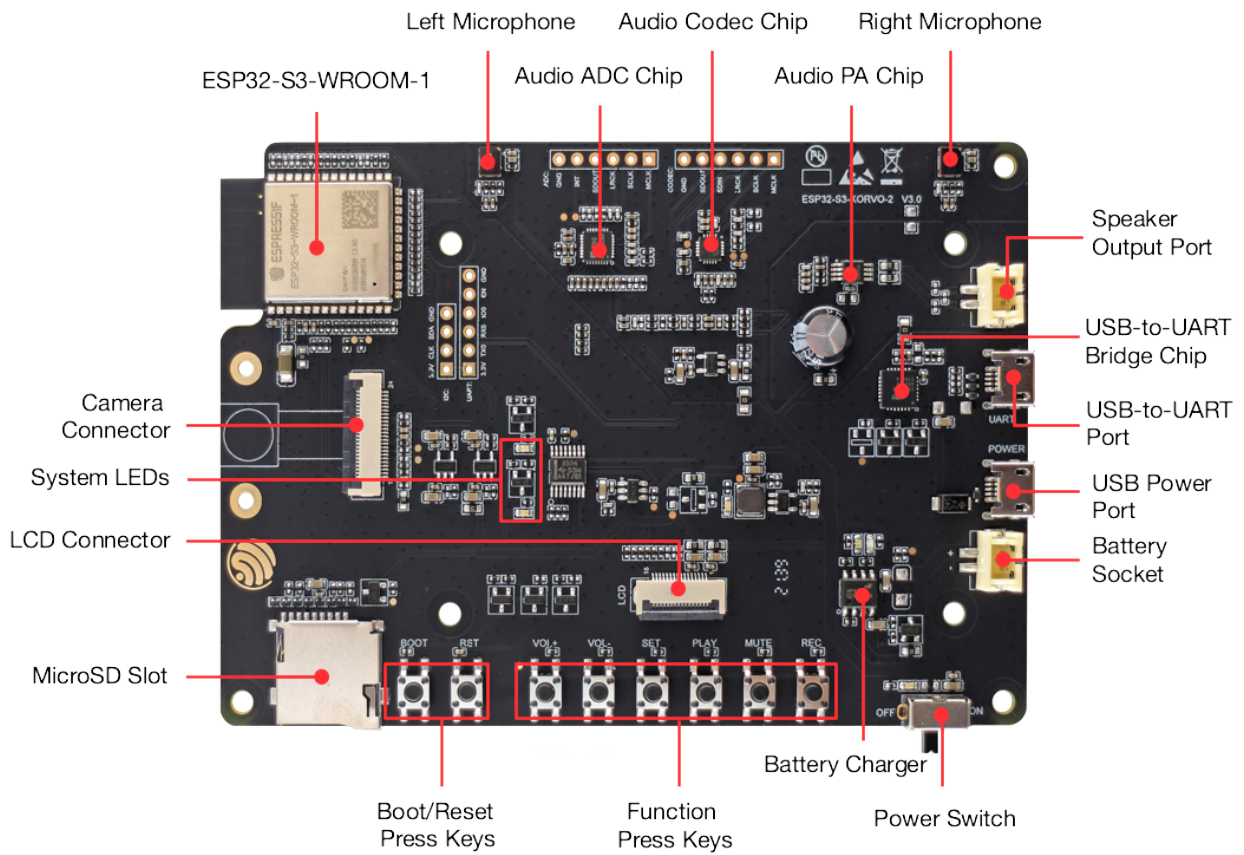


Fig. 32: ESP32-S3-Korvo-2 V3.0 (click to enlarge)

The key components of the board are described in a clockwise direction.

Key Component	Description
ESP32-S3-WROOM-1 Module	The ESP32-S3-WROOM-1 is a powerful, generic Wi-Fi + Bluetooth LE MCU module that is built around the ESP32-S3 series of SoCs. On top of a rich set of peripherals, the acceleration for neural network computing and signal processing workloads provided by the SoC make the modules an ideal choice for a wide variety of application scenarios related to AI and Artificial Intelligence of Things (AIoT), such as wake word detection, speech commands recognition, face detection and recognition, smart home, smart appliances, smart control panel, smart speaker, etc.
Left Microphone	Onboard microphone connected to ADC.
Audio ADC Chip	ES7210 is a high-performance, low-power 4-channel audio analog-to-digital converter for microphone array applications. It is very suitable for music and speech applications. In addition, ES7210 can also be used to collect acoustic echo cancellation (AEC) echo reference signals.
Audio Codec Chip	The audio codec chip, ES8311, is a low-power mono audio codec. It consists of 1-channel ADC, 1-channel DAC, low noise pre-amplifier, headphone driver, digital sound effects, analog mixing, and gain functions. It is interfaced with ESP32-S3-WROOM-1 module over I2S and I2C buses to provide audio processing in hardware independently from the audio application.
Audio PA Chip	NS4150 is an EMI, 3 W mono Class D audio power amplifier, amplifying audio signals from audio codec chips to drive speakers.
Right Microphone	Onboard microphone connected to ADC.
Speaker Output Port	Output socket to connect a speaker. The 4-ohm and 3-watt speaker is recommended. The pins have a 2.00 mm/0.08" pitch.
USB-to-UART Bridge Chip	A single chip USB-UART bridge CP2102N provides up to 3 Mbps transfer rates for software download and debugging.
USB-to-UART Port	Functions as the communication interface between a PC and the ESP32-S3-WROOM-1 module.
USB Power Port	Provides power to the board. It is recommended to use at least 5V/2A power adapter to ensure a stable power supply.
Battery Socket	Two pins socket to connect a single cell Li-ion battery.
Power Switch	Power on/off knob: toggling it down powers the board on; toggling it up powers the board off.
Battery Charger	AP5056 is a constant current and constant voltage linear charger for single cell lithium-ion batteries. Used for charging of a battery connected to the Battery Socket over the Micro USB Port.
Function Press Keys	Six press keys labeled REC, MUTE, PLAY, SET, VOL- and VOL+. They are routed to ESP32-S3-WROOM-1 module and intended for development and testing of a UI for audio applications using a dedicated API.
Boot/Reset Press Keys	<p>Boot: holding down the Boot key and momentarily pressing the Reset key initiates the firmware upload mode. Then you can upload firmware through the serial port.</p> <p>Reset: pressing this button alone resets the system.</p>
MicroSD Slot	The development board supports a microSD card in 1-bit mode, and can store or play audio files in the microSD card.
LCD Connector	A FPC connector with 0.5 mm pitch to connect to the LCD extension board.
System LEDs	Two general-purpose LEDs (green and red) controlled by ESP32-S3-WROOM-1 module to indicate certain operation states of the audio application using dedicated API.
Camera Connector	An external camera module that can be connected to the development board with the connector to transmit images.

### Start Application Development

Before powering up your board, please make sure that it is in good condition with no obvious signs of damage.

### Required Hardware

- 1 x ESP32-S3-Korvo-2 V3.0
- 1 x Speaker
- 2 x USB 2.0 cable (Standard-A to Micro-B)
- 1 x Computer running Windows, Linux, or macOS

---

**Note:** Be sure to use an appropriate USB cable. Some cables are for charging only and do not provide the needed data lines nor work for programming the boards.

---

### Optional Hardware

- 1 x MicroSD card
- 1 x Li-ion battery

---

**Note:** Be sure to use a Li-ion battery that has a built-in protection circuit.

---

### Hardware Setup

1. Connect the speaker to the **Speaker Output**.
2. Plug in the USB cables to the PC and to both USB ports of the board.
3. The standby LED (green) should turn on. Assuming that a battery is not connected, the charging LED (red) will blink every couple of seconds.
4. Toggle the **Power Switch**.
5. The red Power On LED should turn on.

### Software Setup

Please proceed to *Get Started*, where Section *Installation Step by Step* will quickly help you set up the development environment and then flash an [application example](#) onto your board.

## Contents and Packaging

The main board and its accessories can be ordered separately. The accessories include:

- LCD extension board: ESP32-S3-Korvo-2-LCD
- Camera
- Connectors:
  - 20-pin FPC cable
- Fasteners:
  - Copper standoffs (x8)
  - Screws (x4)

## Retail Orders

If you order a few samples, each board comes in an individual package in either antistatic bag or any packaging depending on your retailer.

For retail orders, please go to <https://www.espressif.com/en/company/contact/buy-a-sample>.

## Wholesale Orders

If you order in bulk, the boards come in large cardboard boxes.

For wholesale orders, please go to <https://www.espressif.com/en/contact-us/sales-questions>.

## Hardware Reference

### Block Diagram

The block diagram below shows the components of ESP32-S3-Korvo-2 V3.0 and their interconnections.

### Notes on Power Distribution

#### Power Supply over USB and from Battery

The main power supply is 5 V and provided by a USB. The secondary power supply is 3.7 V and provided by an optional battery. The USB power itself is fed with a dedicated cable, separating from a USB cable used for an application upload. To further reduce noise from the USB, the battery may be used instead of the USB.

As shown in the figure below, if the USB power supply and battery power supply are connected at the same time with a high VBUS, an off-state Q14, and an automatic cut-off VBAT, the USB becomes the power supply for the system.

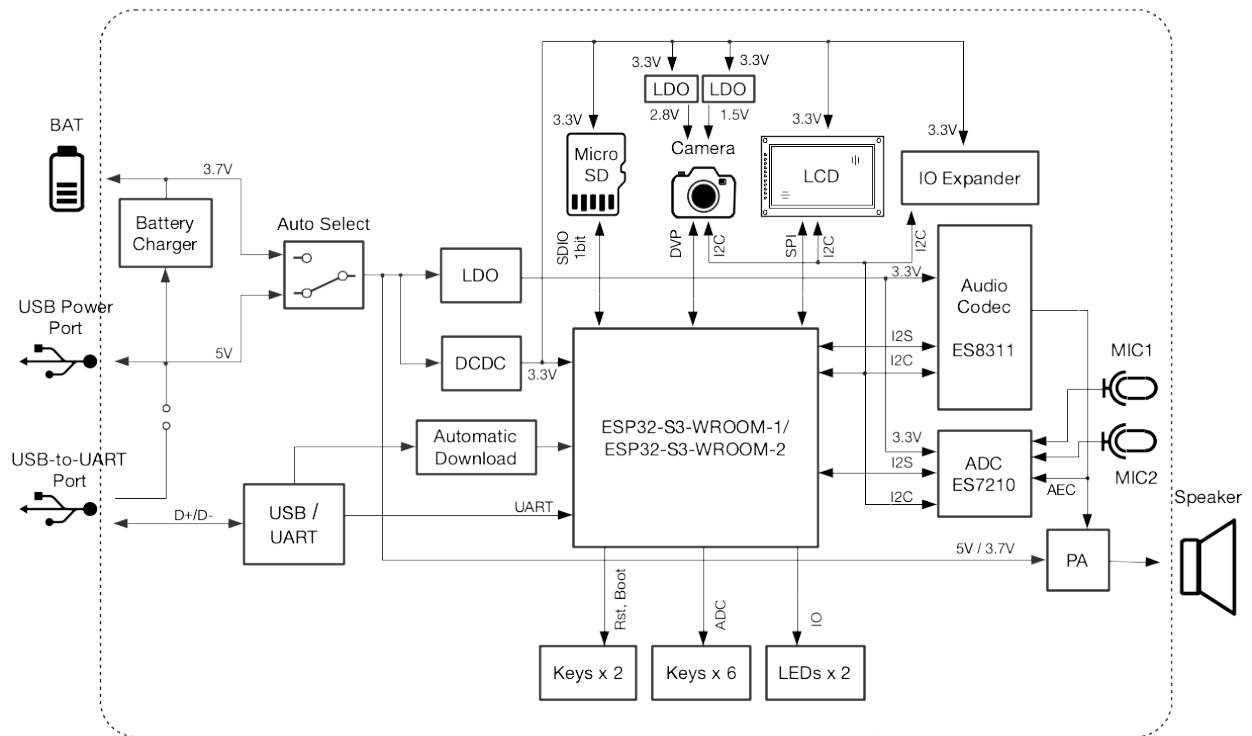


Fig. 33: ESP32-S3-Korvo-2 V3.0 Electrical Block Diagram

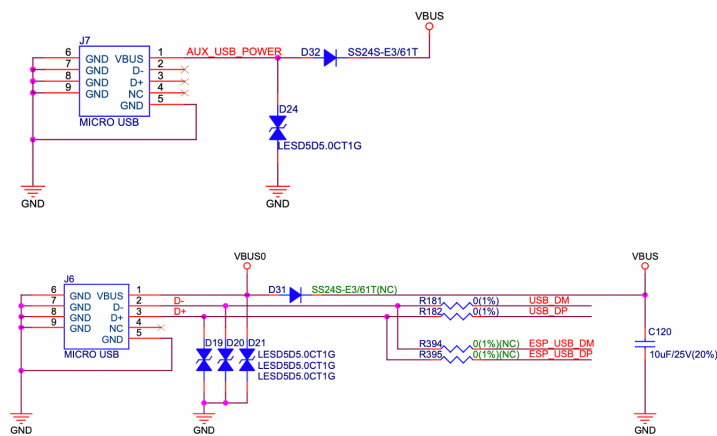
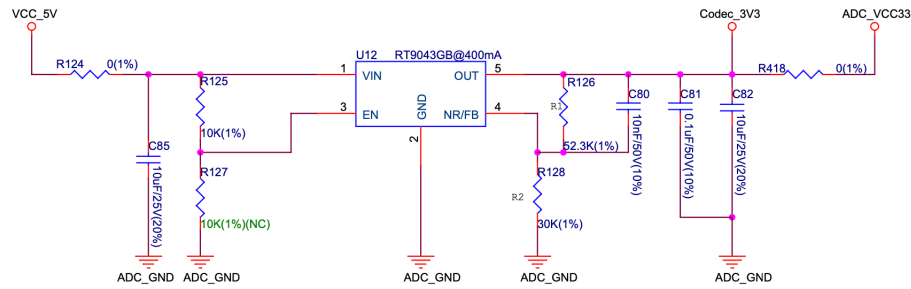


Fig. 34: ESP32-S3-Korvo-2 V3.0 - Dedicated USB Power Supply Socket





Power for Audio:



Notes:  
 1.  $V_{out} = 1.20 * (1 + R1/R2) = 3.296V$ ;  
 R1=52.3K, R2=30.1K are recommended for better performance.

Fig. 38: ESP32-S3-Korvo-2 V3.0 - Audio Power Supply

GPIO Allocation Summary

The table below provides allocation of GPIOs exposed on terminals of ESP32-S3-WROOM-1 module to control specific components or functions of the board.

Table 2: ESP32-S3-WROOM-1 GPIO Allocation

Pin	Pin Name	ES8311	ES7210	Camera	LCD	Keys	MicroSD	IO Expander	Other
3	EN					EN_KEY			
4	IO4						DATA0		
5	IO5					REC, MUTE, PLAY, SET, VOL-, VOL+			
6	IO6								BAT_MEAS_ADC
7	IO7						CMD		
8	IO15						CLK		
9	IO16	I2S0_MCLK	MCLK						
10	IO17	I2C_SDA	I2C_SDA	SIOD	TP_I2C_SDA			I2C_SDA	
11	IO18	I2C_CLK	I2C_CLK	SIOC	TP_I2C_CLK			I2C_CLK	
12	IO8	I2S0_DSDIN							
13	IO19								ESP_USB_DM (Re-serve)
14	IO20								ESP_USB_DP (Re-serve)
15	IO3			D5					
16	IO46								NC
17	IO9	I2S0_SCLK	SCLK						
18	IO10		SDOUT						

continues on next page

Table 2 – continued from previous page

Pin <sup>Page 399</sup>	Pin Name	ES8311	ES7210	Camera	LCD	Keys	MicroSD	IO Expander	Other
19	IO11			PCLK					
20	IO12			D6					
21	IO13			D2					
22	IO14			D4					
23	IO21			VSYNC					
24	IO47			D3					
25	IO48								PA_CTRL
26	IO45	I2S0_LRCK	RCK						
27	IO0				LCD_SPI	BOOT_KEY			
28	IO35								NC
29	IO36								NC
30	IO37								NC
31	IO38			HREF					
32	IO39			D9					
33	IO40			XCLK					
34	IO41			D8					
35	IO42			D7					
36	RXD0								ESP0_UART0_RX
37	TXD0								ESP0_UART0_TX
38	IO2				LCD_SPI	DC			
39	IO1				LCD_SPI	CLK			
41	EPAD								

The GPIOs allocated to the IO expander are further expanded to multiple GPIOs.

Table 3: IO Expander GPIO Allocation

IO Expander Pin	Pin Name	LCD	Other
4	P0		PA_CTRL
5	P1	LCD_CTRL	
6	P2	LCD_RST	
7	P3	LCD_CS	
9	P4	TP_INT	
10	P5		PERI_PWR_ON
11	P6		LED1
12	P7		LED2

<sup>1</sup> Pin - ESP32-S3-WROOM-1 module pin number, GND and power supply pins are not listed.

### Connector

#### Camera Connector

No.	Camera Signal	ESP32-S3 Pin
1	SIOD	GPIO17
2	SIOC	GPIO18
3	D5	GPIO3
4	PCLK	GPIO11
5	D6	GPIO12
6	D2	GPIO13
7	D4	GPIO14
8	VSYNC	GPIO21
9	D3	GPIO47
10	HREF	GPIO38
11	D9	GPIO39
12	XCLK	GPIO40
13	D8	GPIO41
14	D7	GPIO42

#### LCD Connector

No.	LCD Signal	ESP32-S3 Pin
1	TP_I2C_SDA	GPIO17
2	TP_I2C_CLK	GPIO18
3	LCD_SPI_SDA	GPIO0
4	LCD_SPI_DC	GPIO2
5	LCD_SPI_CLK	GPIO1

No.	LCD Signal	IO Expander
1	ESP_LCD_CTRL	P1
2	ESP_LCD_RST	P2
3	ESP_LCD_CS	P3
4	ESP_TP_INT	P4

#### AEC Path

AEC path provides reference signals for AEC algorithm.

ESP32-S3-Korvo-2 provides two compatible echo reference signal source designs. One is Codec (ES8311) DAC output (DAC\_AOUTLN/DAC\_AOUTLP), the other is PA (NS4150) output (PA\_OUTL+/PA\_OUTL-). The former is the default and recommended selection. Resistors R132 and R140 marked NC (no component) in the figure below should not be installed.

The echo reference signal is collected by ADC\_MIC3P/ADC\_MIC3N of ADC (ES7210) and then sent back to ESP32-S3 for AEC algorithm.

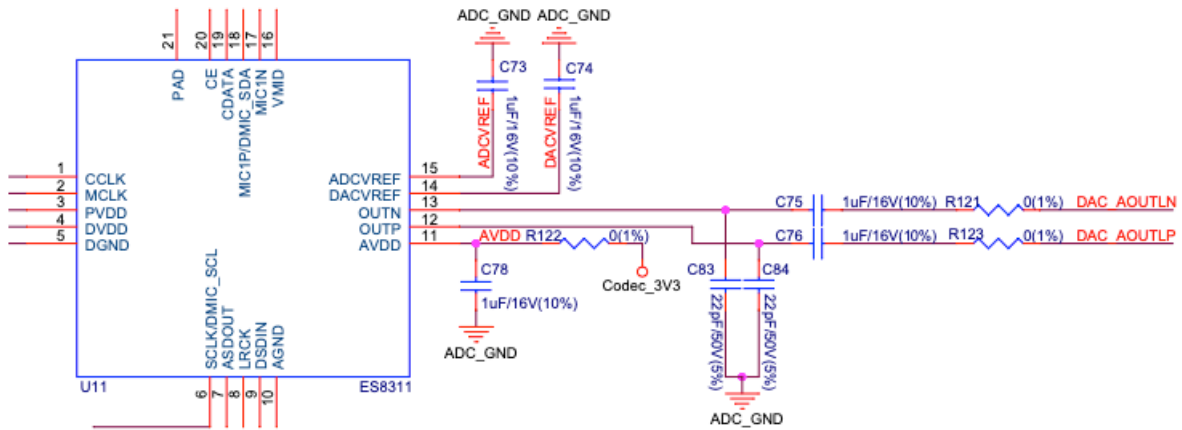


Fig. 39: ESP32-S3-Korvo-2 V3.0 - AEC Codec DAC Output (click to enlarge)

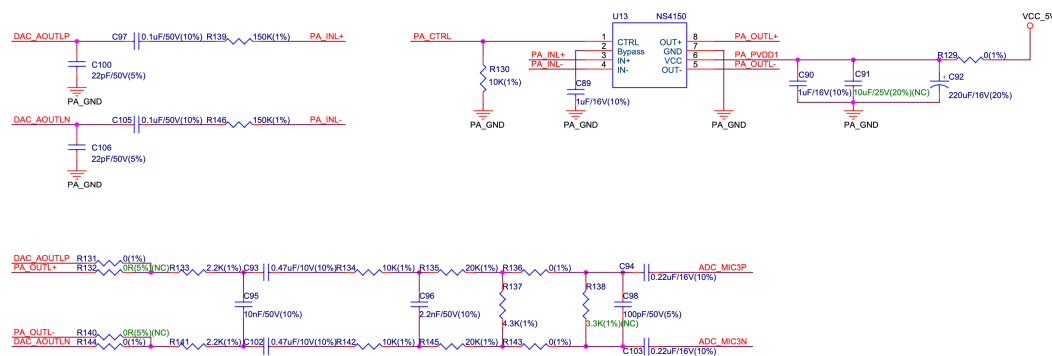


Fig. 40: ESP32-S3-Korvo-2 V3.0 - AEC PA Output (click to enlarge)

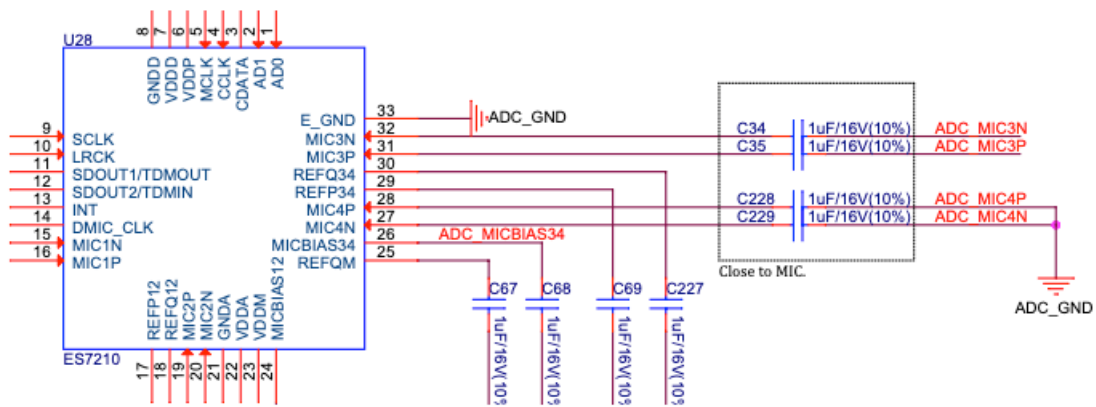


Fig. 41: ESP32-S3-Korvo-2 V3.0 - AEC Reference Signal Collection (click to enlarge)

### Hardware Setup Options

#### Using Automatic Upload

Entering of the ESP board into upload mode may be done in two ways:

- Manually by pressing both Boot and RST keys and then releasing first RST and then Boot key.
- Automatically by software performing the upload. The software is using DTR and RTS signals of the serial interface to control states of EN and IO0 of the ESP board. For details see [ESP32-S3-Korvo-2 V3.0 Schematic \(PDF\)](#).

#### Allocation of ESP Pins to Test Points

This section describes the allocation of test points available on the ESP32-S3-Korvo-2 V3.0 board.

The test points are bare through hole solder pads and have a standard 2.54 mm/0.1” pitch. You may need to populate them with pin headers or sockets for easy connection of external hardware.

#### Codec Test Point/J15

No.	Codec Pin	ESP32-S3 Pin
1	MCLK	GPIO16
2	SCLK	GPIO9
3	LRCK	GPIO45
4	DSDIN	GPIO8
5	ASDOUT	–
6	GND	–

#### ADC Test Point/J16

No.	ADC Pin	ESP32-S3 Pin
1	MCLK	GPIO16
2	SCLK	GPIO9
3	LRCK	GPIO45
4	SDOUT	GPIO10
5	INT	–
6	GND	–

#### UART Test Point/J17

No.	UART Pin
1	3.3V
2	TXD
3	RXD
4	IO0
5	EN
6	GND

## I2C Test Point/J18

No.	I2C Pin	ESP32-S3 Pin
1	3.3V	–
2	CLK	GPIO18
3	SDA	GPIO17
4	GND	–

## Hardware Revision Details

This is the first revision of this board released.

## Related Documents

- [ESP32-S3 Series Datasheet \(PDF\)](#)
- [ESP32-S3-WROOM-1/1U Datasheet \(PDF\)](#)
- [ESP32-S3-Korvo-2 V3.0 Schematic \(PDF\)](#)
- [ESP32-S3-Korvo-2 V3.0 PCB Layout \(PDF\)](#)

For further design documentation for the board, please contact us at [sales@espressif.com](mailto:sales@espressif.com).

### 3.4.10 ESP32-S3-Korvo-2-LCD V1.0

This user guide will help you get started with the ESP32-S3-Korvo-2-LCD extension board and will also provide more in-depth information.

This extension board cannot be bought separately and is sold as part of the *ESP32-S3-Korvo-2 V3.0 accessories*.

ESP32-S3-Korvo-2-LCD extends the functionality of ESP32-S3-Korvo-2 V3.0 (referred to as *main board* below) by adding an LCD graphic display and capacitive touchpad.

The document consists of the following major sections:

- *Getting started*: Overview of the board and hardware/software setup instructions to get started.
- *Hardware Reference*: More detailed information about the board's hardware.
- *Related Documents*: Links to related documentation.

## Getting Started

This extension board adds a 2.4" LCD graphic display with the resolution of 320x240 and a 10-point capacitive touchpad. This display is connected to ESP32-S3 over the SPI bus.



Fig. 42: ESP32-S3-Korvo-2-LCD V1.0



## Description of Components

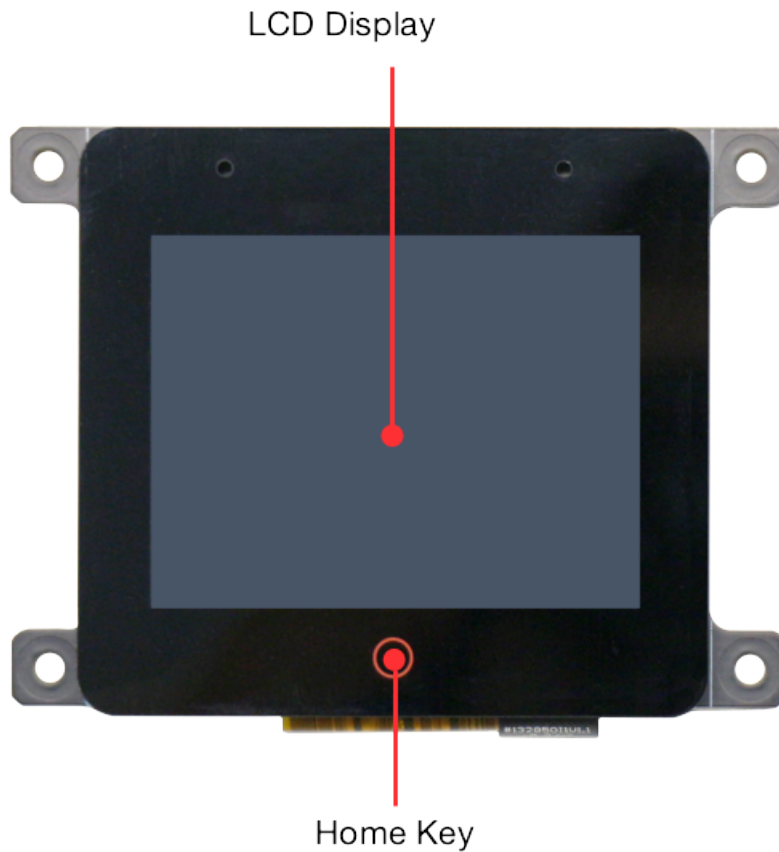


Fig. 43: ESP32-S3-Korvo-2-LCD V1.0 - front

The key components of the board are described in a clockwise direction. **Reserved** means that the functionality is available, but the current version of the board does not use it.

Key Component	Description
LCD Display	A 2.4" 320x240 SPI LCD display module; the display driver/controller is Ilitek ILI934.
Home Key	(Reserved) Returns to homepage or previous page.
Signal Connector	Connects the power, ground, and signal wires between the extension board and the main board with a FPC cable.
LCD Connector	Connects LCD display to the driver circuit of this board.
TP Connector	Connects LCD display to the touch circuit of this board.

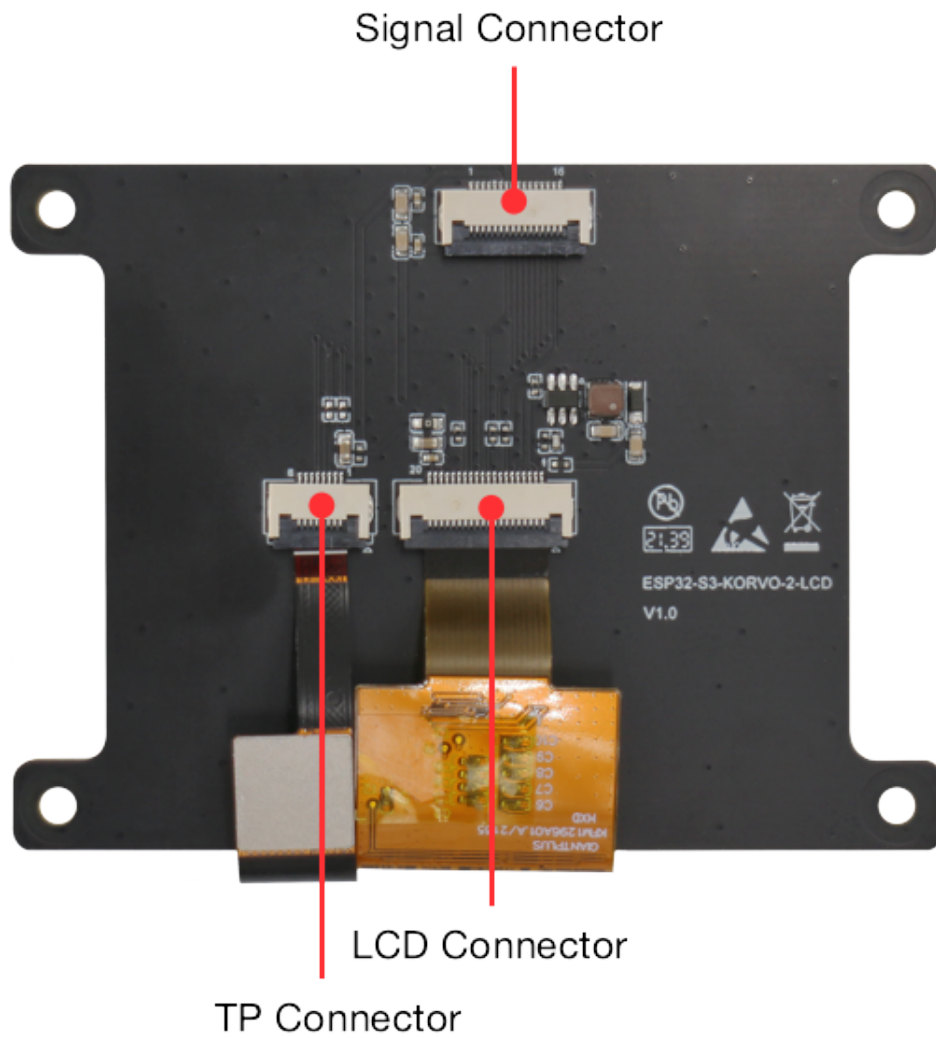


Fig. 44: ESP32-S3-Korvo-2-LCD V1.0 - back

## Start Application Development

Before powering up your board, please make sure that it is in good condition with no obvious signs of damage.

### Required Hardware

- Main board: ESP32-S3-Korvo-2 V3.0
- Extension board: ESP32-S3-Korvo-2-LCD V1.0
- Two USB 2.0 cables (Standard-A to Micro-B)
- Mounting copper standoffs and screws (for stable mounting)
- A FPC cable (for connecting the main board and extension board)
- Computer running Windows, Linux, or macOS

### Hardware Setup

To mount your ESP32-S3-Korvo-2-LCD onto ESP32-S3-Korvo-2:

1. Connect the two boards with the FPC cable.
2. Install copper standoffs and screws for stable mounting.

### Software Setup

See Section *Software Setup* of the main board user guide.

### Hardware Reference

#### Block Diagram

The block diagram below shows the components of ESP32-S3-Korvo-2-LCD and their interconnections.

#### Hardware Revision Details

Initial release.

#### Related Documents

- *ESP32-S3-Korvo-2 V3.0*
- *ESP32-S3-Korvo-2-LCD Schematic (PDF)*
- *ESP32-S3-Korvo-2-LCD PCB layout (PDF)*

For further design documentation for the board, please contact us at [sales@espressif.com](mailto:sales@espressif.com).

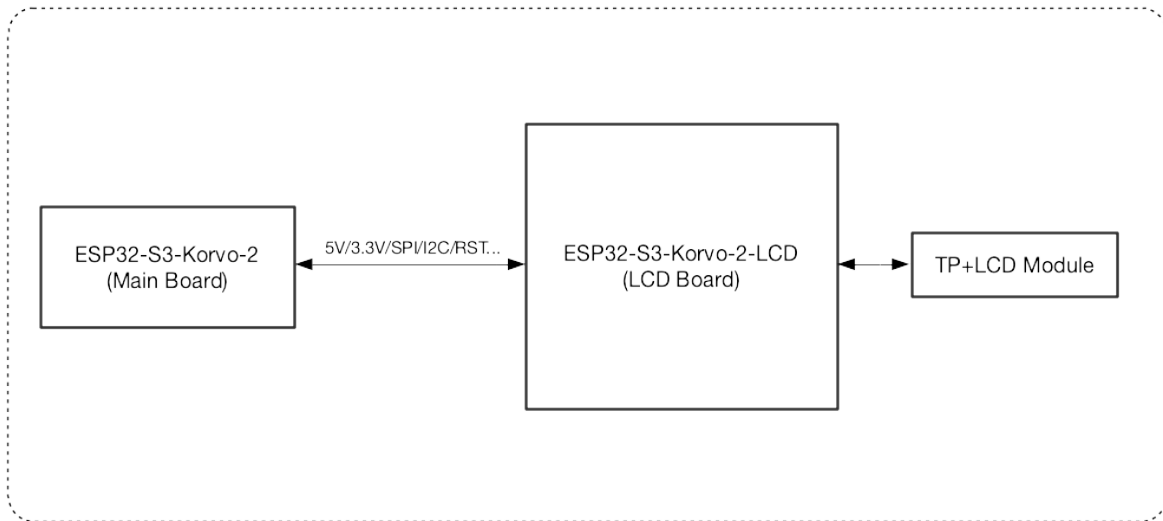


Fig. 45: ESP32-S3-Korvo-2-LCD

### 3.4.11 ESP32-C3-Lyra V2.0

This user guide will help you get started with ESP32-C3-Lyra V2.0 and will also provide more in-depth information.

The document consists of the following sections:

- *Board Overview*: Overview of the board hardware/software.
- *Start Application Development*: How to set up hardware/software to develop applications.
- *Hardware Reference*: More detailed information about the board's hardware.
- *Hardware Revision Details*: Hardware revision history, known issues, and links to user guides for previous versions (if any) of the board.
- *Ordering*: How to buy the board.
- *Related Documents*: Links to related documentation.

#### Board Overview

ESP32-C3-Lyra is an ESP32-C3-based audio development board produced by Espressif for controlling light with audio. The board has control over the microphone, speaker, and LED strip, perfectly matching customers' product development needs for ultra-low-cost and high-performance audio broadcasters and rhythm light strips.

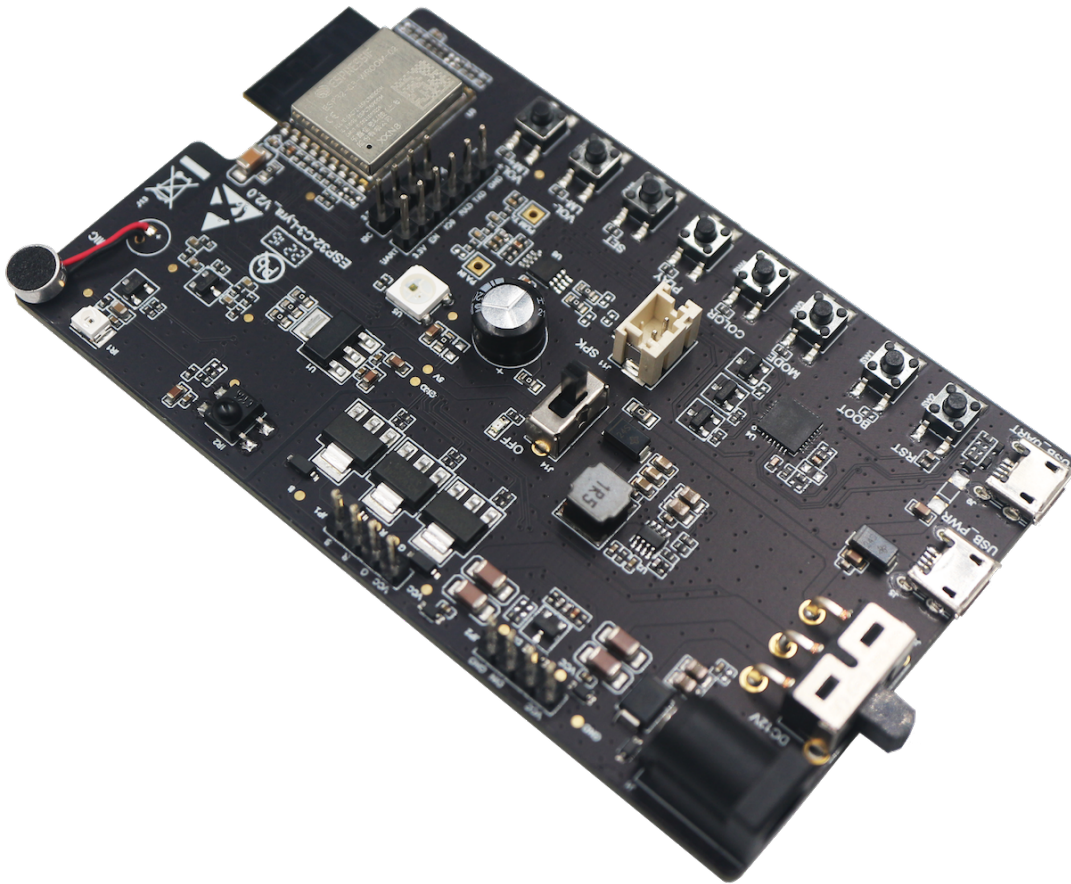


Fig. 46: ESP32-C3-Lyra with ESP32-C3-WROOM-02 module

## Feature List

The main features of the board are listed below:

- **Module Embedded:** ESP32-C3-WROOM-02 module with a 4 MB external SPI flash
- **Audio:** built-in ECM microphone, speaker power amplifier, speaker connector
- **LED Strip Connector:** support for two types of connections, i.e., connections to addressable LED strips and RGB LED strips
- **Infrared Control:** support for infrared (IR) transmitting and receiving
- **Keys:** boot key, reset key, and six function keys (MODE, COLOR, PLAY/PAUSE, SET, VOL+/LM+, VOL-/LM-)
- **USB:** 1 x USB Power Port, 1 x USB-to-UART Port
- **Power Supply:** 5 V power supply over USB or 12 V DC power supply

## Block Diagram

The block diagram below shows the components of ESP32-C3-Lyra and their interconnections.

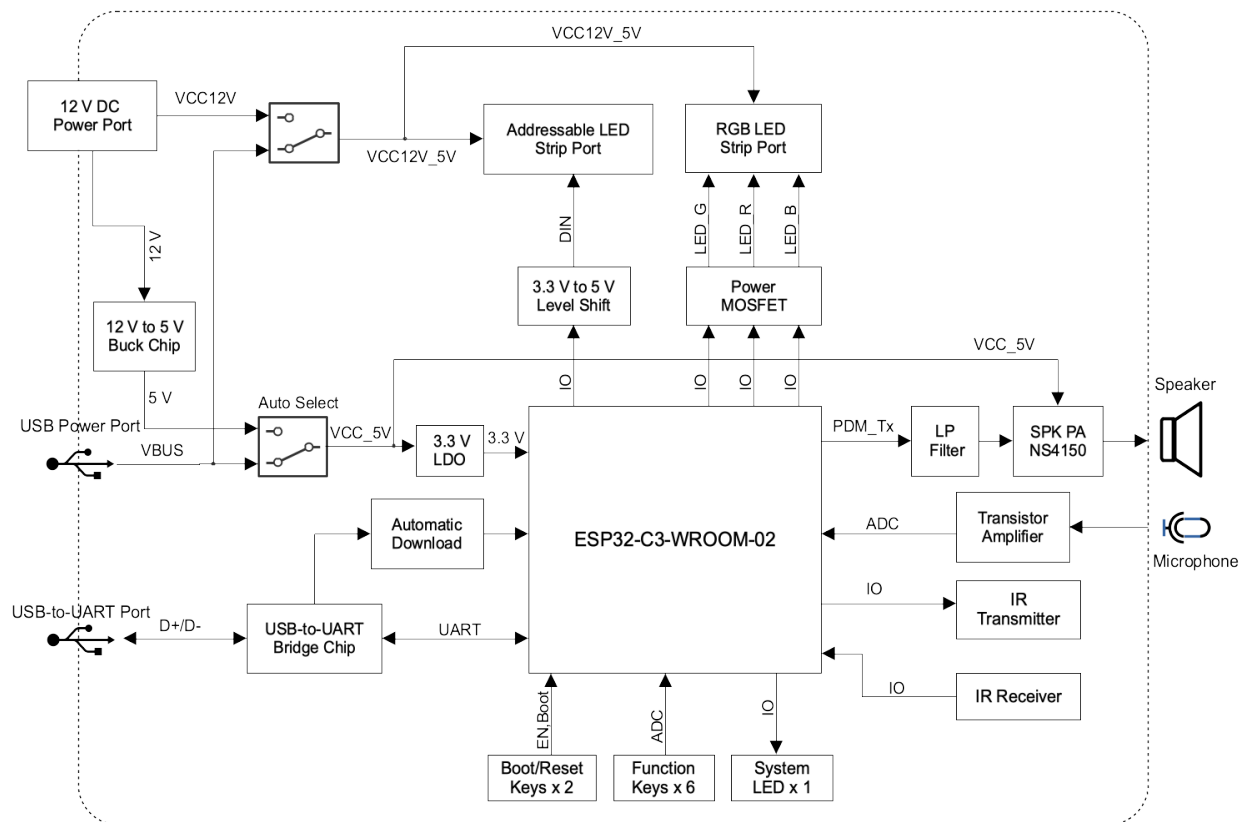


Fig. 47: ESP32-C3-Lyra Block Diagram (click to enlarge)

Description of Components

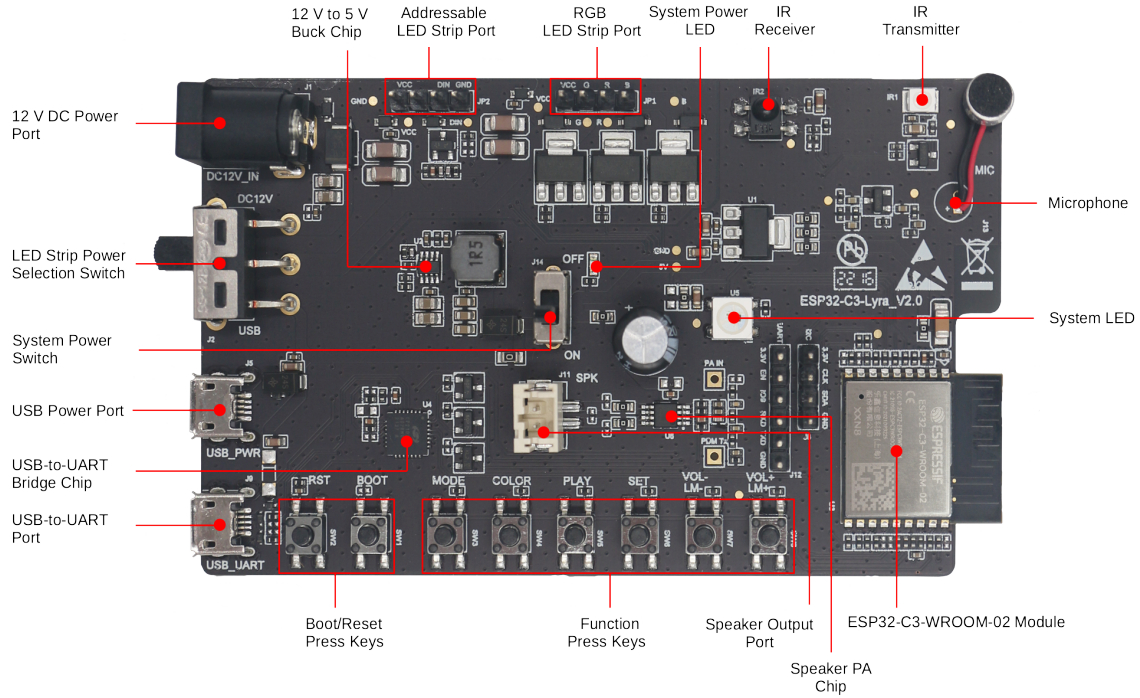


Fig. 48: ESP32-C3-Lyra - front (click to enlarge)

The key components of the board are described in a clockwise direction.

Key Component	Description
ESP32-C3-WROOM-02 Module	It is a general-purpose Wi-Fi and Bluetooth LE module developed based on ESP32-C3, a 32-bit RISC-V single-core processor that operates at up to 160 MHz. The module has a rich set of peripherals and a high performance, making it an ideal choice for smart homes, industrial automation, health care, consumer electronics, etc. It integrates a 4 MB external SPI flash and an on-board PCB antenna. The ESP32-C3-WROOM-02U module is also compatible with this board, but it needs to connect to an external antenna.
Speaker PA Chip	NS4150 is an EMI, 3 W mono Class D audio power amplifier, amplifying audio signals from ESP32-C3 PDM_TX to drive speakers.
Speaker Output Port	Output socket to connect a speaker. The 4-ohm and 3-watt speaker is recommended. The pins have a 2.00 mm/0.08" pitch.
Function Press Keys	Six function press keys, including MODE, COLOR, PLAY/PAUSE, SET, VOL+/LM+, VOL-/LM-. They are routed to the <b>ESP32-C3-WROOM-02 Module</b> and intended for the development and testing of a UI for audio applications or LED strips using dedicated APIs.
Boot/Reset Press Keys	Boot: holding down the Boot key and momentarily pressing the Reset button initiates the firmware upload mode. Then you can upload firmware through the serial port. Reset: pressing this key alone resets the system.
USB-to-UART Port	Functions as the communication interface between the PC and the ESP32-C3-WROOM-02 module.
USB-to-UART Bridge Chip	The single-chip USB-UART bridge CP2102N provides up to 3 Mbps transfer rates for software download and debugging.
USB Power Port	Provides power to the board. It is recommended to use at least 5 V/2 A power adapters to ensure a stable power supply.
System Power Switch	System Power on/off knob. Toggling it to <b>ON</b> turns the 5 V system power on; toggling it to <b>OFF</b> turns the 5 V system power off.
LED Strip Power Selection Switch	Toggle this switch to select between a 5 V power supply over USB and a 12 V DC power supply for your LED strip according to the working voltage of the LED strip and the type of the power adapter you actually use.
12 V DC Power Port	Supports 12 V DC power adapters with a maximum current of 2 A. The DC power Jack contact have an outer diameter of 5.5 mm and an inner contact diameter of 2.5 mm.
12 V to 5 V Buck Chip	The 12 V to 5 V buck chip MP2313 is a high-efficiency synchronous step-down converter that operates at 1 A and 2 MHz.
Addressable LED Strip Port	It is a male pin header connector with 4 x 1 pins and a 2.54 mm pitch. It can connect to the addressable LED strip that is controlled with a single wire. It supports 5 V and 12 V LED strips, such as WS2811 and WS2812 LED. ESP32-C3 can send commands to control the LED strips over RMT or SPI.
RGB LED Strip Port	It is a male pin header connector with 4 x 1 pins and a 2.54 mm pitch. It can connect to regular RGB LED strips (non-addressable, individual wire per color) that operate at 5 V or 12 V. ESP32-C3 can output PWM waveform via this port to control the LED strips.
System Power LED	It turns red when <b>System Power Switch</b> is toggled to ON.
IR Receiver	IRM-H638T/TR2 is a miniature SMD type infrared remote control system receiver. The demodulated output signal can directly be decoded by ESP32-C3.
IR Transmitter	IR67-21C/TR8 is an infrared emitting diode. It is spectrally matched with silicon photodiode and phototransistor.
Microphone	On-board ECM microphone. Signals picked up by it are amplified via transistors and sent to the analog-to-digital converter (ADC) of ESP32-C3-WROOM-02.
System LED	It is an RGB LED, model WS2812C, controlled by the ESP32-C3-WROOM-02 via GPIO, which can be used to indicate the operating status of the audio application.



## Default Firmware and Function Test

Each ESP32-C3-Lyra comes with a pre-built default firmware that allows you to test its functions including LED control (LEDC), remote control transceiver (RMT), ADC, and pulse-density modulation (PDM\_TX). This section describes how to test peripheral's function with the pre-built firmware.

---

**Note:** The default firmware of ESP32-C3-Lyra sold in the Developing IoT Projects with ESP32-C3 Package II provides a musical rhythm light ring effect.

---

## Preparing Hardware

See Section *Required Hardware* and *Optional Hardware* for more information.

- 1 x ESP32-C3-Lyra
- 2 x USB 2.0 cable (Standard-A to Micro-B)
- 1 x Computer running Windows, Linux, or macOS
- 1 x 5 V RGB LED strip WS2812 (optional)
- 1 x Mobile phone or music player
- 1 x Speaker (optional)

## Connecting Hardware

- Before powering up your board, please make sure that it is in good condition with no obvious signs of damage.
- Connect the board to the 5 V power supply through the **USB Power Port** using a USB cable. After the board is powered up, you will notice that the **System Power LED** turns on, which means the board is powered up. If the LED is not on, please toggle the **System Power Switch**.
- Toggle the **LED Strip Power Selection Switch** to the USB power side.
- Connect the board to the computer through the **USB-to-UART Port** using a USB cable.

## Testing Default Firmware

---

**Note:** If you are developing with ESP32-C3-Lyra provided in the Developing IoT Projects with ESP32-C3 Package II, please skip this step.

---

1. Press the **Reset Press Key** on the board.
2. The board automatically starts the flash test. The log shown on a PC connected to **USB-to-UART Port** is as follows:

```
Step1 Flash Test Start
Step1 Flash Test OK
```

3. The board tests the **Function Press Keys**. Please press the key as the log prompts. For example, press **VOL+** when you see the following log:

```
Step2 Keys Test Start
Please press The Key: VOL+
```

4. The board tests the **System LED**. You will see the LED keep switching between red, blue, and green. Then, press the key VOL+/LM+ to proceed to the next step.
5. The board tests LEDC (PWM). If you connect an RGB LED strip to the **RGB LED Strip Port**, you will see the LEDs breathing. Then, press the key VOL+/LM+ to proceed to the next step.
6. The board tests ADC. If you play the 1 kHz sine audio signal close to the **Microphone** with the mobile phone or music player, the following log will be seen when the board detects the audio signal:

```
Step5 Adc Test Start
Please play 1khz audio
Step5 Adc Test OK
```

7. The board tests the PDM\_TX function. Connect the speaker to the **Speaker Output Port** and you will hear the music played from flash.

## Software Support

ESP-ADF is the development framework for ESP32-C3-Lyra. To see which version of ESP-ADF is supported for this board, please go to the section [Hardware](#).

Below are other software repositories developed by Espressif that may help you experiment with the functions of ESP32-C3-Lyra.

- **ESP-IDF**: development framework for Espressif SoCs based on FreeRTOS with a rich set of components, including LED control (LEDC), ADC, RMT, SPI etc.

Application examples for this board can be found at [application example](#) .

## Start Application Development

This section provides instructions on how to do hardware and software setup and flash firmware onto the board to develop your own application.

## Required Hardware

Hardware	Qty	Note
ESP32-C3-Lyra	1	–
USB 2.0 cables (Standard-A to Micro-B)	2	One for USB power supply, the other for flashing firmware onto the board. Be sure to use an appropriate USB cable. Some cables are for charging only and do not provide the needed data lines nor work for programming the boards.
Computer running Windows, Linux, or macOS	1	–
Speaker	1	The 4-ohm 3-watt speaker is recommended. It should be fitted with JST PH 2.0 2-Pin plugs. In case you do not have this type of plug it is also fine to use Dupont female jumper wires during development.

## Optional Hardware

Hardware	Qty	Note
12 V DC adapter	1	The maximum operating current of the adapter is 2 A. It provides power supply for 12 V LED strips.
5 V or 12 V addressable LED strip/ring	1	It is recommended to use WS2812/WS2811 LED strip (a female connector with 4 x 1 pins and a 2.54 mm pitch), or a WS2812 LED ring with 16 individually addressable RGB LEDs assembled (a female connector with 3 x 1 pins and a 2.54 mm pitch). This LED strip/ring should be connected to the <b>Addressable LED Strip Port (JP2)</b> .
5 V or 12 V RGB LED strip	1	It should have a female connector with 4 x 1 pins and a 2.54 mm pitch. This LED strip should be connected to <b>RGB LED Strip Port (JP1)</b> .

## Power Supply Options

There are two ways to provide power to the board:

- **USB Power Port (5 V)**
- **12 V DC Power Port**

## Hardware Setup

Prepare the board for loading of the first sample application:

1. Connect the speaker to the **Speaker Output Port**.
2. (Optional) Connect the LED strip to the development board through the **Addressable LED Strip Port** or the **RGB LED Port** depending on the type of your LED strip.
3. Connect the power supply to the development board through the **USB Power Port (5 V)** or the **12 V DC Power Port** accordingly to supply power for your LED strip.
4. (Optional) Toggle the **LED Strip Power Selection Switch** depending on the working voltage and current of your LED strip.

---

**Note:** If you toggle the switch to the wrong side, the light strip will work abnormally. **Do not** power the 5 V LED strip with the 12 V DC adapter. Otherwise, the light strip will be damaged.

---

5. Toggle the **System Power Switch** to **ON**. The red **System Power LED** should turn on.
6. Connect the board to the computer through the **USB-to-UART Port** using a USB cable.

Now the board is ready for software setup.

## Software Setup

After hardware setup, you can proceed to *Get Started* to prepare development tools.

For more software information on developing applications, please go to *Software Support*.

## Hardware Reference

This section provides more detailed information about the board's hardware.

## GPIO Allocation

The table provides the allocation of GPIOs exposed on terminals of the ESP32-C3-WROOM-02 module to control specific components or functions of the board.

## Power Distribution

### Power Supply over USB or from 12 V/2 A DC Input

There are two ways to power the development board: 5 V USB Power Port or 12 V/2 A DC input.

#### USB-PWR

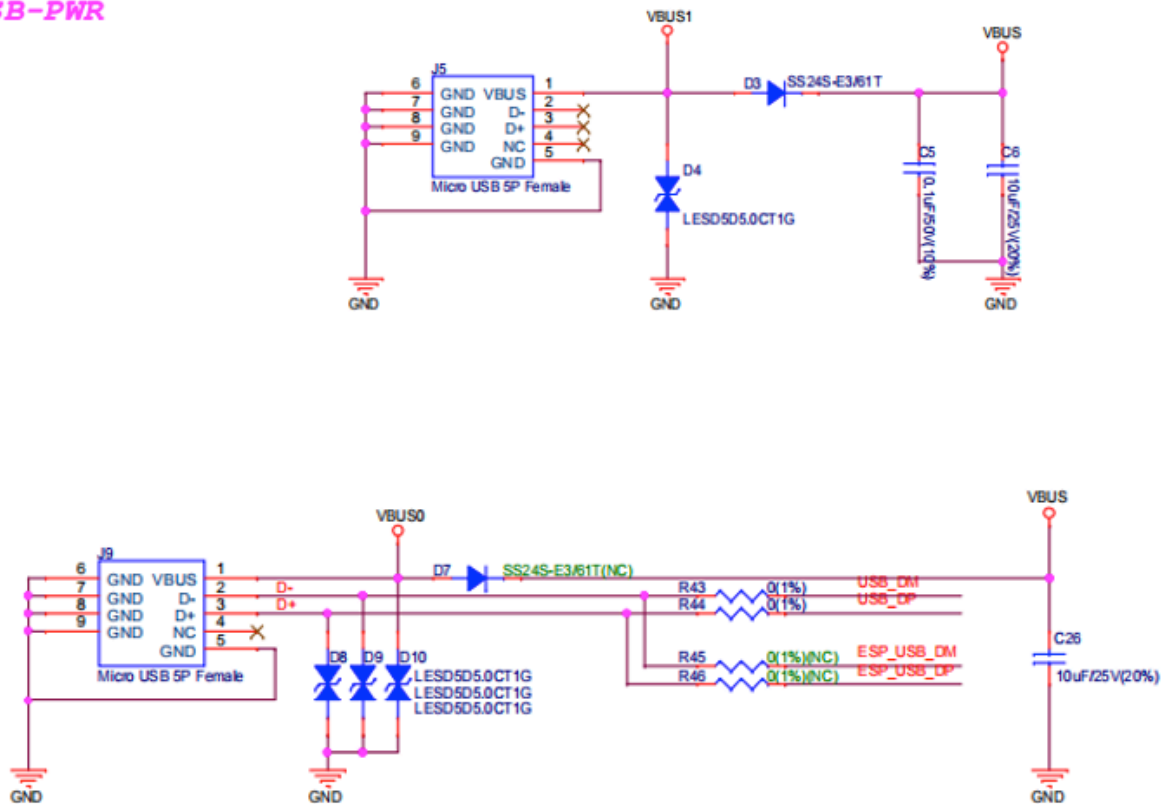


Fig. 49: ESP32-C3-Lyra - Dedicated USB Power Supply Socket

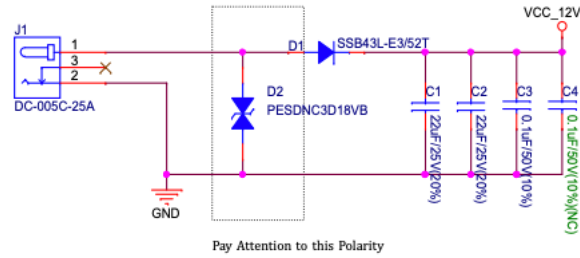


Fig. 50: ESP32-C3-Lyra - Power Supply from 12 V DC Input

### LED Strip Power Selection Switch

According to the working voltage and current of your LED strip, select a proper power adapter and the port, and toggle the **LED Strip Power Selection Switch** to the corresponding side to power up the LED strip.

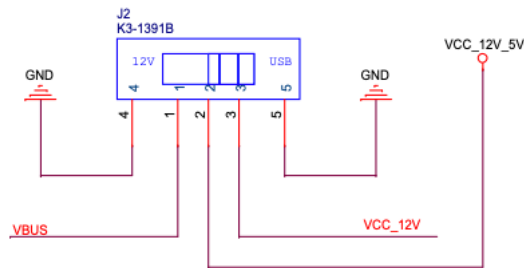


Fig. 51: LED Strip Power Selection Switch

### 12 V to 5 V Buck Power

### System 3.3 V Power

### Connector

### RGB LED Strip Connector (JP1)

No.	Signal Name	ESP32-C3 Pin
1	VCC_12V_5V	-
2	LED_G	GPIO6
3	LED_R	GPIO5
4	LED_B	GPIO4

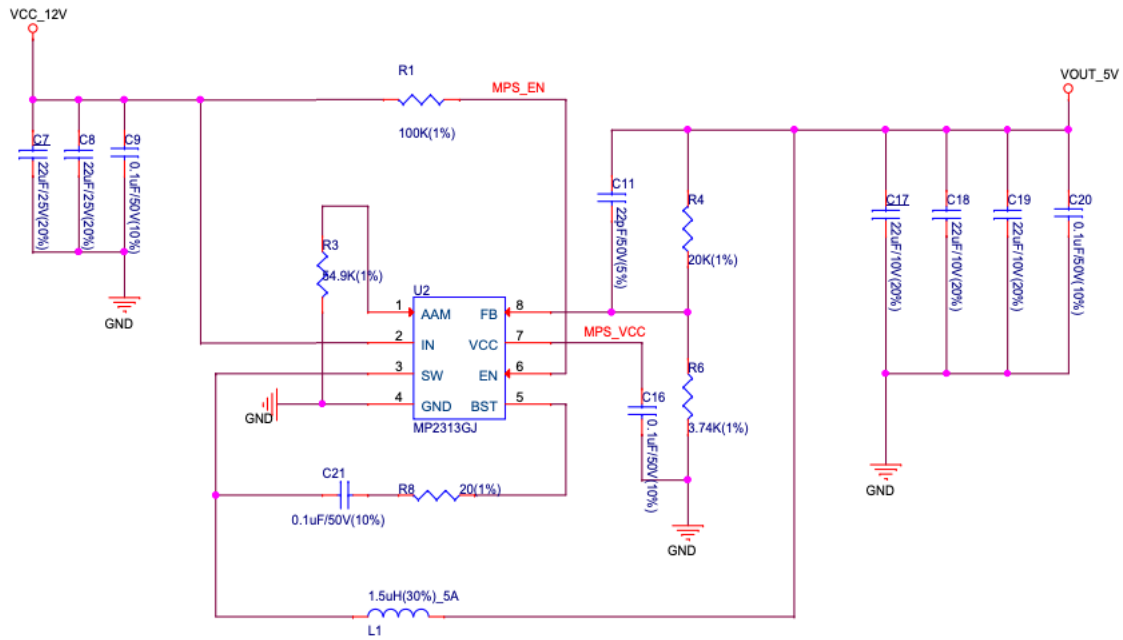


Fig. 52: 12 V to 5 V Buck Power Circuit

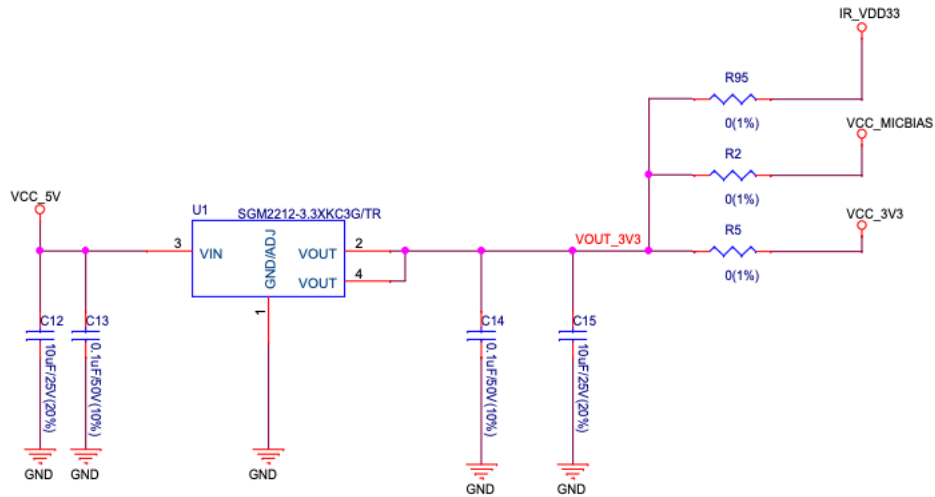


Fig. 53: System 3.3 V Power Circuit

**Addressable LED Strip Connector (JP2)**

No.	Signal Name	ESP32-C3 Pin
1	VCC_12V_5V	–
2	DIN	GPIO7
3	DIN	GPIO7
4	GND	–

**Pinout of Extension Headers**

There are several pin headers available to connect external components, check the state of particular signal bus, or debug operation of ESP32-C3. Note that some signals are shared. See Section *GPIO Allocation* for details.

**UART Header (JP12)**

No.	Signal Name	ESP32-C3 Pin
1	VCC_3V3	–
2	ESP_EN	EN
3	ESP_BOOT	GPIO9
4	ESP_UART_RXD	U0RXD
5	ESP_UART_TXD	U0TXD
6	GND	–

**I2C Header (JP8)**

No.	Signal Name	ESP32-C3 Pin
1	VCC_3V3	–
2	I2C_CLK	GPIO8
3	I2C_DATA	GPIO9
4	GND	–

**Hardware Revision Details**

No previous revisions.

**Ordering**

If you order a few samples, each board comes in an individual package.

For retail orders, please go to the official website <https://www.espressif.com/en/company/contact/buy-a-sample>, or directly order in our Taobao shop <https://world.taobao.com/item/677273363812.htm?spm=a21wu.12321156-tw.recommend-tpp.4.19a61924ZMaqpf>.

For wholesale orders, please go to the official website <https://www.espressif.com/en/contact-us/sales-questions>.

### Related Documents

- Datasheet
  - [ESP32-C3 Series Datasheet \(PDF\)](#)
  - [ESP32-C3-WROOM-02 & ESP32-C3-WROOM-02U Datasheet \(PDF\)](#)
- Schematic
  - [ESP32-C3-Lyra Schematic \(PDF\)](#)
- PCB Layout
  - [ESP32-C3-Lyra PCB Layout \(PDF\)](#)

For further design documentation for the board, please contact us at [sales@espressif.com](mailto:sales@espressif.com).

## 3.5 Audio Samples

Music files in this section are intended for testing of audio applications. The files are organized into different *Formats* and *Sample Rates*.

### 3.5.1 Formats

The tables below provides an audio file converted from 'wav' format into several other audio formats.

#### Long Samples

The audio track duration in this section is 3 minutes and 7 seconds.

#### Two Channel Audio

No	Format	Audio File	Size [kB]
1	aac	ff-16b-2c-44100hz.aac	2,995
2	ac3	ff-16b-2c-44100hz.ac3	2,994
3	aiff	ff-16b-2c-44100hz.aiff	33,002
4	flac	ff-16b-2c-44100hz.flac	22,406
5	m4a	ff-16b-2c-44100hz.m4a	3,028
6	mp3	ff-16b-2c-44100hz.mp3	2,994
7	mp4	ff-16b-2c-44100hz.mp4	3,079
8	ogg	ff-16b-2c-44100hz.ogg	2,612
9	opus	ff-16b-2c-44100hz.opus	2,598
10	ts	ff-16b-2c-44100hz.ts	5,510
11	wav	ff-16b-2c-44100hz.wav	32,229
12	wma	ff-16b-2c-44100hz.wma	3,227

Playlist containing all above files: [ff-16b-2c-playlist.m3u](#)



### Single Channel Audio

No	Format	Audio File	Size [kB]
1	aac	ff-16b-1c-44100hz.aac	1,650
2	ac3	ff-16b-1c-44100hz.ac3	2,193
3	aiff	ff-16b-1c-44100hz.aiff	16,115
4	amr	ff-16b-1c-8000hz.amr	299
5	flac	ff-16b-1c-44100hz.flac	10,655
6	m4a	ff-16b-1c-44100hz.m4a	1,628
7	mp3	ff-16b-1c-44100hz.mp3	1,463
8	ogg	ff-16b-1c-44100hz.ogg	1,558
9	opus	ff-16b-1c-44100hz.opus	1,641
10	wav	ff-16b-1c-44100hz.wav	16,115
11	wma	ff-16b-1c-44100hz.wma	3,151

Playlist containing all above files: ff-16b-1c-playlist.m3u

### Short Samples

If you need shorter audio files for testing, this section provides 16 seconds audio tracks.

### Two Channel Audio

No	Format	Audio File	Size [kB]
1	aac	gs-16b-2c-44100hz.aac	241
2	ac3	gs-16b-2c-44100hz.ac3	380
3	aiff	gs-16b-2c-44100hz.aiff	2,792
4	flac	gs-16b-2c-44100hz.flac	1,336
5	m4a	gs-16b-2c-44100hz.m4a	258
6	mp3	gs-16b-2c-44100hz.mp3	254
7	mp4	gs-16b-2c-44100hz.mp4	259
8	ogg	gs-16b-2c-44100hz.ogg	229
9	opus	gs-16b-2c-44100hz.opus	219
10	ts	gs-16b-2c-44100hz.ts	286
11	wav	gs-16b-2c-44100hz.wav	2,792
12	wma	gs-16b-2c-44100hz.wma	276

Playlist containing all above files: gs-16b-2c-playlist.m3u

### Single Channel Audio

No	Format	Audio File	Size [kB]
1	amr	gs-16b-1c-8000hz.amr	25
2	aac	gs-16b-1c-44100hz.aac	137
3	ac3	gs-16b-1c-44100hz.ac3	190
4	aiff	gs-16b-1c-44100hz.aiff	1,397
5	flac	gs-16b-1c-44100hz.flac	645
6	m4a	gs-16b-1c-44100hz.m4a	258
7	mp3	gs-16b-1c-44100hz.mp3	127
8	ogg	gs-16b-1c-44100hz.ogg	144
9	opus	gs-16b-1c-44100hz.opus	132
10	wav	gs-16b-1c-44100hz.wav	1,497
11	wma	gs-16b-1c-44100hz.wma	276

Playlist containing all above files: gs-16b-1c-playlist.m3u

### 3.5.2 Sample Rates

The files in this section have been prepared by converting a single audio file into different sampling rates defined in MPEG Layer III specification. Both mono and stereo versions of files are provided. The bit depth of files is 16 bits.

	Sample Rate	MPEG III	Channels	Bit Rate	Size
Audio File	[Hz]	ver		[kbit/s]	[kB]
ff-16b-1c-8000hz.mp3	8000	2.5	mono	8	183
ff-16b-1c-11025hz.mp3	11025	2.5	mono	16	366
ff-16b-1c-12000hz.mp3	12000	2.5	mono	16	366
ff-16b-1c-16000hz.mp3	16000	2	mono	24	548
ff-16b-1c-22050hz.mp3	22050	2	mono	32	731
ff-16b-1c-24000hz.mp3	24000	2	mono	32	731
ff-16b-1c-32000hz.mp3	32000	1	mono	48	1,097
ff-16b-1c-44100hz.mp3	44100	1	mono	64	1,462
ff-16b-2c-8000hz.mp3	8000	2.5	joint stereo	24	549
ff-16b-2c-11025hz.mp3	11025	2.5	joint stereo	32	731
ff-16b-2c-12000hz.mp3	12000	2.5	joint stereo	32	731
ff-16b-2c-16000hz.mp3	16000	2	joint stereo	48	1,097
ff-16b-2c-22050hz.mp3	22050	2	joint stereo	64	1,462
ff-16b-2c-24000hz.mp3	24000	2	joint stereo	64	1,462
ff-16b-2c-32000hz.mp3	32000	1	joint stereo	96	2,194
ff-16b-2c-44100hz.mp3	44100	1	joint stereo	128	2,924

Playlist containing all above files: ff-16b-mp3-playlist.m3u

Original music files: “Furious Freak” and “Galway”, Kevin MacLeod (incompetech.com), Licensed under Creative Commons: By Attribution 3.0, <http://creativecommons.org/licenses/by/3.0/>

## RESOURCES

- Third party frameworks and libraries to develop audio applications with Espressif chips:
  - The [JOSH](#) operating system [supports the ESP32](#) and can be used in scenarios such as intelligent voice interaction, smart home appliances, and smart gateways.
- Third party audio development modules and boards that work with ESP-ADF:
  - [ESP32-A1S Audio Module](#) equipped CodeC audio decoding chip that supports music playback and recording, and 4MB PSRAM. The module application schematic is available in [datasheet](#).
- The [esp32.com forum](#) is a place to ask questions and find community resources. The forum has a section dedicated to [ESP-ADF](#).
- This [ESP Audio Development Framework](#) inherits from [ESP IoT Development Framework](#) and you can learn about it in [ESP-IDF Programming Guide](#).
- Check the [Issues](#) section on GitHub if you find a bug or have a feature request. Please check existing [Issues](#) before opening a new one.
- If you're interested in contributing to ESP Audio Development Framework, please check the [Contributions Guide](#).
- Several [books](#) have been written about ESP32 and they are listed on [Espressif](#) web site.
- For additional ESP32 product related information, please refer to [documentation](#) Section of Espressif site.
- To buy audio development boards, check list of distributors under [Get Samples](#) on Espressif web site.



## COPYRIGHTS AND LICENSES

### 5.1 Software Copyrights

All original source code in this repository is Copyright (C) 2015-2018 Espressif Systems. This source code is licensed under the ESPRESSIF MIT License as described in the file LICENSE.

Additional third party copyrighted code is included under the following licenses:

- `mp3` library is Copyright (c) 2005-2008, The Android Open Source Project, and is licensed under the Apache License Version 2.0.
- `aac` library is Copyright (c) 2005-2008, The Android Open Source Project, and is licensed under the Apache License Version 2.0.
- `amr` library is Copyright (C) 2009 Martin Storsjo and is licensed under the Apache License Version 2.0.
- `opus` library, Copyright 2001-2011 Xiph.Org, Skype Limited, Octasic, Jean-Marc Valin, Timothy B. Terriberry, CSIRO, Gregory Maxwell, Mark Borgerding, Erik de Castro Lopo, is licensed under 3-clause BSD license.
- `flac` library, Copyright (C) 2011-2016 Xiph.Org Foundation, is licensed under Xiph.Org's BSD-like license.
- `vorbis` library, Copyright (c) 2002-2020 Xiph.org Foundation, is licensed under the 3-Clause BSD license.
- `aac-enc` library is Copyright 2003-2010, VisualOn, and is licensed under the Apache License Version 2.0.
- `adpcm` library is Copyright (C) 2020 aikiriao <samuraiaiki@gmail.com>.
- `jpeg-dec` library is Copyright (C) 2019, ChaN.
- `alac` library is Copyright (C) 2004, developed by Apple, and is licensed under the Apache License Version 2.0.

Please refer to the [COPYRIGHT](#) in ESP-IDF Programming Guide

Where source code headers specify Copyright & License information, this information takes precedence over the summaries made here.



## ENGLISH-CHINESE GLOSSARY

This document lists terms that are used in Espressif Audio Development Framework Guide and other audio related documentation. Each term is followed by its Chinese equivalents and some have definitions.

**AAC** Chinese equivalent: AAC

Abbreviation for Advanced Audio Coding, an industry-standard audio compression format.

**acoustic** Chinese equivalent: 声学

**acoustic calibrator** Chinese equivalent: 声学校准器

Also known as sound level calibrator.

**acoustic echo cancellation** Chinese equivalent: 声学回声消除

Spelled-out form of AEC.

**AEC** Chinese equivalent: AEC

Abbreviation for acoustic echo cancellation.

**Advanced Audio Distribution Profile** Chinese equivalent: 高级音频分布配置文件

Spelled-out form of A2DP.

**A2DP** Chinese equivalent: A2DP.

Abbreviation for Advanced Audio Distribution Profile.

**A2DP sink** Chinese equivalent: A2DP sink

**A2DP source** Chinese equivalent: A2DP source

**AFE** Chinese equivalent: AFE

Abbreviation for audio front end. [Espressif audio front-end algorithm framework](#) is developed by Espressif AI Lab to provide high-quality and stable audio data to the host.

**AirKiss** Chinese equivalent: AirKiss

AirKiss is a quick-connection technique provided by Weixin device platform for Wi-Fi devices to configure network connection.

**AMR** Chinese equivalent: AMR

Abbreviation for Adaptive Multi-Rate, an audio compression format optimized for speech coding.

**AMR-NB** Chinese equivalent: AMR-NB

Abbreviation for Adaptive Multi-Rate Narrowband, a narrowband speech audio coding standard developed based on Adaptive Multi-Rate encoding.

**AMR-WB** Chinese equivalent: AMR-WB

Abbreviation for Adaptive Multi-Rate Wideband, a wideband speech audio coding standard developed based on Adaptive Multi-Rate encoding.

**analog-to-digital converter** Chinese equivalent: 模数转换器

Spelled-out form of ADC.

**ADC** Chinese equivalent: ADC

Abbreviation for analog-to-digital converter.

**audio codec** Chinese equivalent: 音频编解码器

**audio forge** Chinese equivalent: 音频合成器

A combination of several audio backend processing techniques, including resample, downmix, automatic level control, equalizer and sonic. Users can enable or disable certain techniques as needed.

**audio gate** Chinese equivalent: 音频门限

Spelled-out form of AG.

**AG** Chinese equivalent: AG

Abbreviation for audio gate.

**audio front end** Chinese equivalent: 音频前端

Spelled-out form of AFE.

**audio passthru** Chinese equivalent: 音频直通

Also known as pipeline passthru. It is an audio technique that allows audio files to pass through a pipeline unaltered.

**audio pipeline** Chinese equivalent: 音频流水线

Often used as “pipeline”. It is a chain of audio processing elements arranged in a particular order so that the output of each element is the input of the next.

**Audio Video Remote Control Profile** Chinese equivalent: 音视频遥控配置文件

Spelled-out form of AVRCP.

**AVRCP** Chinese equivalent: AVRCP

Abbreviation for Audio Video Remote Control Profile.

**automatic gain control** Chinese equivalent: 自动增益控制

Spelled-out form of AGC.

**AGC** Chinese equivalent: AGC

Abbreviation for automatic gain control.

**automatic level control** Chinese equivalent: 自动电平控制

Spelled-out form of ALC.

**ALC** Chinese equivalent: ALC

Abbreviation for automatic level control.

**automatic speech recognition** Chinese equivalent: 自动语音识别

Spelled-out form of ASR.



**aux cable** Chinese equivalent: 辅助线

Also known as auxiliary cable.

**ASR** Chinese equivalent: ASR

Abbreviation for automatic speech recognition.

**bandwidth** Chinese equivalent: 带宽

**Bass Frequency** Chinese equivalent: 低频

**BCLK** Chinese equivalent: BCLK

Abbreviation for base clock.

**BluFi** Chinese equivalent: BluFi

A Wi-Fi network configuration function via Bluetooth channel. See [ESP-IDF Programming Guide](#) for more information.

**cavity** Chinese equivalent: 腔体

**command word** Chinese equivalent: 命令词

**core dump** Chinese equivalent: 核心转储

**cutoff frequency** Chinese equivalent: 截止频率

**decoder** Chinese equivalent: 解码器

**digital media renderer** Chinese equivalent: 数字媒体渲染器

Spelled-out form of DMR.

**digital signal processor** Chinese equivalent: 数字信号处理器

Spelled-out form of DSP.

**DSP** Chinese equivalent: DSP

Abbreviation for digital signal processor or digital signal processing.

**digital-to-analog converter** Chinese equivalent: 数字-to-模拟转换器

Spelled-out form of DAC.

**dispatcher** Chinese equivalent: 调度器

**distortion** Chinese equivalent: 失真

**DAC** Chinese equivalent: DAC

Abbreviation for digital-to-analog converter.

**Digital Living Network Alliance** Chinese equivalent: 数字生活网络联盟

Spelled-out form of DLNA.

**DLNA** Chinese equivalent: DLNA

Abbreviation for Digital Living Network Alliance.

**DMR** Chinese equivalent: DMR

Abbreviation for digital media renderer.

**downmix** Chinese equivalent: 混音

An audio processing technique that mixes more audio streams to less output audio streams.

**DuerOS** Chinese equivalent: DuerOS

DuerOS is a conversational AI system developed by Baidu.

**echo** Chinese equivalent: 回音

A reflection of sound that arrives at the listener with a delay after the direct sound.

**echo reference signal** Chinese equivalent: 回音参考信号

**electret condenser microphone** Chinese equivalent: 驻极体电容麦克风

Spelled-out form of ECM.

**ECM** Chinese equivalent: ECM

Abbreviation for electret condenser microphone.

**element** Chinese equivalent: 元素

Also known as audio element. It is the basic building block for the application programmer developing with ADF. Every decoder, encoder, filter, input stream, or output stream is in fact an audio element.

**encoder** Chinese equivalent: 编码器

**equalizer** Chinese equivalent: 均衡器

**ESP VoIP** Chinese equivalent: ESP VoIP

ESP VoIP is a telephone client based on the standard SIP protocol, which can be used in some P2P or audio conference scenarios.

**fast Fourier transform** Chinese equivalent: 快速傅里叶变换

Spelled-out form of FFT.

**FFT** Chinese equivalent: FFT

Abbreviation for fast Fourier transform.

**FatFs** Chinese equivalent: FatFs

**FatFs stream** Chinese equivalent: FatFs 流

**FLAC** Chinese equivalent: FLAC

Abbreviation for Free Lossless Audio Codec, an audio coding format for lossless compression of digital audio.

**flexible pipeline** Chinese equivalent: 灵活管道

**FPS** Chinese equivalent: FPS

Abbreviation for frames per second.

**frames per second** Chinese equivalent: 帧每秒

Spelled-out form of FPS.

**frequency response** Chinese equivalent: 频率响应

**full band** Chinese equivalent: 全频带

Spelled-out form of FB.

**FB** Chinese equivalent: FB

Abbreviation for full band.

**Hands-Free** Chinese equivalent: 免提

Spelled-out form of HF.

**HF** Chinese equivalent: HF

Abbreviation for Hands-Free.

**Hands-Free Audio Gateway** Chinese equivalent: 免提音频网关

Spelled-out form of HFP-AG.

**Hands-Free Profile** Chinese equivalent: 免提配置

**Hands-Free Unit** Chinese equivalent: 免提单元

**HFP** Chinese equivalent: HFP

Abbreviation for Hands-Free Profile.

**hardware abstraction layer** Chinese equivalent: 硬件抽象层

Spelled-out form of HAL.

**HAL** Chinese equivalent: HAL

Abbreviation for hardware abstraction layer.

**headset** Chinese equivalent: 耳机

**HFP-AG** Chinese equivalent: HFP-AG

Abbreviation for Hands-Free Audio Gateway.

**Hi-Fi speaker** Chinese equivalent: 高保真扬声器

Also known as high-fidelity speaker.

**High Frequency** Chinese equivalent: 高频

**high-fidelity microphone** Chinese equivalent: 高保真麦克风

**HLS** Chinese equivalent: HLS

Abbreviation for HTTP Live Streaming.

**HTTP stream** Chinese equivalent: HTTP 流

**HTTP Live Streaming** Chinese equivalent: HTTP 直播

Spelled-out form of HLS.

**I2S stream** Chinese equivalent: I2S 流

**insertion loss** Chinese equivalent: 插入损耗

**Internet radio** Chinese equivalent: 网络收音机

**Internet of Things** Chinese equivalent: 物联网

**IoT** Chinese equivalent: IoT

Abbreviation for Internet of Things.

**JPEG** Chinese equivalent: JPEG

A commonly used method of lossy compression for digital images. Same as JPG.

**JPG** Chinese equivalent: JPG

A commonly used method of lossy compression for digital images. Same as JPEG.

**Light and Versatile Graphics Library** Chinese equivalent: 轻量级通用图形库

Spelled-out form of LVGL

**low-pass filter** Chinese equivalent: 低通滤波器

**LVGL** Chinese equivalent: LVGL

Abbreviation for Light and Versatile Graphics Library.

**M3U8** Chinese equivalent: M3U8

The Unicode version of M3U is M3U8, which uses UTF-8-encoded characters.

**M4A** Chinese equivalent: M4A

An audio encoding format for lossless compression of digital audio.

**mass production** Chinese equivalent: 量产

**maximum output power** Chinese equivalent: 最大输出功率

**MCLK** Chinese equivalent: MCLK

Abbreviation for master clock.

**mel-frequency cepstral coefficients** Chinese equivalent: 梅尔频率倒谱系数

Spelled-out form of MFCC.

**MFCC** Chinese equivalent: MFCC

Abbreviation for mel-frequency cepstral coefficients.

**microphone** Chinese equivalent: 麦克风

**mic** Chinese equivalent: 麦克风

Informal form for microphone.

**micro-electro-mechanical systems microphone** Chinese equivalent: 微机电系统麦克风

Spelled-out form of MEMS mic.

**MEMS mic** Chinese equivalent: MEMS 麦克风

Abbreviation for micro-electro-mechanical systems microphone.

**microphone gain** Chinese equivalent: 麦克风增益

**microphone hole** Chinese equivalent: 麦克风孔

**microSD card** Chinese equivalent: microSD 卡

**MP3** Chinese equivalent: MP3

**MP4** Chinese equivalent: MP4

**MultiNet** Chinese equivalent: MultiNet

**MultiNet** is a lightweight model specially designed based on **CRNN** and **CTC** for the implementation of multi-command recognition.

**multi-room** Chinese equivalent: 多房间

**Multi-Room Music** Chinese equivalent: Multi-Room Music

ESP Multi-Room Music is a Wi-Fi-based communication protocol to share music across multiple interconnected speakers. Under this protocol, those connected speakers form a Group. They can play music synchronously and are controlled together, which can easily achieve a theater-grade stereo surround sound system.

**narrowband** Chinese equivalent: 窄带

Spelled-out form of NB.

**NB** Chinese equivalent: NB

Abbreviation for narrowband.

**NimBLE** Chinese equivalent: NimBLE

An open-source Bluetooth Low Energy or Bluetooth Smart stack.

**noise criteria curve** Chinese equivalent: 噪声标准曲线

Also known as NC curve.

**noise rating curve** Chinese equivalent: 噪声评价曲线

Also known as NR curve.

**noise floor** Chinese equivalent: 噪声底

**noise suppression** Chinese equivalent: 噪声抑制

Spelled-out form of NS.

**NS** Chinese equivalent: NS

Abbreviation for noise suppression.

**non-volatile storage** Chinese equivalent: 非易失性存储

Spelled-out form of NVS.

**NVS** Chinese equivalent: NVS

Abbreviation for non-volatile storage.

**OGG** Chinese equivalent: OGG

An audio compression format.

**OPUS** Chinese equivalent: OPUS

A lossy audio coding format.

**PCM** Chinese equivalent: PCM

Abbreviation for pulse-code modulation.

**pixel** Chinese equivalent: 像素

**playback** Chinese equivalent: 回放

It is a noun. The verb is “play back”.

**programmable gain amplifier** Chinese equivalent: 可编程增益放大器

Spelled-out form of PGA.

**PGA** Chinese equivalent: PGA

Abbreviation for programmable gain amplifier.

**protractor** Chinese equivalent: 量角器

**pulse-code modulation** Chinese equivalent: 脉冲编码调制

Spelled-out form of PCM.

**raw stream** Chinese equivalent: 原始流

**resample** Chinese equivalent: 重采样

**resample filter** Chinese equivalent: 重采样滤波器

**resonant frequency** Chinese equivalent: 共振频率

**reverberation** Chinese equivalent: 混响

**RGB** Chinese equivalent: RGB

The RGB color model is an additive color model in which the red, green, and blue primary colors of light are added together in various ways to reproduce a broad array of colors.

**ring buffer** Chinese equivalent: 环形缓冲区

**SBC** Chinese equivalent: SBC

Abbreviation for subband codec.

**SD card** Chinese equivalent: SD 卡

**Session Initiation Protocol** Chinese equivalent: 会话发起协议

Spelled-out form of SIP.

**signal-to-echo ratio** Chinese equivalent: 回声比

**signal-to-noise ratio** Chinese equivalent: 信噪比

Spelled-out form of SNR.

**SIP** Chinese equivalent: SIP

Abbreviation for Session Initiation Protocol.

**SNR** Chinese equivalent: SNR

Abbreviation for signal-to-noise ratio.

**SmartConfig** Chinese equivalent: SmartConfig

The SmartConfig™ is a provisioning technology developed by TI to connect a new Wi-Fi device to a Wi-Fi network. It uses a mobile app to broadcast the network credentials from a smartphone, or a tablet, to an unprovisioned Wi-Fi device.

**sonic** Chinese equivalent: 声学

An audio processing technique that modifies sound frequency and speed.

**sound card** Chinese equivalent: 声卡

Also known as audio card.

**sound level meter** Chinese equivalent: 声级计

Also known as sound pressure level meter.

**sound pickup hole** Chinese equivalent: 拾音孔

**sound pickup tube** Chinese equivalent: 拾音管

**sound transmission loss** Chinese equivalent: 隔声量

Spelled-out form of STL.

**speech** Chinese equivalent: 语音

**speech recognition** Chinese equivalent: 语音识别

Spelled-out form of SR.

**SR** Chinese equivalent: SR

Abbreviation for speech recognition.

**SPI Flash File System** Chinese equivalent: SPI 闪存文件系统

Spelled-out form of SPIFFS.

**SPIFFS** Chinese equivalent: SPIFFS

Abbreviation for SPI Flash File System.

**SPIFFS stream** Chinese equivalent: SPIFFS 流

**subband codec** Chinese equivalent: 子带编解码器

Spelled-out form of SBC.

**STL** Chinese equivalent: STL

Abbreviation for sound transmission loss.

**super wide band** Chinese equivalent: 超宽带

Spelled-out form of SWB.

**SWB** Chinese equivalent: SWB

Abbreviation for super wide band.

**tape measure** Chinese equivalent: 磁带

**text-to-speech** Chinese equivalent: 文字转语音

Spelled-out form of TTS.

**TTS** Chinese equivalent: TTS

Abbreviation for text-to-speech.

**tolerance** Chinese equivalent: 容差

**tone** Chinese equivalent: 音调

**total harmonic distortion** Chinese equivalent: 总谐波失真

Spelled-out form of THD.

**THD** Chinese equivalent: THD

Abbreviation for total harmonic distortion.

**voice activity detection** Chinese equivalent: 语音活动检测

Spelled-out form of VAD.

**VAD** Chinese equivalent: VAD

Abbreviation for voice activity detection.

**VoIP** Chinese equivalent: VoIP

Abbreviation for Voice over Internet Protocol.

**wake word** Chinese equivalent: 唤醒词

**wake word engine** Chinese equivalent: 唤醒词引擎

Spelled-out form of WWE.

**WakeNet** Chinese equivalent: WakeNet

WakeNet is a wake word engine built upon neural network for low-power embedded MCUs.

**wake-up** Chinese equivalent: 唤醒

It is a noun.

**wideband** Chinese equivalent: 宽带

Spelled-out form of WB.

**WB** Chinese equivalent: WB

Abbreviation for wideband.

**WWE** Chinese equivalent: WVE

Abbreviation for wake word engine.

**YUV** Chinese equivalent: YUV

A color model typically used as part of a color image pipeline.



ABOUT

This is documentation of **ESP-ADF**, the framework to develop audio applications for **ESP32** chip by **Espressif**.

The **ESP32** is 2.4 GHz Wi-Fi and Bluetooth combo, 32 bit dual core chip running up to 240 MHz, designed for mobile, wearable electronics, and Internet-of-Things (IoT) applications. It has several peripherals on board including I2S interfaces to easily integrate with dedicated audio chips. These hardware features together with the ESP-ADF software provide a powerful platform to implement audio applications including native wireless networking and powerful user interface.

The **ESP-ADF** provides a range of API components including **Audio Streams**, **Codecs** and **Services** organized in **Audio Pipeline**, all integrated with audio hardware through **Media HAL** and with **Peripherals** onboard of **ESP32**.

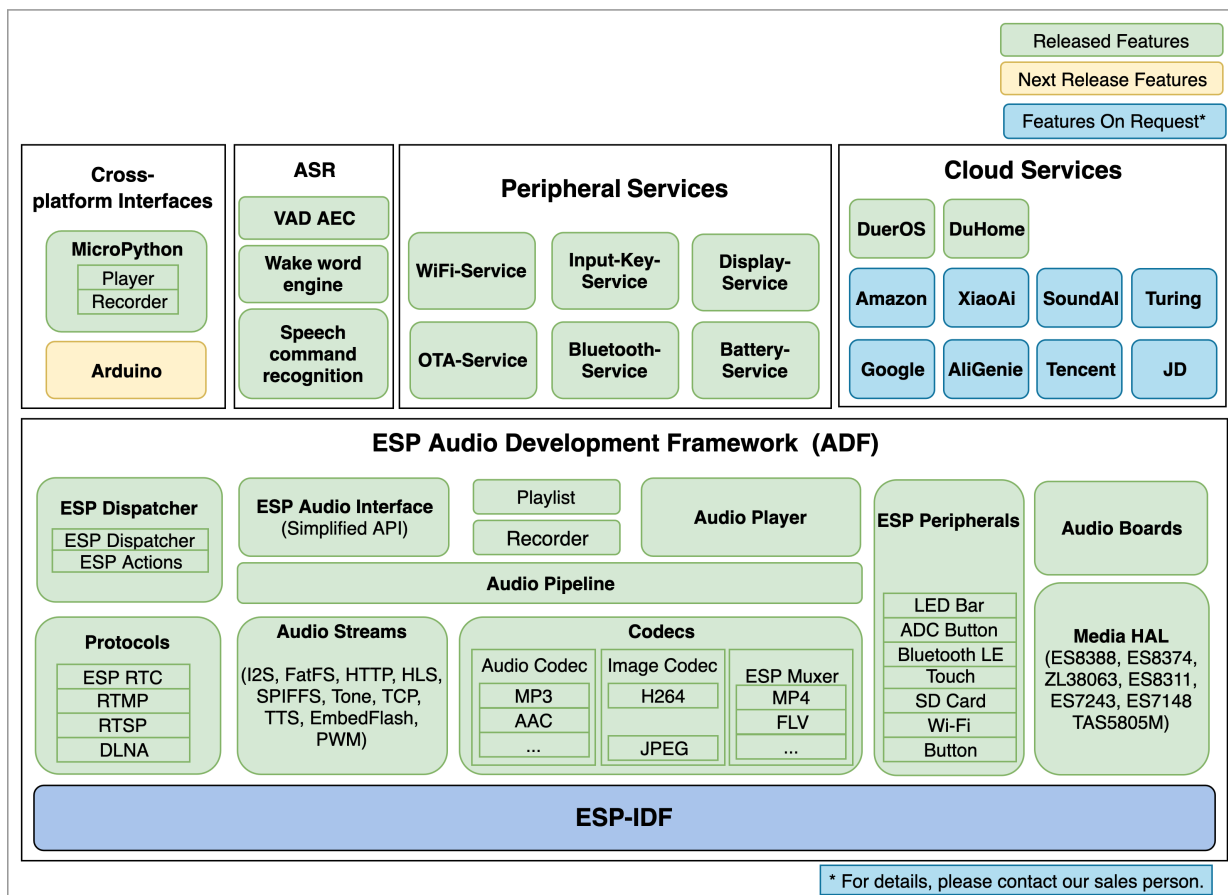


Fig. 1: Espressif Audio Development Framework

The ESP-ADF also provides integration with **Baidu DuerOS** cloud services. A range of components is coming to

provide integration with DeepBrain, Amazon, Google, Alibaba and Turing cloud services.

The **ESP-ADF** builds on well established, FreeRTOS based, Espressif IOT Development Framework [ESP-IDF](#).

- `genindex`

## A

- A2DP, [427](#)
- A2DP sink, [427](#)
- A2DP source, [427](#)
- AAC, [427](#)
- aac\_decoder\_cfg\_t (C++ struct), [120](#)
- aac\_decoder\_cfg\_t::out\_rb\_size (C++ member), [120](#)
- aac\_decoder\_cfg\_t::plus\_enable (C++ member), [120](#)
- aac\_decoder\_cfg\_t::stack\_in\_ext (C++ member), [120](#)
- aac\_decoder\_cfg\_t::task\_core (C++ member), [120](#)
- aac\_decoder\_cfg\_t::task\_prio (C++ member), [120](#)
- aac\_decoder\_cfg\_t::task\_stack (C++ member), [120](#)
- aac\_decoder\_init (C++ function), [120](#)
- AAC\_DECODER\_RINGBUFFER\_SIZE (C macro), [121](#)
- AAC\_DECODER\_TASK\_CORE (C macro), [121](#)
- AAC\_DECODER\_TASK\_PRIO (C macro), [121](#)
- AAC\_DECODER\_TASK\_STACK\_SIZE (C macro), [121](#)
- acoustic, [427](#)
- acoustic calibrator, [427](#)
- acoustic echo cancellation, [427](#)
- ADC, [428](#)
- ADC\_BUTTON\_STACK\_SIZE (C macro), [236](#)
- ADC\_BUTTON\_TASK\_CORE\_ID (C macro), [236](#)
- ADC\_BUTTON\_TASK\_PRIORITY (C macro), [236](#)
- ADC\_DEFAULT\_ARR (C macro), [236](#)
- Advanced Audio Distribution Profile, [427](#)
- AEC, [427](#)
- AFE, [427](#)
- AG, [428](#)
- AGC, [428](#)
- AirKiss, [427](#)
- airkiss\_config\_create (C++ function), [165](#)
- AIRKISS\_CONFIG\_INFO\_DEFAULT (C macro), [166](#)
- airkiss\_config\_info\_t (C++ struct), [166](#)
- airkiss\_config\_info\_t::aes\_key (C++ member), [166](#)
- airkiss\_config\_info\_t::lan\_pack (C++ member), [166](#)
- airkiss\_config\_info\_t::ssdp\_notify\_enable (C++ member), [166](#)
- airkiss\_lan\_pack\_param\_t (C++ struct), [166](#)
- airkiss\_lan\_pack\_param\_t::appid (C++ member), [166](#)
- airkiss\_lan\_pack\_param\_t::deviceid (C++ member), [166](#)
- ALC, [428](#)
- algo\_stream\_init (C++ function), [71](#)
- algo\_stream\_set\_delay (C++ function), [71](#)
- algorithm\_mono\_fix (C++ function), [71](#)
- ALGORITHM\_STREAM\_CFG\_DEFAULT (C macro), [73](#)
- algorithm\_stream\_cfg\_t (C++ struct), [72](#)
- algorithm\_stream\_cfg\_t::aec\_low\_cost (C++ member), [73](#)
- algorithm\_stream\_cfg\_t::agc\_gain (C++ member), [73](#)
- algorithm\_stream\_cfg\_t::algo\_mask (C++ member), [73](#)
- algorithm\_stream\_cfg\_t::debug\_input (C++ member), [72](#)
- algorithm\_stream\_cfg\_t::input\_type (C++ member), [72](#)
- algorithm\_stream\_cfg\_t::mic\_ch (C++ member), [73](#)
- algorithm\_stream\_cfg\_t::out\_rb\_size (C++ member), [72](#)
- algorithm\_stream\_cfg\_t::partition\_label (C++ member), [73](#)
- algorithm\_stream\_cfg\_t::rec\_linear\_factor (C++ member), [72](#)
- algorithm\_stream\_cfg\_t::ref\_linear\_factor (C++ member), [72](#)
- algorithm\_stream\_cfg\_t::sample\_rate (C++ member), [73](#)
- algorithm\_stream\_cfg\_t::stack\_in\_ext (C++ member), [72](#)
- algorithm\_stream\_cfg\_t::swap\_ch (C++ member), [72](#)
- algorithm\_stream\_cfg\_t::task\_core (C++

- member*), 72
- algorithm\_stream\_cfg\_t::task\_prio (C++ *member*), 72
- algorithm\_stream\_cfg\_t::task\_stack (C++ *member*), 72
- ALGORITHM\_STREAM\_DEFAULT\_AGC\_GAIN\_DB (C *macro*), 73
- ALGORITHM\_STREAM\_DEFAULT\_MASK (C *macro*), 73
- ALGORITHM\_STREAM\_DEFAULT\_MIC\_CHANNELS (C *macro*), 73
- ALGORITHM\_STREAM\_DEFAULT\_SAMPLE\_BIT (C *macro*), 73
- ALGORITHM\_STREAM\_DEFAULT\_SAMPLE\_RATE\_HZ (C *macro*), 73
- algorithm\_stream\_input\_type\_t (C++ *enum*), 74
- algorithm\_stream\_input\_type\_t::ALGORITHM\_STREAM\_INPUT\_TYPE\_A (C++ *enumerator*), 74
- algorithm\_stream\_input\_type\_t::ALGORITHM\_STREAM\_INPUT\_TYPE\_B (C++ *enumerator*), 74
- algorithm\_stream\_mask\_t (C++ *enum*), 74
- algorithm\_stream\_mask\_t::ALGORITHM\_STREAM\_USE\_AGC (C++ *enumerator*), 74
- algorithm\_stream\_mask\_t::ALGORITHM\_STREAM\_USE\_AGC2 (C++ *enumerator*), 74
- algorithm\_stream\_mask\_t::ALGORITHM\_STREAM\_USE\_AGC3 (C++ *enumerator*), 74
- algorithm\_stream\_mask\_t::ALGORITHM\_STREAM\_USE\_AGC4 (C++ *enumerator*), 74
- algorithm\_stream\_mask\_t::ALGORITHM\_STREAM\_USE\_AGC5 (C++ *enumerator*), 74
- ALGORITHM\_STREAM\_PINNED\_TO\_CORE (C *macro*), 73
- ALGORITHM\_STREAM\_RINGBUFFER\_SIZE (C *macro*), 73
- ALGORITHM\_STREAM\_TASK\_PERIOD (C *macro*), 73
- ALGORITHM\_STREAM\_TASK\_STACK\_SIZE (C *macro*), 73
- AMR, 427
- amr\_decoder\_cfg\_t (C++ *struct*), 121
- amr\_decoder\_cfg\_t::out\_rb\_size (C++ *member*), 122
- amr\_decoder\_cfg\_t::stack\_in\_ext (C++ *member*), 122
- amr\_decoder\_cfg\_t::task\_core (C++ *member*), 122
- amr\_decoder\_cfg\_t::task\_prio (C++ *member*), 122
- amr\_decoder\_cfg\_t::task\_stack (C++ *member*), 122
- amr\_decoder\_init (C++ *function*), 121
- AMR\_DECODER\_RINGBUFFER\_SIZE (C *macro*), 122
- AMR\_DECODER\_TASK\_CORE (C *macro*), 122
- AMR\_DECODER\_TASK\_PRIO (C *macro*), 122
- AMR\_DECODER\_TASK\_STACK\_SIZE (C *macro*), 122
- AMR-NB, 427
- AMR-WB, 428
- amrnb\_encoder\_bitrate\_t (C++ *enum*), 124
- amrnb\_encoder\_bitrate\_t::AMRNB\_ENC\_BITRATE\_MR102 (C++ *enumerator*), 124
- amrnb\_encoder\_bitrate\_t::AMRNB\_ENC\_BITRATE\_MR122 (C++ *enumerator*), 124
- amrnb\_encoder\_bitrate\_t::AMRNB\_ENC\_BITRATE\_MR475 (C++ *enumerator*), 124
- amrnb\_encoder\_bitrate\_t::AMRNB\_ENC\_BITRATE\_MR515 (C++ *enumerator*), 124
- amrnb\_encoder\_bitrate\_t::AMRNB\_ENC\_BITRATE\_MR59 (C++ *enumerator*), 124
- amrnb\_encoder\_bitrate\_t::AMRNB\_ENC\_BITRATE\_MR67 (C++ *enumerator*), 124
- amrnb\_encoder\_bitrate\_t::AMRNB\_ENC\_BITRATE\_MR74 (C++ *enumerator*), 124
- amrnb\_encoder\_bitrate\_t::AMRNB\_ENC\_BITRATE\_MR795 (C++ *enumerator*), 124
- amrnb\_encoder\_bitrate\_t::AMRNB\_ENC\_BITRATE\_MRDTX (C++ *enumerator*), 124
- amrnb\_encoder\_bitrate\_t::AMRNB\_ENC\_BITRATE\_N\_MODES (C++ *enumerator*), 124
- amrnb\_encoder\_bitrate\_t::AMRNB\_ENC\_BITRATE\_UNKNOW (C++ *enumerator*), 124
- amrnb\_encoder\_cfg\_t (C++ *struct*), 123
- amrnb\_encoder\_cfg\_t::bitrate\_mode (C++ *member*), 123
- amrnb\_encoder\_cfg\_t::contain\_amrnb\_header (C++ *member*), 123
- amrnb\_encoder\_cfg\_t::out\_rb\_size (C++ *member*), 123
- amrnb\_encoder\_cfg\_t::stack\_in\_ext (C++ *member*), 123
- amrnb\_encoder\_cfg\_t::task\_core (C++ *member*), 123
- amrnb\_encoder\_cfg\_t::task\_prio (C++ *member*), 123
- amrnb\_encoder\_cfg\_t::task\_stack (C++ *member*), 123
- amrnb\_encoder\_init (C++ *function*), 123
- AMRNB\_ENCODER\_RINGBUFFER\_SIZE (C *macro*), 124
- amrnb\_encoder\_set\_bitrate (C++ *function*), 122
- AMRNB\_ENCODER\_TASK\_CORE (C *macro*), 124
- AMRNB\_ENCODER\_TASK\_PRIO (C *macro*), 124
- AMRNB\_ENCODER\_TASK\_STACK (C *macro*), 124
- amrwb\_encoder\_bitrate\_t (C++ *enum*), 126
- amrwb\_encoder\_bitrate\_t::AMRWB\_ENC\_BITRATE\_MD1265 (C++ *enumerator*), 126
- amrwb\_encoder\_bitrate\_t::AMRWB\_ENC\_BITRATE\_MD1425 (C++ *enumerator*), 126
- amrwb\_encoder\_bitrate\_t::AMRWB\_ENC\_BITRATE\_MD1585 (C++ *enumerator*), 126

amrwb\_encoder\_bitrate\_t::AMRWB\_ENC\_BITRATE\_MD18 (C++ enumerator), 52  
 (C++ enumerator), 126 audio\_element\_abort\_input\_ringbuf (C++  
 function), 21  
 amrwb\_encoder\_bitrate\_t::AMRWB\_ENC\_BITRATE\_MD19 (C++ enumerator), 127 audio\_element\_abort\_output\_ringbuf (C++  
 function), 21  
 amrwb\_encoder\_bitrate\_t::AMRWB\_ENC\_BITRATE\_MD23 (C++ enumerator), 127 audio\_element\_cfg\_t (C++ struct), 33  
 amrwb\_encoder\_bitrate\_t::AMRWB\_ENC\_BITRATE\_MD28 audio\_element\_cfg\_t::buffer\_len (C++  
 member), 33  
 amrwb\_encoder\_bitrate\_t::AMRWB\_ENC\_BITRATE\_MD66 audio\_element\_cfg\_t::close (C++ member),  
 33  
 amrwb\_encoder\_bitrate\_t::AMRWB\_ENC\_BITRATE\_MD85 audio\_element\_cfg\_t::data (C++ member), 34  
 (C++ enumerator), 126 audio\_element\_cfg\_t::destroy (C++ mem-  
 ber), 33  
 amrwb\_encoder\_bitrate\_t::AMRWB\_ENC\_BITRATE\_MDNONE (C++ enumerator), 126 audio\_element\_cfg\_t::multi\_in\_rb\_num  
 (C++ member), 34  
 amrwb\_encoder\_bitrate\_t::AMRWB\_ENC\_BITRATE\_N\_MONO (C++ enumerator), 127 audio\_element\_cfg\_t::multi\_out\_rb\_num  
 (C++ member), 34  
 amrwb\_encoder\_cfg\_t (C++ struct), 125 audio\_element\_cfg\_t::open (C++ member), 33  
 amrwb\_encoder\_cfg\_t::bitrate\_mode (C++ member), 125 audio\_element\_cfg\_t::out\_rb\_size (C++  
 member), 33  
 amrwb\_encoder\_cfg\_t::contain\_amrwb\_header (C++ member), 125 audio\_element\_cfg\_t::process (C++ mem-  
 ber), 33  
 amrwb\_encoder\_cfg\_t::out\_rb\_size (C++ member), 125 audio\_element\_cfg\_t::read (C++ member), 33  
 amrwb\_encoder\_cfg\_t::stack\_in\_ext (C++ member), 126 audio\_element\_cfg\_t::seek (C++ member), 33  
 amrwb\_encoder\_cfg\_t::task\_core (C++ member), 125 audio\_element\_cfg\_t::stack\_in\_ext (C++  
 member), 34  
 amrwb\_encoder\_cfg\_t::task\_prio (C++ member), 125 audio\_element\_cfg\_t::tag (C++ member), 34  
 amrwb\_encoder\_cfg\_t::task\_stack (C++ member), 125 audio\_element\_cfg\_t::task\_core (C++ mem-  
 ber), 33  
 amrwb\_encoder\_init (C++ function), 125 audio\_element\_cfg\_t::task\_prio (C++ mem-  
 ber), 33  
 AMRWB\_ENCODER\_RINGBUFFER\_SIZE (C macro), 126 audio\_element\_cfg\_t::task\_stack (C++  
 member), 33  
 amrwb\_encoder\_set\_bitrate (C++ function), 125 audio\_element\_cfg\_t::write (C++ member),  
 33  
 AMRWB\_ENCODER\_TASK\_CORE (C macro), 126 audio\_element\_change\_cmd (C++ function), 23  
 AMRWB\_ENCODER\_TASK\_PRIO (C macro), 126 audio\_element\_deinit (C++ function), 16  
 AMRWB\_ENCODER\_TASK\_STACK (C macro), 126 audio\_element\_err\_t (C++ enum), 35  
 analog-to-digital converter, 428 audio\_element\_err\_t::AEL\_IO\_ABORT (C++  
 enumerator), 35  
 ASR, 429 audio\_element\_err\_t::AEL\_IO\_DONE (C++  
 enumerator), 35  
 audio codec, 428 audio\_element\_err\_t::AEL\_IO\_FAIL (C++  
 enumerator), 35  
 audio forge, 428 audio\_element\_err\_t::AEL\_IO\_OK (C++ enu-  
 merator), 35  
 audio front end, 428 audio\_element\_err\_t::AEL\_IO\_TIMEOUT  
 (C++ enumerator), 35  
 audio gate, 428 audio\_element\_err\_t::AEL\_PROCESS\_FAIL  
 (C++ enumerator), 35  
 audio passthru, 428 audio\_element\_finish\_state (C++ function),  
 23  
 audio pipeline, 428 audio\_element\_get\_event\_queue (C++ func-  
 tion), 25  
 Audio Video Remote Control Profile, 428  
 audio\_codec\_type\_t (C++ enum), 52  
 audio\_codec\_type\_t::AUDIO\_CODEC\_TYPE\_DECODER (C++ enumerator), 53  
 audio\_codec\_type\_t::AUDIO\_CODEC\_TYPE\_ENCODER (C++ enumerator), 53  
 audio\_codec\_type\_t::AUDIO\_CODEC\_TYPE\_NONE (C++ enumerator), 52

audio\_element\_get\_input\_ringbuf (C++ function), 20  
 audio\_element\_get\_multi\_input\_ringbuf (C++ function), 27  
 audio\_element\_get\_multi\_output\_ringbuf (C++ function), 27  
 audio\_element\_get\_output\_ringbuf (C++ function), 21  
 audio\_element\_get\_output\_ringbuf\_size (C++ function), 26  
 audio\_element\_get\_read\_cb (C++ function), 25  
 audio\_element\_get\_state (C++ function), 21  
 audio\_element\_get\_tag (C++ function), 17  
 audio\_element\_get\_uri (C++ function), 18  
 audio\_element\_get\_write\_cb (C++ function), 25  
 audio\_element\_getdata (C++ function), 17  
 audio\_element\_getinfo (C++ function), 17  
 audio\_element\_handle\_t (C++ type), 34  
 AUDIO\_ELEMENT\_INFO\_DEFAULT (C macro), 34  
 audio\_element\_info\_t (C++ struct), 32  
 audio\_element\_info\_t::bits (C++ member), 32  
 audio\_element\_info\_t::bps (C++ member), 32  
 audio\_element\_info\_t::byte\_pos (C++ member), 32  
 audio\_element\_info\_t::channels (C++ member), 32  
 audio\_element\_info\_t::codec\_fmt (C++ member), 32  
 audio\_element\_info\_t::duration (C++ member), 32  
 audio\_element\_info\_t::reserve\_data (C++ member), 33  
 audio\_element\_info\_t::sample\_rates (C++ member), 32  
 audio\_element\_info\_t::total\_bytes (C++ member), 32  
 audio\_element\_info\_t::uri (C++ member), 32  
 audio\_element\_init (C++ function), 16  
 audio\_element\_input (C++ function), 24  
 audio\_element\_is\_stopping (C++ function), 28  
 audio\_element\_msg\_cmd\_t (C++ enum), 36  
 audio\_element\_msg\_cmd\_t::AEL\_MSG\_CMD\_DESTROY (C++ enumerator), 36  
 audio\_element\_msg\_cmd\_t::AEL\_MSG\_CMD\_FINALISE (C++ enumerator), 36  
 audio\_element\_msg\_cmd\_t::AEL\_MSG\_CMD\_NONE (C++ enumerator), 36  
 audio\_element\_msg\_cmd\_t::AEL\_MSG\_CMD\_PAUSE (C++ enumerator), 36  
 audio\_element\_msg\_cmd\_t::AEL\_MSG\_CMD\_REPORT\_CODEC\_FMT (C++ enumerator), 36  
 audio\_element\_msg\_cmd\_t::AEL\_MSG\_CMD\_REPORT\_POSITION (C++ enumerator), 36  
 audio\_element\_msg\_cmd\_t::AEL\_MSG\_CMD\_REPORT\_STATUS (C++ enumerator), 36  
 audio\_element\_msg\_cmd\_t::AEL\_MSG\_CMD\_RESUME (C++ enumerator), 36  
 audio\_element\_msg\_cmd\_t::AEL\_MSG\_CMD\_STOP (C++ enumerator), 36  
 audio\_element\_msg\_remove\_listener (C++ function), 20  
 audio\_element\_msg\_set\_listener (C++ function), 19  
 audio\_element\_multi\_input (C++ function), 26  
 audio\_element\_multi\_output (C++ function), 26  
 audio\_element\_output (C++ function), 24  
 audio\_element\_pause (C++ function), 19  
 audio\_element\_process\_deinit (C++ function), 28  
 audio\_element\_process\_init (C++ function), 28  
 audio\_element\_report\_codec\_fmt (C++ function), 22  
 audio\_element\_report\_info (C++ function), 22  
 audio\_element\_report\_pos (C++ function), 22  
 audio\_element\_report\_status (C++ function), 22  
 audio\_element\_reserve\_data\_t (C++ struct), 31  
 audio\_element\_reserve\_data\_t::user\_data\_0 (C++ member), 32  
 audio\_element\_reserve\_data\_t::user\_data\_1 (C++ member), 32  
 audio\_element\_reserve\_data\_t::user\_data\_2 (C++ member), 32  
 audio\_element\_reserve\_data\_t::user\_data\_3 (C++ member), 32  
 audio\_element\_reserve\_data\_t::user\_data\_4 (C++ member), 32  
 audio\_element\_reset\_input\_ringbuf (C++ function), 23  
 audio\_element\_reset\_output\_ringbuf (C++ function), 23  
 audio\_element\_reset\_state (C++ function), 25  
 audio\_element\_resume (C++ function), 19  
 audio\_element\_run (C++ function), 18  
 audio\_element\_seek (C++ function), 28  
 audio\_element\_set\_bps (C++ function), 29  
 audio\_element\_set\_byte\_pos (C++ function), 29  
 audio\_element\_set\_codec\_fmt (C++ function), 29  
 audio\_element\_set\_duration (C++ function),

- 30
- audio\_element\_set\_event\_callback (C++ function), 20
- audio\_element\_set\_input\_ringbuf (C++ function), 20
- audio\_element\_set\_input\_timeout (C++ function), 22
- audio\_element\_set\_multi\_input\_ringbuf (C++ function), 26
- audio\_element\_set\_multi\_output\_ringbuf (C++ function), 27
- audio\_element\_set\_music\_info (C++ function), 30
- audio\_element\_set\_output\_ringbuf (C++ function), 21
- audio\_element\_set\_output\_ringbuf\_size (C++ function), 26
- audio\_element\_set\_output\_timeout (C++ function), 23
- audio\_element\_set\_read\_cb (C++ function), 24
- audio\_element\_set\_reserve\_user0 (C++ function), 30
- audio\_element\_set\_reserve\_user1 (C++ function), 30
- audio\_element\_set\_reserve\_user2 (C++ function), 31
- audio\_element\_set\_reserve\_user3 (C++ function), 31
- audio\_element\_set\_reserve\_user4 (C++ function), 31
- audio\_element\_set\_ringbuf\_done (C++ function), 25
- audio\_element\_set\_tag (C++ function), 17
- audio\_element\_set\_total\_bytes (C++ function), 29
- audio\_element\_set\_uri (C++ function), 17
- audio\_element\_set\_write\_cb (C++ function), 24
- audio\_element\_setdata (C++ function), 16
- audio\_element\_setinfo (C++ function), 17
- audio\_element\_state\_t (C++ enum), 35
- audio\_element\_state\_t::AEL\_STATE\_ERROR (C++ enumerator), 35
- audio\_element\_state\_t::AEL\_STATE\_FINISHED (C++ enumerator), 35
- audio\_element\_state\_t::AEL\_STATE\_INIT (C++ enumerator), 35
- audio\_element\_state\_t::AEL\_STATE\_INITIALIZING (C++ enumerator), 35
- audio\_element\_state\_t::AEL\_STATE\_NONE (C++ enumerator), 35
- audio\_element\_state\_t::AEL\_STATE\_PAUSED (C++ enumerator), 35
- audio\_element\_state\_t::AEL\_STATE\_RUNNING (C++ enumerator), 35
- audio\_element\_state\_t::AEL\_STATE\_STOPPED (C++ enumerator), 35
- audio\_element\_status\_t (C++ enum), 36
- audio\_element\_status\_t::AEL\_STATUS\_ERROR\_CLOSE (C++ enumerator), 36
- audio\_element\_status\_t::AEL\_STATUS\_ERROR\_INPUT (C++ enumerator), 36
- audio\_element\_status\_t::AEL\_STATUS\_ERROR\_OPEN (C++ enumerator), 36
- audio\_element\_status\_t::AEL\_STATUS\_ERROR\_OUTPUT (C++ enumerator), 36
- audio\_element\_status\_t::AEL\_STATUS\_ERROR\_PROCESS (C++ enumerator), 36
- audio\_element\_status\_t::AEL\_STATUS\_ERROR\_TIMEOUT (C++ enumerator), 36
- audio\_element\_status\_t::AEL\_STATUS\_ERROR\_UNKNOWN (C++ enumerator), 37
- audio\_element\_status\_t::AEL\_STATUS\_INPUT\_BUFFERING (C++ enumerator), 37
- audio\_element\_status\_t::AEL\_STATUS\_INPUT\_DONE (C++ enumerator), 37
- audio\_element\_status\_t::AEL\_STATUS\_MOUNTED (C++ enumerator), 37
- audio\_element\_status\_t::AEL\_STATUS\_NONE (C++ enumerator), 36
- audio\_element\_status\_t::AEL\_STATUS\_OUTPUT\_BUFFERING (C++ enumerator), 37
- audio\_element\_status\_t::AEL\_STATUS\_OUTPUT\_DONE (C++ enumerator), 37
- audio\_element\_status\_t::AEL\_STATUS\_STATE\_FINISHED (C++ enumerator), 37
- audio\_element\_status\_t::AEL\_STATUS\_STATE\_PAUSED (C++ enumerator), 37
- audio\_element\_status\_t::AEL\_STATUS\_STATE\_RUNNING (C++ enumerator), 37
- audio\_element\_status\_t::AEL\_STATUS\_STATE\_STOPPED (C++ enumerator), 37
- audio\_element\_status\_t::AEL\_STATUS\_UNMOUNTED (C++ enumerator), 37
- audio\_element\_stop (C++ function), 18
- audio\_element\_terminate (C++ function), 18
- audio\_element\_terminate\_with\_ticks (C++ function), 18
- audio\_element\_type\_t (C++ enum), 52
- audio\_element\_type\_t::AUDIO\_ELEMENT\_TYPE\_ELEMENT (C++ enumerator), 52
- audio\_element\_type\_t::AUDIO\_ELEMENT\_TYPE\_PERIPH (C++ enumerator), 52
- audio\_element\_type\_t::AUDIO\_ELEMENT\_TYPE\_PLAYER (C++ enumerator), 52
- audio\_element\_type\_t::AUDIO\_ELEMENT\_TYPE\_SERVICE (C++ enumerator), 52
- audio\_element\_type\_t::AUDIO\_ELEMENT\_TYPE\_UNKNOWN (C++ enumerator), 52

- `(C++ enumerator)`, 52
- `audio_element_update_byte_pos` (C++ function), 28
- `audio_element_update_total_bytes` (C++ function), 29
- `audio_element_wait_for_buffer` (C++ function), 21
- `audio_element_wait_for_stop` (C++ function), 19
- `audio_element_wait_for_stop_ms` (C++ function), 19
- `audio_err_t` (C++ enum), 54
- `audio_err_t::ESP_ERR_AUDIO_ALREADY_EXISTS` (C++ enumerator), 54
- `audio_err_t::ESP_ERR_AUDIO_CLOSE` (C++ enumerator), 55
- `audio_err_t::ESP_ERR_AUDIO_FAIL` (C++ enumerator), 54
- `audio_err_t::ESP_ERR_AUDIO_HAL_FAIL` (C++ enumerator), 54
- `audio_err_t::ESP_ERR_AUDIO_INPUT` (C++ enumerator), 55
- `audio_err_t::ESP_ERR_AUDIO_INVALID_PARAMETER` (C++ enumerator), 54
- `audio_err_t::ESP_ERR_AUDIO_INVALID_PATH` (C++ enumerator), 54
- `audio_err_t::ESP_ERR_AUDIO_INVALID_URI` (C++ enumerator), 54
- `audio_err_t::ESP_ERR_AUDIO_LINK_FAIL` (C++ enumerator), 54
- `audio_err_t::ESP_ERR_AUDIO_MEMORY_LACK` (C++ enumerator), 54
- `audio_err_t::ESP_ERR_AUDIO_NO_CODEC` (C++ enumerator), 54
- `audio_err_t::ESP_ERR_AUDIO_NO_ERROR` (C++ enumerator), 54
- `audio_err_t::ESP_ERR_AUDIO_NO_INPUT_STREAM` (C++ enumerator), 54
- `audio_err_t::ESP_ERR_AUDIO_NO_OUTPUT_STREAM` (C++ enumerator), 54
- `audio_err_t::ESP_ERR_AUDIO_NOT_READY` (C++ enumerator), 54
- `audio_err_t::ESP_ERR_AUDIO_NOT_SUPPORT` (C++ enumerator), 54
- `audio_err_t::ESP_ERR_AUDIO_OPEN` (C++ enumerator), 55
- `audio_err_t::ESP_ERR_AUDIO_OUT_OF_RANGE` (C++ enumerator), 55
- `audio_err_t::ESP_ERR_AUDIO_OUTPUT` (C++ enumerator), 55
- `audio_err_t::ESP_ERR_AUDIO_PROCESS` (C++ enumerator), 55
- `audio_err_t::ESP_ERR_AUDIO_STOP_BY_USER` (C++ enumerator), 55
- `audio_err_t::ESP_ERR_AUDIO_TIMEOUT` (C++ enumerator), 54
- `audio_err_t::ESP_ERR_AUDIO_UNKNOWN` (C++ enumerator), 55
- `audio_event_iface_cfg_t` (C++ struct), 50
- `audio_event_iface_cfg_t::context` (C++ member), 51
- `audio_event_iface_cfg_t::external_queue_size` (C++ member), 51
- `audio_event_iface_cfg_t::internal_queue_size` (C++ member), 51
- `audio_event_iface_cfg_t::on_cmd` (C++ member), 51
- `audio_event_iface_cfg_t::queue_set_size` (C++ member), 51
- `audio_event_iface_cfg_t::type` (C++ member), 51
- `audio_event_iface_cfg_t::wait_time` (C++ member), 51
- `audio_event_iface_cmd` (C++ function), 48
- `audio_event_iface_cmd_from_isr` (C++ function), 48
- `AUDIO_EVENT_IFACE_DEFAULT_CFG` (C macro), 51
- `audio_event_iface_destroy` (C++ function), 47
- `audio_event_iface_discard` (C++ function), 49
- `audio_event_iface_get_msg_queue_handle` (C++ function), 50
- `audio_event_iface_get_queue_handle` (C++ function), 49
- `audio_event_iface_handle_t` (C++ type), 51
- `audio_event_iface_init` (C++ function), 47
- `audio_event_iface_listen` (C++ function), 49
- `audio_event_iface_msg_t` (C++ struct), 50
- `audio_event_iface_msg_t::cmd` (C++ member), 50
- `audio_event_iface_msg_t::data` (C++ member), 50
- `audio_event_iface_msg_t::data_len` (C++ member), 50
- `audio_event_iface_msg_t::need_free_data` (C++ member), 50
- `audio_event_iface_msg_t::source` (C++ member), 50
- `audio_event_iface_msg_t::source_type` (C++ member), 50
- `audio_event_iface_read` (C++ function), 49
- `audio_event_iface_remove_listener` (C++ function), 47
- `audio_event_iface_sendout` (C++ function), 49
- `audio_event_iface_set_cmd_waiting_timeout` (C++ function), 48
- `audio_event_iface_set_listener` (C++ function), 47
- `audio_event_iface_set_msg_listener` (C++



- function*), 50
- audio\_event\_iface\_waiting\_cmd\_msg (C++ *function*), 48
- audio\_hal (C++ *struct*), 248
- audio\_hal::audio\_codec\_config\_iface (C++ *member*), 248
- audio\_hal::audio\_codec\_ctrl (C++ *member*), 248
- audio\_hal::audio\_codec\_deinitialize (C++ *member*), 248
- audio\_hal::audio\_codec\_enable\_pa (C++ *member*), 248
- audio\_hal::audio\_codec\_get\_volume (C++ *member*), 248
- audio\_hal::audio\_codec\_initialize (C++ *member*), 248
- audio\_hal::audio\_codec\_set\_mute (C++ *member*), 248
- audio\_hal::audio\_codec\_set\_volume (C++ *member*), 248
- audio\_hal::audio\_hal\_lock (C++ *member*), 248
- audio\_hal::handle (C++ *member*), 248
- audio\_hal\_adc\_input\_t (C++ *enum*), 249
- audio\_hal\_adc\_input\_t::AUDIO\_HAL\_ADC\_INPUT\_ALL (C++ *enumerator*), 249
- audio\_hal\_adc\_input\_t::AUDIO\_HAL\_ADC\_INPUT\_DIFFERENCE (C++ *enumerator*), 249
- audio\_hal\_adc\_input\_t::AUDIO\_HAL\_ADC\_INPUT\_LINE1 (C++ *enumerator*), 249
- audio\_hal\_adc\_input\_t::AUDIO\_HAL\_ADC\_INPUT\_LINE2 (C++ *enumerator*), 249
- audio\_hal\_codec\_config\_t (C++ *struct*), 247
- audio\_hal\_codec\_config\_t::adc\_input (C++ *member*), 247
- audio\_hal\_codec\_config\_t::codec\_mode (C++ *member*), 247
- audio\_hal\_codec\_config\_t::dac\_output (C++ *member*), 247
- audio\_hal\_codec\_config\_t::i2s\_iface (C++ *member*), 247
- audio\_hal\_codec\_i2s\_iface\_t (C++ *struct*), 247
- audio\_hal\_codec\_i2s\_iface\_t::bits (C++ *member*), 247
- audio\_hal\_codec\_i2s\_iface\_t::fmt (C++ *member*), 247
- audio\_hal\_codec\_i2s\_iface\_t::mode (C++ *member*), 247
- audio\_hal\_codec\_i2s\_iface\_t::samples (C++ *member*), 247
- audio\_hal\_codec\_iface\_config (C++ *function*), 245
- audio\_hal\_codec\_mode\_t (C++ *enum*), 249
- audio\_hal\_codec\_mode\_t::AUDIO\_HAL\_CODEC\_MODE\_BOTH (C++ *enumerator*), 249
- audio\_hal\_codec\_mode\_t::AUDIO\_HAL\_CODEC\_MODE\_DECODE (C++ *enumerator*), 249
- audio\_hal\_codec\_mode\_t::AUDIO\_HAL\_CODEC\_MODE\_ENCODE (C++ *enumerator*), 249
- audio\_hal\_codec\_mode\_t::AUDIO\_HAL\_CODEC\_MODE\_LINE1 (C++ *enumerator*), 249
- audio\_hal\_codec\_mode\_t::AUDIO\_HAL\_CODEC\_MODE\_LINE2 (C++ *enumerator*), 249
- audio\_hal\_ctrl\_codec (C++ *function*), 245
- audio\_hal\_ctrl\_t (C++ *enum*), 250
- audio\_hal\_ctrl\_t::AUDIO\_HAL\_CTRL\_START (C++ *enumerator*), 250
- audio\_hal\_ctrl\_t::AUDIO\_HAL\_CTRL\_STOP (C++ *enumerator*), 250
- audio\_hal\_dac\_output\_t (C++ *enum*), 250
- audio\_hal\_dac\_output\_t::AUDIO\_HAL\_DAC\_OUTPUT\_ALL (C++ *enumerator*), 250
- audio\_hal\_dac\_output\_t::AUDIO\_HAL\_DAC\_OUTPUT\_LINE1 (C++ *enumerator*), 250
- audio\_hal\_dac\_output\_t::AUDIO\_HAL\_DAC\_OUTPUT\_LINE2 (C++ *enumerator*), 250
- audio\_hal\_deinit (C++ *function*), 245
- audio\_hal\_enable\_pa (C++ *function*), 247
- audio\_hal\_func\_t (C++ *type*), 249
- audio\_hal\_get\_volume (C++ *function*), 246
- audio\_hal\_handle\_t (C++ *type*), 249
- audio\_hal\_iface\_bits\_t (C++ *enum*), 251
- audio\_hal\_iface\_bits\_t::AUDIO\_HAL\_BIT\_LENGTH\_16BITS (C++ *enumerator*), 251
- audio\_hal\_iface\_bits\_t::AUDIO\_HAL\_BIT\_LENGTH\_24BITS (C++ *enumerator*), 251
- audio\_hal\_iface\_bits\_t::AUDIO\_HAL\_BIT\_LENGTH\_32BITS (C++ *enumerator*), 251
- audio\_hal\_iface\_format\_t (C++ *enum*), 251
- audio\_hal\_iface\_format\_t::AUDIO\_HAL\_I2S\_DSP (C++ *enumerator*), 252
- audio\_hal\_iface\_format\_t::AUDIO\_HAL\_I2S\_LEFT (C++ *enumerator*), 251
- audio\_hal\_iface\_format\_t::AUDIO\_HAL\_I2S\_NORMAL (C++ *enumerator*), 251
- audio\_hal\_iface\_format\_t::AUDIO\_HAL\_I2S\_RIGHT (C++ *enumerator*), 251
- audio\_hal\_iface\_mode\_t (C++ *enum*), 250
- audio\_hal\_iface\_mode\_t::AUDIO\_HAL\_MODE\_MASTER (C++ *enumerator*), 250
- audio\_hal\_iface\_mode\_t::AUDIO\_HAL\_MODE\_SLAVE (C++ *enumerator*), 250
- audio\_hal\_iface\_samples\_t (C++ *enum*), 250
- audio\_hal\_iface\_samples\_t::AUDIO\_HAL\_08K\_SAMPLES (C++ *enumerator*), 250
- audio\_hal\_iface\_samples\_t::AUDIO\_HAL\_11K\_SAMPLES (C++ *enumerator*), 250
- audio\_hal\_iface\_samples\_t::AUDIO\_HAL\_16K\_SAMPLES (C++ *enumerator*), 251

audio\_hal\_iface\_samples\_t::AUDIO\_HAL\_22KasAMPLESpipeline\_reset\_ringbuffer (C++  
 (C++ enumerator), 251 function), 44  
 audio\_hal\_iface\_samples\_t::AUDIO\_HAL\_24KasAMPLESpipeline\_resume (C++ function), 40  
 (C++ enumerator), 251 audio\_pipeline\_run (C++ function), 39  
 audio\_hal\_iface\_samples\_t::AUDIO\_HAL\_32KasAMPLESpipeline\_set\_listener (C++ function),  
 (C++ enumerator), 251 42  
 audio\_hal\_iface\_samples\_t::AUDIO\_HAL\_44KasAMPLESpipeline\_stop (C++ function), 40  
 (C++ enumerator), 251 audio\_pipeline\_terminate (C++ function), 39  
 audio\_hal\_iface\_samples\_t::AUDIO\_HAL\_48KasAMPLESpipeline\_terminate\_with\_ticks  
 (C++ enumerator), 251 (C++ function), 39  
 audio\_hal\_init (C++ function), 245 audio\_pipeline\_unlink (C++ function), 41  
 audio\_hal\_set\_mute (C++ function), 246 audio\_pipeline\_unregister (C++ function), 38  
 audio\_hal\_set\_volume (C++ function), 246 audio\_pipeline\_unregister\_more (C++ func-  
 AUDIO\_HAL\_VOL\_DEFAULT (C macro), 249 tion), 43  
 audio\_pipeline\_breakup\_elements (C++ audio\_pipeline\_wait\_for\_stop (C++ func-  
 function), 44 tion), 40  
 audio\_pipeline\_cfg (C++ struct), 46 audio\_pipeline\_wait\_for\_stop\_with\_ticks  
 audio\_pipeline\_cfg::rb\_size (C++ member), (C++ function), 40  
 46 audio\_pwm\_config\_t (C++ struct), 86  
 audio\_pipeline\_cfg\_t (C++ type), 46 audio\_pwm\_config\_t::data\_len (C++ mem-  
 audio\_pipeline\_change\_state (C++ function), ber), 86  
 46 audio\_pwm\_config\_t::duty\_resolution  
 audio\_pipeline\_check\_items\_state (C++ (C++ member), 86  
 function), 43 audio\_pwm\_config\_t::gpio\_num\_left (C++  
 audio\_pipeline\_deinit (C++ function), 38 member), 86  
 audio\_pipeline\_get\_el\_by\_tag (C++ func- audio\_pwm\_config\_t::gpio\_num\_right (C++  
 tion), 41 member), 86  
 audio\_pipeline\_get\_el\_once (C++ function), audio\_pwm\_config\_t::ledc\_channel\_left  
 41 (C++ member), 86  
 audio\_pipeline\_get\_event\_iface (C++ func- audio\_pwm\_config\_t::ledc\_channel\_right  
 tion), 42 (C++ member), 86  
 audio\_pipeline\_handle\_t (C++ type), 46 audio\_pwm\_config\_t::ledc\_timer\_sel (C++  
 audio\_pipeline\_init (C++ function), 38 member), 86  
 audio\_pipeline\_link (C++ function), 41 audio\_pwm\_config\_t::tg\_num (C++ member),  
 audio\_pipeline\_link\_insert (C++ function), 86  
 42 audio\_pwm\_config\_t::timer\_num (C++ mem-  
 audio\_pipeline\_link\_more (C++ function), 43 ber), 86  
 audio\_pipeline\_listen\_more (C++ function), audio\_rec\_cfg\_t (C++ struct), 202  
 43 audio\_rec\_cfg\_t::encoder\_handle (C++  
 audio\_pipeline\_pause (C++ function), 40 member), 203  
 audio\_pipeline\_register (C++ function), 38 audio\_rec\_cfg\_t::encoder\_iface (C++ mem-  
 audio\_pipeline\_register\_more (C++ func- ber), 203  
 tion), 42 audio\_rec\_cfg\_t::event\_cb (C++ member),  
 audio\_pipeline\_relink (C++ function), 45 202  
 audio\_pipeline\_relink\_more (C++ function), audio\_rec\_cfg\_t::pinned\_core (C++ mem-  
 45 ber), 202  
 audio\_pipeline\_remove\_listener (C++ func- audio\_rec\_cfg\_t::read (C++ member), 202  
 tion), 42 audio\_rec\_cfg\_t::sr\_handle (C++ member),  
 audio\_pipeline\_reset\_elements (C++ func- 202  
 tion), 44 audio\_rec\_cfg\_t::sr\_iface (C++ member),  
 audio\_pipeline\_reset\_items\_state (C++ 202  
 function), 44 audio\_rec\_cfg\_t::task\_prio (C++ member),  
 audio\_pipeline\_reset\_kept\_state (C++ 202  
 function), 44 audio\_rec\_cfg\_t::task\_size (C++ member),

- 202
- audio\_rec\_cfg\_t::user\_data (C++ member), 202
- audio\_rec\_cfg\_t::vad\_off (C++ member), 203
- audio\_rec\_cfg\_t::vad\_start (C++ member), 202
- audio\_rec\_cfg\_t::wakeup\_end (C++ member), 203
- audio\_rec\_cfg\_t::wakeup\_time (C++ member), 202
- AUDIO\_REC\_DEF\_TASK\_CORE (C macro), 203
- AUDIO\_REC\_DEF\_TASK\_Prio (C macro), 203
- AUDIO\_REC\_DEF\_TASK\_SZ (C macro), 203
- AUDIO\_REC\_DEF\_VAD\_OFF\_TM (C macro), 203
- AUDIO\_REC\_DEF\_WAKEEND\_TM (C macro), 203
- AUDIO\_REC\_DEF\_WAKEUP\_TM (C macro), 203
- audio\_rec\_evt\_t (C++ struct), 201
- audio\_rec\_evt\_t::data\_len (C++ member), 202
- audio\_rec\_evt\_t::event\_data (C++ member), 202
- audio\_rec\_evt\_t::type (C++ member), 202
- audio\_rec\_evt\_t::[anonymous] (C++ enum), 201
- audio\_rec\_evt\_t::[anonymous]::AUDIO\_REC\_COMMAND\_DETECT (C++ enumerator), 201
- audio\_rec\_evt\_t::[anonymous]::AUDIO\_REC\_VAD\_END (C++ enumerator), 201
- audio\_rec\_evt\_t::[anonymous]::AUDIO\_REC\_VAD\_START (C++ enumerator), 201
- audio\_rec\_evt\_t::[anonymous]::AUDIO\_REC\_WAKEUP\_END (C++ enumerator), 201
- audio\_rec\_evt\_t::[anonymous]::AUDIO\_REC\_WAKEUP\_START (C++ enumerator), 201
- audio\_rec\_handle\_t (C++ type), 204
- AUDIO\_REC\_VAD\_START\_SPEECH\_MS (C macro), 203
- audio\_recorder\_create (C++ function), 199
- audio\_recorder\_data\_read (C++ function), 200
- AUDIO\_RECORDER\_DEFAULT\_CFG (C macro), 203
- audio\_recorder\_destroy (C++ function), 200
- audio\_recorder\_get\_wakeup\_state (C++ function), 201
- audio\_recorder\_multinet\_enable (C++ function), 200
- audio\_recorder\_trigger\_start (C++ function), 199
- audio\_recorder\_trigger\_stop (C++ function), 200
- audio\_recorder\_vad\_check\_enable (C++ function), 200
- audio\_recorder\_wakenet\_enable (C++ function), 200
- audio\_service\_callback (C++ function), 194
- audio\_service\_config\_t (C++ struct), 196
- audio\_service\_config\_t::service\_connect (C++ member), 196
- audio\_service\_config\_t::service\_destroy (C++ member), 196
- audio\_service\_config\_t::service\_disconnect (C++ member), 196
- audio\_service\_config\_t::service\_name (C++ member), 196
- audio\_service\_config\_t::service\_start (C++ member), 196
- audio\_service\_config\_t::service\_stop (C++ member), 196
- audio\_service\_config\_t::task\_core (C++ member), 196
- audio\_service\_config\_t::task\_func (C++ member), 196
- audio\_service\_config\_t::task\_prio (C++ member), 196
- audio\_service\_config\_t::task\_stack (C++ member), 196
- audio\_service\_config\_t::user\_data (C++ member), 197
- audio\_service\_connect (C++ function), 194
- audio\_service\_create (C++ function), 193
- audio\_service\_destroy (C++ function), 193
- audio\_service\_disconnect (C++ function), 195
- audio\_service\_get\_data (C++ function), 195
- audio\_service\_handle\_t (C++ type), 197
- audio\_service\_set\_callback (C++ function), 194
- audio\_service\_set\_data (C++ function), 195
- audio\_service\_start (C++ function), 194
- audio\_service\_stop (C++ function), 194
- audio\_stream\_type\_t (C++ enum), 52
- audio\_stream\_type\_t::AUDIO\_STREAM\_NONE (C++ enumerator), 52
- audio\_stream\_type\_t::AUDIO\_STREAM\_READER (C++ enumerator), 52
- audio\_stream\_type\_t::AUDIO\_STREAM\_WRITER (C++ enumerator), 52
- audio\_termination\_type\_t (C++ enum), 55
- audio\_termination\_type\_t::TERMINATION\_TYPE\_DONE (C++ enumerator), 55
- audio\_termination\_type\_t::TERMINATION\_TYPE\_MAX (C++ enumerator), 55
- audio\_termination\_type\_t::TERMINATION\_TYPE\_NOW (C++ enumerator), 55
- audio\_volume\_get (C++ type), 54
- audio\_volume\_set (C++ type), 54
- automatic gain control, 428
- automatic level control, 428
- automatic speech recognition, 428
- aux cable, 429

AVRCP, 428

aw2013\_led\_bar\_task (C++ function), 179

## B

bandwidth, 429

Bass Frequency, 429

battery\_service\_config\_t (C++ struct), 184

battery\_service\_config\_t::cb\_ctx (C++ member), 184

battery\_service\_config\_t::charger\_monitor (C++ member), 184

battery\_service\_config\_t::evt\_cb (C++ member), 184

battery\_service\_config\_t::extern\_stack (C++ member), 184

battery\_service\_config\_t::task\_core (C++ member), 184

battery\_service\_config\_t::task\_prio (C++ member), 184

battery\_service\_config\_t::task\_stack (C++ member), 184

battery\_service\_config\_t::vol\_monitor (C++ member), 184

battery\_service\_create (C++ function), 183

BATTERY\_SERVICE\_DEFAULT\_CONFIG (C macro), 184

battery\_service\_event\_t (C++ enum), 184

battery\_service\_event\_t::BAT\_SERV\_EVENT\_BAT\_FULL (C++ enumerator), 185

battery\_service\_event\_t::BAT\_SERV\_EVENT\_BAT\_LOW (C++ enumerator), 185

battery\_service\_event\_t::BAT\_SERV\_EVENT\_CHARGING\_BEGIN (C++ enumerator), 185

battery\_service\_event\_t::BAT\_SERV\_EVENT\_CHARGING\_STOP (C++ enumerator), 185

battery\_service\_event\_t::BAT\_SERV\_EVENT\_UNKNOWN (C++ enumerator), 184

battery\_service\_event\_t::BAT\_SERV\_EVENT\_VOLUME\_REPORT (C++ enumerator), 184

battery\_service\_set\_vol\_report\_freq (C++ function), 183

battery\_service\_vol\_report\_switch (C++ function), 183

BCLK, 429

BLUE\_LED\_MAX\_NUM (C macro), 239

BLUETOOTH\_ADDR\_LEN (C macro), 154

bluetooth\_addr\_t (C++ type), 154

bluetooth\_service\_cfg\_t (C++ struct), 153

bluetooth\_service\_cfg\_t::device\_name (C++ member), 153

bluetooth\_service\_cfg\_t::mode (C++ member), 153

bluetooth\_service\_cfg\_t::remote\_name (C++ member), 153

bluetooth\_service\_cfg\_t::user\_callback (C++ member), 153

bluetooth\_service\_create\_periph (C++ function), 150

bluetooth\_service\_create\_stream (C++ function), 150

bluetooth\_service\_destroy (C++ function), 152

bluetooth\_service\_mode\_t (C++ enum), 154

bluetooth\_service\_mode\_t::BLUETOOTH\_A2DP\_SINK (C++ enumerator), 154

bluetooth\_service\_mode\_t::BLUETOOTH\_A2DP\_SOURCE (C++ enumerator), 154

bluetooth\_service\_start (C++ function), 150

bluetooth\_service\_user\_cb\_t (C++ struct), 153

bluetooth\_service\_user\_cb\_t::user\_a2d\_cb (C++ member), 153

bluetooth\_service\_user\_cb\_t::user\_a2d\_sink\_data\_cb (C++ member), 153

bluetooth\_service\_user\_cb\_t::user\_a2d\_source\_data\_cb (C++ member), 153

bluetooth\_service\_user\_cb\_t::user\_avrc\_ct\_cb (C++ member), 153

BluFi, 429

blufi\_config\_create (C++ function), 164

blufi\_send\_customized\_data (C++ function), 165

blufi\_set\_customized\_data (C++ function), 165

blufi\_set\_sta\_connected\_flag (C++ function), 164

blufi\_set\_volume\_report\_freq (C++ function), 164

## C

CHARGING\_STOP

cavity, 429

UNKNOWN word, 429

console\_cmd\_callback\_t (C++ type), 228

CONSOLE\_DEFAULT\_BUFFER\_SIZE (C macro), 228

CONSOLE\_DEFAULT\_PROMPT\_STRING (C macro), 228

CONSOLE\_DEFAULT\_TASK\_PRIO (C macro), 228

CONSOLE\_DEFAULT\_TASK\_STACK (C macro), 228

core dump, 429

coredump\_need\_upload (C++ function), 188

coredump\_upload (C++ function), 188

coredump\_upload\_service\_config\_t (C++ struct), 188

coredump\_upload\_service\_config\_t::cb\_ctx (C++ member), 189

coredump\_upload\_service\_config\_t::do\_post (C++ member), 189

coredump\_upload\_service\_config\_t::evt\_cb (C++ member), 189

coredump\_upload\_service\_config\_t::task\_cdrepatcher, [429](#)  
     (C++ member), [189](#)  
 coredump\_upload\_service\_config\_t::task\_display\_pattern\_t (C++ enum), [178](#)  
     (C++ member), [189](#)  
 coredump\_upload\_service\_config\_t::task\_stack (C++ enumerator), [179](#)  
     (C++ member), [189](#)  
 coredump\_upload\_service\_create (C++ function), [188](#)  
 COREDUMP\_UPLOAD\_SERVICE\_DEFAULT\_CONFIG (C macro), [189](#)  
 ctrl\_func (C++ type), [35](#)  
 cutoff frequency, [429](#)

## D

DAC, [429](#)  
 decoder, [429](#)  
 decoder\_opus\_init (C++ function), [131](#)  
 DEFAULT\_AAC\_DECODER\_CONFIG (C macro), [121](#)  
 DEFAULT\_AMR\_DECODER\_CONFIG (C macro), [122](#)  
 DEFAULT\_AMRNB\_ENCODER\_CONFIG (C macro), [124](#)  
 DEFAULT\_AMRWB\_ENCODER\_CONFIG (C macro), [126](#)  
 DEFAULT\_AUDIO\_ELEMENT\_CONFIG (C macro), [34](#)  
 DEFAULT\_AUDIO\_EVENT\_IFACE\_SIZE (C macro), [51](#)  
 DEFAULT\_AUDIO\_PIPELINE\_CONFIG (C macro), [46](#)  
 DEFAULT\_DOWNMIX\_CONFIG (C macro), [139](#)  
 DEFAULT\_ELEMENT\_BUFFER\_LENGTH (C macro), [34](#)  
 DEFAULT\_ELEMENT\_RINGBUF\_SIZE (C macro), [34](#)  
 DEFAULT\_ELEMENT\_STACK\_SIZE (C macro), [34](#)  
 DEFAULT\_ELEMENT\_TASK\_CORE (C macro), [34](#)  
 DEFAULT\_ELEMENT\_TASK\_PRIO (C macro), [34](#)  
 DEFAULT\_EQUALIZER\_CONFIG (C macro), [142](#)  
 DEFAULT\_ESP\_AUDIO\_CONFIG (C macro), [69](#)  
 DEFAULT\_ESP\_PERIPH\_SET\_CONFIG (C macro), [216](#)  
 DEFAULT\_ESP\_PERIPH\_STACK\_SIZE (C macro), [216](#)  
 DEFAULT\_ESP\_PERIPH\_TASK\_CORE (C macro), [216](#)  
 DEFAULT\_ESP\_PERIPH\_TASK\_PRIO (C macro), [216](#)  
 DEFAULT\_FLAC\_DECODER\_CONFIG (C macro), [128](#)  
 DEFAULT\_MP3\_DECODER\_CONFIG (C macro), [130](#)  
 DEFAULT\_OGG\_DECODER\_CONFIG (C macro), [131](#)  
 DEFAULT\_OPUS\_DECODER\_CONFIG (C macro), [132](#)  
 DEFAULT\_PIPELINE\_RINGBUF\_SIZE (C macro), [46](#)  
 DEFAULT\_RECORDER\_SR\_CFG (C macro), [206](#)  
 DEFAULT\_RESAMPLE\_FILTER\_CONFIG (C macro), [146](#)  
 DEFAULT\_SONIC\_CONFIG (C macro), [149](#)  
 DEFAULT\_WAV\_DECODER\_CONFIG (C macro), [134](#)  
 DEFAULT\_WAV\_ENCODER\_CONFIG (C macro), [135](#)  
 Digital Living Network Alliance, [429](#)  
 digital media renderer, [429](#)  
 digital signal processor, [429](#)  
 digital-to-analog converter, [429](#)  
 display\_destroy (C++ function), [177](#)  
 display\_pattern\_t (C++ enum), [178](#)  
 display\_pattern\_t::DISPLAY\_PATTERN\_BATTERY\_CHARGING (C++ enumerator), [179](#)  
 display\_pattern\_t::DISPLAY\_PATTERN\_BATTERY\_FULL (C++ enumerator), [179](#)  
 display\_pattern\_t::DISPLAY\_PATTERN\_BATTERY\_LOW (C++ enumerator), [179](#)  
 display\_pattern\_t::DISPLAY\_PATTERN\_BT\_CONNECTED (C++ enumerator), [178](#)  
 display\_pattern\_t::DISPLAY\_PATTERN\_BT\_CONNECTING (C++ enumerator), [178](#)  
 display\_pattern\_t::DISPLAY\_PATTERN\_BT\_DISCONNECTED (C++ enumerator), [178](#)  
 display\_pattern\_t::DISPLAY\_PATTERN\_MAX (C++ enumerator), [179](#)  
 display\_pattern\_t::DISPLAY\_PATTERN\_MUSIC\_FINISHED (C++ enumerator), [178](#)  
 display\_pattern\_t::DISPLAY\_PATTERN\_MUSIC\_ON (C++ enumerator), [178](#)  
 display\_pattern\_t::DISPLAY\_PATTERN\_MUTE\_OFF (C++ enumerator), [179](#)  
 display\_pattern\_t::DISPLAY\_PATTERN\_MUTE\_ON (C++ enumerator), [179](#)  
 display\_pattern\_t::DISPLAY\_PATTERN\_POWERON\_INIT (C++ enumerator), [179](#)  
 display\_pattern\_t::DISPLAY\_PATTERN\_RECOGNITION\_START (C++ enumerator), [178](#)  
 display\_pattern\_t::DISPLAY\_PATTERN\_RECOGNITION\_STOP (C++ enumerator), [178](#)  
 display\_pattern\_t::DISPLAY\_PATTERN\_RECORDING\_START (C++ enumerator), [178](#)  
 display\_pattern\_t::DISPLAY\_PATTERN\_RECORDING\_STOP (C++ enumerator), [178](#)  
 display\_pattern\_t::DISPLAY\_PATTERN\_SPEECH\_BEGIN (C++ enumerator), [179](#)  
 display\_pattern\_t::DISPLAY\_PATTERN\_SPEECH\_OVER (C++ enumerator), [179](#)  
 display\_pattern\_t::DISPLAY\_PATTERN\_TURN\_OFF (C++ enumerator), [179](#)  
 display\_pattern\_t::DISPLAY\_PATTERN\_TURN\_ON (C++ enumerator), [179](#)  
 display\_pattern\_t::DISPLAY\_PATTERN\_UNKNOWN (C++ enumerator), [178](#)  
 display\_pattern\_t::DISPLAY\_PATTERN\_VOLUME (C++ enumerator), [179](#)  
 display\_pattern\_t::DISPLAY\_PATTERN\_WAKEUP\_FINISHED (C++ enumerator), [178](#)  
 display\_pattern\_t::DISPLAY\_PATTERN\_WAKEUP\_ON (C++ enumerator), [178](#)  
 display\_pattern\_t::DISPLAY\_PATTERN\_WIFI\_CONNECTED (C++ enumerator), [178](#)  
 display\_pattern\_t::DISPLAY\_PATTERN\_WIFI\_CONNECTING (C++ enumerator), [178](#)

- (C++ enumerator), 178
  - display\_pattern\_t::DISPLAY\_PATTERN\_WIFI\_DISCONNECTED (C++ enumerator), 178
  - display\_pattern\_t::DISPLAY\_PATTERN\_WIFI\_NO\_CFG (C++ enumerator), 179
  - display\_pattern\_t::DISPLAY\_PATTERN\_WIFI\_SETTING (C++ enumerator), 178
  - display\_pattern\_t::DISPLAY\_PATTERN\_WIFI\_SETTINGS\_FINISHED (C++ enumerator), 178
  - display\_service\_config\_t (C++ struct), 177
  - display\_service\_config\_t::based\_cfg (C++ member), 177
  - display\_service\_config\_t::instance (C++ member), 177
  - display\_service\_create (C++ function), 177
  - display\_service\_handle\_t (C++ type), 178
  - display\_service\_set\_pattern (C++ function), 177
  - distortion, 429
  - DLNA, 429
  - DM\_BUF\_SIZE (C macro), 139
  - DMR, 429
  - downmix, 429
  - downmix\_cfg\_t (C++ struct), 139
  - downmix\_cfg\_t::downmix\_info (C++ member), 139
  - downmix\_cfg\_t::max\_sample (C++ member), 139
  - downmix\_cfg\_t::out\_rb\_size (C++ member), 139
  - downmix\_cfg\_t::stack\_in\_ext (C++ member), 139
  - downmix\_cfg\_t::task\_core (C++ member), 139
  - downmix\_cfg\_t::task\_prio (C++ member), 139
  - downmix\_cfg\_t::task\_stack (C++ member), 139
  - downmix\_init (C++ function), 138
  - DOWNMIX\_RINGBUFFER\_SIZE (C macro), 139
  - downmix\_set\_gain\_info (C++ function), 137
  - downmix\_set\_input\_rb (C++ function), 136
  - downmix\_set\_input\_rb\_timeout (C++ function), 136
  - downmix\_set\_out\_ctx\_info (C++ function), 137
  - downmix\_set\_output\_type (C++ function), 136
  - downmix\_set\_source\_stream\_info (C++ function), 137
  - downmix\_set\_transit\_time\_info (C++ function), 138
  - downmix\_set\_work\_mode (C++ function), 137
  - DOWNMIX\_TASK\_CORE (C macro), 139
  - DOWNMIX\_TASK\_PRIO (C macro), 139
  - DOWNMIX\_TASK\_STACK (C macro), 139
  - dram\_list\_choose (C++ function), 106
  - dram\_list\_create (C++ function), 105
  - dram\_list\_current (C++ function), 106
  - dram\_list\_destroy (C++ function), 107
  - dram\_list\_exist (C++ function), 106
  - dram\_list\_get\_url\_id (C++ function), 107
  - dram\_list\_get\_url\_num (C++ function), 106
  - dram\_list\_get\_url\_next (C++ function), 105
  - dram\_list\_prev (C++ function), 105
  - dram\_list\_remove\_by\_url (C++ function), 107
  - dram\_list\_remove\_by\_url\_id (C++ function), 107
  - dram\_list\_reset (C++ function), 106
  - dram\_list\_save (C++ function), 105
  - dram\_list\_show (C++ function), 107
  - DSP, 429
  - DuerOS, 430
  - dueros\_service\_create (C++ function), 175
  - dueros\_service\_state\_get (C++ function), 175
  - dueros\_start\_wifi\_cfg (C++ function), 176
  - dueros\_stop\_wifi\_cfg (C++ function), 176
  - dueros\_voice\_cancel (C++ function), 176
  - dueros\_voice\_upload (C++ function), 175
  - dueros\_wifi\_st\_t (C++ struct), 176
  - dueros\_wifi\_st\_t::err (C++ member), 176
  - dueros\_wifi\_st\_t::status (C++ member), 176
  - dueros\_wifi\_status\_report (C++ function), 176
- ## E
- echo, 430
  - echo reference signal, 430
  - ECM, 430
  - el\_io\_func (C++ type), 34
  - electret condenser microphone, 430
  - element, 430
  - ELEMENT\_SUB\_TYPE\_OFFSET (C macro), 52
  - EMBED\_FLASH\_STREAM\_BUF\_SIZE (C macro), 97
  - EMBED\_FLASH\_STREAM\_CFG\_DEFAULT (C macro), 97
  - embed\_flash\_stream\_cfg\_t (C++ struct), 96
  - embed\_flash\_stream\_cfg\_t::buf\_sz (C++ member), 96
  - embed\_flash\_stream\_cfg\_t::extern\_stack (C++ member), 97
  - embed\_flash\_stream\_cfg\_t::out\_rb\_size (C++ member), 96
  - embed\_flash\_stream\_cfg\_t::task\_core (C++ member), 96
  - embed\_flash\_stream\_cfg\_t::task\_prio (C++ member), 97
  - embed\_flash\_stream\_cfg\_t::task\_stack (C++ member), 96
  - EMBED\_FLASH\_STREAM\_EXT\_STACK (C macro), 97
  - embed\_flash\_stream\_init (C++ function), 96

- EMBED\_FLASH\_STREAM\_RINGBUFFER\_SIZE (C macro), 97
- embed\_flash\_stream\_set\_context (C++ function), 96
- EMBED\_FLASH\_STREAM\_TASK\_CORE (C macro), 97
- EMBED\_FLASH\_STREAM\_TASK\_PRIO (C macro), 97
- EMBED\_FLASH\_STREAM\_TASK\_STACK (C macro), 97
- embed\_item\_info (C++ struct), 97
- embed\_item\_info::address (C++ member), 97
- embed\_item\_info::size (C++ member), 97
- embed\_item\_info\_t (C++ type), 98
- encoder, 430
- equalizer, 430
- equalizer\_cfg (C++ struct), 141
- equalizer\_cfg::channel (C++ member), 142
- equalizer\_cfg::out\_rb\_size (C++ member), 142
- equalizer\_cfg::samplerate (C++ member), 142
- equalizer\_cfg::set\_gain (C++ member), 142
- equalizer\_cfg::stack\_in\_ext (C++ member), 142
- equalizer\_cfg::task\_core (C++ member), 142
- equalizer\_cfg::task\_prio (C++ member), 142
- equalizer\_cfg::task\_stack (C++ member), 142
- equalizer\_cfg\_t (C++ type), 143
- equalizer\_init (C++ function), 141
- EQUALIZER\_RINGBUFFER\_SIZE (C macro), 142
- equalizer\_set\_gain\_info (C++ function), 140
- equalizer\_set\_info (C++ function), 140
- EQUALIZER\_TASK\_CORE (C macro), 142
- EQUALIZER\_TASK\_PRIO (C macro), 142
- EQUALIZER\_TASK\_STACK (C macro), 142
- ES8374\_ADDR (C macro), 262
- es8374\_codec\_config\_i2s (C++ function), 261
- es8374\_codec\_ctrl\_state (C++ function), 261
- es8374\_codec\_deinit (C++ function), 258
- es8374\_codec\_get\_voice\_volume (C++ function), 259
- es8374\_codec\_init (C++ function), 258
- es8374\_codec\_set\_voice\_volume (C++ function), 259
- es8374\_config\_adc\_input (C++ function), 260
- es8374\_config\_dac\_output (C++ function), 260
- es8374\_config\_fmt (C++ function), 258
- es8374\_get\_voice\_mute (C++ function), 260
- es8374\_i2s\_config\_clock (C++ function), 259
- es8374\_pa\_power (C++ function), 261
- es8374\_read\_all (C++ function), 261
- es8374\_set\_bits\_per\_sample (C++ function), 259
- es8374\_set\_mic\_gain (C++ function), 260
- es8374\_set\_voice\_mute (C++ function), 260
- es8374\_start (C++ function), 259
- es8374\_stop (C++ function), 259
- es8374\_write\_reg (C++ function), 260
- ES8388\_ADCCONTROL1 (C macro), 256
- ES8388\_ADCCONTROL10 (C macro), 256
- ES8388\_ADCCONTROL11 (C macro), 256
- ES8388\_ADCCONTROL12 (C macro), 256
- ES8388\_ADCCONTROL13 (C macro), 256
- ES8388\_ADCCONTROL14 (C macro), 256
- ES8388\_ADCCONTROL2 (C macro), 256
- ES8388\_ADCCONTROL3 (C macro), 256
- ES8388\_ADCCONTROL4 (C macro), 256
- ES8388\_ADCCONTROL5 (C macro), 256
- ES8388\_ADCCONTROL6 (C macro), 256
- ES8388\_ADCCONTROL7 (C macro), 256
- ES8388\_ADCCONTROL8 (C macro), 256
- ES8388\_ADCCONTROL9 (C macro), 256
- ES8388\_ADCPOWER (C macro), 255
- ES8388\_ADDR (C macro), 255
- ES8388\_ANAVOLMANAG (C macro), 256
- ES8388\_CHIPLOPOW1 (C macro), 256
- ES8388\_CHIPLOPOW2 (C macro), 256
- ES8388\_CHIPPOWER (C macro), 255
- es8388\_config\_adc\_input (C++ function), 254
- es8388\_config\_dac\_output (C++ function), 254
- es8388\_config\_fmt (C++ function), 252
- es8388\_config\_i2s (C++ function), 255
- ES8388\_CONTROL1 (C macro), 255
- ES8388\_CONTROL2 (C macro), 255
- es8388\_ctrl\_state (C++ function), 255
- ES8388\_DACCONTROL1 (C macro), 256
- ES8388\_DACCONTROL10 (C macro), 257
- ES8388\_DACCONTROL11 (C macro), 257
- ES8388\_DACCONTROL12 (C macro), 257
- ES8388\_DACCONTROL13 (C macro), 257
- ES8388\_DACCONTROL14 (C macro), 257
- ES8388\_DACCONTROL15 (C macro), 257
- ES8388\_DACCONTROL16 (C macro), 257
- ES8388\_DACCONTROL17 (C macro), 257
- ES8388\_DACCONTROL18 (C macro), 257
- ES8388\_DACCONTROL19 (C macro), 257
- ES8388\_DACCONTROL2 (C macro), 256
- ES8388\_DACCONTROL20 (C macro), 257
- ES8388\_DACCONTROL21 (C macro), 257
- ES8388\_DACCONTROL22 (C macro), 257
- ES8388\_DACCONTROL23 (C macro), 257
- ES8388\_DACCONTROL24 (C macro), 257
- ES8388\_DACCONTROL25 (C macro), 258
- ES8388\_DACCONTROL26 (C macro), 258
- ES8388\_DACCONTROL27 (C macro), 258
- ES8388\_DACCONTROL28 (C macro), 258
- ES8388\_DACCONTROL29 (C macro), 258
- ES8388\_DACCONTROL3 (C macro), 256
- ES8388\_DACCONTROL30 (C macro), 258

- ES8388\_DACCONTROL4 (C macro), 257
- ES8388\_DACCONTROL5 (C macro), 257
- ES8388\_DACCONTROL6 (C macro), 257
- ES8388\_DACCONTROL7 (C macro), 257
- ES8388\_DACCONTROL8 (C macro), 257
- ES8388\_DACCONTROL9 (C macro), 257
- ES8388\_DACPOWER (C macro), 255
- es8388\_deinit (C++ function), 252
- es8388\_get\_voice\_mute (C++ function), 254
- es8388\_get\_voice\_volume (C++ function), 253
- es8388\_i2s\_config\_clock (C++ function), 252
- es8388\_init (C++ function), 252
- ES8388\_MASTERMODE (C macro), 256
- es8388\_pa\_power (C++ function), 255
- es8388\_read\_all (C++ function), 255
- es8388\_set\_bits\_per\_sample (C++ function), 253
- es8388\_set\_mic\_gain (C++ function), 254
- es8388\_set\_voice\_mute (C++ function), 254
- es8388\_set\_voice\_volume (C++ function), 253
- es8388\_start (C++ function), 253
- es8388\_stop (C++ function), 253
- es8388\_write\_reg (C++ function), 254
- ESP VoIP, 430
- ESP\_A2DP\_SAMPLE\_RATE (C macro), 154
- esp\_audio\_callback\_set (C++ function), 64
- esp\_audio\_cfg\_t (C++ struct), 66
- esp\_audio\_cfg\_t::cb\_ctx (C++ member), 66
- esp\_audio\_cfg\_t::cb\_func (C++ member), 66
- esp\_audio\_cfg\_t::component\_select (C++ member), 66
- esp\_audio\_cfg\_t::evt\_que (C++ member), 66
- esp\_audio\_cfg\_t::in\_stream\_buf\_size (C++ member), 66
- esp\_audio\_cfg\_t::out\_stream\_buf\_size (C++ member), 66
- esp\_audio\_cfg\_t::prefer\_type (C++ member), 66
- esp\_audio\_cfg\_t::resample\_rate (C++ member), 66
- esp\_audio\_cfg\_t::task\_prio (C++ member), 67
- esp\_audio\_cfg\_t::task\_stack (C++ member), 67
- esp\_audio\_cfg\_t::vol\_get (C++ member), 67
- esp\_audio\_cfg\_t::vol\_handle (C++ member), 66
- esp\_audio\_cfg\_t::vol\_set (C++ member), 67
- esp\_audio\_codec\_lib\_add (C++ function), 57
- esp\_audio\_codec\_lib\_query (C++ function), 58
- ESP\_AUDIO\_COMPONENT\_SELECT\_ALC (C macro), 69
- ESP\_AUDIO\_COMPONENT\_SELECT\_DEFAULT (C macro), 69
- ESP\_AUDIO\_COMPONENT\_SELECT\_EQUALIZER (C macro), 69
- esp\_audio\_create (C++ function), 57
- esp\_audio\_destroy (C++ function), 57
- esp\_audio\_duration\_get (C++ function), 65
- esp\_audio\_eq\_gain\_get (C++ function), 61
- esp\_audio\_eq\_gain\_set (C++ function), 61
- esp\_audio\_event\_callback (C++ type), 54
- esp\_audio\_event\_que\_set (C++ function), 65
- esp\_audio\_handle\_t (C++ type), 69
- esp\_audio\_info\_get (C++ function), 64
- esp\_audio\_info\_set (C++ function), 64
- esp\_audio\_info\_t (C++ struct), 68
- esp\_audio\_info\_t::audio\_speed (C++ member), 68
- esp\_audio\_info\_t::codec\_el (C++ member), 68
- esp\_audio\_info\_t::codec\_info (C++ member), 68
- esp\_audio\_info\_t::filter\_el (C++ member), 68
- esp\_audio\_info\_t::in\_el (C++ member), 68
- esp\_audio\_info\_t::in\_stream\_total\_size (C++ member), 68
- esp\_audio\_info\_t::out\_el (C++ member), 68
- esp\_audio\_info\_t::st (C++ member), 68
- esp\_audio\_info\_t::time\_pos (C++ member), 68
- esp\_audio\_input\_stream\_add (C++ function), 57
- esp\_audio\_media\_type\_set (C++ function), 64
- esp\_audio\_music\_info\_get (C++ function), 64
- esp\_audio\_music\_info\_t (C++ struct), 68
- esp\_audio\_music\_info\_t::bits (C++ member), 69
- esp\_audio\_music\_info\_t::bps (C++ member), 69
- esp\_audio\_music\_info\_t::channels (C++ member), 69
- esp\_audio\_music\_info\_t::codec\_fmt (C++ member), 69
- esp\_audio\_music\_info\_t::sample\_rates (C++ member), 69
- esp\_audio\_output\_stream\_add (C++ function), 57
- esp\_audio\_pause (C++ function), 60
- esp\_audio\_play (C++ function), 58
- esp\_audio\_play\_speed\_t (C++ enum), 69
- esp\_audio\_play\_speed\_t::ESP\_AUDIO\_PLAY\_SPEED\_0\_50 (C++ enumerator), 69
- esp\_audio\_play\_speed\_t::ESP\_AUDIO\_PLAY\_SPEED\_0\_75 (C++ enumerator), 70
- esp\_audio\_play\_speed\_t::ESP\_AUDIO\_PLAY\_SPEED\_1\_00 (C++ enumerator), 70



esp\_audio\_play\_speed\_t::ESP\_AUDIO\_PLAY\_SPEED\_1 (C++ enumerator), 70  
 esp\_audio\_play\_speed\_t::ESP\_AUDIO\_PLAY\_SPEED\_2 (C++ enumerator), 70  
 esp\_audio\_play\_speed\_t::ESP\_AUDIO\_PLAY\_SPEED\_MAX (C++ enumerator), 70  
 esp\_audio\_play\_speed\_t::ESP\_AUDIO\_PLAY\_SPEED\_UNKNOWN (C++ enumerator), 69  
 esp\_audio\_play\_timeout\_set (C++ function), 65  
 esp\_audio\_pos\_get (C++ function), 63  
 esp\_audio\_prefer\_t (C++ enum), 55  
 esp\_audio\_prefer\_t::ESP\_AUDIO\_PREFER\_MEMES (C++ enumerator), 56  
 esp\_audio\_prefer\_t::ESP\_AUDIO\_PREFER\_SPEED (C++ enumerator), 56  
 esp\_audio\_prefer\_type\_get (C++ function), 65  
 esp\_audio\_resume (C++ function), 61  
 esp\_audio\_seek (C++ function), 64  
 esp\_audio\_setup (C++ function), 64  
 esp\_audio\_setup\_t (C++ struct), 67  
 esp\_audio\_setup\_t::set\_channel (C++ member), 67  
 esp\_audio\_setup\_t::set\_codec (C++ member), 67  
 esp\_audio\_setup\_t::set\_in\_stream (C++ member), 67  
 esp\_audio\_setup\_t::set\_out\_stream (C++ member), 67  
 esp\_audio\_setup\_t::set\_pos (C++ member), 67  
 esp\_audio\_setup\_t::set\_sample\_rate (C++ member), 67  
 esp\_audio\_setup\_t::set\_time (C++ member), 67  
 esp\_audio\_setup\_t::set\_type (C++ member), 67  
 esp\_audio\_setup\_t::set\_uri (C++ member), 67  
 esp\_audio\_speed\_get (C++ function), 62  
 esp\_audio\_speed\_idx\_to\_float (C++ function), 62  
 esp\_audio\_speed\_set (C++ function), 62  
 esp\_audio\_state\_get (C++ function), 63  
 esp\_audio\_state\_t (C++ struct), 53  
 esp\_audio\_state\_t::err\_msg (C++ member), 53  
 esp\_audio\_state\_t::media\_src (C++ member), 53  
 esp\_audio\_state\_t::status (C++ member), 53  
 esp\_audio\_status\_t::AUDIO\_STATUS\_ERROR (C++ enumerator), 55  
 esp\_audio\_status\_t::AUDIO\_STATUS\_FINISHED (C++ enumerator), 55  
 esp\_audio\_status\_t::AUDIO\_STATUS\_PAUSED (C++ enumerator), 55  
 esp\_audio\_status\_t::AUDIO\_STATUS\_RUNNING (C++ enumerator), 55  
 esp\_audio\_status\_t::AUDIO\_STATUS\_STOPPED (C++ enumerator), 55  
 esp\_audio\_status\_t::AUDIO\_STATUS\_UNKNOWN (C++ enumerator), 55  
 esp\_audio\_stop (C++ function), 60  
 esp\_audio\_sync\_play (C++ function), 59  
 esp\_audio\_time\_get (C++ function), 63  
 esp\_audio\_vol\_get (C++ function), 63  
 esp\_audio\_vol\_set (C++ function), 62  
 ESP\_ERR\_AUDIO\_BASE (C macro), 53  
 ESP\_ERR\_FS\_OTA\_BASE (C macro), 175  
 ESP\_ERR\_FS\_OTA\_IN\_PROGRESS (C macro), 175  
 esp\_fs\_ota (C++ function), 172  
 esp\_fs\_ota\_begin (C++ function), 173  
 esp\_fs\_ota\_config\_t (C++ struct), 174  
 esp\_fs\_ota\_config\_t::buffer\_size (C++ member), 174  
 esp\_fs\_ota\_config\_t::path (C++ member), 174  
 esp\_fs\_ota\_finish (C++ function), 173  
 esp\_fs\_ota\_get\_image\_len\_read (C++ function), 174  
 esp\_fs\_ota\_get\_img\_desc (C++ function), 174  
 esp\_fs\_ota\_handle\_t (C++ type), 175  
 esp\_fs\_ota\_perform (C++ function), 173  
 esp\_periph\_alloc\_periph\_id (C++ function), 210  
 esp\_periph\_config\_t (C++ struct), 215  
 esp\_periph\_config\_t::extern\_stack (C++ member), 215  
 esp\_periph\_config\_t::task\_core (C++ member), 215  
 esp\_periph\_config\_t::task\_prio (C++ member), 215  
 esp\_periph\_config\_t::task\_stack (C++ member), 215  
 esp\_periph\_create (C++ function), 211  
 esp\_periph\_destroy (C++ function), 214  
 esp\_periph\_event (C++ struct), 215  
 esp\_periph\_event::cb (C++ member), 216  
 esp\_periph\_event::iface (C++ member), 216  
 esp\_periph\_event::user\_ctx (C++ member), 216  
 esp\_periph\_event\_handle\_t (C++ type), 216  
 esp\_periph\_event\_t (C++ type), 216

- esp\_periph\_func (C++ type), 216
- esp\_periph\_get\_data (C++ function), 214
- esp\_periph\_get\_id (C++ function), 214
- esp\_periph\_get\_state (C++ function), 214
- esp\_periph\_handle\_t (C++ type), 216
- esp\_periph\_id\_t (C++ enum), 217
- esp\_periph\_id\_t::PERIPH\_ID\_ADC (C++ enumerator), 217
- esp\_periph\_id\_t::PERIPH\_ID\_ADC\_BTN (C++ enumerator), 217
- esp\_periph\_id\_t::PERIPH\_ID\_AUXIN (C++ enumerator), 217
- esp\_periph\_id\_t::PERIPH\_ID\_AW2013 (C++ enumerator), 217
- esp\_periph\_id\_t::PERIPH\_ID\_BLUETOOTH (C++ enumerator), 217
- esp\_periph\_id\_t::PERIPH\_ID\_BUTTON (C++ enumerator), 217
- esp\_periph\_id\_t::PERIPH\_ID\_CONSOLE (C++ enumerator), 217
- esp\_periph\_id\_t::PERIPH\_ID\_CUSTOM\_BASE (C++ enumerator), 217
- esp\_periph\_id\_t::PERIPH\_ID\_FLASH (C++ enumerator), 217
- esp\_periph\_id\_t::PERIPH\_ID\_GPIO\_ISR (C++ enumerator), 217
- esp\_periph\_id\_t::PERIPH\_ID\_IS31FL3216 (C++ enumerator), 217
- esp\_periph\_id\_t::PERIPH\_ID\_LCD (C++ enumerator), 217
- esp\_periph\_id\_t::PERIPH\_ID\_LED (C++ enumerator), 217
- esp\_periph\_id\_t::PERIPH\_ID\_SDCARD (C++ enumerator), 217
- esp\_periph\_id\_t::PERIPH\_ID\_SPIFFS (C++ enumerator), 217
- esp\_periph\_id\_t::PERIPH\_ID\_TOUCH (C++ enumerator), 217
- esp\_periph\_id\_t::PERIPH\_ID\_WIFI (C++ enumerator), 217
- esp\_periph\_id\_t::PERIPH\_ID\_WS2812 (C++ enumerator), 217
- esp\_periph\_init (C++ function), 214
- esp\_periph\_register\_on\_events (C++ function), 215
- esp\_periph\_remove\_from\_set (C++ function), 210
- esp\_periph\_run (C++ function), 214
- esp\_periph\_run\_func (C++ type), 216
- esp\_periph\_send\_cmd (C++ function), 212
- esp\_periph\_send\_cmd\_from\_isr (C++ function), 212
- esp\_periph\_send\_event (C++ function), 212
- esp\_periph\_set\_change\_waiting\_time (C++ function), 211
- esp\_periph\_set\_data (C++ function), 213
- esp\_periph\_set\_destroy (C++ function), 208
- esp\_periph\_set\_function (C++ function), 211
- esp\_periph\_set\_get\_by\_id (C++ function), 209
- esp\_periph\_set\_get\_event\_iface (C++ function), 209
- esp\_periph\_set\_get\_queue (C++ function), 209
- esp\_periph\_set\_handle\_t (C++ type), 216
- esp\_periph\_set\_id (C++ function), 214
- esp\_periph\_set\_init (C++ function), 208
- esp\_periph\_set\_list\_destroy (C++ function), 210
- esp\_periph\_set\_list\_init (C++ function), 209
- esp\_periph\_set\_list\_run (C++ function), 209
- esp\_periph\_set\_register\_callback (C++ function), 209
- esp\_periph\_set\_stop\_all (C++ function), 208
- esp\_periph\_start (C++ function), 211
- esp\_periph\_start\_timer (C++ function), 213
- esp\_periph\_state\_t (C++ enum), 217
- esp\_periph\_state\_t::PERIPH\_STATE\_ERROR (C++ enumerator), 218
- esp\_periph\_state\_t::PERIPH\_STATE\_INIT (C++ enumerator), 218
- esp\_periph\_state\_t::PERIPH\_STATE\_NULL (C++ enumerator), 218
- esp\_periph\_state\_t::PERIPH\_STATE\_PAUSE (C++ enumerator), 218
- esp\_periph\_state\_t::PERIPH\_STATE\_RUNNING (C++ enumerator), 218
- esp\_periph\_state\_t::PERIPH\_STATE\_STATUS\_MAX (C++ enumerator), 218
- esp\_periph\_state\_t::PERIPH\_STATE\_STOPPING (C++ enumerator), 218
- esp\_periph\_stop (C++ function), 212
- esp\_periph\_stop\_timer (C++ function), 213
- esp\_touch\_pad\_sel\_t (C++ enum), 230
- esp\_touch\_pad\_sel\_t::TOUCH\_PAD\_SEL0 (C++ enumerator), 230
- esp\_touch\_pad\_sel\_t::TOUCH\_PAD\_SEL1 (C++ enumerator), 230
- esp\_touch\_pad\_sel\_t::TOUCH\_PAD\_SEL2 (C++ enumerator), 230
- esp\_touch\_pad\_sel\_t::TOUCH\_PAD\_SEL3 (C++ enumerator), 230
- esp\_touch\_pad\_sel\_t::TOUCH\_PAD\_SEL4 (C++ enumerator), 230
- esp\_touch\_pad\_sel\_t::TOUCH\_PAD\_SEL5 (C++ enumerator), 230
- esp\_touch\_pad\_sel\_t::TOUCH\_PAD\_SEL6 (C++ enumerator), 230
- esp\_touch\_pad\_sel\_t::TOUCH\_PAD\_SEL7 (C++ enumerator), 230

- esp\_touch\_pad\_sel\_t::TOUCH\_PAD\_SEL8 (C++ enumerator), 230
- esp\_touch\_pad\_sel\_t::TOUCH\_PAD\_SEL9 (C++ enumerator), 230
- esp\_wifi\_set\_listen\_interval (C++ function), 219
- esp\_wifi\_setting\_create (C++ function), 161
- esp\_wifi\_setting\_destroy (C++ function), 161
- esp\_wifi\_setting\_get\_data (C++ function), 163
- esp\_wifi\_setting\_handle\_t (C++ type), 163
- esp\_wifi\_setting\_info\_notify (C++ function), 162
- esp\_wifi\_setting\_register\_function (C++ function), 161
- esp\_wifi\_setting\_register\_notify\_handle (C++ function), 162
- esp\_wifi\_setting\_set\_data (C++ function), 162
- esp\_wifi\_setting\_start (C++ function), 163
- esp\_wifi\_setting\_stop (C++ function), 163
- esp\_wifi\_setting\_teardown (C++ function), 163
- event\_cb\_func (C++ type), 35
- ## F
- fast Fourier transform, 430
- FatFs, 430
- FatFs stream, 430
- FATFS\_STREAM\_BUF\_SIZE (C macro), 76
- FATFS\_STREAM\_CFG\_DEFAULT (C macro), 76
- fatfs\_stream\_cfg\_t (C++ struct), 75
- fatfs\_stream\_cfg\_t::buf\_sz (C++ member), 75
- fatfs\_stream\_cfg\_t::ext\_stack (C++ member), 75
- fatfs\_stream\_cfg\_t::out\_rb\_size (C++ member), 75
- fatfs\_stream\_cfg\_t::task\_core (C++ member), 75
- fatfs\_stream\_cfg\_t::task\_prio (C++ member), 75
- fatfs\_stream\_cfg\_t::task\_stack (C++ member), 75
- fatfs\_stream\_cfg\_t::type (C++ member), 75
- fatfs\_stream\_cfg\_t::write\_header (C++ member), 75
- fatfs\_stream\_init (C++ function), 75
- FATFS\_STREAM\_RINGBUFFER\_SIZE (C macro), 76
- FATFS\_STREAM\_TASK\_CORE (C macro), 76
- FATFS\_STREAM\_TASK\_PRIO (C macro), 76
- FATFS\_STREAM\_TASK\_STACK (C macro), 76
- FB, 430
- FEED\_TASK\_PINNED\_CORE (C macro), 206
- FEED\_TASK\_PRIO (C macro), 206
- FEED\_TASK\_STACK\_SZ (C macro), 206
- FETCH\_TASK\_PINNED\_CORE (C macro), 206
- FETCH\_TASK\_PRIO (C macro), 206
- FETCH\_TASK\_STACK\_SZ (C macro), 206
- FFT, 430
- FLAC, 430
- flac\_decoder\_cfg\_t (C++ struct), 127
- flac\_decoder\_cfg\_t::out\_rb\_size (C++ member), 128
- flac\_decoder\_cfg\_t::stack\_in\_ext (C++ member), 128
- flac\_decoder\_cfg\_t::task\_core (C++ member), 128
- flac\_decoder\_cfg\_t::task\_prio (C++ member), 128
- flac\_decoder\_cfg\_t::task\_stack (C++ member), 128
- flac\_decoder\_init (C++ function), 127
- FLAC\_DECODER\_RINGBUFFER\_SIZE (C macro), 128
- FLAC\_DECODER\_TASK\_CORE (C macro), 128
- FLAC\_DECODER\_TASK\_PRIO (C macro), 128
- FLAC\_DECODER\_TASK\_STACK\_SIZE (C macro), 128
- flash\_list\_choose (C++ function), 109
- flash\_list\_create (C++ function), 108
- flash\_list\_current (C++ function), 109
- flash\_list\_destroy (C++ function), 110
- flash\_list\_exist (C++ function), 109
- flash\_list\_get\_url\_id (C++ function), 110
- flash\_list\_get\_url\_num (C++ function), 110
- flash\_list\_next (C++ function), 108
- flash\_list\_prev (C++ function), 109
- flash\_list\_reset (C++ function), 109
- flash\_list\_save (C++ function), 108
- flash\_list\_show (C++ function), 108
- flexible pipeline, 430
- FPS, 430
- frames per second, 430
- frequency response, 430
- full band, 430
- ## G
- get\_input\_key\_service\_state (C++ function), 155
- ## H
- HAL, 431
- Hands-Free, 430
- Hands-Free Audio Gateway, 431
- Hands-Free Profile, 431
- Hands-Free Unit, 431
- hardware abstraction layer, 431
- headset, 431
- HF, 431

- HFP, [431](#)  
HFP-AG, [431](#)  
Hi-Fi speaker, [431](#)  
High Frequency, [431](#)  
high-fidelity microphone, [431](#)  
HLS, [431](#)  
HTTP Live Streaming, [431](#)  
HTTP stream, [431](#)  
HTTP\_STREAM\_CFG\_DEFAULT (*C macro*), [80](#)  
http\_stream\_cfg\_t (*C++ struct*), [78](#)  
http\_stream\_cfg\_t::auto\_connect\_next\_track (C++ member), [79](#)  
http\_stream\_cfg\_t::cert\_pem (C++ member), [79](#)  
http\_stream\_cfg\_t::crt\_bundle\_attach (C++ member), [79](#)  
http\_stream\_cfg\_t::enable\_playlist\_parse (C++ member), [79](#)  
http\_stream\_cfg\_t::event\_handle (C++ member), [79](#)  
http\_stream\_cfg\_t::multi\_out\_num (C++ member), [79](#)  
http\_stream\_cfg\_t::out\_rb\_size (C++ member), [78](#)  
http\_stream\_cfg\_t::request\_range\_size (C++ member), [79](#)  
http\_stream\_cfg\_t::request\_size (C++ member), [79](#)  
http\_stream\_cfg\_t::stack\_in\_ext (C++ member), [78](#)  
http\_stream\_cfg\_t::task\_core (C++ member), [78](#)  
http\_stream\_cfg\_t::task\_prio (C++ member), [78](#)  
http\_stream\_cfg\_t::task\_stack (C++ member), [78](#)  
http\_stream\_cfg\_t::type (C++ member), [78](#)  
http\_stream\_cfg\_t::user\_agent (C++ member), [79](#)  
http\_stream\_cfg\_t::user\_data (C++ member), [79](#)  
http\_stream\_event\_handle\_t (*C++ type*), [80](#)  
http\_stream\_event\_id\_t (*C++ enum*), [80](#)  
http\_stream\_event\_id\_t::HTTP\_STREAM\_FINISH\_PLAYLIST (C++ enumerator), [80](#)  
http\_stream\_event\_id\_t::HTTP\_STREAM\_FINISH\_REQUEST (C++ enumerator), [80](#)  
http\_stream\_event\_id\_t::HTTP\_STREAM\_FINISH\_STREAM (C++ enumerator), [80](#)  
http\_stream\_event\_id\_t::HTTP\_STREAM\_ON\_REQUEST (C++ enumerator), [80](#)  
http\_stream\_event\_id\_t::HTTP\_STREAM\_ON\_RESPONSE (C++ enumerator), [80](#)  
http\_stream\_event\_id\_t::HTTP\_STREAM\_POST\_REQUEST (C++ enumerator), [80](#)  
http\_stream\_event\_id\_t::HTTP\_STREAM\_PRE\_REQUEST (C++ enumerator), [80](#)  
http\_stream\_event\_id\_t::HTTP\_STREAM\_RESOLVE\_ALL\_TRACKS (C++ enumerator), [80](#)  
http\_stream\_event\_msg\_t (*C++ struct*), [78](#)  
http\_stream\_event\_msg\_t::buffer (C++ member), [78](#)  
http\_stream\_event\_msg\_t::buffer\_len (C++ member), [78](#)  
http\_stream\_event\_msg\_t::el (C++ member), [78](#)  
http\_stream\_event\_msg\_t::event\_id (C++ member), [78](#)  
http\_stream\_event\_msg\_t::http\_client (C++ member), [78](#)  
http\_stream\_event\_msg\_t::user\_data (C++ member), [78](#)  
http\_stream\_fetch\_again (C++ function), [77](#)  
http\_stream\_init (C++ function), [76](#)  
http\_stream\_next\_track (C++ function), [76](#)  
http\_stream\_restart (C++ function), [77](#)  
HTTP\_STREAM\_RINGBUFFER\_SIZE (*C macro*), [79](#)  
http\_stream\_set\_server\_cert (C++ function), [77](#)  
HTTP\_STREAM\_TASK\_CORE (*C macro*), [79](#)  
HTTP\_STREAM\_TASK\_PRIO (*C macro*), [79](#)  
HTTP\_STREAM\_TASK\_STACK (*C macro*), [79](#)
- I
- I2S stream, [431](#)  
i2s\_alc\_volume\_get (C++ function), [82](#)  
i2s\_alc\_volume\_set (C++ function), [82](#)  
i2s\_channel\_type\_t (*C++ enum*), [84](#)  
i2s\_channel\_type\_t::I2S\_CHANNEL\_TYPE\_ALL\_LEFT (C++ enumerator), [84](#)  
i2s\_channel\_type\_t::I2S\_CHANNEL\_TYPE\_ALL\_RIGHT (C++ enumerator), [84](#)  
i2s\_channel\_type\_t::I2S\_CHANNEL\_TYPE\_ONLY\_LEFT (C++ enumerator), [85](#)  
i2s\_channel\_type\_t::I2S\_CHANNEL\_TYPE\_ONLY\_RIGHT (C++ enumerator), [84](#)  
i2s\_channel\_type\_t::I2S\_CHANNEL\_TYPE\_RIGHT\_LEFT (C++ enumerator), [84](#)  
I2S\_STD\_PHILIPS\_SLOT\_DEFAULT\_ADF\_CONFIG (*C macro*), [84](#)  
I2S\_STREAM\_BUF\_SIZE (*C macro*), [84](#)  
I2S\_STREAM\_CFG\_DEFAULT (*C macro*), [84](#)  
I2S\_STREAM\_CFG\_DEFAULT\_WITH\_PARAMS (*C macro*), [84](#)  
I2S\_STREAM\_CFG\_DEFAULT\_WITH\_TYPE\_AND\_CHANNELS (*C macro*), [84](#)  
i2s\_stream\_cfg\_t (*C++ struct*), [83](#)

- i2s\_stream\_cfg\_t::buffer\_len (C++ member), 84
- i2s\_stream\_cfg\_t::chan\_cfg (C++ member), 83
- i2s\_stream\_cfg\_t::expand\_src\_bits (C++ member), 83
- i2s\_stream\_cfg\_t::multi\_out\_num (C++ member), 83
- i2s\_stream\_cfg\_t::need\_expand (C++ member), 84
- i2s\_stream\_cfg\_t::out\_rb\_size (C++ member), 83
- i2s\_stream\_cfg\_t::stack\_in\_ext (C++ member), 83
- i2s\_stream\_cfg\_t::std\_cfg (C++ member), 83
- i2s\_stream\_cfg\_t::task\_core (C++ member), 83
- i2s\_stream\_cfg\_t::task\_prio (C++ member), 83
- i2s\_stream\_cfg\_t::task\_stack (C++ member), 83
- i2s\_stream\_cfg\_t::transmit\_mode (C++ member), 83
- i2s\_stream\_cfg\_t::type (C++ member), 83
- i2s\_stream\_cfg\_t::uninstall\_drv (C++ member), 83
- i2s\_stream\_cfg\_t::use\_alc (C++ member), 83
- i2s\_stream\_cfg\_t::volume (C++ member), 83
- i2s\_stream\_init (C++ function), 81
- I2S\_STREAM\_RINGBUFFER\_SIZE (C macro), 84
- i2s\_stream\_set\_channel\_type (C++ function), 81
- i2s\_stream\_set\_clk (C++ function), 81
- i2s\_stream\_sync\_delay (C++ function), 82
- I2S\_STREAM\_TASK\_CORE (C macro), 84
- I2S\_STREAM\_TASK\_PRIO (C macro), 84
- I2S\_STREAM\_TASK\_STACK (C macro), 84
- input\_key\_service\_action\_id\_t (C++ enum), 157
- input\_key\_service\_action\_id\_t::INPUT\_KEY\_SERVICE\_ACTION\_CLICK (C++ enumerator), 157
- input\_key\_service\_action\_id\_t::INPUT\_KEY\_SERVICE\_ACTION\_CLICK\_RELEASE (C++ enumerator), 158
- input\_key\_service\_action\_id\_t::INPUT\_KEY\_SERVICE\_ACTION\_PRESS (C++ enumerator), 158
- input\_key\_service\_action\_id\_t::INPUT\_KEY\_SERVICE\_ACTION\_PRESS\_RELEASE (C++ enumerator), 158
- input\_key\_service\_action\_id\_t::INPUT\_KEY\_SERVICE\_ORDER\_UNKNOWN (C++ enumerator), 157
- input\_key\_service\_add\_key (C++ function), 155
- input\_key\_service\_cfg\_t (C++ struct), 156
- input\_key\_service\_cfg\_t::based\_cfg (C++ member), 156
- input\_key\_service\_cfg\_t::handle (C++ member), 156
- input\_key\_service\_create (C++ function), 155
- INPUT\_KEY\_SERVICE\_DEFAULT\_CONFIG (C macro), 156
- input\_key\_service\_info\_t (C++ struct), 156
- input\_key\_service\_info\_t::act\_id (C++ member), 156
- input\_key\_service\_info\_t::type (C++ member), 156
- input\_key\_service\_info\_t::user\_id (C++ member), 156
- INPUT\_KEY\_SERVICE\_TASK\_ON\_CORE (C macro), 156
- INPUT\_KEY\_SERVICE\_TASK\_PRIORITY (C macro), 156
- INPUT\_KEY\_SERVICE\_TASK\_STACK\_SIZE (C macro), 156
- input\_key\_user\_id\_t (C++ enum), 157
- input\_key\_user\_id\_t::INPUT\_KEY\_USER\_ID\_BATTERY\_CHARGING (C++ enumerator), 158
- input\_key\_user\_id\_t::INPUT\_KEY\_USER\_ID\_CAPTURE (C++ enumerator), 158
- input\_key\_user\_id\_t::INPUT\_KEY\_USER\_ID\_COLOR (C++ enumerator), 158
- input\_key\_user\_id\_t::INPUT\_KEY\_USER\_ID\_MAX (C++ enumerator), 158
- input\_key\_user\_id\_t::INPUT\_KEY\_USER\_ID\_MODE (C++ enumerator), 157
- input\_key\_user\_id\_t::INPUT\_KEY\_USER\_ID\_MSG (C++ enumerator), 158
- input\_key\_user\_id\_t::INPUT\_KEY\_USER\_ID\_MUTE (C++ enumerator), 158
- input\_key\_user\_id\_t::INPUT\_KEY\_USER\_ID\_PLAY (C++ enumerator), 157
- input\_key\_user\_id\_t::INPUT\_KEY\_USER\_ID\_REC (C++ enumerator), 157
- input\_key\_user\_id\_t::INPUT\_KEY\_USER\_ID\_SET (C++ enumerator), 157
- input\_key\_user\_id\_t::INPUT\_KEY\_USER\_ID\_UNKNOWN (C++ enumerator), 157
- input\_key\_user\_id\_t::INPUT\_KEY\_USER\_ID\_VOLDOWN (C++ enumerator), 158
- input\_key\_user\_id\_t::INPUT\_KEY\_USER\_ID\_VOLUP (C++ enumerator), 158
- input\_key\_user\_id\_t::INPUT\_KEY\_USER\_ID\_WAKEUP (C++ enumerator), 158
- insertion loss, 431
- Internet of Things, 431
- Internet radio, 431
- IoT, 431
- IS31FL3216\_CH\_NUM (C macro), 239

**J**

JPEG, [431](#)  
 JPG, [431](#)

**L**

led\_bar\_aw2013\_deinit (C++ function), [180](#)  
 led\_bar\_aw2013\_init (C++ function), [179](#)  
 led\_bar\_aw2013\_pattern (C++ function), [179](#)  
 led\_bar\_aw2013\_set\_blink\_time (C++ function), [180](#)  
 led\_bar\_is31x\_deinit (C++ function), [181](#)  
 led\_bar\_is31x\_init (C++ function), [180](#)  
 led\_bar\_is31x\_pattern (C++ function), [180](#)  
 led\_bar\_ws2812\_deinit (C++ function), [181](#)  
 led\_bar\_ws2812\_handle\_t (C++ type), [182](#)  
 led\_bar\_ws2812\_init (C++ function), [181](#)  
 led\_bar\_ws2812\_pattern (C++ function), [181](#)  
 led\_indicator\_deinit (C++ function), [182](#)  
 led\_indicator\_handle\_t (C++ type), [182](#)  
 led\_indicator\_init (C++ function), [182](#)  
 led\_indicator\_pattern (C++ function), [182](#)  
 Light and Versatile Graphics Library, [431](#)  
 low-pass filter, [432](#)  
 LVGL, [432](#)

**M**

M3U8, [432](#)  
 M4A, [432](#)  
 mass production, [432](#)  
 maximum output power, [432](#)  
 MCLK, [432](#)  
 media\_source\_type\_t (C++ enum), [56](#)  
 media\_source\_type\_t::MEDIA\_SRC\_TYPE\_MUSIC\_A2DP (C++ enumerator), [56](#)  
 media\_source\_type\_t::MEDIA\_SRC\_TYPE\_MUSIC\_BASE (C++ enumerator), [56](#)  
 media\_source\_type\_t::MEDIA\_SRC\_TYPE\_MUSIC\_DINA (C++ enumerator), [56](#)  
 media\_source\_type\_t::MEDIA\_SRC\_TYPE\_MUSIC\_FLASH (C++ enumerator), [56](#)  
 media\_source\_type\_t::MEDIA\_SRC\_TYPE\_MUSIC\_HTTP (C++ enumerator), [56](#)  
 media\_source\_type\_t::MEDIA\_SRC\_TYPE\_MUSIC\_MAX (C++ enumerator), [56](#)  
 media\_source\_type\_t::MEDIA\_SRC\_TYPE\_MUSIC\_RAW (C++ enumerator), [56](#)  
 media\_source\_type\_t::MEDIA\_SRC\_TYPE\_MUSIC\_SD (C++ enumerator), [56](#)  
 media\_source\_type\_t::MEDIA\_SRC\_TYPE\_NULL (C++ enumerator), [56](#)  
 media\_source\_type\_t::MEDIA\_SRC\_TYPE\_RESERVE\_BASE (C++ enumerator), [56](#)  
 media\_source\_type\_t::MEDIA\_SRC\_TYPE\_RESERVE\_MAX (C++ enumerator), [56](#)

media\_source\_type\_t::MEDIA\_SRC\_TYPE\_TONE\_BASE (C++ enumerator), [56](#)  
 media\_source\_type\_t::MEDIA\_SRC\_TYPE\_TONE\_FLASH (C++ enumerator), [56](#)  
 media\_source\_type\_t::MEDIA\_SRC\_TYPE\_TONE\_HTTP (C++ enumerator), [56](#)  
 media\_source\_type\_t::MEDIA\_SRC\_TYPE\_TONE\_MAX (C++ enumerator), [56](#)  
 media\_source\_type\_t::MEDIA\_SRC\_TYPE\_TONE\_SD (C++ enumerator), [56](#)  
 mel-frequency cepstral coefficients, [432](#)  
 mem\_assert (C macro), [52](#)  
 MEMS mic, [432](#)  
 MFCC, [432](#)  
 mic, [432](#)  
 micro-electro-mechanical systems microphone, [432](#)  
 microphone, [432](#)  
 microphone gain, [432](#)  
 microphone hole, [432](#)  
 microSD card, [432](#)  
 MP3, [432](#)  
 mp3\_decoder\_cfg\_t (C++ struct), [129](#)  
 mp3\_decoder\_cfg\_t::id3\_parse\_enable (C++ member), [129](#)  
 mp3\_decoder\_cfg\_t::out\_rb\_size (C++ member), [129](#)  
 mp3\_decoder\_cfg\_t::stack\_in\_ext (C++ member), [129](#)  
 mp3\_decoder\_cfg\_t::task\_core (C++ member), [129](#)  
 mp3\_decoder\_cfg\_t::task\_prio (C++ member), [129](#)  
 mp3\_decoder\_cfg\_t::task\_stack (C++ member), [129](#)  
 mp3\_decoder\_get\_id3\_info (C++ function), [129](#)  
 mp3\_decoder\_init (C++ function), [129](#)  
 MP3\_DECODER\_RINGBUFFER\_SIZE (C macro), [130](#)  
 MP3\_DECODER\_TASK\_CORE (C macro), [130](#)  
 MP3\_DECODER\_TASK\_PRIO (C macro), [130](#)  
 MP3\_DECODER\_TASK\_STACK\_SIZE (C macro), [130](#)  
 MP4, [432](#)  
 multi-room, [432](#)  
 Multi-Room Music, [432](#)  
 MultiNet, [432](#)  
**N**  
 narrowband, [432](#)  
 NB, [433](#)  
 NimBLE, [433](#)  
 noise criteria curve, [433](#)  
 noise floor, [433](#)  
 noise rating curve, [433](#)  
 noise suppression, [433](#)

non-volatile storage, [433](#)

NS, [433](#)

NVS, [433](#)

## O

OGG, [433](#)

ogg\_decoder\_cfg\_t (C++ struct), [130](#)

ogg\_decoder\_cfg\_t::out\_rb\_size (C++ member), [130](#)

ogg\_decoder\_cfg\_t::stack\_in\_ext (C++ member), [131](#)

ogg\_decoder\_cfg\_t::task\_core (C++ member), [130](#)

ogg\_decoder\_cfg\_t::task\_prio (C++ member), [131](#)

ogg\_decoder\_cfg\_t::task\_stack (C++ member), [130](#)

ogg\_decoder\_init (C++ function), [130](#)

OGG\_DECODER\_RINGBUFFER\_SIZE (C macro), [131](#)

OGG\_DECODER\_TASK\_CORE (C macro), [131](#)

OGG\_DECODER\_TASK\_PRIO (C macro), [131](#)

OGG\_DECODER\_TASK\_STACK\_SIZE (C macro), [131](#)

on\_event\_iface\_func (C++ type), [51](#)

OPUS, [433](#)

opus\_decoder\_cfg\_t (C++ struct), [132](#)

opus\_decoder\_cfg\_t::out\_rb\_size (C++ member), [132](#)

opus\_decoder\_cfg\_t::stack\_in\_ext (C++ member), [132](#)

opus\_decoder\_cfg\_t::task\_core (C++ member), [132](#)

opus\_decoder\_cfg\_t::task\_prio (C++ member), [132](#)

opus\_decoder\_cfg\_t::task\_stack (C++ member), [132](#)

OPUS\_DECODER\_RINGBUFFER\_SIZE (C macro), [132](#)

OPUS\_DECODER\_TASK\_CORE (C macro), [132](#)

OPUS\_DECODER\_TASK\_PRIO (C macro), [132](#)

OPUS\_DECODER\_TASK\_STACK\_SIZE (C macro), [132](#)

ota\_app\_get\_default\_proc (C++ function), [171](#)

ota\_data\_get\_default\_proc (C++ function), [171](#)

ota\_data\_image\_stream\_read (C++ function), [171](#)

ota\_data\_partition\_erase\_mark (C++ function), [172](#)

ota\_data\_partition\_write (C++ function), [171](#)

ota\_get\_version\_number (C++ function), [172](#)

ota\_node\_attr\_t (C++ struct), [168](#)

ota\_node\_attr\_t::cert\_pem (C++ member), [169](#)

ota\_node\_attr\_t::label (C++ member), [169](#)

ota\_node\_attr\_t::type (C++ member), [169](#)

ota\_node\_attr\_t::uri (C++ member), [169](#)

ota\_result\_t (C++ struct), [169](#)

ota\_result\_t::id (C++ member), [170](#)

ota\_result\_t::result (C++ member), [170](#)

ota\_service\_config\_t (C++ struct), [168](#)

ota\_service\_config\_t::cb\_ctx (C++ member), [168](#)

ota\_service\_config\_t::evt\_cb (C++ member), [168](#)

ota\_service\_config\_t::task\_core (C++ member), [168](#)

ota\_service\_config\_t::task\_prio (C++ member), [168](#)

ota\_service\_config\_t::task\_stack (C++ member), [168](#)

ota\_service\_create (C++ function), [167](#)

OTA\_SERVICE\_DEFAULT\_CONFIG (C macro), [170](#)

OTA\_SERVICE\_ERR\_REASON\_BASE (C macro), [170](#)

ota\_service\_err\_reason\_t (C++ enum), [170](#)

ota\_service\_err\_reason\_t::OTA\_SERV\_ERR\_REASON\_ERROR (C++ enumerator), [170](#)

ota\_service\_err\_reason\_t::OTA\_SERV\_ERR\_REASON\_ERROR (C++ enumerator), [170](#)

ota\_service\_err\_reason\_t::OTA\_SERV\_ERR\_REASON\_ERROR (C++ enumerator), [170](#)

ota\_service\_err\_reason\_t::OTA\_SERV\_ERR\_REASON\_ERROR (C++ enumerator), [170](#)

ota\_service\_err\_reason\_t::OTA\_SERV\_ERR\_REASON\_ERROR (C++ enumerator), [170](#)

ota\_service\_err\_reason\_t::OTA\_SERV\_ERR\_REASON\_ERROR (C++ enumerator), [170](#)

ota\_service\_err\_reason\_t::OTA\_SERV\_ERR\_REASON\_ERROR (C++ enumerator), [170](#)

ota\_service\_err\_reason\_t::OTA\_SERV\_ERR\_REASON\_ERROR (C++ enumerator), [170](#)

ota\_service\_err\_reason\_t::OTA\_SERV\_ERR\_REASON\_ERROR (C++ enumerator), [170](#)

ota\_service\_err\_reason\_t::OTA\_SERV\_ERR\_REASON\_ERROR (C++ enumerator), [170](#)

ota\_service\_err\_reason\_t::OTA\_SERV\_ERR\_REASON\_ERROR (C++ enumerator), [170](#)

ota\_service\_err\_reason\_t::OTA\_SERV\_ERR\_REASON\_ERROR (C++ enumerator), [170](#)

ota\_service\_err\_reason\_t::OTA\_SERV\_ERR\_REASON\_ERROR (C++ enumerator), [170](#)

ota\_service\_err\_reason\_t::OTA\_SERV\_ERR\_REASON\_ERROR (C++ enumerator), [170](#)

ota\_service\_err\_reason\_t::OTA\_SERV\_ERR\_REASON\_ERROR (C++ enumerator), [170](#)

ota\_service\_event\_type\_t (C++ enum), [170](#)

ota\_service\_event\_type\_t::OTA\_SERV\_EVENT\_TYPE\_FINISHED (C++ enumerator), [170](#)

ota\_service\_event\_type\_t::OTA\_SERV\_EVENT\_TYPE\_RESULT (C++ enumerator), [170](#)

ota\_service\_set\_upgrade\_param (C++ func-

tion), 167  
 ota\_upgrade\_ops\_t (C++ struct), 169  
 ota\_upgrade\_ops\_t::break\_after\_fail (C++ member), 169  
 ota\_upgrade\_ops\_t::execute\_upgrade (C++ member), 169  
 ota\_upgrade\_ops\_t::finished\_check (C++ member), 169  
 ota\_upgrade\_ops\_t::need\_upgrade (C++ member), 169  
 ota\_upgrade\_ops\_t::node (C++ member), 169  
 ota\_upgrade\_ops\_t::prepare (C++ member), 169  
 ota\_upgrade\_ops\_t::reboot\_flag (C++ member), 169

## P

partition\_list\_choose (C++ function), 112  
 partition\_list\_create (C++ function), 111  
 partition\_list\_current (C++ function), 112  
 partition\_list\_destroy (C++ function), 113  
 partition\_list\_exist (C++ function), 112  
 partition\_list\_get\_url\_id (C++ function), 113  
 partition\_list\_get\_url\_num (C++ function), 112  
 partition\_list\_next (C++ function), 111  
 partition\_list\_prev (C++ function), 111  
 partition\_list\_reset (C++ function), 112  
 partition\_list\_save (C++ function), 111  
 partition\_list\_show (C++ function), 113  
 PCM, 433  
 periph\_adc\_button\_cfg\_t (C++ struct), 235  
 periph\_adc\_button\_cfg\_t::arr (C++ member), 235  
 periph\_adc\_button\_cfg\_t::arr\_size (C++ member), 235  
 periph\_adc\_button\_cfg\_t::task\_cfg (C++ member), 235  
 PERIPH\_ADC\_BUTTON\_DEFAULT\_CONFIG (C macro), 236  
 periph\_adc\_button\_event\_id\_t (C++ enum), 236  
 periph\_adc\_button\_event\_id\_t::PERIPH\_ADC\_BUTTON\_IDLE (C++ enumerator), 236  
 periph\_adc\_button\_event\_id\_t::PERIPH\_ADC\_BUTTON\_LONG\_PRESSED (C++ enumerator), 236  
 periph\_adc\_button\_event\_id\_t::PERIPH\_ADC\_BUTTON\_LONG\_RELEASE (C++ enumerator), 236  
 periph\_adc\_button\_event\_id\_t::PERIPH\_ADC\_BUTTON\_PRESSED (C++ enumerator), 236  
 periph\_adc\_button\_event\_id\_t::PERIPH\_ADC\_BUTTON\_RELEASE (C++ enumerator), 236  
 periph\_adc\_button\_init (C++ function), 235

periph\_bluetooth\_cancel\_discover (C++ function), 152  
 periph\_bluetooth\_connect (C++ function), 152  
 periph\_bluetooth\_discover (C++ function), 151  
 periph\_bluetooth\_fast\_forward (C++ function), 151  
 periph\_bluetooth\_get\_a2dp\_sample\_rate (C++ function), 152  
 periph\_bluetooth\_next (C++ function), 151  
 periph\_bluetooth\_pause (C++ function), 151  
 periph\_bluetooth\_play (C++ function), 150  
 periph\_bluetooth\_prev (C++ function), 151  
 periph\_bluetooth\_rewind (C++ function), 151  
 periph\_bluetooth\_stop (C++ function), 151  
 periph\_button\_cfg\_t (C++ struct), 231  
 periph\_button\_cfg\_t::gpio\_mask (C++ member), 231  
 periph\_button\_cfg\_t::long\_press\_time\_ms (C++ member), 231  
 periph\_button\_event\_id\_t (C++ enum), 232  
 periph\_button\_event\_id\_t::PERIPH\_BUTTON\_LONG\_PRESSED (C++ enumerator), 232  
 periph\_button\_event\_id\_t::PERIPH\_BUTTON\_LONG\_RELEASE (C++ enumerator), 232  
 periph\_button\_event\_id\_t::PERIPH\_BUTTON\_PRESSED (C++ enumerator), 232  
 periph\_button\_event\_id\_t::PERIPH\_BUTTON\_RELEASE (C++ enumerator), 232  
 periph\_button\_event\_id\_t::PERIPH\_BUTTON\_UNCHANGED (C++ enumerator), 232  
 periph\_button\_init (C++ function), 231  
 periph\_console\_cfg\_t (C++ struct), 227  
 periph\_console\_cfg\_t::buffer\_size (C++ member), 228  
 periph\_console\_cfg\_t::command\_num (C++ member), 228  
 periph\_console\_cfg\_t::commands (C++ member), 228  
 periph\_console\_cfg\_t::prompt\_string (C++ member), 228  
 periph\_console\_cfg\_t::task\_prio (C++ member), 228  
 periph\_console\_cfg\_t::task\_stack (C++ member), 228  
 periph\_console\_cmd\_t (C++ struct), 227  
 periph\_console\_cmd\_t::cmd (C++ member), 227  
 periph\_console\_cmd\_t::func (C++ member), 227  
 periph\_console\_cmd\_t::help (C++ member), 227  
 periph\_console\_cmd\_t::id (C++ member), 227  
 periph\_console\_init (C++ function), 227



periph\_is31\_shift\_mode\_t (C++ enum), 240  
periph\_is31\_shift\_mode\_t::PERIPH\_IS31\_SHIFT\_MODE\_MEMBER (C++ enumerator), 240  
periph\_is31\_shift\_mode\_t::PERIPH\_IS31\_SHIFT\_MODE\_SINGLE (C++ enumerator), 240  
periph\_is31\_shift\_mode\_t::PERIPH\_IS31\_SHIFT\_MODE\_UNKNOWN (C++ enumerator), 240  
periph\_is31fl3216\_cfg\_t (C++ struct), 239  
periph\_is31fl3216\_cfg\_t::duty (C++ member), 239  
periph\_is31fl3216\_cfg\_t::is31fl3216\_pattern (C++ member), 239  
periph\_is31fl3216\_cfg\_t::state (C++ member), 239  
periph\_is31fl3216\_init (C++ function), 237  
periph\_is31fl3216\_set\_act\_time (C++ function), 239  
periph\_is31fl3216\_set\_blink\_pattern (C++ function), 237  
periph\_is31fl3216\_set\_duty (C++ function), 237  
periph\_is31fl3216\_set\_duty\_step (C++ function), 238  
periph\_is31fl3216\_set\_interval (C++ function), 238  
periph\_is31fl3216\_set\_light\_on\_num (C++ function), 238  
periph\_is31fl3216\_set\_shift\_mode (C++ function), 238  
periph\_is31fl3216\_set\_state (C++ function), 237  
periph\_is31fl3216\_state\_t (C++ enum), 240  
periph\_is31fl3216\_state\_t::IS31FL3216\_STATE\_ADD (C++ enumerator), 240  
periph\_is31fl3216\_state\_t::IS31FL3216\_STATE\_FLASH (C++ enumerator), 240  
periph\_is31fl3216\_state\_t::IS31FL3216\_STATE\_OFF (C++ enumerator), 240  
periph\_is31fl3216\_state\_t::IS31FL3216\_STATE\_ON (C++ enumerator), 240  
periph\_is31fl3216\_state\_t::IS31FL3216\_STATE\_SHIFT (C++ enumerator), 240  
periph\_is31fl3216\_state\_t::IS31FL3216\_STATE\_UNKNOWN (C++ enumerator), 240  
periph\_led\_blink (C++ function), 233  
periph\_led\_cfg\_t (C++ struct), 234  
periph\_led\_cfg\_t::gpio\_num (C++ member), 234  
periph\_led\_cfg\_t::led\_duty\_resolution (C++ member), 234  
periph\_led\_cfg\_t::led\_freq\_hz (C++ member), 234  
periph\_led\_cfg\_t::led\_speed\_mode (C++ member), 234  
periph\_led\_cfg\_t::led\_timer\_num (C++ member), 234  
periph\_led\_event\_id\_t (C++ enum), 234  
periph\_led\_event\_id\_t::PERIPH\_LED\_BLINK\_FINISH (C++ enumerator), 234  
periph\_led\_event\_id\_t::PERIPH\_LED\_UNCHANGE (C++ enumerator), 234  
periph\_led\_idle\_level\_t (C++ enum), 234  
periph\_led\_idle\_level\_t::PERIPH\_LED\_IDLE\_LEVEL\_HIGH (C++ enumerator), 234  
periph\_led\_idle\_level\_t::PERIPH\_LED\_IDLE\_LEVEL\_LOW (C++ enumerator), 234  
periph\_led\_init (C++ function), 233  
periph\_led\_stop (C++ function), 233  
periph\_sdcard\_cfg\_t (C++ struct), 223  
periph\_sdcard\_cfg\_t::card\_detect\_pin (C++ member), 223  
periph\_sdcard\_cfg\_t::mode (C++ member), 223  
periph\_sdcard\_cfg\_t::root (C++ member), 223  
periph\_sdcard\_event\_id\_t (C++ enum), 224  
periph\_sdcard\_event\_id\_t::SDCARD\_STATUS\_CARD\_DETECTED (C++ enumerator), 224  
periph\_sdcard\_event\_id\_t::SDCARD\_STATUS\_MOUNT\_ERROR (C++ enumerator), 224  
periph\_sdcard\_event\_id\_t::SDCARD\_STATUS\_MOUNTED (C++ enumerator), 224  
periph\_sdcard\_event\_id\_t::SDCARD\_STATUS\_UNKNOWN (C++ enumerator), 224  
periph\_sdcard\_event\_id\_t::SDCARD\_STATUS\_UNMOUNT\_ERROR (C++ enumerator), 224  
periph\_sdcard\_event\_id\_t::SDCARD\_STATUS\_UNMOUNTED (C++ enumerator), 224  
periph\_sdcard\_init (C++ function), 223  
periph\_sdcard\_is\_mounted (C++ function), 223  
periph\_sdcard\_mode\_t (C++ enum), 224  
periph\_sdcard\_mode\_t::SD\_MODE\_1\_LINE (C++ enumerator), 224  
periph\_sdcard\_mode\_t::SD\_MODE\_4\_LINE (C++ enumerator), 224  
periph\_sdcard\_mode\_t::SD\_MODE\_8\_LINE (C++ enumerator), 224  
periph\_sdcard\_mode\_t::SD\_MODE\_MAX (C++ enumerator), 224  
periph\_sdcard\_mode\_t::SD\_MODE\_SPI (C++ enumerator), 224  
periph\_service\_callback (C++ function), 190  
periph\_service\_cb (C++ type), 193  
periph\_service\_config\_t (C++ struct), 192  
periph\_service\_config\_t::extern\_stack (C++ member), 192  
periph\_service\_config\_t::service\_destroy (C++ member), 192



periph\_wifi\_config\_mode\_t::WIFI\_CONFIG\_WEP Bayback, 433  
 (C++ enumerator), 222 playlist\_add (C++ function), 113  
 periph\_wifi\_config\_mode\_t::WIFI\_CONFIG\_WEP playlist\_checkout\_by\_id (C++ function), 114  
 (C++ enumerator), 222 playlist\_choose (C++ function), 116  
 periph\_wifi\_config\_start (C++ function), 219 playlist\_create (C++ function), 113  
 periph\_wifi\_config\_wait\_done (C++ function), 219 playlist\_destroy (C++ function), 117  
 playlist\_exist (C++ function), 117  
 periph\_wifi\_init (C++ function), 219 playlist\_get\_current\_list\_id (C++ function), 114  
 periph\_wifi\_is\_connected (C++ function), 219 playlist\_get\_current\_list\_type (C++ function), 114  
 periph\_wifi\_state\_t (C++ enum), 221 playlist\_get\_current\_list\_url (C++ function), 115  
 periph\_wifi\_state\_t::PERIPH\_WIFI\_CONFIG\_DONE playlist\_get\_current\_list\_url\_id (C++ function), 115  
 (C++ enumerator), 221 playlist\_get\_current\_list\_url\_num (C++ function), 115  
 periph\_wifi\_state\_t::PERIPH\_WIFI\_CONNECTED playlist\_get\_list\_num (C++ function), 114  
 (C++ enumerator), 221 playlist\_get\_playlist\_handle\_t (C++ type), 119  
 periph\_wifi\_state\_t::PERIPH\_WIFI\_CONNECTING playlist\_next (C++ function), 115  
 (C++ enumerator), 221 playlist\_operation\_t (C++ struct), 117  
 periph\_wifi\_state\_t::PERIPH\_WIFI\_DISCONNECTED playlist\_operation\_t::choose (C++ member), 118  
 (C++ enumerator), 221 playlist\_operation\_t::current (C++ member), 118  
 periph\_wifi\_state\_t::PERIPH\_WIFI\_ERROR playlist\_operation\_t::destroy (C++ member), 118  
 (C++ enumerator), 221 playlist\_operation\_t::exist (C++ member), 118  
 periph\_wifi\_state\_t::PERIPH\_WIFI\_SETTING playlist\_operation\_t::get\_url\_id (C++ member), 118  
 (C++ enumerator), 221 playlist\_operation\_t::get\_url\_num (C++ member), 118  
 periph\_wifi\_state\_t::PERIPH\_WIFI\_UNCHANGE playlist\_operation\_t::next (C++ member), 118  
 (C++ enumerator), 221 playlist\_operation\_t::prev (C++ member), 118  
 periph\_wifi\_wait\_for\_connected (C++ function), 219 playlist\_operation\_t::remove\_by\_id (C++ member), 118  
 periph\_wpa2\_enterprise\_cfg\_t (C++ struct), 220 playlist\_operation\_t::remove\_by\_url (C++ member), 118  
 periph\_wpa2\_enterprise\_cfg\_t::ca\_pem\_end playlist\_operation\_t::reset (C++ member), 118  
 (C++ member), 220 playlist\_operation\_t::save (C++ member), 118  
 periph\_wpa2\_enterprise\_cfg\_t::ca\_pem\_start playlist\_operation\_t::show (C++ member), 118  
 (C++ member), 220 playlist\_operation\_t::type (C++ member), 118  
 periph\_wpa2\_enterprise\_cfg\_t::diasble\_wpa2\_e 118  
 (C++ member), 220 playlist\_operation\_t::type (C++ member), 118  
 periph\_wpa2\_enterprise\_cfg\_t::eap\_id 118  
 (C++ member), 220 playlist\_operator\_handle\_t (C++ type), 119  
 periph\_wpa2\_enterprise\_cfg\_t::eap\_method playlist\_operator\_t (C++ struct), 118  
 (C++ member), 220 playlist\_operator\_t::get\_operation (C++ member), 119  
 periph\_wpa2\_enterprise\_cfg\_t::eap\_password (C++ member), 118  
 (C++ member), 221 playlist\_operator\_t::playlist (C++ member), 119  
 periph\_wpa2\_enterprise\_cfg\_t::eap\_username 118  
 (C++ member), 220  
 periph\_wpa2\_enterprise\_cfg\_t::wpa2\_e\_cert\_end 118  
 (C++ member), 220  
 periph\_wpa2\_enterprise\_cfg\_t::wpa2\_e\_cert\_start 118  
 (C++ member), 220  
 periph\_wpa2\_enterprise\_cfg\_t::wpa2\_e\_key\_end 118  
 (C++ member), 220  
 periph\_wpa2\_enterprise\_cfg\_t::wpa2\_e\_key\_start 118  
 (C++ member), 220  
 PGA, 433  
 pixel, 433

ber), 119  
 playlist\_prev (C++ function), 116  
 playlist\_remove\_by\_url (C++ function), 116  
 playlist\_remove\_by\_url\_id (C++ function), 117  
 playlist\_reset (C++ function), 116  
 playlist\_save (C++ function), 115  
 playlist\_show (C++ function), 116  
 playlist\_type\_t (C++ enum), 119  
 playlist\_type\_t::PLAYLIST\_DRAM (C++ enumerator), 119  
 playlist\_type\_t::PLAYLIST\_FLASH (C++ enumerator), 119  
 playlist\_type\_t::PLAYLIST\_PARTITION (C++ enumerator), 119  
 playlist\_type\_t::PLAYLIST\_SDCARD (C++ enumerator), 119  
 playlist\_type\_t::PLAYLIST\_UNKNOWN (C++ enumerator), 119  
 process\_func (C++ type), 34  
 programmable gain amplifier, 433  
 protractor, 433  
 pulse-code modulation, 433  
 PWM\_CONFIG\_RINGBUFFER\_SIZE (C macro), 87  
 PWM\_STREAM\_BUF\_SIZE (C macro), 87  
 PWM\_STREAM\_CFG\_DEFAULT (C macro), 87  
 pwm\_stream\_cfg\_t (C++ struct), 86  
 pwm\_stream\_cfg\_t::buffer\_len (C++ member), 87  
 pwm\_stream\_cfg\_t::ext\_stack (C++ member), 87  
 pwm\_stream\_cfg\_t::out\_rb\_size (C++ member), 87  
 pwm\_stream\_cfg\_t::pwm\_config (C++ member), 87  
 pwm\_stream\_cfg\_t::task\_core (C++ member), 87  
 pwm\_stream\_cfg\_t::task\_prio (C++ member), 87  
 pwm\_stream\_cfg\_t::task\_stack (C++ member), 87  
 pwm\_stream\_cfg\_t::type (C++ member), 87  
 PWM\_STREAM\_GPIO\_NUM\_LEFT (C macro), 87  
 PWM\_STREAM\_GPIO\_NUM\_RIGHT (C macro), 87  
 pwm\_stream\_init (C++ function), 85  
 PWM\_STREAM\_RINGBUFFER\_SIZE (C macro), 87  
 pwm\_stream\_set\_clk (C++ function), 85  
 PWM\_STREAM\_TASK\_CORE (C macro), 87  
 PWM\_STREAM\_TASK\_PRIO (C macro), 87  
 PWM\_STREAM\_TASK\_STACK (C macro), 87

## R

raw stream, 433  
 RAW\_STREAM\_CFG\_DEFAULT (C macro), 89

raw\_stream\_cfg\_t (C++ struct), 89  
 raw\_stream\_cfg\_t::out\_rb\_size (C++ member), 89  
 raw\_stream\_cfg\_t::type (C++ member), 89  
 raw\_stream\_init (C++ function), 88  
 raw\_stream\_read (C++ function), 88  
 RAW\_STREAM\_RINGBUFFER\_SIZE (C macro), 89  
 raw\_stream\_write (C++ function), 88  
 RB\_ABORT (C macro), 244  
 rb\_abort (C++ function), 241  
 rb\_bytes\_available (C++ function), 242  
 rb\_bytes\_filled (C++ function), 242  
 rb\_create (C++ function), 241  
 rb\_destroy (C++ function), 241  
 RB\_DONE (C macro), 244  
 rb\_done\_write (C++ function), 243  
 RB\_FAIL (C macro), 244  
 rb\_get\_reader\_holder (C++ function), 243  
 rb\_get\_size (C++ function), 242  
 rb\_get\_writer\_holder (C++ function), 244  
 RB\_OK (C macro), 244  
 rb\_read (C++ function), 242  
 rb\_reset (C++ function), 242  
 rb\_reset\_is\_done\_write (C++ function), 242  
 rb\_set\_reader\_holder (C++ function), 243  
 rb\_set\_writer\_holder (C++ function), 244  
 RB\_TIMEOUT (C macro), 244  
 rb\_unblock\_reader (C++ function), 243  
 rb\_write (C++ function), 243  
 rec\_event\_cb\_t (C++ type), 204  
 recorder\_encoder\_cfg\_t (C++ struct), 207  
 recorder\_encoder\_cfg\_t::encoder (C++ member), 207  
 recorder\_encoder\_cfg\_t::resample (C++ member), 207  
 recorder\_encoder\_create (C++ function), 207  
 recorder\_encoder\_destroy (C++ function), 207  
 recorder\_encoder\_handle\_t (C++ type), 207  
 recorder\_sr\_cfg\_t (C++ struct), 205  
 recorder\_sr\_cfg\_t::afe\_cfg (C++ member), 205  
 recorder\_sr\_cfg\_t::feed\_task\_core (C++ member), 205  
 recorder\_sr\_cfg\_t::feed\_task\_prio (C++ member), 205  
 recorder\_sr\_cfg\_t::feed\_task\_stack (C++ member), 205  
 recorder\_sr\_cfg\_t::fetch\_task\_core (C++ member), 205  
 recorder\_sr\_cfg\_t::fetch\_task\_prio (C++ member), 205  
 recorder\_sr\_cfg\_t::fetch\_task\_stack (C++ member), 205

- recorder\_sr\_cfg\_t::input\_order (C++ member), 205
- recorder\_sr\_cfg\_t::mn\_language (C++ member), 205
- recorder\_sr\_cfg\_t::multinet\_init (C++ member), 205
- recorder\_sr\_cfg\_t::partition\_label (C++ member), 205
- recorder\_sr\_cfg\_t::rb\_size (C++ member), 205
- recorder\_sr\_cfg\_t::wn\_wakeword (C++ member), 206
- recorder\_sr\_create (C++ function), 204
- recorder\_sr\_destroy (C++ function), 204
- recorder\_sr\_handle\_t (C++ type), 206
- recorder\_sr\_reset\_speech\_cmd (C++ function), 204
- resample, 433
- resample filter, 433
- resonant frequency, 434
- reverberation, 434
- RGB, 434
- ring buffer, 434
- ringbuf\_handle\_t (C++ type), 244
- RSP\_FILTER\_BUFFER\_BYTE (C macro), 146
- rsp\_filter\_cfg\_t (C++ struct), 145
- rsp\_filter\_cfg\_t::complexity (C++ member), 145
- rsp\_filter\_cfg\_t::dest\_bits (C++ member), 145
- rsp\_filter\_cfg\_t::dest\_ch (C++ member), 145
- rsp\_filter\_cfg\_t::dest\_rate (C++ member), 145
- rsp\_filter\_cfg\_t::down\_ch\_idx (C++ member), 145
- rsp\_filter\_cfg\_t::max\_indata\_bytes (C++ member), 145
- rsp\_filter\_cfg\_t::mode (C++ member), 145
- rsp\_filter\_cfg\_t::out\_len\_bytes (C++ member), 145
- rsp\_filter\_cfg\_t::out\_rb\_size (C++ member), 146
- rsp\_filter\_cfg\_t::prefer\_flag (C++ member), 146
- rsp\_filter\_cfg\_t::src\_bits (C++ member), 145
- rsp\_filter\_cfg\_t::src\_ch (C++ member), 145
- rsp\_filter\_cfg\_t::src\_rate (C++ member), 145
- rsp\_filter\_cfg\_t::stack\_in\_ext (C++ member), 146
- rsp\_filter\_cfg\_t::task\_core (C++ member), 146
- rsp\_filter\_cfg\_t::task\_prio (C++ member), 146
- rsp\_filter\_cfg\_t::task\_stack (C++ member), 146
- rsp\_filter\_cfg\_t::type (C++ member), 145
- rsp\_filter\_change\_src\_info (C++ function), 144
- rsp\_filter\_init (C++ function), 144
- RSP\_FILTER\_RINGBUFFER\_SIZE (C macro), 146
- rsp\_filter\_set\_src\_info (C++ function), 144
- RSP\_FILTER\_TASK\_CORE (C macro), 146
- RSP\_FILTER\_TASK\_PRIO (C macro), 146
- RSP\_FILTER\_TASK\_STACK (C macro), 146
- ## S
- SBC, 434
- SD card, 434
- sdcard\_list\_choose (C++ function), 103
- sdcard\_list\_create (C++ function), 102
- sdcard\_list\_current (C++ function), 103
- sdcard\_list\_destroy (C++ function), 104
- sdcard\_list\_exist (C++ function), 103
- sdcard\_list\_get\_url\_id (C++ function), 104
- sdcard\_list\_get\_url\_num (C++ function), 104
- sdcard\_list\_next (C++ function), 102
- sdcard\_list\_prev (C++ function), 103
- sdcard\_list\_reset (C++ function), 103
- sdcard\_list\_save (C++ function), 104
- sdcard\_list\_show (C++ function), 102
- sdcard\_scan (C++ function), 101
- sdcard\_scan\_cb\_t (C++ type), 102
- service\_callback (C++ type), 197
- service\_ctrl (C++ type), 197
- service\_event\_t (C++ struct), 195
- service\_event\_t::data (C++ member), 196
- service\_event\_t::len (C++ member), 196
- service\_event\_t::source (C++ member), 196
- service\_event\_t::type (C++ member), 196
- service\_state\_t (C++ enum), 197
- service\_state\_t::SERVICE\_STATE\_CONNECTED (C++ enumerator), 197
- service\_state\_t::SERVICE\_STATE\_CONNECTING (C++ enumerator), 197
- service\_state\_t::SERVICE\_STATE\_IDLE (C++ enumerator), 197
- service\_state\_t::SERVICE\_STATE\_RUNNING (C++ enumerator), 197
- service\_state\_t::SERVICE\_STATE\_STOPPED (C++ enumerator), 197
- service\_state\_t::SERVICE\_STATE\_UNKNOWN (C++ enumerator), 197
- Session Initiation Protocol, 434
- signal-to-echo ratio, 434
- signal-to-noise ratio, 434

- SIP, [434](#)
  - smart\_config\_create (C++ function), [164](#)
  - SMART\_CONFIG\_INFO\_DEFAULT (C macro), [164](#)
  - smart\_config\_info\_t (C++ struct), [164](#)
  - smart\_config\_info\_t::type (C++ member), [164](#)
  - SmartConfig, [434](#)
  - SNR, [434](#)
  - sonic, [434](#)
  - sonic\_cfg\_t (C++ struct), [148](#)
  - sonic\_cfg\_t::out\_rb\_size (C++ member), [149](#)
  - sonic\_cfg\_t::sonic\_info (C++ member), [149](#)
  - sonic\_cfg\_t::stack\_in\_ext (C++ member), [149](#)
  - sonic\_cfg\_t::task\_core (C++ member), [149](#)
  - sonic\_cfg\_t::task\_prio (C++ member), [149](#)
  - sonic\_cfg\_t::task\_stack (C++ member), [149](#)
  - sonic\_info\_t (C++ struct), [148](#)
  - sonic\_info\_t::channel (C++ member), [148](#)
  - sonic\_info\_t::pitch (C++ member), [148](#)
  - sonic\_info\_t::resample\_linear\_interpolate (C++ member), [148](#)
  - sonic\_info\_t::samplerate (C++ member), [148](#)
  - sonic\_info\_t::speed (C++ member), [148](#)
  - sonic\_init (C++ function), [148](#)
  - SONIC\_RINGBUFFER\_SIZE (C macro), [149](#)
  - sonic\_set\_info (C++ function), [147](#)
  - sonic\_set\_pitch\_and\_speed\_info (C++ function), [147](#)
  - SONIC\_SET\_VALUE\_FOR\_INITIALIZATION (C macro), [149](#)
  - SONIC\_TASK\_CORE (C macro), [149](#)
  - SONIC\_TASK\_PRIO (C macro), [149](#)
  - SONIC\_TASK\_STACK (C macro), [149](#)
  - sound card, [434](#)
  - sound level meter, [434](#)
  - sound pickup hole, [434](#)
  - sound pickup tube, [434](#)
  - sound transmission loss, [434](#)
  - source\_info\_init (C++ function), [138](#)
  - speech, [434](#)
  - speech recognition, [434](#)
  - SPI Flash File System, [435](#)
  - SPIFFS, [435](#)
  - SPIFFS stream, [435](#)
  - SPIFFS\_STREAM\_BUF\_SIZE (C macro), [91](#)
  - SPIFFS\_STREAM\_CFG\_DEFAULT (C macro), [91](#)
  - spifffs\_stream\_cfg\_t (C++ struct), [90](#)
  - spifffs\_stream\_cfg\_t::buf\_sz (C++ member), [90](#)
  - spifffs\_stream\_cfg\_t::out\_rb\_size (C++ member), [90](#)
  - spifffs\_stream\_cfg\_t::task\_core (C++ member), [90](#)
  - spifffs\_stream\_cfg\_t::task\_prio (C++ member), [90](#)
  - spifffs\_stream\_cfg\_t::task\_stack (C++ member), [90](#)
  - spifffs\_stream\_cfg\_t::type (C++ member), [90](#)
  - spifffs\_stream\_cfg\_t::write\_header (C++ member), [90](#)
  - spifffs\_stream\_init (C++ function), [90](#)
  - SPIFFS\_STREAM\_RINGBUFFER\_SIZE (C macro), [91](#)
  - SPIFFS\_STREAM\_TASK\_CORE (C macro), [91](#)
  - SPIFFS\_STREAM\_TASK\_PRIO (C macro), [91](#)
  - SPIFFS\_STREAM\_TASK\_STACK (C macro), [91](#)
  - SR, [434](#)
  - SR\_OUTPUT\_RB\_SIZE (C macro), [206](#)
  - STL, [435](#)
  - stream\_func (C++ type), [34](#)
  - subband codec, [435](#)
  - super wide band, [435](#)
  - SWB, [435](#)
- ## T
- tape measure, [435](#)
  - TCP\_SERVER\_DEFAULT\_RESPONSE\_LENGTH (C macro), [93](#)
  - TCP\_STREAM\_BUF\_SIZE (C macro), [93](#)
  - TCP\_STREAM\_CFG\_DEFAULT (C macro), [93](#)
  - tcp\_stream\_cfg\_t (C++ struct), [92](#)
  - tcp\_stream\_cfg\_t::event\_ctx (C++ member), [92](#)
  - tcp\_stream\_cfg\_t::event\_handler (C++ member), [92](#)
  - tcp\_stream\_cfg\_t::ext\_stack (C++ member), [92](#)
  - tcp\_stream\_cfg\_t::host (C++ member), [92](#)
  - tcp\_stream\_cfg\_t::port (C++ member), [92](#)
  - tcp\_stream\_cfg\_t::task\_core (C++ member), [92](#)
  - tcp\_stream\_cfg\_t::task\_prio (C++ member), [92](#)
  - tcp\_stream\_cfg\_t::task\_stack (C++ member), [92](#)
  - tcp\_stream\_cfg\_t::timeout\_ms (C++ member), [92](#)
  - tcp\_stream\_cfg\_t::type (C++ member), [92](#)
  - TCP\_STREAM\_DEFAULT\_PORT (C macro), [93](#)
  - tcp\_stream\_event\_handle\_cb (C++ type), [93](#)
  - tcp\_stream\_event\_msg (C++ struct), [91](#)
  - tcp\_stream\_event\_msg::data (C++ member), [92](#)
  - tcp\_stream\_event\_msg::data\_len (C++ member), [92](#)
  - tcp\_stream\_event\_msg::sock\_fd (C++ member), [92](#)

- tcp\_stream\_event\_msg::source (C++ member), 92
- tcp\_stream\_event\_msg\_t (C++ type), 93
- tcp\_stream\_init (C++ function), 91
- tcp\_stream\_status\_t (C++ enum), 93
- tcp\_stream\_status\_t::TCP\_STREAM\_STATE\_CONNECTED (C++ enumerator), 93
- tcp\_stream\_status\_t::TCP\_STREAM\_STATE\_NONE (C++ enumerator), 93
- TCP\_STREAM\_TASK\_CORE (C macro), 93
- TCP\_STREAM\_TASK\_PRIO (C macro), 93
- TCP\_STREAM\_TASK\_STACK (C macro), 93
- text-to-speech, 435
- THD, 435
- timer\_callback (C++ type), 216
- tolerance, 435
- tone, 435
- TONE\_STREAM\_BUF\_SIZE (C macro), 95
- TONE\_STREAM\_CFG\_DEFAULT (C macro), 95
- tone\_stream\_cfg\_t (C++ struct), 94
- tone\_stream\_cfg\_t::buf\_sz (C++ member), 94
- tone\_stream\_cfg\_t::extern\_stack (C++ member), 95
- tone\_stream\_cfg\_t::label (C++ member), 95
- tone\_stream\_cfg\_t::out\_rb\_size (C++ member), 94
- tone\_stream\_cfg\_t::task\_core (C++ member), 94
- tone\_stream\_cfg\_t::task\_prio (C++ member), 94
- tone\_stream\_cfg\_t::task\_stack (C++ member), 94
- tone\_stream\_cfg\_t::type (C++ member), 94
- tone\_stream\_cfg\_t::use\_delegate (C++ member), 95
- TONE\_STREAM\_EXT\_STACK (C macro), 95
- tone\_stream\_init (C++ function), 94
- TONE\_STREAM\_RINGBUFFER\_SIZE (C macro), 95
- TONE\_STREAM\_TASK\_CORE (C macro), 95
- TONE\_STREAM\_TASK\_PRIO (C macro), 95
- TONE\_STREAM\_TASK\_STACK (C macro), 95
- TONE\_STREAM\_USE\_DELEGATE (C macro), 95
- total harmonic distortion, 435
- TTS, 435
- TTS\_STREAM\_BUF\_SIZE (C macro), 100
- TTS\_STREAM\_CFG\_DEFAULT (C macro), 100
- tts\_stream\_cfg\_t (C++ struct), 99
- tts\_stream\_cfg\_t::buf\_sz (C++ member), 99
- tts\_stream\_cfg\_t::ext\_stack (C++ member), 99
- tts\_stream\_cfg\_t::out\_rb\_size (C++ member), 99
- tts\_stream\_cfg\_t::task\_core (C++ member), 99
- tts\_stream\_cfg\_t::task\_prio (C++ member), 99
- tts\_stream\_cfg\_t::task\_stack (C++ member), 99
- tts\_stream\_cfg\_t::type (C++ member), 99
- ttstream\_get\_speed (C++ function), 98
- tts\_stream\_init (C++ function), 98
- TTS\_STREAM\_RINGBUFFER\_SIZE (C macro), 100
- tts\_stream\_set\_speed (C++ function), 98
- tts\_stream\_set\_strings (C++ function), 98
- TTS\_STREAM\_TASK\_CORE (C macro), 100
- TTS\_STREAM\_TASK\_PRIO (C macro), 100
- TTS\_STREAM\_TASK\_STACK (C macro), 100
- tts\_voice\_speed\_t (C++ enum), 100
- tts\_voice\_speed\_t::TTS\_VOICE\_SPEED\_0 (C++ enumerator), 100
- tts\_voice\_speed\_t::TTS\_VOICE\_SPEED\_1 (C++ enumerator), 100
- tts\_voice\_speed\_t::TTS\_VOICE\_SPEED\_2 (C++ enumerator), 100
- tts\_voice\_speed\_t::TTS\_VOICE\_SPEED\_3 (C++ enumerator), 100
- tts\_voice\_speed\_t::TTS\_VOICE\_SPEED\_4 (C++ enumerator), 100
- tts\_voice\_speed\_t::TTS\_VOICE\_SPEED\_5 (C++ enumerator), 100
- tts\_voice\_speed\_t::TTS\_VOICE\_SPEED\_MAX (C++ enumerator), 100
- ## V
- VAD, 435
- voice activity detection, 435
- VoIP, 435
- vol\_monitor\_create (C++ function), 185
- vol\_monitor\_destroy (C++ function), 185
- vol\_monitor\_event\_cb (C++ type), 187
- vol\_monitor\_event\_t (C++ enum), 187
- vol\_monitor\_event\_t::VOL\_MONITOR\_EVENT\_BAT\_FULL (C++ enumerator), 187
- vol\_monitor\_event\_t::VOL\_MONITOR\_EVENT\_BAT\_LOW (C++ enumerator), 187
- vol\_monitor\_event\_t::VOL\_MONITOR\_EVENT\_FREQ\_REPORT (C++ enumerator), 187
- vol\_monitor\_handle\_t (C++ type), 187
- vol\_monitor\_param\_t (C++ struct), 186
- vol\_monitor\_param\_t::deinit (C++ member), 186
- vol\_monitor\_param\_t::init (C++ member), 186
- vol\_monitor\_param\_t::read\_freq (C++ member), 186
- vol\_monitor\_param\_t::report\_freq (C++ member), 187

- vol\_monitor\_param\_t::user\_data (C++ member), 186
  - vol\_monitor\_param\_t::vol\_full\_threshold (C++ member), 187
  - vol\_monitor\_param\_t::vol\_get (C++ member), 186
  - vol\_monitor\_param\_t::vol\_low\_threshold (C++ member), 187
  - vol\_monitor\_set\_event\_cb (C++ function), 185
  - vol\_monitor\_set\_report\_freq (C++ function), 186
  - vol\_monitor\_start\_freq\_report (C++ function), 185
  - vol\_monitor\_stop\_freq\_report (C++ function), 186
- W**
- wake word, 435
  - wake word engine, 435
  - wake-up, 436
  - WakeNet, 435
  - wav\_decoder\_cfg\_t (C++ struct), 133
  - wav\_decoder\_cfg\_t::out\_rb\_size (C++ member), 133
  - wav\_decoder\_cfg\_t::stack\_in\_ext (C++ member), 133
  - wav\_decoder\_cfg\_t::task\_core (C++ member), 133
  - wav\_decoder\_cfg\_t::task\_prio (C++ member), 133
  - wav\_decoder\_cfg\_t::task\_stack (C++ member), 133
  - wav\_decoder\_init (C++ function), 133
  - WAV\_DECODER\_RINGBUFFER\_SIZE (C macro), 134
  - WAV\_DECODER\_TASK\_CORE (C macro), 134
  - WAV\_DECODER\_TASK\_PRIO (C macro), 134
  - WAV\_DECODER\_TASK\_STACK (C macro), 134
  - wav\_encoder\_cfg\_t (C++ struct), 134
  - wav\_encoder\_cfg\_t::out\_rb\_size (C++ member), 134
  - wav\_encoder\_cfg\_t::stack\_in\_ext (C++ member), 135
  - wav\_encoder\_cfg\_t::task\_core (C++ member), 134
  - wav\_encoder\_cfg\_t::task\_prio (C++ member), 134
  - wav\_encoder\_cfg\_t::task\_stack (C++ member), 134
  - wav\_encoder\_init (C++ function), 134
  - WAV\_ENCODER\_RINGBUFFER\_SIZE (C macro), 135
  - WAV\_ENCODER\_TASK\_CORE (C macro), 135
  - WAV\_ENCODER\_TASK\_PRIO (C macro), 135
  - WAV\_ENCODER\_TASK\_STACK (C macro), 135
  - WB, 436
  - wideband, 436
  - wifi\_service\_config\_t (C++ struct), 159
  - wifi\_service\_config\_t::cb\_ctx (C++ member), 160
  - wifi\_service\_config\_t::evt\_cb (C++ member), 160
  - wifi\_service\_config\_t::extern\_stack (C++ member), 159
  - wifi\_service\_config\_t::max\_prov\_retry\_time (C++ member), 160
  - wifi\_service\_config\_t::max\_retry\_time (C++ member), 160
  - wifi\_service\_config\_t::max\_ssid\_num (C++ member), 160
  - wifi\_service\_config\_t::setting\_timeout\_s (C++ member), 160
  - wifi\_service\_config\_t::task\_core (C++ member), 159
  - wifi\_service\_config\_t::task\_prio (C++ member), 159
  - wifi\_service\_config\_t::task\_stack (C++ member), 159
  - wifi\_service\_config\_t::user\_data (C++ member), 160
  - wifi\_service\_connect (C++ function), 159
  - wifi\_service\_create (C++ function), 159
  - WIFI\_SERVICE\_DEFAULT\_CONFIG (C macro), 160
  - wifi\_service\_destroy (C++ function), 159
  - wifi\_service\_disconnect (C++ function), 159
  - wifi\_service\_disconnect\_reason\_get (C++ function), 159
  - wifi\_service\_disconnect\_reason\_t (C++ enum), 161
  - wifi\_service\_disconnect\_reason\_t::WIFI\_SERV\_STA\_AP (C++ enumerator), 161
  - wifi\_service\_disconnect\_reason\_t::WIFI\_SERV\_STA\_AU (C++ enumerator), 161
  - wifi\_service\_disconnect\_reason\_t::WIFI\_SERV\_STA\_BY (C++ enumerator), 161
  - wifi\_service\_disconnect\_reason\_t::WIFI\_SERV\_STA\_CO (C++ enumerator), 161
  - wifi\_service\_disconnect\_reason\_t::WIFI\_SERV\_STA\_SE (C++ enumerator), 161
  - wifi\_service\_disconnect\_reason\_t::WIFI\_SERV\_STA\_UN (C++ enumerator), 161
  - wifi\_service\_erase\_ssid\_manager\_info (C++ function), 159
  - wifi\_service\_event\_t (C++ enum), 160
  - wifi\_service\_event\_t::WIFI\_SERV\_EVENT\_CONNECTED (C++ enumerator), 160
  - wifi\_service\_event\_t::WIFI\_SERV\_EVENT\_CONNECTING (C++ enumerator), 160
  - wifi\_service\_event\_t::WIFI\_SERV\_EVENT\_DISCONNECTED (C++ enumerator), 160



wifi\_service\_event\_t::WIFI\_SERV\_EVENT\_SETTING\_FAILED  
 (C++ *enumerator*), 161  
 wifi\_service\_event\_t::WIFI\_SERV\_EVENT\_SETTING\_FINISHED  
 (C++ *enumerator*), 161  
 wifi\_service\_event\_t::WIFI\_SERV\_EVENT\_SETTING\_TIMEOUT  
 (C++ *enumerator*), 160  
 wifi\_service\_event\_t::WIFI\_SERV\_EVENT\_UNKNOWN  
 (C++ *enumerator*), 160  
 wifi\_service\_get\_last\_ssid\_cfg (C++ *function*),  
 159  
 wifi\_service\_register\_setting\_handle  
 (C++ *function*), 159  
 wifi\_service\_set\_sta\_info (C++ *function*),  
 159  
 wifi\_service\_setting\_start (C++ *function*),  
 159  
 wifi\_service\_setting\_stop (C++ *function*),  
 159  
 wifi\_service\_state\_get (C++ *function*), 159  
 wifi\_service\_update\_sta\_info (C++ *function*),  
 159  
 wifi\_setting\_func (C++ *type*), 163  
 wifi\_setting\_teardown\_func (C++ *type*), 163  
 wifi\_ssid\_manager\_create (C++ *function*), 167  
 wifi\_ssid\_manager\_destroy (C++ *function*),  
 167  
 wifi\_ssid\_manager\_erase\_all (C++ *function*),  
 167  
 wifi\_ssid\_manager\_get\_best\_config (C++  
*function*), 167  
 wifi\_ssid\_manager\_get\_latest\_config  
 (C++ *function*), 167  
 wifi\_ssid\_manager\_get\_ssid\_num (C++ *function*),  
 167  
 wifi\_ssid\_manager\_handle\_t (C++ *type*), 167  
 wifi\_ssid\_manager\_list\_show (C++ *function*),  
 167  
 wifi\_ssid\_manager\_save (C++ *function*), 167  
 WWE, 436

## Y

YUV, 436

## Z

zl38063\_codec\_config\_i2s (C++ *function*), 262  
 zl38063\_codec\_ctrl\_state (C++ *function*), 262  
 zl38063\_codec\_deinit (C++ *function*), 262  
 zl38063\_codec\_get\_voice\_volume (C++ *function*),  
 263  
 zl38063\_codec\_init (C++ *function*), 262  
 zl38063\_codec\_set\_voice\_mute (C++ *function*),  
 263  
 zl38063\_codec\_set\_voice\_volume (C++ *function*),  
 263