Read the Docs Template Documentation

发布 *v4.1*

Read the Docs

2020 年 08 月 17 日

Contents

1 快速入门 **3**

1.1 概述 . . . 3
1.2 准备工作 . . . 4
1.3 开发板简介 . . . 4
1.4 详细安装步骤 . . . 77
1.5 第一步：安装准备 . . . 77
1.6 第二步：获取 ESP-IDF . . . 88
1.7 第三步：设置工具 . . . 89
1.8 第四步：设置环境变量 . . . 90
1.9 第五步：开始创建工程 . . . 91
1.10 第六步：连接设备 . . . 91
1.11 第七步：配置 . . . 92
1.12 第八步：编译工程 . . . 93
1.13 第九步：烧录到设备 . . . 93
1.14 第十步：监视器 . . . 95
1.15 更新 ESP-IDF . . . 96
1.16 相关文档 . . . 96

2 API 参考 **143**

2.1 蓝牙 API . . . 143
2.2 连网 API . . . 653
2.3 外设 API . . . 803
2.4 应用层协议 . . . 1125
2.5 配网 API . . . 1262
2.6 存储 API . . . 1305
2.7 System API . . . 1413
2.8 Project Configuration . . . 1736

2.9 Error Codes Reference . . . 1974

3 ESP32 H/W 硬件参考 1981

3.1 ESP32 系列模组和开发板 . . . 1981
3.2 Previous Versions of ESP32 Modules and Boards . . . 1988

4 ESP32-S2 Beta Preview Support 1999

4.1 Documentation . . . 1999
4.2 Setting up for ESP32-S2 Beta . . . 1999
4.3 Building a project for ESP32-S2 Beta . . . 2000
4.4 Limitations . . . 2001
4.5 Roadmap . . . 2001
4.6 Issues . . . 2001

5 API 指南 2003

5.1 ESP-IDF 编程注意事项 . . . 2003
5.2 构建系统（CMake 版） . . . 2009
5.3 构建系统 (传统 GNU Make) . . . 2036
5.4 错误处理 . . . 2050
5.5 严重错误 . . . 2053
5.6 Event Handling . . . 2061
5.7 Deep Sleep Wake Stubs . . . 2064
5.8 ESP32 Core Dump . . . 2067
5.9 Flash 加密 . . . 2070
5.10 ESP-IDF FreeRTOS SMP Changes . . . 2087
5.11 Thread Local Storage . . . 2096
5.12 High-Level Interrupts . . . 2097
5.13 JTAG 调试 . . . 2099
5.14 引导加载程序（Bootloader） . . . 2163
5.15 分区表 . . . 2165
5.16 Secure Boot . . . 2169
5.17 Secure Boot V2 . . . 2178
5.18 ULP 协处理器编程 . . . 2185
5.19 ULP 协处理器 (传统的 GNU Make) . . . 2240
5.20 ESP32 中的单元测试 . . . 2246
5.21 单元测试（传统的 GNU Make） . . . 2251
5.22 控制台终端 . . . 2256
5.23 ESP32 ROM console . . . 2259
5.24 RF calibration . . . 2262
5.25 Wi-Fi Driver . . . 2263
5.26 ESP-BLE-MESH . . . 2320
5.27 ESP-MESH . . . 2368
5.28 BluFi . . . 2396

5.29 片外 RAM . . . 2406
5.30 链接脚本生成机制 . . . 2409
5.31 lwIP . . . 2420
5.32 工具 . . . 2430

6 Libraries and Frameworks 2435
6.1 Cloud Frameworks . . . 2435

7 Contributions Guide 2437
7.1 How to Contribute . . . 2437
7.2 Before Contributing . . . 2437
7.3 Pull Request Process . . . 2438
7.4 Legal Part . . . 2438
7.5 Related Documents . . . 2438

8 ESP-IDF 版本简介 2467
8.1 发布版本 . . . 2467
8.2 我该选择哪个版本? . . . 2468
8.3 版本管理 . . . 2469
8.4 查看当前版本 . . . 2469
8.5 Git 工作流 . . . 2470
8.6 更新 ESP-IDF . . . 2471

9 资源 2475

10 Copyrights and Licenses 2477
10.1 Software Copyrights . . . 2477
10.2 ROM Source Code Copyrights . . . 2479
10.3 Xtensa libhal MIT License . . . 2479
10.4 TinyBasic Plus MIT License . . . 2480
10.5 TJpgDec License . . . 2480

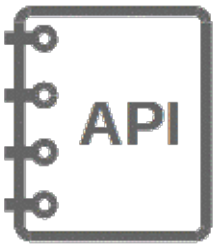
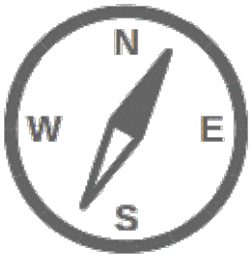
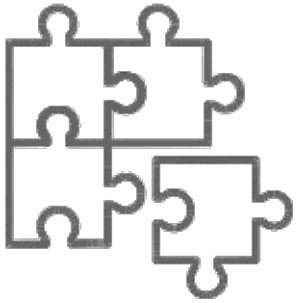
11 关于本指南 2481

12 Switch Between Languages/切换语言 2483

索引 2485

[English]

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

		
快速入门	API 参考	H/W 参考
		
API 指南	贡献代码	相关资源

CHAPTER 1

快速入门

[English]

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

注解： 这是 ESP-IDF 稳定版本 v4.1 的文档，还有其他版本的文档*ESP-IDF 版本简介* 供参考。

1.1 概述

ESP32 SoC 芯片支持以下功能：

- 2.4 GHz Wi-Fi
- 蓝牙 4.2
- 高性能双核
- 超低功耗协处理器
- 多种外设

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

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

1.2 准备工作

硬件：

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

软件：

- 设置 **工具链**，用于编译 ESP32 代码；
- **编译工具**——CMake 和 Ninja 编译工具，用于编译 ESP32 **应用程序**；
- 获取 **ESP-IDF** 软件开发框架。该框架已经基本包含 ESP32 使用的 API（软件库和源代码）和运行 **工具链**的脚本；
- 安装 C 语言编程（**工程**）的 **文本编辑器**，例如 Eclipse。

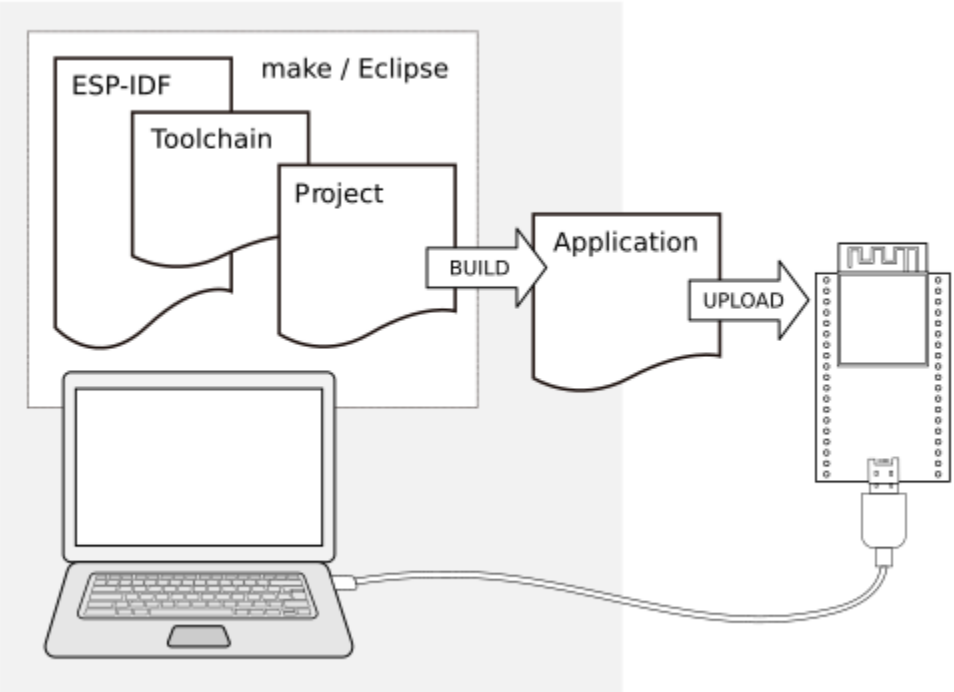


图 1: ESP32 应用程序开发

1.3 开发板简介

请点击下方连接，了解有关具体开发板的详细信息。

1.3.1 ESP32-DevKitC V4 入门指南

[English]

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

准备工作

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

您可以跳过介绍部分，直接前往*应用程序开发* 章节。

概述

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

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

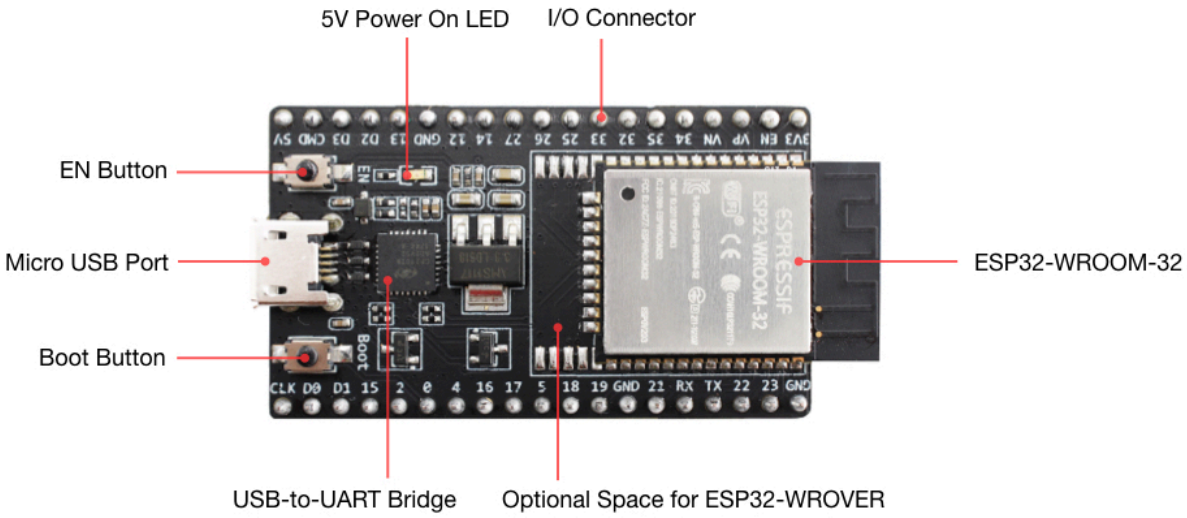
- 可选多款 ESP32 模组
  - *ESP32-WROOM-32* 模组
  - *ESP32-WROOM-32D*
  - *ESP32-WROOM-32U*
  - *ESP32-SOLO-1* 模组
  - *ESP32-WROVER*
  - *ESP32-WROVER-B*
  - *ESP32-WROVER-I*
  - *ESP32-WROVER-I (IPEX)*
- 可选排针或排母

详情请见 《乐鑫产品订购信息》。

功能说明

ESP32-DevKitC V4 开发板的主要组件、接口及控制方式见下。

ESP32-DevKitC V4（板载 ESP32-WROOM-32）



主要组件	基本介绍
ESP32-WROOM-32	基于 ESP32 的模组。更多详情，请见 《ESP32-WROOM-32 技术规格书》 。
EN	复位按键。
Boot	下载按键。按下 **Boot** 键并保持，同时按一下 **EN** 键（此时不要松开 **Boot** 键）进入“固件下载”模式，通过串口下载固件。
USB-to-UART 桥接器 | 单芯片 USB-UART 桥接器，可提供高达 3 Mbps 的传输速率。	Micro USB 端口
USB 接口。	可用作电路板的供电电源，或连接 PC 和 ESP32-WROOM-32 模组的通信接口。
5V Power On LED	开发板通电后（USB 或外部 5 V），该指示灯将亮起。更多信息，请见相关文档 中的原理图。
I/O	板上模组的绝大部分管脚均已引出至开发板的排针。用户可以对 ESP32 进行编程，实现 PWM、ADC、DAC、I2C、I2S、SPI 等多种功能。

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

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

电源选项

开发板可从以下三种供电方式中任选其一：

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

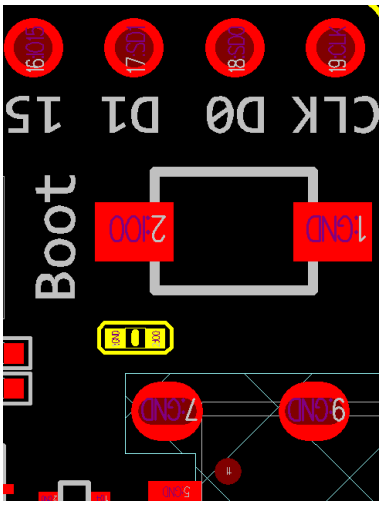
警告： 上述供电模式 **不可同时连接**，否则可能会损坏开发板和/或电源。

有关 C15 的提示

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

- 开发板上电后可能进入下载模式；
- 如果用户通过 GPIO0 输出时钟，C15 可能会影响信号。

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



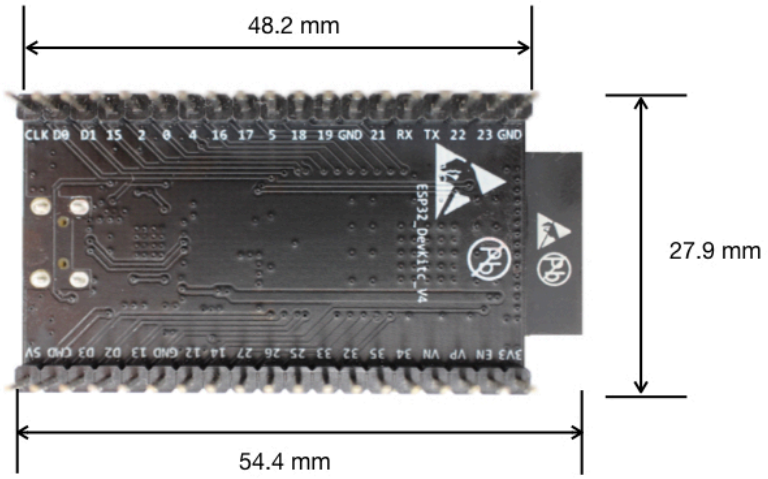
C15（黄色）在 ESP32-DevKitC V4 开发板上的位置

应用程序开发

ESP32-DevKitC V4 上电前，请首先确认开发板完好无损。

现在，请前往*ESP32 H/W 硬件参考* 中的详细安装步骤 章节，查看如何设置开发环境，并尝试将示例项目烧录至您的开发板。

开发板尺寸



ESP32-DevKitC 开发板尺寸 – 仰视图

相关文档

- ESP32-DevKitC V4 原理图 (PDF)
- 《ESP32 技术规格书》 (PDF)
- 《ESP32-WROOM-32 技术规格书》 (PDF)
- 《ESP32-WROOM-32D & ESP32-WROOM-32U 技术规格书》 (PDF)
- 《ESP32-WROVER 技术规格书》 (PDF)
- 《ESP32-WROVER-B 技术规格书》 (PDF)
- 《乐鑫产品订购信息》 (PDF)

ESP32-DevKitC V2 入门指南

[English]

本指南介绍了如何开始使用 ESP32-DevKitC V2 开发板。

准备工作

- *ESP32-DevKitC V2* 开发板
- USB A / micro USB B 数据线
- PC（Windows、Linux 或 Mac OS）

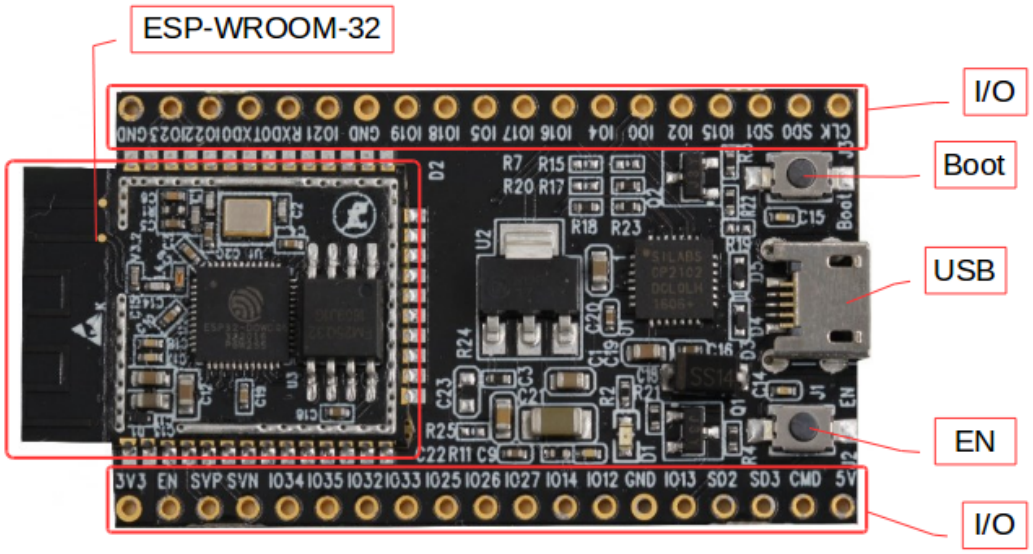
您可以跳过介绍部分，直接前往应用程序开发 章节。

概述

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

功能说明

ESP32-DevKitC V2 开发板的主要组件、接口及控制方式见下。



ESP32-DevKitC V2 开发板

主要组件	基本介绍
ESP32-WROOM-32	基于 ESP32 的模组。更多详情，请见 《ESP32-WROOM-32 技术规格书》 。
EN	复位按键。
Boot	下载按键。按下 **Boot** 键并保持，同时按一下 **EN** 键（此时不要松开 **Boot** 键）进入“固件下载”模式，通过串口下载固件。
Micro USB 端口	USB 接口。可用作电路板的供电电源，或连接 PC 和 ESP32-WROOM-32 模组的通信接口。
I/O	板上模组的绝大部分管脚均已引出至开发板的排针。用户可以对 ESP32 进行编程，实现 PWM、ADC、DAC、I2C、I2S、SPI 等多种功能。

电源选项

开发板可从以下三种供电方式中任选其一：

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

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

应用程序开发

ESP32-DevKitC V2 上电前，请首先确认开发板完好无损。

现在，请前往*ESP32 H/W 硬件参考* 中的*详细安装步骤* 章节，查看如何设置开发环境，并尝试将示例项目烧录至您的开发板。

相关文档

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

1.3.2 ESP-WROVER-KIT V4.1 入门指南

[English]

本指南介绍了如何开始使用 ESP-WROVER-KIT V4.1 开发板及其功能和相关配置。有关 ESP-WROVER-KIT 其他版本的介绍，请见：*ESP32 H/W 硬件参考*。

准备工作

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

您可以跳过介绍部分，直接前往应用程序开发 章节。

概述

ESP-WROVER-KIT 是 乐鑫 一款基于 ESP32 的开发板。

ESP-WROVER-KIT 开发板已集成了如下组件：

- ESP32-WROVER-B 模组
- LCD 屏
- MicroSD 卡槽

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

为了便于使用，板上模组的绝大部分管脚均已引出至开发板的引脚。

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

功能概述

ESP-WROVER-KIT 开发板的主要组件和连接方式如下图所示。

功能说明

ESP-WROVER-KIT 开发板的主要组件、接口及控制方式见下。

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

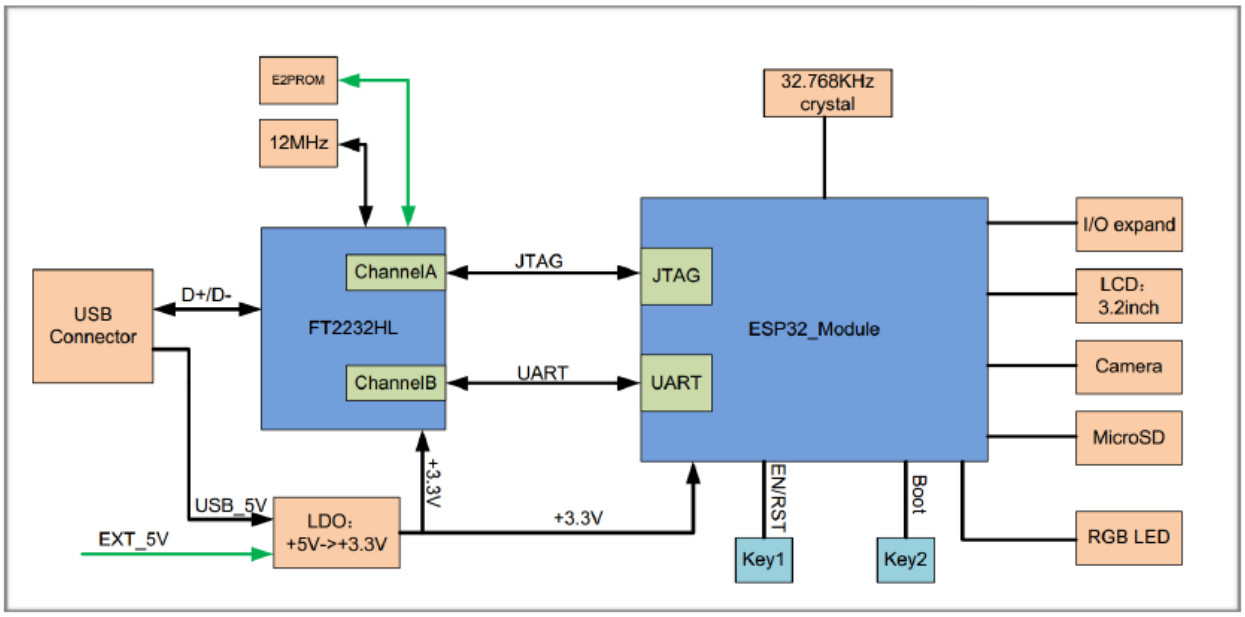


图 2: ESP-WROVER-KIT 功能框图

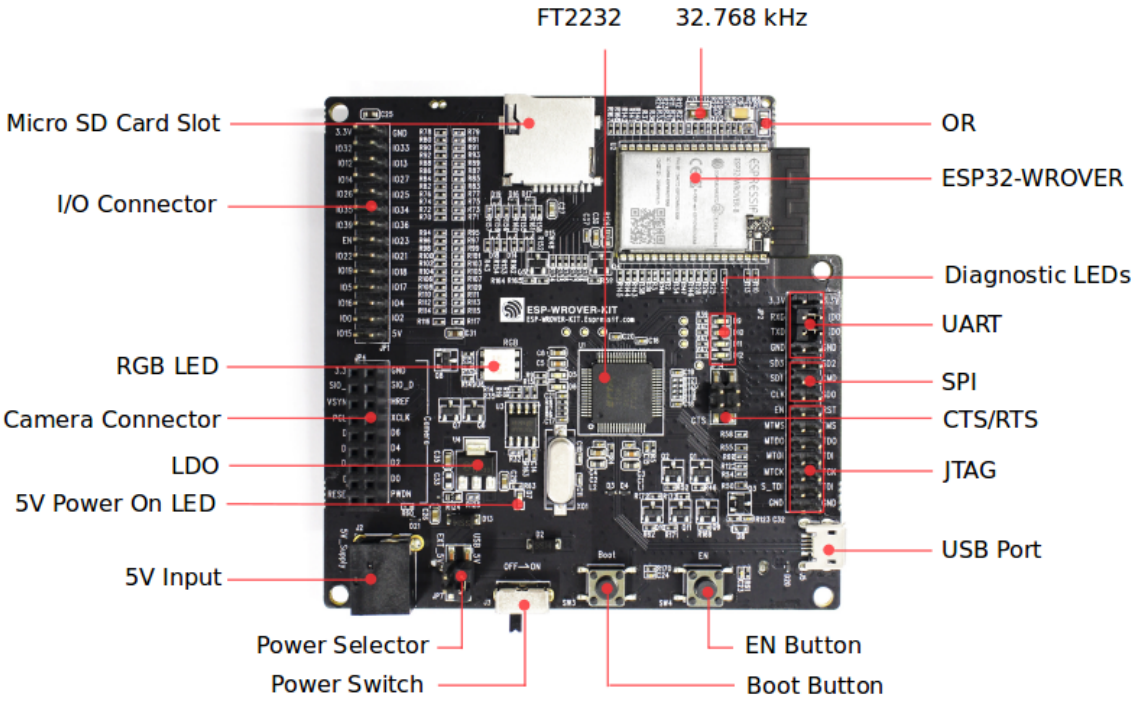


图 3: ESP-WROVER-KIT 开发板布局 – 俯视图

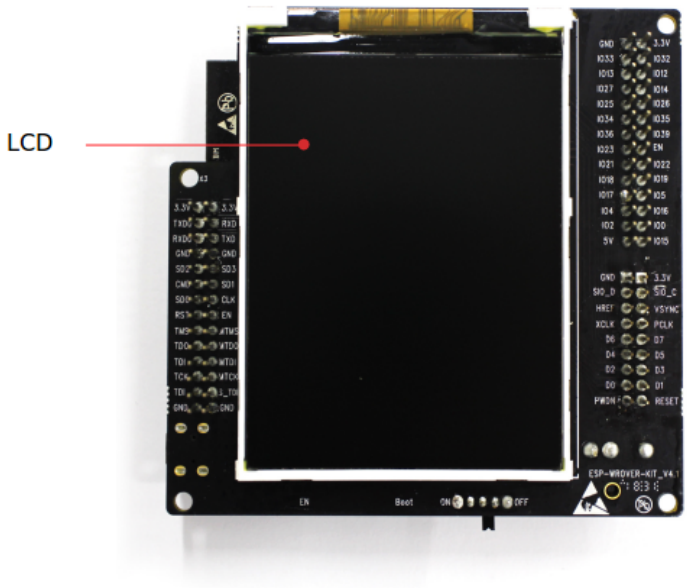
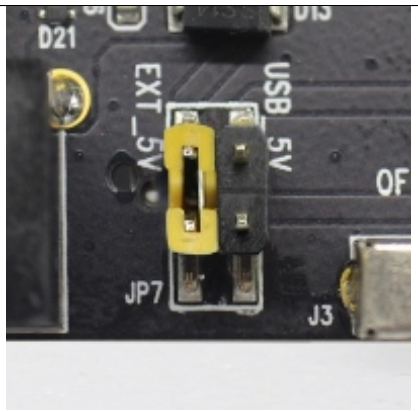
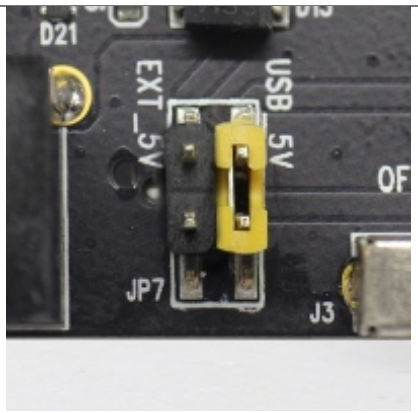
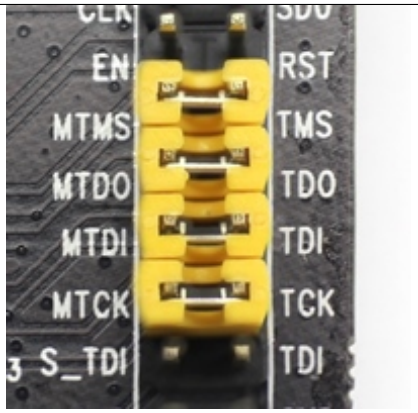
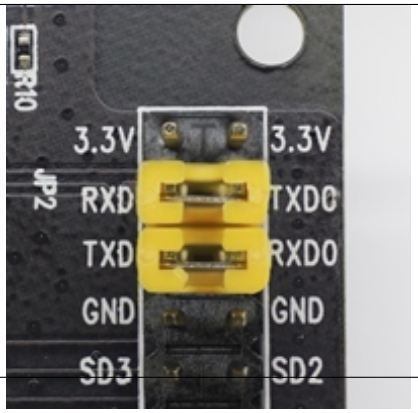


图 4: ESP-WROVER-KIT 开发板布局 – 仰视图

主要组件	基本介绍
FT2232	FT2232 多协议 USB 转串口桥接器。开发人员可通过 USB 接口对 FT2232 芯片进行控制和编程，与 ESP32 建立连接。FT2232 芯片可在通道 A 提供 USB-to-JTAG 接口功能，并在通道 B 提供 USB-to-Serial 接口功能，便利开发人员的应用开发与调试。详见 ESP-WROVER-KIT V4.1 原理图。
32.768 kHz	32.768 kHz 晶振，可提供 Deep-sleep 下使用的低功耗时钟。
0 欧电阻	ESP-WROVER-KIT 开发板设计了一个 0 欧电阻，可在测量 ESP32 系列模组在不同功耗模式下的电流时，直接移除或替换为分流器。
ESP32-WROVER-B 模组	这款 ESP32 模组内置 64-Mbit PSRAM，可提供灵活的额外存储空间和数据处理能力。
诊断 LED 信号灯	本开发板 FT2232 芯片的 GPIO 管脚连接了 4 个红色 LED 信号灯，以备后用。
UART	串口。FT2232 和 ESP32 的串行 TX/RX 信号已引出至 JP2 的两端。默认情况下，这两路信号由跳线帽连接。如果仅需使用 ESP32 模组串口，则可移除相关跳线帽，将模组连接至其他外部串口设备。
SPI	默认情况下，ESP32 使用 SPI 接口访问内置 flash 和 PSRAM。使用这些引脚连接 ESP32 和其他 SPI 设备。这种情况下，需增加额外的片选 (CS) 信号。注意，本接口的工作电压为 3.3 V。
CTS/RTS	串口流控信号。管脚默认不连接至电路。为了使能该功能，必须用跳线帽断路掉 JP14 的相应管脚。
JTAG	JTAG 接口。FT2232 和 ESP32 的 JTAG 信号已引出至 JP2 的两端。默认情况下，这两路信号不连接。如需使能 JTAG，请按照设置选项 的介绍，连接跳线帽。
USB 端口	USB 接口。可用作开发板的供电电源，或连接 PC 和开发板的通信接口。
EN	复位按键。
BOOT 按键	下载按键。按下 **Boot** 键并保持，同时按一下 **EN** 键（此时不要松开 **Boot** 键）进入“固件下载”模式，通过串口下载固件。

设置选项

用户可通过 3 组排针，设置开发板功能，其中常见功能见下表:

排针	跳线设置	功能描述
JP7		使用外部电源为 ESP-WROVER-KIT 开发板供电
JP7		使用 USB 端口为 ESP-WROVER-KIT 开发板供电
JP2		使能 JTAG 功能
		使能 UART 通信

ESP32 管脚分配

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

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

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

更多外设共享管脚的介绍，请见下一章节中的表格。

主要 I/O 连接器 / JP1

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

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

说明：

- NC/XTAL - *32.768 kHz 晶振*
- JTAG - *JTAG / JP2*
- Boot - Boot 按键 / SW2
- 摄像头 - *摄像头 / JP4*

- LED - *RGB LED*
- MicroSD - *MicroSD Card / J4*
- LCD - *LCD / U5*
- PSRAM - ESP32-WROVER-B 的 PSRAM

32.768 kHz 晶振

.	ESP32 管脚
1.	GPIO32
2.	GPIO33

注解： 默认情况下，管脚 GPIO32 和 GPIO33 已连接至晶振。因此，为了保证信号的完整性，这两个管脚并未连接至 JP1 I/O 连接器。用户可通过将 R11/R23 处的 0 欧电阻移至 R12/R24 处，以将 GP1O32 和 GPIO33 的连接从晶振移至 JP1。

SPI Flash / JP2

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

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

JTAG / JP2

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

摄像头 / JP4

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

- D0 到 D7 为摄像头的数据总线

RGB LED

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

MicroSD 卡

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

LCD / U5

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

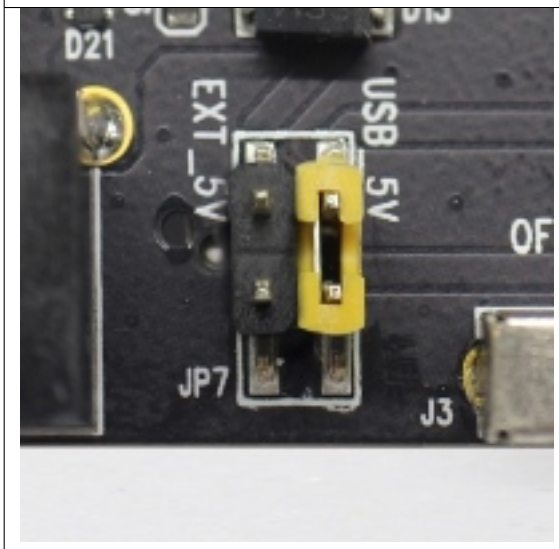
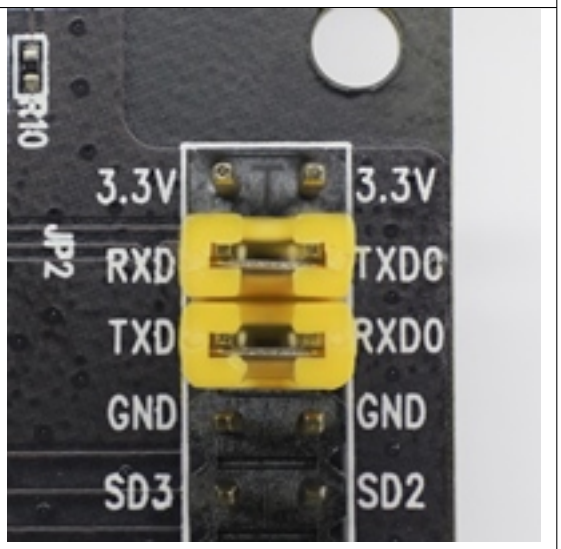
应用程序开发

ESP-WROVER-KIT 上电前，请首先确认开发板完好无损。

初始设置

请严格按照下图所示连接跳线帽，注意不要额外连接其他跳线帽。

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

USB 供电	使能 UART 通信
	

注意不要连接其他跳线帽。

打开 **电源开关**，**5V Power On LED** 应点亮。

正式开始开发

现在，请前往*ESP32 H/W 硬件参考* 中的*详细安装步骤* 章节，查看如何设置开发环境，并尝试将示例项目烧录至您的开发板。

相关文档

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

ESP-WROVER-KIT V3 入门指南

[English]

本指南介绍了如何开始使用 ESP-WROVER-KIT V3 开发板及其功能和相关配置。有关 ESP-WROVER-KIT 其他版本的介绍，请见：*ESP32 H/W 硬件参考*。

准备工作

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

您可以跳过介绍部分，直接前往*应用程序开发* 章节。

概述

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

ESP-WROVER-KIT 可选贴以下 ESP32 模组：

- *ESP32-WROOM-32 模组*
- *ESP32-WROVER*

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

为了便于使用，板上模组的绝大部分管脚均已引出至开发板的引脚。

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

功能概述

ESP-WROVER-KIT 开发板的主要组件和连接方式如下图所示。

功能说明

ESP-WROVER-KIT 开发板的主要组件、接口及控制方式见下。

下表从图片右上角开始，以顺时针顺序介绍了图 1 中的主要组件，然后以同样的顺序介绍图 2 中的主要组件。

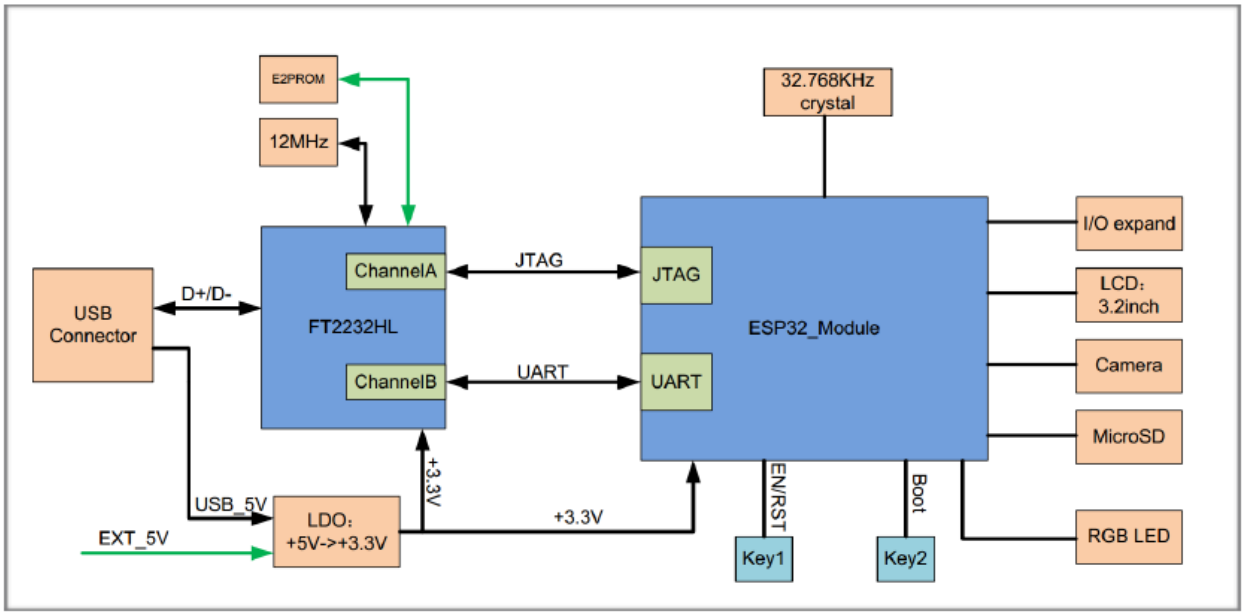


图 5: ESP-WROVER-KIT 功能框图

主要组件	基本介绍
32.768 kHz	外接 32.768 kHz 晶振，可提供 Deep-sleep 下使用的低功耗时钟。
0 欧电阻	ESP-WROVER-KIT 开发板设计了一个 0 欧电阻，可在测量 ESP32 系列模组在不同功耗模式下的电流时，直接移除或替换为分流器。
ESP32 模组	可选贴 ESP32-WROOM-32 或 ESP32-WROVER。ESP32-WROVER 模组完整集成了 ESP32-WROOM-32 的所有功能，且内置 32-Mbit PSRAM，可提供灵活的额外存储空间和数据处理能力。
FT2232	FT2232 多协议 USB 转串口桥接器。开发人员可通过 USB 接口对 FT2232 芯片进行控制和编程，与 ESP32 建立连接。FT2232 芯片可在通道 A 提供 USB-to-JTAG 接口功能，并在通道 B 提供 USB-to-Serial 接口功能，便利开发人员的应用开发与调试。见 ESP-WROVER-KIT V3 原理图。
UART	串口。FT2232HL 和 ESP32 的串行 TX/RX 信号已引出至 JP11 的两端。默认情况下，这两路信号由跳线帽连接。如果要跳过 FT2232 使用 ESP32 模组串口，则可移除相关跳线帽，将模组连接至其他外部串口设备。
SPI	默认情况下，ESP32 使用 SPI 接口访问内置 flash 和 PSRAM。使用这些引脚连接 ESP32 和其他 SPI 设备。这种情况下，需增加额外的片选 (CS) 信号。注意，选贴 ESP32-WROVER 模组时，该接口的工作电压为 1.8 V；选贴 ESP32-WROOM-32 时，该接口的工作电压为 3.3V。
CTS/RTS	串口流控信号。管脚默认不连接至电路。为了使能该功能，必须用跳线帽短路掉 JP14 的相应管脚。
JTAG	JTAG 接口。FT2232HL 和 ESP32 的 JTAG 信号已引出至 JP8 的两端。默认情况下，这两路信号不连接。如需使能 JTAG，请按照设置选项 的介绍，连接跳线帽。
EN	复位按键。
Boot	下载按键。按下 **Boot** 键并保持，同时按一下 **EN** 键（此时不要松开 **Boot** 键）进入“固件下载”模式，通过串口下载固件。
USB	USB 接口。可用作开发板的供电电源，或连接 PC 和开发板的通信接口。
电源开关	拨向 **USB** 按键一侧，开发板上电；拨离 **USB** 按键一侧，开发板掉电。

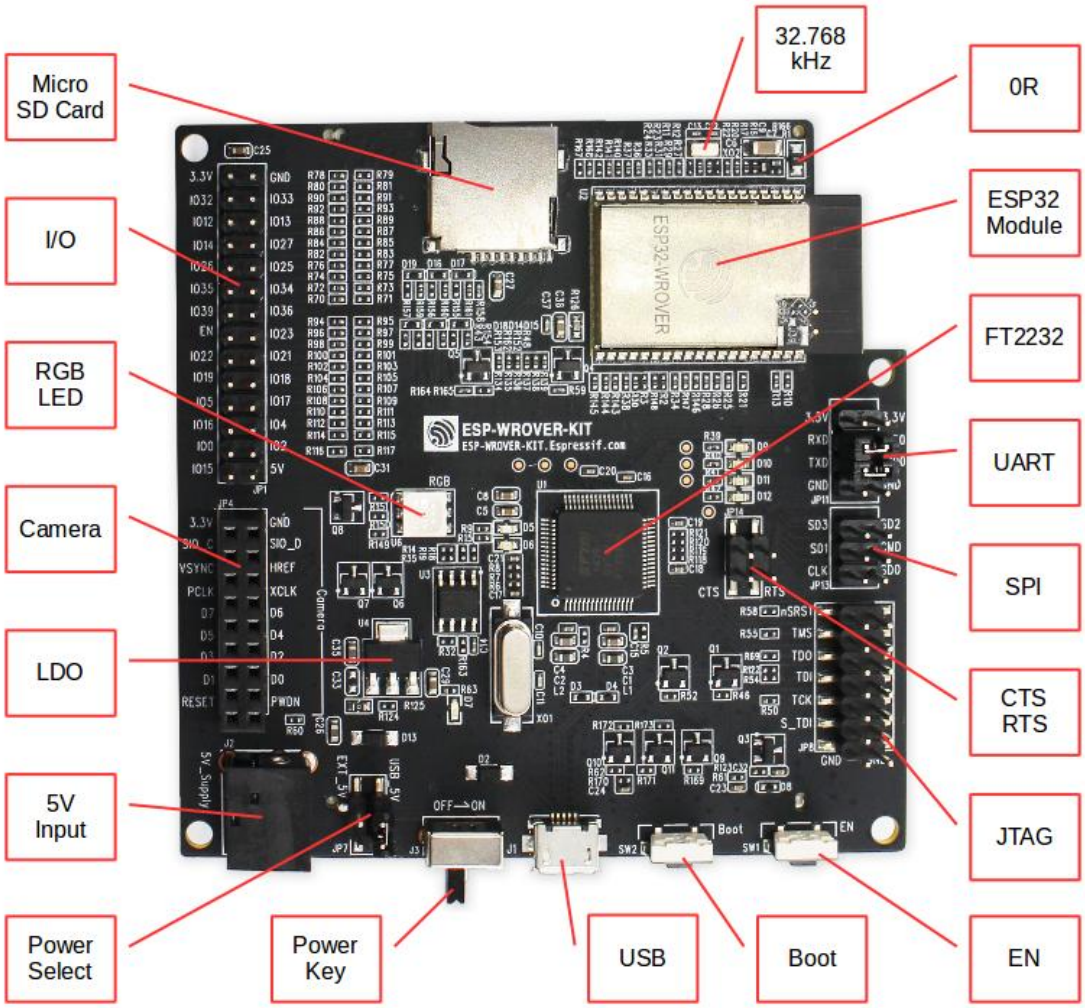


图 6: ESP-WROVER-KIT 开发板布局 – 俯视图

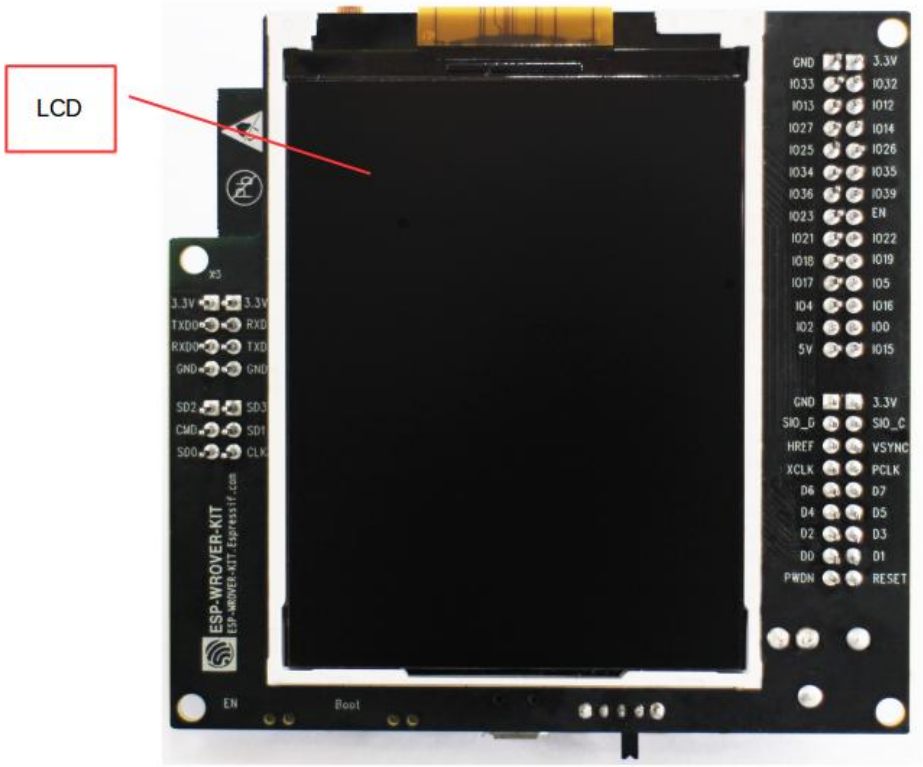
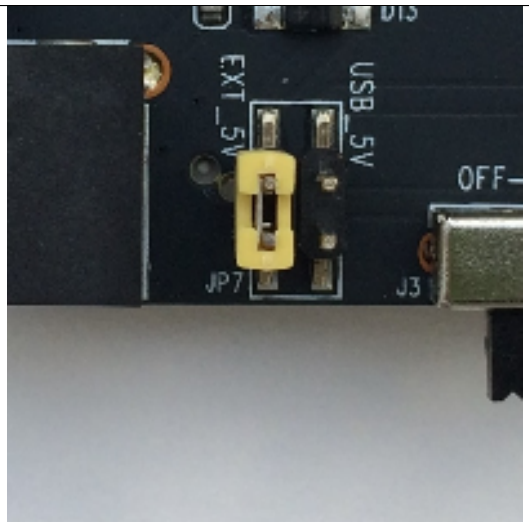
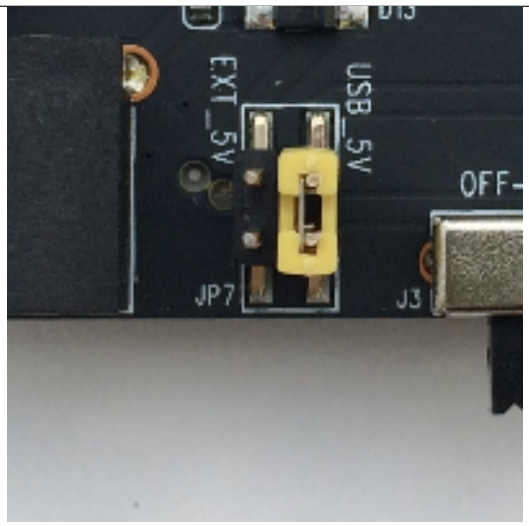
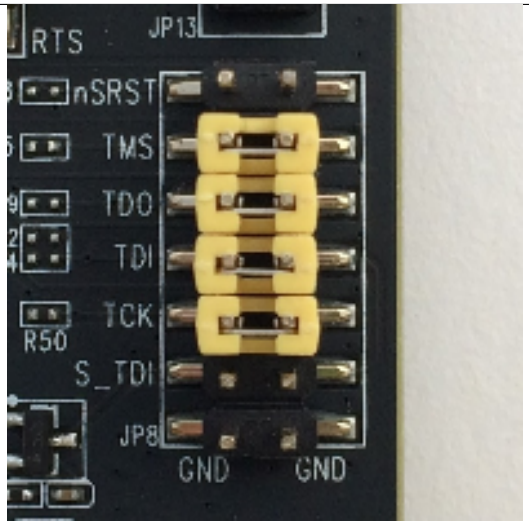
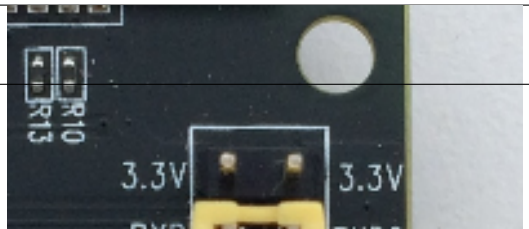


图 7: ESP-WROVER-KIT 开发板布局 – 仰视图

设置选项

用户可通过 5 组排针，设置开发板功能，其中常见功能见下表:

排针	跳线设置	功能描述
JP7		使用外部电源为 ESP-WROVER-KIT 开发板供电
JP7		使用 USB 端口为 ESP-WROVER-KIT 开发板供电
JP8		使能 JTAG 功能
		

ESP32 管脚分配

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

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

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

更多外设共享管脚的介绍，请见下一章节中的表格。

主要 I/O 连接器 / JP1

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

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

说明：

- NC/XTAL - *32.768 kHz Oscillator*
- JTAG - *JTAG / JP8*
- Boot - Boot 按键 / SW2
- 摄像头 - *摄像头 / JP4*

- LED - *RGB LED*
- MicroSD - *MicroSD Card / J4*
- LCD - *LCD / U5*
- PSRAM - 仅适用于选贴 ESP32-WROVER 的情况。

32.768 kHz 晶振

.	ESP32 管脚
1.	GPIO32
2.	GPIO33

注解： 默认情况下，管脚 GPIO32 和 GPIO33 已连接至晶振。因此，为了保证信号的完整性，这两个管脚并未连接至 JP1 I/O 连接器。用户可通过将 R11/R23 处的 0 欧电阻移至 R12/R24 处，以将 GP1O32 和 GPIO33 的连接从晶振移至 JP1。

SPI Flash / JP13

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

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

JTAG / JP8

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

摄像头 / JP4

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

- D0 到 D7 为摄像头的数据总线

RGB LED

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

MicroSD 卡

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

LCD / U5

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

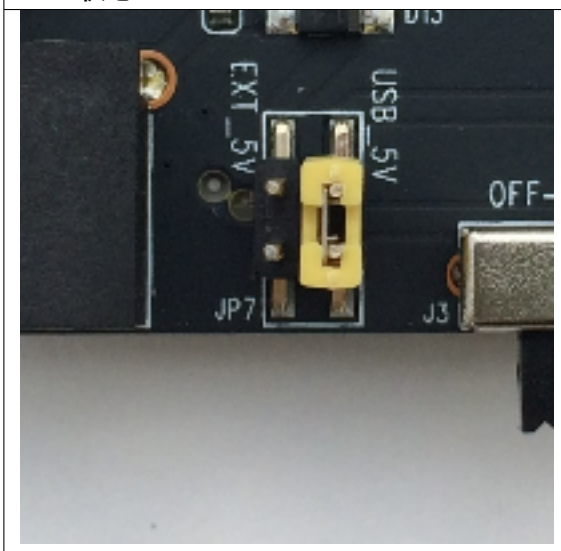
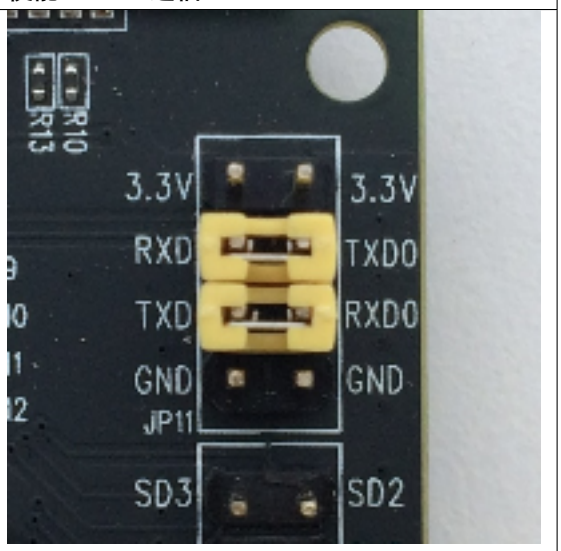
应用程序开发

ESP-WROVER-KIT 上电前，请首先确认开发板完好无损。

初始设置

请严格按照下图所示连接跳线帽，注意不要额外连接其他跳线帽。

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

USB 供电	使能 UART 通信
	

注意不要连接其他跳线帽。

打开 **电源开关**，**5V Power On LED** 应点亮。

正式开始开发

现在，请前往*ESP32 H/W 硬件参考* 中的*详细安装步骤* 章节，查看如何设置开发环境，并尝试将示例项目烧录至您的开发板。

相关文档

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

ESP-WROVER-KIT V2 入门指南

[English]

本指南介绍了如何开始使用 ESP-WROVER-KIT V2 开发板及其功能和相关配置。有关 ESP-WROVER-KIT 其他版本的介绍，请见：*ESP32 H/W 硬件参考*。

准备工作

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

您可以跳过介绍部分，直接前往应用程序开发 章节。

概述

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

ESP-WROVER-KIT 可选贴以下 ESP32 模组：

- *ESP32-WROOM-32* 模组
- *ESP32-WROVER*

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

为了便于使用，板上模组的绝大部分管脚均已引出至开发板的引脚。

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

功能概述

ESP-WROVER-KIT 开发板的主要组件和连接方式如下图所示。

功能说明

ESP-WROVER-KIT 开发板的主要组件、接口及控制方式见下。

下表从图片右上角开始，以顺时针顺序介绍了图 1 中的主要组件，然后以同样的顺序介绍了图 2 中的主要组件。

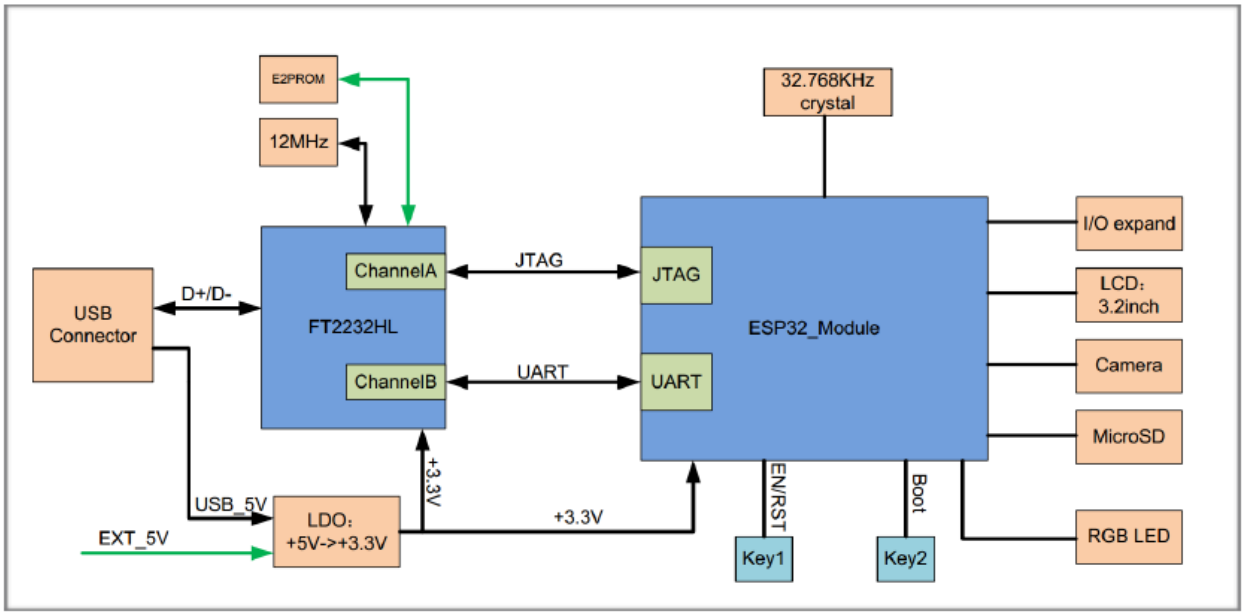


图 8: ESP-WROVER-KIT 功能框图

主要组件	基本介绍
32.768 kHz	外接 32.768 kHz 晶振，可提供 Deep-sleep 下使用的低功耗时钟。
ESP32 模组	可选贴 ESP32-WROOM-32 或 ESP32-WROVER。ESP32-WROVER 模组完整集成了 ESP32-WROOM-32 的所有功能，且内置 32-Mbit PSRAM，可提供灵活的额外存储空间和数据处理能力。
CTS/RTS	串口流控信号。管脚默认不连接至电路。为了使能该功能，必须用跳线帽短路掉 JP14 的相应管脚。
UART	串口。FT2232HL 和 ESP32 的串行 TX/RX 信号已引出至 JP11 的两端。默认情况下，这两路信号由跳线帽连接。如果要跳过 FT2232 使用 ESP32 模组串口，则可移除相关跳线帽，将模组连接至其他外部串口设备。
SPI	默认情况下，ESP32 使用 SPI 接口访问内置 flash 和 PSRAM。使用这些引脚连接 ESP32 和其他 SPI 设备。这种情况下，需增加额外的片选 (CS) 信号。注意，选贴 ESP32-WROVER 模组时，该接口的工作电压为 1.8 V；选贴 ESP32-WROOM-32 时，该接口的工作电压为 3.3V。
JTAG	JTAG 接口。FT2232HL 和 ESP32 的 JTAG 信号已引出至 JP8 的两端。默认情况下，这两路信号不连接。如需使能 JTAG，请按照设置选项 的介绍，连接跳线帽。
FT2232	FT2232 多协议 USB 转串口桥接器。开发人员可通过 USB 接口对 FT2232 芯片进行控制和编程，与 ESP32 建立连接。FT2232 具有 USB-to-UART 和 USB-to-JTAG 功能。
EN	复位按键。
Boot	下载按键。按下 **Boot** 键并保持，同时按一下 **EN** 键（此时不要松开 **Boot** 键）进入“固件下载”模式，通过串口下载固件。
USB	USB 接口。可用作开发板的供电电源，或连接 PC 和开发板的通信接口。
电源选择开关	ESP-WROVER-KIT 开发板可通过 USB 端口或 5V 输入接口供电。用户可使用跳线帽在两种供电模式中进行选择。更多详细信息，请见章节设置选项 中有关 JP7 连接器的描述。
电源开关	拨向 **USB** 按键一侧，开发板上电；拨离 **USB** 按键一侧，开发板掉电。

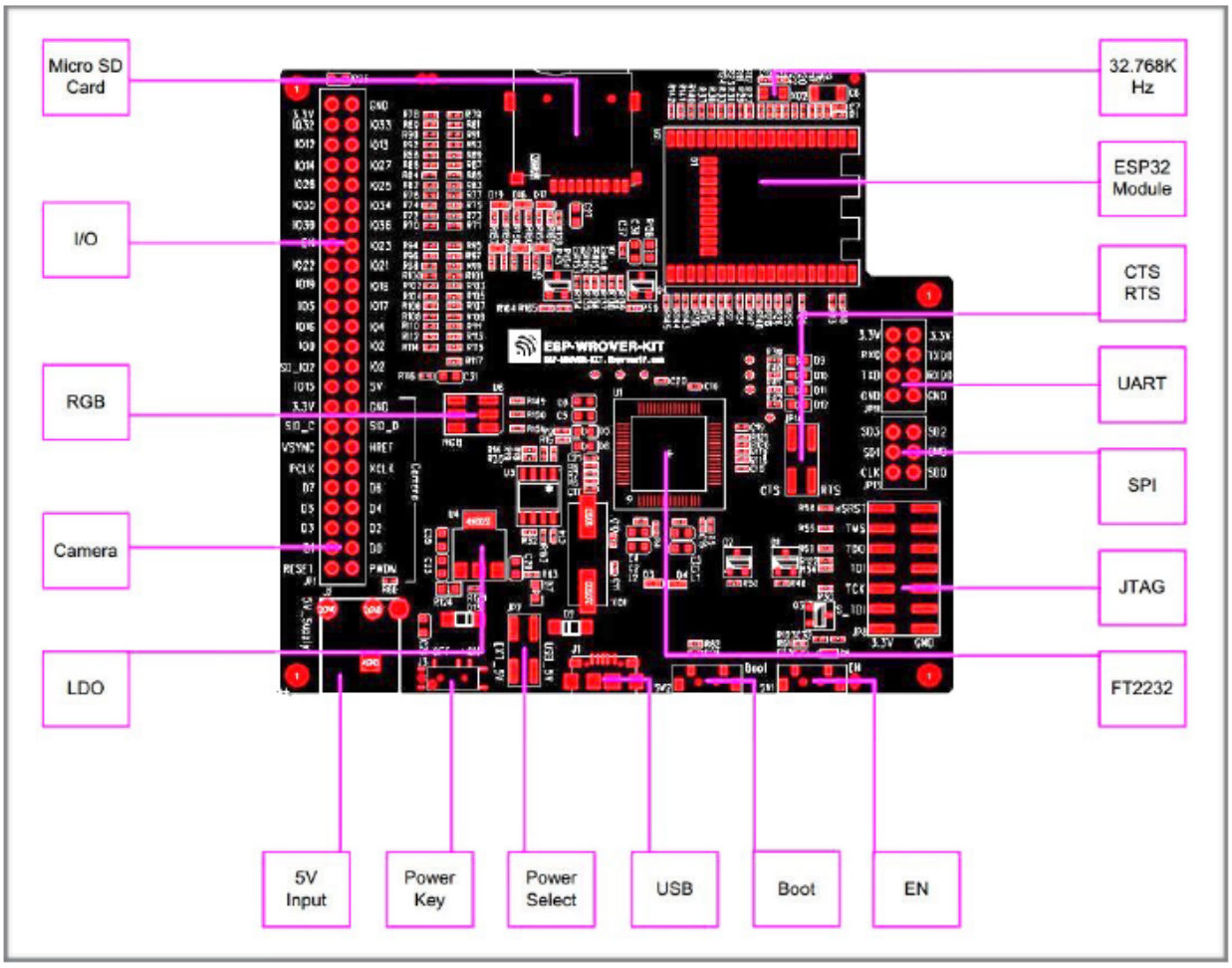


图 9: ESP-WROVER-KIT 开发板布局 – 俯视图

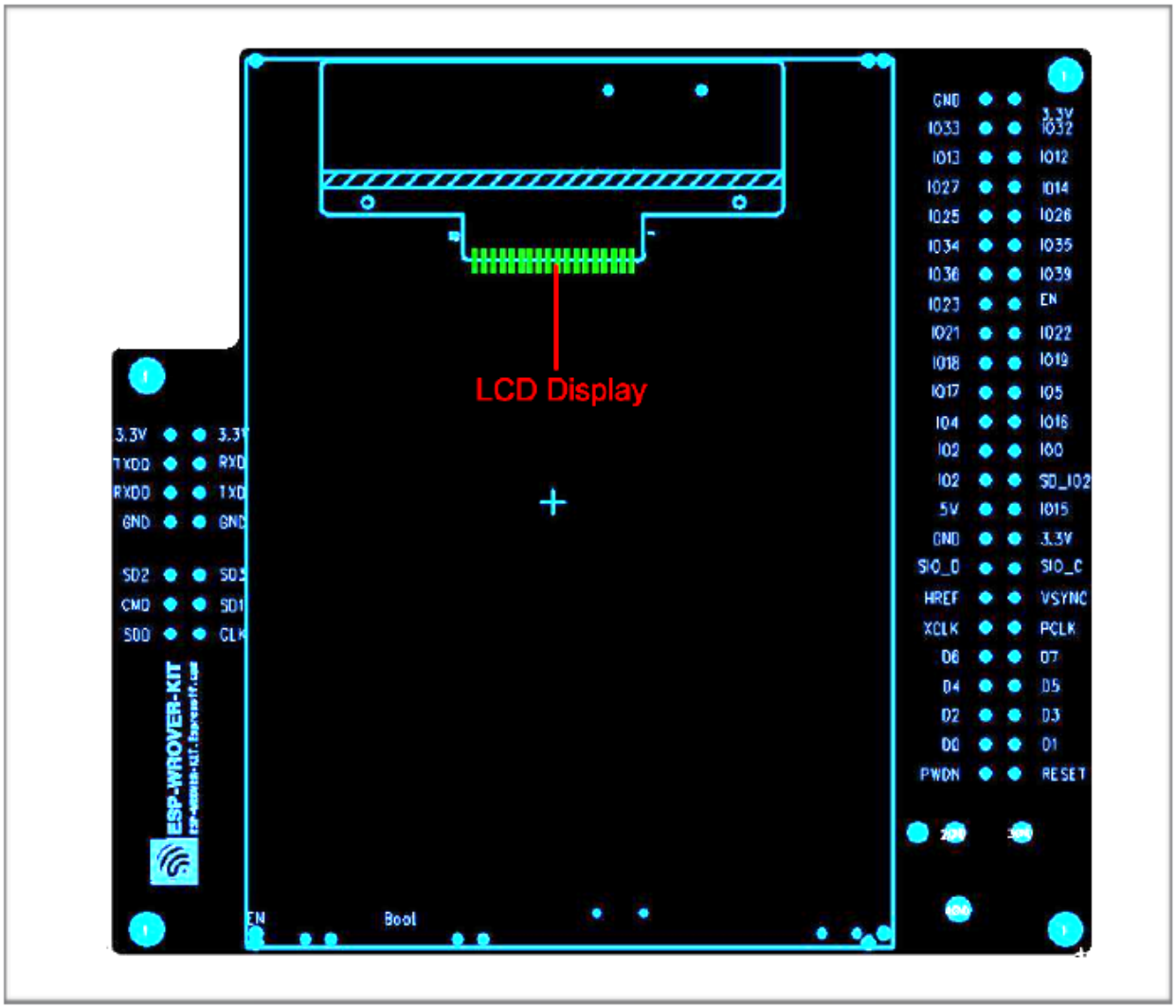
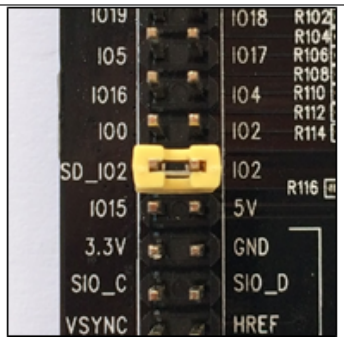
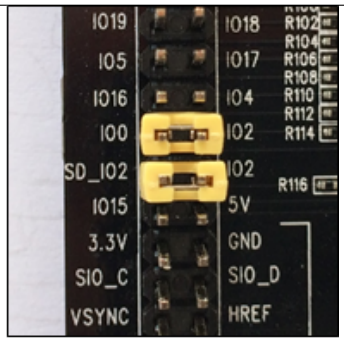
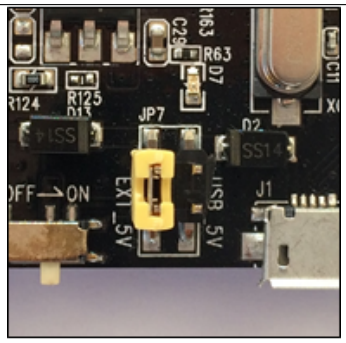
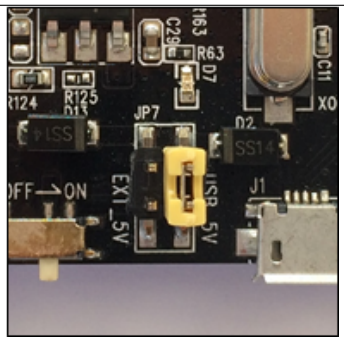
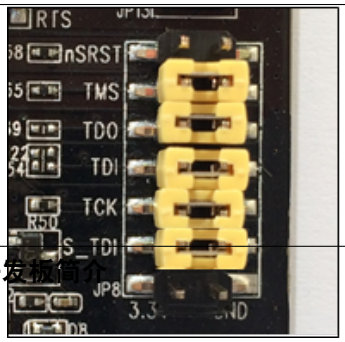


图 10: ESP-WROVER-KIT 开发板布局 – 仰视图

设置选项

用户可通过 5 组排针，设置开发板功能，其中常见功能见下表:

排针	跳线设置	功能描述
JP1		使能 MicroSD Card 功能的上拉电阻
JP1		确保开发板处于下载模式时，GPIO2 时钟处于低位（将 JP1 连接至 GPIO0）
JP7		使用外部电源为 ESP-WROVER-KIT 开发板供电
JP7		使用 USB 端口为 ESP-WROVER-KIT 开发板供电
JP8		使能 JTAG 功能

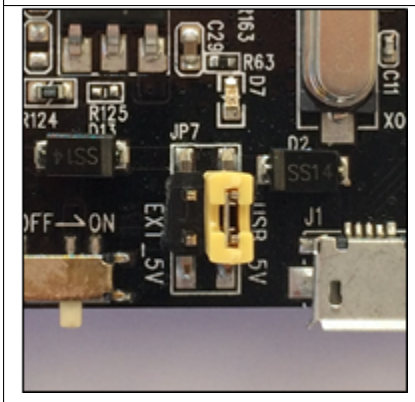
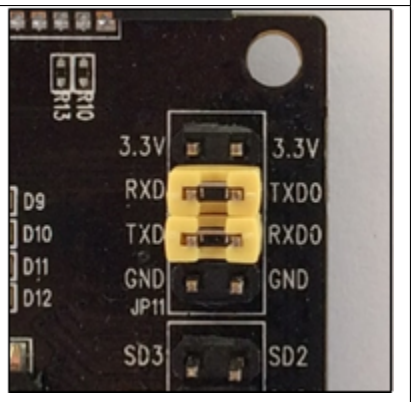
应用程序开发

ESP-WROVER-KIT 上电前，请首先确认开发板完好无损。

初始设置

请严格按照下图所示连接跳线帽，注意不要额外连接其他跳线帽。

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

USB 供电	使能 UART 通信
	

注意不要连接其他跳线帽。

打开 **电源开关**，**5V Power On LED** 应点亮。

正式开始开发

现在，请前往*ESP32 H/W 硬件参考* 中的*详细安装步骤* 章节，查看如何设置开发环境，并尝试将示例项目烧录至您的开发板。

相关文档

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

1.3.3 ESP32-PICO-KIT V4/V4.1 入门指南

[English]

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

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

准备工作

- *ESP32-PICO-KIT 迷你开发板*
- USB 2.0 线（A 型转 Micro-B 型）
- PC（Windows、Linux 或 Mac OS）

您可以跳过介绍部分，直接前往应用程序开发 章节。

概述

ESP32-PICO-KIT 是一款来自 乐鑫 的迷你开发板，其核心是具有完整 Wi-Fi 和蓝牙功能的 ESP32 系列 SiP 模组 ESP32-PICO-D4。与其他 ESP32 系列模组相比，ESP32-PICO-D4 模组已完整集成以下外围器件：

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

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

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

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

注解:

1. 每排未引出至排针的 3 个管脚已连接至 ESP32-PICO-D4 SiP 模组的内置 flash 模块。更多信息，请见相关文档 中的模组技术规格书。
2. ESP32-PICO-D4 开发板默认采用排针。

功能概述

ESP32-PICO-KIT 开发板的主要组件和连接方式见下。

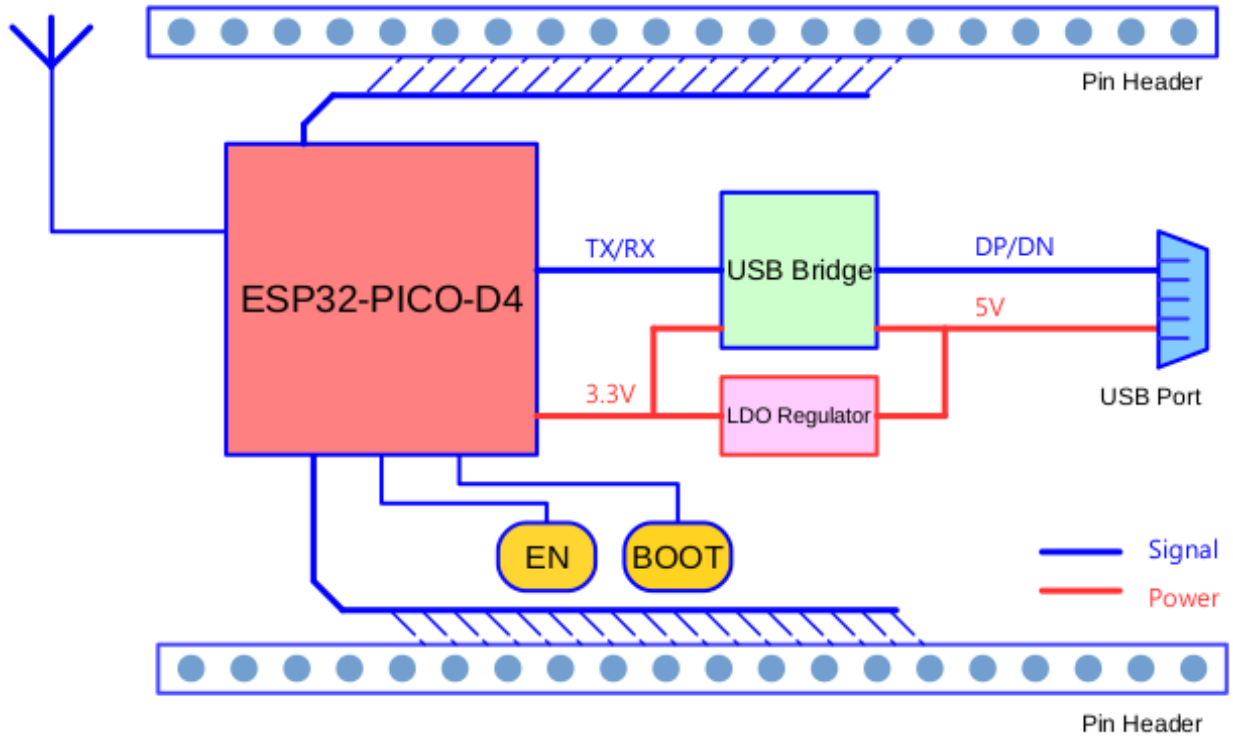


图 11: ESP32-PICO-KIT 框图

功能说明

ESP32-PICO-KIT 开发板的主要组件、接口及控制方式见下。

ESP32-PICO-KIT 开发板的主要组件描述见下表（从左上角起顺时针顺序）。

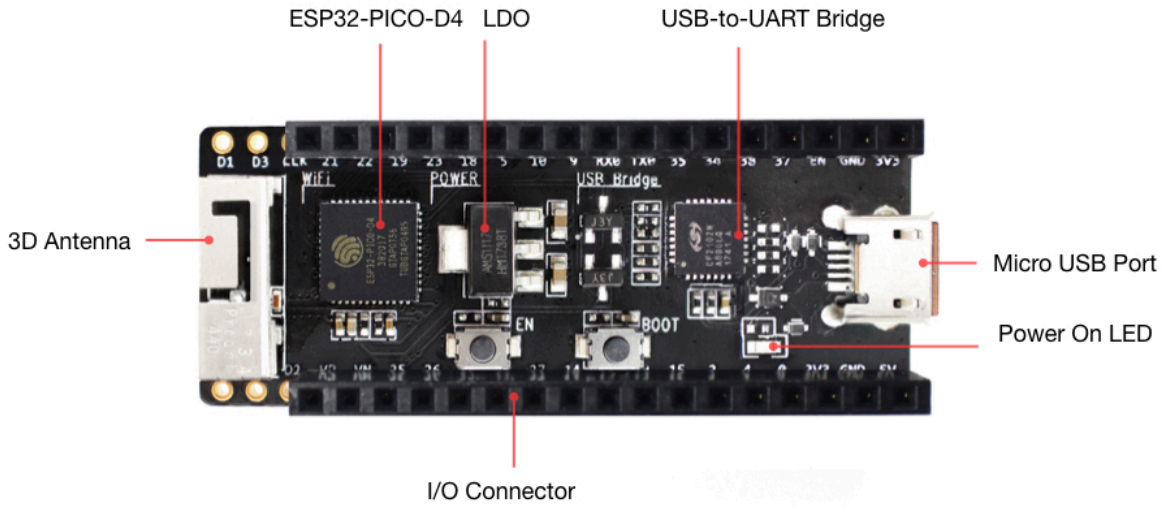


图 12: ESP32-PICO-KIT 开发板布局

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

电源选项

开发板可任一选用以下三种供电方式：

- Micro USB 供电（默认）

- 5V / GND 管脚供电
- 3V3 / GND 管脚供电

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

管脚说明

下表介绍了开发板 I/O 管脚的 **名称**和 **功能**，具体布局请见相关文档 中的原理图。请参考*ESP32-PICO-KIT 开发板布局*。

Header J2

编号	名称	类型	功能
1	FLASH_SD1 (FSD1)	I/O	GPIO8, SD_DATA1, SPID, HS1_DATA1 (见说明 1) , U2CTS
2	FLASH_SD3 (FSD3)	I/O	GPIO7, SD_DATA0, SPIQ, HS1_DATA0 (见说明 1) , U2RTS
3	FLASH_CLK (FCLK)	I/O	GPIO6, SD_CLK, SPICLK, HS1_CLK (见说明 1) , U1CTS
4	IO21	I/O	GPIO21, VSPIHD, EMAC_TX_EN
5	IO22	I/O	GPIO22, VSPIWP, U0RTS, EMAC_TXD1
6	IO19	I/O	GPIO19, VSPIQ, U0CTS, EMAC_TXD0
7	IO23	I/O	GPIO23, VSPID, HS1_STROBE
8	IO18	I/O	GPIO18, VSPICLK, HS1_DATA7
9	IO5	I/O	GPIO5, VSPICS0, HS1_DATA6,

Header J3

No.	Name	Type	Function
1	FLASH_CS (FCS)	I/O	GPIO16, HS1_DATA4 (见说明 1) , U2RXD, EMAC_CLK_OUT
2	FLASH_SD0 (FSD0)	I/O	GPIO17, HS1_DATA5 (见说明 1) , U2TXD, EMAC_CLK_OUT_180
3	FLASH_SD2 (FSD2)	I/O	GPIO11, SD_CMD, SPICS0, HS1_CMD (见说明 1) , U1RTS
4	SENSOR_VP (FSVP)	I	GPIO36, ADC1_CH0, RTC_GPIO0
5	SENSOR_VN (FSVN)	I	GPIO39, ADC1_CH3, RTC_GPIO3
6	IO25	I/O	GPIO25, DAC_1, ADC2_CH8, RTC_GPIO6, EMAC_RXD0
7	IO26	I/O	GPIO26, DAC_2, ADC2_CH9, RTC_GPIO7, EMAC_RXD1
8	IO32	I/O	32K_XP (见说明 2a) , ADC1_CH4, TOUCH9,

有关上表的说明：

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

应用程序开发

ESP32-PICO-KIT 上电前，请首先确认开发板完好无损。

现在，请前往*ESP32 H/W 硬件参考* 中的*详细安装步骤* 章节，查看如何设置开发环境，并尝试将示例项目烧录至您的开发板。

开发板尺寸

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

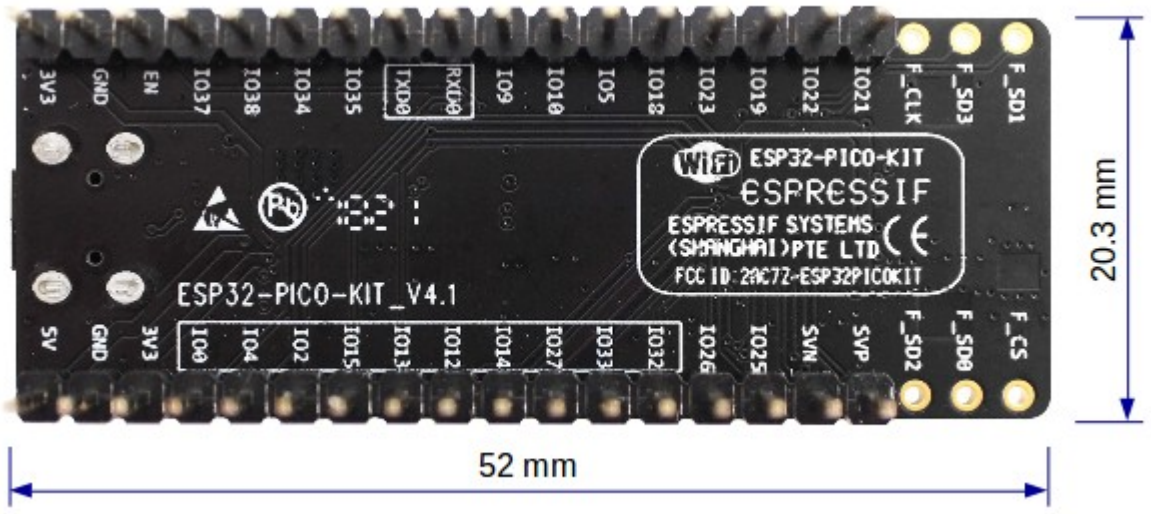


图 13: ESP32-PICO-KIT 尺寸图 – 背面

有关开发板的物理结构细节，请见下方参考设计。

相关文档

- ESP32-PICO-KIT V4 原理图 (PDF)
- ESP32-PICO-KIT V4.1 原理图 (PDF)

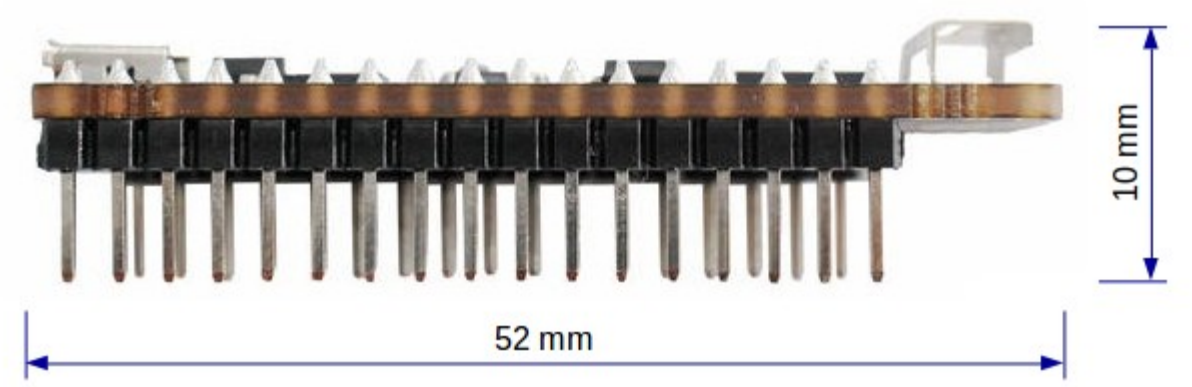


图 14: ESP32-PICO-KIT 尺寸图 – 侧面

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

ESP32-PICO-KIT V3 入门指南

[English]

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

准备工作

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

您可以跳过介绍部分，直接前往应用程序开发 章节。

概述

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

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

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

功能说明

ESP32-PICO-KIT V3 开发板的主要组件、接口及控制方式见下。

ESP32-PICO-KIT 开发板的主要组件描述见下表。

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

应用程序开发

ESP32-PICO-KIT V3 上电前，请首先确认开发板完好无损。

现在，请前往*ESP32 H/W 硬件参考* 中的*详细安装步骤* 章节，查看如何设置开发环境，并尝试将示例项目烧录至您的开发板。

相关文档

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

1.3.4 ESP32-Ethernet-Kit V1.1 入门指南

[English]

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

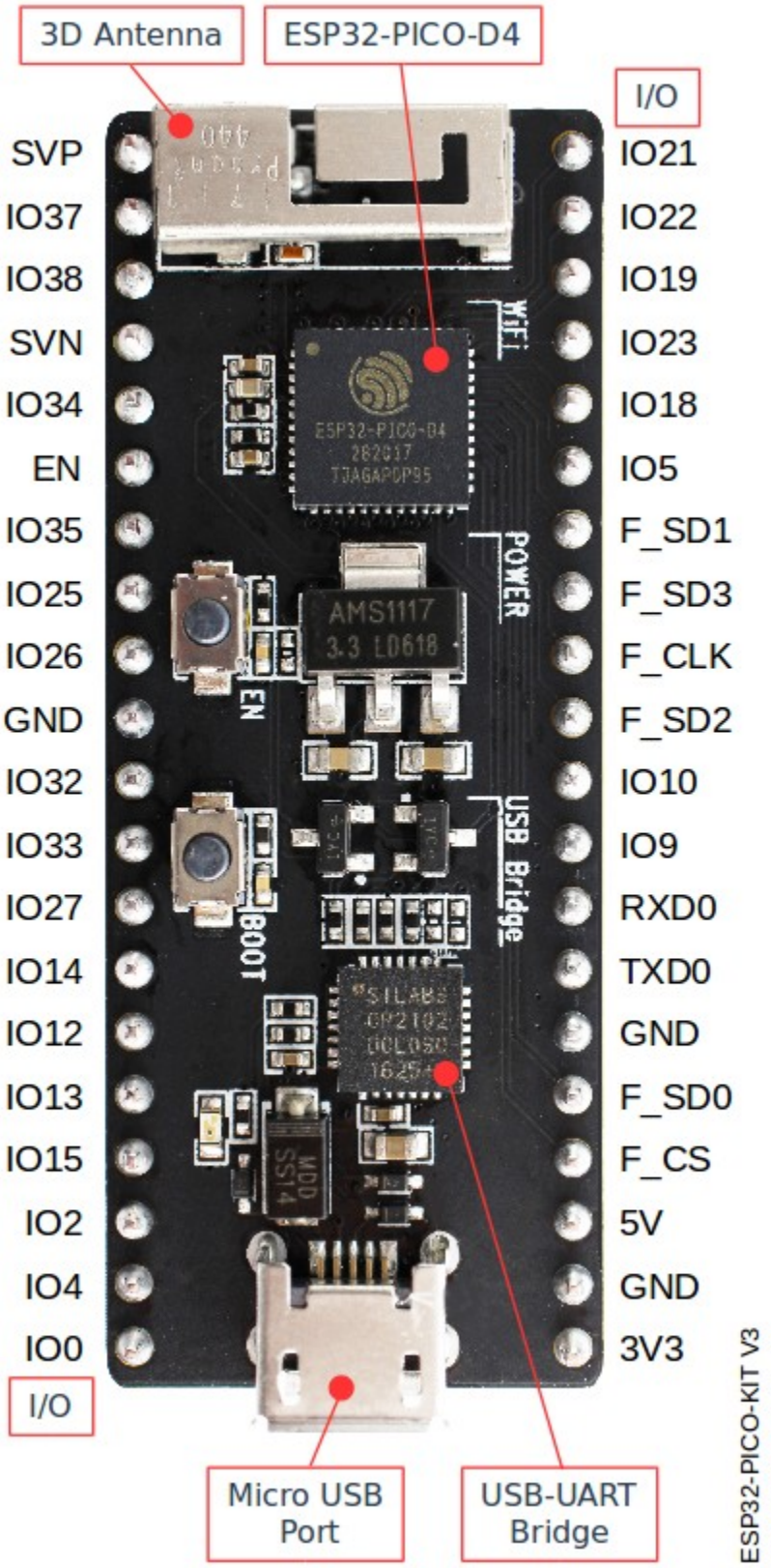


图 15: ESP32-PICO-KIT V3 开发板布局

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

准备工作

- *ESP32-Ethernet-Kit V1.1 开发板*
- USB 数据线（A 转 Micro-B）
- PC（Windows、Linux 或 Mac OS）

您可以跳过介绍部分，直接前往*应用程序开发* 章节。

概述

ESP32-Ethernet-Kit 是一款来自 乐鑫 的开发板。

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

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

功能概述

ESP32-Ethernet-Kit 开发板的主要组件和连接方式如下。

功能说明

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

以太网母板（A 板）

下表将从图片右上角开始，以顺时针顺序介绍图中的主要组件。

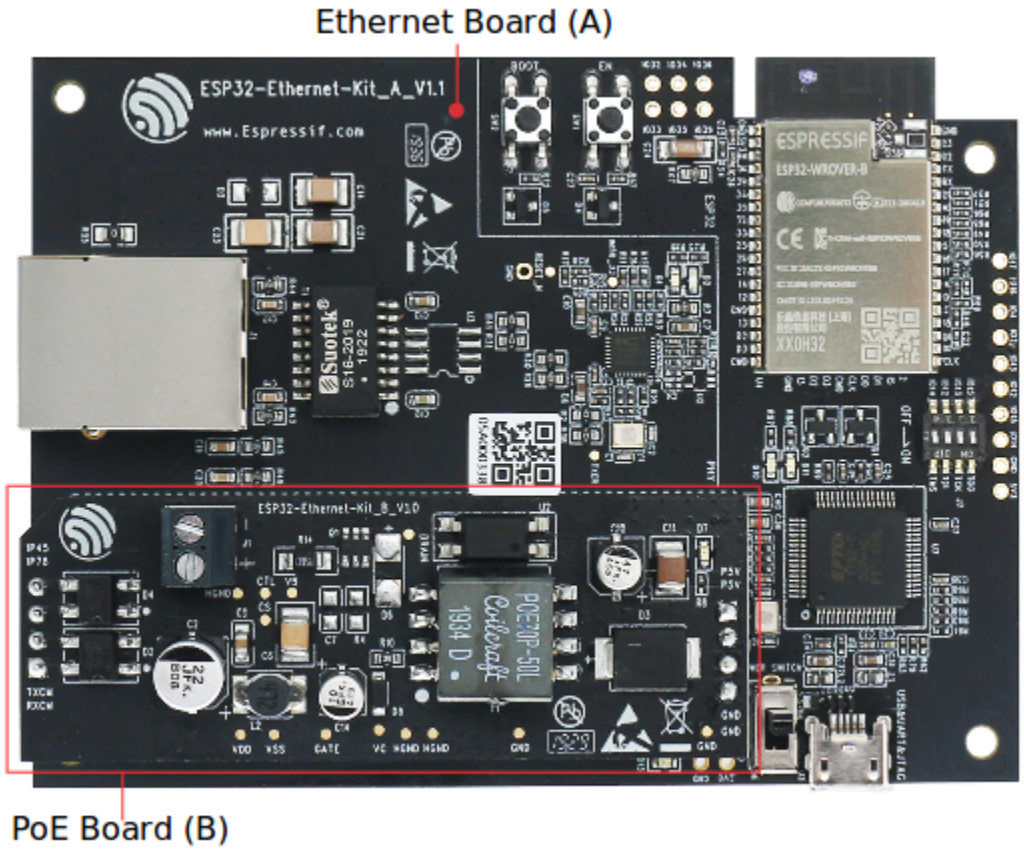


图 16: ESP32-Ethernet-Kit V1.1

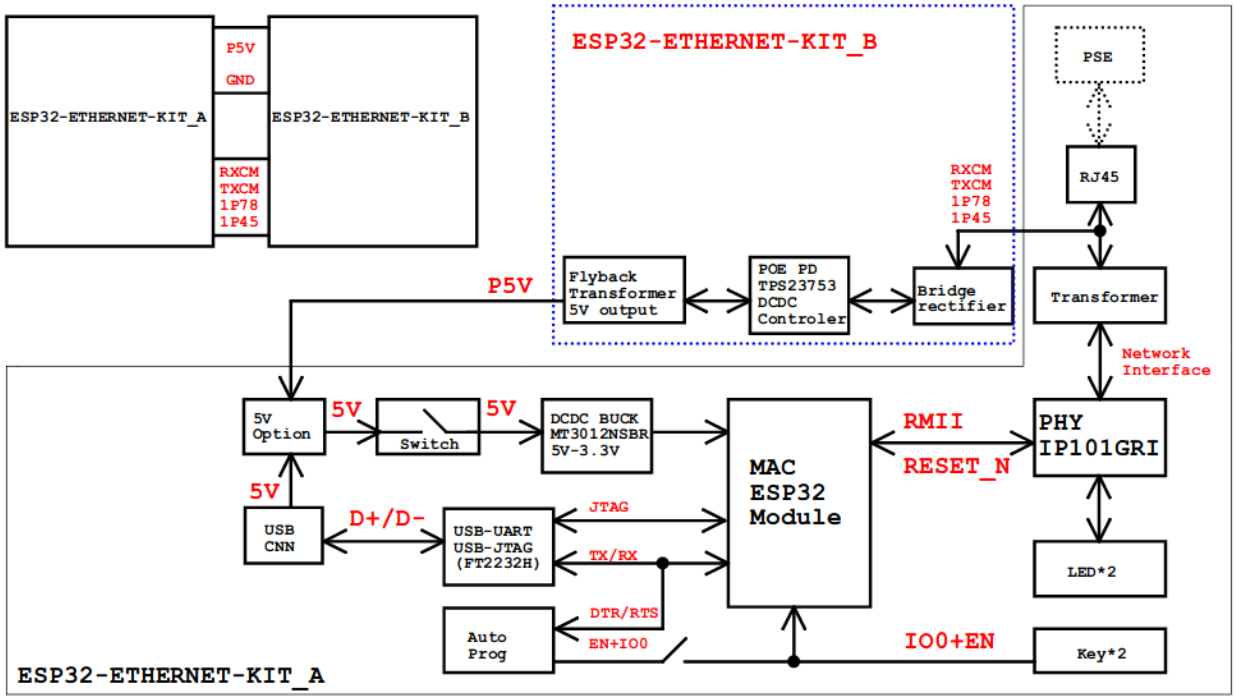


图 17: ESP32-Ethernet-Kit 功能框图（点击放大）

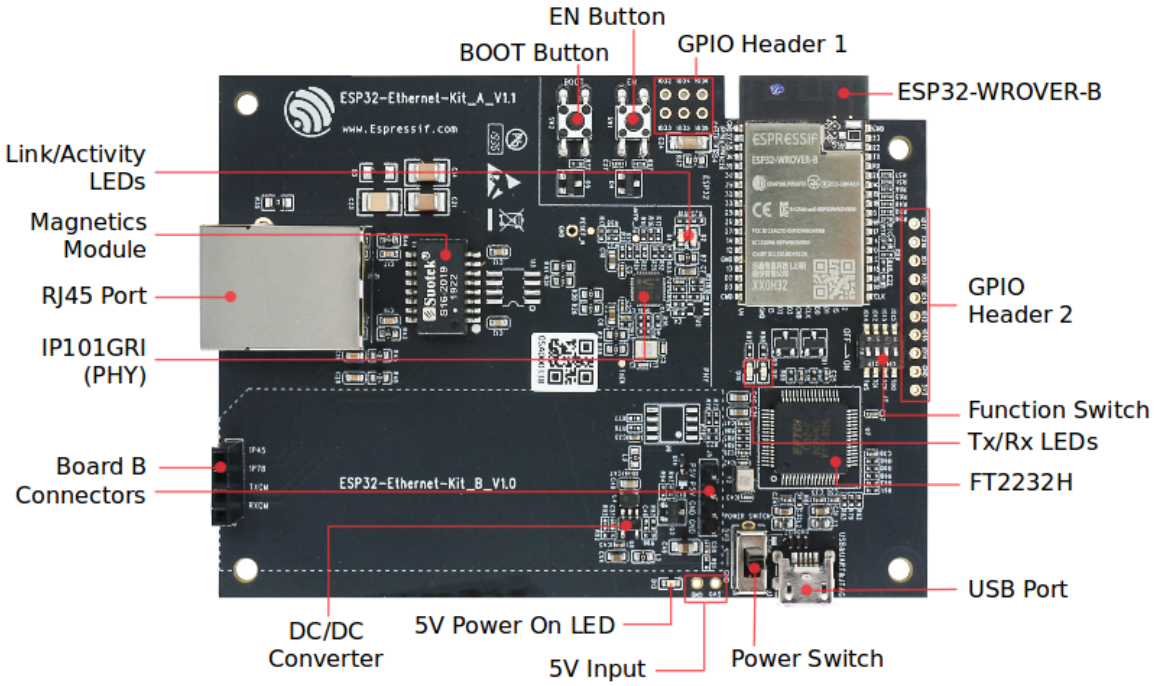


图 18: ESP32-Ethernet-Kit - 以太网母板（A 板）布局（点击放大）（请更新图片）

表 1: 表格 1 组件介绍

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

PoE 子板（B 板）

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

PoE 子板（B 板）具有以下特性：

- 支持 IEEE 802.3at 标准
- 电源输出：5 V，1.4 A

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

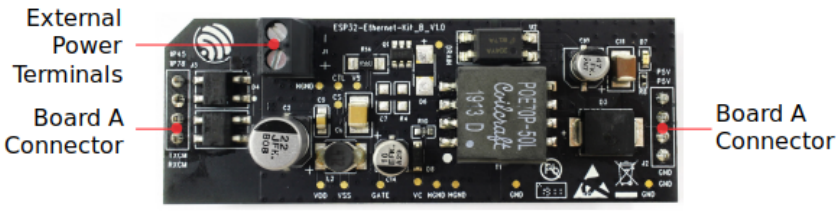


图 19: ESP32-Ethernet-Kit - PoE 子板（B 板）布局（点击放大）

表 2: 表格 2 PoE 子板（B 板）

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

设置选项

本节介绍用于 ESP32-Ethernet-Kit 开发板的硬件配置选项。

功能选择开关

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

拨码开关	GPIO 管脚
1	GPIO13
2	GPIO12
3	GPIO15
4	GPIO14

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

RMII 时钟源选择

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

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

PHY 侧提供 RMII 时钟

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

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

ESP32 APLL 内部提供的 RMII 时钟

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

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

GPIO 分配

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

IP101GRI (PHY) 接口

下表显示了 ESP32 (MAC) 与 IP101GRI (PHY) 的管脚对应关系。ESP32-Ethernet-Kit 的实现默认设置为简化媒体独立接口。

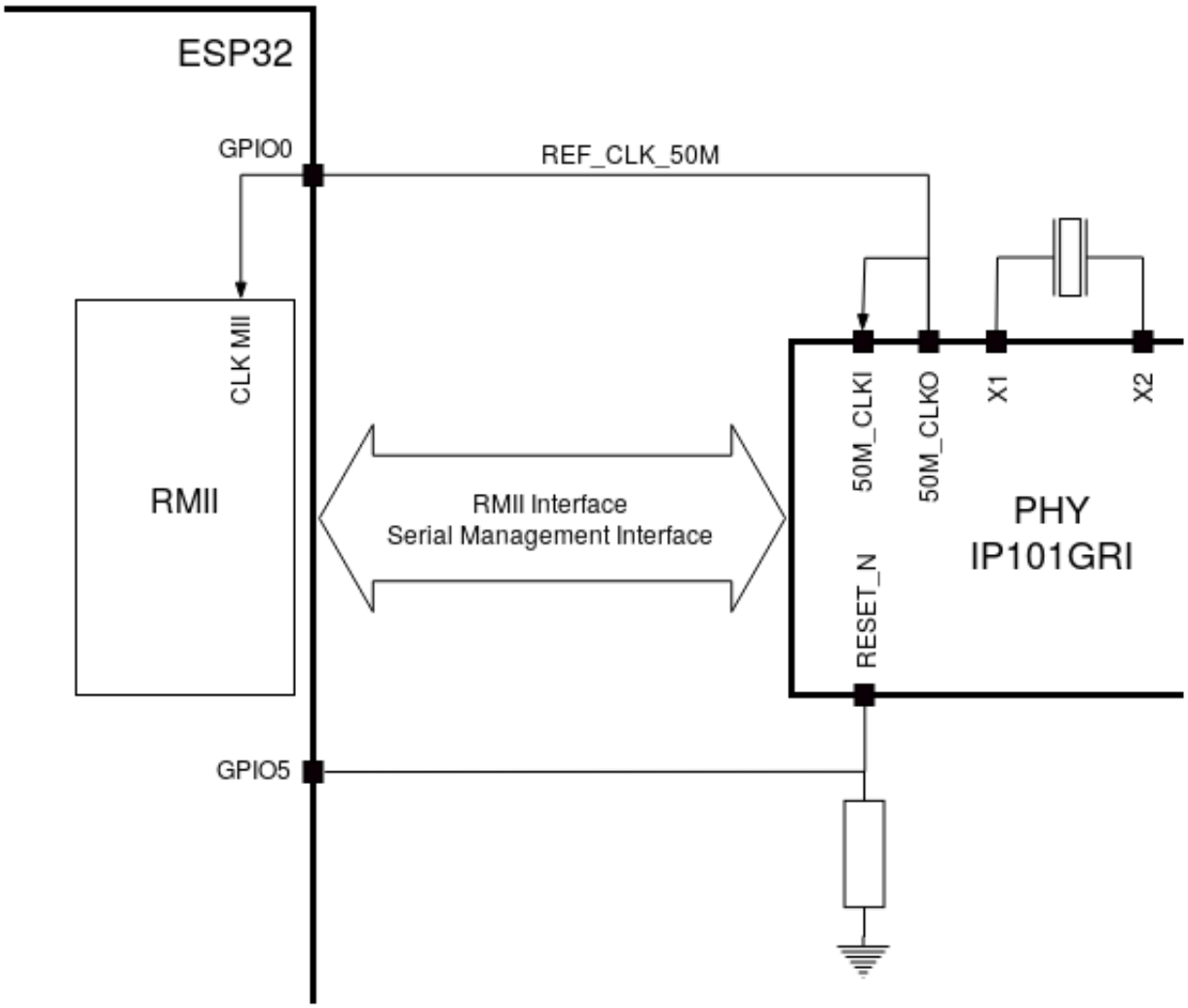


图 20: PHY 侧提供 RMII 时钟

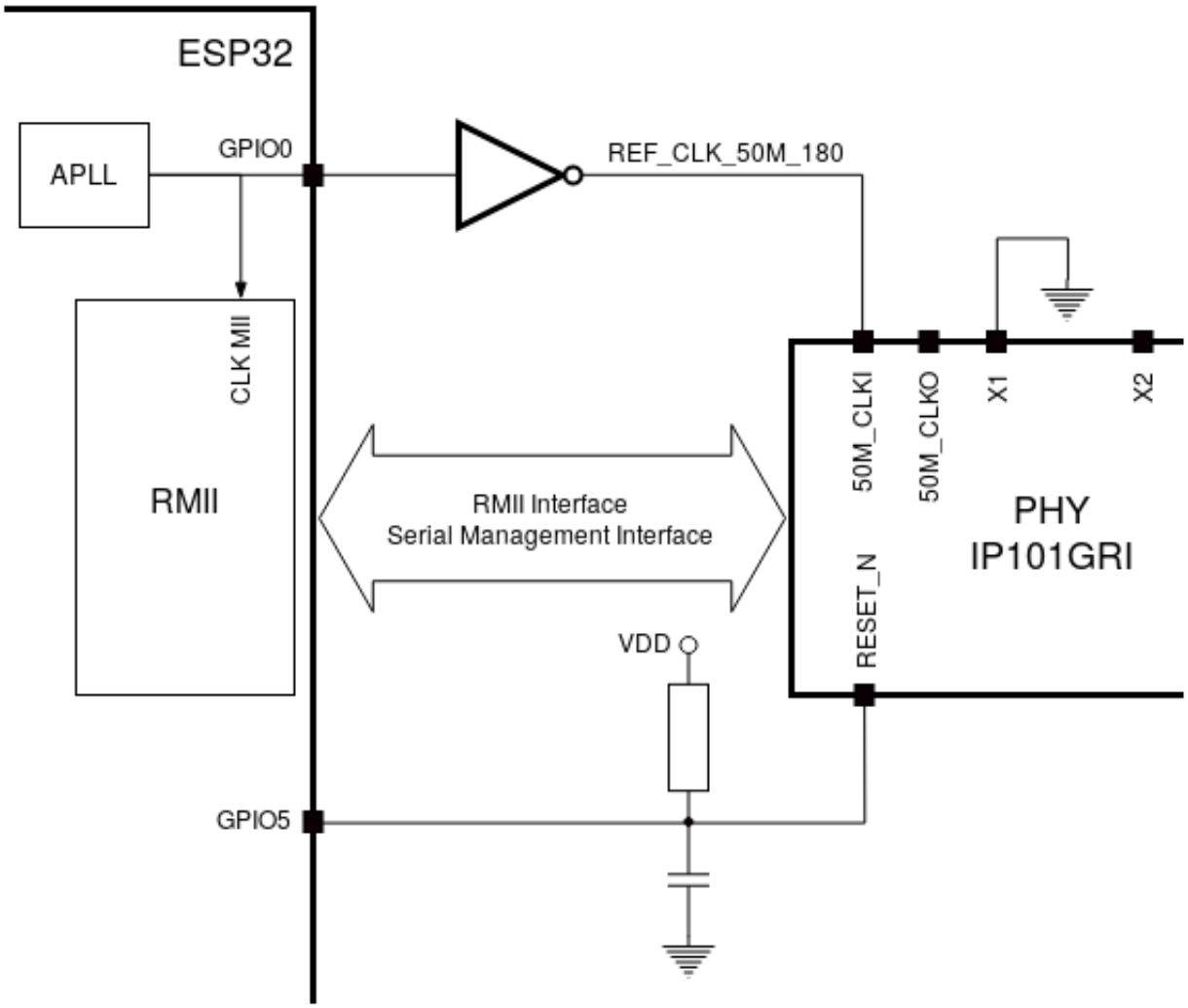


图 21: ESP32 APLL 内部提供的 RMII 时钟

.	ESP32 管脚 (MAC)	IP101GRI (PHY)
RMII 接口		
1	GPIO21	TX_EN
2	GPIO19	TXD[0]
3	GPIO22	TXD[1]
4	GPIO25	RXD[0]
5	GPIO26	RXD[1]
6	GPIO27	CRS_DV
7	GPIO0	REF_CLK
串行管理接口		
8	GPIO23	MDC
9	GPIO18	MDIO
PHY 复位		
10	GPIO5	Reset_N

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

GPIO Header 1

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

.	ESP32 管脚
1	GPIO32
2	GPIO33
3	GPIO34
4	GPIO35
5	GPIO36
6	GPIO39

GPIO Header 2

根据“说明”描述的不同情形，本连接器包含可用做他用的 GPIO。

.	ESP32 管脚	说明
1	GPIO17	见下方说明 1
2	GPIO16	见下方说明 1
3	GPIO4	
4	GPIO2	
5	GPIO13	见下方说明 2
6	GPIO12	见下方说明 2
7	GPIO15	见下方说明 2
8	GPIO14	见下方说明 2
9	GND	Ground
10	3V3	3.3 V 电源

注解:

1. ESP32 芯片的 GPIO16 和 GPIO17 管脚没有引出至 ESP32-WROVER-B 模组的管脚，因此无法使用。如需使用 ESP32 的 GP1016 和 GPIO17 管脚，建议更换其他不含 PSRAM 的模组，比如 ESP32-WROOM-32D 或 ESP32-SOLO-1。
2. 具体功能取决与功能选择开关 的设置。

GPIO 管脚分配总结

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

注解:

1. 为防止 ESP32 侧 GPIO0 的上电状态受 PHY 侧时钟输出的影响，PHY 侧 RESET_N 默认为低，以关闭 PHY 侧时钟输出。上电后，您可通过 GPIO5 控制 RESET_N 以打开该时钟输出。参见*PHY 侧提供 RMII 时钟*。对于无法通过 RESET_N 关闭时钟输出的 PHY，PHY 侧建议使用可在外部禁用/使能的有源晶振。与使用 RESET_N 类似，默认情况下晶振模块应禁用，并在上电后由 ESP32 开启。有关参考设计，请参见 ESP32-Ethernet-Kit V1.1 以太网母板（A 板）原理图。

2. ESP32 芯片的 GPIO16 和 GPIO17 管脚没有引出至 ESP32-WROVER-B 模组的管脚，因此无法使用。如需使用 ESP32 的 GP1016 和 GPIO17 管脚，建议更换其他不含 PSRAM 的模组，比如 ESP32-

WROOM-32D 或 ESP32-SOLO-1。

应用程序开发

ESP32-Ethernet-Kit 上电前，请首先确认开发板完好无损。

初始设置

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

正式开始开发

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

请务必在进入下一步前，确保您已完成上述所有步骤。

配置与加载以太网示例

在完成开发环境设置和开发板测试后，您可以配置并烧录 ethernet/ethernet 示例。本示例专门用于测试以太网功能，支持不同 PHY，包括*ESP32-Ethernet-Kit V1.1* 开发板使用的 **IP101GRI** 。

针对 ESP32-Ethernet-Kit V1.0 的主要修改：

- 原 GPIO0 反相后时钟提供给 PHY 方案改为由 PHY 侧外接无源晶振，提供时钟给 GPIO0。原用于控制有源晶振的 OSC_EN 的 IO2 释放，可用作其他用途。
- 为防止 ESP32 侧 GPIO0 的上电状态受到 PHY 侧时钟输出的影响，PHY 侧 RESET_N 默认为低，关闭 PHY 侧时钟输出。而后可通过 GPIO5 控制 RESET_N 打开该时钟输出。
- 移除 FT2232H 芯片的外部 SPI Flash U6。
- 移除流控的测试排针 J4。
- 移除 nTRST JTAG 信号，相应的 GPIO4 可用作其他用途。
- GPIO15 线上的上拉电阻 R68 移至 JTAG 的 MTDO 侧。

- 为了加强 A 板和 B 板连接间的防呆设计（减少反向插入 B 板的机会），将原先 A 板上的 2 排 4 针排针改为 1 排 4 针排母和 1 排 4 针排针。相应的 4 针排针排和排母排则安装在 B 板上。

ESP32-Ethernet-Kit 的其他版本

- *ESP32-Ethernet-Kit V1.0 入门指南*

相关文档

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

ESP32-Ethernet-Kit V1.0 入门指南

[English]

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

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

准备工作

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

您可以跳过介绍部分，直接前往*应用程序开发* 章节。

概述

ESP32-Ethernet-Kit 是一款来自 乐鑫 的开发板，由以太网子板（A 板）和 PoE 子板（B 板）两部分组成。其中*以太网子板（A 板）* 贴蓝牙 / Wi-Fi 双模 ESP32-WROVER-B 模组和单端口 10/100 快速以太网收发器 (PHY) IP101GRI。*PoE 子板（B 板）* 提供以太网供电功能。ESP32-Ethernet-Kit 的 A 板可在不连接 B 板的情况下独立工作。

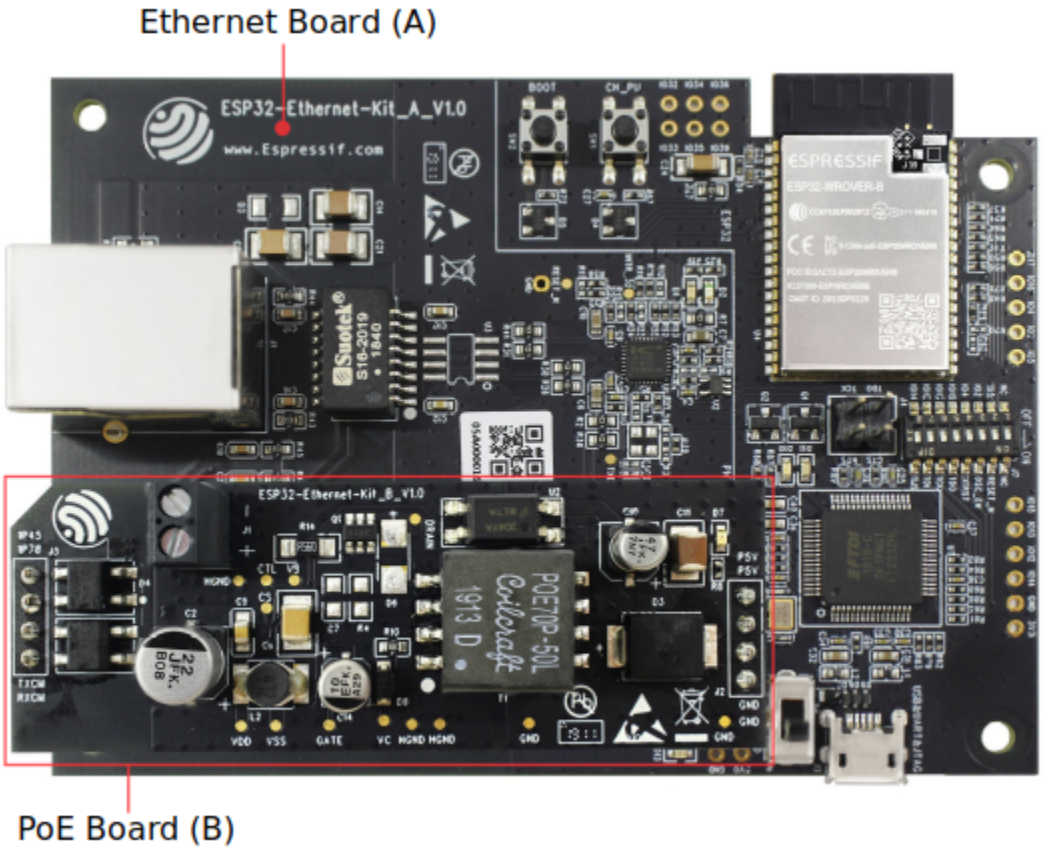


图 22: ESP32-Ethernet-Kit V1.0

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

功能概述

ESP32-Ethernet-Kit 开发板的主要组件和连接方式见下。

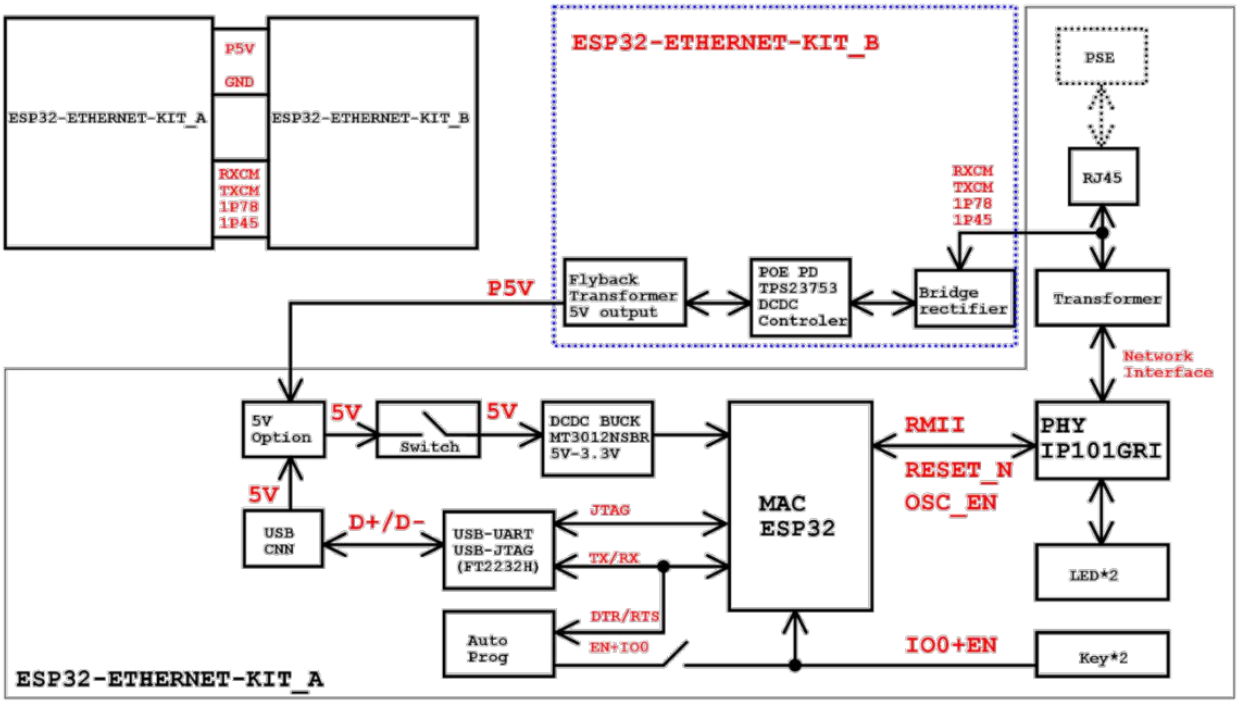


图 23: ESP32-Ethernet-Kit 功能框图（点击放大）

功能说明

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

以太网子板（A 板）

下表将从图片右上角开始，以顺时针顺序介绍图中的主要组件。

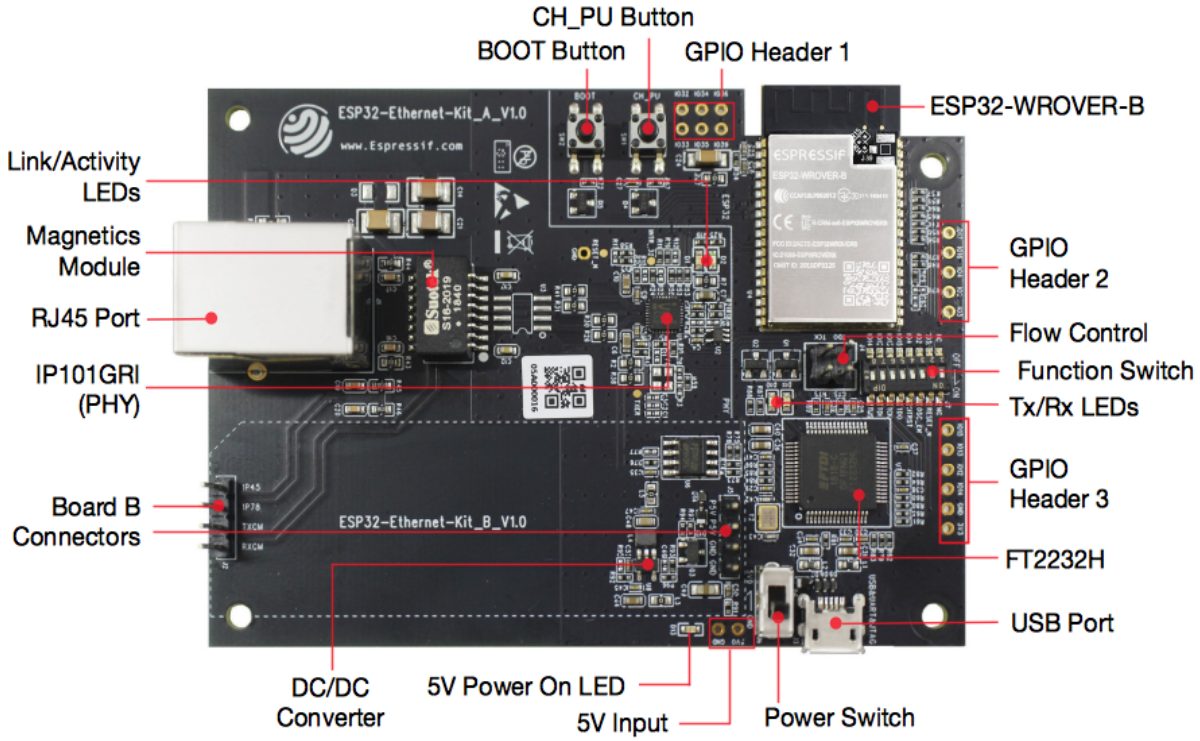


图 24: ESP32-Ethernet-Kit - 以太网子板（A 板）布局（点击放大）

主要组件	基本介绍
ESP32-WROVER-B 模组	这款 ESP32 模组内置 64-Mbit PSRAM，可提供灵活的额外存储空间和数据处理能力。
GPIO Header 2	由 5 个未引出通孔组成，可连接至 ESP32 的部分 GPIO。具体介绍，请见*GPIO Header 2*。
流控	跳线帽，可接入开发板信号。具体介绍，请见流控。
功能选择开关	DIP 开关，可配置 ESP32 部分 GPIO 的功能。具体介绍，请见功能选择开关。
Tx/Rx LED	2 个 LED，可显示 UART 传输的状态。
GPIO Header 3	可连接至 ESP32 的部分 GPIO，根据功能选择开关 的位置有不同功能。
FT2232H	FT2232H 多协议 USB 转串口桥接器。开发人员可通过 USB 接口对 FT2232H 芯片进行控制和编程，与 ESP32 建立连接。FT2232H 芯片可在通道 A 提供 USB-to-JTAG 接口功能，并在通道 B 提供 USB-to-Serial 接口功能，便利开发人员的应用开发与调试。见 ESP32-Ethernet-Kit V1.0 以太网子板（A 板）原理图。
USB 口	USB 接口。可用作开发板的供电电源，或连接 PC 和开发板的通信接口。
电源开关	电源开关。拨向 **Boot** 按键一侧，开发板上电；拨离 **Boot** 按键一侧，开发板掉电。

PoE 子板（B 板）

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

PoE 子板（B 板）具有以下特性：

- 支持 IEEE 802.3at
- 电源输出：5 V，1.4 A

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

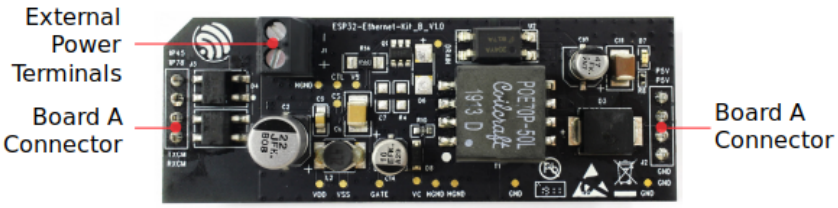


图 25: ESP32-Ethernet-Kit - PoE 子板（B 板）布局（点击放大）

主要组件	基本介绍
A 板连接器	1 个 4 针排母，用于将 B 板连接至以太网子板（*A 板*）。
外部电源终端	PoE 子板（B 板）备用电源。

设置选项

本节介绍用于 ESP32-Ethernet-Kit 开发板的硬件配置选项。

功能选择开关

用户可使用 **功能选择开关**配置特定 GPIO 管脚的功能。

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

用户还可以关闭 DIP DW，将部分 GPIO 用于其他用途。

流控

2 x 2 跳线连接器，可用于 UART 流控功能。

.	信号	备注
1.	MTDO	GPIO13，见功能选择开关。
2.	MTCK	GPIO15，见功能选择开关。
3.	RTS	FT2232H 的 RTS 信号
4.	CTS	FT2232H 的 CTS 信号

GPIO 分配

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

IP101GRI (PHY) 接口

ESP32 (MAC) 与 IP101GRI (PHY) 的管脚对应关系见下：

.	ESP32 管脚 (MAC)	IP101GRI (PHY)
RMII 接口		
1.	GPIO21	TX_EN
2.	GPIO19	TXD[0]
3.	GPIO22	TXD[1]
4.	GPIO25	RXD[0]
5.	GPIO26	RXD[1]
6.	GPIO27	CRS_DV
7.	GPIO0	REF_CLK
串行管理接口		
8	GPIO23	MDC
9	GPIO18	MDIO
PHY 复位		
10	GPIO5	Reset_N

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

GPIO Header 1

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

.	ESP32 管脚
1.	GPIO32
2.	GPIO33
3.	GPIO34
4.	GPIO35
5.	GPIO36
6.	GPIO39

GPIO Header 2

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

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

注解:

1. ESP32 芯片的 GPIO16 和 GPIO17 管脚没有引出至 ESP32-WROVER-B 模组的管脚，因此无法使用。如需使用 ESP32 的 GP1016 和 GPIO17 管脚，建议更换其他不含 SPIRAM 的模组，比如 ESP32-

WROOM-32D 或 ESP32-SOLO-1。

2. 具体功能取决与功能选择开关 的设置。

GPIO Header 3

本连接器中 GPIO 的功能取决与功能选择开关 的设置。

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

GPIO 管脚分配

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

注解:

1. GPIO2 用于使能 PHY 的外部振荡器。
2. GPIO0 用于为 PHY 提供 50 MHz 基准时钟源。为了防止传输线路延迟对时钟相位带来的影响，该时钟信号将首先被反相，而后提供给 PHY。
3. 为防止 PHY 端 GPIO0 的上电状态受到时钟输出的影响，PHY 的外部晶振将在 ESP32 上电后通过 GPIO2 使能。
4. ESP32 芯片的 GPIO16 和 GPIO17 管脚没有引出至 ESP32-WROVER-B 模组的管脚，因此无法使用。如需使用 ESP32 的 GP1016 和 GPIO17 管脚，建议更换其他不含 SPIRAM 的模组，比如 ESP32-

WROOM-32D 或 ESP32-SOLO-1。

应用程序开发

ESP32-Ethernet-Kit 上电前，请首先确认开发板完好无损。

初始设置

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

正式开始开发

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

如需使用较早 GNU Make 编译系统，则请参考*详细安装步骤* 章节。

请务必在进入下一步前，确保您已完成上述所有步骤。

配置与加载以太网示例

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

相关文档

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

1.4 详细安装步骤

请根据下方详细步骤，完成安装过程。

1.4.1 设置开发环境

- 第一步：安装准备 （*Windows* 、*Linux* 和*macOS*）
- 第二步：获取 *ESP-IDF*
- 第三步：设置工具
- 第四步：设置环境变量

1.4.2 创建您的第一个工程

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

1.5 第一步：安装准备

在正式开始创建工程前，请先完成工具的安装，具体步骤见下：

1.5.1 Windows 平台工具链的标准设置

[English]

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

概述

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

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

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

ESP-IDF 工具安装器

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

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

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

本安装器还可用于下载任意 ESP-IDF 发布版本。

使用命令提示符

在后续步骤中，我们将使用 Windows 的命令提示符进行操作。

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

注意，本快捷方式仅适用 ESP-IDF 工具安装器中指定的 ESP-IDF 路径。如果您的电脑上存在多个 ESP-IDF（比如您需要不同的 ESP-IDF 版本）需要使用快捷方式，您可以：

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

后续步骤

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

相关文档

想要自定义安装流程的高阶用户可参照：

从零开始设置 Windows 环境下的工具链

[English]

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

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

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

获取 ESP-IDF

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

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

打开命令提示符，运行以下命令：

```
mkdir %userprofile%\esp
cd %userprofile%\esp
git clone -b v4.1 --recursive https://github.com/espressif/esp-idf.git
```

ESP-IDF 将下载至 `%userprofile%\esp\esp-idf`。

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

注解： `git clone` 命令的 `-b v4.1` 选项告诉 git 从 ESP-IDF 仓库中克隆与此版本的文档对应的分支。

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

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

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

工具

cmake 工具

下载最新发布的 Windows 平台稳定版 CMake，并运行安装器。

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

Ninja 编译工具

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

从（下载页面）下载最新发布的 Windows 平台稳定版 ninja。

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

Python

下载并运行适用于 Windows 安装器的最新版 Python。

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

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

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

适用于 IDF 的 MConf

从 kconfig-frontends releases page 下载配置工具 mconf-idf。此为 `mconf` 配置工具，可针对 ESP-IDF 进行少量自定义操作。

请将此工具解压到目录，并添加到您的路径。

工具链设置

下载预编译的 Windows 工具链:

https://dl.espressif.com/dl/xtensa-esp32-elf-gcc8_2_0-esp-2020r2-win32.zip

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

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

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

添加目录到路径

在 Windows 环境下，向 Path 环境变量增加任何新目录，请:

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

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

后续步骤

继续设置开发环境，请前往第二步：获取 *ESP-IDF* 章节。

在 Windows 环境下更新 ESP-IDF 工具

使用脚本安装 ESP-IDF 工具

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

```
install.bat
```

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

```
install.ps1
```

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

使用“导出脚本”将 ESP-IDF 工具添加至 PATH

ESP-IDF 工具安装器将在“开始菜单”为“ESP-IDF 命令提示符”创建快捷方式。点击该快捷方式可打开 Windows 命令提示符窗口，您可在该窗口使用所有已安装的工具。

有些情况下，您正在使用的 ESP-IDF 版本可能并未创建命令提示符快捷方式，此时您可以根据下方步骤将 ESP-IDF 工具添加至 PATH。

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

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

对于 Powershell 用户，请同样切换至 ESP-IDF 的安装路径，然后执行 `export.ps1`:

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

运行完成后，您就可以通过命令提示符使用 ESP-IDF 工具了。

1.5.2 Linux 平台工具链的标准设置

[English]

安装准备

编译 ESP-IDF 需要以下软件包：

- CentOS 7:

  ```
  sudo yum install git wget flex bison gperf python cmake ninja-build ccache
  ```

- Ubuntu 和 Debian:

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

- Arch:

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

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

其他提示

权限问题 /dev/ttyUSB0

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

后续步骤

继续设置开发环境，请前往*第二步：获取 ESP-IDF* 章节。

相关文档

从零开始设置 Linux 环境下的工具链

[English]

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

安装准备

编译 ESP-IDF 需要以下软件包：

- CentOS 7:

```
sudo yum install git wget ncurses-devel flex bison gperf python pyserial python-
↪pyelftools cmake ninja-build ccache
```

- Ubuntu 和 Debian:

```
sudo apt-get install git wget libncurses-dev flex bison gperf python python-pip␣
↪python-setuptools python-serial python-click python-cryptography python-future␣
↪python-pyparsing python-pyelftools cmake ninja-build ccache libffi-dev libssl-dev
```

- Arch:

```
sudo pacman -S --needed gcc git make ncurses flex bison gperf python-pyserial␣
↪python-click python-cryptography python-future python-pyparsing python-pyelftools␣
↪cmake ninja ccache
```

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

从源代码编译工具链

安装依赖项:

- CentOS 7:

```
sudo yum install gawk gperf grep gettext ncurses-devel python python-devel automake␣
↪bison flex texinfo help2man libtool make
```

- Ubuntu pre-16.04:

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

- Ubuntu 16.04 或以上

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

- Debian 9:

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

- Arch:

```
TODO
```

创建工作目录，并进入该目录:

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

下载并编译 crosstool-NG :

```
git clone https://github.com/espressif/crosstool-NG.git
cd crosstool-NG
git checkout esp-2020r2
```

(下页继续)

(续上页)

```
git submodule update --init
./bootstrap && ./configure --enable-local && make
```

编译工具链:

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

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

后续步骤

继续设置开发环境，请前往第二步：获取 *ESP-IDF* 章节。

1.5.3 MacOS 平台工具链的标准设置

[English]

安装准备

ESP-IDF 将使用 Mac OS 上默认安装的 Python 版本。

- 安装 pip:

  ```
  sudo easy_install pip
  ```

- 安装 pyserial:

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

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

    ```
    brew install cmake ninja
    ```

  - 若有 MacPorts，您可以运行:

    ```
    sudo port install cmake ninja
    ```

  - 若以上均不适用，请访问 CMake 和 Ninja 主页，查询有关 Mac OS 平台的下载安装问题。

- 强烈建议同时安装 ccache 以获得更快的编译速度。如有 HomeBrew，可通过 MacPorts 上的 `brew install ccache` 或 `sudo port install ccache` 完成安装。

注解： 如您在上述任何步骤中遇到以下错误:

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

则必须安装 XCode 命令行工具，具体可运行 `xcode-select --install`。

后续步骤

继续设置开发环境，请前往第二步：获取 *ESP-IDF* 章节。

相关文档

从零开始设置 MacOS 环境下的工具链

[English]

软件包管理器

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

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

请参考工具链自定义设置 章节，查看可能需要从头开始设置工具链的情况。

安装准备

- 安装 pip:

  ```
  sudo easy_install pip
  ```

- 安装 pyserial:

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

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

```
brew install cmake ninja
```

– 若有 MacPorts，您可以运行:

```
sudo port install cmake ninja
```

从源代码编译工具链

安装依赖项:

- 对于 MacPorts:

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

- 对于 homebrew:

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

创建一个文件系统镜像（区分大小写）:

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

挂载:

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

创建指向您工作目录的符号链接:

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

前往新创建的目录

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

下载并编译 crosstool-NG

```
git clone https://github.com/espressif/crosstool-NG.git
cd crosstool-NG
git checkout esp-2020r2
git submodule update --init
./bootstrap && ./configure --enable-local && make
```

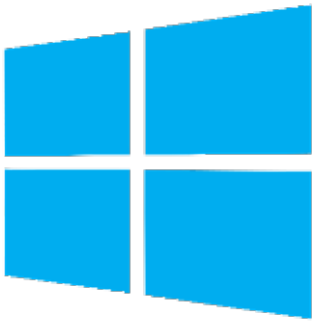
编译工具链:

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

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

后续步骤

继续设置开发环境，请前往*第二步：获取 ESP-IDF* 章节。

		
Windows	Linux	Mac OS

1.6 第二步：获取 ESP-IDF

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

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

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

1.6.1 Linux 和 MacOS 操作系统

打开终端，后运行以下命令:

```
cd ~/esp
git clone -b v4.1 --recursive https://github.com/espressif/esp-idf.git
```

ESP-IDF 将下载至 ~/esp/esp-idf。

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

1.6.2 Windows 操作系统

除了安装必要工具外，第一步中介绍的*ESP-IDF 工具安装器* 也能同时下载 ESP-IDF 本地副本。

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

除了使用 ESP-IDF 工具安装器，您也可以参考指南 手动下载 ESP-IDF。

1.7 第三步：设置工具

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

1.7.1 Windows 操作系统

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

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

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

或使用 Windows PowerShell

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

1.7.2 Linux 和 MacOS 操作系统

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

1.7.3 自定义工具安装路径

本步骤中介绍的脚本将 ESP-IDF 所需的编译工具默认安装在用户根文件夹中，即 Linux 和 MacOS 系统中的 `$HOME/.espressif` 和 Windows 系统的 `%USERPROFILE%\.espressif`。此外，您可以可以将工具安装到其他目录中，但请在运行安装脚本前，重新设置环境变量 `IDF_TOOLS_PATH`。注意，请确保您的用户已经具备了读写该路径的权限。

如果修改了 `IDF_TOOLS_PATH` 变量，请确保该变量在每次执行“安装脚本”(`install.bat`、`install.ps1` 或 `install.sh`) 和导出脚本 (`export.bat`、`export.ps1` 或 `export.sh`) 均保持一致。

1.8 第四步：设置环境变量

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

1.8.1 Windows 操作系统

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

此外，如果您希望在当下命令提示符窗口使用 ESP-IDF，请使用下方代码：

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

或使用 Windows PowerShell

```
.$HOME/esp/esp-idf/export.ps1
```

1.8.2 Linux 和 MacOS 操作系统

请在您需要运行 ESP-IDF 的“命令提示符”窗口运行以下命令：

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

注意，命令开始的“.”与路径之间应有一个空格！

此外，您也可以将这行代码增加至您的 `.profile` 或 `.bash_profile` 脚本中，这样您就可以在任何命令窗口使用 ESP-IDF 工具了。

1.9 第五步：开始创建工程

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

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

1.9.1 Linux 和 MacOS 操作系统

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

1.9.2 Windows 操作系统

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

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

重要： ESP-IDF 编译系统不支持带有空格的路径。

1.10 第六步：连接设备

现在，请将您的 ESP32 开发板连接到 PC，并查看开发板使用的串口。

通常，串口在不同操作系统下显示的名称有所不同：

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

有关如何查看串口名称的详细信息，请见*与 ESP32 创建串口连接*。

注解： 请记住串口名，您会在下面的步骤中用到。

1.11 第七步：配置

请进入第五步：开始创建工程 中提到的 hello_world 目录，并运行工程配置工具 menuconfig。

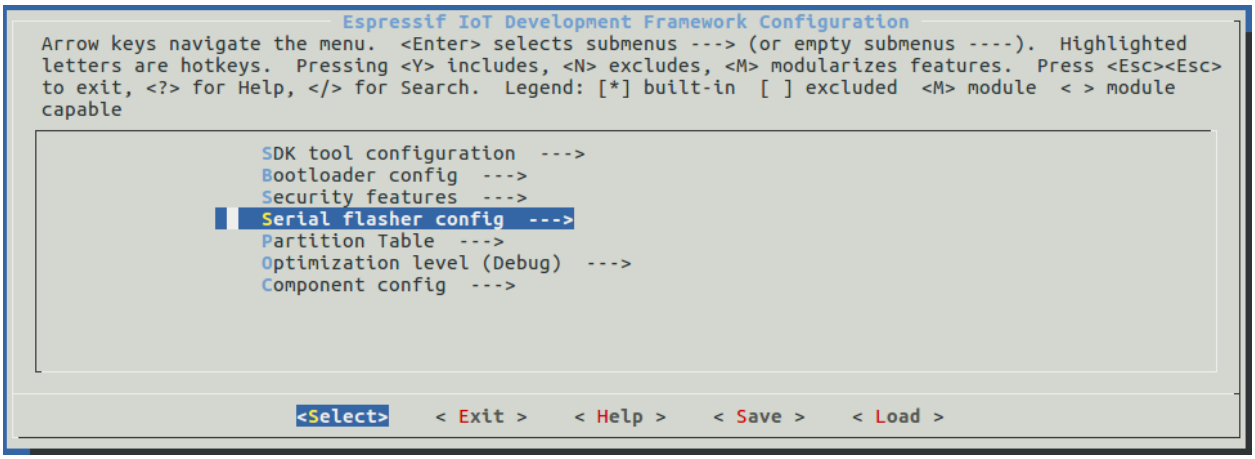
1.11.1 Linux 和 MacOS 操作系统

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

1.11.2 Windows 操作系统

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

如果之前的步骤都正确，则会显示下面的菜单：



工程配置—主窗口

menuconfig 工具的常见操作见下。

- 上下箭头：移动
- 回车：进入子菜单
- ESC 键：返回上级菜单或退出
- 英文问号：调出帮助菜单（退出帮助菜单，请按回车键）。
- 空格 ``或 ``Y 键：选择 [*] 配置选项；N 键：禁用 [*] 配置选项
- 英文问号（查询配置选项）：调出有关该选项的帮助菜单
- / 键：寻找配置工程

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

1.12 第八步：编译工程

请使用以下命令，编译烧录工程：：

```
idf.py build
```

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

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

... (more lines of build system output)

[527/527] Generating hello-world.bin
esptool.py v2.3.1

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

如果一切正常，编译完成后将生成.bin 文件。

1.13 第九步：烧录到设备

请使用以下命令，将刚刚生成的二进制文件烧录至您的 ESP32 开发板：

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

请将 PORT 替换为 ESP32 开发板的串口名称，具体可见第六步：连接设备。

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

更多有关 idf.py 参数的详情，请见*idf.py*。

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

```
Running esptool.py in directory [...]/esp/hello_world
Executing "python [...]/esp-idf/components/esptool_py/esptool/esptool.py -b 460800 write_
↪flash @flash_project_args"...
esptool.py -b 460800 write_flash --flash_mode dio --flash_size detect --flash_freq 40m␣
↪0x1000 bootloader/bootloader.bin 0x8000 partition_table/partition-table.bin 0x10000␣
↪hello-world.bin
esptool.py v2.3.1
Connecting....
Detecting chip type... ESP32
Chip is ESP32D0WDQ6 (revision 1)
Features: WiFi, BT, Dual Core
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 460800
Changed.
Configuring flash size...
Auto-detected Flash size: 4MB
Flash params set to 0x0220
Compressed 22992 bytes to 13019...
Wrote 22992 bytes (13019 compressed) at 0x00001000 in 0.3 seconds (effective 558.9 kbit/
↪s)...
Hash of data verified.
Compressed 3072 bytes to 82...
Wrote 3072 bytes (82 compressed) at 0x00008000 in 0.0 seconds (effective 5789.3 kbit/s)..
↪.
Hash of data verified.
Compressed 136672 bytes to 67544...
Wrote 136672 bytes (67544 compressed) at 0x00010000 in 1.9 seconds (effective 567.5 kbit/
↪s)...
Hash of data verified.

Leaving...
```

(下页继续)

(续上页)

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

如果一切顺利，烧录完成后，开发板将会复位，应用程序“hello_world”开始运行。

注解: （目前不支持）如果您希望使用 Eclipse IDE，而非 `idf.py`，请参考*Eclipse 指南*。

1.14 第十步：监视器

您可以使用 `make monitor` 命令，监视“hello_world”的运行情况。注意，不要忘记将 PORT 替换为您的串口名称。

运行该命令后，*IDF 监视器* 应用程序将启动：：

```
$ idf.py -p /dev/ttyUSB0 monitor
Running idf_monitor in directory [...]/esp/hello_world/build
Executing "python [...]/esp-idf/tools/idf_monitor.py -b 115200 [...]/esp/hello_world/
↪build/hello-world.elf"...
--- idf_monitor on /dev/ttyUSB0 115200 ---
--- Quit: Ctrl+] | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---
ets Jun  8 2016 00:22:57

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
ets Jun  8 2016 00:22:57
...
```

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

```
...
Hello world!
Restarting in 10 seconds...
I (211) cpu_start: Starting scheduler on APP CPU.
Restarting in 9 seconds...
Restarting in 8 seconds...
Restarting in 7 seconds...
```

您可使用快捷键 `Ctrl+]`，退出 IDF 监视器。

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

此时，您可以：

```
e���)(Xn@@y.!@@(@PW+)@@Hn9a￣/9@!@t5@@P@~@k@@e@ea@5@jA
~zY@@Y(1@,1 @@ e���)(Xn@@y.!Dr@zY(@ jpi@|@+z5Ymvp
```

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

注解： 您也可以运行以下命令，一次性执行构建、烧录和监视过程：

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

此外，

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

恭喜，您已完成 ESP32 的入门学习！

现在，您可以尝试一些其他 examples，或者直接开发自己的应用程序。

1.15 更新 ESP-IDF

乐鑫会不时推出更新版本的 ESP-IDF，修复 bug 或提出新的特性。因此，您在使用时，也应注意更新您本地的版本。最简单的方法是：直接删除您本地的 `esp-idf` 文件夹，然后按照第二步：获取 *ESP-IDF* 中的指示，重新完成克隆。

此外，您可以仅更新变更部分。具体方式，请前往更新 章节查看。

注意，更新完成后，请执行 `install.sh` （Windows 系统中为 `install.bat`）脚本，避免新版 ESP-IDF 所需的工具也有所更新。具体请参考第三步：设置工具。

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

1.16 相关文档

1.16.1 与 ESP32 创建串口连接

[English]

本章节主要介绍如何创建 ESP32 和 PC 之间的串口连接。

连接 ESP32 和 PC

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

以下是乐鑫 ESP32 开发板驱动程序的链接：

开发板	USB 驱动	备注
ESP32-DevKitC	CP210x	
ESP32-LyraT	CP210x	
ESP32-LyraTD-MSC	CP210x	
ESP32-PICO-KIT	CP210x	
ESP-WROVER-KIT	FTDI	
ESP32 Demo 板	FTDI	
ESP-Prog	FTDI	编程板 (w/o ESP32)
ESP32-MeshKit-Sense	n/a	搭配 ESP-Prog 使用
ESP32-Sense Kit	n/a	搭配 ESP-Prog 使用

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

以上驱动仅用于参考。一般情况下，当上述任一 ESP32 开发板与 PC 连接时，对应驱动程序应该已经被打包在操作系统中，并已经自动安装。

在 Windows 上查看端口

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

以下为 ESP32 DevKitC 和 ESP32 WROVER KIT 串口：

在 Linux 和 MacOS 上查看端口

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

Linux

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

MacOS

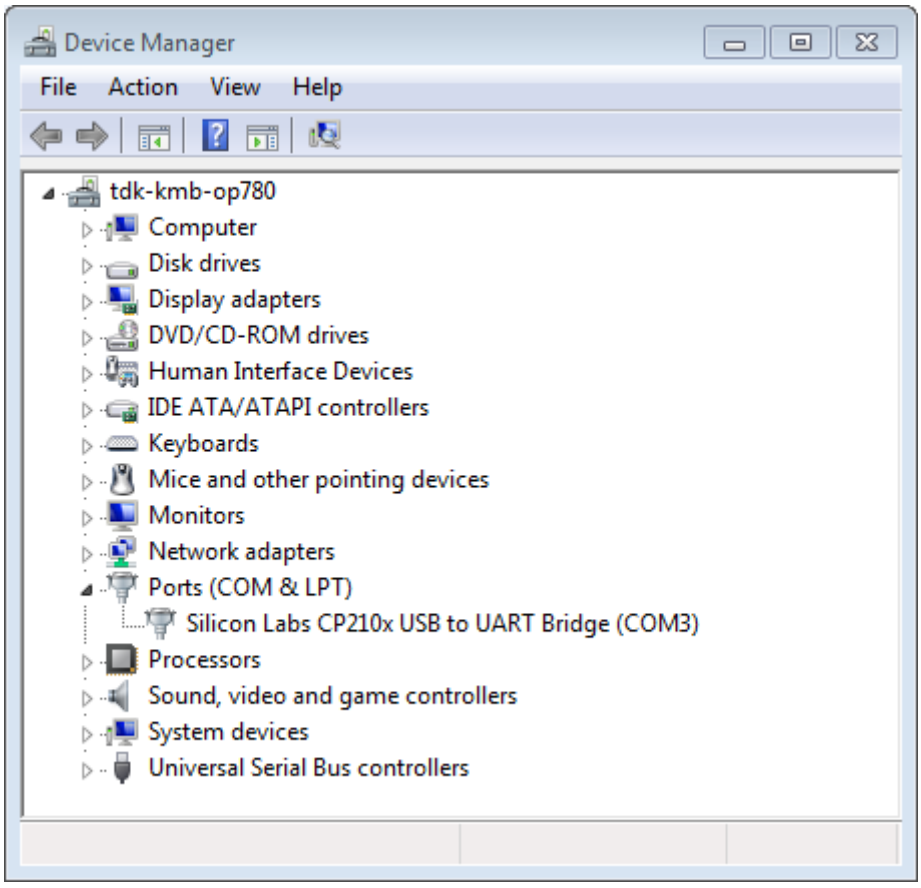


图 26: 设备管理器中 ESP32-DevKitC 的 USB 至 UART 桥

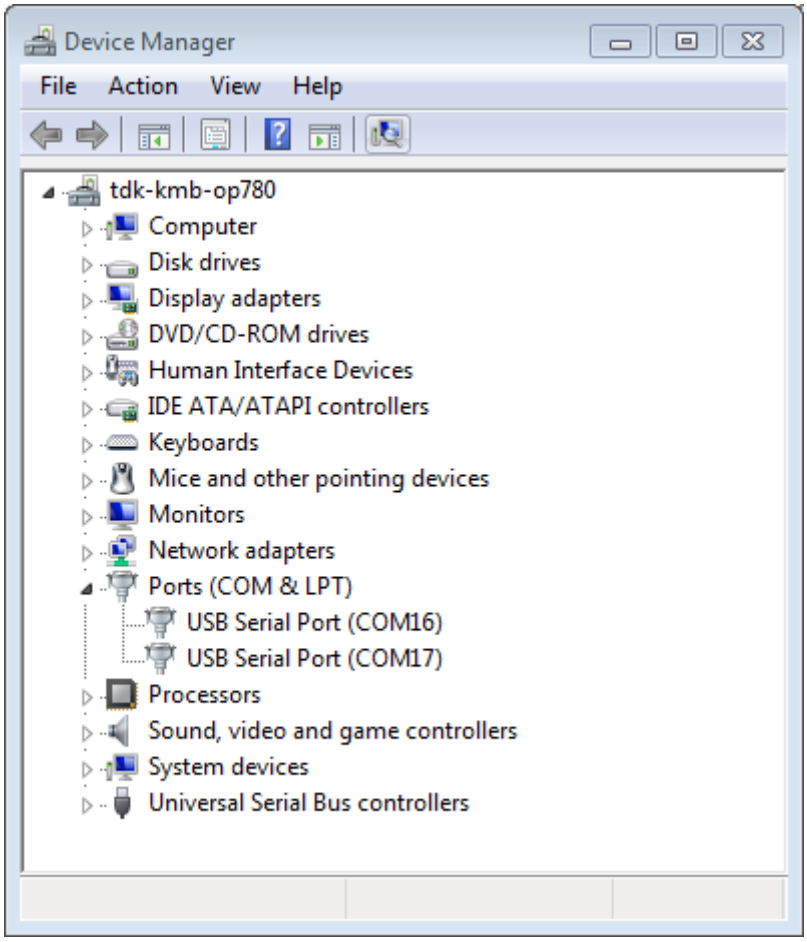


图 27: Windows 设备管理器中 ESP-WROVER-KIT 的两个 USB 串行端口

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

注解: 对于 MacOS 用户：若你没有看到串口，请检查你是否已按照《入门指南》安装了适用于你特定开发板的 USB/串口驱动程序。对于 MacOS High Sierra (10.13) 的用户，你可能还需要手动允许驱动程序的加载，具体可打开 *系统偏好设置* -> *安全和隐私* -> *通用*，检查是否有信息显示："来自开发人员的系统软件…"，其中开发人员的名称为 Silicon Labs 或 FTDI。

在 Linux 中添加用户到 dialout

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

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

在 Arch Linux 中，需要通过以下命令将用户添加到 uucp 组中:

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

请重新登录，确保串口读写权限可以生效。

确认串口连接

现在，请使用串口终端程序，验证串口连接是否可用。在本示例中，我们将使用 PuTTY SSH Client，PuTTY SSH Client 既可用于 Windows 也可用于 Linux。你也可以使用其他串口程序并设置如下的通信参数。

运行终端，配置串口：波特率 = 115200，数据位 = 8，停止位 = 1，奇偶校验 = N。以下截屏分别展示了在 Windows 和 Linux 中配置串口和上述通信参数（如 115200-8-1-N）。注意，这里一定要选择在上述步骤中确认的串口进行配置。

然后，请检查 ESP32 是否有打印日志。如有，请在终端打开串口进行查看。这里，日志内容取决于加载到 ESP32 的应用程序，下图即为一个示例。

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

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

rst:0x7 (TG0WDT_SYS_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0x00
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
```

(下页继续)

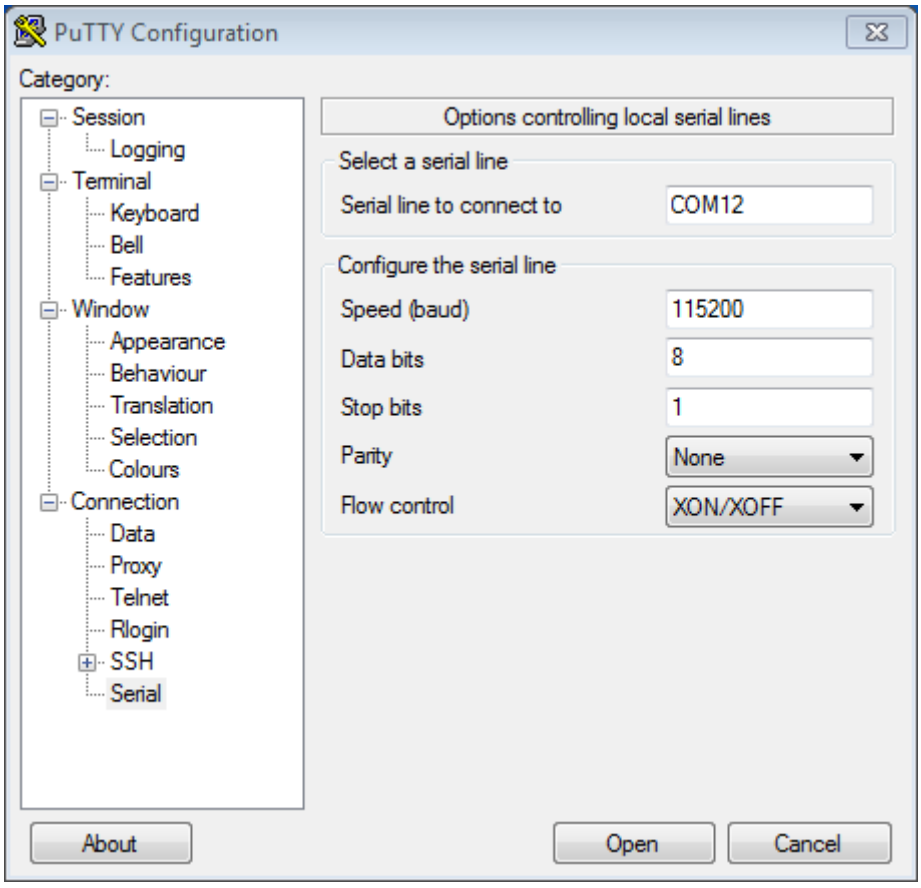


图 28: 在 Windows 操作系统中使用 PuTTY 设置串口通信参数

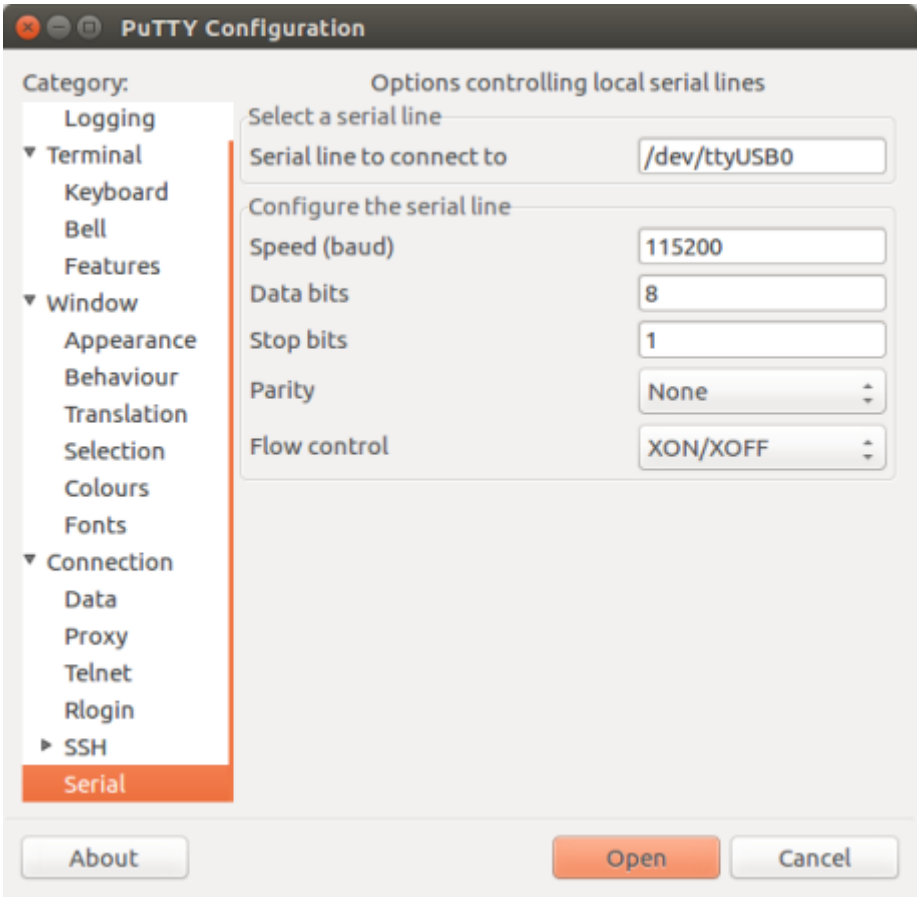


图 29: 在 Linux 操作系统中使用 PuTTY 设置串口通信参数

(续上页)

```
load:0x3fff0008,len:8
load:0x3fff0010,len:3464
load:0x40078000,len:7828
load:0x40080000,len:252
entry 0x40080034
I (44) boot: ESP-IDF v2.0-rc1-401-gf9fba35 2nd stage bootloader
I (45) boot: compile time 18:48:10

...
```

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

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

注解： 请在验证完串口通信正常后，关闭串口终端。下一步，我们将使用另一个应用程序将新的固件上传到 ESP32。此时，如果串口被占用则无法成功。

如你在安装 ESP32 硬件开发的软件环境时，从第六步：连接设备 跳转到了这里，请从第七步：配置 继续阅读。

1.16.2 Eclipse IDE 创建和烧录指南

[English]

ESP-IDF V4.0 将默认采用基于 CMake 的编译系统。

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

如您仍需要对传统 GNU Make 编译系统的 Eclipse 支持，请前往传统 *GNU Make* 编译系统入门指南，查看使用 *Eclipse IDE* 进行编译与烧录 章节。

1.16.3 IDF 监视器

[English]

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

在 IDF 中调用以下目标函数可以启用此监视器：

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

操作快捷键

为了方便与 IDF 监视器进行交互，请使用表中给出的快捷键。

快捷键	操作	描述
Ctrl+]	退出监视器程序	
Ctrl+T	菜单退出键	按下如下给出的任意键，并按指示操作。
• Ctrl+T	将菜单字符发送至远程	
• Ctrl+]	将 exit 字符发送至远程	
• Ctrl+P	重置目标设备，进入 Bootloader，通过 RTS 线暂停应用程序	重置目标设备，通过 RTS 线（如已连接）进入 Bootloader，此时开发板不运行任何程序。等待其他设备启动时可以使用此操作。
• Ctrl+R	通过 RTS 线重置目标设备	重置设备，并通过 RTS 线（如已连接）重新启动应用程序。
• Ctrl+F	编译并烧录此项目	暂停 idf_monitor，运行 `idf.py flash` 目标，然后恢复 idf_monitor。任何改动的源文件都会被重新编译，然后重新烧录。
• Ctrl+A (A)	仅编译及烧录应用程序	暂停 idf_monitor，运行 `app-flash` 目标，然后恢复 idf_monitor。这与 `flash` 类似，但只有主应用程序被编译并被重新烧录。
• Ctrl+Y	停止/恢复日志输出在屏幕上打印	激活时，会丢弃所有传入的串行数据。允许在不退出监视器的情况下快速暂停和检查日志输出。
• Ctrl+L	停止/恢复向文件写入日志输出	在工程目录下创建一个文件，用于写入日志输出。可使用快捷键停止/恢复该功能（退出 IDF 监视器也会终止该功能）
• Ctrl+H (H)	显示所有快捷键	

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

兼具 IDF 特性

自动解码地址

ESP-IDF 输出形式为 0x4_______ 的十六进制代码地址后，IDF 监视器将使用 addr2line 查找该地址在源代码中的位置和对应的函数名。

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

```
Guru Meditation Error of type StoreProhibited occurred on core  0. Exception was␣
↪unhandled.
Register dump:
PC      : 0x400f360d  PS      : 0x00060330  A0      : 0x800dbf56  A1      : 0x3ffb7e00
A2      : 0x3ffb136c  A3      : 0x00000005  A4      : 0x00000000  A5      : 0x00000000
A6      : 0x00000000  A7      : 0x00000080  A8      : 0x00000000  A9      : 0x3ffb7dd0
A10     : 0x00000003  A11     : 0x00060f23  A12     : 0x00060f20  A13     : 0x3ffba6d0
A14     : 0x00000047  A15     : 0x0000000f  SAR     : 0x00000019  EXCCAUSE: 0x0000001d
EXCVADDR: 0x00000000  LBEG    : 0x4000c46c  LEND    : 0x4000c477  LCOUNT  : 0x00000000

Backtrace: 0x400f360d:0x3ffb7e00 0x400dbf56:0x3ffb7e20 0x400dbf5e:0x3ffb7e40␣
↪0x400dbf82:0x3ffb7e60 0x400d071d:0x3ffb7e90
```

IDF 监视器为寄存器转储补充如下信息:

```
Guru Meditation Error of type StoreProhibited occurred on core  0. Exception was␣
↪unhandled.
Register dump:
PC      : 0x400f360d  PS      : 0x00060330  A0      : 0x800dbf56  A1      : 0x3ffb7e00
0x400f360d: do_something_to_crash at /home/gus/esp/32/idf/examples/get-started/hello_
↪world/main/./hello_world_main.c:57
(inlined by) inner_dont_crash at /home/gus/esp/32/idf/examples/get-started/hello_world/
↪main/./hello_world_main.c:52
A2      : 0x3ffb136c  A3      : 0x00000005  A4      : 0x00000000  A5      : 0x00000000
A6      : 0x00000000  A7      : 0x00000080  A8      : 0x00000000  A9      : 0x3ffb7dd0
A10     : 0x00000003  A11     : 0x00060f23  A12     : 0x00060f20  A13     : 0x3ffba6d0
A14     : 0x00000047  A15     : 0x0000000f  SAR     : 0x00000019  EXCCAUSE: 0x0000001d
EXCVADDR: 0x00000000  LBEG    : 0x4000c46c  LEND    : 0x4000c477  LCOUNT  : 0x00000000

Backtrace: 0x400f360d:0x3ffb7e00 0x400dbf56:0x3ffb7e20 0x400dbf5e:0x3ffb7e40␣
↪0x400dbf82:0x3ffb7e60 0x400d071d:0x3ffb7e90
0x400f360d: do_something_to_crash at /home/gus/esp/32/idf/examples/get-started/hello_
↪world/main/./hello_world_main.c:57
(inlined by) inner_dont_crash at /home/gus/esp/32/idf/examples/get-started/hello_world/
↪main/./hello_world_main.c:52
```

(下页继续)

(续上页)

```
0x400dbf56: still_dont_crash at /home/gus/esp/32/idf/examples/get-started/hello_world/
↪main/./hello_world_main.c:47
0x400dbf5e: dont_crash at /home/gus/esp/32/idf/examples/get-started/hello_world/main/./
↪hello_world_main.c:42
0x400dbf82: app_main at /home/gus/esp/32/idf/examples/get-started/hello_world/main/./
↪hello_world_main.c:33
0x400d071d: main_task at /home/gus/esp/32/idf/components/esp32/./cpu_start.c:254
```

IDF 监视器在后台运行以下命令，解码各地址:

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

配置 GDBStub 以启用 GDB

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

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

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

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

IDF 监控器在后台运行如下命令:

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

输出筛选

IDF 监视器有两种启用方式：运行 `idf.py monitor PRINT_FILTER=""`（适用于 CMake）或者 `make monitor PRINT_FILTER=""`（适用于传统 GNU Make），其中，`--print-filter` 是输出筛选的参数。参数默认值为空字符串，可打印任何内容。

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

例如，`PRINT_FILTER="tag1:W"` 只匹配并打印 `ESP_LOGW("tag1", ...)` 所写的输出，或者写在较低冗余级别的输出，即 `ESP_LOGE("tag1", ...)`。请勿指定 `<log_level>` 或使用冗余级别默认值 `*`。

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

应用程序标签不能包含空格、星号 *、分号 :，以便兼容输出筛选功能。

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

筛选规则示例

- * 可用于匹配任何类型标签。但 PRINT_FILTER="*:I tag1:E" 打印关于 tag1 的输出时会报错，这是因为 tag1 规则比 * 规则的优先级高。
- 默认规则（空）等价于 *:V，因为在冗余级别或更低级别匹配任意标签即意味匹配所有内容。
- "*:N" 不仅抑制了日志功能的输出，也抑制了 printf 的打印输出。为了避免这一问题，请使用 *:E 或更高的冗余级别。
- 规则 "tag1:V"、"tag1:v"、"tag1:"、"tag1:*" 和 "tag1" 等同。
- 规则 "tag1:W tag1:E" 等同于 "tag1:E"，这是因为后续出现的具有相同名称的标签会覆盖掉前一个标签。
- 规则 "tag1:I tag2:W" 仅在 Info 冗余级别或更低级别打印 tag1，在 Warning 冗余级别或更低级别打印 tag2。
- 规则 "tag1:I tag2:W tag3:N" 在本质上等同于上一规则，这是因为 tag3:N 指定 tag3 不打印。
- tag3:N 在规则 "tag1:I tag2:W tag3:N *:V" 中更有意义，这是因为如果没有 tag3:N，tag3 信息就可能打印出来了；tag1 和 tag2 错误信息会打印在指定的冗余级别（或更低级别），并默认打印所有内容。

高级筛选规则示例

如下日志是在没有设置任何筛选选项的情况下获得的:

```
load:0x40078000,len:13564
entry 0x40078d4c
E (31) esp_image: image at 0x30000 has invalid magic byte
W (31) esp_image: image at 0x30000 has invalid SPI mode 255
E (39) boot: Factory app partition is not bootable
I (568) cpu_start: Pro cpu up.
I (569) heap_init: Initializing. RAM available for dynamic allocation:
```

(下页继续)

(续上页)

```
I (603) cpu_start: Pro cpu start user code
D (309) light_driver: [light_init, 74]:status: 1, mode: 2
D (318) vfs: esp_vfs_register_fd_range is successful for range <54; 64) and VFS ID 1
I (328) wifi: wifi driver task: 3ffdbf84, prio:23, stack:4096, core=0
```

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

```
E (31) esp_image: image at 0x30000 has invalid magic byte
I (328) wifi: wifi driver task: 3ffdbf84, prio:23, stack:4096, core=0
```

`PRINT_FILTER="light_driver:D esp_image:N boot:N cpu_start:N vfs:N wifi:N *:V"` 选项的输出如下:

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

IDF 监视器已知问题

Windows 环境下已知问题

- 若在 Windows 环境下，出现 “winpty: command not found” 错误，请运行 `pacman -S winpty` 进行修复。
- 由于 Windows 控制台限制，有些箭头键及其他一些特殊键无法在 GDB 中使用。
- 偶然情况下，`idf.py` 或 `make` 退出时，可能会在 IDF 监视器恢复之前暂停 30 秒。
- GDB 运行时，可能会暂停一段时间，然后才开始与 GDBStub 进行通信。

1.16.4 工具链的自定义设置

[English]

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

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

- 需要定制工具链编译配置
- 需要使用其他 GCC 版本（如 4.8.5）
- 需要破解 gcc、newlib 或 libstdc++

- 有相关兴趣或时间充裕
- 不信任从网站下载的 bin 文件

如需自行编译工具链，请查看以下文档：

1.16.5 快速入门（传统 GNU Make）

[English]

注解： ESP-IDF V4.0 及之后版本的默认构建系统为 CMake。本文档主要针对之前基于 GNU Make 的传统构建系统。请注意，未来，我们可能不会继续支持基于 GNU Make 的构建系统。

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

注解： 这是 ESP-IDF 稳定版本 v4.1 的文档，还有其他版本的文档*ESP-IDF 版本简介* 供参考。

概述

ESP32 SoC 芯片支持以下功能：

- 2.4 GHz Wi-Fi
- 蓝牙 4.2
- 高性能双核
- 超低功耗协处理器
- 多种外设

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

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

准备工作

硬件：

- 一款 **ESP32** 开发板
- **USB 数据线** (A 转 Micro-B)

- PC（Windows、Linux 或 Mac OS）

软件：

- 设置 **工具链**，用于编译 ESP32 **应用程序**；
- 获取 **ESP-IDF** 软件开发框架。该框架已经基本包含 ESP32 使用的 API（软件库和源代码）和运行 **工具链**的脚本；
- 安装 C 语言编程（**工程**）的 **文本编辑器**，例如 Eclipse。

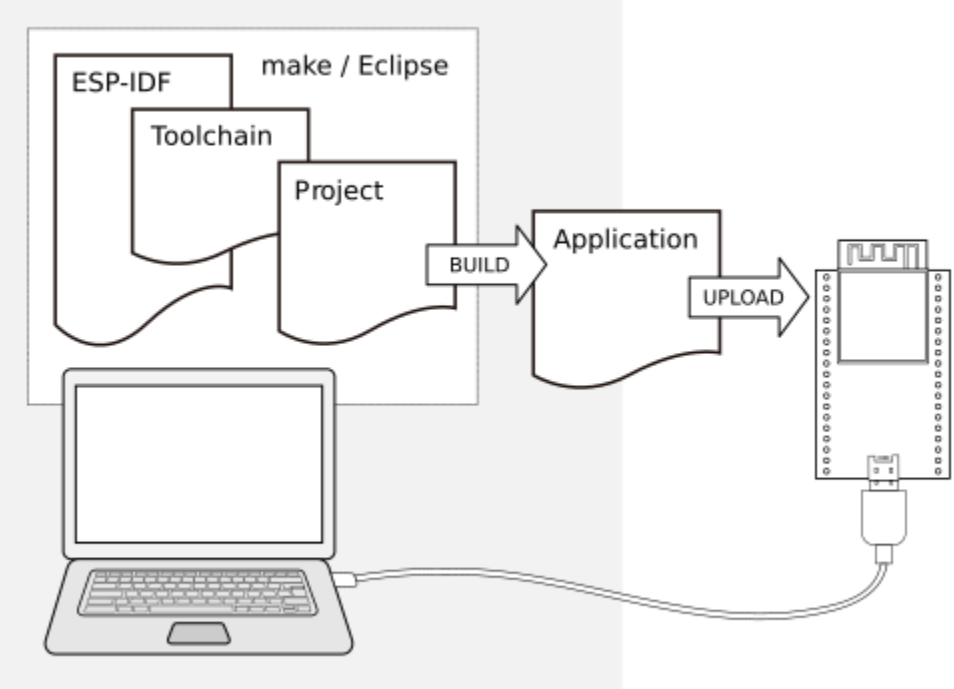


图 30: ESP32 应用程序开发

开发板简介

请点击下方连接，了解有关具体开发板的详细信息。

详细安装步骤

请根据下方详细步骤，完成安装过程。

设置开发环境

- *Windows*、*Linux* 和*macOS* 的第一步：设置工具链

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

创建您的第一个工程

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

第一步：设置工具链

工具链指一套用于编译代码和应用程序的程序。

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

Windows 平台工具链的标准设置（传统 GNU Make）

[English]

注解： ESP-IDF V4.0 及之后版本的默认构建系统为 CMake。本文档主要针对之前基于 GNU Make 的传统构建系统。请注意，未来，我们可能不会继续支持基于 GNU Make 的构建系统。

概述

Windows 系统没有内置的“make”环境，所以除了安装工具链之外，您还需要一个兼容 GNU 的环境。这里，我们使用 MSYS2 环境兼容 GNU。您无需一直使用这个环境，比如您可以使用*Eclipse* 或其他前端，仅需在后台运行 MSYS2 即可。

工具链设置

最简便的工具链设置方法是从下方地址下载 Windows 多合一工具链和 MSYS2 压缩包文件：

https://dl.espressif.com/dl/esp32_win32_msys2_environment_and_esp2020r2_toolchain-20200601.zip

请将压缩包文件解压至 C:\`` (或其他目录，但本文档中以 ``C:\ 为例)，该文件将为您创建一个带有预配置环境的 msys32 目录。

开始尝试

请运行 C:\msys32\mingw32.exe 文件，打开一个 MSYS2 MINGW32 终端窗口。该窗口的环境为 bash shell。请创建一个名为 esp 的文件夹，作为 ESP32 应用程序开发的默认目录。您可以使用以下命令创建文件夹:

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

您还可以通过 cd ~/esp 命令，进入刚刚创建的文件夹。如无其他问题，本步骤到此结束。

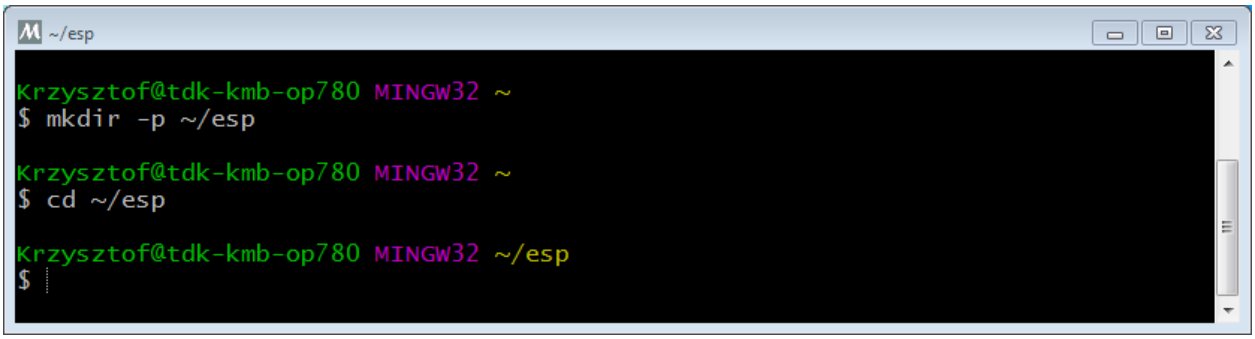


图 31: MSYS2 MINGW32 shell 窗口

请在后续步骤中，使用本窗口配置 ESP32 的开发环境。

后续步骤

继续设置开发环境，请前往第二步：获取 *ESP-IDF* 章节。

更新环境

当 IDF 有更新时，有时需要安装新的工具链或为 Windows MSYS2 环境添加新的要求。如需将旧版本预编译环境中的数据迁移至新版本，您可以:

1. 复制旧的 MSYS2 环境（即 C:\msys32)，并将其移动/重命名到不同目录下（即 C:\msys32_old)。
2. 使用上述步骤，下载新的预编译环境。
3. 将新的 MSYS2 环境解压缩至 C:\msys32 （或您指定的其他位置)。
4. 找到旧的 C:\msys32_old\home 文件夹，并将其移动至 C:\msys32。
5. 此时，如无其他需要，您可以删除旧的 C:\msys32_old\home 文件夹。

注意，您可以在电脑中安装多个不同的 MSYS2 环境，仅需将它们保存在不同的路径下即可。

此外，您还可以直接更新现有环境（无需下载新的版本），但步骤更加复杂。

相关文档

从零开始设置 Windows 环境下的工具链（传统 GNU Make）

注解： ESP-IDF V4.0 及之后版本的默认构建系统为 CMake。本文档主要针对之前基于 GNU Make 的传统构建系统。请注意，未来，我们可能不会继续支持基于 GNU Make 的构建系统。

手动安装所有工具能更好地控制整个安装流程，同时也方便高阶用户进行自定义安装。此外，经验不足的用户还可以参考预编译环境 中的步骤进行准备。

使用预编译环境对工具链进行快速标准设置，请参照 *Windows 平台工具链的标准设置（传统 GNU Make）* 。

从零开始配置工具链和环境

本流程包括：首先，安装 MSYS2；其次，安装 ESP-IDF 所需的 Python 工具包；最后，下载并安装 Xtensa 工具链。

- 请前往 MSYS2 安装器页面，并下载 `msys2-i686-xxxxxxx.exe` 安装器（我们仅支持 32-bit MSYS 环境，32 位和 64 位 Window 均可使用）。截止至本文最新更新之时，最新版安装器为 `msys2-i686-20161025.exe`。
- 完成所有安装步骤。**最后一步时，请不要勾选 “Run MSYS2 32-bit now”**。
- 安装完成后，请从 “开始” 菜单中找到 “MSYS2 MinGW 32-bit”，运行 “命令提示符” 窗口。
  - 为什么要特别打开这个终端窗口？MSYS2 会对不同环境进行区分。默认的 “MSYS” 环境与 Cygwin 相仿，会为 Windows 系统的 API 调用增加一个转换层。但为了使用支持 COM 端口的原生 Python，我们需要准备好 “MinGW” 环境。
- GitHub 上的 ESP-IDF 仓库的 *tools* 目录下可以找到名为 `windows_install_prerequisites.sh` 的脚本。如果您还没有本地 ESP-IDF 副本，也可以从以下地址进行下载（请下载 raw 格式）：tools/windows/windows_install_prerequisites.sh，并将其保存至您的电脑。
- 请在 MSYS2 终端窗口中指定该脚本的保存路径。注意，路径格式与 Window 路径相同，但需使用正斜杠 (/) 而不是反斜杠 (\)。例 `C:/Users/myuser/Downloads/windows_install_prerequisites.sh`。当然，您也可以直接打开查看该脚本的内容。
- `windows_install_prerequisites.sh` 脚本将帮您下载并安装支持 ESP-IDF 的软件包和 ESP32 工具链。

疑难解答

- MSYS 可能在脚本运行过程中进行自动升级，导致无法使用。此时，您会看到以下错误信息:

```
*** fatal error - cygheap base mismatch detected - 0x612E5408/0x612E4408
```

这个问题很大可能是由于 cygwin DLL 版本不兼容。

这种情况下，请完全关闭终端窗口（相当于终止所有进程），并重新打开一个窗口。然后，请在新窗口中重新运行 `windows_install_prerequisites.sh` （小技巧：您可以使用“向上箭头”找到之前运行的命令）。此时，更新流程将重启。

- 注意，MSYS2 是一个“滚动”发行版，因此安装脚本可能会为您安装比“预编译环境”中更新的软件包。因此，如果遇到与 MSYS2 安装包有关的错误，请前往 MSYS2-packages 问题列表 页面寻找答案。如果未找到所需答案，请 提交一个 IDF Issue。

中国地区的 MSYS2 镜像

中国地区有一些（非官方）的 MSYS2 镜像，这可以大幅提高中国地区的下载速度。

如需添加这些镜像，请在运行安装脚本前修改以下两个 MSYS2 镜像列表文件。镜像文件的保存路径为 `/etc/pacman.d`，比如 `c:\msys2\etc\pacman.d`。

请在 `mirrorlist.mingw32` 最上方增加如下语句:

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

请在 `mirrorlist.msys` 最上方增加如下语句:

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

HTTP 代理

您可以在运行“设置脚本”之前，在终端中设置 `http_proxy` 变量，从而允许使用 HTTP 代理下载 MSYS 和 PIP:

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

或者修改证书:

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

如需始终用代理使用 MSYS，请在 MSYS 目录中增加 `/etc/profile`。

其他设置：下载工具链

如果您已经安装了 MSYS2，或者想要以不同的方式执行操作，您可以在此处下载工具链：

https://dl.espressif.com/dl/xtensa-esp32-elf-gcc8_2_0-esp-2020r2-win32.zip

---

注解： 如果您已完成了从零开始配置工具链和环境 中介绍的所有步骤，则已经拥有了工具链，这里无需重复下载。

重要： 仅拥有工具链 尚无法让您在 Windows 中使用 ESP-IDF。除此之外，您还至少需要安装 GNU make、bash 和 sed。上述环境已经包括这些配置，此外还有一个主机编译器（这样才能使用 menuconfig）。

后续步骤

继续设置开发环境，请前往第二步：获取 *ESP-IDF* 章节。

更新环境

当 IDF 有更新时，有时需要安装新的工具链或为 Windows MSYS2 环境添加新的系统要求。

此时，您无需重新配置环境，仅需更新现有 Windows 环境和工具链即可。

- 将 IDF 更新至您希望的版本。
- 请运行 IDF 仓库中的 `tools/windows/windows_install_prerequisites.sh` 脚本。该脚本将帮您安装所有新的软件包，并下载更新工具链至最新版本。

注意，该脚本在更新 MSYS2 时也会遇到疑难解答 中介绍的问题。

如需要同时支持多个 IDF 版本，您可以在不同的目录下配置独立的 MSYS2 环境。或者，您还可以下载多个工具链，并将其解压缩到不同的目录下，然后使用 PATH 环境变量指定默认工具链。

Linux 平台工具链的标准设置（传统 GNU Make）

[English]

注解： ESP-IDF V4.0 及之后版本的默认构建系统为 CMake。本文档主要针对之前基于 GNU Make 的传统构建系统。请注意，未来，我们可能不会继续支持基于 GNU Make 的构建系统。

安装准备

编译 ESP-IDF 需要以下软件包：

- CentOS 7:

```
sudo yum install gcc git wget make flex bison gperf python python2-cryptography
```

- Ubuntu and Debian:

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

- Arch:

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

注解： 一些旧的（2014 年之前）Linux 发行版中使用的 `pyserial` 版本可能是 2.x，ESP-IDF 并不支持。这种情况下，请按照*第四步：安装 Python 软件包* 章节的介绍，使用 `pip` 安装软件包。

工具链设置

Linux 版的 ESP32 工具链可以从 Espressif 的网站下载：

- 64 位 Linux：

  https://dl.espressif.com/dl/xtensa-esp32-elf-gcc8_2_0-esp-2020r2-linux-amd64.tar.gz

- 32 位 Linux：

  https://dl.espressif.com/dl/xtensa-esp32-elf-gcc8_2_0-esp-2020r2-linux-i686.tar.gz

1. 下载压缩文件之后，解压到 `~/esp` 目录中：

   - 64 位 Linux：

```
mkdir -p ~/esp
cd ~/esp
tar -xzf ~/Downloads/xtensa-esp32-elf-gcc8_2_0-esp-2020r2-linux-amd64.tar.gz
```

   - 32 位 Linux：

```
mkdir -p ~/esp
cd ~/esp
tar -xzf ~/Downloads/xtensa-esp32-elf-gcc8_2_0-esp-2020r2-linux-i686.tar.gz
```

2. 工具链将被解压到 ~/esp/xtensa-esp32-elf/ 路径下。

   为了正常使用工具链，您必须更新 ~/.profile 文件中的 PATH 环境变量。此外，您还可以在 ~/.profile 文件中增加以下代码，这样一来，所有终端窗口均可以使用 xtensa-esp32-elf:

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

   或者，您可以为上述命令创建一个别名。这样，您只有在需要时才可以使用工具链。如需设置别名，请将以下代码增加至您的 🄴/ .profile 文件中:

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

   这样，您可以在终端输入 get_esp32 命令将工具链添加至您的 PATH，从而使用工具链。

   注解: 如果您已将 /bin/bash 设置为登录 shell，且同时存在 .bash_profile 和 .profile，则请更新 .bash_profile 。在 CentOS 环境下，alias 需要添加到 .bashrc 文件中。

   ---

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

   ```
   printenv PATH
   ```

   检查字符串的开头是否包含类似的工具链路径:

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

   这里，您需要将 /home/user-name 替换为您的安装主目录。

权限问题 /dev/ttyUSB0

某些 Linux 版本可能在烧写 ESP32 时会出现 Failed to open port /dev/ttyUSB0 错误消息，这可以通过*将当前用户添加到 dialout 组* 来解决。

后续步骤

继续设置开发环境，请前往*第二步：获取 ESP-IDF* 章节。

相关文档

从零开始设置 Linux 环境下的工具链（传统 GNU Make）

[English]

注解： ESP-IDF V4.0 及之后版本的默认构建系统为 CMake。本文档主要针对之前基于 GNU Make 的传统构建系统。请注意，未来，我们可能不会继续支持基于 GNU Make 的构建系统。

注解： 安装工具链的标准方法请见这里。请参考工具链自定义设置 章节，查看可能需要从头开始设置工具链的情况。

安装准备

编译 ESP-IDF 需要以下软件包：

- Ubuntu and Debian:

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

- Arch:

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

注解： 一些旧的（2014 年之前）Linux 发行版中使用的 `pyserial` 版本可能是 2.x，ESP-IDF 并不支持。这种情况下，请按照第四步：安装 *Python* 软件包 章节的介绍，使用 `pip` 安装软件包。

从源代码编译工具链

- 安装依赖项：
  - CentOS 7:

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

- Ubuntu pre-16.04:

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

- Ubuntu 16.04 或以上

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

- Debian 9:

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

- Arch:

```
TODO
```

- 创建工作目录，并进入该目录:

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

- 下载并编译 crosstool-NG：

```
git clone https://github.com/espressif/crosstool-NG.git
cd crosstool-NG
git checkout esp-2020r2
git submodule update --init
./bootstrap && ./configure --enable-local && make
```

- 编译工具链:

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

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

后续步骤

继续设置开发环境，请前往第二步：获取 *ESP-IDF* 章节。

Mac OS 平台工具链的标准设置（传统 GNU Make）

[English]

注解： ESP-IDF V4.0 及之后版本的默认构建系统为 CMake。本文档主要针对之前基于 GNU Make 的传统构建系统。请注意，未来，我们可能不会继续支持基于 GNU Make 的构建系统。

安装准备

- 安装 pip:

```
sudo easy_install pip
```

注解： pip 稍后将用于安装*所需 Python 包*。

工具链设置

Mac OS 版本的 ESP32 工具链可以从以下地址下载：

https://dl.espressif.com/dl/xtensa-esp32-elf-gcc8_2_0-esp-2020r2-macos.tar.gz

下载压缩文件之后，解压到 ~/esp 目录中:

```
mkdir -p ~/esp
cd ~/esp
tar -xzf ~/Downloads/xtensa-esp32-elf-gcc8_2_0-esp-2020r2-macos.tar.gz
```

工具链将被解压到 ~/esp/xtensa-esp32-elf/ 路径下。

为了正常使用工具链，您必须更新 ~/.profile 文件中的 PATH 环境变量。此外，您还可以在 ~/.profile 文件中增加以下代码，这样一来，所有终端窗口均可以使用 xtensa-esp32-elf:

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

或者，您可以为上述命令创建一个别名。这样，您只有在需要时才可以使用工具链。如需设置别名，请将以下代码增加至您的 F/ .profile 文件中:

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

这样，您可以在终端输入 get_esp32 命令将工具链添加至您的 PATH，从而使用工具链。

后续步骤

继续设置开发环境，请前往第二步：获取 *ESP-IDF* 章节。

相关文档

从零开始设置 Mac OS 环境下的工具链（传统 GNU Make）

[English]

注解： ESP-IDF V4.0 及之后版本的默认构建系统为 CMake。本文档主要针对之前基于 GNU Make 的传统构建系统。请注意，未来，我们可能不会继续支持基于 GNU Make 的构建系统。

注解： 安装工具链的标准方法请见这里。请参考工具链自定义设置 章节，查看可能需要从头开始设置工具链的情况。

安装准备

- 安装 pip:

```
sudo easy_install pip
```

注解： pip 稍后将用于安装必要的 *Python* 软件包。

从源代码编译工具链

- 安装依赖项：
  - 安装 MacPorts 或 homebrew 安装包管理器。MacPorts 需要完整的 XCode 软件，而 homebrew 只需要安装 XCode 命令行工具即可。
  - 对于 MacPorts:

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

  - 对于 homebrew:

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

- 创建一个文件系统镜像（区分大小写）：

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

- 挂载:

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

- 创建指向您工作目录的符号链接:

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

- 前往新创建的目录

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

- 下载并编译 crosstool-NG：

```
git clone https://github.com/espressif/crosstool-NG.git
cd crosstool-NG
git checkout esp-2020r2
git submodule update --init
./bootstrap && ./configure --enable-local && make
```

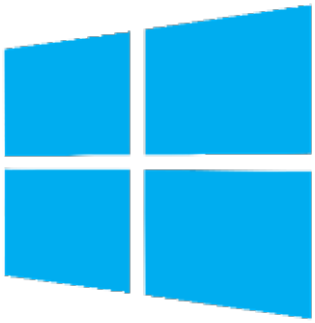
- 编译工具链:

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

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

后续步骤

继续设置开发环境，请前往第二步：获取 *ESP-IDF* 章节。

		
Windows	Linux	macOS

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

此外，您也可以根据自身经验和实际需求，对环境进行个性化设置，而非使用预制工具链。此时，请前往*工具链自定义设置（传统 GNU Make 系统）* 章节获取更多信息。

第二步：获取 ESP-IDF

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

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

打开终端，后运行以下命令：

```
cd ~/esp
git clone -b v4.1 --recursive https://github.com/espressif/esp-idf.git
```

ESP-IDF 将下载至 `~/esp/esp-idf`。

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

注解： `git clone` 命令的 `-b v4.1` 选项告诉 git 从 ESP-IDF 仓库中克隆与此版本的文档对应的分支。

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

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

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

第三步：设置环境变量

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

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

第四步：安装 Python 软件包

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

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

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

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

第五步：开始创建工程

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

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

Linux 和 MacOS 操作系统

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

Windows 操作系统

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

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

重要： ESP-IDF 编译系统不支持带有空格的路径。

第六步：连接设备

现在，请将您的 ESP32 开发板连接到 PC，并查看开发板使用的串口。

通常，串口在不同操作系统下显示的名称有所不同：

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

有关如何查看串口名称的详细信息，请见*与 ESP32 创建串口连接（传统 GNU Make 系统）*。

注解： 请记住串口名，您会在下面的步骤中用到。

第七步：配置

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

Linux 和 MacOS 操作系统

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

Windows 操作系统

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

如果之前的步骤都正确，则会显示下面的菜单：

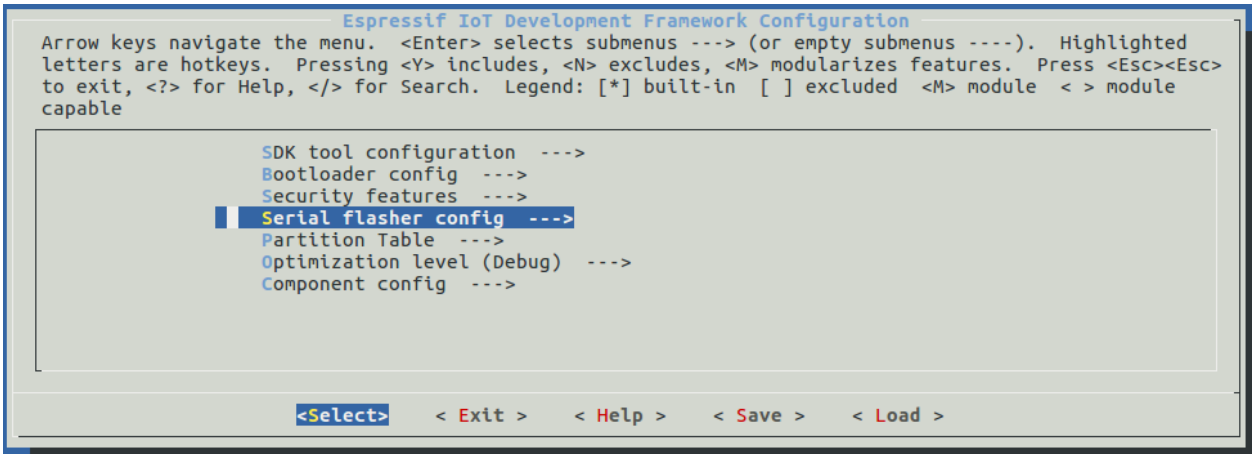


图 32: 工程配置—主窗口

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

`menuconfig` 工具的常见操作见下。

- 上下箭头：移动
- 回车：进入子菜单
- `ESC` 键：返回上级菜单或退出
- 英文问号：调出帮助菜单（退出帮助菜单，请按回车键）。
- 空格 ``或 ``Y` 键：选择 `[*]` 配置选项；`N` 键：禁用 `[*]` 配置选项
- 英文问号（查询配置选项）：调出有关该选项的帮助菜单
- `/` 键：寻找配置工程

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

第八步：编译和烧录

请使用以下命令，编译烧录工程：

```
make flash
```

运行以上命令可以编译应用程序和所有 ESP-IDF 组件，接着生成 bootloader、分区表和应用程序二进制文件。接着，这些二进制文件将被烧录至 ESP32 开发板。

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

```
esptool.py v2.0-beta2
Flashing binaries to serial port /dev/ttyUSB0 (app at offset 0x10000)...
esptool.py v2.0-beta2
Connecting........___
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 921600
Changed.
Attaching SPI flash...
Configuring flash size...
Auto-detected Flash size: 4MB
Flash params set to 0x0220
Compressed 11616 bytes to 6695...
Wrote 11616 bytes (6695 compressed) at 0x00001000 in 0.1 seconds (effective 920.5 kbit/
↪s)...
Hash of data verified.
Compressed 408096 bytes to 171625...
Wrote 408096 bytes (171625 compressed) at 0x00010000 in 3.9 seconds (effective 847.3␣
↪kbit/s)...
Hash of data verified.
Compressed 3072 bytes to 82...
Wrote 3072 bytes (82 compressed) at 0x00008000 in 0.0 seconds (effective 8297.4 kbit/s)..
↪.
Hash of data verified.

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

如果您希望使用 Eclipse IDE，而非 `make` 编译系统，请参考*Eclipse guide*。

第九步：监视器

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

运行该命令后，*IDF 监视器* 应用程序将启动:

```
$ make monitor
MONITOR
```

(下页继续)

(续上页)

```
--- idf_monitor on /dev/ttyUSB0 115200 ---
--- Quit: Ctrl+] | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---
ets Jun  8 2016 00:22:57

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
ets Jun  8 2016 00:22:57
...
```

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

```
...
Hello world!
Restarting in 10 seconds...
I (211) cpu_start: Starting scheduler on APP CPU.
Restarting in 9 seconds...
Restarting in 8 seconds...
Restarting in 7 seconds...
```

您可使用快捷键 `Ctrl+]`，退出 IDF 监视器。

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

```
e???)(Xn@?y.!??(?PW+)??Hn9a￣/9?!?t5??P?~?k??e?ea?5?jA
~zY??Y(1?,1 ?? e???)(Xn@?y.!Dr?zY(? jpi?|?+z5Ymvp
```

此时，请您：

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

注解： 您也可以运行以下命令，一次性执行构建、烧录和监视过程:

```
make flash monitor
```

此外，请前往*IDF 监视器*，了解更多使用 IDF 监视器的快捷键和其他详情。

恭喜，您已完成 ESP32 的入门学习！

现在，您可以尝试一些其他 examples，或者直接开发自己的应用程序。

环境变量

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

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

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

更新 ESP-IDF

乐鑫会不时推出更新版本的 ESP-IDF，修复 bug 或推出新的特性。因此，您在使用时，也应注意更新您本地的版本。最简单的方法是：直接删除您本地的 esp-idf 文件夹，然后按照第二步：获取 *ESP-IDF* 中的指示，重新完成克隆。

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

此外，您可以仅更新变更部分。具体方式，请前往更新 章节查看。

相关文档

在用户配置文件中添加 IDF_PATH（传统 GNU Make）

[English]

注解： ESP-IDF V4.0 及之后版本的默认构建系统为 CMake。本文档主要针对之前基于 GNU Make 的传统构建系统。请注意，未来，我们可能不会继续支持基于 GNU Make 的构建系统。

为了在系统多次启动时，保留 IDF_PATH 环境变量的设置，请按照以下说明将该变量的设置增加至用户配置文件。

Windows 操作系统

用户配置文件脚本保存在 C:/msys32/etc/profile.d/ 路径下，每次打开 MSYS2 窗口时均会执行。

1. 在 C:/msys32/etc/profile.d/ 目录下创建一个新的脚本文件，并将其命名为 export_idf_path.sh。
2. 指定您的 ESP-IDF 保存路径，这通常与您的系统配置有关，比如 C:\msys32\home\user-name\esp\esp-idf。
3. 在脚本文件中加入下方 export 命令行，例:

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

   注意请将反斜杠替换为 Windows 操作系统路径要求的正斜杠。

4. 保存脚本文件。
5. 关闭 MSYS2 窗口，并重新打开。运行以下命令，检查 IDF_PATH 是否成功设置:

```
printenv IDF_PATH
```

   如果设置成功，则此时将打印之前在脚本中输入的路径。

   如果您不希望永久设置 IDF_PATH 变量，也可以在每次打开 MSYS2 窗口时进行手动设置:

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

如果您是在安装 ESP32 软件的过程中从第三步：设置环境变量 章节跳转至此，请返回第五步：开始创建工程 章节。

Linux 和 MacOS 操作系统

在 ~/.profile 文件中增加以下命令，设置 IDF_PATH 变量:

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

退出并重新登陆，检查设置是否生效。

注解: 如果您已将 /bin/bash 设置为登录 shell，且同时存在 .bash_profile 和 .profile，则请更新 .bash_profile 。

运行以下命令，检查 IDF_PATH 是否成功设置:

```
printenv IDF_PATH
```

如果设置成功，则此时将打印之前在 ~/.profile 文件中输入（或手动设置）的路径。

如果您不希望永久设置 IDF_PATH 变量，也可以在每次重新打开终端窗口时进行手动设置:

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

如果您是在安装 ESP32 软件的过程中从第三步：设置环境变量 章节跳转至此，请返回第五步：开始创建工程 章节。

与 ESP32 创建串口连接（传统 GNU Make 系统）

[English]

注解： ESP-IDF V4.0 及之后版本的默认构建系统为 CMake。本文档主要针对之前基于 GNU Make 的传统构建系统。请注意，未来，我们可能不会继续支持基于 GNU Make 的构建系统。

本章节主要介绍如何创建 ESP32 和 PC 之间的串口连接。

连接 ESP32 和 PC

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

以下是乐鑫 ESP32 等开发板驱动程序的链接:

开发板	USB 驱动	备注
ESP32-DevKitC	CP210x	
ESP32-LyraT	CP210x	
ESP32-LyraTD-MSC	CP210x	
ESP32-PICO-KIT	CP210x	
ESP-WROVER-KIT	FTDI	
ESP32 Demo 板	FTDI	
ESP-Prog	FTDI	编程板 (w/o ESP32)
ESP32-MeshKit-Sense	n/a	搭配 ESP-Prog 使用
ESP32-Sense Kit	n/a	搭配 ESP-Prog 使用

- CP210x: CP210x USB to UART Bridge VCP Drivers
- FTDI: FTDI Virtual COM Port Drivers

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

查看端口（Windows 用户）

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

以下为 ESP32 DevKitC 和 ESP32 WROVER KIT 串口：

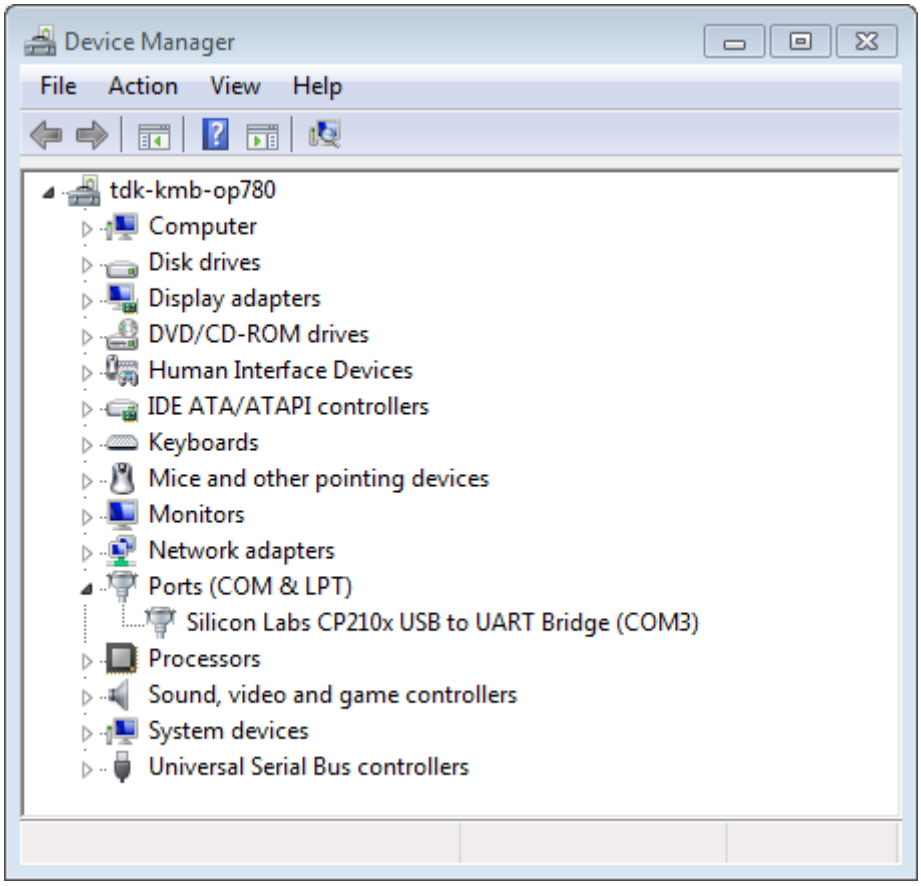


图 33: Windows 设备管理器中 ESP32-DevKitC 的 USB 至 UART 桥

查看端口（Linux 和 MacOS 用户）

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

Linux

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

MacOS

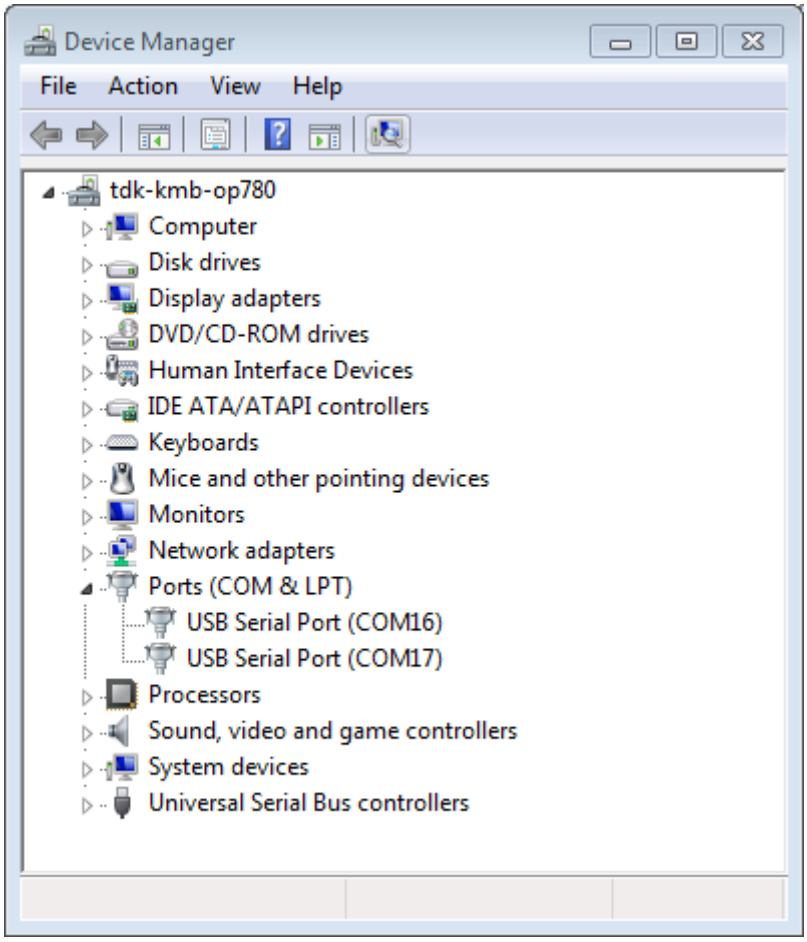


图 34: Windows 设备管理器中 ESP-WROVER-KIT 的两个 USB 串行端口

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

将用户增加至 Linux 的 dialout 组

当前登录用户应当拥有通过 USB 对串口进行读写的权限。在多数 Linux 版本中，您都可以通过以下命令，将用户添加到 dialout 组，来获取读写权限:

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

在 Arch Linux 中，需要通过以下命令将用户添加到 uucp 组中:

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

请重新登录，确保串口读写权限可以生效。

确认串口连接

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

运行终端，配置串口：波特率 = 115200，数据位 = 8，停止位 = 1，奇偶校验 = N。在 Windows 和 Linux 中配置串口和通信参数（如 115200-8-1-N）的截图如下。注意，这里一定要选择在上述步骤中确认的串口进行配置。

然后，请在终端打开串口，查看 ESP32 是否有任何打印，具体打印内容取决于加载至 ESP32 的程序。ESP32 打印示例

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

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

rst:0x7 (TG0WDT_SYS_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0x00
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0008,len:8
load:0x3fff0010,len:3464
load:0x40078000,len:7828
load:0x40080000,len:252
entry 0x40080034
I (44) boot: ESP-IDF v2.0-rc1-401-gf9fba35 2nd stage bootloader
```

(下页继续)

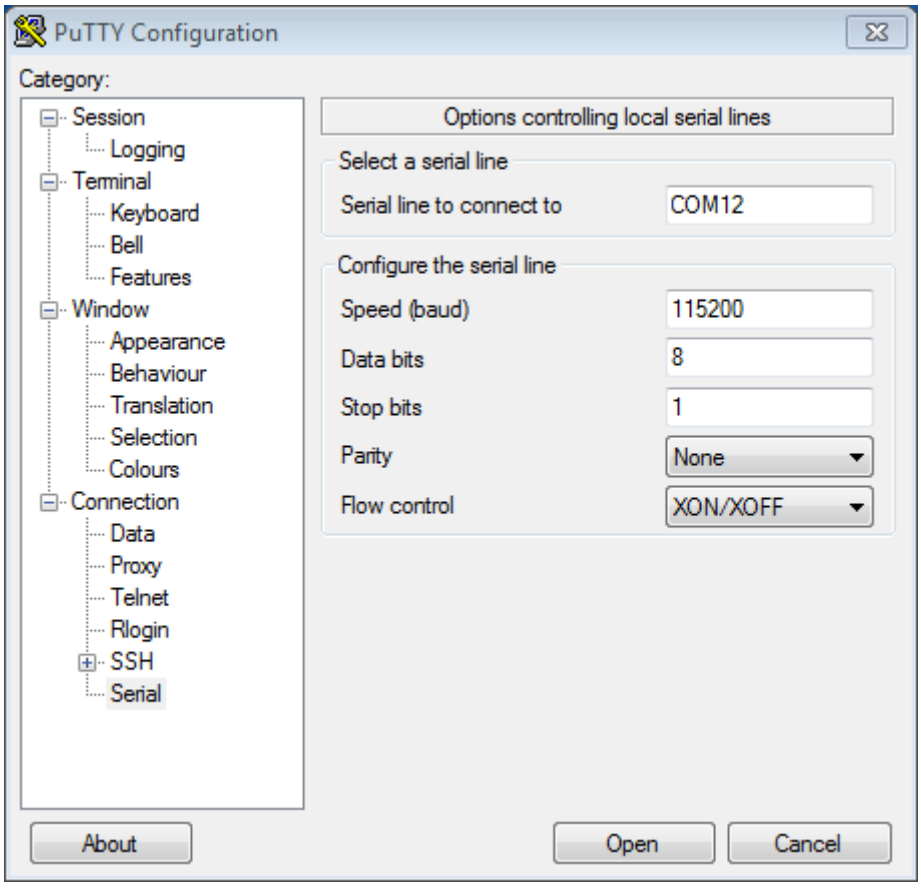


图 35: 在 Windows 操作系统中使用 PuTTY 设置串口通信参数

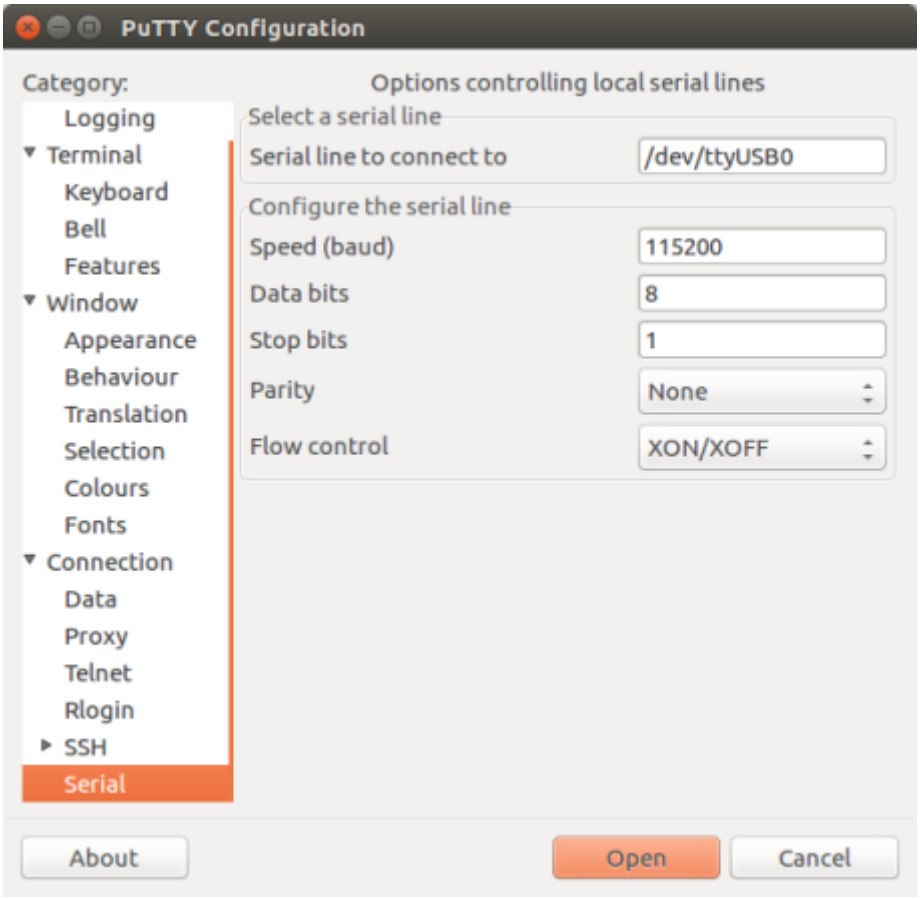


图 36: 在 Linux 操作系统中使用 PuTTY 设置串口通信参数

(续上页)

```
I (45) boot: compile time 18:48:10

...
```

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

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

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

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

通过 make 指令创建和烧录项目（传统 GNU Make）

[English]

注解： ESP-IDF V4.0 及之后版本的默认构建系统为 CMake。本文档主要针对之前基于 GNU Make 的传统构建系统。请注意，未来，我们可能不会继续支持基于 GNU Make 的构建系统。

寻找项目

除了 esp-idf-template 项目模版外，ESP-IDF 还在 GitHub 仓库中的 examples 目录下提供多个示例项目。

请找到并进入您想要的项目，开始配置、构建该项目。

配置项目

```
make menuconfig
```

编译项目

```
make all
```

该命令将配置 app 和 bootloader，并根据配置生成分区表。

烧录项目

`make all` 完成后将打印一行命令，提示您如何使用 esptool.py 烧录芯片。不过，您可以使用以下命令进行烧录:

```
make flash
```

该命令可以将整个项目（包括 app、bootloader 和分区表）烧录至新芯片。此外，如果分区表中存在 ota_data，则该命令还会同时将初始 ota_data 烧录至芯片。该命令允许您直接从 factory 分区中运行新加载的 app（如果 factory 分区不存在，则从第一个 OTA 分区开始运行）。您可以使用 `make menuconfig` 配置串口的烧录。

在运行 `make flash` 之前无需单独运行 `make all`。`make flash` 命令本身就可以自动构建所需的文件。

仅编译和烧录应用程序

在首次烧录完成后，您可以只希望烧录并编译应用程序，而不再包括 bootloader 和分区表。这种情况可以使用以下命令:

- `make app` – 仅编译应用程序
- `make app-flash` – 仅烧录应用程序

`make app-flash` – 在有需要时自动重新编译应用程序

如果没有变化，每次都重新烧录、编译 bootloader 和分区表也没有坏处。

分区表

项目编译完成后，"build"目录下将出现一个名为"my_app.bin"（例）的二进制文件。这是一个可由 bootloader 加载的 ESP32 映像文件。

ESP32 的一块 flash 可以包含多个应用程序，以及多种数据（校准数据、文件系统、参数存储等）。因此，在向 flash 烧录分区表时，通常保留 0x8000 的偏移量。

分区表中的每个条目都有名称（标签）、类型（app、数据或其他）、子类型及在 flash 中的偏移量。

使用分区表最简便的方法是：运行 `make menuconfig`，选择一个预定义分区表:

- "Single factory app, no OTA"
- "Factory app, two OTA definitions"

上述两种情况中，factory app 在 flash 中的烧录偏移地址均为 0x10000。运行 `make partition_table` 命令可以打印分区表摘要。

更多有关分区表 及自定义分区表的内容，请见相关文档。

Eclipse IDE 的创建和烧录指南（传统 GNU Make）

[English]

注解： ESP-IDF V4.0 及之后版本的默认构建系统为 CMake。本文档主要针对之前基于 GNU Make 的传统构建系统。请注意，未来，我们可能不会继续支持基于 GNU Make 的构建系统。

安装 Eclipse IDE

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

- 首先，请在您的平台上安装相应的 ESP-IDF，具体步骤请按照您的操作系统（Windows、OS X 和 Linux）对应的指示进行安装。
- 我们建议，您可以首先使用命令行创建一个项目，大致熟悉项目的创建流程。此外，您还需要使用命令行 (`make menuconfig`) 对您的 ESP-IDF 项目进行配置。目前，Eclipse 还无法配置 ESP-IDF 项目。
- 从 eclipse.org 下载对应操作系统的 Eclipse 安装器。
- 运行 Eclipse Installer，选择 **“Eclipse for C/C++ Development”**（有的版本也可能显示为 CDT）。

配置 Eclipse

打开安装好的 Eclipse IDE，并按照以下步骤进行操作：

导入新项目

- Eclipse 需使用 ESP-IDF 的 Makefile 功能。因此，在使用 Eclipse 前，您需要先创建一个 ESP-IDF 项目。在创建 ESP-IDF 项目时，您可以使用 GitHub 中的 idf-template 项目模版，或从 ESP-IDF 子目录中复制一个 example。
- 运行 Eclipse，选择 **“File”** -> **“Import…”**。
- 在弹出的对话框中选择 **“C/C++”** -> **“Existing Code as Makefile Project”**，点击 **“Next”**。
- 进入下一界面，在 **“Existing Code Location”** 处输入您的 IDF 项目路径。注意，这里应输入 ESP-IDF 项目的路径，而非 ESP-IDF 本身的路径（这个稍后再填）。此外，您指定的目标路径中应包含名为 `Makefile` （项目 Makefile）的文件。
- 本界面中，找到 **“Toolchain for Indexer Settings”**，选择 **“Cross GCC”**，最后点击 **“Finish”**。

项目属性

- 新项目将出现在 **"Project Explorer"** 下。右键选择该项目，并在菜单中选择 **"Properties"**。
- 点击 **"C/C++ Build"**下的 **"Environment"**属性页，选择 **"Add…"**，并在对应位置输入 `BATCH_BUILD` 和 `1`。
- 再次点击 **"Add…"**，并在 `IDF_PATH` 处输入 ESP-IDF 所在的完整安装路径。Windows 用户可以从 Windows 浏览器复制 `IDF_PATH`。
- 编辑 `PATH` 环境变量。保留当前值，并将该路径增加至 Xtensa 工具链安装路径后，作为 IDF 设置的一部分（如已增加至 PATH，则可忽略）。工具链的常见路径为 `/home/user-name/esp/xtensa-esp32-elf/bin`（例）。注意，您需要在补充的路径前增加一个冒号（*:*）。Windows 用户需要将 `PATH` 环境变量前增加 `C:\msys32\mingw32\bin;C:\msys32\opt\xtensa-esp32-elf\bin;C:\msys32\usr\bin`。注意，请根据 msys32 的具体安装路径，修改 `PATH` 环境变量。
- MacOS 用户需要增加一个 `PYTHONPATH` 环境变量，并将其设置为 `/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/site-packages`。保证系统中预先安装的 Python （需安装 pyserial 模块）可以覆盖 Eclipse 内置的任何 Python。

特别说明：如果您的 IDF_PATH 或项目不在 `C:\msys32\home` 之下，则应使用自定义编译命令：`python ${IDF_PATH}/tools/windows/eclipse_make.py` （请注意，这种方法可能导致编译时间显著增加。）

- 前往 **"C/C++ General"** -> **"Preprocessor Include Paths"** 属性页面。点击 **"Providers"** 选项卡。
  - 从 **"Providers"** 列表中选择 **" CDT Cross GCC Built-in Compiler Settings"** ，将 **"Command to get compiler specs"** 修改为 `xtensa-esp32-elf-gcc ${FLAGS} -std=c++11 -E -P -v -dD "${INPUTS}"`。
  - 从 **"Providers"**列表中选择 **"CDT GCC Build Output Parser"**，将 **"Compiler command pattern"** 修改为 `xtensa-esp32-elf-(gcc|g\+\+|c\+\+|cc|cpp|clang)`。
- 前往 **"C/C++ General"** -> **"Indexer"** 属性页面。
  - 勾选 **"Enable project specific settings"**，启用本页面上的其他设置。
  - 取消勾选 **"Allow heuristic resolution of includes"**。勾选该选项有时会导致 Eclipse 无法找到正确的头文件目录。
- 前往 **"C/C++ Build"** -> **"Behavior"** 属性页面。
  - 勾选 **"Enable parallel build"**，启用“并行编译”配置。

在 Eclipse 中构建项目

在首次创建项目前，Eclipse IDE 可能会显示大量有关未定义值的错误和警告，这是因为 ESP-IDF 在编译过程中会自动生成一些源文件。因此，这些错误和警告均会在项目编译完成后消失。

- 点击 **"OK"**，关闭 Eclipse IDE 中的 **"Properties"** 对话框。

- 在 Eclipse IDE 界面外，打开命令提示符窗口。进入项目目录，运行 `make menuconfig` 设置项目的 ESP-IDF 配置。现阶段，您还无法在 Eclipse 中完成本操作。
- 回到 Eclipse IDE 界面，选择 **“Project”** -> **“Build”** 开始编译项目。

小提示：如果您的项目不是通过 Eclipse 创建的，则在选择 **“Project”** -> **“Build”** 前还需进行 **“Project”** -> **“Clean”** 操作。如此操作后，Eclipse 才能查看所有源文件的编译器参数，并借此确定头文件包含路径。

在 Eclipse 中烧录项目

您可以将 `make flash` 目标集成在 Eclipse 项目中，这样就可以通过 Eclipse UI 调用 `esptool.py` 进行烧录：

- 打开 **“Project Explorer”**，并右击您的项目（请注意右击项目本身，而非项目下的子文件，否则 Eclipse 可能会找到错误的 `Makefile`）。
- 从菜单中选择 **“Build Targets”** -> **“Create…”**。
- 目标名称输入 “flash”，其他选项使用默认值。
- 这时，您可以选择 **“Project”** -> **“Build Target”** -> **“Build”**（快捷键：Shift + F9），创建自定义烧录目标，用于编译、烧录项目。

注意，您仍需要通过 `make menuconfig` 设置串行端口和其他烧录选项。`make menuconfig` 仍需通过命令行操作（具体请见对应操作系统的指南）。

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

工具链自定义设置（传统 GNU Make 系统）

除了从乐鑫官网（请见第一步：设置工具链）下载二进制工具链外，您还可以自行编译工具链。

注解： ESP-IDF V4.0 及之后版本的默认构建系统为 CMake。本文档主要针对之前基于 GNU Make 的传统构建系统。请注意，未来，我们可能不会继续支持基于 GNU Make 的构建系统。

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

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

如需自行编译工具链，请查看以下文档：

CHAPTER 2

API 参考

[English]

2.1 蓝牙 API

[English]

2.1.1 Controller && VHCI

Overview

Instructions

Application Example

Check bluetooth/bluedroid/hci folder in ESP-IDF examples, which contains the following application:

- This is a BLE advertising demo with virtual HCI interface. Send Reset/ADV_PARAM/ADV_DATA/ADV_ENABLE HCI command for BLE advertising - bluetooth/bluedroid/hci/controller_vhci_ble_adv.

API Reference

Header File

- bt/include/esp_bt.h

Functions

esp_err_t **esp_ble_tx_power_set**(*esp_ble_power_type_t power_type*, *esp_power_level_t power_level*)

Set BLE TX power Connection Tx power should only be set after connection created.

Return ESP_OK - success, other - failed

Parameters

- `power_type`: : The type of which tx power, could set Advertising/Connection/Default and etc
- `power_level`: Power level(index) corresponding to absolute value(dbm)

esp_power_level_t **esp_ble_tx_power_get**(*esp_ble_power_type_t power_type*)

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

Return >= 0 - Power level, < 0 - Invalid

Parameters

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

esp_err_t **esp_bredr_tx_power_set**(*esp_power_level_t min_power_level*, *esp_power_level_t max_power_level*)

Set BR/EDR TX power BR/EDR power control will use the power in range of minimum value and maximum value. The power level will effect the global BR/EDR TX power, such inquire, page, connection and so on. Please call the function after esp_bt_controller_enable and before any function which cause RF do TX. So you can call the function before doing discovery, profile init and so on. For example, if you want BR/EDR use the new TX power to do inquire, you should call this function before inquire. Another word, If call this function when BR/EDR is in inquire(ING), please do inquire again after call this function. Default minimum power level is ESP_PWR_LVL_N0, and maximum power level is ESP_PWR_LVL_P3.

Return ESP_OK - success, other - failed

Parameters

- `min_power_level`: The minimum power level

- `max_power_level`: The maximum power level

esp_err_t **esp_bredr_tx_power_get**(*esp_power_level_t* **min_power_level*, *esp_power_level_t* **max_power_level*)

Get BR/EDR TX power If the argument is not NULL, then store the corresponding value.

Return ESP_OK - success, other - failed

Parameters

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

esp_err_t **esp_bredr_sco_datapath_set**(*esp_sco_data_path_t data_path*)

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

Return ESP_OK - success, other - failed

Parameters

- `data_path`: SCO data path

esp_err_t **esp_bt_controller_init**(*esp_bt_controller_config_t* **cfg*)

Initialize BT controller to allocate task and other resource. This function should be called only once, before any other BT functions are called.

Return ESP_OK - success, other - failed

Parameters

- `cfg`: Initial configuration of BT controller. Different from previous version, there' s a mode and some connection configuration in "cfg" to configure controller work mode and allocate the resource which is needed.

esp_err_t **esp_bt_controller_deinit**(void)

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

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

Return ESP_OK - success, other - failed

esp_err_t **esp_bt_controller_enable**(*esp_bt_mode_t mode*)

Enable BT controller. Due to a known issue, you cannot call esp_bt_controller_enable() a second time to change the controller mode dynamically. To change controller mode, call esp_bt_controller_disable() and then call esp_bt_controller_enable() with the new mode.

Return ESP_OK - success, other - failed

Parameters

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

esp_err_t **esp_bt_controller_disable**(void)

Disable BT controller.

Return ESP_OK - success, other - failed

esp_bt_controller_status_t **esp_bt_controller_get_status**(void)

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

Return status value

bool **esp_vhci_host_check_send_available**(void)

esp_vhci_host_check_send_available used for check actively if the host can send packet to controller or not.

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

void **esp_vhci_host_send_packet**(uint8_t **data*, uint16_t *len*)

esp_vhci_host_send_packet host send packet to controller

Should not call this function from within a critical section or when the scheduler is suspended.

Parameters

- `data`: the packet point
- `len`: the packet length

esp_err_t **esp_vhci_host_register_callback**(const *esp_vhci_host_callback_t* **callback*)

esp_vhci_host_register_callback register the vhci reference callback struct defined by vhci_host_callback structure.

Return ESP_OK - success, ESP_FAIL - failed

Parameters

- `callback`: *esp_vhci_host_callback* type variable

esp_err_t **esp_bt_controller_mem_release**(*esp_bt_mode_t mode*)

esp_bt_controller_mem_release release the controller memory as per the mode

This function releases the BSS, data and other sections of the controller to heap. The total size is about 70k bytes.

esp_bt_controller_mem_release(mode) should be called only before esp_bt_controller_init() or after esp_bt_controller_deinit().

Note that once BT controller memory is released, the process cannot be reversed. It means you cannot use the bluetooth mode which you have released by this function.

If your firmware will later upgrade the Bluetooth controller mode (BLE -> BT Classic or disabled -> enabled) then do not call this function.

If the app calls esp_bt_controller_enable(ESP_BT_MODE_BLE) to use BLE only then it is safe to call esp_bt_controller_mem_release(ESP_BT_MODE_CLASSIC_BT) at initialization time to free unused BT Classic memory.

If the mode is ESP_BT_MODE_BTDM, then it may be useful to call API esp_bt_mem_release(ESP_BT_MODE_BTDM) instead, which internally calls esp_bt_controller_mem_release(ESP_BT_MODE_BTDM) and additionally releases the BSS and data consumed by the BT/BLE host stack to heap. For more details about usage please refer to the documentation of esp_bt_mem_release() function

Return ESP_OK - success, other - failed

Parameters

- `mode`: : the mode want to release memory

esp_err_t **esp_bt_mem_release**(*esp_bt_mode_t mode*)

esp_bt_mem_release release controller memory and BSS and data section of the BT/BLE host stack as per the mode

This function first releases controller memory by internally calling esp_bt_controller_mem_release(). Additionally, if the mode is set to ESP_BT_MODE_BTDM, it also releases the BSS and data consumed by the BT/BLE host stack to heap

Note that once BT memory is released, the process cannot be reversed. It means you cannot use the bluetooth mode which you have released by this function.

If your firmware will later upgrade the Bluetooth controller mode (BLE -> BT Classic or disabled -> enabled) then do not call this function.

If you never intend to use bluetooth in a current boot-up cycle, you can call esp_bt_mem_release(ESP_BT_MODE_BTDM) before esp_bt_controller_init or after esp_bt_controller_deinit.

For example, if a user only uses bluetooth for setting the WiFi configuration, and does not use bluetooth in the rest of the product operation" . In such cases, after receiving the WiFi configuration, you can disable/deinit bluetooth and release its memory. Below is the sequence of APIs to be called for such scenarios:

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

Note In case of NimBLE host, to release BSS and data memory to heap, the mode needs to be set to ESP_BT_MODE_BTDM as controller is dual mode.

Return ESP_OK - success, other - failed

Parameters

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

esp_err_t **esp_bt_sleep_enable**(void)

enable bluetooth to enter modem sleep

Note that this function shall not be invoked before esp_bt_controller_enable()

There are currently two options for bluetooth modem sleep, one is ORIG mode, and another is EVED Mode. EVED Mode is intended for BLE only.

For ORIG mode: Bluetooth modem sleep is enabled in controller start up by default if CONFIG_BTDM_MODEM_SLEEP is set and "ORIG mode" is selected. In ORIG modem sleep mode, bluetooth controller will switch off some components and pause to work every now and then, if there is no event to process; and wakeup according to the scheduled interval and resume the work. It can also wakeup earlier upon external request using function "esp_bt_controller_wakeup_request" .

Return

- ESP_OK : success
- other : failed

esp_err_t **esp_bt_sleep_disable**(void)

disable bluetooth modem sleep

Note that this function shall not be invoked before esp_bt_controller_enable()

If esp_bt_sleep_disable() is called, bluetooth controller will not be allowed to enter modem sleep;

If ORIG modem sleep mode is in use, if this function is called, bluetooth controller may not immediately wake up if it is dormant then. In this case, esp_bt_controller_wakeup_request() can be used to shorten the time for wakeup.

Return

- ESP_OK : success

- other : failed

bool **esp_bt_controller_is_sleeping**(void)

to check whether bluetooth controller is sleeping at the instant, if modem sleep is enabled

Note that this function shall not be invoked before esp_bt_controller_enable() This function is supposed to be used ORIG mode of modem sleep

Return true if in modem sleep state, false otherwise

void **esp_bt_controller_wakeup_request**(void)

request controller to wakeup from sleeping state during sleep mode

Note that this function shall not be invoked before esp_bt_controller_enable() Note that this function is supposed to be used ORIG mode of modem sleep Note that after this request, bluetooth controller may again enter sleep as long as the modem sleep is enabled

Profiling shows that it takes several milliseconds to wakeup from modem sleep after this request. Generally it takes longer if 32kHz XTAL is used than the main XTAL, due to the lower frequency of the former as the bluetooth low power clock source.

esp_err_t **esp_ble_scan_dupilcate_list_flush**(void)

Manually clear scan duplicate list.

Note that scan duplicate list will be automatically cleared when the maximum amount of device in the filter is reached the amount of device in the filter can be configured in menuconfig.

Return

- ESP_OK : success
- other : failed

Structures

struct esp_bt_controller_config_t

Controller config options, depend on config mask. Config mask indicate which functions enabled, this means some options or parameters of some functions enabled by config mask.

Public Members

uint16_t **controller_task_stack_size**

Bluetooth controller task stack size

uint8_t **controller_task_prio**

Bluetooth controller task priority

uint8_t **hci_uart_no**
If use UART1/2 as HCI IO interface, indicate UART number

uint32_t **hci_uart_baudrate**
If use UART1/2 as HCI IO interface, indicate UART baudrate

uint8_t **scan_duplicate_mode**
scan duplicate mode

uint8_t **scan_duplicate_type**
scan duplicate type

uint16_t **normal_adv_size**
Normal adv size for scan duplicate

uint16_t **mesh_adv_size**
Mesh adv size for scan duplicate

uint16_t **send_adv_reserved_size**
Controller minimum memory value

uint32_t **controller_debug_flag**
Controller debug log flag

uint8_t **mode**
Controller mode: BR/EDR, BLE or Dual Mode

uint8_t **ble_max_conn**
BLE maximum connection numbers

uint8_t **bt_max_acl_conn**
BR/EDR maximum ACL connection numbers

uint8_t **bt_sco_datapath**
SCO data path, i.e. HCI or PCM module

bool **auto_latency**
BLE auto latency, used to enhance classic BT performance

uint8_t **bt_max_sync_conn**
BR/EDR maximum ACL connection numbers. Effective in menuconfig

uint8_t **ble_sca**
BLE low power crystal accuracy index

uint32_t **magic**
Magic number

struct esp_vhci_host_callback
esp_vhci_host_callback used for vhci call host function to notify what host need to do

Public Members

void (*notify_host_send_available)(void)
callback used to notify that the host can send packet to controller

int (*notify_host_recv)(uint8_t *data, uint16_t len)
callback used to notify that the controller has a packet to send to the host

Macros

ESP_BT_CONTROLLER_CONFIG_MAGIC_VAL

BT_CONTROLLER_INIT_CONFIG_DEFAULT()

Type Definitions

typedef struct *esp_vhci_host_callback* esp_vhci_host_callback_t
esp_vhci_host_callback used for vhci call host function to notify what host need to do

Enumerations

enum esp_bt_mode_t
Bluetooth mode for controller enable/disable.

Values:

ESP_BT_MODE_IDLE = 0x00
Bluetooth is not running

ESP_BT_MODE_BLE = 0x01
Run BLE mode

ESP_BT_MODE_CLASSIC_BT = 0x02
Run Classic BT mode

ESP_BT_MODE_BTDM = 0x03
Run dual mode

enum [anonymous]
BLE sleep clock accuracy(SCA), values for ble_sca field in *esp_bt_controller_config_t*, currently only ESP_BLE_SCA_500PPM and ESP_BLE_SCA_250PPM are supported.

Values:

ESP_BLE_SCA_500PPM = 0
BLE SCA at 500ppm

ESP_BLE_SCA_250PPM
BLE SCA at 250ppm

ESP_BLE_SCA_150PPM
BLE SCA at 150ppm

ESP_BLE_SCA_100PPM
BLE SCA at 100ppm

ESP_BLE_SCA_75PPM
BLE SCA at 75ppm

ESP_BLE_SCA_50PPM
BLE SCA at 50ppm

ESP_BLE_SCA_30PPM
BLE SCA at 30ppm

ESP_BLE_SCA_20PPM
BLE SCA at 20ppm

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

Values:

ESP_BT_CONTROLLER_STATUS_IDLE = 0

ESP_BT_CONTROLLER_STATUS_INITED

ESP_BT_CONTROLLER_STATUS_ENABLED

ESP_BT_CONTROLLER_STATUS_NUM

enum esp_ble_power_type_t
BLE tx power type ESP_BLE_PWR_TYPE_CONN_HDL0-8: for each connection, and only be set after connection completed. when disconnect, the correspond TX power is not effected. ESP_BLE_PWR_TYPE_ADV : for advertising/scan response. ESP_BLE_PWR_TYPE_SCAN : for scan. ESP_BLE_PWR_TYPE_DEFAULT : if each connection's TX power is not set, it will use this default value. if neither in scan mode nor in adv mode, it will use this default value. If none of power type is set, system will use ESP_PWR_LVL_P3 as default for ADV/SCAN/CONN0-9.

Values:

ESP_BLE_PWR_TYPE_CONN_HDL0 = 0
For connection handle 0

ESP_BLE_PWR_TYPE_CONN_HDL1 = 1
For connection handle 1

ESP_BLE_PWR_TYPE_CONN_HDL2 = 2
For connection handle 2

ESP_BLE_PWR_TYPE_CONN_HDL3 = 3
For connection handle 3

ESP_BLE_PWR_TYPE_CONN_HDL4 = 4
For connection handle 4

ESP_BLE_PWR_TYPE_CONN_HDL5 = 5
For connection handle 5

ESP_BLE_PWR_TYPE_CONN_HDL6 = 6
For connection handle 6

ESP_BLE_PWR_TYPE_CONN_HDL7 = 7
For connection handle 7

ESP_BLE_PWR_TYPE_CONN_HDL8 = 8
For connection handle 8

ESP_BLE_PWR_TYPE_ADV = 9
For advertising

ESP_BLE_PWR_TYPE_SCAN = 10
For scan

ESP_BLE_PWR_TYPE_DEFAULT = 11
For default, if not set other, it will use default value

ESP_BLE_PWR_TYPE_NUM = 12
TYPE numbers

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

Values:

ESP_PWR_LVL_N12 = 0
Corresponding to -12dbm

ESP_PWR_LVL_N9 = 1
Corresponding to -9dbm

ESP_PWR_LVL_N6 = 2
Corresponding to -6dbm

ESP_PWR_LVL_N3 = 3
Corresponding to -3dbm

ESP_PWR_LVL_N0 = 4
Corresponding to 0dbm

ESP_PWR_LVL_P3 = 5
Corresponding to +3dbm

`ESP_PWR_LVL_P6` = 6
Corresponding to +6dbm

`ESP_PWR_LVL_P9` = 7
Corresponding to +9dbm

`ESP_PWR_LVL_N14` = ESP_PWR_LVL_N12
Backward compatibility! Setting to -14dbm will actually result to -12dbm

`ESP_PWR_LVL_N11` = ESP_PWR_LVL_N9
Backward compatibility! Setting to -11dbm will actually result to -9dbm

`ESP_PWR_LVL_N8` = ESP_PWR_LVL_N6
Backward compatibility! Setting to -8dbm will actually result to -6dbm

`ESP_PWR_LVL_N5` = ESP_PWR_LVL_N3
Backward compatibility! Setting to -5dbm will actually result to -3dbm

`ESP_PWR_LVL_N2` = ESP_PWR_LVL_N0
Backward compatibility! Setting to -2dbm will actually result to 0dbm

`ESP_PWR_LVL_P1` = ESP_PWR_LVL_P3
Backward compatibility! Setting to +1dbm will actually result to +3dbm

`ESP_PWR_LVL_P4` = ESP_PWR_LVL_P6
Backward compatibility! Setting to +4dbm will actually result to +6dbm

`ESP_PWR_LVL_P7` = ESP_PWR_LVL_P9
Backward compatibility! Setting to +7dbm will actually result to +9dbm

`enum esp_sco_data_path_t`
Bluetooth audio data transport path.

Values:

`ESP_SCO_DATA_PATH_HCI` = 0
data over HCI transport

`ESP_SCO_DATA_PATH_PCM` = 1
data over PCM interface

2.1.2 BT COMMON

BT GENERIC DEFINES

Overview

Instructions

Application Example

Instructions

API Reference

Header File

- bt/host/bluedroid/api/include/api/esp_bt_defs.h

Structures

struct esp_bt_uuid_t

UUID type.

Public Members

uint16_t **len**

UUID length, 16bit, 32bit or 128bit

union *esp_bt_uuid_t*::[**anonymous**] **uuid**

UUID

Macros

ESP_BLUEDROID_STATUS_CHECK(status)

ESP_BT_OCTET16_LEN

ESP_BT_OCTET8_LEN

ESP_DEFAULT_GATT_IF

Default GATT interface id.

ESP_BLE_CONN_INT_MIN

relate to BTM_BLE_CONN_INT_MIN in stack/btm_ble_api.h

ESP_BLE_CONN_INT_MAX

relate to BTM_BLE_CONN_INT_MAX in stack/btm_ble_api.h

ESP_BLE_CONN_LATENCY_MAX

relate to ESP_BLE_CONN_LATENCY_MAX in stack/btm_ble_api.h

ESP_BLE_CONN_SUP_TOUT_MIN

relate to BTM_BLE_CONN_SUP_TOUT_MIN in stack/btm_ble_api.h

ESP_BLE_CONN_SUP_TOUT_MAX
relate to ESP_BLE_CONN_SUP_TOUT_MAX in stack/btm_ble_api.h

ESP_BLE_CONN_PARAM_UNDEF

ESP_BLE_SCAN_PARAM_UNDEF

ESP_BLE_IS_VALID_PARAM(x, min, max)
Check the param is valid or not.

ESP_UUID_LEN_16

ESP_UUID_LEN_32

ESP_UUID_LEN_128

ESP_BD_ADDR_LEN
Bluetooth address length.

ESP_BLE_ENC_KEY_MASK
Used to exchange the encryption key in the init key & response key.

ESP_BLE_ID_KEY_MASK
Used to exchange the IRK key in the init key & response key.

ESP_BLE_CSR_KEY_MASK
Used to exchange the CSRK key in the init key & response key.

ESP_BLE_LINK_KEY_MASK
Used to exchange the link key(this key just used in the BLE & BR/EDR coexist mode) in the init key & response key.

ESP_APP_ID_MIN
Minimum of the application id.

ESP_APP_ID_MAX
Maximum of the application id.

ESP_BD_ADDR_STR

ESP_BD_ADDR_HEX(addr)

Type Definitions

typedef uint8_t esp_bt_octet16_t[ESP_BT_OCTET16_LEN]

typedef uint8_t esp_bt_octet8_t[ESP_BT_OCTET8_LEN]

typedef uint8_t esp_link_key[ESP_BT_OCTET16_LEN]

typedef uint8_t esp_bd_addr_t[ESP_BD_ADDR_LEN]
Bluetooth device address.

typedef uint8_t esp_ble_key_mask_t

Enumerations

enum esp_bt_status_t

Status Return Value.

Values:

ESP_BT_STATUS_SUCCESS = 0

ESP_BT_STATUS_FAIL

ESP_BT_STATUS_NOT_READY

ESP_BT_STATUS_NOMEM

ESP_BT_STATUS_BUSY

ESP_BT_STATUS_DONE = 5

ESP_BT_STATUS_UNSUPPORTED

ESP_BT_STATUS_PARM_INVALID

ESP_BT_STATUS_UNHANDLED

ESP_BT_STATUS_AUTH_FAILURE

ESP_BT_STATUS_RMT_DEV_DOWN = 10

ESP_BT_STATUS_AUTH_REJECTED

ESP_BT_STATUS_INVALID_STATIC_RAND_ADDR

ESP_BT_STATUS_PENDING

ESP_BT_STATUS_UNACCEPT_CONN_INTERVAL

ESP_BT_STATUS_PARAM_OUT_OF_RANGE

ESP_BT_STATUS_TIMEOUT

ESP_BT_STATUS_PEER_LE_DATA_LEN_UNSUPPORTED

ESP_BT_STATUS_CONTROL_LE_DATA_LEN_UNSUPPORTED

ESP_BT_STATUS_ERR_ILLEGAL_PARAMETER_FMT

ESP_BT_STATUS_MEMORY_FULL = 20

ESP_BT_STATUS_EIR_TOO_LARGE

enum esp_bt_dev_type_t

Bluetooth device type.

Values:

ESP_BT_DEVICE_TYPE_BREDR = 0x01

ESP_BT_DEVICE_TYPE_BLE = 0x02

ESP_BT_DEVICE_TYPE_DUMO = 0x03

enum esp_ble_addr_type_t

BLE device address type.

Values:

BLE_ADDR_TYPE_PUBLIC = 0x00

BLE_ADDR_TYPE_RANDOM = 0x01

BLE_ADDR_TYPE_RPA_PUBLIC = 0x02

BLE_ADDR_TYPE_RPA_RANDOM = 0x03

enum esp_ble_wl_addr_type_t

white list address type

Values:

BLE_WL_ADDR_TYPE_PUBLIC = 0x00

BLE_WL_ADDR_TYPE_RANDOM = 0x01

BT MAIN API

Overview

Instructions

Application Example

Instructions

API Reference

Header File

- bt/host/bluedroid/api/include/api/esp_bt_main.h

Functions

esp_bluedroid_status_t esp_bluedroid_get_status(void)

Get bluetooth stack status.

Return Bluetooth stack status

esp_err_t **esp_bluedroid_enable**(void)

Enable bluetooth, must after esp_bluedroid_init()

Return

- ESP_OK : Succeed
- Other : Failed

esp_err_t **esp_bluedroid_disable**(void)

Disable bluetooth, must prior to esp_bluedroid_deinit()

Return

- ESP_OK : Succeed
- Other : Failed

esp_err_t **esp_bluedroid_init**(void)

Init and alloc the resource for bluetooth, must be prior to every bluetooth stuff.

Return

- ESP_OK : Succeed
- Other : Failed

esp_err_t **esp_bluedroid_deinit**(void)

Deinit and free the resource for bluetooth, must be after every bluetooth stuff.

Return

- ESP_OK : Succeed
- Other : Failed

Enumerations

enum esp_bluedroid_status_t

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

Values:

ESP_BLUEDROID_STATUS_UNINITIALIZED = 0

Bluetooth not initialized

ESP_BLUEDROID_STATUS_INITIALIZED

Bluetooth initialized but not enabled

`ESP_BLUEDROID_STATUS_ENABLED`
Bluetooth initialized and enabled

BT DEVICE APIs

Overview

Bluetooth device reference APIs.

Instructions

Application Example

Instructions

API Reference

Header File

- bt/host/bluedroid/api/include/api/esp_bt_device.h

Functions

`const` uint8_t *`esp_bt_dev_get_address`(void)
Get bluetooth device address. Must use after "esp_bluedroid_enable" .

Return bluetooth device address (six bytes), or NULL if bluetooth stack is not enabled

esp_err_t `esp_bt_dev_set_device_name`(`const` char **name*)
Set bluetooth device name. This function should be called after esp_bluedroid_enable() completes successfully. A BR/EDR/LE device type shall have a single Bluetooth device name which shall be identical irrespective of the physical channel used to perform the name discovery procedure.

Return

- ESP_OK : Succeed
- ESP_ERR_INVALID_ARG : if name is NULL pointer or empty, or string length out of limit
- ESP_ERR_INVALID_STATE : if bluetooth stack is not yet enabled
- ESP_FAIL : others

Parameters

- `name`: : device name to be set

2.1.3 BT LE

GAP API

Overview

Instructions

Application Example

Check bluetooth/bluedroid/ble folder in ESP-IDF examples, which contains the following demos and their tutorials:

- This is a SMP security client demo and its tutorial. This demo initiates its security parameters and acts as a GATT client, which can send a security request to the peer device and then complete the encryption procedure.
  - bluetooth/bluedroid/ble/gatt_security_client
  - GATT Security Client Example Walkthrough
- This is a SMP security server demo and its tutorial. This demo initiates its security parameters and acts as a GATT server, which can send a pair request to the peer device and then complete the encryption procedure.
  - bluetooth/bluedroid/ble/gatt_security_server
  - GATT Security Server Example Walkthrough

API Reference

Header File

- bt/host/bluedroid/api/include/api/esp_gap_ble_api.h

Functions

esp_err_t **esp_ble_gap_register_callback**(*esp_gap_ble_cb_t callback*)

This function is called to occur gap event, such as scan result.

Return

- ESP_OK : success

- other : failed

Parameters

- `callback`: callback function

esp_err_t **esp_ble_gap_config_adv_data**(*esp_ble_adv_data_t* **adv_data*)

This function is called to override the BTA default ADV parameters.

Return

- ESP_OK : success
- other : failed

Parameters

- `adv_data`: Pointer to User defined ADV data structure. This memory space can not be freed until callback of config_adv_data is received.

esp_err_t **esp_ble_gap_set_scan_params**(*esp_ble_scan_params_t* **scan_params*)

This function is called to set scan parameters.

Return

- ESP_OK : success
- other : failed

Parameters

- `scan_params`: Pointer to User defined scan_params data structure. This memory space can not be freed until callback of set_scan_params

esp_err_t **esp_ble_gap_start_scanning**(uint32_t *duration*)

This procedure keep the device scanning the peer device which advertising on the air.

Return

- ESP_OK : success
- other : failed

Parameters

- `duration`: Keeping the scanning time, the unit is second.

esp_err_t **esp_ble_gap_stop_scanning**(void)

This function call to stop the device scanning the peer device which advertising on the air.

Return

- ESP_OK : success

– other : failed

esp_err_t **esp_ble_gap_start_advertising**(*esp_ble_adv_params_t* **adv_params*)

This function is called to start advertising.

Return

- ESP_OK : success
- other : failed

Parameters

- `adv_params`: pointer to User defined adv_params data structure.

esp_err_t **esp_ble_gap_stop_advertising**(void)

This function is called to stop advertising.

Return

- ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_update_conn_params**(*esp_ble_conn_update_params_t* **params*)

Update connection parameters, can only be used when connection is up.

Return

- ESP_OK : success
- other : failed

Parameters

- `params`: - connection update parameters

esp_err_t **esp_ble_gap_set_pkt_data_len**(*esp_bd_addr_t remote_device*, uint16_t *tx_data_length*)

This function is to set maximum LE data packet size.

Return

- ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_set_rand_addr**(*esp_bd_addr_t rand_addr*)

This function sets the random address for the application.

Return

- ESP_OK : success
- other : failed

Parameters

- `rand_addr`: the random address which should be setting

esp_err_t **esp_ble_gap_clear_rand_addr**(void)

This function clears the random address for the application.

Return

- ESP_OK : success
- other : failed

esp_err_t **esp_ble_gap_config_local_privacy**(bool *privacy_enable*)

Enable/disable privacy on the local device.

Return

- ESP_OK : success
- other : failed

Parameters

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

esp_err_t **esp_ble_gap_config_local_icon**(uint16_t *icon*)

set local gap appearance icon

Return

- ESP_OK : success
- other : failed

Parameters

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

esp_err_t **esp_ble_gap_update_whitelist**(bool *add_remove*, *esp_bd_addr_t* *remote_bda*, *esp_ble_wl_addr_type_t* *wl_addr_type*)

Add or remove device from white list.

Return

- ESP_OK : success

- other : failed

Parameters

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

esp_err_t **esp_ble_gap_get_whitelist_size**(uint16_t **length*)

Get the whitelist size in the controller.

Return

- ESP_OK : success
- other : failed

Parameters

- `length`: the white list length.

esp_err_t **esp_ble_gap_set_prefer_conn_params**(*esp_bd_addr_t bd_addr*, uint16_t *min_conn_int*, uint16_t *max_conn_int*, uint16_t *slave_latency*, uint16_t *supervision_tout*)

This function is called to set the preferred connection parameters when default connection parameter is not desired before connecting. This API can only be used in the master role.

Return

- ESP_OK : success
- other : failed

Parameters

- `bd_addr`: BD address of the peripheral
- `min_conn_int`: minimum preferred connection interval
- `max_conn_int`: maximum preferred connection interval
- `slave_latency`: preferred slave latency
- `supervision_tout`: preferred supervision timeout

esp_err_t **esp_ble_gap_set_device_name**(**const** char **name*)

Set device name to the local device.

Return

- ESP_OK : success
- other : failed

Parameters

- `name`: - device name.

esp_err_t **esp_ble_gap_get_local_used_addr**(*esp_bd_addr_t local_used_addr*, uint8_t **addr_type*)

This function is called to get local used address and adress type. uint8_t *esp_bt_dev_get_address(void) get the public address.

Return - ESP_OK : success

- other : failed

Parameters

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

uint8_t ***esp_ble_resolve_adv_data**(uint8_t **adv_data*, uint8_t *type*, uint8_t **length*)

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

Return pointer of ADV data

Parameters

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

esp_err_t **esp_ble_gap_config_adv_data_raw**(uint8_t **raw_data*, uint32_t *raw_data_len*)

This function is called to set raw advertising data. User need to fill ADV data by self.

Return

- ESP_OK : success
- other : failed

Parameters

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

esp_err_t **esp_ble_gap_config_scan_rsp_data_raw**(uint8_t **raw_data*, uint32_t *raw_data_len*)

This function is called to set raw scan response data. User need to fill scan response data by self.

Return

- ESP_OK : success
- other : failed

Parameters

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

esp_err_t **esp_ble_gap_read_rssi**(*esp_bd_addr_t remote_addr*)

This function is called to read the RSSI of remote device. The address of link policy results are returned in the gap callback function with ESP_GAP_BLE_READ_RSSI_COMPLETE_EVT event.

Return

- ESP_OK : success
- other : failed

Parameters

- `remote_addr`: : The remote connection device address.

esp_err_t **esp_ble_gap_add_duplicate_scan_exceptional_device**(*esp_ble_duplicate_exceptional_info_type_t type*, *esp_duplicate_info_t device_info*)

This function is called to add a device info into the duplicate scan exceptional list.

Return

- ESP_OK : success
- other : failed

Parameters

- `type`: device info type, it is defined in esp_ble_duplicate_exceptional_info_type_t when type is MESH_BEACON_TYPE, MESH_PROV_SRV_ADV or MESH_PROXY_SRV_ADV , device_info is invalid.
- `device_info`: the device information.

esp_err_t **esp_ble_gap_remove_duplicate_scan_exceptional_device**(*esp_ble_duplicate_exceptional_info_type_t type*, *esp_duplicate_info_t device_info*)

This function is called to remove a device info from the duplicate scan exceptional list.

Return

- ESP_OK : success

- other : failed

Parameters

- `type`: device info type, it is defined in esp_ble_duplicate_exceptional_info_type_t when type is MESH_BEACON_TYPE, MESH_PROV_SRV_ADV or MESH_PROXY_SRV_ADV , device_info is invalid.
- `device_info`: the device information.

esp_err_t **esp_ble_gap_clean_duplicate_scan_exceptional_list**(*esp_duplicate_scan_exceptional_list_type_t list_type*)

This function is called to clean the duplicate scan exceptional list. This API will delete all device information in the duplicate scan exceptional list.

Return

- ESP_OK : success
- other : failed

Parameters

- `list_type`: duplicate scan exceptional list type, the value can be one or more of esp_duplicate_scan_exceptional_list_type_t.

esp_err_t **esp_ble_gap_set_security_param**(*esp_ble_sm_param_t param_type*, void **value*, uint8_t *len*)

Set a GAP security parameter value. Overrides the default value.

Return - ESP_OK : success

- other : failed

Parameters

- `param_type`: : the type of the param which to be set
- `value`: : the param value
- `len`: : the length of the param value

esp_err_t **esp_ble_gap_security_rsp**(*esp_bd_addr_t bd_addr*, bool *accept*)

Grant security request access.

Return - ESP_OK : success

- other : failed

Parameters

- `bd_addr`: : BD address of the peer
- `accept`: : accept the security request or not

esp_err_t **esp_ble_set_encryption**(*esp_bd_addr_t bd_addr*, *esp_ble_sec_act_t sec_act*)

Set a gap parameter value. Use this function to change the default GAP parameter values.

Return - ESP_OK : success

- other : failed

Parameters

- `bd_addr`: : the address of the peer device need to encryption
- `sec_act`: : This is the security action to indicate what kind of BLE security level is required for the BLE link if the BLE is supported

esp_err_t **esp_ble_passkey_reply**(*esp_bd_addr_t bd_addr*, bool *accept*, uint32_t *passkey*)

Reply the key value to the peer device in the legacy connection stage.

Return - ESP_OK : success

- other : failed

Parameters

- `bd_addr`: : BD address of the peer
- `accept`: : passkey entry successful or declined.
- `passkey`: : passkey value, must be a 6 digit number, can be lead by 0.

esp_err_t **esp_ble_confirm_reply**(*esp_bd_addr_t bd_addr*, bool *accept*)

Reply the confirm value to the peer device in the secure connection stage.

Return - ESP_OK : success

- other : failed

Parameters

- `bd_addr`: : BD address of the peer device
- `accept`: : numbers to compare are the same or different.

esp_err_t **esp_ble_remove_bond_device**(*esp_bd_addr_t bd_addr*)

Removes a device from the security database list of peer device. It manages unpairing event while connected.

Return - ESP_OK : success

- other : failed

Parameters

- `bd_addr`: : BD address of the peer device

int **esp_ble_get_bond_device_num**(void)

Get the device number from the security database list of peer device. It will return the device bonded number immediately.

Return - >= 0 : bonded devices number.

- ESP_FAIL : failed

esp_err_t **esp_ble_get_bond_device_list**(int **dev_num*, *esp_ble_bond_dev_t *dev_list*)

Get the device from the security database list of peer device. It will return the device bonded information immediately.

Return - ESP_OK : success

- other : failed

Parameters

- `dev_num`: Indicate the dev_list array(buffer) size as input. If dev_num is large enough, it means the actual number as output. Suggest that dev_num value equal to esp_ble_get_bond_device_num().
- `dev_list`: an array(buffer) of *esp_ble_bond_dev_t* type. Use for storing the bonded devices address. The dev_list should be allocated by who call this API.

esp_err_t **esp_ble_oob_req_reply**(*esp_bd_addr_t bd_addr*, uint8_t **TK*, uint8_t *len*)

This function is called to provide the OOB data for SMP in response to ESP_GAP_BLE_OOB_REQ_EVT.

Return - ESP_OK : success

- other : failed

Parameters

- `bd_addr`: BD address of the peer device.
- `TK`: TK value, the TK value shall be a 128-bit random number
- `len`: length of tk, should always be 128-bit

esp_err_t **esp_ble_gap_disconnect**(*esp_bd_addr_t remote_device*)

This function is to disconnect the physical connection of the peer device gattc may have multiple virtual GATT server connections when multiple app_id registered. esp_ble_gattc_close (esp_gatt_if_t gattc_if, uint16_t conn_id) only close one virtual GATT server connection. if there exist other virtual GATT server connections, it does not disconnect the physical connection. esp_ble_gap_disconnect(esp_bd_addr_t remote_device) disconnect the physical connection directly.

Return - ESP_OK : success

- other : failed

Parameters

- `remote_device`: : BD address of the peer device

esp_err_t **esp_ble_get_current_conn_params**(*esp_bd_addr_t bd_addr*, *esp_gap_conn_params_t *conn_params*)

This function is called to read the connection parameters information of the device.

Return - ESP_OK : success

- other : failed

Parameters

- `bd_addr`: BD address of the peer device.
- `conn_params`: the connection parameters information

esp_err_t **esp_gap_ble_set_channels**(*esp_gap_ble_channels channels*)

BLE set channels.

Return - ESP_OK : success

- ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
- other : failed

Parameters

- `channels`: : The n th such field (in the range 0 to 36) contains the value for the link layer channel index n. 0 means channel n is bad. 1 means channel n is unknown. The most significant bits are reserved and shall be set to 0. At least one channel shall be marked as unknown.

Unions

union esp_ble_key_value_t

#include <esp_gap_ble_api.h> union type of the security key value

Public Members

esp_ble_penc_keys_t **penc_key**

received peer encryption key

esp_ble_pcsrk_keys_t **pcsrk_key**

received peer device SRK

esp_ble_pid_keys_t **pid_key**
peer device ID key

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

esp_ble_lcsrk_keys **lcsrk_key**
local device CSRK = d1(ER,DIV,1)

union esp_ble_sec_t
#include <esp_gap_ble_api.h> union associated with ble security

Public Members

esp_ble_sec_key_notif_t **key_notif**
passkey notification

esp_ble_sec_req_t **ble_req**
BLE SMP related request

esp_ble_key_t **ble_key**
BLE SMP keys used when pairing

esp_ble_local_id_keys_t **ble_id_keys**
BLE IR event

esp_ble_auth_cmpl_t **auth_cmpl**
Authentication complete indication.

union esp_ble_gap_cb_param_t
#include <esp_gap_ble_api.h> Gap callback parameters union.

Public Members

struct *esp_ble_gap_cb_param_t*::*ble_adv_data_cmpl_evt_param* **adv_data_cmpl**
Event parameter of ESP_GAP_BLE_ADV_DATA_SET_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_scan_rsp_data_cmpl_evt_param* **scan_rsp_data_cmpl**
Event parameter of ESP_GAP_BLE_SCAN_RSP_DATA_SET_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_scan_param_cmpl_evt_param* **scan_param_cmpl**
Event parameter of ESP_GAP_BLE_SCAN_PARAM_SET_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_scan_result_evt_param* **scan_rst**
Event parameter of ESP_GAP_BLE_SCAN_RESULT_EVT

struct *esp_ble_gap_cb_param_t*::*ble_adv_data_raw_cmpl_evt_param* **adv_data_raw_cmpl**
Event parameter of ESP_GAP_BLE_ADV_DATA_RAW_SET_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_scan_rsp_data_raw_cmpl_evt_param* **scan_rsp_data_raw_cmpl**
Event parameter of ESP_GAP_BLE_SCAN_RSP_DATA_RAW_SET_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_adv_start_cmpl_evt_param* **adv_start_cmpl**
Event parameter of ESP_GAP_BLE_ADV_START_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_scan_start_cmpl_evt_param* **scan_start_cmpl**
Event parameter of ESP_GAP_BLE_SCAN_START_COMPLETE_EVT

esp_ble_sec_t **ble_security**
ble gap security union type

struct *esp_ble_gap_cb_param_t*::*ble_scan_stop_cmpl_evt_param* **scan_stop_cmpl**
Event parameter of ESP_GAP_BLE_SCAN_STOP_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_adv_stop_cmpl_evt_param* **adv_stop_cmpl**
Event parameter of ESP_GAP_BLE_ADV_STOP_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_set_rand_cmpl_evt_param* **set_rand_addr_cmpl**
Event parameter of ESP_GAP_BLE_SET_STATIC_RAND_ADDR_EVT

struct *esp_ble_gap_cb_param_t*::*ble_update_conn_params_evt_param* **update_conn_params**
Event parameter of ESP_GAP_BLE_UPDATE_CONN_PARAMS_EVT

struct *esp_ble_gap_cb_param_t*::*ble_pkt_data_length_cmpl_evt_param* **pkt_data_lenth_cmpl**
Event parameter of ESP_GAP_BLE_SET_PKT_LENGTH_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_local_privacy_cmpl_evt_param* **local_privacy_cmpl**
Event parameter of ESP_GAP_BLE_SET_LOCAL_PRIVACY_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_remove_bond_dev_cmpl_evt_param* **remove_bond_dev_cmpl**
Event parameter of ESP_GAP_BLE_REMOVE_BOND_DEV_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_clear_bond_dev_cmpl_evt_param* **clear_bond_dev_cmpl**
Event parameter of ESP_GAP_BLE_CLEAR_BOND_DEV_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_get_bond_dev_cmpl_evt_param* **get_bond_dev_cmpl**
Event parameter of ESP_GAP_BLE_GET_BOND_DEV_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_read_rssi_cmpl_evt_param* **read_rssi_cmpl**
Event parameter of ESP_GAP_BLE_READ_RSSI_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_update_whitelist_cmpl_evt_param* **update_whitelist_cmpl**
Event parameter of ESP_GAP_BLE_UPDATE_WHITELIST_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_update_duplicate_exceptional_list_cmpl_evt_param* **update_duplicate_exc**
Event parameter of ESP_GAP_BLE_UPDATE_DUPLICATE_EXCEPTIONAL_LIST_COMPLETE_EVT

struct *esp_ble_gap_cb_param_t*::*ble_set_channels_evt_param* **ble_set_channels**
Event parameter of ESP_GAP_BLE_SET_CHANNELS_EVT

struct ble_adv_data_cmpl_evt_param

#include <esp_gap_ble_api.h> ESP_GAP_BLE_ADV_DATA_SET_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate the set advertising data operation success status

struct ble_adv_data_raw_cmpl_evt_param

#include <esp_gap_ble_api.h> ESP_GAP_BLE_ADV_DATA_RAW_SET_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate the set raw advertising data operation success status

struct ble_adv_start_cmpl_evt_param

#include <esp_gap_ble_api.h> ESP_GAP_BLE_ADV_START_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate advertising start operation success status

struct ble_adv_stop_cmpl_evt_param

#include <esp_gap_ble_api.h> ESP_GAP_BLE_ADV_STOP_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate adv stop operation success status

struct ble_clear_bond_dev_cmpl_evt_param

#include <esp_gap_ble_api.h> ESP_GAP_BLE_CLEAR_BOND_DEV_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate the clear bond device operation success status

struct ble_get_bond_dev_cmpl_evt_param

#include <esp_gap_ble_api.h> ESP_GAP_BLE_GET_BOND_DEV_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate the get bond device operation success status

uint8_t **dev_num**

Indicate the get number device in the bond list

esp_ble_bond_dev_t ***bond_dev**

the pointer to the bond device Structure

struct ble_local_privacy_cmpl_evt_param

#include <esp_gap_ble_api.h> ESP_GAP_BLE_SET_LOCAL_PRIVACY_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate the set local privacy operation success status

struct ble_pkt_data_length_cmpl_evt_param

#include <esp_gap_ble_api.h> ESP_GAP_BLE_SET_PKT_LENGTH_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate the set pkt data length operation success status

esp_ble_pkt_data_length_params_t **params**

pkt data length value

struct ble_read_rssi_cmpl_evt_param

#include <esp_gap_ble_api.h> ESP_GAP_BLE_READ_RSSI_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate the read adv tx power operation success status

int8_t **rssi**

The ble remote device rssi value, the range is from -127 to 20, the unit is dbm, if the RSSI cannot be read, the RSSI metric shall be set to 127.

esp_bd_addr_t **remote_addr**

The remote device address

struct ble_remove_bond_dev_cmpl_evt_param

#include <esp_gap_ble_api.h> ESP_GAP_BLE_REMOVE_BOND_DEV_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate the remove bond device operation success status

esp_bd_addr_t **bd_addr**

The device address which has been remove from the bond list

struct ble_scan_param_cmpl_evt_param

#include <esp_gap_ble_api.h> ESP_GAP_BLE_SCAN_PARAM_SET_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate the set scan param operation success status

struct ble_scan_result_evt_param

#include <esp_gap_ble_api.h> ESP_GAP_BLE_SCAN_RESULT_EVT.

Public Members

esp_gap_search_evt_t **search_evt**

Search event type

esp_bd_addr_t **bda**

Bluetooth device address which has been searched

esp_bt_dev_type_t **dev_type**

Device type

esp_ble_addr_type_t **ble_addr_type**

Ble device address type

esp_ble_evt_type_t **ble_evt_type**

Ble scan result event type

int **rssi**

Searched device’ s RSSI

uint8_t **ble_adv**[**ESP_BLE_ADV_DATA_LEN_MAX** + **ESP_BLE_SCAN_RSP_DATA_LEN_MAX**]

Received EIR

int **flag**

Advertising data flag bit

int **num_resps**

Scan result number

uint8_t **adv_data_len**

Adv data length

uint8_t **scan_rsp_len**

Scan response length

uint32_t **num_dis**

The number of discard packets

struct ble_scan_rsp_data_cmpl_evt_param

#include <esp_gap_ble_api.h> ESP_GAP_BLE_SCAN_RSP_DATA_SET_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate the set scan response data operation success status

struct ble_scan_rsp_data_raw_cmpl_evt_param

#include <esp_gap_ble_api.h> ESP_GAP_BLE_SCAN_RSP_DATA_RAW_SET_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate the set raw advertising data operation success status

struct ble_scan_start_cmpl_evt_param

#include <esp_gap_ble_api.h> ESP_GAP_BLE_SCAN_START_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate scan start operation success status

struct ble_scan_stop_cmpl_evt_param

#include <esp_gap_ble_api.h> ESP_GAP_BLE_SCAN_STOP_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate scan stop operation success status

struct ble_set_channels_evt_param

#include <esp_gap_ble_api.h> ESP_GAP_BLE_SET_CHANNELS_EVT.

Public Members

esp_bt_status_t **stat**
BLE set channel status

struct ble_set_rand_cmpl_evt_param
#include <esp_gap_ble_api.h> ESP_GAP_BLE_SET_STATIC_RAND_ADDR_EVT.

Public Members

esp_bt_status_t **status**
Indicate set static rand address operation success status

struct ble_update_conn_params_evt_param
#include <esp_gap_ble_api.h> ESP_GAP_BLE_UPDATE_CONN_PARAMS_EVT.

Public Members

esp_bt_status_t **status**
Indicate update connection parameters success status

esp_bd_addr_t **bda**
Bluetooth device address

uint16_t **min_int**
Min connection interval

uint16_t **max_int**
Max connection interval

uint16_t **latency**
Slave latency for the connection in number of connection events. Range: 0x0000 to 0x01F3

uint16_t **conn_int**
Current connection interval

uint16_t **timeout**
Supervision timeout for the LE Link. Range: 0x000A to 0x0C80. Mandatory Range: 0x000A to 0x0C80 Time = N * 10 msec

struct ble_update_duplicate_exceptional_list_cmpl_evt_param
#include <esp_gap_ble_api.h> ESP_GAP_BLE_UPDATE_DUPLICATE_EXCEPTIONAL_LIST_COMPL

Public Members

esp_bt_status_t **status**
Indicate update duplicate scan exceptional list operation success status

uint8_t **subcode**

Define in esp_bt_duplicate_exceptional_subcode_type_t

uint16_t **length**

The length of device_info

esp_duplicate_info_t **device_info**

device information, when subcode is ESP_BLE_DUPLICATE_EXCEPTIONAL_LIST_CLEAN, the value is invalid

struct ble_update_whitelist_cmpl_evt_param

#include <esp_gap_ble_api.h> ESP_GAP_BLE_UPDATE_WHITELIST_COMPLETE_EVT.

Public Members

esp_bt_status_t **status**

Indicate the add or remove whitelist operation success status

esp_ble_wl_opration_t **wl_opration**

The value is ESP_BLE_WHITELIST_ADD if add address to whitelist operation success, ESP_BLE_WHITELIST_REMOVE if remove address from the whitelist operation success

Structures

struct esp_ble_adv_params_t

Advertising parameters.

Public Members

uint16_t **adv_int_min**

Minimum advertising interval for undirected and low duty cycle directed advertising. Range: 0x0020 to 0x4000 Default: N = 0x0800 (1.28 second) Time = N * 0.625 msec Time Range: 20 ms to 10.24 sec

uint16_t **adv_int_max**

Maximum advertising interval for undirected and low duty cycle directed advertising. Range: 0x0020 to 0x4000 Default: N = 0x0800 (1.28 second) Time = N * 0.625 msec Time Range: 20 ms to 10.24 sec Advertising max interval

esp_ble_adv_type_t **adv_type**

Advertising type

esp_ble_addr_type_t **own_addr_type**

Owner bluetooth device address type

esp_bd_addr_t **peer_addr**

Peer device bluetooth device address

esp_ble_addr_type_t **peer_addr_type**

Peer device bluetooth device address type, only support public address type and random address type

esp_ble_adv_channel_t **channel_map**

Advertising channel map

esp_ble_adv_filter_t **adv_filter_policy**

Advertising filter policy

struct esp_ble_adv_data_t

Advertising data content, according to "Supplement to the Bluetooth Core Specification".

Public Members

bool **set_scan_rsp**

Set this advertising data as scan response or not

bool **include_name**

Advertising data include device name or not

bool **include_txpower**

Advertising data include TX power

int **min_interval**

Advertising data show slave preferred connection min interval. The connection interval in the following manner: connIntervalmin = Conn_Interval_Min * 1.25 ms Conn_Interval_Min range: 0x0006 to 0x0C80 Value of 0xFFFF indicates no specific minimum. Values not defined above are reserved for future use.

int **max_interval**

Advertising data show slave preferred connection max interval. The connection interval in the following manner: connIntervalmax = Conn_Interval_Max * 1.25 ms Conn_Interval_Max range: 0x0006 to 0x0C80 Conn_Interval_Max shall be equal to or greater than the Conn_Interval_Min. Value of 0xFFFF indicates no specific maximum. Values not defined above are reserved for future use.

int **appearance**

External appearance of device

uint16_t **manufacturer_len**

Manufacturer data length

uint8_t ***p_manufacturer_data**

Manufacturer data point

uint16_t **service_data_len**
Service data length

uint8_t ***p_service_data**
Service data point

uint16_t **service_uuid_len**
Service uuid length

uint8_t ***p_service_uuid**
Service uuid array point

uint8_t **flag**
Advertising flag of discovery mode, see BLE_ADV_DATA_FLAG detail

struct esp_ble_scan_params_t
Ble scan parameters.

Public Members

esp_ble_scan_type_t **scan_type**
Scan type

esp_ble_addr_type_t **own_addr_type**
Owner address type

esp_ble_scan_filter_t **scan_filter_policy**
Scan filter policy

uint16_t **scan_interval**
Scan interval. This is defined as the time interval from when the Controller started its last LE scan until it begins the subsequent LE scan. Range: 0x0004 to 0x4000 Default: 0x0010 (10 ms) Time = N * 0.625 msec Time Range: 2.5 msec to 10.24 seconds

uint16_t **scan_window**
Scan window. The duration of the LE scan. LE_Scan_Window shall be less than or equal to LE_Scan_Interval Range: 0x0004 to 0x4000 Default: 0x0010 (10 ms) Time = N * 0.625 msec Time Range: 2.5 msec to 10240 msec

esp_ble_scan_duplicate_t **scan_duplicate**
The Scan_Duplicates parameter controls whether the Link Layer should filter out duplicate advertising reports (BLE_SCAN_DUPLICATE_ENABLE) to the Host, or if the Link Layer should generate advertising reports for each packet received

struct esp_gap_conn_params_t
connection parameters information

Public Members

uint16_t **interval**

connection interval

uint16_t **latency**

Slave latency for the connection in number of connection events. Range: 0x0000 to 0x01F3

uint16_t **timeout**

Supervision timeout for the LE Link. Range: 0x000A to 0x0C80. Mandatory Range: 0x000A to 0x0C80 Time = N * 10 msec Time Range: 100 msec to 32 seconds

struct esp_ble_conn_update_params_t

Connection update parameters.

Public Members

esp_bd_addr_t **bda**

Bluetooth device address

uint16_t **min_int**

Min connection interval

uint16_t **max_int**

Max connection interval

uint16_t **latency**

Slave latency for the connection in number of connection events. Range: 0x0000 to 0x01F3

uint16_t **timeout**

Supervision timeout for the LE Link. Range: 0x000A to 0x0C80. Mandatory Range: 0x000A to 0x0C80 Time = N * 10 msec Time Range: 100 msec to 32 seconds

struct esp_ble_pkt_data_length_params_t

BLE pkt date length keys.

Public Members

uint16_t **rx_len**

pkt rx data length value

uint16_t **tx_len**

pkt tx data length value

struct esp_ble_penc_keys_t

BLE encryption keys.

Public Members

esp_bt_octet16_t **ltk**
The long term key

esp_bt_octet8_t **rand**
The random number

uint16_t **ediv**
The ediv value

uint8_t **sec_level**
The security level of the security link

uint8_t **key_size**
The key size(7~16) of the security link

struct esp_ble_pcsrk_keys_t
BLE CSRK keys.

Public Members

uint32_t **counter**
The counter

esp_bt_octet16_t **csrk**
The csrk key

uint8_t **sec_level**
The security level

struct esp_ble_pid_keys_t
BLE pid keys.

Public Members

esp_bt_octet16_t **irk**
The irk value

esp_ble_addr_type_t **addr_type**
The address type

esp_bd_addr_t **static_addr**
The static address

struct esp_ble_lenc_keys_t
BLE Encryption reproduction keys.

Public Members

esp_bt_octet16_t **ltk**
The long term key

uint16_t **div**
The div value

uint8_t **key_size**
The key size of the security link

uint8_t **sec_level**
The security level of the security link

struct esp_ble_lcsrk_keys
BLE SRK keys.

Public Members

uint32_t **counter**
The counter value

uint16_t **div**
The div value

uint8_t **sec_level**
The security level of the security link

esp_bt_octet16_t **csrk**
The csrk key value

struct esp_ble_sec_key_notif_t
Structure associated with ESP_KEY_NOTIF_EVT.

Public Members

esp_bd_addr_t **bd_addr**
peer address

uint32_t **passkey**
the numeric value for comparison. If just_works, do not show this number to UI

struct esp_ble_sec_req_t
Structure of the security request.

Public Members

esp_bd_addr_t **bd_addr**
peer address

struct esp_ble_bond_key_info_t
struct type of the bond key information value

Public Members

esp_ble_key_mask_t **key_mask**
the key mask to indicate witch key is present

esp_ble_penc_keys_t **penc_key**
received peer encryption key

esp_ble_pcsrk_keys_t **pcsrk_key**
received peer device SRK

esp_ble_pid_keys_t **pid_key**
peer device ID key

struct esp_ble_bond_dev_t
struct type of the bond device value

Public Members

esp_bd_addr_t **bd_addr**
peer address

esp_ble_bond_key_info_t **bond_key**
the bond key information

struct esp_ble_key_t
union type of the security key value

Public Members

esp_bd_addr_t **bd_addr**
peer address

esp_ble_key_type_t **key_type**
key type of the security link

esp_ble_key_value_t **p_key_value**
the pointer to the key value

`struct esp_ble_local_id_keys_t`

structure type of the ble local id keys value

Public Members

esp_bt_octet16_t **ir**

the 16 bits of the ir value

esp_bt_octet16_t **irk**

the 16 bits of the ir key value

esp_bt_octet16_t **dhk**

the 16 bits of the dh key value

`struct esp_ble_auth_cmpl_t`

Structure associated with ESP_AUTH_CMPL_EVT.

Public Members

esp_bd_addr_t **bd_addr**

BD address peer device.

bool **key_present**

Valid link key value in key element

esp_link_key **key**

Link key associated with peer device.

uint8_t **key_type**

The type of Link Key

bool **success**

TRUE of authentication succeeded, FALSE if failed.

uint8_t **fail_reason**

The HCI reason/error code for when success=FALSE

esp_ble_addr_type_t **addr_type**

Peer device address type

esp_bt_dev_type_t **dev_type**

Device type

esp_ble_auth_req_t **auth_mode**

authentication mode

Macros

ESP_BLE_ADV_FLAG_LIMIT_DISC

BLE_ADV_DATA_FLAG data flag bit definition used for advertising data flag

ESP_BLE_ADV_FLAG_GEN_DISC

ESP_BLE_ADV_FLAG_BREDR_NOT_SPT

ESP_BLE_ADV_FLAG_DMT_CONTROLLER_SPT

ESP_BLE_ADV_FLAG_DMT_HOST_SPT

ESP_BLE_ADV_FLAG_NON_LIMIT_DISC

ESP_LE_KEY_NONE

ESP_LE_KEY_PENC

ESP_LE_KEY_PID

ESP_LE_KEY_PCSRK

ESP_LE_KEY_PLK

ESP_LE_KEY_LLK

ESP_LE_KEY_LENC

ESP_LE_KEY_LID

ESP_LE_KEY_LCSRK

ESP_LE_AUTH_NO_BOND

ESP_LE_AUTH_BOND

ESP_LE_AUTH_REQ_MITM

ESP_LE_AUTH_REQ_BOND_MITM

0101

ESP_LE_AUTH_REQ_SC_ONLY

ESP_LE_AUTH_REQ_SC_BOND

ESP_LE_AUTH_REQ_SC_MITM

ESP_LE_AUTH_REQ_SC_MITM_BOND

ESP_BLE_ONLY_ACCEPT_SPECIFIED_AUTH_DISABLE

ESP_BLE_ONLY_ACCEPT_SPECIFIED_AUTH_ENABLE

ESP_BLE_OOB_DISABLE

ESP_BLE_OOB_ENABLE

ESP_IO_CAP_OUT

ESP_IO_CAP_IO

ESP_IO_CAP_IN

ESP_IO_CAP_NONE

ESP_IO_CAP_KBDISP

ESP_BLE_APPEARANCE_UNKNOWN

ESP_BLE_APPEARANCE_GENERIC_PHONE

ESP_BLE_APPEARANCE_GENERIC_COMPUTER

ESP_BLE_APPEARANCE_GENERIC_WATCH

ESP_BLE_APPEARANCE_SPORTS_WATCH

ESP_BLE_APPEARANCE_GENERIC_CLOCK

ESP_BLE_APPEARANCE_GENERIC_DISPLAY

ESP_BLE_APPEARANCE_GENERIC_REMOTE

ESP_BLE_APPEARANCE_GENERIC_EYEGLASSES

ESP_BLE_APPEARANCE_GENERIC_TAG

ESP_BLE_APPEARANCE_GENERIC_KEYRING

ESP_BLE_APPEARANCE_GENERIC_MEDIA_PLAYER

ESP_BLE_APPEARANCE_GENERIC_BARCODE_SCANNER

ESP_BLE_APPEARANCE_GENERIC_THERMOMETER

ESP_BLE_APPEARANCE_THERMOMETER_EAR

ESP_BLE_APPEARANCE_GENERIC_HEART_RATE

ESP_BLE_APPEARANCE_HEART_RATE_BELT

ESP_BLE_APPEARANCE_GENERIC_BLOOD_PRESSURE

ESP_BLE_APPEARANCE_BLOOD_PRESSURE_ARM

ESP_BLE_APPEARANCE_BLOOD_PRESSURE_WRIST

ESP_BLE_APPEARANCE_GENERIC_HID

ESP_BLE_APPEARANCE_HID_KEYBOARD

ESP_BLE_APPEARANCE_HID_MOUSE

ESP_BLE_APPEARANCE_HID_JOYSTICK

ESP_BLE_APPEARANCE_HID_GAMEPAD

ESP_BLE_APPEARANCE_HID_DIGITIZER_TABLET

ESP_BLE_APPEARANCE_HID_CARD_READER

ESP_BLE_APPEARANCE_HID_DIGITAL_PEN

ESP_BLE_APPEARANCE_HID_BARCODE_SCANNER

ESP_BLE_APPEARANCE_GENERIC_GLUCOSE

ESP_BLE_APPEARANCE_GENERIC_WALKING

ESP_BLE_APPEARANCE_WALKING_IN_SHOE

ESP_BLE_APPEARANCE_WALKING_ON_SHOE

ESP_BLE_APPEARANCE_WALKING_ON_HIP

ESP_BLE_APPEARANCE_GENERIC_CYCLING

ESP_BLE_APPEARANCE_CYCLING_COMPUTER

ESP_BLE_APPEARANCE_CYCLING_SPEED

ESP_BLE_APPEARANCE_CYCLING_CADENCE

ESP_BLE_APPEARANCE_CYCLING_POWER

ESP_BLE_APPEARANCE_CYCLING_SPEED_CADENCE

ESP_BLE_APPEARANCE_GENERIC_PULSE_OXIMETER

ESP_BLE_APPEARANCE_PULSE_OXIMETER_FINGERTIP

ESP_BLE_APPEARANCE_PULSE_OXIMETER_WRIST

ESP_BLE_APPEARANCE_GENERIC_WEIGHT

ESP_BLE_APPEARANCE_GENERIC_PERSONAL_MOBILITY_DEVICE

ESP_BLE_APPEARANCE_POWERED_WHEELCHAIR

ESP_BLE_APPEARANCE_MOBILITY_SCOOTER

ESP_BLE_APPEARANCE_GENERIC_CONTINUOUS_GLUCOSE_MONITOR

ESP_BLE_APPEARANCE_GENERIC_INSULIN_PUMP

ESP_BLE_APPEARANCE_INSULIN_PUMP_DURABLE_PUMP

ESP_BLE_APPEARANCE_INSULIN_PUMP_PATCH_PUMP

ESP_BLE_APPEARANCE_INSULIN_PEN

ESP_BLE_APPEARANCE_GENERIC_MEDICATION_DELIVERY

ESP_BLE_APPEARANCE_GENERIC_OUTDOOR_SPORTS

ESP_BLE_APPEARANCE_OUTDOOR_SPORTS_LOCATION

ESP_BLE_APPEARANCE_OUTDOOR_SPORTS_LOCATION_AND_NAV

ESP_BLE_APPEARANCE_OUTDOOR_SPORTS_LOCATION_POD

ESP_BLE_APPEARANCE_OUTDOOR_SPORTS_LOCATION_POD_AND_NAV

ESP_GAP_BLE_CHANNELS_LEN

ESP_GAP_BLE_ADD_WHITELIST_COMPLETE_EVT

This is the old name, just for backwards compatibility.

ESP_BLE_ADV_DATA_LEN_MAX

Advertising data maximum length.

ESP_BLE_SCAN_RSP_DATA_LEN_MAX

Scan response data maximum length.

BLE_BIT(n)

Type Definitions

typedef uint8_t **esp_ble_key_type_t**

typedef uint8_t **esp_ble_auth_req_t**

combination of the above bit pattern

typedef uint8_t **esp_ble_io_cap_t**

combination of the io capability

typedef uint8_t **esp_gap_ble_channels**[ESP_GAP_BLE_CHANNELS_LEN]

typedef uint8_t **esp_duplicate_info_t**[ESP_BD_ADDR_LEN]

typedef void (***esp_gap_ble_cb_t**)(*esp_gap_ble_cb_event_t* event, *esp_ble_gap_cb_param_t* *param)

GAP callback function type.

Parameters

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

Enumerations

enum **esp_gap_ble_cb_event_t**

GAP BLE callback event type.

Values:

ESP_GAP_BLE_ADV_DATA_SET_COMPLETE_EVT = 0
When advertising data set complete, the event comes

ESP_GAP_BLE_SCAN_RSP_DATA_SET_COMPLETE_EVT
When scan response data set complete, the event comes

ESP_GAP_BLE_SCAN_PARAM_SET_COMPLETE_EVT
When scan parameters set complete, the event comes

ESP_GAP_BLE_SCAN_RESULT_EVT
When one scan result ready, the event comes each time

ESP_GAP_BLE_ADV_DATA_RAW_SET_COMPLETE_EVT
When raw advertising data set complete, the event comes

ESP_GAP_BLE_SCAN_RSP_DATA_RAW_SET_COMPLETE_EVT
When raw advertising data set complete, the event comes

ESP_GAP_BLE_ADV_START_COMPLETE_EVT
When starting advertising complete, the event comes

ESP_GAP_BLE_SCAN_START_COMPLETE_EVT
When starting scan complete, the event comes

ESP_GAP_BLE_AUTH_CMPL_EVT
Authentication complete indication.

ESP_GAP_BLE_KEY_EVT
BLE key event for peer device keys

ESP_GAP_BLE_SEC_REQ_EVT
BLE security request

ESP_GAP_BLE_PASSKEY_NOTIF_EVT
passkey notification event

ESP_GAP_BLE_PASSKEY_REQ_EVT
passkey request event

ESP_GAP_BLE_OOB_REQ_EVT
OOB request event

ESP_GAP_BLE_LOCAL_IR_EVT
BLE local IR event

ESP_GAP_BLE_LOCAL_ER_EVT
BLE local ER event

ESP_GAP_BLE_NC_REQ_EVT
Numeric Comparison request event

ESP_GAP_BLE_ADV_STOP_COMPLETE_EVT
When stopping adv complete, the event comes

ESP_GAP_BLE_SCAN_STOP_COMPLETE_EVT
When stopping scan complete, the event comes

ESP_GAP_BLE_SET_STATIC_RAND_ADDR_EVT
When setting the static rand address complete, the event comes

ESP_GAP_BLE_UPDATE_CONN_PARAMS_EVT
When updating connection parameters complete, the event comes

ESP_GAP_BLE_SET_PKT_LENGTH_COMPLETE_EVT
When setting pkt length complete, the event comes

ESP_GAP_BLE_SET_LOCAL_PRIVACY_COMPLETE_EVT
When enabling/disabling privacy on the local device complete, the event comes

ESP_GAP_BLE_REMOVE_BOND_DEV_COMPLETE_EVT
When removing the bond device complete, the event comes

ESP_GAP_BLE_CLEAR_BOND_DEV_COMPLETE_EVT
When clearing the bond device clear complete, the event comes

ESP_GAP_BLE_GET_BOND_DEV_COMPLETE_EVT
When getting the bond device list complete, the event comes

ESP_GAP_BLE_READ_RSSI_COMPLETE_EVT
When reading the rssi complete, the event comes

ESP_GAP_BLE_UPDATE_WHITELIST_COMPLETE_EVT
When adding or removing whitelist complete, the event comes

ESP_GAP_BLE_UPDATE_DUPLICATE_EXCEPTIONAL_LIST_COMPLETE_EVT
When updating duplicate exceptional list complete, the event comes

ESP_GAP_BLE_SET_CHANNELS_EVT
When setting BLE channels complete, the event comes

ESP_GAP_BLE_EVT_MAX

enum esp_ble_adv_data_type
The type of advertising data(not adv_type)

Values:

ESP_BLE_AD_TYPE_FLAG = 0x01

ESP_BLE_AD_TYPE_16SRV_PART = 0x02

ESP_BLE_AD_TYPE_16SRV_CMPL = 0x03

ESP_BLE_AD_TYPE_32SRV_PART = 0x04

ESP_BLE_AD_TYPE_32SRV_CMPL = 0x05

ESP_BLE_AD_TYPE_128SRV_PART = 0x06

ESP_BLE_AD_TYPE_128SRV_CMPL = 0x07

ESP_BLE_AD_TYPE_NAME_SHORT = 0x08

ESP_BLE_AD_TYPE_NAME_CMPL = 0x09

ESP_BLE_AD_TYPE_TX_PWR = 0x0A

ESP_BLE_AD_TYPE_DEV_CLASS = 0x0D

ESP_BLE_AD_TYPE_SM_TK = 0x10

ESP_BLE_AD_TYPE_SM_OOB_FLAG = 0x11

ESP_BLE_AD_TYPE_INT_RANGE = 0x12

ESP_BLE_AD_TYPE_SOL_SRV_UUID = 0x14

ESP_BLE_AD_TYPE_128SOL_SRV_UUID = 0x15

ESP_BLE_AD_TYPE_SERVICE_DATA = 0x16

ESP_BLE_AD_TYPE_PUBLIC_TARGET = 0x17

ESP_BLE_AD_TYPE_RANDOM_TARGET = 0x18

ESP_BLE_AD_TYPE_APPEARANCE = 0x19

ESP_BLE_AD_TYPE_ADV_INT = 0x1A

ESP_BLE_AD_TYPE_LE_DEV_ADDR = 0x1b

ESP_BLE_AD_TYPE_LE_ROLE = 0x1c

ESP_BLE_AD_TYPE_SPAIR_C256 = 0x1d

ESP_BLE_AD_TYPE_SPAIR_R256 = 0x1e

ESP_BLE_AD_TYPE_32SOL_SRV_UUID = 0x1f

ESP_BLE_AD_TYPE_32SERVICE_DATA = 0x20

ESP_BLE_AD_TYPE_128SERVICE_DATA = 0x21

ESP_BLE_AD_TYPE_LE_SECURE_CONFIRM = 0x22

ESP_BLE_AD_TYPE_LE_SECURE_RANDOM = 0x23

ESP_BLE_AD_TYPE_URI = 0x24

ESP_BLE_AD_TYPE_INDOOR_POSITION = 0x25

ESP_BLE_AD_TYPE_TRANS_DISC_DATA = 0x26

ESP_BLE_AD_TYPE_LE_SUPPORT_FEATURE = 0x27

ESP_BLE_AD_TYPE_CHAN_MAP_UPDATE = 0x28

ESP_BLE_AD_MANUFACTURER_SPECIFIC_TYPE = 0xFF

enum esp_ble_adv_type_t

Advertising mode.

Values:

ADV_TYPE_IND = 0x00

ADV_TYPE_DIRECT_IND_HIGH = 0x01

ADV_TYPE_SCAN_IND = 0x02

ADV_TYPE_NONCONN_IND = 0x03

ADV_TYPE_DIRECT_IND_LOW = 0x04

enum esp_ble_adv_channel_t

Advertising channel mask.

Values:

ADV_CHNL_37 = 0x01

ADV_CHNL_38 = 0x02

ADV_CHNL_39 = 0x04

ADV_CHNL_ALL = 0x07

enum esp_ble_adv_filter_t

Values:

ADV_FILTER_ALLOW_SCAN_ANY_CON_ANY = 0x00

Allow both scan and connection requests from anyone.

ADV_FILTER_ALLOW_SCAN_WLST_CON_ANY

Allow both scan req from White List devices only and connection req from anyone.

ADV_FILTER_ALLOW_SCAN_ANY_CON_WLST

Allow both scan req from anyone and connection req from White List devices only.

ADV_FILTER_ALLOW_SCAN_WLST_CON_WLST

Allow scan and connection requests from White List devices only.

enum esp_ble_sec_act_t

Values:

ESP_BLE_SEC_ENCRYPT = 1

ESP_BLE_SEC_ENCRYPT_NO_MITM

ESP_BLE_SEC_ENCRYPT_MITM

enum esp_ble_sm_param_t

Values:

ESP_BLE_SM_PASSKEY = 0

ESP_BLE_SM_AUTHEN_REQ_MODE

ESP_BLE_SM_IOCAP_MODE

ESP_BLE_SM_SET_INIT_KEY

ESP_BLE_SM_SET_RSP_KEY

ESP_BLE_SM_MAX_KEY_SIZE

ESP_BLE_SM_MIN_KEY_SIZE

ESP_BLE_SM_SET_STATIC_PASSKEY

ESP_BLE_SM_CLEAR_STATIC_PASSKEY

ESP_BLE_SM_ONLY_ACCEPT_SPECIFIED_SEC_AUTH

ESP_BLE_SM_OOB_SUPPORT

ESP_BLE_SM_MAX_PARAM

enum esp_ble_scan_type_t

Ble scan type.

Values:

BLE_SCAN_TYPE_PASSIVE = 0x0

Passive scan

BLE_SCAN_TYPE_ACTIVE = 0x1

Active scan

enum esp_ble_scan_filter_t

Ble scan filter type.

Values:

BLE_SCAN_FILTER_ALLOW_ALL = 0x0

Accept all :

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

BLE_SCAN_FILTER_ALLOW_ONLY_WLST = 0x1

Accept only :

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

BLE_SCAN_FILTER_ALLOW_UND_RPA_DIR = 0x2

Accept all :

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

BLE_SCAN_FILTER_ALLOW_WLIST_PRA_DIR = 0x3

Accept all :

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

enum esp_ble_scan_duplicate_t

Ble scan duplicate type.

Values:

BLE_SCAN_DUPLICATE_DISABLE = 0x0

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

BLE_SCAN_DUPLICATE_ENABLE = 0x1

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

BLE_SCAN_DUPLICATE_MAX = 0x2

0x02 – 0xFF, Reserved for future use

enum esp_gap_search_evt_t

Sub Event of ESP_GAP_BLE_SCAN_RESULT_EVT.

Values:

ESP_GAP_SEARCH_INQ_RES_EVT = 0

Inquiry result for a peer device.

ESP_GAP_SEARCH_INQ_CMPL_EVT = 1

Inquiry complete.

ESP_GAP_SEARCH_DISC_RES_EVT = 2

Discovery result for a peer device.

ESP_GAP_SEARCH_DISC_BLE_RES_EVT = 3

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

ESP_GAP_SEARCH_DISC_CMPL_EVT = 4

Discovery complete.

ESP_GAP_SEARCH_DI_DISC_CMPL_EVT = 5

Discovery complete.

ESP_GAP_SEARCH_SEARCH_CANCEL_CMPL_EVT = 6

Search cancelled

ESP_GAP_SEARCH_INQ_DISCARD_NUM_EVT = 7

The number of pkt discarded by flow control

enum esp_ble_evt_type_t

Ble scan result event type, to indicate the result is scan response or advertising data or other.

Values:

ESP_BLE_EVT_CONN_ADV = 0x00

Connectable undirected advertising (ADV_IND)

ESP_BLE_EVT_CONN_DIR_ADV = 0x01

Connectable directed advertising (ADV_DIRECT_IND)

ESP_BLE_EVT_DISC_ADV = 0x02

Scannable undirected advertising (ADV_SCAN_IND)

ESP_BLE_EVT_NON_CONN_ADV = 0x03

Non connectable undirected advertising (ADV_NONCONN_IND)

ESP_BLE_EVT_SCAN_RSP = 0x04

Scan Response (SCAN_RSP)

enum esp_ble_wl_opration_t

Values:

ESP_BLE_WHITELIST_REMOVE = 0X00

remove mac from whitelist

ESP_BLE_WHITELIST_ADD = 0X01

add address to whitelist

enum esp_bt_duplicate_exceptional_subcode_type_t

Values:

ESP_BLE_DUPLICATE_EXCEPTIONAL_LIST_ADD = 0

Add device info into duplicate scan exceptional list

ESP_BLE_DUPLICATE_EXCEPTIONAL_LIST_REMOVE

Remove device info from duplicate scan exceptional list

ESP_BLE_DUPLICATE_EXCEPTIONAL_LIST_CLEAN

Clean duplicate scan exceptional list

enum esp_ble_duplicate_exceptional_info_type_t

Values:

ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_INFO_ADV_ADDR = 0

BLE advertising address , device info will be added into

ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_ADDR_LIST

ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_INFO_MESH_LINK_ID

BLE mesh link ID, it is for BLE mesh, device info will be added into ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_MESH_LINK_ID_LIST

ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_INFO_MESH_BEACON_TYPE

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

ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_INFO_MESH_PROV_SRV_ADV

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

ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_INFO_MESH_PROXY_SRV_ADV

BLE mesh adv with proxy service uuid, the format is | 0x02 | 0x01 | flags | 0x03 | 0x03 | 0x1828 | ···. |‘

enum esp_duplicate_scan_exceptional_list_type_t

Values:

ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_ADDR_LIST = BLE_BIT(0)

duplicate scan exceptional addr list

ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_MESH_LINK_ID_LIST = BLE_BIT(1)

duplicate scan exceptional mesh link ID list

ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_MESH_BEACON_TYPE_LIST = BLE_BIT(2)

duplicate scan exceptional mesh beacon type list

ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_MESH_PROV_SRV_ADV_LIST = BLE_BIT(3)

duplicate scan exceptional mesh adv with provisioning service uuid

ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_MESH_PROXY_SRV_ADV_LIST = BLE_BIT(4)

duplicate scan exceptional mesh adv with provisioning service uuid

ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_ALL_LIST = 0xFFFF

duplicate scan exceptional all list

GATT DEFINES

Overview

Instructions

Application Example

Instructions

API Reference

Header File

- bt/host/bluedroid/api/include/api/esp_gatt_defs.h

Unions

union esp_gatt_rsp_t

#include <esp_gatt_defs.h> GATT remote read request response type.

Public Members

esp_gatt_value_t **attr_value**

Gatt attribute structure

uint16_t **handle**

Gatt attribute handle

Structures

struct esp_gatt_id_t

Gatt id, include uuid and instance id.

Public Members

esp_bt_uuid_t **uuid**

UUID

uint8_t **inst_id**

Instance id

struct esp_gatt_srvc_id_t

Gatt service id, include id (uuid and instance id) and primary flag.

Public Members

esp_gatt_id_t **id**

Gatt id, include uuid and instance

bool **is_primary**

This service is primary or not

struct esp_attr_desc_t

Attribute description (used to create database)

Public Members

uint16_t **uuid_length**

UUID length

uint8_t ***uuid_p**

UUID value

uint16_t **perm**

Attribute permission

uint16_t **max_length**

Maximum length of the element

uint16_t **length**

Current length of the element

uint8_t ***value**

Element value array

struct esp_attr_control_t

attribute auto response flag

Public Members

uint8_t **auto_rsp**

if auto_rsp set to ESP_GATT_RSP_BY_APP, means the response of Write/Read operation will by replied by application. if auto_rsp set to ESP_GATT_AUTO_RSP, means the response of Write/Read operation will be replied by GATT stack automatically.

struct esp_gatts_attr_db_t

attribute type added to the gatt server database

Public Members

esp_attr_control_t **attr_control**

The attribute control type

esp_attr_desc_t **att_desc**

The attribute type

struct esp_attr_value_t

set the attribute value type

Public Members

uint16_t **attr_max_len**
attribute max value length

uint16_t **attr_len**
attribute current value length

uint8_t ***attr_value**
the pointer to attribute value

struct esp_gatts_incl_svc_desc_t
Gatt include service entry element.

Public Members

uint16_t **start_hdl**
Gatt start handle value of included service

uint16_t **end_hdl**
Gatt end handle value of included service

uint16_t **uuid**
Gatt attribute value UUID of included service

struct esp_gatts_incl128_svc_desc_t
Gatt include 128 bit service entry element.

Public Members

uint16_t **start_hdl**
Gatt start handle value of included 128 bit service

uint16_t **end_hdl**
Gatt end handle value of included 128 bit service

struct esp_gatt_value_t
Gatt attribute value.

Public Members

uint8_t **value**[**ESP_GATT_MAX_ATTR_LEN**]
Gatt attribute value

uint16_t **handle**
Gatt attribute handle

uint16_t **offset**

Gatt attribute value offset

uint16_t **len**

Gatt attribute value length

uint8_t **auth_req**

Gatt authentication request

struct esp_gatt_conn_params_t

Connection parameters information.

Public Members

uint16_t **interval**

connection interval

uint16_t **latency**

Slave latency for the connection in number of connection events. Range: 0x0000 to 0x01F3

uint16_t **timeout**

Supervision timeout for the LE Link. Range: 0x000A to 0x0C80. Mandatory Range: 0x000A to 0x0C80 Time = N * 10 msec Time Range: 100 msec to 32 seconds

struct esp_gattc_multi_t

read multiple attribute

Public Members

uint8_t **num_attr**

The number of the attribute

uint16_t **handles**[**ESP_GATT_MAX_READ_MULTI_HANDLES**]

The handles list

struct esp_gattc_db_elem_t

data base attribute element

Public Members

esp_gatt_db_attr_type_t **type**

The attribute type

uint16_t **attribute_handle**

The attribute handle, it’ s valid for all of the type

uint16_t **start_handle**

The service start handle, it' s valid only when the type = ESP_GATT_DB_PRIMARY_SERVICE or ESP_GATT_DB_SECONDARY_SERVICE

uint16_t **end_handle**

The service end handle, it's valid only when the type = ESP_GATT_DB_PRIMARY_SERVICE or ESP_GATT_DB_SECONDARY_SERVICE

esp_gatt_char_prop_t **properties**

The characteristic properties, it' s valid only when the type = ESP_GATT_DB_CHARACTERISTIC

esp_bt_uuid_t **uuid**

The attribute uuid, it' s valid for all of the type

struct esp_gattc_service_elem_t

service element

Public Members

bool **is_primary**

The service flag, true if the service is primary service, else is secondary service

uint16_t **start_handle**

The start handle of the service

uint16_t **end_handle**

The end handle of the service

esp_bt_uuid_t **uuid**

The uuid of the service

struct esp_gattc_char_elem_t

characteristic element

Public Members

uint16_t **char_handle**

The characteristic handle

esp_gatt_char_prop_t **properties**

The characteristic properties

esp_bt_uuid_t **uuid**

The characteristic uuid

struct esp_gattc_descr_elem_t

descriptor element

Public Members

uint16_t **handle**

The characteristic descriptor handle

esp_bt_uuid_t **uuid**

The characteristic descriptor uuid

struct esp_gattc_incl_svc_elem_t

include service element

Public Members

uint16_t **handle**

The include service current attribute handle

uint16_t **incl_srvc_s_handle**

The start handle of the service which has been included

uint16_t **incl_srvc_e_handle**

The end handle of the service which has been included

esp_bt_uuid_t **uuid**

The include service uuid

Macros

ESP_GATT_UUID_IMMEDIATE_ALERT_SVC

All "ESP_GATT_UUID_xxx" is attribute types

ESP_GATT_UUID_LINK_LOSS_SVC

ESP_GATT_UUID_TX_POWER_SVC

ESP_GATT_UUID_CURRENT_TIME_SVC

ESP_GATT_UUID_REF_TIME_UPDATE_SVC

ESP_GATT_UUID_NEXT_DST_CHANGE_SVC

ESP_GATT_UUID_GLUCOSE_SVC

ESP_GATT_UUID_HEALTH_THERMOM_SVC

ESP_GATT_UUID_DEVICE_INFO_SVC

ESP_GATT_UUID_HEART_RATE_SVC

ESP_GATT_UUID_PHONE_ALERT_STATUS_SVC

ESP_GATT_UUID_BATTERY_SERVICE_SVC

ESP_GATT_UUID_BLOOD_PRESSURE_SVC

ESP_GATT_UUID_ALERT_NTF_SVC

ESP_GATT_UUID_HID_SVC

ESP_GATT_UUID_SCAN_PARAMETERS_SVC

ESP_GATT_UUID_RUNNING_SPEED_CADENCE_SVC

ESP_GATT_UUID_Automation_IO_SVC

ESP_GATT_UUID_CYCLING_SPEED_CADENCE_SVC

ESP_GATT_UUID_CYCLING_POWER_SVC

ESP_GATT_UUID_LOCATION_AND_NAVIGATION_SVC

ESP_GATT_UUID_ENVIRONMENTAL_SENSING_SVC

ESP_GATT_UUID_BODY_COMPOSITION

ESP_GATT_UUID_USER_DATA_SVC

ESP_GATT_UUID_WEIGHT_SCALE_SVC

ESP_GATT_UUID_BOND_MANAGEMENT_SVC

ESP_GATT_UUID_CONT_GLUCOSE_MONITOR_SVC

ESP_GATT_UUID_PRI_SERVICE

ESP_GATT_UUID_SEC_SERVICE

ESP_GATT_UUID_INCLUDE_SERVICE

ESP_GATT_UUID_CHAR_DECLARE

ESP_GATT_UUID_CHAR_EXT_PROP

ESP_GATT_UUID_CHAR_DESCRIPTION

ESP_GATT_UUID_CHAR_CLIENT_CONFIG

ESP_GATT_UUID_CHAR_SRVR_CONFIG

ESP_GATT_UUID_CHAR_PRESENT_FORMAT

ESP_GATT_UUID_CHAR_AGG_FORMAT

ESP_GATT_UUID_CHAR_VALID_RANGE

ESP_GATT_UUID_EXT_RPT_REF_DESCR

ESP_GATT_UUID_RPT_REF_DESCR

ESP_GATT_UUID_NUM_DIGITALS_DESCR

ESP_GATT_UUID_VALUE_TRIGGER_DESCR

ESP_GATT_UUID_ENV_SENSING_CONFIG_DESCR

ESP_GATT_UUID_ENV_SENSING_MEASUREMENT_DESCR

ESP_GATT_UUID_ENV_SENSING_TRIGGER_DESCR

ESP_GATT_UUID_TIME_TRIGGER_DESCR

ESP_GATT_UUID_GAP_DEVICE_NAME

ESP_GATT_UUID_GAP_ICON

ESP_GATT_UUID_GAP_PREF_CONN_PARAM

ESP_GATT_UUID_GAP_CENTRAL_ADDR_RESOL

ESP_GATT_UUID_GATT_SRV_CHGD

ESP_GATT_UUID_ALERT_LEVEL

ESP_GATT_UUID_TX_POWER_LEVEL

ESP_GATT_UUID_CURRENT_TIME

ESP_GATT_UUID_LOCAL_TIME_INFO

ESP_GATT_UUID_REF_TIME_INFO

ESP_GATT_UUID_NW_STATUS

ESP_GATT_UUID_NW_TRIGGER

ESP_GATT_UUID_ALERT_STATUS

ESP_GATT_UUID_RINGER_CP

ESP_GATT_UUID_RINGER_SETTING

ESP_GATT_UUID_GM_MEASUREMENT

ESP_GATT_UUID_GM_CONTEXT

ESP_GATT_UUID_GM_CONTROL_POINT

ESP_GATT_UUID_GM_FEATURE

ESP_GATT_UUID_SYSTEM_ID

ESP_GATT_UUID_MODEL_NUMBER_STR

ESP_GATT_UUID_SERIAL_NUMBER_STR

ESP_GATT_UUID_FW_VERSION_STR

ESP_GATT_UUID_HW_VERSION_STR

ESP_GATT_UUID_SW_VERSION_STR

ESP_GATT_UUID_MANU_NAME

ESP_GATT_UUID_IEEE_DATA

ESP_GATT_UUID_PNP_ID

ESP_GATT_UUID_HID_INFORMATION

ESP_GATT_UUID_HID_REPORT_MAP

ESP_GATT_UUID_HID_CONTROL_POINT

ESP_GATT_UUID_HID_REPORT

ESP_GATT_UUID_HID_PROTO_MODE

ESP_GATT_UUID_HID_BT_KB_INPUT

ESP_GATT_UUID_HID_BT_KB_OUTPUT

ESP_GATT_UUID_HID_BT_MOUSE_INPUT

ESP_GATT_HEART_RATE_MEAS
: Heart Rate Measurement.

ESP_GATT_BODY_SENSOR_LOCATION
: Body Sensor Location.

ESP_GATT_HEART_RATE_CNTL_POINT
: Heart Rate Control Point.

ESP_GATT_UUID_BATTERY_LEVEL

ESP_GATT_UUID_SC_CONTROL_POINT

ESP_GATT_UUID_SENSOR_LOCATION

ESP_GATT_UUID_RSC_MEASUREMENT

ESP_GATT_UUID_RSC_FEATURE

ESP_GATT_UUID_CSC_MEASUREMENT

ESP_GATT_UUID_CSC_FEATURE

ESP_GATT_UUID_SCAN_INT_WINDOW

ESP_GATT_UUID_SCAN_REFRESH

ESP_GATT_ILLEGAL_UUID
: GATT INVALID UUID.

ESP_GATT_ILLEGAL_HANDLE
: GATT INVALID HANDLE.

ESP_GATT_ATTR_HANDLE_MAX
: GATT attribute max handle.

ESP_GATT_MAX_READ_MULTI_HANDLES

ESP_GATT_PERM_READ
Attribute permissions.

ESP_GATT_PERM_READ_ENCRYPTED

ESP_GATT_PERM_READ_ENC_MITM

ESP_GATT_PERM_WRITE

ESP_GATT_PERM_WRITE_ENCRYPTED

ESP_GATT_PERM_WRITE_ENC_MITM

ESP_GATT_PERM_WRITE_SIGNED

ESP_GATT_PERM_WRITE_SIGNED_MITM

ESP_GATT_CHAR_PROP_BIT_BROADCAST

ESP_GATT_CHAR_PROP_BIT_READ

ESP_GATT_CHAR_PROP_BIT_WRITE_NR

ESP_GATT_CHAR_PROP_BIT_WRITE

ESP_GATT_CHAR_PROP_BIT_NOTIFY

ESP_GATT_CHAR_PROP_BIT_INDICATE

ESP_GATT_CHAR_PROP_BIT_AUTH

ESP_GATT_CHAR_PROP_BIT_EXT_PROP

ESP_GATT_MAX_ATTR_LEN
GATT maximum attribute length.

ESP_GATT_RSP_BY_APP

ESP_GATT_AUTO_RSP

ESP_GATT_IF_NONE
If callback report gattc_if/gatts_if as this macro, means this event is not correspond to any app

Type Definitions

typedef uint16_t **esp_gatt_perm_t**

typedef uint8_t **esp_gatt_char_prop_t**

typedef uint8_t **esp_gatt_if_t**
Gatt interface type, different application on GATT client use different gatt_if

Enumerations

enum esp_gatt_prep_write_type

Attribute write data type from the client.

Values:

ESP_GATT_PREP_WRITE_CANCEL = 0x00

Prepare write cancel

ESP_GATT_PREP_WRITE_EXEC = 0x01

Prepare write execute

enum esp_gatt_status_t

GATT success code and error codes.

Values:

ESP_GATT_OK = 0x0

ESP_GATT_INVALID_HANDLE = 0x01

ESP_GATT_READ_NOT_PERMIT = 0x02

ESP_GATT_WRITE_NOT_PERMIT = 0x03

ESP_GATT_INVALID_PDU = 0x04

ESP_GATT_INSUF_AUTHENTICATION = 0x05

ESP_GATT_REQ_NOT_SUPPORTED = 0x06

ESP_GATT_INVALID_OFFSET = 0x07

ESP_GATT_INSUF_AUTHORIZATION = 0x08

ESP_GATT_PREPARE_Q_FULL = 0x09

ESP_GATT_NOT_FOUND = 0x0a

ESP_GATT_NOT_LONG = 0x0b

ESP_GATT_INSUF_KEY_SIZE = 0x0c

ESP_GATT_INVALID_ATTR_LEN = 0x0d

ESP_GATT_ERR_UNLIKELY = 0x0e

ESP_GATT_INSUF_ENCRYPTION = 0x0f

ESP_GATT_UNSUPPORT_GRP_TYPE = 0x10

ESP_GATT_INSUF_RESOURCE = 0x11

ESP_GATT_NO_RESOURCES = 0x80

ESP_GATT_INTERNAL_ERROR = 0x81

ESP_GATT_WRONG_STATE = 0x82

ESP_GATT_DB_FULL = 0x83

ESP_GATT_BUSY = 0x84

ESP_GATT_ERROR = 0x85

ESP_GATT_CMD_STARTED = 0x86

ESP_GATT_ILLEGAL_PARAMETER = 0x87

ESP_GATT_PENDING = 0x88

ESP_GATT_AUTH_FAIL = 0x89

ESP_GATT_MORE = 0x8a

ESP_GATT_INVALID_CFG = 0x8b

ESP_GATT_SERVICE_STARTED = 0x8c

ESP_GATT_ENCRYPED_MITM = ESP_GATT_OK

ESP_GATT_ENCRYPED_NO_MITM = 0x8d

ESP_GATT_NOT_ENCRYPTED = 0x8e

ESP_GATT_CONGESTED = 0x8f

ESP_GATT_DUP_REG = 0x90

ESP_GATT_ALREADY_OPEN = 0x91

ESP_GATT_CANCEL = 0x92

ESP_GATT_STACK_RSP = 0xe0

ESP_GATT_APP_RSP = 0xe1

ESP_GATT_UNKNOWN_ERROR = 0xef

ESP_GATT_CCC_CFG_ERR = 0xfd

ESP_GATT_PRC_IN_PROGRESS = 0xfe

ESP_GATT_OUT_OF_RANGE = 0xff

enum esp_gatt_conn_reason_t

Gatt Connection reason enum.

Values:

ESP_GATT_CONN_UNKNOWN = 0

Gatt connection unknown

ESP_GATT_CONN_L2C_FAILURE = 1

General L2cap failure

ESP_GATT_CONN_TIMEOUT = 0x08
Connection timeout

ESP_GATT_CONN_TERMINATE_PEER_USER = 0x13
Connection terminate by peer user

ESP_GATT_CONN_TERMINATE_LOCAL_HOST = 0x16
Connection terminated by local host

ESP_GATT_CONN_FAIL_ESTABLISH = 0x3e
Connection fail to establish

ESP_GATT_CONN_LMP_TIMEOUT = 0x22
Connection fail for LMP response tout

ESP_GATT_CONN_CONN_CANCEL = 0x0100
L2CAP connection cancelled

ESP_GATT_CONN_NONE = 0x0101
No connection to cancel

enum esp_gatt_auth_req_t
Gatt authentication request type.

Values:

ESP_GATT_AUTH_REQ_NONE = 0

ESP_GATT_AUTH_REQ_NO_MITM = 1

ESP_GATT_AUTH_REQ_MITM = 2

ESP_GATT_AUTH_REQ_SIGNED_NO_MITM = 3

ESP_GATT_AUTH_REQ_SIGNED_MITM = 4

enum esp_service_source_t
Values:

ESP_GATT_SERVICE_FROM_REMOTE_DEVICE = 0

ESP_GATT_SERVICE_FROM_NVS_FLASH = 1

ESP_GATT_SERVICE_FROM_UNKNOWN = 2

enum esp_gatt_write_type_t
Gatt write type.

Values:

ESP_GATT_WRITE_TYPE_NO_RSP = 1
Gatt write attribute need no response

`ESP_GATT_WRITE_TYPE_RSP`
Gatt write attribute need remote response

`enum esp_gatt_db_attr_type_t`
the type of attribute element

Values:

`ESP_GATT_DB_PRIMARY_SERVICE`
Gattc primary service attribute type in the cache

`ESP_GATT_DB_SECONDARY_SERVICE`
Gattc secondary service attribute type in the cache

`ESP_GATT_DB_CHARACTERISTIC`
Gattc characteristic attribute type in the cache

`ESP_GATT_DB_DESCRIPTOR`
Gattc characteristic descriptor attribute type in the cache

`ESP_GATT_DB_INCLUDED_SERVICE`
Gattc include service attribute type in the cache

`ESP_GATT_DB_ALL`
Gattc all the attribute (primary service & secondary service & include service & char & descriptor) type in the cache

GATT SERVER API

Overview

Instructions

Application Example

Check bluetooth/bluedroid/ble folder in ESP-IDF examples, which contains the following demos and their tutorials:

- This is a GATT sever demo and its tutorial. This demo creates a GATT service with an attribute table, which releases the user from adding attributes one by one. This is the recommended method of adding attributes.
  - bluetooth/bluedroid/ble/gatt_server_service_table
  - GATT Server Service Table Example Walkthrough
- This is a GATT server demo and its tutorial. This demo creates a GATT service by adding attributes one by one as defined by Bluedroid. The recommended method of adding attributes is presented in example above.

  - bluetooth/bluedroid/ble/gatt_server
  - GATT Server Example Walkthrough

- This is a BLE SPP-Like demo. This demo, which acts as a GATT server, can receive data from UART and then send the data to the peer device automatically.
  - bluetooth/bluedroid/ble/ble_spp_server

API Reference

Header File

- bt/host/bluedroid/api/include/api/esp_gatts_api.h

Functions

esp_err_t **esp_ble_gatts_register_callback**(*esp_gatts_cb_t callback*)

This function is called to register application callbacks with BTA GATTS module.

Return

- ESP_OK : success
- other : failed

esp_err_t **esp_ble_gatts_app_register**(uint16_t *app_id*)

This function is called to register application identifier.

Return

- ESP_OK : success
- other : failed

esp_err_t **esp_ble_gatts_app_unregister**(*esp_gatt_if_t gatts_if*)

unregister with GATT Server.

Return

- ESP_OK : success
- other : failed

Parameters

- `gatts_if`: GATT server access interface

esp_err_t **esp_ble_gatts_create_service**(*esp_gatt_if_t gatts_if*, *esp_gatt_srvc_id_t* ***service_id*, uint16_t *num_handle*)

Create a service. When service creation is done, a callback event BTA_GATTS_CREATE_SRVC_EVT is called to report status and service ID to the profile. The service ID obtained in the callback function needs to be used when adding included service and characteristics/descriptors into the service.

Return

- ESP_OK : success
- other : failed

Parameters

- `gatts_if`: GATT server access interface
- `service_id`: service ID.
- `num_handle`: number of handle requested for this service.

esp_err_t **esp_ble_gatts_create_attr_tab**(**const** *esp_gatts_attr_db_t* ***gatts_attr_db*, *esp_gatt_if_t gatts_if*, uint8_t *max_nb_attr*, uint8_t *srvc_inst_id*)

Create a service attribute tab.

Return

- ESP_OK : success
- other : failed

Parameters

- `gatts_attr_db`: the pointer to the service attr tab
- `gatts_if`: GATT server access interface
- `max_nb_attr`: the number of attribute to be added to the service database.
- `srvc_inst_id`: the instance id of the service

esp_err_t **esp_ble_gatts_add_included_service**(uint16_t *service_handle*, uint16_t *included_service_handle*)

This function is called to add an included service. This function have to be called between 'esp_ble_gatts_create_service' and 'esp_ble_gatts_add_char' . After included service is included, a callback event BTA_GATTS_ADD_INCL_SRVC_EVT is reported the included service ID.

Return

- ESP_OK : success
- other : failed

Parameters

- `service_handle`: service handle to which this included service is to be added.
- `included_service_handle`: the service ID to be included.

esp_err_t **esp_ble_gatts_add_char**(uint16_t *service_handle*, *esp_bt_uuid_t* **char_uuid*, *esp_gatt_perm_t* *perm*, *esp_gatt_char_prop_t* *property*, *esp_attr_value_t* **char_val*, *esp_attr_control_t* **control*)

This function is called to add a characteristic into a service.

Return

- ESP_OK : success
- other : failed

Parameters

- `service_handle`: service handle to which this included service is to be added.
- `char_uuid`: : Characteristic UUID.
- `perm`: : Characteristic value declaration attribute permission.
- `property`: : Characteristic Properties
- `char_val`: : Characteristic value
- `control`: : attribute response control byte

esp_err_t **esp_ble_gatts_add_char_descr**(uint16_t *service_handle*, *esp_bt_uuid_t* **descr_uuid*, *esp_gatt_perm_t* *perm*, *esp_attr_value_t* **char_descr_val*, *esp_attr_control_t* **control*)

This function is called to add characteristic descriptor. When it' s done, a callback event BTA_GATTS_ADD_DESCR_EVT is called to report the status and an ID number for this descriptor.

Return

- ESP_OK : success
- other : failed

Parameters

- `service_handle`: service handle to which this characteristic descriptor is to be added.
- `perm`: descriptor access permission.
- `descr_uuid`: descriptor UUID.
- `char_descr_val`: : Characteristic descriptor value
- `control`: : attribute response control byte

esp_err_t **esp_ble_gatts_delete_service**(uint16_t *service_handle*)

This function is called to delete a service. When this is done, a callback event BTA_GATTS_DELETE_EVT is report with the status.

Return

- ESP_OK : success
- other : failed

Parameters

- `service_handle`: service_handle to be deleted.

esp_err_t **esp_ble_gatts_start_service**(uint16_t *service_handle*)

This function is called to start a service.

Return

- ESP_OK : success
- other : failed

Parameters

- `service_handle`: the service handle to be started.

esp_err_t **esp_ble_gatts_stop_service**(uint16_t *service_handle*)

This function is called to stop a service.

Return

- ESP_OK : success
- other : failed

Parameters

- `service_handle`: - service to be topped.

esp_err_t **esp_ble_gatts_send_indicate**(*esp_gatt_if_t gatts_if*, uint16_t *conn_id*, uint16_t *attr_handle*, uint16_t *value_len*, uint8_t **value*, bool *need_confirm*)

Send indicate or notify to GATT client. Set param need_confirm as false will send notification, otherwise indication.

Return

- ESP_OK : success
- other : failed

Parameters

- `gatts_if`: GATT server access interface
- `conn_id`: - connection id to indicate.
- `attr_handle`: - attribute handle to indicate.
- `value_len`: - indicate value length.
- `value`: value to indicate.
- `need_confirm`: - Whether a confirmation is required. false sends a GATT notification, true sends a GATT indication.

esp_err_t **esp_ble_gatts_send_response**(*esp_gatt_if_t gatts_if*, uint16_t *conn_id*, uint32_t *trans_id*, *esp_gatt_status_t status*, *esp_gatt_rsp_t* **rsp*)

This function is called to send a response to a request.

Return

- ESP_OK : success
- other : failed

Parameters

- `gatts_if`: GATT server access interface
- `conn_id`: - connection identifier.
- `trans_id`: - transfer id
- `status`: - response status
- `rsp`: - response data.

esp_err_t **esp_ble_gatts_set_attr_value**(uint16_t *attr_handle*, uint16_t *length*, **const** uint8_t **value*)

This function is called to set the attribute value by the application.

Return

- ESP_OK : success
- other : failed

Parameters

- `attr_handle`: the attribute handle which to be set
- `length`: the value length
- `value`: the pointer to the attribute value

esp_gatt_status_t **esp_ble_gatts_get_attr_value**(uint16_t *attr_handle*, uint16_t **length*, **const** uint8_t ***value*)

Retrieve attribute value.

Return

- ESP_GATT_OK : success
- other : failed

Parameters

- `attr_handle`: Attribute handle.
- `length`: pointer to the attribute value length
- `value`: Pointer to attribute value payload, the value cannot be modified by user

esp_err_t **esp_ble_gatts_open**(*esp_gatt_if_t* *gatts_if*, *esp_bd_addr_t* *remote_bda*, bool *is_direct*)

Open a direct open connection or add a background auto connection.

Return

- ESP_OK : success
- other : failed

Parameters

- `gatts_if`: GATT server access interface
- `remote_bda`: remote device bluetooth device address.
- `is_direct`: direct connection or background auto connection

esp_err_t **esp_ble_gatts_close**(*esp_gatt_if_t* *gatts_if*, uint16_t *conn_id*)

Close a connection a remote device.

Return

- ESP_OK : success
- other : failed

Parameters

- `gatts_if`: GATT server access interface
- `conn_id`: connection ID to be closed.

esp_err_t **esp_ble_gatts_send_service_change_indication**(*esp_gatt_if_t* *gatts_if*, *esp_bd_addr_t* *remote_bda*)

Send service change indication.

Return

- ESP_OK : success
- other : failed

Parameters

- `gatts_if`: GATT server access interface
- `remote_bda`: remote device bluetooth device address. If remote_bda is NULL then it will send service change indication to all the connected devices and if not then to a specific device

Unions

`union esp_ble_gatts_cb_param_t`

#include <esp_gatts_api.h> Gatt server callback parameters union.

Public Members

`struct` *esp_ble_gatts_cb_param_t*::*gatts_reg_evt_param* `reg`

Gatt server callback param of ESP_GATTS_REG_EVT

`struct` *esp_ble_gatts_cb_param_t*::*gatts_read_evt_param* `read`

Gatt server callback param of ESP_GATTS_READ_EVT

`struct` *esp_ble_gatts_cb_param_t*::*gatts_write_evt_param* `write`

Gatt server callback param of ESP_GATTS_WRITE_EVT

`struct` *esp_ble_gatts_cb_param_t*::*gatts_exec_write_evt_param* `exec_write`

Gatt server callback param of ESP_GATTS_EXEC_WRITE_EVT

`struct` *esp_ble_gatts_cb_param_t*::*gatts_mtu_evt_param* `mtu`

Gatt server callback param of ESP_GATTS_MTU_EVT

`struct` *esp_ble_gatts_cb_param_t*::*gatts_conf_evt_param* `conf`

Gatt server callback param of ESP_GATTS_CONF_EVT (confirm)

`struct` *esp_ble_gatts_cb_param_t*::*gatts_create_evt_param* `create`

Gatt server callback param of ESP_GATTS_CREATE_EVT

`struct` *esp_ble_gatts_cb_param_t*::*gatts_add_incl_srvc_evt_param* `add_incl_srvc`

Gatt server callback param of ESP_GATTS_ADD_INCL_SRVC_EVT

`struct` *esp_ble_gatts_cb_param_t*::*gatts_add_char_evt_param* `add_char`

Gatt server callback param of ESP_GATTS_ADD_CHAR_EVT

`struct` *esp_ble_gatts_cb_param_t*::*gatts_add_char_descr_evt_param* `add_char_descr`

Gatt server callback param of ESP_GATTS_ADD_CHAR_DESCR_EVT

struct *esp_ble_gatts_cb_param_t*::*gatts_delete_evt_param* **del**

Gatt server callback param of ESP_GATTS_DELETE_EVT

struct *esp_ble_gatts_cb_param_t*::*gatts_start_evt_param* **start**

Gatt server callback param of ESP_GATTS_START_EVT

struct *esp_ble_gatts_cb_param_t*::*gatts_stop_evt_param* **stop**

Gatt server callback param of ESP_GATTS_STOP_EVT

struct *esp_ble_gatts_cb_param_t*::*gatts_connect_evt_param* **connect**

Gatt server callback param of ESP_GATTS_CONNECT_EVT

struct *esp_ble_gatts_cb_param_t*::*gatts_disconnect_evt_param* **disconnect**

Gatt server callback param of ESP_GATTS_DISCONNECT_EVT

struct *esp_ble_gatts_cb_param_t*::*gatts_open_evt_param* **open**

Gatt server callback param of ESP_GATTS_OPEN_EVT

struct *esp_ble_gatts_cb_param_t*::*gatts_cancel_open_evt_param* **cancel_open**

Gatt server callback param of ESP_GATTS_CANCEL_OPEN_EVT

struct *esp_ble_gatts_cb_param_t*::*gatts_close_evt_param* **close**

Gatt server callback param of ESP_GATTS_CLOSE_EVT

struct *esp_ble_gatts_cb_param_t*::*gatts_congest_evt_param* **congest**

Gatt server callback param of ESP_GATTS_CONGEST_EVT

struct *esp_ble_gatts_cb_param_t*::*gatts_rsp_evt_param* **rsp**

Gatt server callback param of ESP_GATTS_RESPONSE_EVT

struct *esp_ble_gatts_cb_param_t*::*gatts_add_attr_tab_evt_param* **add_attr_tab**

Gatt server callback param of ESP_GATTS_CREAT_ATTR_TAB_EVT

struct *esp_ble_gatts_cb_param_t*::*gatts_set_attr_val_evt_param* **set_attr_val**

Gatt server callback param of ESP_GATTS_SET_ATTR_VAL_EVT

struct *esp_ble_gatts_cb_param_t*::*gatts_send_service_change_evt_param* **service_change**

Gatt server callback param of ESP_GATTS_SEND_SERVICE_CHANGE_EVT

struct gatts_add_attr_tab_evt_param

#include <esp_gatts_api.h> ESP_GATTS_CREAT_ATTR_TAB_EVT.

Public Members

esp_gatt_status_t **status**

Operation status

esp_bt_uuid_t **svc_uuid**

Service uuid type

uint8_t **svc_inst_id**

Service id

uint16_t **num_handle**

The number of the attribute handle to be added to the gatts database

uint16_t ***handles**

The number to the handles

struct gatts_add_char_descr_evt_param

#include <esp_gatts_api.h> ESP_GATTS_ADD_CHAR_DESCR_EVT.

Public Members

esp_gatt_status_t **status**

Operation status

uint16_t **attr_handle**

Descriptor attribute handle

uint16_t **service_handle**

Service attribute handle

esp_bt_uuid_t **descr_uuid**

Characteristic descriptor uuid

struct gatts_add_char_evt_param

#include <esp_gatts_api.h> ESP_GATTS_ADD_CHAR_EVT.

Public Members

esp_gatt_status_t **status**

Operation status

uint16_t **attr_handle**

Characteristic attribute handle

uint16_t **service_handle**

Service attribute handle

esp_bt_uuid_t **char_uuid**

Characteristic uuid

struct gatts_add_incl_srvc_evt_param

#include <esp_gatts_api.h> ESP_GATTS_ADD_INCL_SRVC_EVT.

Public Members

esp_gatt_status_t **status**
Operation status

uint16_t **attr_handle**
Included service attribute handle

uint16_t **service_handle**
Service attribute handle

struct gatts_cancel_open_evt_param
#include <esp_gatts_api.h> ESP_GATTS_CANCEL_OPEN_EVT.

Public Members

esp_gatt_status_t **status**
Operation status

struct gatts_close_evt_param
#include <esp_gatts_api.h> ESP_GATTS_CLOSE_EVT.

Public Members

esp_gatt_status_t **status**
Operation status

uint16_t **conn_id**
Connection id

struct gatts_conf_evt_param
#include <esp_gatts_api.h> ESP_GATTS_CONF_EVT.

Public Members

esp_gatt_status_t **status**
Operation status

uint16_t **conn_id**
Connection id

uint16_t **handle**
attribute handle

uint16_t **len**
The indication or notification value length, len is valid when send notification or indication failed

uint8_t *value
The indication or notification value , value is valid when send notification or indication failed

struct gatts_congest_evt_param
#include <esp_gatts_api.h> ESP_GATTS_LISTEN_EVT.

ESP_GATTS_CONGEST_EVT

Public Members

uint16_t conn_id
Connection id

bool congested
Congested or not

struct gatts_connect_evt_param
#include <esp_gatts_api.h> ESP_GATTS_CONNECT_EVT.

Public Members

uint16_t conn_id
Connection id

esp_bd_addr_t remote_bda
Remote bluetooth device address

esp_gatt_conn_params_t conn_params
current Connection parameters

struct gatts_create_evt_param
#include <esp_gatts_api.h> ESP_GATTS_UNREG_EVT.

ESP_GATTS_CREATE_EVT

Public Members

esp_gatt_status_t status
Operation status

uint16_t service_handle
Service attribute handle

esp_gatt_srvc_id_t service_id
Service id, include service uuid and other information

struct gatts_delete_evt_param
#include <esp_gatts_api.h> ESP_GATTS_DELETE_EVT.

Public Members

esp_gatt_status_t **status**
Operation status

uint16_t **service_handle**
Service attribute handle

struct gatts_disconnect_evt_param
#include <esp_gatts_api.h> ESP_GATTS_DISCONNECT_EVT.

Public Members

uint16_t **conn_id**
Connection id

esp_bd_addr_t **remote_bda**
Remote bluetooth device address

esp_gatt_conn_reason_t **reason**
Indicate the reason of disconnection

struct gatts_exec_write_evt_param
#include <esp_gatts_api.h> ESP_GATTS_EXEC_WRITE_EVT.

Public Members

uint16_t **conn_id**
Connection id

uint32_t **trans_id**
Transfer id

esp_bd_addr_t **bda**
The bluetooth device address which been written

uint8_t **exec_write_flag**
Execute write flag

struct gatts_mtu_evt_param
#include <esp_gatts_api.h> ESP_GATTS_MTU_EVT.

Public Members

uint16_t **conn_id**
Connection id

uint16_t **mtu**

MTU size

struct gatts_open_evt_param

#include <esp_gatts_api.h> ESP_GATTS_OPEN_EVT.

Public Members

esp_gatt_status_t **status**

Operation status

struct gatts_read_evt_param

#include <esp_gatts_api.h> ESP_GATTS_READ_EVT.

Public Members

uint16_t **conn_id**

Connection id

uint32_t **trans_id**

Transfer id

esp_bd_addr_t **bda**

The bluetooth device address which been read

uint16_t **handle**

The attribute handle

uint16_t **offset**

Offset of the value, if the value is too long

bool **is_long**

The value is too long or not

bool **need_rsp**

The read operation need to do response

struct gatts_reg_evt_param

#include <esp_gatts_api.h> ESP_GATTS_REG_EVT.

Public Members

esp_gatt_status_t **status**

Operation status

uint16_t **app_id**

Application id which input in register API

struct gatts_rsp_evt_param

#include <esp_gatts_api.h> ESP_GATTS_RESPONSE_EVT.

Public Members

esp_gatt_status_t **status**

Operation status

uint16_t **handle**

Attribute handle which send response

struct gatts_send_service_change_evt_param

#include <esp_gatts_api.h> ESP_GATTS_SEND_SERVICE_CHANGE_EVT.

Public Members

esp_gatt_status_t **status**

Operation status

struct gatts_set_attr_val_evt_param

#include <esp_gatts_api.h> ESP_GATTS_SET_ATTR_VAL_EVT.

Public Members

uint16_t **srvc_handle**

The service handle

uint16_t **attr_handle**

The attribute handle

esp_gatt_status_t **status**

Operation status

struct gatts_start_evt_param

#include <esp_gatts_api.h> ESP_GATTS_START_EVT.

Public Members

esp_gatt_status_t **status**

Operation status

uint16_t **service_handle**

Service attribute handle

struct gatts_stop_evt_param

#include <esp_gatts_api.h> ESP_GATTS_STOP_EVT.

Public Members

esp_gatt_status_t **status**
Operation status

uint16_t **service_handle**
Service attribute handle

struct gatts_write_evt_param
#include <esp_gatts_api.h> ESP_GATTS_WRITE_EVT.

Public Members

uint16_t **conn_id**
Connection id

uint32_t **trans_id**
Transfer id

esp_bd_addr_t **bda**
The bluetooth device address which been written

uint16_t **handle**
The attribute handle

uint16_t **offset**
Offset of the value, if the value is too long

bool **need_rsp**
The write operation need to do response

bool **is_prep**
This write operation is prepare write

uint16_t **len**
The write attribute value length

uint8_t ***value**
The write attribute value

Macros

ESP_GATT_PREP_WRITE_CANCEL
Prepare write flag to indicate cancel prepare write

ESP_GATT_PREP_WRITE_EXEC
Prepare write flag to indicate execute prepare write

Type Definitions

typedef void (*esp_gatts_cb_t)(*esp_gatts_cb_event_t* event, *esp_gatt_if_t* gatts_if, *esp_ble_gatts_cb_param_t* *param)

GATT Server callback function type.

Parameters

- event: : Event type
- gatts_if: : GATT server access interface, normally different gatts_if correspond to different profile
- param: : Point to callback parameter, currently is union type

Enumerations

enum esp_gatts_cb_event_t

GATT Server callback function events.

Values:

ESP_GATTS_REG_EVT = 0

When register application id, the event comes

ESP_GATTS_READ_EVT = 1

When gatt client request read operation, the event comes

ESP_GATTS_WRITE_EVT = 2

When gatt client request write operation, the event comes

ESP_GATTS_EXEC_WRITE_EVT = 3

When gatt client request execute write, the event comes

ESP_GATTS_MTU_EVT = 4

When set mtu complete, the event comes

ESP_GATTS_CONF_EVT = 5

When receive confirm, the event comes

ESP_GATTS_UNREG_EVT = 6

When unregister application id, the event comes

ESP_GATTS_CREATE_EVT = 7

When create service complete, the event comes

ESP_GATTS_ADD_INCL_SRVC_EVT = 8

When add included service complete, the event comes

ESP_GATTS_ADD_CHAR_EVT = 9

When add characteristic complete, the event comes

ESP_GATTS_ADD_CHAR_DESCR_EVT = 10

When add descriptor complete, the event comes

ESP_GATTS_DELETE_EVT = 11

When delete service complete, the event comes

ESP_GATTS_START_EVT = 12

When start service complete, the event comes

ESP_GATTS_STOP_EVT = 13

When stop service complete, the event comes

ESP_GATTS_CONNECT_EVT = 14

When gatt client connect, the event comes

ESP_GATTS_DISCONNECT_EVT = 15

When gatt client disconnect, the event comes

ESP_GATTS_OPEN_EVT = 16

When connect to peer, the event comes

ESP_GATTS_CANCEL_OPEN_EVT = 17

When disconnect from peer, the event comes

ESP_GATTS_CLOSE_EVT = 18

When gatt server close, the event comes

ESP_GATTS_LISTEN_EVT = 19

When gatt listen to be connected the event comes

ESP_GATTS_CONGEST_EVT = 20

When congest happen, the event comes

ESP_GATTS_RESPONSE_EVT = 21

When gatt send response complete, the event comes

ESP_GATTS_CREAT_ATTR_TAB_EVT = 22

When gatt create table complete, the event comes

ESP_GATTS_SET_ATTR_VAL_EVT = 23

When gatt set attr value complete, the event comes

ESP_GATTS_SEND_SERVICE_CHANGE_EVT = 24

When gatt send service change indication complete, the event comes

GATT CLIENT API

Overview

Instructions

Application Example

Check bluetooth/bluedroid/ble folder in ESP-IDF examples, which contains the following demos and their tutorials:

- This is a GATT client demo and its tutorial. This demo can scan for devices, connect to the GATT server and discover its services.
  - bluetooth/bluedroid/ble/gatt_client
  - GATT Client Example Walkthrough
- This is a multiple connection demo and its tutorial. This demo can connect to multiple GATT server devices and discover their services.
  - bluetooth/bluedroid/ble/gattc_multi_connect
  - GATT Client Multi-connection Example Walkthrough
- This is a BLE SPP-Like demo. This demo, which acts as a GATT client, can receive data from UART and then send the data to the peer device automatically.
  - bluetooth/bluedroid/ble/ble_spp_client

API Reference

Header File

- bt/host/bluedroid/api/include/api/esp_gattc_api.h

Functions

esp_err_t **esp_ble_gattc_register_callback**(*esp_gattc_cb_t callback*)

This function is called to register application callbacks with GATTC module.

Return

- ESP_OK: success
- other: failed

Parameters

- `callback`: : pointer to the application callback function.

esp_err_t **esp_ble_gattc_app_register**(uint16_t *app_id*)

This function is called to register application callbacks with GATTC module.

Return

- ESP_OK: success
- other: failed

Parameters

- `app_id`: : Application Identify (UUID), for different application

esp_err_t **esp_ble_gattc_app_unregister**(*esp_gatt_if_t gattc_if*)

This function is called to unregister an application from GATTC module.

Return

- ESP_OK: success
- other: failed

Parameters

- `gattc_if`: Gatt client access interface.

esp_err_t **esp_ble_gattc_open**(*esp_gatt_if_t gattc_if*, *esp_bd_addr_t remote_bda*, *esp_ble_addr_type_t remote_addr_type*, bool *is_direct*)

Open a direct connection or add a background auto connection.

Return

- ESP_OK: success
- other: failed

Parameters

- `gattc_if`: Gatt client access interface.
- `remote_bda`: remote device bluetooth device address.
- `remote_addr_type`: remote device bluetooth device the address type.
- `is_direct`: direct connection or background auto connection

esp_err_t **esp_ble_gattc_close**(*esp_gatt_if_t gattc_if*, uint16_t *conn_id*)

Close the virtual connection to the GATT server. gattc may have multiple virtual GATT server connections when multiple app_id registered, this API only close one virtual GATT server connection. if there exist other virtual GATT server connections, it does not disconnect the physical connection. if you want to disconnect the physical connection directly, you can use esp_ble_gap_disconnect(esp_bd_addr_t remote_device).

Return

- ESP_OK: success
- other: failed

Parameters

- `gattc_if`: Gatt client access interface.
- `conn_id`: connection ID to be closed.

esp_err_t **esp_ble_gattc_send_mtu_req**(*esp_gatt_if_t gattc_if*, uint16_t *conn_id*)

Configure the MTU size in the GATT channel. This can be done only once per connection. Before using, use esp_ble_gatt_set_local_mtu() to configure the local MTU size.

Return

- ESP_OK: success
- other: failed

Parameters

- `gattc_if`: Gatt client access interface.
- `conn_id`: connection ID.

esp_err_t **esp_ble_gattc_search_service**(*esp_gatt_if_t gattc_if*, uint16_t *conn_id*, *esp_bt_uuid_t* **filter_uuid*)

This function is called to get service from local cache. This function report service search result by a callback event, and followed by a service search complete event.

Return

- ESP_OK: success
- other: failed

Parameters

- `gattc_if`: Gatt client access interface.
- `conn_id`: connection ID.
- `filter_uuid`: a UUID of the service application is interested in. If Null, discover for all services.

esp_gatt_status_t **esp_ble_gattc_get_service**(*esp_gatt_if_t gattc_if*, uint16_t *conn_id*, *esp_bt_uuid_t* **svc_uuid*, *esp_gattc_service_elem_t* **result*, uint16_t **count*, uint16_t *offset*)

Find all the service with the given service uuid in the gattc cache, if the svc_uuid is NULL, find all

the service. Note: It just get service from local cache, won't get from remote devices. If want to get it from remote device, need to used the esp_ble_gattc_search_service.

Return

- ESP_OK: success
- other: failed

Parameters

- `gattc_if`: Gatt client access interface.
- `conn_id`: connection ID which identify the server.
- `svc_uuid`: the pointer to the service uuid.
- `result`: The pointer to the service which has been found in the gattc cache.
- `count`: input the number of service want to find, it will output the number of service has been found in the gattc cache with the given service uuid.
- `offset`: Offset of the service position to get.

esp_gatt_status_t **esp_ble_gattc_get_all_char**(*esp_gatt_if_t gattc_if*, uint16_t *conn_id*, uint16_t *start_handle*, uint16_t *end_handle*, *esp_gattc_char_elem_t* **result*, uint16_t **count*, uint16_t *offset*)

Find all the characteristic with the given service in the gattc cache Note: It just get characteristic from local cache, won't get from remote devices.

Return

- ESP_OK: success
- other: failed

Parameters

- `gattc_if`: Gatt client access interface.
- `conn_id`: connection ID which identify the server.
- `start_handle`: the attribute start handle.
- `end_handle`: the attribute end handle
- `result`: The pointer to the characteristic in the service.
- `count`: input the number of characteristic want to find, it will output the number of characteristic has been found in the gattc cache with the given service.
- `offset`: Offset of the characteristic position to get.

esp_gatt_status_t **esp_ble_gattc_get_all_descr**(*esp_gatt_if_t gattc_if*, uint16_t *conn_id*, uint16_t *char_handle*, *esp_gattc_descr_elem_t* **result*, uint16_t **count*, uint16_t *offset*)

Find all the descriptor with the given characteristic in the gattc cache Note: It just get descriptor from local cache, won't get from remote devices.

Return

- ESP_OK: success
- other: failed

Parameters

- `gattc_if`: Gatt client access interface.
- `conn_id`: connection ID which identify the server.
- `char_handle`: the given characteristic handle
- `result`: The pointer to the descriptor in the characteristic.
- `count`: input the number of descriptor want to find, it will output the number of descriptor has been found in the gattc cache with the given characteristic.
- `offset`: Offset of the descriptor position to get.

esp_gatt_status_t **esp_ble_gattc_get_char_by_uuid**(*esp_gatt_if_t gattc_if*, uint16_t *conn_id*, uint16_t *start_handle*, uint16_t *end_handle*, *esp_bt_uuid_t char_uuid*, *esp_gattc_char_elem_t* **result*, uint16_t **count*)

Find the characteristic with the given characteristic uuid in the gattc cache Note: It just get characteristic from local cache, won't get from remote devices.

Return

- ESP_OK: success
- other: failed

Parameters

- `gattc_if`: Gatt client access interface.
- `conn_id`: connection ID which identify the server.
- `start_handle`: the attribute start handle
- `end_handle`: the attribute end handle
- `char_uuid`: the characteristic uuid
- `result`: The pointer to the characteristic in the service.

- `count`: input the number of characteristic want to find, it will output the number of characteristic has been found in the gattc cache with the given service.

esp_gatt_status_t **esp_ble_gattc_get_descr_by_uuid**(*esp_gatt_if_t gattc_if*, uint16_t *conn_id*, uint16_t *start_handle*, uint16_t *end_handle*, *esp_bt_uuid_t char_uuid*, *esp_bt_uuid_t descr_uuid*, *esp_gattc_descr_elem_t* **result*, uint16_t **count*)

Find the descriptor with the given characteristic uuid in the gattc cache Note: It just get descriptor from local cache, won't get from remote devices.

Return

- ESP_OK: success
- other: failed

Parameters

- `gattc_if`: Gatt client access interface.
- `conn_id`: connection ID which identify the server.
- `start_handle`: the attribute start handle
- `end_handle`: the attribute end handle
- `char_uuid`: the characteristic uuid.
- `descr_uuid`: the descriptor uuid.
- `result`: The pointer to the descriptor in the given characteristic.
- `count`: input the number of descriptor want to find, it will output the number of descriptor has been found in the gattc cache with the given characteristic.

esp_gatt_status_t **esp_ble_gattc_get_descr_by_char_handle**(*esp_gatt_if_t gattc_if*, uint16_t *conn_id*, uint16_t *char_handle*, *esp_bt_uuid_t descr_uuid*, *esp_gattc_descr_elem_t* **result*, uint16_t **count*)

Find the descriptor with the given characteristic handle in the gattc cache Note: It just get descriptor from local cache, won't get from remote devices.

Return

- ESP_OK: success
- other: failed

Parameters

- `gattc_if`: Gatt client access interface.
- `conn_id`: connection ID which identify the server.
- `char_handle`: the characteristic handle.
- `descr_uuid`: the descriptor uuid.
- `result`: The pointer to the descriptor in the given characteristic.
- `count`: input the number of descriptor want to find, it will output the number of descriptor has been found in the gattc cache with the given characteristic.

esp_gatt_status_t **esp_ble_gattc_get_include_service**(*esp_gatt_if_t gattc_if*, uint16_t *conn_id*, uint16_t *start_handle*, uint16_t *end_handle*, *esp_bt_uuid_t* **incl_uuid*, *esp_gattc_incl_svc_elem_t* **result*, uint16_t **count*)

Find the include service with the given service handle in the gattc cache Note: It just get include service from local cache, won't get from remote devices.

Return

- ESP_OK: success
- other: failed

Parameters

- `gattc_if`: Gatt client access interface.
- `conn_id`: connection ID which identify the server.
- `start_handle`: the attribute start handle
- `end_handle`: the attribute end handle
- `incl_uuid`: the include service uuid
- `result`: The pointer to the include service in the given service.
- `count`: input the number of include service want to find, it will output the number of include service has been found in the gattc cache with the given service.

esp_gatt_status_t **esp_ble_gattc_get_attr_count**(*esp_gatt_if_t gattc_if*, uint16_t *conn_id*, *esp_gatt_db_attr_type_t type*, uint16_t *start_handle*, uint16_t *end_handle*, uint16_t *char_handle*, uint16_t **count*)

Find the attribute count with the given service or characteristic in the gattc cache.

Return

- ESP_OK: success

- other: failed

Parameters

- `gattc_if`: Gatt client access interface.
- `conn_id`: connection ID which identify the server.
- `type`: the attribute type.
- `start_handle`: the attribute start handle, if the type is ESP_GATT_DB_DESCRIPTOR, this parameter should be ignore
- `end_handle`: the attribute end handle, if the type is ESP_GATT_DB_DESCRIPTOR, this parameter should be ignore
- `char_handle`: the characteristic handle, this parameter valid when the type is ESP_GATT_DB_DESCRIPTOR. If the type isn't ESP_GATT_DB_DESCRIPTOR, this parameter should be ignore.
- `count`: output the number of attribute has been found in the gattc cache with the given attribute type.

esp_gatt_status_t **esp_ble_gattc_get_db**(*esp_gatt_if_t gattc_if*, uint16_t *conn_id*, uint16_t *start_handle*, uint16_t *end_handle*, *esp_gattc_db_elem_t* **db*, uint16_t **count*)

This function is called to get the GATT database. Note: It just get attribute data base from local cache, won't get from remote devices.

Return

- ESP_OK: success
- other: failed

Parameters

- `gattc_if`: Gatt client access interface.
- `start_handle`: the attribute start handle
- `end_handle`: the attribute end handle
- `conn_id`: connection ID which identify the server.
- `db`: output parameter which will contain the GATT database copy. Caller is responsible for freeing it.
- `count`: number of elements in database.

esp_err_t **esp_ble_gattc_read_char**(*esp_gatt_if_t gattc_if*, uint16_t *conn_id*, uint16_t *handle*, *esp_gatt_auth_req_t auth_req*)

This function is called to read a service's characteristics of the given characteristic handle.

Return

- ESP_OK: success
- other: failed

Parameters

- `gattc_if`: Gatt client access interface.
- `conn_id`: : connection ID.
- `handle`: : characteritic handle to read.
- `auth_req`: : authenticate request type

esp_err_t **esp_ble_gattc_read_multiple**(*esp_gatt_if_t gattc_if*, uint16_t *conn_id*, *esp_gattc_multi_t* **read_multi*, *esp_gatt_auth_req_t auth_req*)

This function is called to read multiple characteristic or characteristic descriptors.

Return

- ESP_OK: success
- other: failed

Parameters

- `gattc_if`: Gatt client access interface.
- `conn_id`: : connection ID.
- `read_multi`: : pointer to the read multiple parameter.
- `auth_req`: : authenticate request type

esp_err_t **esp_ble_gattc_read_char_descr**(*esp_gatt_if_t gattc_if*, uint16_t *conn_id*, uint16_t *handle*, *esp_gatt_auth_req_t auth_req*)

This function is called to read a characteristics descriptor.

Return

- ESP_OK: success
- other: failed

Parameters

- `gattc_if`: Gatt client access interface.
- `conn_id`: : connection ID.
- `handle`: : descriptor handle to read.
- `auth_req`: : authenticate request type

esp_err_t **esp_ble_gattc_write_char**(*esp_gatt_if_t gattc_if*, uint16_t *conn_id*, uint16_t *handle*, uint16_t *value_len*, uint8_t **value*, *esp_gatt_write_type_t write_type*, *esp_gatt_auth_req_t auth_req*)

This function is called to write characteristic value.

Return

- ESP_OK: success
- other: failed

Parameters

- `gattc_if`: Gatt client access interface.
- `conn_id`: : connection ID.
- `handle`: : characteristic handle to write.
- `value_len`: length of the value to be written.
- `value`: : the value to be written.
- `write_type`: : the type of attribute write operation.
- `auth_req`: : authentication request.

esp_err_t **esp_ble_gattc_write_char_descr**(*esp_gatt_if_t gattc_if*, uint16_t *conn_id*, uint16_t *handle*, uint16_t *value_len*, uint8_t **value*, *esp_gatt_write_type_t write_type*, *esp_gatt_auth_req_t auth_req*)

This function is called to write characteristic descriptor value.

Return

- ESP_OK: success
- other: failed

Parameters

- `gattc_if`: Gatt client access interface.
- `conn_id`: : connection ID
- `handle`: : descriptor hadle to write.
- `value_len`: length of the value to be written.
- `value`: : the value to be written.
- `write_type`: : the type of attribute write operation.
- `auth_req`: : authentication request.

esp_err_t **esp_ble_gattc_prepare_write**(*esp_gatt_if_t gattc_if*, uint16_t *conn_id*, uint16_t *handle*, uint16_t *offset*, uint16_t *value_len*, uint8_t **value*, *esp_gatt_auth_req_t auth_req*)

This function is called to prepare write a characteristic value.

Return

- ESP_OK: success
- other: failed

Parameters

- `gattc_if`: Gatt client access interface.
- `conn_id`: : connection ID.
- `handle`: : characteristic handle to prepare write.
- `offset`: : offset of the write value.
- `value_len`: length of the value to be written.
- `value`: : the value to be written.
- `auth_req`: : authentication request.

esp_err_t **esp_ble_gattc_prepare_write_char_descr**(*esp_gatt_if_t gattc_if*, uint16_t *conn_id*, uint16_t *handle*, uint16_t *offset*, uint16_t *value_len*, uint8_t **value*, *esp_gatt_auth_req_t auth_req*)

This function is called to prepare write a characteristic descriptor value.

Return

- ESP_OK: success
- other: failed

Parameters

- `gattc_if`: Gatt client access interface.
- `conn_id`: : connection ID.
- `handle`: : characteristic descriptor handle to prepare write.
- `offset`: : offset of the write value.
- `value_len`: length of the value to be written.
- `value`: : the value to be written.
- `auth_req`: : authentication request.

esp_err_t **esp_ble_gattc_execute_write**(*esp_gatt_if_t gattc_if*, uint16_t *conn_id*, bool *is_execute*)

This function is called to execute write a prepare write sequence.

Return

- ESP_OK: success
- other: failed

Parameters

- `gattc_if`: Gatt client access interface.
- `conn_id`: : connection ID.
- `is_execute`: : execute or cancel.

esp_err_t **esp_ble_gattc_register_for_notify**(*esp_gatt_if_t gattc_if*, *esp_bd_addr_t server_bda*, uint16_t *handle*)

This function is called to register for notification of a service.

Return

- ESP_OK: registration succeeds
- other: failed

Parameters

- `gattc_if`: Gatt client access interface.
- `server_bda`: : target GATT server.
- `handle`: : GATT characteristic handle.

esp_err_t **esp_ble_gattc_unregister_for_notify**(*esp_gatt_if_t gattc_if*, *esp_bd_addr_t server_bda*, uint16_t *handle*)

This function is called to de-register for notification of a service.

Return

- ESP_OK: unregister succeeds
- other: failed

Parameters

- `gattc_if`: Gatt client access interface.
- `server_bda`: : target GATT server.
- `handle`: : GATT characteristic handle.

esp_err_t **esp_ble_gattc_cache_refresh**(*esp_bd_addr_t remote_bda*)

Refresh the server cache store in the gattc stack of the remote device. If the device is connected, this API will restart the discovery of service information of the remote device.

Return

- ESP_OK: success
- other: failed

Parameters

- `remote_bda`: remote device BD address.

esp_err_t **esp_ble_gattc_cache_assoc**(*esp_gatt_if_t gattc_if*, *esp_bd_addr_t src_addr*, *esp_bd_addr_t assoc_addr*, bool *is_assoc*)

Add or delete the associated address with the source address. Note: The role of this API is mainly when the client side has stored a server-side database, when it needs to connect another device, but the device' s attribute database is the same as the server database stored on the client-side, calling this API can use the database that the device has stored used as the peer server database to reduce the attribute database search and discovery process and speed up the connection time. The associated address mains that device want to used the database has stored in the local cache. The source address mains that device want to share the database to the associated address device.

Return

- ESP_OK: success
- other: failed

Parameters

- `gattc_if`: Gatt client access interface.
- `src_addr`: the source address which provide the attribute table.
- `assoc_addr`: the associated device address which went to share the attribute table with the source address.
- `is_assoc`: true add the associated device address, false remove the associated device address.

esp_err_t **esp_ble_gattc_cache_get_addr_list**(*esp_gatt_if_t gattc_if*)

Get the address list which has store the attribute table in the gattc cache. There will callback ESP_GATTC_GET_ADDR_LIST_EVT event when get address list complete.

Return

- ESP_OK: success
- other: failed

Parameters

- `gattc_if`: Gatt client access interface.

esp_err_t **esp_ble_gattc_cache_clean**(*esp_bd_addr_t remote_bda*)

Clean the service cache of this device in the gattc stack,.

Return

- ESP_OK: success
- other: failed

Parameters

- `remote_bda`: remote device BD address.

Unions

union esp_ble_gattc_cb_param_t

#include <esp_gattc_api.h> Gatt client callback parameters union.

Public Members

struct *esp_ble_gattc_cb_param_t*::*gattc_reg_evt_param* **reg**

Gatt client callback param of ESP_GATTC_REG_EVT

struct *esp_ble_gattc_cb_param_t*::*gattc_open_evt_param* **open**

Gatt client callback param of ESP_GATTC_OPEN_EVT

struct *esp_ble_gattc_cb_param_t*::*gattc_close_evt_param* **close**

Gatt client callback param of ESP_GATTC_CLOSE_EVT

struct *esp_ble_gattc_cb_param_t*::*gattc_cfg_mtu_evt_param* **cfg_mtu**

Gatt client callback param of ESP_GATTC_CFG_MTU_EVT

struct *esp_ble_gattc_cb_param_t*::*gattc_search_cmpl_evt_param* **search_cmpl**

Gatt client callback param of ESP_GATTC_SEARCH_CMPL_EVT

struct *esp_ble_gattc_cb_param_t*::*gattc_search_res_evt_param* **search_res**

Gatt client callback param of ESP_GATTC_SEARCH_RES_EVT

struct *esp_ble_gattc_cb_param_t*::*gattc_read_char_evt_param* **read**

Gatt client callback param of ESP_GATTC_READ_CHAR_EVT

struct *esp_ble_gattc_cb_param_t*::*gattc_write_evt_param* **write**

Gatt client callback param of ESP_GATTC_WRITE_DESCR_EVT

struct *esp_ble_gattc_cb_param_t*::*gattc_exec_cmpl_evt_param* **exec_cmpl**

Gatt client callback param of ESP_GATTC_EXEC_EVT

struct *esp_ble_gattc_cb_param_t*::*gattc_notify_evt_param* **notify**
Gatt client callback param of ESP_GATTC_NOTIFY_EVT

struct *esp_ble_gattc_cb_param_t*::*gattc_srvc_chg_evt_param* **srvc_chg**
Gatt client callback param of ESP_GATTC_SRVC_CHG_EVT

struct *esp_ble_gattc_cb_param_t*::*gattc_congest_evt_param* **congest**
Gatt client callback param of ESP_GATTC_CONGEST_EVT

struct *esp_ble_gattc_cb_param_t*::*gattc_reg_for_notify_evt_param* **reg_for_notify**
Gatt client callback param of ESP_GATTC_REG_FOR_NOTIFY_EVT

struct *esp_ble_gattc_cb_param_t*::*gattc_unreg_for_notify_evt_param* **unreg_for_notify**
Gatt client callback param of ESP_GATTC_UNREG_FOR_NOTIFY_EVT

struct *esp_ble_gattc_cb_param_t*::*gattc_connect_evt_param* **connect**
Gatt client callback param of ESP_GATTC_CONNECT_EVT

struct *esp_ble_gattc_cb_param_t*::*gattc_disconnect_evt_param* **disconnect**
Gatt client callback param of ESP_GATTC_DISCONNECT_EVT

struct *esp_ble_gattc_cb_param_t*::*gattc_set_assoc_addr_cmp_evt_param* **set_assoc_cmp**
Gatt client callback param of ESP_GATTC_SET_ASSOC_EVT

struct *esp_ble_gattc_cb_param_t*::*gattc_get_addr_list_evt_param* **get_addr_list**
Gatt client callback param of ESP_GATTC_GET_ADDR_LIST_EVT

struct *esp_ble_gattc_cb_param_t*::*gattc_queue_full_evt_param* **queue_full**
Gatt client callback param of ESP_GATTC_QUEUE_FULL_EVT

struct *esp_ble_gattc_cb_param_t*::*gattc_dis_srvc_cmpl_evt_param* **dis_srvc_cmpl**
Gatt client callback param of ESP_GATTC_DIS_SRVC_CMPL_EVT

struct gattc_cfg_mtu_evt_param
#include <esp_gattc_api.h> ESP_GATTC_CFG_MTU_EVT.

Public Members

esp_gatt_status_t **status**
Operation status

uint16_t **conn_id**
Connection id

uint16_t **mtu**
MTU size

struct gattc_close_evt_param
#include <esp_gattc_api.h> ESP_GATTC_CLOSE_EVT.

Public Members

esp_gatt_status_t **status**
Operation status

uint16_t **conn_id**
Connection id

esp_bd_addr_t **remote_bda**
Remote bluetooth device address

esp_gatt_conn_reason_t **reason**
The reason of gatt connection close

struct gattc_congest_evt_param
#include <esp_gattc_api.h> ESP_GATTC_CONGEST_EVT.

Public Members

uint16_t **conn_id**
Connection id

bool **congested**
Congested or not

struct gattc_connect_evt_param
#include <esp_gattc_api.h> ESP_GATTC_CONNECT_EVT.

Public Members

uint16_t **conn_id**
Connection id

esp_bd_addr_t **remote_bda**
Remote bluetooth device address

esp_gatt_conn_params_t **conn_params**
current connection parameters

struct gattc_dis_srvc_cmpl_evt_param
#include <esp_gattc_api.h> ESP_GATTC_DIS_SRVC_CMPL_EVT.

Public Members

esp_gatt_status_t **status**
Operation status

uint16_t **conn_id**
Connection id

struct gattc_disconnect_evt_param
#include <esp_gattc_api.h> ESP_GATTC_DISCONNECT_EVT.

Public Members

esp_gatt_conn_reason_t **reason**
disconnection reason

uint16_t **conn_id**
Connection id

esp_bd_addr_t **remote_bda**
Remote bluetooth device address

struct gattc_exec_cmpl_evt_param
#include <esp_gattc_api.h> ESP_GATTC_EXEC_EVT.

Public Members

esp_gatt_status_t **status**
Operation status

uint16_t **conn_id**
Connection id

struct gattc_get_addr_list_evt_param
#include <esp_gattc_api.h> ESP_GATTC_GET_ADDR_LIST_EVT.

Public Members

esp_gatt_status_t **status**
Operation status

uint8_t **num_addr**
The number of address in the gattc cache address list

esp_bd_addr_t ***addr_list**
The pointer to the address list which has been get from the gattc cache

struct gattc_notify_evt_param
#include <esp_gattc_api.h> ESP_GATTC_NOTIFY_EVT.

Public Members

uint16_t **conn_id**
Connection id

esp_bd_addr_t **remote_bda**
Remote bluetooth device address

uint16_t **handle**
The Characteristic or descriptor handle

uint16_t **value_len**
Notify attribute value

uint8_t ***value**
Notify attribute value

bool **is_notify**
True means notify, false means indicate

struct gattc_open_evt_param
#include <esp_gattc_api.h> ESP_GATTC_OPEN_EVT.

Public Members

esp_gatt_status_t **status**
Operation status

uint16_t **conn_id**
Connection id

esp_bd_addr_t **remote_bda**
Remote bluetooth device address

uint16_t **mtu**
MTU size

struct gattc_queue_full_evt_param
#include <esp_gattc_api.h> ESP_GATTC_QUEUE_FULL_EVT.

Public Members

esp_gatt_status_t **status**
Operation status

uint16_t **conn_id**
Connection id

bool **is_full**

The gattc command queue is full or not

struct gattc_read_char_evt_param

#include <esp_gattc_api.h> ESP_GATTC_READ_CHAR_EVT, ESP_GATTC_READ_DESCR_EVT.

Public Members

esp_gatt_status_t **status**

Operation status

uint16_t **conn_id**

Connection id

uint16_t **handle**

Characteristic handle

uint8_t ***value**

Characteristic value

uint16_t **value_len**

Characteristic value length

struct gattc_reg_evt_param

#include <esp_gattc_api.h> ESP_GATTC_REG_EVT.

Public Members

esp_gatt_status_t **status**

Operation status

uint16_t **app_id**

Application id which input in register API

struct gattc_reg_for_notify_evt_param

#include <esp_gattc_api.h> ESP_GATTC_REG_FOR_NOTIFY_EVT.

Public Members

esp_gatt_status_t **status**

Operation status

uint16_t **handle**

The characteristic or descriptor handle

struct gattc_search_cmpl_evt_param

#include <esp_gattc_api.h> ESP_GATTC_SEARCH_CMPL_EVT.

Public Members

esp_gatt_status_t **status**

Operation status

uint16_t **conn_id**

Connection id

esp_service_source_t **searched_service_source**

The source of the service information

struct gattc_search_res_evt_param

#include <esp_gattc_api.h> ESP_GATTC_SEARCH_RES_EVT.

Public Members

uint16_t **conn_id**

Connection id

uint16_t **start_handle**

Service start handle

uint16_t **end_handle**

Service end handle

esp_gatt_id_t **srvc_id**

Service id, include service uuid and other information

bool **is_primary**

True if this is the primary service

struct gattc_set_assoc_addr_cmp_evt_param

#include <esp_gattc_api.h> ESP_GATTC_SET_ASSOC_EVT.

Public Members

esp_gatt_status_t **status**

Operation status

struct gattc_srvc_chg_evt_param

#include <esp_gattc_api.h> ESP_GATTC_SRVC_CHG_EVT.

Public Members

esp_bd_addr_t **remote_bda**
Remote bluetooth device address

struct gattc_unreg_for_notify_evt_param
#include <esp_gattc_api.h> ESP_GATTC_UNREG_FOR_NOTIFY_EVT.

Public Members

esp_gatt_status_t **status**
Operation status

uint16_t **handle**
The characteristic or descriptor handle

struct gattc_write_evt_param
#include <esp_gattc_api.h> ESP_GATTC_WRITE_CHAR_EVT, ESP_GATTC_PREP_WRITE_EVT, ESP_GATTC_WRITE_DESCR_EVT.

Public Members

esp_gatt_status_t **status**
Operation status

uint16_t **conn_id**
Connection id

uint16_t **handle**
The Characteristic or descriptor handle

uint16_t **offset**
The prepare write offset, this value is valid only when prepare write

Type Definitions

typedef void (***esp_gattc_cb_t**)(*esp_gattc_cb_event_t* event, *esp_gatt_if_t* gattc_if, *esp_ble_gattc_cb_param_t* *param)
GATT Client callback function type.

Parameters

- `event`: : Event type
- `gatts_if`: : GATT client access interface, normally different gattc_if correspond to different profile

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

Enumerations

enum esp_gattc_cb_event_t

GATT Client callback function events.

Values:

ESP_GATTC_REG_EVT = 0

When GATT client is registered, the event comes

ESP_GATTC_UNREG_EVT = 1

When GATT client is unregistered, the event comes

ESP_GATTC_OPEN_EVT = 2

When GATT virtual connection is set up, the event comes

ESP_GATTC_READ_CHAR_EVT = 3

When GATT characteristic is read, the event comes

ESP_GATTC_WRITE_CHAR_EVT = 4

When GATT characteristic write operation completes, the event comes

ESP_GATTC_CLOSE_EVT = 5

When GATT virtual connection is closed, the event comes

ESP_GATTC_SEARCH_CMPL_EVT = 6

When GATT service discovery is completed, the event comes

ESP_GATTC_SEARCH_RES_EVT = 7

When GATT service discovery result is got, the event comes

ESP_GATTC_READ_DESCR_EVT = 8

When GATT characteristic descriptor read completes, the event comes

ESP_GATTC_WRITE_DESCR_EVT = 9

When GATT characteristic descriptor write completes, the event comes

ESP_GATTC_NOTIFY_EVT = 10

When GATT notification or indication arrives, the event comes

ESP_GATTC_PREP_WRITE_EVT = 11

When GATT prepare-write operation completes, the event comes

ESP_GATTC_EXEC_EVT = 12

When write execution completes, the event comes

ESP_GATTC_ACL_EVT = 13

When ACL connection is up, the event comes

ESP_GATTC_CANCEL_OPEN_EVT = 14

When GATT client ongoing connection is cancelled, the event comes

ESP_GATTC_SRVC_CHG_EVT = 15

When "service changed" occurs, the event comes

ESP_GATTC_ENC_CMPL_CB_EVT = 17

When encryption procedure completes, the event comes

ESP_GATTC_CFG_MTU_EVT = 18

When configuration of MTU completes, the event comes

ESP_GATTC_ADV_DATA_EVT = 19

When advertising of data, the event comes

ESP_GATTC_MULT_ADV_ENB_EVT = 20

When multi-advertising is enabled, the event comes

ESP_GATTC_MULT_ADV_UPD_EVT = 21

When multi-advertising parameters are updated, the event comes

ESP_GATTC_MULT_ADV_DATA_EVT = 22

When multi-advertising data arrives, the event comes

ESP_GATTC_MULT_ADV_DIS_EVT = 23

When multi-advertising is disabled, the event comes

ESP_GATTC_CONGEST_EVT = 24

When GATT connection congestion comes, the event comes

ESP_GATTC_BTH_SCAN_ENB_EVT = 25

When batch scan is enabled, the event comes

ESP_GATTC_BTH_SCAN_CFG_EVT = 26

When batch scan storage is configured, the event comes

ESP_GATTC_BTH_SCAN_RD_EVT = 27

When Batch scan read event is reported, the event comes

ESP_GATTC_BTH_SCAN_THR_EVT = 28

When Batch scan threshold is set, the event comes

ESP_GATTC_BTH_SCAN_PARAM_EVT = 29

When Batch scan parameters are set, the event comes

ESP_GATTC_BTH_SCAN_DIS_EVT = 30

When Batch scan is disabled, the event comes

ESP_GATTC_SCAN_FLT_CFG_EVT = 31

When Scan filter configuration completes, the event comes

`ESP_GATTC_SCAN_FLT_PARAM_EVT` = 32
When Scan filter parameters are set, the event comes

`ESP_GATTC_SCAN_FLT_STATUS_EVT` = 33
When Scan filter status is reported, the event comes

`ESP_GATTC_ADV_VSC_EVT` = 34
When advertising vendor spec content event is reported, the event comes

`ESP_GATTC_REG_FOR_NOTIFY_EVT` = 38
When register for notification of a service completes, the event comes

`ESP_GATTC_UNREG_FOR_NOTIFY_EVT` = 39
When unregister for notification of a service completes, the event comes

`ESP_GATTC_CONNECT_EVT` = 40
When the ble physical connection is set up, the event comes

`ESP_GATTC_DISCONNECT_EVT` = 41
When the ble physical connection disconnected, the event comes

`ESP_GATTC_READ_MULTIPLE_EVT` = 42
When the ble characteristic or descriptor multiple complete, the event comes

`ESP_GATTC_QUEUE_FULL_EVT` = 43
When the gattc command queue full, the event comes

`ESP_GATTC_SET_ASSOC_EVT` = 44
When the ble gattc set the associated address complete, the event comes

`ESP_GATTC_GET_ADDR_LIST_EVT` = 45
When the ble get gattc address list in cache finish, the event comes

`ESP_GATTC_DIS_SRVC_CMPL_EVT` = 46
When the ble discover service complete, the event comes

BLUFI API

Overview

BLUFI is a profile based GATT to config ESP32 WIFI to connect/disconnect AP or setup a softap and etc. Use should concern these things:

1. The event sent from profile. Then you need to do something as the event indicate.
2. Security reference. You can write your own Security functions such as symmetrical encryption/decryption and checksum functions. Even you can define the "Key Exchange/Negotiation" procedure.

Application Example

Check bluetooth/bluedroid/ble folder in ESP-IDF examples, which contains the following application:

- This is the BLUFI demo. This demo can set ESP32' s wifi to softap/station/softap&station mode and config wifi connections - bluetooth/bluedroid/ble/blufi

API Reference

Header File

- bt/host/bluedroid/api/include/api/esp_blufi_api.h

Functions

esp_err_t **esp_blufi_register_callbacks**(*esp_blufi_callbacks_t* **callbacks*)

This function is called to receive blufi callback event.

Return ESP_OK - success, other - failed

Parameters

- `callbacks`: callback functions

esp_err_t **esp_blufi_profile_init**(void)

This function is called to initialize blufi_profile.

Return ESP_OK - success, other - failed

esp_err_t **esp_blufi_profile_deinit**(void)

This function is called to de-initialize blufi_profile.

Return ESP_OK - success, other - failed

esp_err_t **esp_blufi_send_wifi_conn_report**(*wifi_mode_t opmode*, *esp_blufi_sta_conn_state_t sta_conn_state*, uint8_t *softap_conn_num*, *esp_blufi_extra_info_t* **extra_info*)

This function is called to send wifi connection report.

Return ESP_OK - success, other - failed

Parameters

- `opmode`: : wifi opmode
- `sta_conn_state`: : station is already in connection or not

- `softap_conn_num`: : softap connection number
- `extra_info`: : extra information, such as sta_ssid, softap_ssid and etc.

esp_err_t **esp_blufi_send_wifi_list**(uint16_t *apCount*, *esp_blufi_ap_record_t* **list*)

This function is called to send wifi list.

Return ESP_OK - success, other - failed

Parameters

- `apCount`: : wifi list count
- `list`: : wifi list

uint16_t **esp_blufi_get_version**(void)

Get BLUFI profile version.

Return Most 8bit significant is Great version, Least 8bit is Sub version

esp_err_t **esp_blufi_close**(*esp_gatt_if_t* *gatts_if*, uint16_t *conn_id*)

Close a connection a remote device.

Return

- ESP_OK : success
- other : failed

Parameters

- `gatts_if`: GATT server access interface
- `conn_id`: connection ID to be closed.

esp_err_t **esp_blufi_send_error_info**(*esp_blufi_error_state_t* *state*)

This function is called to send blufi error information.

Return ESP_OK - success, other - failed

Parameters

- `state`: : error state

esp_err_t **esp_blufi_send_custom_data**(uint8_t **data*, uint32_t *data_len*)

This function is called to custom data.

Return ESP_OK - success, other - failed

Parameters

- `data`: : custom data value

- `data_len`: : the length of custom data

Unions

`union esp_blufi_cb_param_t`

#include <esp_blufi_api.h> BLUFI callback parameters union.

Public Members

`struct` *esp_blufi_cb_param_t*::*blufi_init_finish_evt_param* `init_finish`

Blufi callback param of ESP_BLUFI_EVENT_INIT_FINISH

`struct` *esp_blufi_cb_param_t*::*blufi_deinit_finish_evt_param* `deinit_finish`

Blufi callback param of ESP_BLUFI_EVENT_DEINIT_FINISH

`struct` *esp_blufi_cb_param_t*::*blufi_set_wifi_mode_evt_param* `wifi_mode`

Blufi callback param of ESP_BLUFI_EVENT_INIT_FINISH

`struct` *esp_blufi_cb_param_t*::*blufi_connect_evt_param* `connect`

Blufi callback param of ESP_BLUFI_EVENT_CONNECT

`struct` *esp_blufi_cb_param_t*::*blufi_disconnect_evt_param* `disconnect`

Blufi callback param of ESP_BLUFI_EVENT_DISCONNECT

`struct` *esp_blufi_cb_param_t*::*blufi_recv_sta_bssid_evt_param* `sta_bssid`

Blufi callback param of ESP_BLUFI_EVENT_RECV_STA_BSSID

`struct` *esp_blufi_cb_param_t*::*blufi_recv_sta_ssid_evt_param* `sta_ssid`

Blufi callback param of ESP_BLUFI_EVENT_RECV_STA_SSID

`struct` *esp_blufi_cb_param_t*::*blufi_recv_sta_passwd_evt_param* `sta_passwd`

Blufi callback param of ESP_BLUFI_EVENT_RECV_STA_PASSWD

`struct` *esp_blufi_cb_param_t*::*blufi_recv_softap_ssid_evt_param* `softap_ssid`

Blufi callback param of ESP_BLUFI_EVENT_RECV_SOFTAP_SSID

`struct` *esp_blufi_cb_param_t*::*blufi_recv_softap_passwd_evt_param* `softap_passwd`

Blufi callback param of ESP_BLUFI_EVENT_RECV_SOFTAP_PASSWD

`struct` *esp_blufi_cb_param_t*::*blufi_recv_softap_max_conn_num_evt_param* `softap_max_conn_num`

Blufi callback param of ESP_BLUFI_EVENT_RECV_SOFTAP_MAX_CONN_NUM

`struct` *esp_blufi_cb_param_t*::*blufi_recv_softap_auth_mode_evt_param* `softap_auth_mode`

Blufi callback param of ESP_BLUFI_EVENT_RECV_SOFTAP_AUTH_MODE

`struct` *esp_blufi_cb_param_t*::*blufi_recv_softap_channel_evt_param* `softap_channel`

Blufi callback param of ESP_BLUFI_EVENT_RECV_SOFTAP_CHANNEL

struct *esp_blufi_cb_param_t*::*blufi_recv_username_evt_param* **username**
Blufi callback param of ESP_BLUFI_EVENT_RECV_USERNAME

struct *esp_blufi_cb_param_t*::*blufi_recv_ca_evt_param* **ca**
Blufi callback param of ESP_BLUFI_EVENT_RECV_CA_CERT

struct *esp_blufi_cb_param_t*::*blufi_recv_client_cert_evt_param* **client_cert**
Blufi callback param of ESP_BLUFI_EVENT_RECV_CLIENT_CERT

struct *esp_blufi_cb_param_t*::*blufi_recv_server_cert_evt_param* **server_cert**
Blufi callback param of ESP_BLUFI_EVENT_RECV_SERVER_CERT

struct *esp_blufi_cb_param_t*::*blufi_recv_client_pkey_evt_param* **client_pkey**
Blufi callback param of ESP_BLUFI_EVENT_RECV_CLIENT_PRIV_KEY

struct *esp_blufi_cb_param_t*::*blufi_recv_server_pkey_evt_param* **server_pkey**
Blufi callback param of ESP_BLUFI_EVENT_RECV_SERVER_PRIV_KEY

struct *esp_blufi_cb_param_t*::*blufi_get_error_evt_param* **report_error**
Blufi callback param of ESP_BLUFI_EVENT_REPORT_ERROR

struct *esp_blufi_cb_param_t*::*blufi_recv_custom_data_evt_param* **custom_data**
Blufi callback param of ESP_BLUFI_EVENT_RECV_CUSTOM_DATA

struct blufi_connect_evt_param
#include <esp_blufi_api.h> ESP_BLUFI_EVENT_CONNECT.

Public Members

esp_bd_addr_t **remote_bda**
Blufi Remote bluetooth device address

uint8_t **server_if**
server interface

uint16_t **conn_id**
Connection id

struct blufi_deinit_finish_evt_param
#include <esp_blufi_api.h> ESP_BLUFI_EVENT_DEINIT_FINISH.

Public Members

esp_blufi_deinit_state_t **state**
De-initial status

struct blufi_disconnect_evt_param
#include <esp_blufi_api.h> ESP_BLUFI_EVENT_DISCONNECT.

Public Members

esp_bd_addr_t **remote_bda**
Blufi Remote bluetooth device address

struct blufi_get_error_evt_param
#include <esp_blufi_api.h> ESP_BLUFI_EVENT_REPORT_ERROR.

Public Members

esp_blufi_error_state_t **state**
Blufi error state

struct blufi_init_finish_evt_param
#include <esp_blufi_api.h> ESP_BLUFI_EVENT_INIT_FINISH.

Public Members

esp_blufi_init_state_t **state**
Initial status

struct blufi_recv_ca_evt_param
#include <esp_blufi_api.h> ESP_BLUFI_EVENT_RECV_CA_CERT.

Public Members

uint8_t ***cert**
CA certificate point

int **cert_len**
CA certificate length

struct blufi_recv_client_cert_evt_param
#include <esp_blufi_api.h> ESP_BLUFI_EVENT_RECV_CLIENT_CERT

Public Members

uint8_t ***cert**
Client certificate point

int **cert_len**
Client certificate length

struct blufi_recv_client_pkey_evt_param
#include <esp_blufi_api.h> ESP_BLUFI_EVENT_RECV_CLIENT_PRIV_KEY

Public Members

uint8_t *pkey
Client Private Key point, if Client certificate not contain Key

int pkey_len
Client Private key length

struct blufi_recv_custom_data_evt_param
#include <esp_blufi_api.h> ESP_BLUFI_EVENT_RECV_CUSTOM_DATA.

Public Members

uint8_t *data
Custom data

uint32_t data_len
Custom data Length

struct blufi_recv_server_cert_evt_param
#include <esp_blufi_api.h> ESP_BLUFI_EVENT_RECV_SERVER_CERT

Public Members

uint8_t *cert
Client certificate point

int cert_len
Client certificate length

struct blufi_recv_server_pkey_evt_param
#include <esp_blufi_api.h> ESP_BLUFI_EVENT_RECV_SERVER_PRIV_KEY

Public Members

uint8_t *pkey
Client Private Key point, if Client certificate not contain Key

int pkey_len
Client Private key length

struct blufi_recv_softap_auth_mode_evt_param
#include <esp_blufi_api.h> ESP_BLUFI_EVENT_RECV_SOFTAP_AUTH_MODE.

Public Members

wifi_auth_mode_t **auth_mode**
Authentication mode

struct blufi_recv_softap_channel_evt_param
#include <esp_blufi_api.h> ESP_BLUFI_EVENT_RECV_SOFTAP_CHANNEL.

Public Members

uint8_t **channel**
Authentication mode

struct blufi_recv_softap_max_conn_num_evt_param
#include <esp_blufi_api.h> ESP_BLUFI_EVENT_RECV_SOFTAP_MAX_CONN_NUM.

Public Members

int **max_conn_num**
SSID

struct blufi_recv_softap_passwd_evt_param
#include <esp_blufi_api.h> ESP_BLUFI_EVENT_RECV_SOFTAP_PASSWD.

Public Members

uint8_t ***passwd**
Password

int **passwd_len**
Password Length

struct blufi_recv_softap_ssid_evt_param
#include <esp_blufi_api.h> ESP_BLUFI_EVENT_RECV_SOFTAP_SSID.

Public Members

uint8_t ***ssid**
SSID

int **ssid_len**
SSID length

struct blufi_recv_sta_bssid_evt_param
#include <esp_blufi_api.h> ESP_BLUFI_EVENT_RECV_STA_BSSID.

Public Members

uint8_t **bssid**[6]
BSSID

struct blufi_recv_sta_passwd_evt_param
#include <esp_blufi_api.h> ESP_BLUFI_EVENT_RECV_STA_PASSWD.

Public Members

uint8_t ***passwd**
Password

int **passwd_len**
Password Length

struct blufi_recv_sta_ssid_evt_param
#include <esp_blufi_api.h> ESP_BLUFI_EVENT_RECV_STA_SSID.

Public Members

uint8_t ***ssid**
SSID

int **ssid_len**
SSID length

struct blufi_recv_username_evt_param
#include <esp_blufi_api.h> ESP_BLUFI_EVENT_RECV_USERNAME.

Public Members

uint8_t ***name**
Username point

int **name_len**
Username length

struct blufi_set_wifi_mode_evt_param
#include <esp_blufi_api.h> ESP_BLUFI_EVENT_SET_WIFI_MODE.

Public Members

wifi_mode_t **op_mode**
Wifi operation mode

Structures

struct esp_blufi_extra_info_t

BLUFI extra information structure.

Public Members

uint8_t **sta_bssid**[6]

BSSID of station interface

bool **sta_bssid_set**

is BSSID of station interface set

uint8_t ***sta_ssid**

SSID of station interface

int **sta_ssid_len**

length of SSID of station interface

uint8_t ***sta_passwd**

password of station interface

int **sta_passwd_len**

length of password of station interface

uint8_t ***softap_ssid**

SSID of softap interface

int **softap_ssid_len**

length of SSID of softap interface

uint8_t ***softap_passwd**

password of station interface

int **softap_passwd_len**

length of password of station interface

uint8_t **softap_authmode**

authentication mode of softap interface

bool **softap_authmode_set**

is authentication mode of softap interface set

uint8_t **softap_max_conn_num**

max connection number of softap interface

bool **softap_max_conn_num_set**

is max connection number of softap interface set

uint8_t **softap_channel**
channel of softap interface

bool **softap_channel_set**
is channel of softap interface set

struct esp_blufi_ap_record_t
Description of an WiFi AP.

Public Members

uint8_t **ssid**[33]
SSID of AP

int8_t **rssi**
signal strength of AP

struct esp_blufi_callbacks_t
BLUFI callback functions type.

Public Members

esp_blufi_event_cb_t **event_cb**
BLUFI event callback

esp_blufi_negotiate_data_handler_t **negotiate_data_handler**
BLUFI negotiate data function for negotiate share key

esp_blufi_encrypt_func_t **encrypt_func**
BLUFI encrypt data function with share key generated by negotiate_data_handler

esp_blufi_decrypt_func_t **decrypt_func**
BLUFI decrypt data function with share key generated by negotiate_data_handler

esp_blufi_checksum_func_t **checksum_func**
BLUFI check sum function (FCS)

Type Definitions

typedef void (***esp_blufi_event_cb_t**)(*esp_blufi_cb_event_t* event, *esp_blufi_cb_param_t* *param)
BLUFI event callback function type.

Parameters

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

typedef void (*esp_blufi_negotiate_data_handler_t)(uint8_t *data, int len, uint8_t **output_data, int *output_len, bool *need_free)

BLUFI negotiate data handler.

Parameters

- data: : data from phone
- len: : length of data from phone
- output_data: : data want to send to phone
- output_len: : length of data want to send to phone

typedef int (*esp_blufi_encrypt_func_t)(uint8_t iv8, uint8_t *crypt_data, int cyprt_len)

BLUFI encrypt the data after negotiate a share key.

Return Nonnegative number is encrypted length, if error, return negative number;

Parameters

- iv8: : initial vector(8bit), normally, blufi core will input packet sequence number
- crypt_data: : plain text and encrypted data, the encrypt function must support autochthonous encrypt
- crypt_len: : length of plain text

typedef int (*esp_blufi_decrypt_func_t)(uint8_t iv8, uint8_t *crypt_data, int crypt_len)

BLUFI decrypt the data after negotiate a share key.

Return Nonnegative number is decrypted length, if error, return negative number;

Parameters

- iv8: : initial vector(8bit), normally, blufi core will input packet sequence number
- crypt_data: : encrypted data and plain text, the encrypt function must support autochthonous decrypt
- crypt_len: : length of encrypted text

typedef uint16_t (*esp_blufi_checksum_func_t)(uint8_t iv8, uint8_t *data, int len)

BLUFI checksum.

Parameters

- iv8: : initial vector(8bit), normally, blufi core will input packet sequence number
- data: : data need to checksum
- len: : length of data

Enumerations

enum esp_blufi_cb_event_t

Values:

ESP_BLUFI_EVENT_INIT_FINISH = 0

ESP_BLUFI_EVENT_DEINIT_FINISH

ESP_BLUFI_EVENT_SET_WIFI_OPMODE

ESP_BLUFI_EVENT_BLE_CONNECT

ESP_BLUFI_EVENT_BLE_DISCONNECT

ESP_BLUFI_EVENT_REQ_CONNECT_TO_AP

ESP_BLUFI_EVENT_REQ_DISCONNECT_FROM_AP

ESP_BLUFI_EVENT_GET_WIFI_STATUS

ESP_BLUFI_EVENT_DEAUTHENTICATE_STA

ESP_BLUFI_EVENT_RECV_STA_BSSID

ESP_BLUFI_EVENT_RECV_STA_SSID

ESP_BLUFI_EVENT_RECV_STA_PASSWD

ESP_BLUFI_EVENT_RECV_SOFTAP_SSID

ESP_BLUFI_EVENT_RECV_SOFTAP_PASSWD

ESP_BLUFI_EVENT_RECV_SOFTAP_MAX_CONN_NUM

ESP_BLUFI_EVENT_RECV_SOFTAP_AUTH_MODE

ESP_BLUFI_EVENT_RECV_SOFTAP_CHANNEL

ESP_BLUFI_EVENT_RECV_USERNAME

ESP_BLUFI_EVENT_RECV_CA_CERT

ESP_BLUFI_EVENT_RECV_CLIENT_CERT

ESP_BLUFI_EVENT_RECV_SERVER_CERT

ESP_BLUFI_EVENT_RECV_CLIENT_PRIV_KEY

ESP_BLUFI_EVENT_RECV_SERVER_PRIV_KEY

ESP_BLUFI_EVENT_RECV_SLAVE_DISCONNECT_BLE

ESP_BLUFI_EVENT_GET_WIFI_LIST

ESP_BLUFI_EVENT_REPORT_ERROR

ESP_BLUFI_EVENT_RECV_CUSTOM_DATA

enum esp_blufi_sta_conn_state_t

BLUFI config status.

Values:

ESP_BLUFI_STA_CONN_SUCCESS = 0x00

ESP_BLUFI_STA_CONN_FAIL = 0x01

enum esp_blufi_init_state_t

BLUFI init status.

Values:

ESP_BLUFI_INIT_OK = 0

ESP_BLUFI_INIT_FAILED

enum esp_blufi_deinit_state_t

BLUFI deinit status.

Values:

ESP_BLUFI_DEINIT_OK = 0

ESP_BLUFI_DEINIT_FAILED

enum esp_blufi_error_state_t

Values:

ESP_BLUFI_SEQUENCE_ERROR = 0

ESP_BLUFI_CHECKSUM_ERROR

ESP_BLUFI_DECRYPT_ERROR

ESP_BLUFI_ENCRYPT_ERROR

ESP_BLUFI_INIT_SECURITY_ERROR

ESP_BLUFI_DH_MALLOC_ERROR

ESP_BLUFI_DH_PARAM_ERROR

ESP_BLUFI_READ_PARAM_ERROR

ESP_BLUFI_MAKE_PUBLIC_ERROR

ESP_BLUFI_DATA_FORMAT_ERROR

2.1.4 CLASSIC BT

CLASSIC BLUETOOTH GAP API

Overview

Instructions

Application Example

Instructions

API Reference

Header File

- bt/host/bluedroid/api/include/api/esp_gap_bt_api.h

Functions

`static` uint32_t **esp_bt_gap_get_cod_srvc**(uint32_t *cod*)

get major service field of COD

Return major service bits

Parameters

- `cod`: Class of Device

`static` uint32_t **esp_bt_gap_get_cod_major_dev**(uint32_t *cod*)

get major device field of COD

Return major device bits

Parameters

- `cod`: Class of Device

`static` uint32_t **esp_bt_gap_get_cod_minor_dev**(uint32_t *cod*)

get minor service field of COD

Return minor service bits

Parameters

- `cod`: Class of Device

`static` uint32_t **esp_bt_gap_get_cod_format_type**(uint32_t *cod*)

get format type of COD

Return format type

Parameters

- `cod`: Class of Device

`static` bool **esp_bt_gap_is_valid_cod**(uint32_t *cod*)

decide the integrity of COD

Return

- true if cod is valid
- false otherise

Parameters

- `cod`: Class of Device

esp_err_t **esp_bt_gap_register_callback**(*esp_bt_gap_cb_t callback*)

register callback function. This function should be called after esp_bluedroid_enable() completes successfully

Return

- ESP_OK : Succeed
- ESP_FAIL: others

esp_err_t **esp_bt_gap_set_scan_mode**(*esp_bt_connection_mode_t c_mode*, *esp_bt_discovery_mode_t d_mode*)

Set discoverability and connectability mode for legacy bluetooth. This function should be called after esp_bluedroid_enable() completes successfully.

Return

- ESP_OK : Succeed
- ESP_ERR_INVALID_ARG: if argument invalid
- ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

Parameters

- `c_mode`: : one of the enums of esp_bt_connection_mode_t
- `d_mode`: : one of the enums of esp_bt_discovery_mode_t

esp_err_t **esp_bt_gap_start_discovery**(*esp_bt_inq_mode_t mode*, uint8_t *inq_len*, uint8_t *num_rsps*)

Start device discovery. This function should be called after esp_bluedroid_enable() completes successfully. esp_bt_gap_cb_t will be called with ESP_BT_GAP_DISC_STATE_CHANGED_EVT if discovery is started or halted. esp_bt_gap_cb_t will be called with ESP_BT_GAP_DISC_RES_EVT if discovery result is got.

Return

- ESP_OK : Succeed
- ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_ERR_INVALID_ARG: if invalid parameters are provided
- ESP_FAIL: others

Parameters

- `mode`: - inquiry mode
- `inq_len`: - inquiry duration in 1.28 sec units, ranging from 0x01 to 0x30
- `num_rsps`: - number of inquiry responses that can be received, value 0 indicates an unlimited number of responses

esp_err_t **esp_bt_gap_cancel_discovery**(void)

Cancel device discovery. This function should be called after esp_bluedroid_enable() completes successfully esp_bt_gap_cb_t will be called with ESP_BT_GAP_DISC_STATE_CHANGED_EVT if discovery is stopped.

Return

- ESP_OK : Succeed
- ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

esp_err_t **esp_bt_gap_get_remote_services**(*esp_bd_addr_t remote_bda*)

Start SDP to get remote services. This function should be called after esp_bluedroid_enable() completes successfully. esp_bt_gap_cb_t will be called with ESP_BT_GAP_RMT_SRVCS_EVT after service discovery ends.

Return

- ESP_OK : Succeed
- ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

esp_err_t **esp_bt_gap_get_remote_service_record**(*esp_bd_addr_t remote_bda*, *esp_bt_uuid_t* **uuid*)

Start SDP to look up the service matching uuid on the remote device. This function should be called after esp_bluedroid_enable() completes successfully.

esp_bt_gap_cb_t will be called with ESP_BT_GAP_RMT_SRVC_REC_EVT after service discovery ends

Return

- ESP_OK : Succeed
- ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

uint8_t ***esp_bt_gap_resolve_eir_data**(uint8_t **eir*, *esp_bt_eir_type_t type*, uint8_t **length*)

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

Return pointer of starting position of eir data excluding eir data type, NULL if not found

Parameters

- `eir`: - pointer of raw eir data to be resolved
- `type`: - specific EIR data type
- `length`: - return the length of EIR data excluding fields of length and data type

esp_err_t **esp_bt_gap_config_eir_data**(*esp_bt_eir_data_t* **eir_data*)

This function is called to config EIR data.

esp_bt_gap_cb_t will be called with ESP_BT_GAP_CONFIG_EIR_DATA_EVT after config EIR ends.

Return

- ESP_OK : Succeed
- ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_ERR_INVALID_ARG: if param is invalid
- ESP_FAIL: others

Parameters

- `eir_data`: - pointer of EIR data content

esp_err_t **esp_bt_gap_set_cod**(*esp_bt_cod_t cod*, *esp_bt_cod_mode_t mode*)

This function is called to set class of device. esp_bt_gap_cb_t will be called with ESP_BT_GAP_SET_COD_EVT after set COD ends Some profile have special restrictions on class of device, changes may cause these profile do not work.

Return

- ESP_OK : Succeed
- ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_ERR_INVALID_ARG: if param is invalid
- ESP_FAIL: others

Parameters

- `cod`: - class of device
- `mode`: - setting mode

esp_err_t **esp_bt_gap_get_cod**(*esp_bt_cod_t* **cod*)

This function is called to get class of device.

Return

- ESP_OK : Succeed
- ESP_FAIL: others

Parameters

- `cod`: - class of device

esp_err_t **esp_bt_gap_read_rssi_delta**(*esp_bd_addr_t remote_addr*)

This function is called to read RSSI delta by address after connected. The RSSI value returned by ESP_BT_GAP_READ_RSSI_DELTA_EVT.

Return

- ESP_OK : Succeed
- ESP_FAIL: others

Parameters

- `remote_addr`: - remote device address, corresponding to a certain connection handle.

esp_err_t **esp_bt_gap_remove_bond_device**(*esp_bd_addr_t bd_addr*)

Removes a device from the security database list of peer device.

Return - ESP_OK : success

- ESP_FAIL : failed

Parameters

- `bd_addr`: : BD address of the peer device

int **esp_bt_gap_get_bond_device_num**(void)

Get the device number from the security database list of peer device. It will return the device bonded number immediately.

Return - >= 0 : bonded devices number.

- ESP_FAIL : failed

esp_err_t **esp_bt_gap_get_bond_device_list**(int **dev_num*, *esp_bd_addr_t *dev_list*)

Get the device from the security database list of peer device. It will return the device bonded information immediately.

Return

- ESP_OK : Succeed
- ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

Parameters

- **dev_num**: Indicate the dev_list array(buffer) size as input. If dev_num is large enough, it means the actual number as output. Suggest that dev_num value equal to esp_ble_get_bond_device_num().
- **dev_list**: an array(buffer) of **esp_bd_addr_t** type. Use for storing the bonded devices address. The dev_list should be allocated by who call this API.

esp_err_t **esp_bt_gap_set_pin**(*esp_bt_pin_type_t pin_type*, uint8_t *pin_code_len*, *esp_bt_pin_code_t pin_code*)

Set pin type and default pin code for legacy pairing.

Return - ESP_OK : success

- ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
- other : failed

Parameters

- **pin_type**: Use variable or fixed pin. If pin_type is ESP_BT_PIN_TYPE_VARIABLE, pin_code and pin_code_len will be ignored, and ESP_BT_GAP_PIN_REQ_EVT will come when control requests for pin code. Else, will use fixed pin code and not callback to users.
- **pin_code_len**: Length of pin_code
- **pin_code**: Pin_code

esp_err_t **esp_bt_gap_pin_reply**(*esp_bd_addr_t bd_addr*, bool *accept*, uint8_t *pin_code_len*, *esp_bt_pin_code_t pin_code*)

Reply the pin_code to the peer device for legacy pairing when ESP_BT_GAP_PIN_REQ_EVT is coming.

Return - ESP_OK : success

- ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
- other : failed

Parameters

- `bd_addr`: BD address of the peer
- `accept`: Pin_code reply successful or declined.
- `pin_code_len`: Length of pin_code
- `pin_code`: Pin_code

esp_err_t **esp_bt_gap_set_security_param**(*esp_bt_sp_param_t param_type*, void **value*, uint8_t *len*)

Set a GAP security parameter value. Overrides the default value.

Return - ESP_OK : success

- ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
- other : failed

Parameters

- `param_type`: : the type of the param which is to be set
- `value`: : the param value
- `len`: : the length of the param value

esp_err_t **esp_bt_gap_ssp_passkey_reply**(*esp_bd_addr_t bd_addr*, bool *accept*, uint32_t *passkey*)

Reply the key value to the peer device in the legacy connection stage.

Return - ESP_OK : success

- ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
- other : failed

Parameters

- `bd_addr`: : BD address of the peer
- `accept`: : passkey entry successful or declined.
- `passkey`: : passkey value, must be a 6 digit number, can be lead by 0.

esp_err_t **esp_bt_gap_ssp_confirm_reply**(*esp_bd_addr_t bd_addr*, bool *accept*)

Reply the confirm value to the peer device in the legacy connection stage.

Return - ESP_OK : success

- ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
- other : failed

Parameters

- `bd_addr`: : BD address of the peer device
- `accept`: : numbers to compare are the same or different.

esp_err_t **esp_bt_gap_set_afh_channels**(*esp_bt_gap_afh_channels channels*)

Set the AFH channels.

Return - ESP_OK : success

- ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
- other : failed

Parameters

- `channels`: : The n th such field (in the range 0 to 78) contains the value for channel n : 0 means channel n is bad. 1 means channel n is unknown. The most significant bit is reserved and shall be set to 0. At least 20 channels shall be marked as unknown.

esp_err_t **esp_bt_gap_read_remote_name**(*esp_bd_addr_t remote_bda*)

Read the remote device name.

Return - ESP_OK : success

- ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
- other : failed

Parameters

- `remote_bda`: The remote device' s address

Unions

union esp_bt_gap_cb_param_t

#include <esp_gap_bt_api.h> A2DP state callback parameters.

Public Members

struct *esp_bt_gap_cb_param_t*::*disc_res_param* **disc_res**
discovery result parameter struct

struct *esp_bt_gap_cb_param_t*::*disc_state_changed_param* **disc_st_chg**
discovery state changed parameter struct

struct *esp_bt_gap_cb_param_t*::*rmt_srvcs_param* **rmt_srvcs**
services of remote device parameter struct

struct *esp_bt_gap_cb_param_t*::*rmt_srvc_rec_param* **rmt_srvc_rec**
specific service record from remote device parameter struct

struct *esp_bt_gap_cb_param_t*::*read_rssi_delta_param* **read_rssi_delta**
read rssi parameter struct

struct *esp_bt_gap_cb_param_t*::*config_eir_data_param* **config_eir_data**
config EIR data

struct *esp_bt_gap_cb_param_t*::*auth_cmpl_param* **auth_cmpl**
authentication complete parameter struct

struct *esp_bt_gap_cb_param_t*::*pin_req_param* **pin_req**
pin request parameter struct

struct *esp_bt_gap_cb_param_t*::*cfm_req_param* **cfm_req**
confirm request parameter struct

struct *esp_bt_gap_cb_param_t*::*key_notif_param* **key_notif**
passkey notif parameter struct

struct *esp_bt_gap_cb_param_t*::*key_req_param* **key_req**
passkey request parameter struct

struct *esp_bt_gap_cb_param_t*::*set_afh_channels_param* **set_afh_channels**
set AFH channel parameter struct

struct *esp_bt_gap_cb_param_t*::*read_rmt_name_param* **read_rmt_name**
read Remote Name parameter struct

struct **auth_cmpl_param**
#include <esp_gap_bt_api.h> ESP_BT_GAP_AUTH_CMPL_EVT.

Public Members

esp_bd_addr_t **bda**
remote bluetooth device address

esp_bt_status_t **stat**

authentication complete status

uint8_t **device_name**[**ESP_BT_GAP_MAX_BDNAME_LEN** + 1]

device name

struct cfm_req_param

#include <esp_gap_bt_api.h> ESP_BT_GAP_CFM_REQ_EVT.

Public Members

esp_bd_addr_t **bda**

remote bluetooth device address

uint32_t **num_val**

the numeric value for comparison.

struct config_eir_data_param

#include <esp_gap_bt_api.h> ESP_BT_GAP_CONFIG_EIR_DATA_EVT *.

Public Members

esp_bt_status_t **stat**

config EIR status: ESP_BT_STATUS_SUCCESS: config success ESP_BT_STATUS_EIR_TOO_LARGE: the EIR data is more than 240B. The EIR may not contain the whole data. others: failed

uint8_t **eir_type_num**

the number of EIR types in EIR type

esp_bt_eir_type_t **eir_type**[**ESP_BT_EIR_TYPE_MAX_NUM**]

EIR types in EIR type

struct disc_res_param

#include <esp_gap_bt_api.h> ESP_BT_GAP_DISC_RES_EVT.

Public Members

esp_bd_addr_t **bda**

remote bluetooth device address

int **num_prop**

number of properties got

esp_bt_gap_dev_prop_t ***prop**

properties discovered from the new device

struct disc_state_changed_param

#include <esp_gap_bt_api.h> ESP_BT_GAP_DISC_STATE_CHANGED_EVT.

Public Members

esp_bt_gap_discovery_state_t **state**

discovery state

struct key_notif_param

#include <esp_gap_bt_api.h> ESP_BT_GAP_KEY_NOTIF_EVT.

Public Members

esp_bd_addr_t **bda**

remote bluetooth device address

uint32_t **passkey**

the numeric value for passkey entry.

struct key_req_param

#include <esp_gap_bt_api.h> ESP_BT_GAP_KEY_REQ_EVT.

Public Members

esp_bd_addr_t **bda**

remote bluetooth device address

struct pin_req_param

#include <esp_gap_bt_api.h> ESP_BT_GAP_PIN_REQ_EVT.

Public Members

esp_bd_addr_t **bda**

remote bluetooth device address

bool **min_16_digit**

TRUE if the pin returned must be at least 16 digits

struct read_rmt_name_param

#include <esp_gap_bt_api.h> ESP_BT_GAP_READ_REMOTE_NAME_EVT.

Public Members

esp_bt_status_t **stat**
read Remote Name status

uint8_t **rmt_name**[**ESP_BT_GAP_MAX_BDNAME_LEN** + 1]
Remote device name

struct read_rssi_delta_param
#include <esp_gap_bt_api.h> ESP_BT_GAP_READ_RSSI_DELTA_EVT *.

Public Members

esp_bd_addr_t **bda**
remote bluetooth device address

esp_bt_status_t **stat**
read rssi status

int8_t **rssi_delta**
rssi delta value range -128 ~127, The value zero indicates that the RSSI is inside the Golden Receive Power Range, the Golden Receive Power Range is from ESP_BT_GAP_RSSI_LOW_THRLD to ESP_BT_GAP_RSSI_HIGH_THRLD

struct rmt_srvc_rec_param
#include <esp_gap_bt_api.h> ESP_BT_GAP_RMT_SRVC_REC_EVT.

Public Members

esp_bd_addr_t **bda**
remote bluetooth device address

esp_bt_status_t **stat**
service search status

struct rmt_srvcs_param
#include <esp_gap_bt_api.h> ESP_BT_GAP_RMT_SRVCS_EVT.

Public Members

esp_bd_addr_t **bda**
remote bluetooth device address

esp_bt_status_t **stat**
service search status

int **num_uuids**
number of UUID in uuid_list

esp_bt_uuid_t ***uuid_list**
list of service UUIDs of remote device

struct set_afh_channels_param
#include <esp_gap_bt_api.h> ESP_BT_GAP_SET_AFH_CHANNELS_EVT.

Public Members

esp_bt_status_t **stat**
set AFH channel status

Structures

struct esp_bt_cod_t
Class of device.

Public Members

uint32_t **reserved_2**
undefined

uint32_t **minor**
minor class

uint32_t **major**
major class

uint32_t **service**
service class

uint32_t **reserved_8**
undefined

struct esp_bt_gap_dev_prop_t
Bluetooth Device Property Descriptor.

Public Members

esp_bt_gap_dev_prop_type_t **type**
device property type

int **len**
device property value length

void *val
device property value

struct esp_bt_eir_data_t
EIR data content, according to "Supplement to the Bluetooth Core Specification".

Public Members

bool fec_required
FEC is required or not, true by default

bool include_txpower
EIR data include TX power, false by default

bool include_uuid
EIR data include UUID, false by default

uint8_t flag
EIR flags, see ESP_BT_EIR_FLAG for details, EIR will not include flag if it is 0, 0 by default

uint16_t manufacturer_len
Manufacturer data length, 0 by default

uint8_t *p_manufacturer_data
Manufacturer data point

uint16_t url_len
URL length, 0 by default

uint8_t *p_url
URL point

Macros

ESP_BT_GAP_RSSI_HIGH_THRLD
RSSI threshold.

High RSSI threshold

ESP_BT_GAP_RSSI_LOW_THRLD
Low RSSI threshold

ESP_BT_GAP_AFH_CHANNELS_LEN

ESP_BT_GAP_MAX_BDNAME_LEN
Maximum bytes of Bluetooth device name.

ESP_BT_GAP_EIR_DATA_LEN
Maximum size of EIR Significant part.

ESP_BT_EIR_TYPE_FLAGS
: Extended Inquiry Response data type.

  Flag with information such as BR/EDR and LE support

ESP_BT_EIR_TYPE_INCMPL_16BITS_UUID
: Incomplete list of 16-bit service UUIDs

ESP_BT_EIR_TYPE_CMPL_16BITS_UUID
: Complete list of 16-bit service UUIDs

ESP_BT_EIR_TYPE_INCMPL_32BITS_UUID
: Incomplete list of 32-bit service UUIDs

ESP_BT_EIR_TYPE_CMPL_32BITS_UUID
: Complete list of 32-bit service UUIDs

ESP_BT_EIR_TYPE_INCMPL_128BITS_UUID
: Incomplete list of 128-bit service UUIDs

ESP_BT_EIR_TYPE_CMPL_128BITS_UUID
: Complete list of 128-bit service UUIDs

ESP_BT_EIR_TYPE_SHORT_LOCAL_NAME
: Shortened Local Name

ESP_BT_EIR_TYPE_CMPL_LOCAL_NAME
: Complete Local Name

ESP_BT_EIR_TYPE_TX_POWER_LEVEL
: Tx power level, value is 1 octet ranging from -127 to 127, unit is dBm

ESP_BT_EIR_TYPE_URL
: Uniform resource identifier

ESP_BT_EIR_TYPE_MANU_SPECIFIC
: Manufacturer specific data

ESP_BT_EIR_TYPE_MAX_NUM
: MAX number of EIR type

ESP_BT_EIR_FLAG_LIMIT_DISC

ESP_BT_EIR_FLAG_GEN_DISC

ESP_BT_EIR_FLAG_BREDR_NOT_SPT

ESP_BT_EIR_FLAG_DMT_CONTROLLER_SPT

ESP_BT_EIR_FLAG_DMT_HOST_SPT

ESP_BT_EIR_MAX_LEN

ESP_BT_PIN_CODE_LEN

Max pin code length

ESP_BT_IO_CAP_OUT

ESP_BT_IO_CAP_IO

ESP_BT_IO_CAP_IN

ESP_BT_IO_CAP_NONE

ESP_BT_COD_SRVC_BIT_MASK

Bits of major service class field.

Major service bit mask

ESP_BT_COD_SRVC_BIT_OFFSET

Major service bit offset

ESP_BT_COD_MAJOR_DEV_BIT_MASK

Bits of major device class field.

Major device bit mask

ESP_BT_COD_MAJOR_DEV_BIT_OFFSET

Major device bit offset

ESP_BT_COD_MINOR_DEV_BIT_MASK

Bits of minor device class field.

Minor device bit mask

ESP_BT_COD_MINOR_DEV_BIT_OFFSET

Minor device bit offset

ESP_BT_COD_FORMAT_TYPE_BIT_MASK

Bits of format type.

Format type bit mask

ESP_BT_COD_FORMAT_TYPE_BIT_OFFSET

Format type bit offset

ESP_BT_COD_FORMAT_TYPE_1

Class of device format type 1.

ESP_BT_GAP_MIN_INQ_LEN

Minimum and Maximum inquiry length Minimum inquiry duration, unit is 1.28s

ESP_BT_GAP_MAX_INQ_LEN

Maximum inquiry duration, unit is 1.28s

Type Definitions

typedef uint8_t **esp_bt_gap_afh_channels**[ESP_BT_GAP_AFH_CHANNELS_LEN]

typedef uint8_t **esp_bt_eir_type_t**

typedef uint8_t **esp_bt_pin_code_t**[ESP_BT_PIN_CODE_LEN]

Pin Code (upto 128 bits) MSB is 0

typedef uint8_t **esp_bt_io_cap_t**

combination of the io capability

typedef void (***esp_bt_gap_cb_t**)(*esp_bt_gap_cb_event_t* event, *esp_bt_gap_cb_param_t* *param)

bluetooth GAP callback function type

Parameters

- `event`: : Event type
- `param`: : Pointer to callback parameter

Enumerations

enum esp_bt_cod_mode_t

class of device settings

Values:

ESP_BT_SET_COD_MAJOR_MINOR = 0x01

overwrite major, minor class

ESP_BT_SET_COD_SERVICE_CLASS = 0x02

set the bits in the input, the current bit will remain

ESP_BT_CLR_COD_SERVICE_CLASS = 0x04

clear the bits in the input, others will remain

ESP_BT_SET_COD_ALL = 0x08

overwrite major, minor, set the bits in service class

ESP_BT_INIT_COD = 0x0a

overwrite major, minor, and service class

enum esp_bt_connection_mode_t

Discoverability and Connectability mode.

Values:

ESP_BT_NON_CONNECTABLE

Non-connectable

ESP_BT_CONNECTABLE
Connectable

enum esp_bt_discovery_mode_t
Values:

ESP_BT_NON_DISCOVERABLE
Non-discoverable

ESP_BT_LIMITED_DISCOVERABLE
Limited Discoverable

ESP_BT_GENERAL_DISCOVERABLE
General Discoverable

enum esp_bt_gap_dev_prop_type_t
Bluetooth Device Property type.

Values:

ESP_BT_GAP_DEV_PROP_BDNAME = 1
Bluetooth device name, value type is int8_t []

ESP_BT_GAP_DEV_PROP_COD
Class of Device, value type is uint32_t

ESP_BT_GAP_DEV_PROP_RSSI
Received Signal strength Indication, value type is int8_t, ranging from -128 to 127

ESP_BT_GAP_DEV_PROP_EIR
Extended Inquiry Response, value type is uint8_t []

enum esp_bt_cod_srvc_t
Major service class field of Class of Device, mutiple bits can be set.

Values:

ESP_BT_COD_SRVC_NONE = 0
None indicates an invalid value

ESP_BT_COD_SRVC_LMTD_DISCOVER = 0x1
Limited Discoverable Mode

ESP_BT_COD_SRVC_POSITIONING = 0x8
Positioning (Location identification)

ESP_BT_COD_SRVC_NETWORKING = 0x10
Networking, e.g. LAN, Ad hoc

ESP_BT_COD_SRVC_RENDERING = 0x20
Rendering, e.g. Printing, Speakers

ESP_BT_COD_SRVC_CAPTURING = 0x40
Capturing, e.g. Scanner, Microphone

ESP_BT_COD_SRVC_OBJ_TRANSFER = 0x80
Object Transfer, e.g. v-Inbox, v-Folder

ESP_BT_COD_SRVC_AUDIO = 0x100
Audio, e.g. Speaker, Microphone, Headset service

ESP_BT_COD_SRVC_TELEPHONY = 0x200
Telephony, e.g. Cordless telephony, Modem, Headset service

ESP_BT_COD_SRVC_INFORMATION = 0x400
Information, e.g., WEB-server, WAP-server

enum esp_bt_pin_type_t
Values:

ESP_BT_PIN_TYPE_VARIABLE = 0
Refer to BTM_PIN_TYPE_VARIABLE

ESP_BT_PIN_TYPE_FIXED = 1
Refer to BTM_PIN_TYPE_FIXED

enum esp_bt_sp_param_t
Values:

ESP_BT_SP_IOCAP_MODE = 0
Set IO mode

enum esp_bt_cod_major_dev_t
Major device class field of Class of Device.

Values:

ESP_BT_COD_MAJOR_DEV_MISC = 0
Miscellaneous

ESP_BT_COD_MAJOR_DEV_COMPUTER = 1
Computer

ESP_BT_COD_MAJOR_DEV_PHONE = 2
Phone(cellular, cordless, pay phone, modem

ESP_BT_COD_MAJOR_DEV_LAN_NAP = 3
LAN, Network Access Point

ESP_BT_COD_MAJOR_DEV_AV = 4
Audio/Video(headset, speaker, stereo, video display, VCR

ESP_BT_COD_MAJOR_DEV_PERIPHERAL = 5
Peripheral(mouse, joystick, keyboard)

ESP_BT_COD_MAJOR_DEV_IMAGING = 6
Imaging(printer, scanner, camera, display

ESP_BT_COD_MAJOR_DEV_WEARABLE = 7
Wearable

ESP_BT_COD_MAJOR_DEV_TOY = 8
Toy

ESP_BT_COD_MAJOR_DEV_HEALTH = 9
Health

ESP_BT_COD_MAJOR_DEV_UNCATEGORIZED = 31
Uncategorized: device not specified

enum esp_bt_gap_discovery_state_t
Bluetooth Device Discovery state

Values:

ESP_BT_GAP_DISCOVERY_STOPPED
device discovery stopped

ESP_BT_GAP_DISCOVERY_STARTED
device discovery started

enum esp_bt_gap_cb_event_t
BT GAP callback events.

Values:

ESP_BT_GAP_DISC_RES_EVT = 0
device discovery result event

ESP_BT_GAP_DISC_STATE_CHANGED_EVT
discovery state changed event

ESP_BT_GAP_RMT_SRVCS_EVT
get remote services event

ESP_BT_GAP_RMT_SRVC_REC_EVT
get remote service record event

ESP_BT_GAP_AUTH_CMPL_EVT
AUTH complete event

ESP_BT_GAP_PIN_REQ_EVT
Legacy Pairing Pin code request

ESP_BT_GAP_CFM_REQ_EVT
Simple Pairing User Confirmation request.

ESP_BT_GAP_KEY_NOTIF_EVT
: Simple Pairing Passkey Notification

ESP_BT_GAP_KEY_REQ_EVT
: Simple Pairing Passkey request

ESP_BT_GAP_READ_RSSI_DELTA_EVT
: read rssi event

ESP_BT_GAP_CONFIG_EIR_DATA_EVT
: config EIR data event

ESP_BT_GAP_SET_AFH_CHANNELS_EVT
: set AFH channels event

ESP_BT_GAP_READ_REMOTE_NAME_EVT
: read Remote Name event

ESP_BT_GAP_EVT_MAX

enum esp_bt_inq_mode_t
: Inquiry Mode

Values:

ESP_BT_INQ_MODE_GENERAL_INQUIRY
: General inquiry mode

ESP_BT_INQ_MODE_LIMITED_INQUIRY
: Limited inquiry mode

Bluetooth A2DP API

Overview

Instructions

Application Example

Check bluetooth/bluedroid/classic_bt folder in ESP-IDF examples, which contains the following application:

- This is a A2DP sink client demo. This demo can be discovered and connected by A2DP source device and receive the audio stream from remote device - bluetooth/bluedroid/classic_bt/a2dp_sink

API Reference

Header File

- bt/host/bluedroid/api/include/api/esp_a2dp_api.h

Functions

esp_err_t **esp_a2d_register_callback**(*esp_a2d_cb_t callback*)

Register application callback function to A2DP module. This function should be called only after esp_bluedroid_enable() completes successfully, used by both A2DP source and sink.

Return

- ESP_OK: success
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: if callback is a NULL function pointer

Parameters

- `callback`: A2DP event callback function

esp_err_t **esp_a2d_sink_register_data_callback**(*esp_a2d_sink_data_cb_t callback*)

Register A2DP sink data output function; For now the output is PCM data stream decoded from SBC format. This function should be called only after esp_bluedroid_enable() completes successfully, used only by A2DP sink. The callback is invoked in the context of A2DP sink task whose stack size is configurable through menuconfig.

Return

- ESP_OK: success
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: if callback is a NULL function pointer

Parameters

- `callback`: A2DP sink data callback function

esp_err_t **esp_a2d_sink_init**(void)

Initialize the bluetooth A2DP sink module. This function should be called after esp_bluedroid_enable() completes successfully.

Return

- ESP_OK: if the initialization request is sent successfully
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled

- ESP_FAIL: others

esp_err_t **esp_a2d_sink_deinit**(void)

De-initialize for A2DP sink module. This function should be called only after esp_bluedroid_enable() completes successfully.

Return

- ESP_OK: success
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

esp_err_t **esp_a2d_sink_connect**(*esp_bd_addr_t remote_bda*)

Connect to remote bluetooth A2DP source device, must after esp_a2d_sink_init()

Return

- ESP_OK: connect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

Parameters

- `remote_bda`: remote bluetooth device address

esp_err_t **esp_a2d_sink_disconnect**(*esp_bd_addr_t remote_bda*)

Disconnect from the remote A2DP source device.

Return

- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

Parameters

- `remote_bda`: remote bluetooth device address

esp_err_t **esp_a2d_media_ctrl**(*esp_a2d_media_ctrl_t ctrl*)

media control commands; this API can be used for both A2DP sink and source

Return

- ESP_OK: control command is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled

- ESP_FAIL: others

Parameters

- `ctrl`: control commands for A2DP data channel

esp_err_t **esp_a2d_source_init**(void)

Initialize the bluetooth A2DP source module. This function should be called after esp_bluedroid_enable() completes successfully.

Return

- ESP_OK: if the initialization request is sent successfully
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

esp_err_t **esp_a2d_source_deinit**(void)

De-initialize for A2DP source module. This function should be called only after esp_bluedroid_enable() completes successfully.

Return

- ESP_OK: success
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

esp_err_t **esp_a2d_source_register_data_callback**(*esp_a2d_source_data_cb_t callback*)

Register A2DP source data input function; For now the input is PCM data stream. This function should be called only after esp_bluedroid_enable() completes successfully. The callback is invoked in the context of A2DP source task whose stack size is configurable through menuconfig.

Return

- ESP_OK: success
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: if callback is a NULL function pointer

Parameters

- `callback`: A2DP source data callback function

esp_err_t **esp_a2d_source_connect**(*esp_bd_addr_t remote_bda*)

Connect to remote A2DP sink device, must after esp_a2d_source_init()

Return

- ESP_OK: connect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

Parameters

- `remote_bda`: remote bluetooth device address

esp_err_t **esp_a2d_source_disconnect**(*esp_bd_addr_t remote_bda*)

Disconnect from the remote A2DP sink device.

Return

- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

Parameters

- `remote_bda`: remote bluetooth device address

Unions

union esp_a2d_cb_param_t

#include <esp_a2dp_api.h> A2DP state callback parameters.

Public Members

struct *esp_a2d_cb_param_t*::*a2d_conn_stat_param* **conn_stat**

A2DP connection status

struct *esp_a2d_cb_param_t*::*a2d_audio_stat_param* **audio_stat**

audio stream playing state

struct *esp_a2d_cb_param_t*::*a2d_audio_cfg_param* **audio_cfg**

media codec configuration information

struct *esp_a2d_cb_param_t*::*media_ctrl_stat_param* **media_ctrl_stat**

status in acknowledgement to media control commands

struct a2d_audio_cfg_param

#include <esp_a2dp_api.h> ESP_A2D_AUDIO_CFG_EVT.

Public Members

esp_bd_addr_t **remote_bda**

remote bluetooth device address

esp_a2d_mcc_t **mcc**

A2DP media codec capability information

struct a2d_audio_stat_param

#include <esp_a2dp_api.h> ESP_A2D_AUDIO_STATE_EVT.

Public Members

esp_a2d_audio_state_t **state**

one of the values from esp_a2d_audio_state_t

esp_bd_addr_t **remote_bda**

remote bluetooth device address

struct a2d_conn_stat_param

#include <esp_a2dp_api.h> ESP_A2D_CONNECTION_STATE_EVT.

Public Members

esp_a2d_connection_state_t **state**

one of values from esp_a2d_connection_state_t

esp_bd_addr_t **remote_bda**

remote bluetooth device address

esp_a2d_disc_rsn_t **disc_rsn**

reason of disconnection for "DISCONNECTED"

struct media_ctrl_stat_param

#include <esp_a2dp_api.h> ESP_A2D_MEDIA_CTRL_ACK_EVT.

Public Members

esp_a2d_media_ctrl_t **cmd**

media control commands to acknowledge

esp_a2d_media_ctrl_ack_t **status**

acknowledgement to media control commands

Structures

`struct esp_a2d_mcc_t`

A2DP media codec capabilities union.

Public Members

esp_a2d_mct_t **type**

A2DP media codec type

union *esp_a2d_mcc_t*::[**anonymous**] **cie**

A2DP codec information element

Macros

ESP_A2D_MCT_SBC

Media codec types supported by A2DP.

SBC

ESP_A2D_MCT_M12

MPEG-1, 2 Audio

ESP_A2D_MCT_M24

MPEG-2, 4 AAC

ESP_A2D_MCT_ATRAC

ATRAC family

ESP_A2D_MCT_NON_A2DP

ESP_A2D_CIE_LEN_SBC

ESP_A2D_CIE_LEN_M12

ESP_A2D_CIE_LEN_M24

ESP_A2D_CIE_LEN_ATRAC

Type Definitions

typedef uint8_t **esp_a2d_mct_t**

typedef void (***esp_a2d_cb_t**)(*esp_a2d_cb_event_t* event, *esp_a2d_cb_param_t* *param)

A2DP profile callback function type.

Parameters

- `event`: : Event type
- `param`: : Pointer to callback parameter

typedef void (***esp_a2d_sink_data_cb_t**)(**const** uint8_t *buf, uint32_t len)

A2DP profile data callback function.

Parameters

- `buf`: : data received from A2DP source device and is PCM format decoder from SBC decoder; buf references to a static memory block and can be overwritten by upcoming data
- `len`: : size(in bytes) in buf

typedef int32_t (***esp_a2d_source_data_cb_t**)(uint8_t *buf, int32_t len)

A2DP source data read callback function.

Return size of bytes read successfully, if the argument len is -1, this value is ignored.

Parameters

- `buf`: : buffer to be filled with PCM data stream from higher layer
- `len`: : size(in bytes) of data block to be copied to buf. -1 is an indication to user that data buffer shall be flushed

Enumerations

enum esp_a2d_connection_state_t

Bluetooth A2DP connection states.

Values:

`ESP_A2D_CONNECTION_STATE_DISCONNECTED` = 0

connection released

`ESP_A2D_CONNECTION_STATE_CONNECTING`

connecting remote device

`ESP_A2D_CONNECTION_STATE_CONNECTED`

connection established

`ESP_A2D_CONNECTION_STATE_DISCONNECTING`

disconnecting remote device

enum esp_a2d_disc_rsn_t

Bluetooth A2DP disconnection reason.

Values:

ESP_A2D_DISC_RSN_NORMAL = 0

Finished disconnection that is initiated by local or remote device

ESP_A2D_DISC_RSN_ABNORMAL

Abnormal disconnection caused by signal loss

enum esp_a2d_audio_state_t

Bluetooth A2DP datapath states.

Values:

ESP_A2D_AUDIO_STATE_REMOTE_SUSPEND = 0

audio stream datapath suspended by remote device

ESP_A2D_AUDIO_STATE_STOPPED

audio stream datapath stopped

ESP_A2D_AUDIO_STATE_STARTED

audio stream datapath started

enum esp_a2d_media_ctrl_ack_t

A2DP media control command acknowledgement code.

Values:

ESP_A2D_MEDIA_CTRL_ACK_SUCCESS = 0

media control command is acknowledged with success

ESP_A2D_MEDIA_CTRL_ACK_FAILURE

media control command is acknowledged with failure

ESP_A2D_MEDIA_CTRL_ACK_BUSY

media control command is rejected, as previous command is not yet acknowledged

enum esp_a2d_media_ctrl_t

A2DP media control commands.

Values:

ESP_A2D_MEDIA_CTRL_NONE = 0

dummy command

ESP_A2D_MEDIA_CTRL_CHECK_SRC_RDY

check whether AVDTP is connected, only used in A2DP source

ESP_A2D_MEDIA_CTRL_START

command to set up media transmission channel

ESP_A2D_MEDIA_CTRL_STOP

command to stop media transmission

ESP_A2D_MEDIA_CTRL_SUSPEND

command to suspend media transmission

enum esp_a2d_cb_event_t

A2DP callback events.

Values:

ESP_A2D_CONNECTION_STATE_EVT = 0

connection state changed event

ESP_A2D_AUDIO_STATE_EVT

audio stream transmission state changed event

ESP_A2D_AUDIO_CFG_EVT

audio codec is configured, only used for A2DP SINK

ESP_A2D_MEDIA_CTRL_ACK_EVT

acknowledge event in response to media control commands

BT AVRCP APIs

Overview

Bluetooth AVRCP reference APIs.

Instructions

Application Example

Instructions

API Reference

Header File

- bt/host/bluedroid/api/include/api/esp_avrc_api.h

Functions

esp_err_t esp_avrc_ct_register_callback(*esp_avrc_ct_cb_t callback*)

Register application callbacks to AVRCP module. This function should be called after esp_bluedroid_enable() completes successfully.

Return

- ESP_OK: success
- ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled

- ESP_FAIL: others

Parameters

- `callback`: AVRCP controller callback function

esp_err_t **esp_avrc_ct_init**(void)

Initialize the bluetooth AVRCP controller module, This function should be called after esp_bluedroid_enable() completes successfully.

Return

- ESP_OK: success
- ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

esp_err_t **esp_avrc_ct_deinit**(void)

De-initialize AVRCP controller module. This function should be called after after esp_bluedroid_enable() completes successfully.

Return

- ESP_OK: success
- ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

esp_err_t **esp_avrc_ct_send_set_player_value_cmd**(uint8_t *tl*, uint8_t *attr_id*, uint8_t *value_id*)

Send player application settings command to AVRCP target. This function should be called after ESP_AVRC_CT_CONNECTION_STATE_EVT is received and AVRCP connection is established.

Return

- ESP_OK: success
- ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

Parameters

- `tl`: : transaction label, 0 to 15, consecutive commands should use different values.
- `attr_id`: : player application setting attribute IDs from one of esp_avrc_ps_attr_ids_t
- `value_id`: : attribute value defined for the specific player application setting attribute

esp_err_t **esp_avrc_ct_send_get_rn_capabilities_cmd**(uint8_t *tl*)

Send GetCapabilities PDU to AVRCP target to retrieve remote device' s supported notification event_ids. This function should be called after ESP_AVRC_CT_CONNECTION_STATE_EVT is received and AVRCP connection is established.

Return

- ESP_OK: success
- ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

Parameters

- `tl`: : transaction label, 0 to 15, consecutive commands should use different values.

esp_err_t **esp_avrc_ct_send_register_notification_cmd**(uint8_t *tl*, uint8_t *event_id*, uint32_t *event_parameter*)

Send register notification command to AVRCP target, This function should be called after ESP_AVRC_CT_CONNECTION_STATE_EVT is received and AVRCP connection is established.

Return

- ESP_OK: success
- ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_ERR_NOT_SUPPORTED: if the event_id is not supported in current implementation
- ESP_FAIL: others

Parameters

- `tl`: : transaction label, 0 to 15, consecutive commands should use different values.
- `event_id`: : id of events, e.g. ESP_AVRC_RN_PLAY_STATUS_CHANGE, ESP_AVRC_RN_TRACK_CHANGE, etc.
- `event_parameter`: : playback interval for ESP_AVRC_RN_PLAY_POS_CHANGED; For other events , value of this parameter is ignored.

esp_err_t **esp_avrc_ct_send_set_absolute_volume_cmd**(uint8_t *tl*, uint8_t *volume*)

Send set absolute volume command to AVRCP target, This function should be called after ESP_AVRC_CT_CONNECTION_STATE_EVT is received and AVRCP connection is established.

Return

- ESP_OK: success
- ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_ERR_NOT_SUPPORTED: if the event_id is not supported in current implementation

- ESP_FAIL: others

Parameters

- `tl`: : transaction label, 0 to 15, consecutive commands should use different values.
- `volume`: : volume, 0 to 0x7f, means 0% to 100%

esp_err_t **esp_avrc_ct_send_metadata_cmd**(uint8_t *tl*, uint8_t *attr_mask*)

Send metadata command to AVRCP target, This function should be called after ESP_AVRC_CT_CONNECTION_STATE_EVT is received and AVRCP connection is established.

Return

- ESP_OK: success
- ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

Parameters

- `tl`: : transaction label, 0 to 15, consecutive commands should use different values.
- `attr_mask`: : mask of attributes, e.g. ESP_AVRC_MD_ATTR_ID_TITLE | ESP_AVRC_MD_ATTR_ID_ARTIST.

esp_err_t **esp_avrc_ct_send_passthrough_cmd**(uint8_t *tl*, uint8_t *key_code*, uint8_t *key_state*)

Send passthrough command to AVRCP target, This function should be called after ESP_AVRC_CT_CONNECTION_STATE_EVT is received and AVRCP connection is established.

Return

- ESP_OK: success
- ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

Parameters

- `tl`: : transaction label, 0 to 15, consecutive commands should use different values.
- `key_code`: : passthrough command code, e.g. ESP_AVRC_PT_CMD_PLAY, ESP_AVRC_PT_CMD_STOP, etc.
- `key_state`: : passthrough command key state, ESP_AVRC_PT_CMD_STATE_PRESSED or ESP_AVRC_PT_CMD_STATE_RELEASED

esp_err_t **esp_avrc_tg_register_callback**(*esp_avrc_tg_cb_t callback*)

Register application callbacks to AVRCP target module; This function should be called after esp_bluedroid_enable() completes successfully.

Return

- ESP_OK: success
- ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

Parameters

- `callback`: AVRCP target callback function

esp_err_t **esp_avrc_tg_init**(void)

Initialize the bluetooth AVRCP target module, This function should be called after esp_bluedroid_enable() completes successfully.

Return

- ESP_OK: success
- ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

esp_err_t **esp_avrc_tg_deinit**(void)

De-initialize AVRCP target module. This function should be called after after esp_bluedroid_enable() completes successfully.

Return

- ESP_OK: success
- ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

esp_err_t **esp_avrc_tg_get_psth_cmd_filter**(*esp_avrc_psth_filter_t filter*, *esp_avrc_psth_bit_mask_t* **cmd_set*)

Get the current filter of remote passthrough commands on AVRC target. Filter is given by filter type and bit mask for the passthrough commands. This function should be called after esp_avrc_tg_init(). For filter type ESP_AVRC_PSTH_FILTER_ALLOWED_CMD, the retrieved command set is constant and it covers all of the passthrough commands that can possibly be supported. For filter type ESP_AVRC_PSTH_FILTER_SUPPORT_COMMANDS, the retrieved command set covers the passthrough commands selected to be supported according to current configuration. The configuration can be changed using esp_avrc_tg_set_psth_cmd_filter()

Return

- ESP_OK: success

- ESP_ERR_INVALID_STATE: if bluetooth stack is not enabled or AVRC TG is not initialized
- ESP_ERR_INVALID_ARG: if filter type is invalid or cmd_set is NULL
- ESP_FAIL: otherwise

esp_err_t **esp_avrc_tg_set_psth_cmd_filter**(*esp_avrc_psth_filter_t filter*, **const** *esp_avrc_psth_bit_mask_t* **cmd_set*)

Set the filter of remote passthrough commands on AVRC target. Filter is given by filter type and bit mask for the passthrough commands. This function should be called after esp_avrc_tg_init(). If filter type is ESP_AVRC_PSTH_FILTER_SUPPORT_CMD, the passthrough commands which are set "1" as given in cmd_set will generate ESP_AVRC_CT_PASSTHROUGH_RSP_EVT callback event and are auto-accepted in the protocol stack, other commands are replied with response type "NOT IMPLEMENTED" (8). The set of supported commands should be a subset of allowed command set. The allowed command set can be retrieved using esp_avrc_tg_get_psth_cmd_filter() with filter type "ESP_AVRC_PSTH_FILTER_ALLOWED_CMD" .

Filter type "ESP_AVRC_PSTH_FILTER_ALLOWED_CMD" does not apply to this function

Return

- ESP_OK: success
- ESP_ERR_INVALID_STATE: if bluetooth stack is not enabled
- ESP_ERR_INVALID_ARG: if filter type is invalid or cmd_set is NULL
- ESP_ERR_NOT_SUPPORTED:: if filter type is ESP_AVRC_PSTH_FILTER_ALLOWED_CMD, or cmd_set includes unallowed commands

bool **esp_avrc_psth_bit_mask_operation**(*esp_avrc_bit_mask_op_t op*, *esp_avrc_psth_bit_mask_t* **psth*, *esp_avrc_pt_cmd_t cmd*)

Operate on the type *esp_avrc_psth_bit_mask_t* with regard to a specific PASSTHROUGH command.

Return For operation ESP_AVRC_BIT_MASK_OP_SET or ESP_AVRC_BIT_MASK_OP_CLEAR, return true for a successful operation, otherwise return false For operation ESP_AVRC_BIT_MASK_OP_TEST, return true if the corresponding bit is set, otherwise false

Parameters

- `op`: operation requested on the bit mask field
- `psth`: pointer to passthrough command bit mask structure
- `cmd`: passthrough command code

esp_err_t **esp_avrc_tg_get_rn_evt_cap**(*esp_avrc_rn_evt_cap_t* *cap*, *esp_avrc_rn_evt_cap_mask_t* **evt_set*)

Get the requested event notification capabilies on local AVRC target. The capability is returned in a bit mask representation in evt_set. This function should be called after esp_avrc_tg_init(). For capability type "ESP_AVRC_RN_CAP_ALLOWED_EVT, the retrieved event set is constant and it covers all of the notifcation events that can possibly be supported with current implementation. For capability type ESP_AVRC_RN_CAP_SUPPORTED_EVT, the event set covers the notification events selected to be supported under current configuration, The configuration can be changed using esp_avrc_tg_set_rn_evt_cap()

Return

- ESP_OK: success
- ESP_ERR_INVALID_STATE: if bluetooth stack is not enabled or AVRC TG is not initialized
- ESP_ERR_INVALID_ARG: if cap is invalid or evt_set is NULL
- ESP_FAIL: otherwise

esp_err_t **esp_avrc_tg_set_rn_evt_cap**(**const** *esp_avrc_rn_evt_cap_mask_t* **evt_set*)

Set the event notification capabilities on local AVRCP target. The capability is given in a bit mask representation in evt_set and must be a subset of allowed event IDs with current implementation. This function should be called after esp_avrc_tg_init().

Return

- ESP_OK: success
- ESP_ERR_INVALID_STATE: if bluetooth stack is not enabled
- ESP_ERR_INVALID_ARG: if evt_set is NULL

bool **esp_avrc_rn_evt_bit_mask_operation**(*esp_avrc_bit_mask_op_t* *op*, *esp_avrc_rn_evt_cap_mask_t* **events*, *esp_avrc_rn_event_ids_t* *event_id*)

Operate on the type *esp_avrc_rn_evt_cap_mask_t* with regard to a specific event.

Return For operation ESP_AVRC_BIT_MASK_OP_SET or ESP_AVRC_BIT_MASK_OP_CLEAR, return true for a successful operation, otherwise return false For operation ESP_AVRC_BIT_MASK_OP_TEST, return true if the corresponding bit is set, otherwise false

Parameters

- `op`: operation requested on the bit mask field
- `events`: pointer to event notification capability bit mask structure

- `event_id`: notification event code

esp_err_t **esp_avrc_tg_send_rn_rsp**(*esp_avrc_rn_event_ids_t event_id*, *esp_avrc_rn_rsp_t rsp*, *esp_avrc_rn_param_t *param*)

Send RegisterNotification Response to remote AVRCP controller. Local event notification capability can be set using esp_avrc_tg_set_rn_evt_cap(), in a bit mask representation in evt_set. This function should be called after esp_avrc_tg_init()

Return

- ESP_OK: success
- ESP_ERR_INVALID_STATE: if bluetooth stack is not enabled or AVRC TG is not initialized
- ESP_ERR_INVALID_ARG: if evt_set is NULL

Parameters

- `event_id`: notification event ID that remote AVRCP CT registers
- `rsp`: notification response code
- `param`: parameters included in the specific notification

Unions

union esp_avrc_rn_param_t

#include <esp_avrc_api.h> AVRCP notification parameters.

Public Members

uint8_t **volume**

response data for ESP_AVRC_RN_VOLUME_CHANGE, ranges 0..127

esp_avrc_playback_stat_t **playback**

response data for ESP_AVRC_RN_PLAY_STATUS_CHANGE

uint8_t **elm_id**[8]

response data for ESP_AVRC_RN_TRACK_CHANGE

uint32_t **play_pos**

response data for ESP_AVRC_RN_PLAY_POS_CHANGED, in millisecond

esp_avrc_batt_stat_t **batt**

response data for ESP_AVRC_RN_BATTERY_STATUS_CHANGE

union esp_avrc_ct_cb_param_t

#include <esp_avrc_api.h> AVRC controller callback parameters.

Public Members

struct *esp_avrc_ct_cb_param_t*::*avrc_ct_conn_stat_param* **conn_stat**
AVRC connection status

struct *esp_avrc_ct_cb_param_t*::*avrc_ct_psth_rsp_param* **psth_rsp**
passthrough command response

struct *esp_avrc_ct_cb_param_t*::*avrc_ct_meta_rsp_param* **meta_rsp**
metadata attributes response

struct *esp_avrc_ct_cb_param_t*::*avrc_ct_change_notify_param* **change_ntf**
notifications

struct *esp_avrc_ct_cb_param_t*::*avrc_ct_rmt_feats_param* **rmt_feats**
AVRC features discovered from remote SDP server

struct *esp_avrc_ct_cb_param_t*::*avrc_ct_get_rn_caps_rsp_param* **get_rn_caps_rsp**
get supported event capabilities response from AVRCP target

struct *esp_avrc_ct_cb_param_t*::*avrc_ct_set_volume_rsp_param* **set_volume_rsp**
set absolute volume response event

struct avrc_ct_change_notify_param
#include <esp_avrc_api.h> ESP_AVRC_CT_CHANGE_NOTIFY_EVT.

Public Members

uint8_t **event_id**
id of AVRC event notification

esp_avrc_rn_param_t **event_parameter**
event notification parameter

struct avrc_ct_conn_stat_param
#include <esp_avrc_api.h> ESP_AVRC_CT_CONNECTION_STATE_EVT.

Public Members

bool **connected**
whether AVRC connection is set up

esp_bd_addr_t **remote_bda**
remote bluetooth device address

struct avrc_ct_get_rn_caps_rsp_param
#include <esp_avrc_api.h> ESP_AVRC_CT_GET_RN_CAPABILITIES_RSP_EVT.

Public Members

uint8_t **cap_count**

number of items provided in event or company_id according to cap_id used

esp_avrc_rn_evt_cap_mask_t **evt_set**

supported event_ids represented in bit-mask

struct avrc_ct_meta_rsp_param

#include <esp_avrc_api.h> ESP_AVRC_CT_METADATA_RSP_EVT.

Public Members

uint8_t **attr_id**

id of metadata attribute

uint8_t ***attr_text**

attribute itself

int **attr_length**

attribute character length

struct avrc_ct_psth_rsp_param

#include <esp_avrc_api.h> ESP_AVRC_CT_PASSTHROUGH_RSP_EVT.

Public Members

uint8_t **tl**

transaction label, 0 to 15

uint8_t **key_code**

passthrough command code

uint8_t **key_state**

0 for PRESSED, 1 for RELEASED

struct avrc_ct_rmt_feats_param

#include <esp_avrc_api.h> ESP_AVRC_CT_REMOTE_FEATURES_EVT.

Public Members

uint32_t **feat_mask**

AVRC feature mask of remote device

uint16_t **tg_feat_flag**

feature flag of remote device as TG

esp_bd_addr_t **remote_bda**
remote bluetooth device address

struct avrc_ct_set_volume_rsp_param
#include <esp_avrc_api.h> ESP_AVRC_CT_SET_ABSOLUTE_VOLUME_RSP_EVT.

Public Members

uint8_t **volume**
the volume which has actually been set, range is 0 to 0x7f, means 0% to 100%

union esp_avrc_tg_cb_param_t
#include <esp_avrc_api.h> AVRC target callback parameters.

Public Members

struct *esp_avrc_tg_cb_param_t*::*avrc_tg_conn_stat_param* **conn_stat**
AVRC connection status

struct *esp_avrc_tg_cb_param_t*::*avrc_tg_rmt_feats_param* **rmt_feats**
AVRC features discovered through SDP

struct *esp_avrc_tg_cb_param_t*::*avrc_tg_psth_cmd_param* **psth_cmd**
passthrough command

struct *esp_avrc_tg_cb_param_t*::*avrc_tg_set_abs_vol_param* **set_abs_vol**
set absolute volume command targeted on audio sink

struct *esp_avrc_tg_cb_param_t*::*avrc_tg_reg_ntf_param* **reg_ntf**
register notification

struct avrc_tg_conn_stat_param
#include <esp_avrc_api.h> ESP_AVRC_TG_CONNECTION_STATE_EVT.

Public Members

bool **connected**
whether AVRC connection is set up

esp_bd_addr_t **remote_bda**
remote bluetooth device address

struct avrc_tg_psth_cmd_param
#include <esp_avrc_api.h> ESP_AVRC_TG_PASSTHROUGH_CMD_EVT.

Public Members

uint8_t **key_code**
passthrough command code

uint8_t **key_state**
0 for PRESSED, 1 for RELEASED

struct avrc_tg_reg_ntf_param
#include <esp_avrc_api.h> ESP_AVRC_TG_REGISTER_NOTIFICATION_EVT.

Public Members

uint8_t **event_id**
event id of AVRC RegisterNotification

uint32_t **event_parameter**
event notification parameter

struct avrc_tg_rmt_feats_param
#include <esp_avrc_api.h> ESP_AVRC_TG_REMOTE_FEATURES_EVT.

Public Members

uint32_t **feat_mask**
AVRC feature mask of remote device

uint16_t **ct_feat_flag**
feature flag of remote device as CT

esp_bd_addr_t **remote_bda**
remote bluetooth device address

struct avrc_tg_set_abs_vol_param
#include <esp_avrc_api.h> ESP_AVRC_TG_SET_ABSOLUTE_VOLUME_CMD_EVT.

Public Members

uint8_t **volume**
volume ranges from 0 to 127

Structures

struct esp_avrc_psth_bit_mask_t
AVRC passthrough command bit mask.

Public Members

uint16_t **bits**[8]
bit mask representation of PASSTHROUGH commands

struct esp_avrc_rn_evt_cap_mask_t
AVRC target notification event capability bit mask.

Public Members

uint16_t **bits**
bit mask representation of PASSTHROUGH commands

Macros

ESP_AVRC_TRANS_LABEL_MAX
max transaction label

Type Definitions

typedef void (***esp_avrc_ct_cb_t**)(*esp_avrc_ct_cb_event_t* event, *esp_avrc_ct_cb_param_t* *param)
AVRCP controller callback function type.

Parameters

- `event`: : Event type
- `param`: : Pointer to callback parameter union

typedef void (***esp_avrc_tg_cb_t**)(*esp_avrc_tg_cb_event_t* event, *esp_avrc_tg_cb_param_t* *param)
AVRCP target callback function type.

Parameters

- `event`: : Event type
- `param`: : Pointer to callback parameter union

Enumerations

enum esp_avrc_features_t
AVRC feature bit mask.

Values:

ESP_AVRC_FEAT_RCTG = 0x0001
remote control target

ESP_AVRC_FEAT_RCCT = 0x0002
remote control controller

ESP_AVRC_FEAT_VENDOR = 0x0008
remote control vendor dependent commands

ESP_AVRC_FEAT_BROWSE = 0x0010
use browsing channel

ESP_AVRC_FEAT_META_DATA = 0x0040
remote control metadata transfer command/response

ESP_AVRC_FEAT_ADV_CTRL = 0x0200
remote control advanced control commmand/response

enum esp_avrc_feature_flag_t
AVRC supported features flag retrieved in SDP record.

Values:

ESP_AVRC_FEAT_FLAG_CAT1 = 0x0001
category 1

ESP_AVRC_FEAT_FLAG_CAT2 = 0x0002
category 2

ESP_AVRC_FEAT_FLAG_CAT3 = 0x0004
category 3

ESP_AVRC_FEAT_FLAG_CAT4 = 0x0008
category 4

ESP_AVRC_FEAT_FLAG_BROWSING = 0x0040
browsing

ESP_AVRC_FEAT_FLAG_COVER_ART_GET_IMAGE_PROP = 0x0080
Cover Art GetImageProperties

ESP_AVRC_FEAT_FLAG_COVER_ART_GET_IMAGE = 0x0100
Cover Art GetImage

ESP_AVRC_FEAT_FLAG_COVER_ART_GET_LINKED_THUMBNAIL = 0x0200
Cover Art GetLinkedThumbnail

enum esp_avrc_pt_cmd_t
AVRC passthrough command code.

Values:

`ESP_AVRC_PT_CMD_SELECT` = 0x00
: select

`ESP_AVRC_PT_CMD_UP` = 0x01
: up

`ESP_AVRC_PT_CMD_DOWN` = 0x02
: down

`ESP_AVRC_PT_CMD_LEFT` = 0x03
: left

`ESP_AVRC_PT_CMD_RIGHT` = 0x04
: right

`ESP_AVRC_PT_CMD_RIGHT_UP` = 0x05
: right-up

`ESP_AVRC_PT_CMD_RIGHT_DOWN` = 0x06
: right-down

`ESP_AVRC_PT_CMD_LEFT_UP` = 0x07
: left-up

`ESP_AVRC_PT_CMD_LEFT_DOWN` = 0x08
: left-down

`ESP_AVRC_PT_CMD_ROOT_MENU` = 0x09
: root menu

`ESP_AVRC_PT_CMD_SETUP_MENU` = 0x0A
: setup menu

`ESP_AVRC_PT_CMD_CONT_MENU` = 0x0B
: contents menu

`ESP_AVRC_PT_CMD_FAV_MENU` = 0x0C
: favorite menu

`ESP_AVRC_PT_CMD_EXIT` = 0x0D
: exit

`ESP_AVRC_PT_CMD_0` = 0x20
: 0

`ESP_AVRC_PT_CMD_1` = 0x21
: 1

`ESP_AVRC_PT_CMD_2` = 0x22
: 2

ESP_AVRC_PT_CMD_3 = 0x23
3

ESP_AVRC_PT_CMD_4 = 0x24
4

ESP_AVRC_PT_CMD_5 = 0x25
5

ESP_AVRC_PT_CMD_6 = 0x26
6

ESP_AVRC_PT_CMD_7 = 0x27
7

ESP_AVRC_PT_CMD_8 = 0x28
8

ESP_AVRC_PT_CMD_9 = 0x29
9

ESP_AVRC_PT_CMD_DOT = 0x2A
dot

ESP_AVRC_PT_CMD_ENTER = 0x2B
enter

ESP_AVRC_PT_CMD_CLEAR = 0x2C
clear

ESP_AVRC_PT_CMD_CHAN_UP = 0x30
channel up

ESP_AVRC_PT_CMD_CHAN_DOWN = 0x31
channel down

ESP_AVRC_PT_CMD_PREV_CHAN = 0x32
previous channel

ESP_AVRC_PT_CMD_SOUND_SEL = 0x33
sound select

ESP_AVRC_PT_CMD_INPUT_SEL = 0x34
input select

ESP_AVRC_PT_CMD_DISP_INFO = 0x35
display information

ESP_AVRC_PT_CMD_HELP = 0x36
help

ESP_AVRC_PT_CMD_PAGE_UP = 0x37
page up

ESP_AVRC_PT_CMD_PAGE_DOWN = 0x38
page down

ESP_AVRC_PT_CMD_POWER = 0x40
power

ESP_AVRC_PT_CMD_VOL_UP = 0x41
volume up

ESP_AVRC_PT_CMD_VOL_DOWN = 0x42
volume down

ESP_AVRC_PT_CMD_MUTE = 0x43
mute

ESP_AVRC_PT_CMD_PLAY = 0x44
play

ESP_AVRC_PT_CMD_STOP = 0x45
stop

ESP_AVRC_PT_CMD_PAUSE = 0x46
pause

ESP_AVRC_PT_CMD_RECORD = 0x47
record

ESP_AVRC_PT_CMD_REWIND = 0x48
rewind

ESP_AVRC_PT_CMD_FAST_FORWARD = 0x49
fast forward

ESP_AVRC_PT_CMD_EJECT = 0x4A
eject

ESP_AVRC_PT_CMD_FORWARD = 0x4B
forward

ESP_AVRC_PT_CMD_BACKWARD = 0x4C
backward

ESP_AVRC_PT_CMD_ANGLE = 0x50
angle

ESP_AVRC_PT_CMD_SUBPICT = 0x51
subpicture

ESP_AVRC_PT_CMD_F1 = 0x71
F1

ESP_AVRC_PT_CMD_F2 = 0x72
F2

ESP_AVRC_PT_CMD_F3 = 0x73
F3

ESP_AVRC_PT_CMD_F4 = 0x74
F4

ESP_AVRC_PT_CMD_F5 = 0x75
F5

ESP_AVRC_PT_CMD_VENDOR = 0x7E
vendor unique

enum esp_avrc_psth_filter_t
AVRC passthrough command filter.

Values:

ESP_AVRC_PSTH_FILTER_ALLOWED_CMD = 0
all of the PASSTHROUGH commands that can possibly be used, immuateble

ESP_AVRC_PSTH_FILTER_SUPPORTED_CMD = 1
PASSTHROUGH commands selectively supported according to the current configuration

ESP_AVRC_PSTH_FILTER_SUPPORT_MAX

enum esp_avrc_bit_mask_op_t
Values:

ESP_AVRC_BIT_MASK_OP_TEST = 0
operation code to test a specific bit

ESP_AVRC_BIT_MASK_OP_SET = 1
operation code to set a specific bit

ESP_AVRC_BIT_MASK_OP_CLEAR = 2
operation code to clear a specific bit

enum esp_avrc_pt_cmd_state_t
AVRC passthrough command state.

Values:

ESP_AVRC_PT_CMD_STATE_PRESSED = 0
key pressed

ESP_AVRC_PT_CMD_STATE_RELEASED = 1
key released

enum esp_avrc_ct_cb_event_t

AVRC Controller callback events.

Values:

ESP_AVRC_CT_CONNECTION_STATE_EVT = 0

connection state changed event

ESP_AVRC_CT_PASSTHROUGH_RSP_EVT = 1

passthrough response event

ESP_AVRC_CT_METADATA_RSP_EVT = 2

metadata response event

ESP_AVRC_CT_PLAY_STATUS_RSP_EVT = 3

play status response event

ESP_AVRC_CT_CHANGE_NOTIFY_EVT = 4

notification event

ESP_AVRC_CT_REMOTE_FEATURES_EVT = 5

feature of remote device indication event

ESP_AVRC_CT_GET_RN_CAPABILITIES_RSP_EVT = 6

supported notification events capability of peer device

ESP_AVRC_CT_SET_ABSOLUTE_VOLUME_RSP_EVT = 7

set absolute volume response event

enum esp_avrc_tg_cb_event_t

AVRC Target callback events.

Values:

ESP_AVRC_TG_CONNECTION_STATE_EVT = 0

connection state changed event

ESP_AVRC_TG_REMOTE_FEATURES_EVT = 1

feature of remote device indication event

ESP_AVRC_TG_PASSTHROUGH_CMD_EVT = 2

passthrough command event

ESP_AVRC_TG_SET_ABSOLUTE_VOLUME_CMD_EVT = 3

set absolute volume command from remote device

ESP_AVRC_TG_REGISTER_NOTIFICATION_EVT = 4

register notification event

enum esp_avrc_md_attr_mask_t

AVRC metadata attribute mask.

Values:

ESP_AVRC_MD_ATTR_TITLE = 0x1
: title of the playing track

ESP_AVRC_MD_ATTR_ARTIST = 0x2
: track artist

ESP_AVRC_MD_ATTR_ALBUM = 0x4
: album name

ESP_AVRC_MD_ATTR_TRACK_NUM = 0x8
: track position on the album

ESP_AVRC_MD_ATTR_NUM_TRACKS = 0x10
: number of tracks on the album

ESP_AVRC_MD_ATTR_GENRE = 0x20
: track genre

ESP_AVRC_MD_ATTR_PLAYING_TIME = 0x40
: total album playing time in miliseconds

enum esp_avrc_rn_event_ids_t

AVRC event notification ids.

Values:

ESP_AVRC_RN_PLAY_STATUS_CHANGE = 0x01
: track status change, eg. from playing to paused

ESP_AVRC_RN_TRACK_CHANGE = 0x02
: new track is loaded

ESP_AVRC_RN_TRACK_REACHED_END = 0x03
: current track reached end

ESP_AVRC_RN_TRACK_REACHED_START = 0x04
: current track reached start position

ESP_AVRC_RN_PLAY_POS_CHANGED = 0x05
: track playing position changed

ESP_AVRC_RN_BATTERY_STATUS_CHANGE = 0x06
: battery status changed

ESP_AVRC_RN_SYSTEM_STATUS_CHANGE = 0x07
: system status changed

ESP_AVRC_RN_APP_SETTING_CHANGE = 0x08
: application settings changed

ESP_AVRC_RN_NOW_PLAYING_CHANGE = 0x09
: now playing content changed

ESP_AVRC_RN_AVAILABLE_PLAYERS_CHANGE = 0x0a
available players changed

ESP_AVRC_RN_ADDRESSED_PLAYER_CHANGE = 0x0b
the addressed player changed

ESP_AVRC_RN_UIDS_CHANGE = 0x0c
UIDs changed

ESP_AVRC_RN_VOLUME_CHANGE = 0x0d
volume changed locally on TG

ESP_AVRC_RN_MAX_EVT

enum esp_avrc_rn_evt_cap_t
AVRC target notification event notification capability.

Values:

ESP_AVRC_RN_CAP_ALLOWED_EVT = 0
all of the notification events that can possibly be supported, immutable

ESP_AVRC_RN_CAP_SUPPORTED_EVT = 1
notification events selectively supported according to the current configuration

ESP_AVRC_RN_CAP_MAX

enum esp_avrc_rn_rsp_t
AVRC notification response type.

Values:

ESP_AVRC_RN_RSP_INTERIM = 13
initial response to RegisterNotification, should be sent T_mtp(1000ms) from receiving the command

ESP_AVRC_RN_RSP_CHANGED = 15
final response to RegisterNotification command

enum esp_avrc_ps_attr_ids_t
AVRC player setting ids.

Values:

ESP_AVRC_PS_EQUALIZER = 0x01
equalizer, on or off

ESP_AVRC_PS_REPEAT_MODE = 0x02
repeat mode

ESP_AVRC_PS_SHUFFLE_MODE = 0x03
shuffle mode

ESP_AVRC_PS_SCAN_MODE = 0x04
scan mode on or off

ESP_AVRC_PS_MAX_ATTR

enum esp_avrc_ps_eq_value_ids_t
AVRC equalizer modes.

Values:

ESP_AVRC_PS_EQUALIZER_OFF = 0x1
equalizer OFF

ESP_AVRC_PS_EQUALIZER_ON = 0x2
equalizer ON

enum esp_avrc_ps_rpt_value_ids_t
AVRC repeat modes.

Values:

ESP_AVRC_PS_REPEAT_OFF = 0x1
repeat mode off

ESP_AVRC_PS_REPEAT_SINGLE = 0x2
single track repeat

ESP_AVRC_PS_REPEAT_GROUP = 0x3
group repeat

enum esp_avrc_ps_shf_value_ids_t
AVRC shuffle modes.

Values:

ESP_AVRC_PS_SHUFFLE_OFF = 0x1

ESP_AVRC_PS_SHUFFLE_ALL = 0x2

ESP_AVRC_PS_SHUFFLE_GROUP = 0x3

enum esp_avrc_ps_scn_value_ids_t
AVRC scan modes.

Values:

ESP_AVRC_PS_SCAN_OFF = 0x1
scan off

ESP_AVRC_PS_SCAN_ALL = 0x2
all tracks scan

ESP_AVRC_PS_SCAN_GROUP = 0x3
group scan

enum esp_avrc_rsp_t

AVCTP response codes.

Values:

ESP_AVRC_RSP_NOT_IMPL = 8

not implemented

ESP_AVRC_RSP_ACCEPT = 9

accept

ESP_AVRC_RSP_REJECT = 10

reject

ESP_AVRC_RSP_IN_TRANS = 11

in transition

ESP_AVRC_RSP_IMPL_STBL = 12

implemented/stable

ESP_AVRC_RSP_CHANGED = 13

changed

ESP_AVRC_RSP_INTERIM = 15

interim

enum esp_avrc_batt_stat_t

AVRCP battery status.

Values:

ESP_AVRC_BATT_NORMAL = 0

normal state

ESP_AVRC_BATT_WARNING = 1

unable to operate soon

ESP_AVRC_BATT_CRITICAL = 2

cannot operate any more

ESP_AVRC_BATT_EXTERNAL = 3

plugged to external power supply

ESP_AVRC_BATT_FULL_CHARGE = 4

when completely charged from external power supply

enum esp_avrc_playback_stat_t

AVRCP current status of playback.

Values:

ESP_AVRC_PLAYBACK_STOPPED = 0

stopped

`ESP_AVRC_PLAYBACK_PLAYING` = 1
playing

`ESP_AVRC_PLAYBACK_PAUSED` = 2
paused

`ESP_AVRC_PLAYBACK_FWD_SEEK` = 3
forward seek

`ESP_AVRC_PLAYBACK_REV_SEEK` = 4
reverse seek

`ESP_AVRC_PLAYBACK_ERROR` = 0xFF
error

SPP API

Overview

Instructions

Application Example

Check bluetooth/bluedroid/classic_bt folder in ESP-IDF examples, which contains the following application:

- This is a SPP demo. This demo can discover the service, connect, send and recive SPP data bluetooth/bluedroid/classic_bt/bt_spp_acceptor, bluetooth/bluedroid/classic_bt/bt_spp_initiator

API Reference

Header File

- bt/host/bluedroid/api/include/api/esp_spp_api.h

Functions

esp_err_t **esp_spp_register_callback**(esp_spp_cb_t *callback*)
This function is called to init callbacks with SPP module.

Return

- ESP_OK: success
- other: failed

Parameters

- `callback`: pointer to the init callback function.

esp_err_t **esp_spp_init**(*esp_spp_mode_t mode*)

This function is called to init SPP.

Return

- ESP_OK: success
- other: failed

Parameters

- `mode`: Choose the mode of SPP, ESP_SPP_MODE_CB or ESP_SPP_MODE_VFS.

esp_err_t **esp_spp_deinit**(void)

This function is called to uninit SPP.

Return

- ESP_OK: success
- other: failed

esp_err_t **esp_spp_start_discovery**(*esp_bd_addr_t bd_addr*)

This function is called to performs service discovery for the services provided by the given peer device. When the operation is complete the callback function will be called with a ESP_SPP_DISCOVERY_COMP_EVT.

Return

- ESP_OK: success
- other: failed

Parameters

- `bd_addr`: Remote device bluetooth device address.

esp_err_t **esp_spp_connect**(*esp_spp_sec_t sec_mask*, *esp_spp_role_t role*, uint8_t *remote_scn*, *esp_bd_addr_t peer_bd_addr*)

This function makes an SPP connection to a remote BD Address. When the connection is initiated or failed to initiate, the callback is called with ESP_SPP_CL_INIT_EVT. When the connection is established or failed, the callback is called with ESP_SPP_OPEN_EVT.

Return

- ESP_OK: success
- other: failed

Parameters

- `sec_mask`: Security Setting Mask. Suggest to use ESP_SPP_SEC_NONE, ESP_SPP_SEC_AUTHORIZE or ESP_SPP_SEC_AUTHENTICATE only.
- `role`: Master or slave.
- `remote_scn`: Remote device bluetooth device SCN.
- `peer_bd_addr`: Remote device bluetooth device address.

esp_err_t **esp_spp_disconnect**(uint32_t *handle*)

This function closes an SPP connection.

Return

- ESP_OK: success
- other: failed

Parameters

- `handle`: The connection handle.

esp_err_t **esp_spp_start_srv**(*esp_spp_sec_t sec_mask*, *esp_spp_role_t role*, uint8_t *local_scn*, **const** char ***name*)

This function create a SPP server and starts listening for an SPP connection request from a remote Bluetooth device. When the server is started successfully, the callback is called with ESP_SPP_START_EVT. When the connection is established, the callback is called with ESP_SPP_SRV_OPEN_EVT.

Return

- ESP_OK: success
- other: failed

Parameters

- `sec_mask`: Security Setting Mask. Suggest to use ESP_SPP_SEC_NONE, ESP_SPP_SEC_AUTHORIZE or ESP_SPP_SEC_AUTHENTICATE only.
- `role`: Master or slave.
- `local_scn`: The specific channel you want to get. If channel is 0, means get any channel.
- `name`: Server' s name.

esp_err_t **esp_spp_write**(uint32_t *handle*, int *len*, uint8_t ***p_data*)

This function is used to write data, only for ESP_SPP_MODE_CB.

Return

- ESP_OK: success

- other: failed

Parameters

- `handle`: The connection handle.
- `len`: The length of the data written.
- `p_data`: The data written.

esp_err_t **esp_spp_vfs_register**(void)

This function is used to register VFS.

Return

- ESP_OK: success
- other: failed

Unions

union esp_spp_cb_param_t

#include <esp_spp_api.h> SPP callback parameters union.

Public Members

struct *esp_spp_cb_param_t*::*spp_init_evt_param* **init**

SPP callback param of SPP_INIT_EVT

struct *esp_spp_cb_param_t*::*spp_discovery_comp_evt_param* **disc_comp**

SPP callback param of SPP_DISCOVERY_COMP_EVT

struct *esp_spp_cb_param_t*::*spp_open_evt_param* **open**

SPP callback param of ESP_SPP_OPEN_EVT

struct *esp_spp_cb_param_t*::*spp_srv_open_evt_param* **srv_open**

SPP callback param of ESP_SPP_SRV_OPEN_EVT

struct *esp_spp_cb_param_t*::*spp_close_evt_param* **close**

SPP callback param of ESP_SPP_CLOSE_EVT

struct *esp_spp_cb_param_t*::*spp_start_evt_param* **start**

SPP callback param of ESP_SPP_START_EVT

struct *esp_spp_cb_param_t*::*spp_cl_init_evt_param* **cl_init**

SPP callback param of ESP_SPP_CL_INIT_EVT

struct *esp_spp_cb_param_t*::*spp_write_evt_param* **write**

SPP callback param of ESP_SPP_WRITE_EVT

struct *esp_spp_cb_param_t*::*spp_data_ind_evt_param* **data_ind**

SPP callback param of ESP_SPP_DATA_IND_EVT

struct *esp_spp_cb_param_t*::*spp_cong_evt_param* **cong**

SPP callback param of ESP_SPP_CONG_EVT

struct spp_cl_init_evt_param

#include <esp_spp_api.h> ESP_SPP_CL_INIT_EVT.

Public Members

esp_spp_status_t **status**

status

uint32_t **handle**

The connection handle

uint8_t **sec_id**

security ID used by this server

bool **use_co**

TRUE to use co_rfc_data

struct spp_close_evt_param

#include <esp_spp_api.h> ESP_SPP_CLOSE_EVT.

Public Members

esp_spp_status_t **status**

status

uint32_t **port_status**

PORT status

uint32_t **handle**

The connection handle

bool **async**

FALSE, if local initiates disconnect

struct spp_cong_evt_param

#include <esp_spp_api.h> ESP_SPP_CONG_EVT.

Public Members

esp_spp_status_t **status**

status

uint32_t **handle**

The connection handle

bool **cong**

TRUE, congested. FALSE, uncongested

struct spp_data_ind_evt_param

#include <esp_spp_api.h> ESP_SPP_DATA_IND_EVT.

Public Members

esp_spp_status_t **status**

status

uint32_t **handle**

The connection handle

uint16_t **len**

The length of data

uint8_t ***data**

The data received

struct spp_discovery_comp_evt_param

#include <esp_spp_api.h> SPP_DISCOVERY_COMP_EVT.

Public Members

esp_spp_status_t **status**

status

uint8_t **scn_num**

The num of scn_num

uint8_t **scn**[**ESP_SPP_MAX_SCN**]

channel #

struct spp_init_evt_param

#include <esp_spp_api.h> SPP_INIT_EVT.

Public Members

esp_spp_status_t **status**

status

struct spp_open_evt_param

#include <esp_spp_api.h> ESP_SPP_OPEN_EVT.

Public Members

esp_spp_status_t **status**
status

uint32_t **handle**
The connection handle

int **fd**
The file descriptor only for ESP_SPP_MODE_VFS

esp_bd_addr_t **rem_bda**
The peer address

struct spp_srv_open_evt_param
#include <esp_spp_api.h> ESP_SPP_SRV_OPEN_EVT.

Public Members

esp_spp_status_t **status**
status

uint32_t **handle**
The connection handle

uint32_t **new_listen_handle**
The new listen handle

int **fd**
The file descriptor only for ESP_SPP_MODE_VFS

esp_bd_addr_t **rem_bda**
The peer address

struct spp_start_evt_param
#include <esp_spp_api.h> ESP_SPP_START_EVT.

Public Members

esp_spp_status_t **status**
status

uint32_t **handle**
The connection handle

uint8_t **sec_id**
security ID used by this server

bool **use_co**

TRUE to use co_rfc_data

struct spp_write_evt_param

#include <esp_spp_api.h> ESP_SPP_WRITE_EVT.

Public Members

esp_spp_status_t **status**

status

uint32_t **handle**

The connection handle

int **len**

The length of the data written.

bool **cong**

congestion status

Macros

ESP_SPP_SEC_NONE

No security. relate to BTA_SEC_NONE in bta/bta_api.h

ESP_SPP_SEC_AUTHORIZE

Authorization required (only needed for out going connection) relate to BTA_SEC_AUTHORIZE in bta/bta_api.h

ESP_SPP_SEC_AUTHENTICATE

Authentication required. relate to BTA_SEC_AUTHENTICATE in bta/bta_api.h

ESP_SPP_SEC_ENCRYPT

Encryption required. relate to BTA_SEC_ENCRYPT in bta/bta_api.h

ESP_SPP_SEC_MODE4_LEVEL4

Mode 4 level 4 service, i.e. incoming/outgoing MITM and P-256 encryption relate to BTA_SEC_MODE4_LEVEL4 in bta/bta_api.h

ESP_SPP_SEC_MITM

Man-In-The_Middle protection relate to BTA_SEC_MITM in bta/bta_api.h

ESP_SPP_SEC_IN_16_DIGITS

Min 16 digit for pin code relate to BTA_SEC_IN_16_DIGITS in bta/bta_api.h

ESP_SPP_MAX_MTU

SPP max MTU

ESP_SPP_MAX_SCN
SPP max SCN

Type Definitions

typedef uint16_t esp_spp_sec_t

typedef void() esp_spp_cb_t(esp_spp_cb_event_t event, esp_spp_cb_param_t *param)
SPP callback function type.

Parameters

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

Enumerations

enum esp_spp_status_t
Values:

ESP_SPP_SUCCESS = 0
Successful operation.

ESP_SPP_FAILURE
Generic failure.

ESP_SPP_BUSY
Temporarily can not handle this request.

ESP_SPP_NO_DATA
no data.

ESP_SPP_NO_RESOURCE
No more set pm control block

enum esp_spp_role_t
Values:

ESP_SPP_ROLE_MASTER = 0
Role: master

ESP_SPP_ROLE_SLAVE = 1
Role: slave

enum esp_spp_mode_t
Values:

ESP_SPP_MODE_CB = 0

When data is coming, a callback will come with data

ESP_SPP_MODE_VFS = 1

Use VFS to write/read data

enum esp_spp_cb_event_t

SPP callback function events.

Values:

ESP_SPP_INIT_EVT = 0

When SPP is inited, the event comes

ESP_SPP_DISCOVERY_COMP_EVT = 8

When SDP discovery complete, the event comes

ESP_SPP_OPEN_EVT = 26

When SPP Client connection open, the event comes

ESP_SPP_CLOSE_EVT = 27

When SPP connection closed, the event comes

ESP_SPP_START_EVT = 28

When SPP server started, the event comes

ESP_SPP_CL_INIT_EVT = 29

When SPP client initiated a connection, the event comes

ESP_SPP_DATA_IND_EVT = 30

When SPP connection received data, the event comes, only for ESP_SPP_MODE_CB

ESP_SPP_CONG_EVT = 31

When SPP connection congestion status changed, the event comes, only for ESP_SPP_MODE_CB

ESP_SPP_WRITE_EVT = 33

When SPP write operation completes, the event comes, only for ESP_SPP_MODE_CB

ESP_SPP_SRV_OPEN_EVT = 34

When SPP Server connection open, the event comes

HFP DEFINES

Overview

Instructions

API Reference

Header File

- bt/host/bluedroid/api/include/api/esp_hf_defs.h

Macros

ESP_BT_HF_NUMBER_LEN

ESP_BT_HF_OPERATOR_NAME_LEN

BTC_HSAG_SERVICE_NAME

BTC_HFAG_SERVICE_NAME

BTC_HF_SERVICES

BTC_HF_SERVICE_NAMES

BTC_HF_SECURITY

BTC_HF_CALL_END_TIMEOUT

BTC_HF_INVALID_IDX

Type Definitions

typedef void (***esp_hf_connection_state_callback**)(*esp_hf_connection_state_t* state, *esp_bd_addr_t* *bd_addr)

Callback for connection state change. state will have one of the values from BtHfConnectionState

typedef void (***esp_hf_audio_state_callback**)(*esp_hf_audio_state_t* state, *esp_bd_addr_t* *bd_addr)

Callback for audio connection state change. state will have one of the values from BtHfAudioState

typedef void (***esp_hf_vr_cmd_callback**)(*esp_hf_vr_state_t* state, *esp_bd_addr_t* *bd_addr)

Callback for VR connection state change. state will have one of the values from BtHfVRState

typedef void (***esp_hf_answer_call_cmd_callback**)(*esp_bd_addr_t* *bd_addr)

Callback for answer incoming call (ATA)

typedef void (***esp_hf_hangup_call_cmd_callback**)(*esp_bd_addr_t* *bd_addr)

Callback for disconnect call (AT+CHUP)

typedef void (***esp_hf_volume_cmd_callback**)(*esp_hf_volume_control_target_t* type, int volume, *esp_bd_addr_t* *bd_addr)

Callback for disconnect call (AT+CHUP) type will denote Speaker/Mic gain (BtHfVolumeControl).

typedef void (***esp_hf_dial_call_cmd_callback**)(char *number, *esp_bd_addr_t* *bd_addr)

Callback for dialing an outgoing call If number is NULL, redial

typedef void (***esp_hf_dtmf_cmd_callback**)(char tone, *esp_bd_addr_t* *bd_addr)

Callback for sending DTMF tones tone contains the dtmf character to be sent

typedef void (***esp_hf_nrec_cmd_callback**)(*esp_hf_nrec_t* nrec, *esp_bd_addr_t* *bd_addr)

Callback for enabling/disabling noise reduction/echo cancellation value will be 1 to enable, 0 to disable

typedef void (***esp_hf_wbs_callback**)(*esp_hf_wbs_config_t* wbs, *esp_bd_addr_t* *bd_addr)

Callback for AT+BCS and event from BAC WBS enable, WBS disable

typedef void (***esp_hf_chld_cmd_callback**)(*esp_hf_chld_type_t* chld, *esp_bd_addr_t* *bd_addr)

Callback for call hold handling (AT+CHLD) value will contain the call hold command (0, 1, 2, 3)

typedef void (***esp_hf_cnum_cmd_callback**)(*esp_bd_addr_t* *bd_addr)

Callback for CNUM (subscriber number)

typedef void (***esp_hf_cind_cmd_callback**)(*esp_bd_addr_t* *bd_addr)

Callback for indicators (CIND)

typedef void (***esp_hf_cops_cmd_callback**)(*esp_bd_addr_t* *bd_addr)

Callback for operator selection (COPS)

typedef void (***esp_hf_clcc_cmd_callback**)(*esp_bd_addr_t* *bd_addr)

Callback for call list (AT+CLCC)

typedef void (***esp_hf_unknown_at_cmd_callback**)(char *at_string, *esp_bd_addr_t* *bd_addr)

Callback for unknown AT command recd from AG at_string will contain the unparsed AT string

typedef void (***esp_hf_key_pressed_cmd_callback**)(*esp_bd_addr_t* *bd_addr)

Callback for keypressed (HSP) event.

Enumerations

enum esp_hf_in_band_ring_state_t

in-band ring tone state

Values:

ESP_HF_IN_BAND_RINGTONE_NOT_PROVIDED = 0

ESP_HF_IN_BAND_RINGTONE_PROVIDED

enum esp_hf_vr_state_t

voice recognition state

Values:

ESP_HF_VR_STATE_DISABLED = 0

voice recognition disabled

ESP_HF_VR_STATE_ENABLED

voice recognition enabled

enum esp_hf_volume_control_target_t

Bluetooth HFP audio volume control target.

Values:

ESP_HF_VOLUME_CONTROL_TARGET_SPK = 0

speaker

ESP_HF_VOLUME_CONTROL_TARGET_MIC

microphone

enum esp_hf_audio_state_t

Bluetooth HFP audio connection status.

Values:

ESP_HF_AUDIO_STATE_DISCONNECTED = 0

audio connection released

ESP_HF_AUDIO_STATE_CONNECTING

audio connection has been initiated

ESP_HF_AUDIO_STATE_CONNECTED

audio connection is established

ESP_HF_AUDIO_STATE_CONNECTED_MSBC

mSBC audio connection is established

enum esp_hf_volume_type_t

Values:

ESP_HF_VOLUME_TYPE_SPK = 0

ESP_HF_VOLUME_TYPE_MIC

enum esp_hf_network_state_t

+CIND network service availability status

Values:

ESP_HF_NETWORK_STATE_NOT_AVAILABLE = 0

ESP_HF_NETWORK_STATE_AVAILABLE

enum esp_hf_service_type_t

+CIEV Service type

Values:

ESP_HF_SERVICE_TYPE_HOME = 0

ESP_HF_SERVICE_TYPE_ROAMING

enum esp_hf_call_status_t

+CIND call status indicator values

Values:

ESP_HF_CALL_STATUS_NO_CALLS = 0

no call in progress

ESP_HF_CALL_STATUS_CALL_IN_PROGRESS = 1

call is present(active or held)

enum esp_hf_call_setup_status_t

+CIND call setup status indicator values

Values:

ESP_HF_CALL_SETUP_STATUS_IDLE = 0

no call setup in progress

ESP_HF_CALL_SETUP_STATUS_INCOMING = 1

incoming call setup in progress

ESP_HF_CALL_SETUP_STATUS_OUTGOING_DIALING = 2

outgoing call setup in dialing state

ESP_HF_CALL_SETUP_STATUS_OUTGOING_ALERTING = 3

outgoing call setup in alerting state

enum esp_hf_roaming_status_t

+CIND roaming status indicator values

Values:

ESP_HF_ROAMING_STATUS_INACTIVE = 0

roaming is not active

ESP_HF_ROAMING_STATUS_ACTIVE

a roaming is active

enum esp_hf_call_held_status_t

+CIND call held indicator values

Values:

ESP_HF_CALL_HELD_STATUS_NONE = 0

no calls held

ESP_HF_CALL_HELD_STATUS_HELD_AND_ACTIVE = 1

both active and held call

ESP_HF_CALL_HELD_STATUS_HELD = 2

call on hold, no active call

enum esp_hf_current_call_status_t

+CLCC status of the call

Values:

ESP_HF_CURRENT_CALL_STATUS_ACTIVE = 0

active

ESP_HF_CURRENT_CALL_STATUS_HELD = 1

held

ESP_HF_CURRENT_CALL_STATUS_DIALING = 2

dialing (outgoing calls only)

ESP_HF_CURRENT_CALL_STATUS_ALERTING = 3

alerting (outgoing calls only)

ESP_HF_CURRENT_CALL_STATUS_INCOMING = 4

incoming (incoming calls only)

ESP_HF_CURRENT_CALL_STATUS_WAITING = 5

waiting (incoming calls only)

ESP_HF_CURRENT_CALL_STATUS_HELD_BY_RESP_HOLD = 6

call held by response and hold

enum esp_hf_current_call_direction_t

+CLCC direction of the call

Values:

ESP_HF_CURRENT_CALL_DIRECTION_OUTGOING = 0

outgoing

ESP_HF_CURRENT_CALL_DIRECTION_INCOMING = 1

incoming

enum esp_hf_current_call_mpty_type_t

+CLCC multi-party call flag

Values:

ESP_HF_CURRENT_CALL_MPTY_TYPE_SINGLE = 0

not a member of a multi-party call

ESP_HF_CURRENT_CALL_MPTY_TYPE_MULTI = 1

member of a multi-party call

enum esp_hf_current_call_mode_t

+CLCC call mode

Values:

ESP_HF_CURRENT_CALL_MODE_VOICE = 0

ESP_HF_CURRENT_CALL_MODE_DATA = 1

ESP_HF_CURRENT_CALL_MODE_FAX = 2

enum esp_hf_call_addr_type_t

+CLCC address type

Values:

ESP_HF_CALL_ADDR_TYPE_UNKNOWN = 0x81

unkown address type

ESP_HF_CALL_ADDR_TYPE_INTERNATIONAL = 0x91

international address

enum esp_hf_subscriber_service_type_t

+CNUM service type of the phone number

Values:

ESP_HF_SUBSCRIBER_SERVICE_TYPE_UNKNOWN = 0

unknown

ESP_HF_SUBSCRIBER_SERVICE_TYPE_VOICE

voice service

ESP_HF_SUBSCRIBER_SERVICE_TYPE_FAX

fax service

enum esp_hf_btrh_status_t

+BTRH response and hold result code

Values:

ESP_HF_BTRH_STATUS_HELD = 0

incoming call is put on held in AG

ESP_HF_BTRH_STATUS_ACCEPTED

held incoming call is accepted in AG

ESP_HF_BTRH_STATUS_REJECTED

held incoming call is rejected in AG

enum esp_hf_btrh_cmd_t

AT+BTRH response and hold action code.

Values:

ESP_HF_BTRH_CMD_HOLD = 0

put the incoming call on hold

ESP_HF_BTRH_CMD_ACCEPT = 1
accept a held incoming call

ESP_HF_BTRH_CMD_REJECT = 2
reject a held incoming call

enum esp_hf_nrec_t
Values:

ESP_HF_NREC_STOP = 0

ESP_HF_NREC_START

enum esp_hf_call_waiting_status_t
+CCWA resposne status

Values:

ESP_HF_CALL_WAITING_INACTIVE

ESP_HF_CALL_WAITING_ACTIVE

enum esp_hf_wbs_config_t
Values:

ESP_HF_WBS_NONE

ESP_HF_WBS_NO

ESP_HF_WBS_YES

enum esp_hf_connection_state_t
Bluetooth HFP RFCOMM connection and service level connection status.

Values:

ESP_HF_CONNECTION_STATE_DISCONNECTED = 0
RFCOMM data link channel released

ESP_HF_CONNECTION_STATE_CONNECTING
connecting remote device on the RFCOMM data link

ESP_HF_CONNECTION_STATE_CONNECTED
RFCOMM connection established

ESP_HF_CONNECTION_STATE_SLC_CONNECTED
service level connection established

ESP_HF_CONNECTION_STATE_DISCONNECTING
disconnecting with remote device on the RFCOMM data link

enum esp_hf_chld_type_t
AT+CHLD command values.

Values:

ESP_HF_CHLD_TYPE_REL = 0

<0>, Terminate all held or set UDUB("busy") to a waiting call

ESP_HF_CHLD_TYPE_REL_ACC

<1>, Terminate all active calls and accepts a waiting/held call

ESP_HF_CHLD_TYPE_HOLD_ACC

<2>, Hold all active calls and accepts a waiting/held call

ESP_HF_CHLD_TYPE_MERGE

<3>, Add all held calls to a conference

ESP_HF_CHLD_TYPE_MERGE_DETACH

<4>, connect the two calls and disconnects the subscriber from both calls

ESP_HF_CHLD_TYPE_REL_X

<1x>, releases specified calls only

ESP_HF_CHLD_TYPE_PRIV_X

<2x>, request private consultation mode with specified call

enum esp_hf_at_response_code_t

Values:

ESP_HF_AT_RESPONSE_CODE_OK = 0

acknowledges execution of a command line

ESP_HF_AT_RESPONSE_CODE_ERR

command not accepted

ESP_HF_AT_RESPONSE_CODE_NO_CARRIER

connection terminated

ESP_HF_AT_RESPONSE_CODE_BUSY

busy signal detected

ESP_HF_AT_RESPONSE_CODE_NO_ANSWER

connection completion timeout

ESP_HF_AT_RESPONSE_CODE_DELAYED

delayed

ESP_HF_AT_RESPONSE_CODE_BLACKLISTED

blacklisted

ESP_HF_AT_RESPONSE_CODE_CME

CME error

enum esp_hf_at_response_t

Values:

ESP_HF_AT_RESPONSE_ERROR = 0

ESP_HF_AT_RESPONSE_OK

enum esp_hf_cme_err_t

Extended Audio Gateway Error Result Code Response.

Values:

ESP_HF_CME_AG_FAILURE = 0

ag failure

ESP_HF_CME_NO_CONNECTION_TO_PHONE = 1

no connection to phone

ESP_HF_CME_OPERATION_NOT_ALLOWED = 3

operation not allowed

ESP_HF_CME_OPERATION_NOT_SUPPORTED = 4

operation not supported

ESP_HF_CME_PH_SIM_PIN_REQUIRED = 5

PH-SIM PIN Required

ESP_HF_CME_SIM_NOT_INSERTED = 10

SIM not inserted

ESP_HF_CME_SIM_PIN_REQUIRED = 11

SIM PIN required

ESP_HF_CME_SIM_PUK_REQUIRED = 12

SIM PUK required

ESP_HF_CME_SIM_FAILURE = 13

SIM failure

ESP_HF_CME_SIM_BUSY = 14

SIM busy

ESP_HF_CME_INCORRECT_PASSWORD = 16

incorrect password

ESP_HF_CME_SIM_PIN2_REQUIRED = 17

SIM PIN2 required

ESP_HF_CME_SIM_PUK2_REQUIRED = 18

SIM PUK2 required

ESP_HF_CME_MEMEORY_FULL = 20

memory full

ESP_HF_CME_INVALID_INDEX = 21

invalid index

ESP_HF_CME_MEMEORY_FAILURE = 23
memory failure

ESP_HF_CME_TEXT_STRING_TOO_LONG = 24
test string too long

ESP_HF_CME_INVALID_CHARACTERS_IN_TEXT_STRING = 25
invalid characters in text string

ESP_HF_CME_DIAL_STRING_TOO_LONG = 26
dial string too long

ESP_HF_CME_INVALID_CHARACTERS_IN_DIAL_STRING = 27
invalid characters in dial string

ESP_HF_CME_NO_NETWORK_SERVICE = 30
no network service

ESP_HF_CME_NETWORK_TIMEOUT = 31
network timeout

ESP_HF_CME_NETWORK_NOT_ALLOWED = 32
network not allowed emergency calls only

HFP CLIENT API

Overview

Instructions

API Reference

Header File

- bt/host/bluedroid/api/include/api/esp_hf_client_api.h

Functions

esp_err_t **esp_hf_client_register_callback**(*esp_hf_client_cb_t callback*)
Register application callback function to HFP client module. This function should be called only after esp_bluedroid_enable() completes successfully, used by HFP client.

Return

- ESP_OK: success
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled

- ESP_FAIL: if callback is a NULL function pointer

Parameters

- `callback`: HFP client event callback function

esp_err_t **esp_hf_client_init**(void)

Initialize the bluetooth HFP client module. This function should be called after esp_bluedroid_enable() completes successfully.

Return

- ESP_OK: if the initialization request is sent successfully
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

esp_err_t **esp_hf_client_deinit**(void)

De-initialize for HFP client module. This function should be called only after esp_bluedroid_enable() completes successfully.

Return

- ESP_OK: success
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

esp_err_t **esp_hf_client_connect**(*esp_bd_addr_t remote_bda*)

Connect to remote bluetooth HFP audio gateway(AG) device, must after esp_hf_client_init()

Return

- ESP_OK: connect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

Parameters

- `remote_bda`: remote bluetooth device address

esp_err_t **esp_hf_client_disconnect**(*esp_bd_addr_t remote_bda*)

Disconnect from the remote HFP audio gateway.

Return

- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled

- ESP_FAIL: others

Parameters

- `remote_bda`: remote bluetooth device address

esp_err_t **esp_hf_client_connect_audio**(*esp_bd_addr_t remote_bda*)

Create audio connection with remote HFP AG. As a precondition to use this API, Service Level Connection shall exist with AG.

Return

- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

Parameters

- `remote_bda`: remote bluetooth device address

esp_err_t **esp_hf_client_disconnect_audio**(*esp_bd_addr_t remote_bda*)

Release the established audio connection with remote HFP AG.

Return

- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

Parameters

- `remote_bda`: remote bluetooth device address

esp_err_t **esp_hf_client_start_voice_recognition**(void)

Enable voice recognition in the AG. As a precondition to use this API, Service Level Connection shall exist with AG.

Return

- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

esp_err_t **esp_hf_client_stop_voice_recognition**(void)

Disable voice recognition in the AG. As a precondition to use this API, Service Level Connection shall exist with AG.

Return

- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

esp_err_t **esp_hf_client_volume_update**(*esp_hf_volume_control_target_t type*, int *volume*)

Volume synchronization with AG. As a precondition to use this API, Service Level Connection shall exist with AG.

Return

- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

Parameters

- `type`: volume control target, speaker or microphone
- `volume`: gain of the speaker of microphone, ranges 0 to 15

esp_err_t **esp_hf_client_dial**(**const** char **number*)

Place a call with a specified number, if number is NULL, last called number is called. As a precondition to use this API, Service Level Connection shall exist with AG.

Return

- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

Parameters

- `number`: number string of the call. If NULL, the last number is called(aka re-dial)

esp_err_t **esp_hf_client_dial_memory**(int *location*)

Place a call with number specified by location(speed dial). As a precondition, to use this API, Service Level Connection shall exist with AG.

Return

- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

Parameters

- `location`: location of the number in the memory

esp_err_t **esp_hf_client_send_chld_cmd**(*esp_hf_chld_type_t chld*, int *idx*)

Send call hold and multiparty commands, or enhanced call control commands(Use AT+CHLD). As a precondition to use this API, Service Level Connection shall exist with AG.

Return

- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

Parameters

- `chld`: AT+CHLD call hold and multiparty handling AT command.
- `idx`: used in Enhanced Call Control Mechanisms, used if chld is ESP_HF_CHLD_TYPE_REL_X or ESP_HF_CHLD_TYPE_PRIV_X

esp_err_t **esp_hf_client_send_btrh_cmd**(*esp_hf_btrh_cmd_t btrh*)

Send response and hold action command(Send AT+BTRH command) As a precondition to use this API, Service Level Connection shall exist with AG.

Return

- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

Parameters

- `btrh`: response and hold action to send

esp_err_t **esp_hf_client_answer_call**(void)

Answer an incoming call(send ATA command). As a precondition to use this API, Service Level Connection shall exist with AG.

Return

- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

esp_err_t **esp_hf_client_reject_call**(void)

Reject an incoming call(send AT+CHUP command), As a precondition to use this API, Service Level Connection shall exist with AG.

Return

- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

esp_err_t **esp_hf_client_query_current_calls**(void)

Query list of current calls in AG(send AT+CLCC command), As a precondition to use this API, Service Level Connection shall exist with AG.

Return

- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

esp_err_t **esp_hf_client_query_current_operator_name**(void)

Query the name of currently selected network operator in AG(use AT+COPS commands) As a precondition to use this API, Service Level Connection shall exist with AG.

Return

- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

esp_err_t **esp_hf_client_retrieve_subscriber_info**(void)

Get subscriber information number from AG(send AT+CNUM command) As a precondition to use this API, Service Level Connection shall exist with AG.

Return

- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

esp_err_t **esp_hf_client_send_dtmf**(char *code*)

Transmit DTMF codes during an ongoing call(use AT+VTS commands) As a precondition to use this API, Service Level Connection shall exist with AG.

Return

- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

Parameters

- `code`: dtmf code, single ascii character in the set 0-9, #, *, A-D

esp_err_t **esp_hf_client_request_last_voice_tag_number**(void)

Request a phone number from AG corresponding to last voice tag recorded (send AT+BINP command). As a precondition to use this API, Service Level Connection shall exist with AG.

Return

- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

esp_err_t **esp_hf_client_send_nrec**(void)

Disable echo cancellation and noise reduction in the AG (use AT+NREC=0 command) As a precondition to use this API, Service Level Connection shall exist with AG.

Return

- ESP_OK: NREC=0 request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

esp_err_t **esp_hf_client_register_data_callback**(*esp_hf_client_incoming_data_cb_t recv*, *esp_hf_client_outgoing_data_cb_t send*)

Register HFP client data output function; the callback is only used in the case that Voice Over HCI is enabled.

Return

- ESP_OK: success
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: if callback is a NULL function pointer

Parameters

- `recv`: HFP client incoming data callback function
- `send`: HFP client outgoing data callback function

void **esp_hf_client_outgoing_data_ready**(void)

Trigger the lower-layer to fetch and send audio data. This function is only only used in the case that Voice Over HCI is enabled. Precondition is that the HFP audio connection is connected. After this function is called, lower layer will invoke esp_hf_client_outgoing_data_cb_t to fetch data.

void **esp_hf_client_pcm_resample_init**(uint32_t *src_sps*, uint32_t *bits*, uint32_t *channels*)

Initialize the down sampling converter. This is a utility function that can only be used in the case that Voice Over HCI is enabled.

Parameters

- `src_sps`: original samples per second(source audio data, i.e. 48000, 32000, 16000, 44100, 22050, 11025)
- `bits`: number of bits per pcm sample (16)
- `channels`: number of channels (i.e. mono(1), stereo(2)···)

void **esp_hf_client_pcm_resample_deinit**(void)

Deinitialize the down sampling converter.

int32_t **esp_hf_client_pcm_resample**(void **src*, uint32_t *in_bytes*, void **dst*)

Down sampling utility to convert high sampling rate into 8K/16bits 1-channel mode PCM samples. This can only be used in the case that Voice Over HCI is enabled.

Return number of samples converted

Parameters

- `src`: pointer to the buffer where the original sampling PCM are stored
- `in_bytes`: length of the input PCM sample buffer in byte
- `dst`: pointer to the buffer which is to be used to store the converted PCM samples

Unions

union esp_hf_client_cb_param_t

#include <esp_hf_client_api.h> HFP client callback parameters.

Public Members

struct *esp_hf_client_cb_param_t*::*hf_client_conn_stat_param* **conn_stat**

HF callback param of ESP_HF_CLIENT_CONNECTION_STATE_EVT

struct *esp_hf_client_cb_param_t*::*hf_client_audio_stat_param* **audio_stat**

HF callback param of ESP_HF_CLIENT_AUDIO_STATE_EVT

struct *esp_hf_client_cb_param_t*::*hf_client_bvra_param* **bvra**
HF callback param of ESP_HF_CLIENT_BVRA_EVT

struct *esp_hf_client_cb_param_t*::*hf_client_service_availability_param* **service_availability**
HF callback param of ESP_HF_CLIENT_CIND_SERVICE_AVAILABILITY_EVT

struct *esp_hf_client_cb_param_t*::*hf_client_network_roaming_param* **roaming**
HF callback param of ESP_HF_CLIENT_CIND_ROAMING_STATUS_EVT

struct *esp_hf_client_cb_param_t*::*hf_client_signal_strength_ind_param* **signal_strength**
HF callback param of ESP_HF_CLIENT_CIND_SIGNAL_STRENGTH_EVT

struct *esp_hf_client_cb_param_t*::*hf_client_battery_level_ind_param* **battery_level**
HF callback param of ESP_HF_CLIENT_CIND_BATTERY_LEVEL_EVT

struct *esp_hf_client_cb_param_t*::*hf_client_current_operator_param* **cops**
HF callback param of ESP_HF_CLIENT_COPS_CURRENT_OPERATOR_EVT

struct *esp_hf_client_cb_param_t*::*hf_client_call_ind_param* **call**
HF callback param of ESP_HF_CLIENT_CIND_CALL_EVT

struct *esp_hf_client_cb_param_t*::*hf_client_call_setup_ind_param* **call_setup**
HF callback param of ESP_HF_CLIENT_BVRA_EVT

struct *esp_hf_client_cb_param_t*::*hf_client_call_held_ind_param* **call_held**
HF callback param of ESP_HF_CLIENT_CIND_CALL_HELD_EVT

struct *esp_hf_client_cb_param_t*::*hf_client_btrh_param* **btrh**
HF callback param of ESP_HF_CLIENT_BRTH_EVT

struct *esp_hf_client_cb_param_t*::*hf_client_clip_param* **clip**
HF callback param of ESP_HF_CLIENT_CLIP_EVT

struct *esp_hf_client_cb_param_t*::*hf_client_ccwa_param* **ccwa**
HF callback param of ESP_HF_CLIENT_BVRA_EVT

struct *esp_hf_client_cb_param_t*::*hf_client_clcc_param* **clcc**
HF callback param of ESP_HF_CLIENT_CLCC_EVT

struct *esp_hf_client_cb_param_t*::*hf_client_volume_control_param* **volume_control**
HF callback param of ESP_HF_CLIENT_VOLUME_CONTROL_EVT

struct *esp_hf_client_cb_param_t*::*hf_client_at_response_param* **at_response**
HF callback param of ESP_HF_CLIENT_AT_RESPONSE_EVT

struct *esp_hf_client_cb_param_t*::*hf_client_cnum_param* **cnum**
HF callback param of ESP_HF_CLIENT_CNUM_EVT

struct *esp_hf_client_cb_param_t*::*hf_client_bsirparam* **bsir**
HF callback param of ESP_HF_CLIENT_BSIR_EVT

struct *esp_hf_client_cb_param_t*::*hf_client_binp_param* **binp**

HF callback param of ESP_HF_CLIENT_BINP_EVT

struct hf_client_at_response_param

#include <esp_hf_client_api.h> ESP_HF_CLIENT_AT_RESPONSE_EVT.

Public Members

esp_hf_at_response_code_t **code**

AT response code

esp_hf_cme_err_t **cme**

Extended Audio Gateway Error Result Code

struct hf_client_audio_stat_param

#include <esp_hf_client_api.h> ESP_HF_CLIENT_AUDIO_STATE_EVT.

Public Members

esp_hf_client_audio_state_t **state**

audio connection state

esp_bd_addr_t **remote_bda**

remote bluetooth device address

struct hf_client_battery_level_ind_param

#include <esp_hf_client_api.h> ESP_HF_CLIENT_CIND_BATTERY_LEVEL_EVT.

Public Members

int **value**

battery charge value, ranges from 0 to 5

struct hf_client_binp_param

#include <esp_hf_client_api.h> ESP_HF_CLIENT_BINP_EVT.

Public Members

const char ***number**

phone number corresponding to the last voice tag in the HF

struct hf_client_bsirparam

#include <esp_hf_client_api.h> ESP_HF_CLIENT_BSIR_EVT.

Public Members

esp_hf_client_in_band_ring_state_t **state**
setting state of in-band ring tone

struct hf_client_btrh_param
#include <esp_hf_client_api.h> ESP_HF_CLIENT_BTRH_EVT.

Public Members

esp_hf_btrh_status_t **status**
call hold and response status result code

struct hf_client_bvra_param
#include <esp_hf_client_api.h> ESP_HF_CLIENT_BVRA_EVT.

Public Members

esp_hf_vr_state_t **value**
voice recognition state

struct hf_client_call_held_ind_param
#include <esp_hf_client_api.h> ESP_HF_CLIENT_CIND_CALL_HELD_EVT.

Public Members

esp_hf_call_held_status_t **status**
bluetooth proprietary call hold status indicator

struct hf_client_call_ind_param
#include <esp_hf_client_api.h> ESP_HF_CLIENT_CIND_CALL_EVT.

Public Members

esp_hf_call_status_t **status**
call status indicator

struct hf_client_call_setup_ind_param
#include <esp_hf_client_api.h> ESP_HF_CLIENT_CIND_CALL_SETUP_EVT.

Public Members

esp_hf_call_setup_status_t **status**
call setup status indicator

struct hf_client_ccwa_param

#include <esp_hf_client_api.h> ESP_HF_CLIENT_CCWA_EVT.

Public Members

const char ***number**

phone number string of waiting call

struct hf_client_clcc_param

#include <esp_hf_client_api.h> ESP_HF_CLIENT_CLCC_EVT.

Public Members

int **idx**

numbering(starting with 1) of the call

esp_hf_current_call_direction_t **dir**

direction of the call

esp_hf_current_call_status_t **status**

status of the call

esp_hf_current_call_mpty_type_t **mpty**

multi-party flag

char ***number**

phone number(optional)

struct hf_client_clip_param

#include <esp_hf_client_api.h> ESP_HF_CLIENT_CLIP_EVT.

Public Members

const char ***number**

phone number string of call

struct hf_client_cnum_param

#include <esp_hf_client_api.h> ESP_HF_CLIENT_CNUM_EVT.

Public Members

const char ***number**

phone number string

esp_hf_subscriber_service_type_t **type**

service type that the phone number relates to

struct hf_client_conn_stat_param

#include <esp_hf_client_api.h> ESP_HF_CLIENT_CONNECTION_STATE_EVT.

Public Members

esp_hf_client_connection_state_t **state**

HF connection state

uint32_t **peer_feat**

AG supported features

uint32_t **chld_feat**

AG supported features on call hold and multiparty services

esp_bd_addr_t **remote_bda**

remote bluetooth device address

struct hf_client_current_operator_param

#include <esp_hf_client_api.h> ESP_HF_CLIENT_COPS_CURRENT_OPERATOR_EVT.

Public Members

const char ***name**

name of the network operator

struct hf_client_network_roaming_param

#include <esp_hf_client_api.h> ESP_HF_CLIENT_CIND_ROAMING_STATUS_EVT.

Public Members

esp_hf_roaming_status_t **status**

roaming status

struct hf_client_service_availability_param

#include <esp_hf_client_api.h> ESP_HF_CLIENT_CIND_SERVICE_AVAILABILITY_EVT.

Public Members

esp_hf_network_state_t **status**

service availability status

struct hf_client_signal_strength_ind_param

#include <esp_hf_client_api.h> ESP_HF_CLIENT_CIND_SIGNAL_STRENGTH_EVT.

Public Members

int **value**

signal strength value, ranges from 0 to 5

struct hf_client_volume_control_param

#include <esp_hf_client_api.h> ESP_HF_CLIENT_VOLUME_CONTROL_EVT.

Public Members

esp_hf_volume_control_target_t **type**

volume control target, speaker or microphone

int **volume**

gain, ranges from 0 to 15

Macros

ESP_BT_HF_CLIENT_NUMBER_LEN

ESP_BT_HF_CLIENT_OPERATOR_NAME_LEN

ESP_HF_CLIENT_PEER_FEAT_3WAY

ESP_HF_CLIENT_PEER_FEAT_ECNR

ESP_HF_CLIENT_PEER_FEAT_VREC

ESP_HF_CLIENT_PEER_FEAT_INBAND

ESP_HF_CLIENT_PEER_FEAT_VTAG

ESP_HF_CLIENT_PEER_FEAT_REJECT

ESP_HF_CLIENT_PEER_FEAT_ECS

ESP_HF_CLIENT_PEER_FEAT_ECC

ESP_HF_CLIENT_PEER_FEAT_EXTERR

ESP_HF_CLIENT_PEER_FEAT_CODEC

ESP_HF_CLIENT_CHLD_FEAT_REL

ESP_HF_CLIENT_CHLD_FEAT_REL_ACC

ESP_HF_CLIENT_CHLD_FEAT_REL_X

ESP_HF_CLIENT_CHLD_FEAT_HOLD_ACC

ESP_HF_CLIENT_CHLD_FEAT_PRIV_X

ESP_HF_CLIENT_CHLD_FEAT_MERGE

ESP_HF_CLIENT_CHLD_FEAT_MERGE_DETACH

Type Definitions

typedef void (*esp_hf_client_incoming_data_cb_t)(const uint8_t *buf, uint32_t len)

HFP client incoming data callback function, the callback is useful in case of Voice Over HCI.

Parameters

- buf: : pointer to incoming data(payload of HCI synchronous data packet), the buffer is allocated inside bluetooth protocol stack and will be released after invoke of the callback is finished.
- len: : size(in bytes) in buf

typedef uint32_t (*esp_hf_client_outgoing_data_cb_t)(uint8_t *buf, uint32_t len)

HFP client outgoing data callback function, the callback is useful in case of Voice Over HCI. Once audio connection is set up and the application layer has prepared data to send, the lower layer will call this function to read data and then send. This callback is supposed to be implemented as non-blocking, and if data is not enough, return value 0 is supposed.

Parameters

- buf: : pointer to incoming data(payload of HCI synchronous data packet), the buffer is allocated inside bluetooth protocol stack and will be released after invoke of the callback is finished.
- len: : size(in bytes) in buf
- length: of data successfully read

typedef void (*esp_hf_client_cb_t)(*esp_hf_client_cb_event_t* event, *esp_hf_client_cb_param_t* *param)

HFP client callback function type.

Parameters

- event: : Event type
- param: : Pointer to callback parameter

Enumerations

enum esp_hf_client_connection_state_t

Bluetooth HFP RFCOMM connection and service level connection status.

Values:

ESP_HF_CLIENT_CONNECTION_STATE_DISCONNECTED = 0
RFCOMM data link channel released

ESP_HF_CLIENT_CONNECTION_STATE_CONNECTING
connecting remote device on the RFCOMM data link

ESP_HF_CLIENT_CONNECTION_STATE_CONNECTED
RFCOMM connection established

ESP_HF_CLIENT_CONNECTION_STATE_SLC_CONNECTED
service level connection established

ESP_HF_CLIENT_CONNECTION_STATE_DISCONNECTING
disconnecting with remote device on the RFCOMM dat link

enum esp_hf_client_audio_state_t
Bluetooth HFP audio connection status.

Values:

ESP_HF_CLIENT_AUDIO_STATE_DISCONNECTED = 0
audio connection released

ESP_HF_CLIENT_AUDIO_STATE_CONNECTING
audio connection has been initiated

ESP_HF_CLIENT_AUDIO_STATE_CONNECTED
audio connection is established

ESP_HF_CLIENT_AUDIO_STATE_CONNECTED_MSBC
mSBC audio connection is established

enum esp_hf_client_in_band_ring_state_t
in-band ring tone state

Values:

ESP_HF_CLIENT_IN_BAND_RINGTONE_NOT_PROVIDED = 0

ESP_HF_CLIENT_IN_BAND_RINGTONE_PROVIDED

enum esp_hf_client_cb_event_t
HF CLIENT callback events.

Values:

ESP_HF_CLIENT_CONNECTION_STATE_EVT = 0
connection state changed event

ESP_HF_CLIENT_AUDIO_STATE_EVT
audio connection state change event

ESP_HF_CLIENT_BVRA_EVT
: voice recognition state change event

ESP_HF_CLIENT_CIND_CALL_EVT
: call indication

ESP_HF_CLIENT_CIND_CALL_SETUP_EVT
: call setup indication

ESP_HF_CLIENT_CIND_CALL_HELD_EVT
: call held indication

ESP_HF_CLIENT_CIND_SERVICE_AVAILABILITY_EVT
: network service availability indication

ESP_HF_CLIENT_CIND_SIGNAL_STRENGTH_EVT
: signal strength indication

ESP_HF_CLIENT_CIND_ROAMING_STATUS_EVT
: roaming status indication

ESP_HF_CLIENT_CIND_BATTERY_LEVEL_EVT
: battery level indication

ESP_HF_CLIENT_COPS_CURRENT_OPERATOR_EVT
: current operator information

ESP_HF_CLIENT_BTRH_EVT
: call response and hold event

ESP_HF_CLIENT_CLIP_EVT
: Calling Line Identification notification

ESP_HF_CLIENT_CCWA_EVT
: call waiting notification

ESP_HF_CLIENT_CLCC_EVT
: list of current calls notification

ESP_HF_CLIENT_VOLUME_CONTROL_EVT
: audio volume control command from AG, provided by +VGM or +VGS message

ESP_HF_CLIENT_AT_RESPONSE_EVT
: AT command response event

ESP_HF_CLIENT_CNUM_EVT
: subscriber information response from AG

ESP_HF_CLIENT_BSIR_EVT
: setting of in-band ring tone

`ESP_HF_CLIENT_BINP_EVT`
requested number of last voice tag from AG

`ESP_HF_CLIENT_RING_IND_EVT`
ring indication event

HFP AG API

Overview

Instructions

API Reference

Header File

- bt/host/bluedroid/api/include/api/esp_hf_ag_api.h

Functions

esp_err_t **esp_bt_hf_register_callback**(*esp_hf_cb_t callback*)
Register application callback function to HFP AG module. This function should be called only after esp_bluedroid_enable() completes successfully, used by HFP AG.

Return

- ESP_OK: success
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: if callback is a NULL function pointer

Parameters

- `callback`: HFP AG event callback function

esp_err_t **esp_bt_hf_init**(*esp_bd_addr_t remote_addr*)
Initialize the bluetooth HF AG module. This function should be called after esp_bluedroid_enable() completes successfully.

Return

- ESP_OK: if the initialization request is sent successfully
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

Parameters

- `remote_addr`: remote bluetooth device address

esp_err_t **esp_bt_hf_deinit**(*esp_bd_addr_t remote_addr*)

De-initialize for HF AG module. This function should be called only after esp_bluedroid_enable() completes successfully.

Return

- ESP_OK: success
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

Parameters

- `remote_addr`: remote bluetooth device address

esp_err_t **esp_bt_hf_connect**(*esp_bd_addr_t remote_bda*)

Connect to remote bluetooth HFP client device, must after esp_bt_hf_init()

Return

- ESP_OK: connect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

Parameters

- `remote_bda`: remote bluetooth HFP client device address

esp_err_t **esp_bt_hf_disconnect**(*esp_bd_addr_t remote_bda*)

Disconnect from the remote HFP client.

Return

- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

Parameters

- `remote_bda`: remote bluetooth device address

esp_err_t **esp_bt_hf_connect_audio**(*esp_bd_addr_t remote_bda*)

Create audio connection with remote HFP client. As a precondition to use this API, Service Level Connection shall exist between HF client and AG.

Return

- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

Parameters

- `remote_bda`: remote bluetooth device address

esp_err_t **esp_bt_hf_disconnect_audio**(*esp_bd_addr_t remote_bda*)

Release the established audio connection with remote HFP client.

Return

- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

Parameters

- `remote_bda`: remote bluetooth device address

esp_err_t **esp_bt_hf_vra**(*esp_bd_addr_t remote_bda*, *esp_hf_vr_state_t value*)

Response of Volume Recognition Command(AT+VRA) from HFP client. As a precondition to use this API, Service Level Connection shall exist with HFP client.

Return

- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

Parameters

- `remote_bda`: volume control target, speaker or microphone
- `value`: gain of the speaker of microphone, ranges 0 to 15

esp_err_t **esp_bt_hf_volume_control**(*esp_bd_addr_t remote_bda*, *esp_hf_volume_control_target_t type*, int *volume*)

Volume synchronization with HFP client. As a precondition to use this API, Service Level Connection shall exist with HFP client.

Return

- ESP_OK: disconnect request is sent to lower layer

- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

Parameters

- `remote_bda`: remote bluetooth device address
- `type`: volume control target, speaker or microphone
- `volume`: gain of the speaker of microphone, ranges 0 to 15

esp_err_t **esp_hf_unat_response**(*esp_bd_addr_t remote_addr*, char **unat*)

Handle Unknown AT command from HFP Client. As a precondition to use this API, Service Level Connection shall exist between AG and HF Client.

Return

- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

Parameters

- `remote_addr`: remote bluetooth device address
- `unat`: User AT command response to HF Client. It will response "ERROR" by default if unat is NULL.

esp_err_t **esp_bt_hf_cmee_response**(*esp_bd_addr_t remote_bda*, *esp_hf_at_response_code_t response_code*, *esp_hf_cme_err_t error_code*)

Unsolicited send extend AT error code to HFP Client. As a precondition to use this API, Service Level Connection shall exist between AG and HF Client.

Return

- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

Parameters

- `remote_bda`: remote bluetooth device address
- `response_code`: AT command response code
- `error_code`: CME error code

esp_err_t **esp_bt_hf_indchange_notification**(*esp_bd_addr_t* *remote_addr*, *esp_hf_call_status_t* *call_state*, *esp_hf_call_setup_status_t* *call_setup_state*, *esp_hf_network_state_t* *ntk_state*, int *signal*)

Usolicited send device status notificationto HFP Client. As a precondition to use this API, Service Level Connection shall exist between AG and HF Client.

Return

- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

Parameters

- `remote_addr`: remote bluetooth device address
- `call_state`: call state
- `call_setup_state`: call setup state
- `ntk_state`: network service state
- `signal`: signal strength from 0 to 5

esp_err_t **esp_bt_hf_cind_response**(*esp_bd_addr_t* *remote_addr*, *esp_hf_call_status_t* *call_state*, *esp_hf_call_setup_status_t* *call_setup_state*, *esp_hf_network_state_t* *ntk_state*, int *signal*, *esp_hf_roaming_status_t* *roam*, int *batt_lev*, *esp_hf_call_held_status_t* *call_held_status*)

Response to device individual indicatiors to HFP Client. As a precondition to use this API, Service Level Connection shall exist between AG and HF Client.

Return

- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

Parameters

- `remote_addr`: remote bluetooth device address
- `call_state`: call state
- `call_setup_state`: call setup state
- `ntk_state`: network service state
- `signal`: signal strength from 0 to 5

- `roam`: roam state
- `batt_lev`: batery level from 0 to 5
- `call_held_status`: call held status

esp_err_t **esp_bt_hf_cops_response**(*esp_bd_addr_t remote_addr*, char **name*)

Reponse for AT+COPS command from HF Client. As a precondition to use this API, Service Level Connection shall exist with HFP Client.

Return

- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

Parameters

- `remote_addr`: remote bluetooth device address
- `name`: current operator name

esp_err_t **esp_bt_hf_clcc_response**(*esp_bd_addr_t remote_addr*, int *index*, *esp_hf_current_call_direction_t dir*, *esp_hf_current_call_status_t current_call_state*, *esp_hf_current_call_mode_t mode*, *esp_hf_current_call_mpty_type_t mpty*, char **number*, *esp_hf_call_addr_type_t type*)

Response to AT+CLCC command from HFP Client. As a precondition to use this API, Service Level Connection shall exist between AG and HF Client.

Return

- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

Parameters

- `remote_addr`: remote bluetooth device address
- `index`: the index of current call
- `dir`: call direction (incoming/outgoing)
- `current_call_state`: current call state
- `mode`: current call mode (voice/data/fax)
- `mpty`: single or multi type

- `number`: current call number
- `type`: international type or unknow

esp_err_t **esp_bt_hf_cnum_response**(*esp_bd_addr_t remote_addr*, char **number*, *esp_hf_subscriber_service_type_t type*)

Response for AT+CNUM command from HF Client. As a precondition to use this API, Service Level Connection shall exist with AG.

Return

- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

Parameters

- `remote_addr`: remote bluetooth device address
- `number`: registration number
- `type`: service type (unknown/voice/fax)

esp_err_t **esp_bt_hf_bsir**(*esp_bd_addr_t remote_addr*, *esp_hf_in_band_ring_state_t state*)

Inform HF Client that AG Provided in-band ring tone or not. As a precondition to use this API, Service Level Connection shall exist with AG.

Return

- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

Parameters

- `remote_addr`: remote bluetooth device address
- `state`: in-band ring tone state

esp_err_t **esp_bt_hf_answer_call**(*esp_bd_addr_t remote_addr*, int *num_active*, int *num_held*, *esp_hf_call_status_t call_state*, *esp_hf_call_setup_status_t call_setup_state*, char **number*, *esp_hf_call_addr_type_t call_addr_type*)

Answer Incoming Call from AG. As a precondition to use this API, Service Level Connection shall exist with AG.

Return

- ESP_OK: disconnect request is sent to lower layer

- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

Parameters

- `remote_addr`: remote bluetooth device address
- `num_active`: the number of active call
- `num_held`: the number of held call
- `call_state`: call state
- `call_setup_state`: call setup state
- `number`: number of the incoming call
- `call_addr_type`: call address type

esp_err_t **esp_bt_hf_reject_call**(*esp_bd_addr_t remote_addr*, int *num_active*, int *num_held*, *esp_hf_call_status_t call_state*, *esp_hf_call_setup_status_t call_setup_state*, char **number*, *esp_hf_call_addr_type_t call_addr_type*)

Reject Incoming Call from AG. As a precondition to use this API, Service Level Connection shall exist with AG.

Return

- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

Parameters

- `remote_addr`: remote bluetooth device address
- `num_active`: the number of active call
- `num_held`: the number of held call
- `call_state`: call state
- `call_setup_state`: call setup state
- `number`: number of the incoming call
- `call_addr_type`: call address type

esp_err_t **esp_bt_hf_out_call**(*esp_bd_addr_t remote_addr*, int *num_active*, int *num_held*, *esp_hf_call_status_t call_state*, *esp_hf_call_setup_status_t call_setup_state*, char **number*, *esp_hf_call_addr_type_t call_addr_type*)

Reject incoming call from AG. As a precondition to use this API, Service Level Connection shall exist with AG.

Return

- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

Parameters

- `remote_addr`: remote bluetooth device address
- `num_active`: the number of active call
- `num_held`: the number of held call
- `call_state`: call state
- `call_setup_state`: call setup state
- `number`: number of the outgoing call
- `call_addr_type`: call address type

esp_err_t **esp_bt_hf_end_call**(*esp_bd_addr_t remote_addr*, int *num_active*, int *num_held*, *esp_hf_call_status_t call_state*, *esp_hf_call_setup_status_t call_setup_state*, char **number*, *esp_hf_call_addr_type_t call_addr_type*)

End an ongoing call. As a precondition to use this API, Service Level Connection shall exist with AG.

Return

- ESP_OK: disconnect request is sent to lower layer
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

Parameters

- `remote_addr`: remote bluetooth device address
- `num_active`: the number of active call
- `num_held`: the number of held call
- `call_state`: call state
- `call_setup_state`: call setup state
- `number`: number of the call
- `call_addr_type`: call address type

esp_err_t **esp_bt_hf_register_data_callback**(*esp_hf_incoming_data_cb_t recv*, *esp_hf_outgoing_data_cb_t send*)

Register AG data output function; the callback is only used in the case that Voice Over HCI is enabled.

Return

- ESP_OK: success
- ESP_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: if callback is a NULL function pointer

Parameters

- `recv`: HFP client incoming data callback function
- `send`: HFP client outgoing data callback function

void **esp_hf_outgoing_data_ready**(void)

Trigger the lower-layer to fetch and send audio data. This function is only only used in the case that Voice Over HCI is enabled. Precondition is that the HFP audio connection is connected. After this function is called, lower layer will invoke esp_hf_client_outgoing_data_cb_t to fetch data.

Unions

union esp_hf_cb_param_t

#include <esp_hf_ag_api.h> HFP AG callback parameters.

Public Members

struct *esp_hf_cb_param_t*::*hf_conn_stat_param* **conn_stat**

AG callback param of ESP_HF_CONNECTION_STATE_EVT

struct *esp_hf_cb_param_t*::*hf_audio_stat_param* **audio_stat**

AG callback param of ESP_HF_AUDIO_STATE_EVT

struct *esp_hf_cb_param_t*::*hf_vra_rep_param* **vra_rep**

AG callback param of ESP_HF_BVRA_RESPONSE_EVT

struct *esp_hf_cb_param_t*::*hf_volume_control_param* **volume_control**

AG callback param of ESP_HF_VOLUME_CONTROL_EVT

struct *esp_hf_cb_param_t*::*hf_unat_rep_param* **unat_rep**

AG callback param of ESP_HF_UNAT_RESPONSE_EVT

struct *esp_hf_cb_param_t*::*hf_cind_param* **cind**

AG callback param of ESP_HF_CIND_RESPONSE_EVT

struct *esp_hf_cb_param_t*::*hf_out_call_param* **out_call**

AG callback param of ESP_HF_DIAL_EVT

struct *esp_hf_cb_param_t*::*hf_vts_rep_param* **vts_rep**

AG callback param of ESP_HF_VTS_RESPONSE_EVT

struct *esp_hf_cb_param_t*::*hf_nrec_param* **nrec**

AG callback param of ESP_HF_NREC_RESPONSE_EVT

struct *esp_hf_cb_param_t*::*hf_wbs_rep_param* **wbs_rep**

AG callback param of ESP_HF_WBS_RESPONSE_EVT

struct *esp_hf_cb_param_t*::*hf_bcs_rep_param* **bcs_rep**

AG callback param of ESP_HF_BCS_RESPONSE_EVT

struct hf_audio_stat_param

#include <esp_hf_ag_api.h> ESP_HF_AUDIO_STATE_EVT.

Public Members

esp_bd_addr_t **remote_addr**

remote bluetooth device address

esp_hf_audio_state_t **state**

audio connection state

struct hf_bcs_rep_param

#include <esp_hf_ag_api.h> ESP_HF_BCS_RESPONSE_EVT.

Public Members

esp_hf_wbs_config_t **mode**

codec mode CVSD or mSBC

struct hf_cind_param

#include <esp_hf_ag_api.h> ESP_HF_CIND_RESPONSE_EVT.

Public Members

esp_hf_call_status_t **call_status**

call status indicator

esp_hf_call_setup_status_t **call_setup_status**

call setup status indicator

esp_hf_network_state_t **svc**

bluetooth proprietary call hold status indicator

int **signal_strength**

bluetooth proprietary call hold status indicator

esp_hf_roaming_status_t **roam**

bluetooth proprietary call hold status indicator

int **battery_level**

battery charge value, ranges from 0 to 5

esp_hf_call_held_status_t **call_held_status**

bluetooth proprietary call hold status indicator

struct hf_conn_stat_param

#include <esp_hf_ag_api.h> ESP_HS_CONNECTION_STATE_EVT.

Public Members

esp_bd_addr_t **remote_bda**

remote bluetooth device address

esp_hf_connection_state_t **state**

Connection state

uint32_t **peer_feat**

HF supported features

uint32_t **chld_feat**

AG supported features on call hold and multiparty services

struct hf_nrec_param

#include <esp_hf_ag_api.h> ESP_HF_NREC_RESPOSNE_EVT.

Public Members

esp_hf_nrec_t **state**

NREC enabled or disabled

struct hf_out_call_param

#include <esp_hf_ag_api.h> ESP_HF_DIAL_EVT.

Public Members

esp_bd_addr_t **remote_addr**

remote bluetooth device address

char ***num_or_loc**

location in phone memory

struct hf_unat_rep_param

#include <esp_hf_ag_api.h> ESP_HF_UNAT_RESPOSNE_EVT.

Public Members

char ***unat**

unknown AT command string

struct hf_volume_control_param

#include <esp_hf_ag_api.h> ESP_HF_VOLUME_CONTROL_EVT.

Public Members

esp_hf_volume_type_t **type**

volume control target, speaker or microphone

int **volume**

gain, ranges from 0 to 15

struct hf_vra_rep_param

#include <esp_hf_ag_api.h> ESP_HF_BVRA_RESPONSE_EVT.

Public Members

esp_bd_addr_t **remote_addr**

remote bluetooth device address

esp_hf_vr_state_t **value**

voice recognition state

struct hf_vts_rep_param

#include <esp_hf_ag_api.h> ESP_HF_VTS_RESPOSNE_EVT.

Public Members

char ***code**

MTF code from HF Client

struct hf_wbs_rep_param

#include <esp_hf_ag_api.h> ESP_HF_WBS_RESPONSE_EVT.

Public Members

esp_hf_wbs_config_t **codec**

codec mode CVSD or mSBC

Macros

ESP_HF_PEER_FEAT_3WAY

ESP_HF_PEER_FEAT_ECNR

ESP_HF_PEER_FEAT_VREC

ESP_HF_PEER_FEAT_INBAND

ESP_HF_PEER_FEAT_VTAG

ESP_HF_PEER_FEAT_REJECT

ESP_HF_PEER_FEAT_ECS

ESP_HF_PEER_FEAT_ECC

ESP_HF_PEER_FEAT_EXTERR

ESP_HF_PEER_FEAT_CODEC

ESP_HF_CHLD_FEAT_REL

ESP_HF_CHLD_FEAT_REL_ACC

ESP_HF_CHLD_FEAT_REL_X

ESP_HF_CHLD_FEAT_HOLD_ACC

ESP_HF_CHLD_FEAT_PRIV_X

ESP_HF_CHLD_FEAT_MERGE

ESP_HF_CHLD_FEAT_MERGE_DETACH

Type Definitions

typedef void (***esp_hf_incoming_data_cb_t**)(**const** uint8_t *buf, uint32_t len)

AG incoming data callback function, the callback is useful in case of Voice Over HCI.

Parameters

- **buf**: : pointer to incoming data(payload of HCI synchronous data packet), the buffer is allocated inside bluetooth protocol stack and will be released after invoke of the callback is finished.
- **len**: : size(in bytes) in buf

typedef uint32_t (***esp_hf_outgoing_data_cb_t**)(uint8_t *buf, uint32_t len)

AG outgoing data callback function, the callback is useful in case of Voice Over HCI. Once audio connection is set up and the application layer has prepared data to send, the lower layer will call this

function to read data and then send. This callback is supposed to be implemented as non-blocking, and if data is not enough, return value 0 is supposed.

Parameters

- `buf`: : pointer to incoming data(payload of HCI synchronous data packet), the buffer is allocated inside bluetooth protocol stack and will be released after invoke of the callback is finished.
- `len`: : size(in bytes) in buf
- `length`: of data successfully read

`typedef` void (*`esp_hf_cb_t`)(*esp_hf_cb_event_t* event, *esp_hf_cb_param_t* *param)

HF AG callback function type.

Parameters

- `event`: : Event type
- `param`: : Pointer to callback parameter

Enumerations

`enum esp_hf_cb_event_t`

HF callback events.

Values:

`ESP_HF_CONNECTION_STATE_EVT` = 0

Connection state changed event

`ESP_HF_AUDIO_STATE_EVT`

Audio connection state change event

`ESP_HF_BVRA_RESPONSE_EVT`

Voice recognition state change event

`ESP_HF_VOLUME_CONTROL_EVT`

Audio volume control command from HF Client, provided by +VGM or +VGS message

`ESP_HF_UNAT_RESPONSE_EVT`

Unknown AT cmd Response

`ESP_HF_IND_UPDATE_EVT`

Indicator Update Event

`ESP_HF_CIND_RESPONSE_EVT`

Call And Device Indicator Response

`ESP_HF_COPS_RESPONSE_EVT`
Current operator information

`ESP_HF_CLCC_RESPONSE_EVT`
List of current calls notification

`ESP_HF_CNUM_RESPONSE_EVT`
Subscriber information response from HF Client

`ESP_HF_VTS_RESPONSE_EVT`
Enable or not DTMF

`ESP_HF_NREC_RESPONSE_EVT`
Enable or not NREC

`ESP_HF_ATA_RESPONSE_EVT`
Answer an Incoming Call

`ESP_HF_CHUP_RESPONSE_EVT`
Reject an Incoming Call

`ESP_HF_DIAL_EVT`
Origin an outgoing call with specific number or the dial the last number

`ESP_HF_WBS_RESPONSE_EVT`
Codec Status

`ESP_HF_BCS_RESPONSE_EVT`
Final Codec Choice

2.1.5 NimBLE-based host APIs

Overview

Apache MyNewt NimBLE is a highly configurable and BT SIG qualifiable BLE stack providing both host and controller functionalities. ESP-IDF supports NimBLE host stack which is specifically ported for ESP32 platform and FreeRTOS. The underlying controller is still the same (as in case of Bluedroid) providing VHCI interface. Refer to NimBLE user guide for a complete list of features and additional information on NimBLE stack. Most features of NimBLE including BLE Mesh are supported by ESP-IDF. The porting layer is kept cleaner by maintaining all the existing APIs of NimBLE along with a single ESP-NimBLE API for initialization, making it simpler for the application developers.

Architecture

Currently, NimBLE host and controller support different transports such as UART and RAM between them. However, RAM transport cannot be used as is in case of ESP as ESP controller supports VHCI interface and buffering schemes used by NimBLE host is incompatible with that used by ESP controller. Therefore,

a new transport between NimBLE host and ESP controller has been added. This is depicted in the figure below. This layer is responsible for maintaining pool of transport buffers and formatting buffers exchanges between host and controller as per the requirements.

图 1: ESP NimBLE Stack

Threading Model

The NimBLE host can run inside the application thread or can have its own independent thread. This flexibility is inherently provided by NimBLE design. By default, a thread is spawned by the porting function `nimble_port_freertos_init`. This behavior can be changed by overriding the same function. For BLE Mesh, additional thread (advertising thread) is used which keeps on feeding advertisement events to the main thread.

Programming Sequence

To begin with, make sure that the NimBLE stack is enabled from menuconfig *choose NimBLE for the Bluetooth host*.

Typical programming sequence with NimBLE stack consists of the following steps:

- Initialize NVS flash using *`nvs_flash_init()`* API. This is because ESP controller uses NVS during initialization.
- Call *`esp_nimble_hci_and_controller_init()`* to initialize ESP controller as well as transport layer. This will also link the host and controller modules together. Alternatively, if ESP controller is already initialized, then *`esp_nimble_hci_init()`* can be called for the remaining initialization.
- Initialize the host stack using `nimble_port_init`.
- Initialize the required NimBLE host configuration parameters and callbacks
- Perform application specific tasks/initialization
- Run the thread for host stack using `nimble_port_freertos_init`

This documentation does not cover NimBLE APIs. Refer to NimBLE tutorial for more details on the programming sequence/NimBLE APIs for different scenarios.

API Reference

Header File

- bt/host/nimble/esp-hci/include/esp_nimble_hci.h

Functions

esp_err_t **esp_nimble_hci_init**(void)

Initialize VHCI transport layer between NimBLE Host and ESP Bluetooth controller.

This function initializes the transport buffers to be exchanged between NimBLE host and ESP controller. It also registers required host callbacks with the controller.

Return

- ESP_OK if the initialization is successful
- Appropriate error code from esp_err_t in case of an error

esp_err_t **esp_nimble_hci_and_controller_init**(void)

Initialize ESP Bluetooth controller(link layer) and VHCI transport layer between NimBLE Host and ESP Bluetooth controller.

This function initializes ESP controller in BLE only mode and the transport buffers to be exchanged between NimBLE host and ESP controller. It also registers required host callbacks with the controller.

Below is the sequence of APIs to be called to init/enable NimBLE host and ESP controller:

```
void ble_host_task(void *param)
{
     nimble_port_run(); //This function will return only when nimble_port_stop() is␣
↪executed.
     nimble_port_freertos_deinit();
}

int ret = esp_nimble_hci_and_controller_init();
if (ret != ESP_OK) {
     ESP_LOGE(TAG, "esp_nimble_hci_and_controller_init() failed with error: %d",␣
↪ret);
     return;
}

nimble_port_init();
```

(下页继续)

(续上页)

```
//Initialize the NimBLE Host configuration

nimble_port_freertos_init(ble_host_task);
```

nimble_port_freertos_init() is an optional call that creates a new task in which the NimBLE host will run. The task function should have a call to nimble_port_run(). If a separate task is not required, calling nimble_port_run() will run the NimBLE host in the current task.

Return

- ESP_OK if the initialization is successful
- Appropriate error code from esp_err_t in case of an error

esp_err_t **esp_nimble_hci_deinit**(void)

Deinitialize VHCI transport layer between NimBLE Host and ESP Bluetooth controller.

Note This function should be called after the NimBLE host is deinitialized.

Return

- ESP_OK if the deinitialization is successful
- Appropriate error codes from esp_err_t in case of an error

esp_err_t **esp_nimble_hci_and_controller_deinit**(void)

Deinitialize VHCI transport layer between NimBLE Host and ESP Bluetooth controller and disable and deinitialize the controller.

Below is the sequence of APIs to be called to disable/deinit NimBLE host and ESP controller:

Note This function should not be executed in the context of Bluetooth host task.

Note This function should be called after the NimBLE host is deinitialized.

```
int ret = nimble_port_stop();
if (ret == 0) {
     nimble_port_deinit();

     ret = esp_nimble_hci_and_controller_deinit();
     if (ret != ESP_OK) {
         ESP_LOGE(TAG, "esp_nimble_hci_and_controller_deinit() failed with error: %d
↪", ret);
     }
}
```

If nimble_port_freertos_init() is used during initialization, then nimble_port_freertos_deinit() should be called in the host task after nimble_port_run().

Return

- ESP_OK if the deinitialization is successful
- Appropriate error codes from esp_err_t in case of an error

Macros

`BLE_HCI_UART_H4_NONE`

`BLE_HCI_UART_H4_CMD`

`BLE_HCI_UART_H4_ACL`

`BLE_HCI_UART_H4_SCO`

`BLE_HCI_UART_H4_EVT`

2.1.6 ESP-BLE-MESH

With various features of ESP-BLE-MESH, users can create a managed flooding mesh network for several scenarios, such as lighting, sensor and etc.

For an ESP32 to join and work on a ESP-BLE-MESH network, it must be provisioned firstly. By provisioning, the ESP32, as an unprovisioned device, will join the ESP-BLE-MESH network and become a ESP-BLE-MESH node, communicating with other nodes within or beyond the radio range.

Apart from ESP-BLE-MESH nodes, inside ESP-BLE-MESH network, there is also ESP32 that works as ESP-BLE-MESH Provisioner, which could provision unprovisioned devices into ESP-BLE-MESH nodes and configure the nodes with various features.

For information how to start using ESP32 and ESP-BLE-MESH, please see the Section *ESP-BLE-MESH 快速入门*. If you are interested in information on ESP-BLE-MESH architecture, including some details of software implementation, please see Section *ESP-BLE-MESH 架构*.

Application Examples and Demos

Please refer to Sections *ESP-BLE-MESH 示例* and *ESP-BLE-MESH 演示视频*.

API Reference

ESP-BLE-MESH APIs are divided into the following parts:

- *ESP-BLE-MESH Definitions*

- *ESP-BLE-MESH Core API Reference*
- *ESP-BLE-MESH Models API Reference*

ESP-BLE-MESH Definitions

This section contains only one header file, which lists the following items of ESP-BLE-MESH.

- ID of all the models and related message opcodes
- Structs of model, element and Composition Data
- Structs of used by ESP-BLE-MESH Node/Provisioner for provisioning
- Structs used to transmit/receive messages
- Event types and related event parameters

Header File

- bt/esp_ble_mesh/api/esp_ble_mesh_defs.h

Unions

union esp_ble_mesh_prov_cb_param_t

#include <esp_ble_mesh_defs.h> BLE Mesh Node/Provisioner callback parameters union.

Public Members

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_prov_register_comp_param* **prov_register_comp**

Event parameter of ESP_BLE_MESH_PROV_REGISTER_COMP_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_set_unprov_dev_name_comp_param* **node_set_unprov_dev_n**

Event parameter of ESP_BLE_MESH_NODE_SET_UNPROV_DEV_NAME_COMP_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_prov_enable_comp_param* **node_prov_enable_comp**

Event parameter of ESP_BLE_MESH_NODE_PROV_ENABLE_COMP_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_prov_disable_comp_param* **node_prov_disable_comp**

Event parameter of ESP_BLE_MESH_NODE_PROV_DISABLE_COMP_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_link_open_evt_param* **node_prov_link_open**

Event parameter of ESP_BLE_MESH_NODE_PROV_LINK_OPEN_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_link_close_evt_param* **node_prov_link_close**

Event parameter of ESP_BLE_MESH_NODE_PROV_LINK_CLOSE_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_output_num_evt_param* **node_prov_output_num**
Event parameter of ESP_BLE_MESH_NODE_PROV_OUTPUT_NUMBER_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_output_str_evt_param* **node_prov_output_str**
Event parameter of ESP_BLE_MESH_NODE_PROV_OUTPUT_STRING_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_input_evt_param* **node_prov_input**
Event parameter of ESP_BLE_MESH_NODE_PROV_INPUT_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_provision_complete_evt_param* **node_prov_complete**
Event parameter of ESP_BLE_MESH_NODE_PROV_COMPLETE_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_provision_reset_param* **node_prov_reset**
Event parameter of ESP_BLE_MESH_NODE_PROV_RESET_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_set_oob_pub_key_comp_param* **node_prov_set_oob_pub_key**
Event parameter of ESP_BLE_MESH_NODE_PROV_SET_OOB_PUB_KEY_COMP_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_input_number_comp_param* **node_prov_input_num_comp**
Event parameter of ESP_BLE_MESH_NODE_PROV_INPUT_NUM_COMP_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_input_string_comp_param* **node_prov_input_str_comp**
Event parameter of ESP_BLE_MESH_NODE_PROV_INPUT_STR_COMP_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_proxy_identity_enable_comp_param* **node_proxy_identity_en**
Event parameter of ESP_BLE_MESH_NODE_PROXY_IDENTITY_ENABLE_COMP_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_proxy_gatt_enable_comp_param* **node_proxy_gatt_enable_co**
Event parameter of ESP_BLE_MESH_NODE_PROXY_GATT_ENABLE_COMP_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_proxy_gatt_disable_comp_param* **node_proxy_gatt_disable_**
Event parameter of ESP_BLE_MESH_NODE_PROXY_GATT_DISABLE_COMP_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_provisioner_recv_unprov_adv_pkt_param* **provisioner_recv_**
Event parameter of ESP_BLE_MESH_PROVISIONER_RECV_UNPROV_ADV_PKT_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_provisioner_prov_enable_comp_param* **provisioner_prov_ena**
Event parameter of ESP_BLE_MESH_PROVISIONER_PROV_ENABLE_COMP_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_provisioner_prov_disable_comp_param* **provisioner_prov_di**
Event parameter of ESP_BLE_MESH_PROVISIONER_PROV_DISABLE_COMP_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_provisioner_link_open_evt_param* **provisioner_prov_link_o**
Event parameter of ESP_BLE_MESH_PROVISIONER_PROV_LINK_OPEN_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_provisioner_prov_read_oob_pub_key_evt_param* **provisioner**
Event parameter of ESP_BLE_MESH_PROVISIONER_PROV_READ_OOB_PUB_KEY_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_provisioner_prov_input_evt_param* **provisioner_prov_input**
Event parameter of ESP_BLE_MESH_PROVISIONER_PROV_INPUT_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_provisioner_prov_output_evt_param* **provisioner_prov_outp**

Event parameter of ESP_BLE_MESH_PROVISIONER_PROV_OUTPUT_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_provisioner_link_close_evt_param* **provisioner_prov_link_c**

Event parameter of ESP_BLE_MESH_PROVISIONER_PROV_LINK_CLOSE_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_provisioner_prov_comp_param* **provisioner_prov_complete**

Event parameter of ESP_BLE_MESH_PROVISIONER_PROV_COMPLETE_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_provisioner_add_unprov_dev_comp_param* **provisioner_add**

Event parameter of ESP_BLE_MESH_PROVISIONER_ADD_UNPROV_DEV_COMP_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_provisioner_prov_dev_with_addr_comp_param* **provisioner_**

Event parameter of ESP_BLE_MESH_PROVISIONER_PROV_DEV_WITH_ADDR_COMP_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_provisioner_delete_dev_comp_param* **provisioner_delete_de**

Event parameter of ESP_BLE_MESH_PROVISIONER_DELETE_DEV_COMP_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_provisioner_set_dev_uuid_match_comp_param* **provisioner_**

Event parameter of ESP_BLE_MESH_PROVISIONER_SET_DEV_UUID_MATCH_COMP_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_provisioner_set_prov_data_info_comp_param* **provisioner_s**

Event parameter of ESP_BLE_MESH_PROVISIONER_SET_PROV_DATA_INFO_COMP_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_provisioner_set_static_oob_val_comp_param* **provisioner_se**

Event parameter of ESP_BLE_MESH_PROVISIONER_SET_STATIC_OOB_VALUE_COMP_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_provisioner_set_primary_elem_addr_comp_param* **provision**

Event parameter of ESP_BLE_MESH_PROVISIONER_SET_PRIMARY_ELEM_ADDR_COMP_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_provisioner_prov_read_oob_pub_key_comp_param* **provisio**

Event parameter of ESP_BLE_MESH_PROVISIONER_PROV_READ_OOB_PUB_KEY_COMP_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_provisioner_prov_input_num_comp_param* **provisioner_pro**

Event parameter of ESP_BLE_MESH_PROVISIONER_PROV_INPUT_NUMBER_COMP_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_provisioner_prov_input_str_comp_param* **provisioner_prov_**

Event parameter of ESP_BLE_MESH_PROVISIONER_PROV_INPUT_STRING_COMP_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_provisioner_set_node_name_comp_param* **provisioner_set_**

Event parameter of ESP_BLE_MESH_PROVISIONER_SET_NODE_NAME_COMP_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_provisioner_add_local_app_key_comp_param* **provisioner_a**

Event parameter of ESP_BLE_MESH_PROVISIONER_ADD_LOCAL_APP_KEY_COMP_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_provisioner_update_local_app_key_comp_param* **provisioner**

Event parameter of ESP_BLE_MESH_PROVISIONER_UPDATE_LOCAL_APP_KEY_COMP_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_provisioner_bind_local_mod_app_comp_param* **provisioner_**

Event parameter of ESP_BLE_MESH_PROVISIONER_BIND_APP_KEY_TO_MODEL_COMP_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_provisioner_add_local_net_key_comp_param* **provisioner_ad**
Event parameter of ESP_BLE_MESH_PROVISIONER_ADD_LOCAL_NET_KEY_COMP_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_provisioner_update_local_net_key_comp_param* **provisioner**
Event parameter of ESP_BLE_MESH_PROVISIONER_UPDATE_LOCAL_NET_KEY_COMP_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_provisioner_store_node_comp_data_comp_param* **provisione**
Event parameter of ESP_BLE_MESH_PROVISIONER_STORE_NODE_COMP_DATA_COMP_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_provisioner_delete_node_with_uuid_comp_data_comp_param*
Event parameter of ESP_BLE_MESH_PROVISIONER_DELETE_NODE_WITH_UUID_COMP_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_provisioner_delete_node_with_addr_comp_data_comp_param*
Event parameter of ESP_BLE_MESH_PROVISIONER_DELETE_NODE_WITH_ADDR_COMP_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_set_fast_prov_info_comp_param* **set_fast_prov_info_comp**
Event parameter of ESP_BLE_MESH_SET_FAST_PROV_INFO_COMP_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_set_fast_prov_action_comp_param* **set_fast_prov_action_c**
Event parameter of ESP_BLE_MESH_SET_FAST_PROV_ACTION_COMP_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_heartbeat_msg_recv_param* **heartbeat_msg_recv**
Event parameter of ESP_BLE_MESH_HEARTBEAT_MESSAGE_RECV_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_lpn_enable_comp_param* **lpn_enable_comp**
Event parameter of ESP_BLE_MESH_LPN_ENABLE_COMP_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_lpn_disable_comp_param* **lpn_disable_comp**
Event parameter of ESP_BLE_MESH_LPN_DISABLE_COMP_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_lpn_poll_comp_param* **lpn_poll_comp**
Event parameter of ESP_BLE_MESH_LPN_POLL_COMP_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_lpn_friendship_establish_param* **lpn_friendship_establish**
Event parameter of ESP_BLE_MESH_LPN_FRIENDSHIP_ESTABLISH_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_lpn_friendship_terminate_param* **lpn_friendship_terminate**
Event parameter of ESP_BLE_MESH_LPN_FRIENDSHIP_TERMINATE_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_friend_friendship_establish_param* **frnd_friendship_establi**
Event parameter of ESP_BLE_MESH_FRIEND_FRIENDSHIP_ESTABLISH_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_friend_friendship_terminate_param* **frnd_friendship_termin**
Event parameter of ESP_BLE_MESH_FRIEND_FRIENDSHIP_TERMINATE_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_proxy_client_recv_adv_pkt_param* **proxy_client_recv_adv_p**
Event parameter of ESP_BLE_MESH_PROXY_CLIENT_RECV_ADV_PKT_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_proxy_client_connected_param* **proxy_client_connected**
Event parameter of ESP_BLE_MESH_PROXY_CLIENT_CONNECTED_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_proxy_client_disconnected_param* **proxy_client_disconnecte**
Event parameter of ESP_BLE_MESH_PROXY_CLIENT_DISCONNECTED_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_proxy_client_recv_filter_status_param* **proxy_client_recv_f**
Event parameter of ESP_BLE_MESH_PROXY_CLIENT_RECV_FILTER_STATUS_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_proxy_client_connect_comp_param* **proxy_client_connect_c**
Event parameter of ESP_BLE_MESH_PROXY_CLIENT_CONNECT_COMP_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_proxy_client_disconnect_comp_param* **proxy_client_disconn**
Event parameter of ESP_BLE_MESH_PROXY_CLIENT_DISCONNECT_COMP_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_proxy_client_set_filter_type_comp_param* **proxy_client_set**
Event parameter of ESP_BLE_MESH_PROXY_CLIENT_SET_FILTER_TYPE_COMP_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_proxy_client_add_filter_addr_comp_param* **proxy_client_ad**
Event parameter of ESP_BLE_MESH_PROXY_CLIENT_ADD_FILTER_ADDR_COMP_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_proxy_client_remove_filter_addr_comp_param* **proxy_client**
Event parameter of ESP_BLE_MESH_PROXY_CLIENT_REMOVE_FILTER_ADDR_COMP_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_start_ble_advertising_comp_param* **start_ble_advertising_**
Event parameter of ESP_BLE_MESH_START_BLE_ADVERTISING_COMP_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_stop_ble_advertising_comp_param* **stop_ble_advertising_co**
Event parameter of ESP_BLE_MESH_STOP_BLE_ADVERTISING_COMP_EVT

struct *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_deinit_mesh_comp_param* **deinit_mesh_comp**
Event parameter of ESP_BLE_MESH_DEINIT_MESH_COMP_EVT

struct ble_mesh_deinit_mesh_comp_param
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_DEINIT_MESH_COMP_EVT.

Public Members

int **err_code**
Indicate the result of BLE Mesh deinitialization

struct ble_mesh_friend_friendship_establish_param
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_FRIEND_FRIENDSHIP_ESTABLISH_EVT.

Public Members

uint16_t **lpn_addr**
Low Power Node unicast address

struct ble_mesh_friend_friendship_terminate_param
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_FRIEND_FRIENDSHIP_TERMINATE_EVT.

Public Types

enum [anonymous]

This enum value is the reason of friendship termination on the friend node side

Values:

ESP_BLE_MESH_FRND_FRIENDSHIP_TERMINATE_ESTABLISH_FAIL

Friend Offer has been sent, but Friend Offer is not received within 1 second, friendship fails to be established

ESP_BLE_MESH_FRND_FRIENDSHIP_TERMINATE_POLL_TIMEOUT

Friendship is established, PollTimeout timer expires and no Friend Poll/Sub Add/Sub Remove is received

ESP_BLE_MESH_FRND_FRIENDSHIP_TERMINATE_RECV_FRND_REQ

Receive Friend Request from existing Low Power Node

ESP_BLE_MESH_FRND_FRIENDSHIP_TERMINATE_RECV_FRND_CLEAR

Receive Friend Clear from other friend node

ESP_BLE_MESH_FRND_FRIENDSHIP_TERMINATE_DISABLE

Friend feature disabled or corresponding NetKey is deleted

Public Members

uint16_t **lpn_addr**

Low Power Node unicast address

esp_ble_mesh_prov_cb_param_t::*ble_mesh_friend_friendship_terminate_param*::[**anonymous**] **reason**

This enum value is the reason of friendship termination on the friend node side Friendship terminated reason

struct ble_mesh_heartbeat_msg_recv_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_HEARTBEAT_MESSAGE_RECV_EVT.

Public Members

uint8_t **hops**

Heartbeat hops (InitTTL - RxTTL + 1)

uint16_t **feature**

Bit field of currently active features of the node

struct ble_mesh_input_evt_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_NODE_PROV_INPUT_EVT.

Public Members

esp_ble_mesh_input_action_t **action**

Action of Input OOB Authentication

uint8_t **size**

Size of Input OOB Authentication

struct ble_mesh_input_number_comp_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_NODE_PROV_INPUT_NUM_COMP_EVT.

Public Members

int **err_code**

Indicate the result of inputting number

struct ble_mesh_input_string_comp_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_NODE_PROV_INPUT_STR_COMP_EVT.

Public Members

int **err_code**

Indicate the result of inputting string

struct ble_mesh_link_close_evt_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_NODE_PROV_LINK_CLOSE_EVT.

Public Members

esp_ble_mesh_prov_bearer_t **bearer**

Type of the bearer used when device link is closed

struct ble_mesh_link_open_evt_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_NODE_PROV_LINK_OPEN_EVT.

Public Members

esp_ble_mesh_prov_bearer_t **bearer**

Type of the bearer used when device link is open

struct ble_mesh_lpn_disable_comp_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_LPN_DISABLE_COMP_EVT.

Public Members

int **err_code**

Indicate the result of disabling LPN functionality

struct ble_mesh_lpn_enable_comp_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_LPN_ENABLE_COMP_EVT.

Public Members

int **err_code**

Indicate the result of enabling LPN functionality

struct ble_mesh_lpn_friendship_establish_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_LPN_FRIENDSHIP_ESTABLISH_EVT.

Public Members

uint16_t **friend_addr**

Friend Node unicast address

struct ble_mesh_lpn_friendship_terminate_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_LPN_FRIENDSHIP_TERMINATE_EVT.

Public Members

uint16_t **friend_addr**

Friend Node unicast address

struct ble_mesh_lpn_poll_comp_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_LPN_POLL_COMP_EVT.

Public Members

int **err_code**

Indicate the result of sending Friend Poll

struct ble_mesh_output_num_evt_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_NODE_PROV_OUTPUT_NUMBER_EVT.

Public Members

esp_ble_mesh_output_action_t **action**

Action of Output OOB Authentication

uint32_t **number**

Number of Output OOB Authentication

struct ble_mesh_output_str_evt_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_NODE_PROV_OUTPUT_STRING_EVT.

Public Members

char **string**[8]

String of Output OOB Authentication

struct ble_mesh_prov_disable_comp_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_NODE_PROV_DISABLE_COMP_EVT.

Public Members

int **err_code**

Indicate the result of disabling BLE Mesh device

struct ble_mesh_prov_enable_comp_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_NODE_PROV_ENABLE_COMP_EVT.

Public Members

int **err_code**

Indicate the result of enabling BLE Mesh device

struct ble_mesh_prov_register_comp_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROV_REGISTER_COMP_EVT.

Public Members

int **err_code**

Indicate the result of BLE Mesh initialization

struct ble_mesh_provision_complete_evt_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_NODE_PROV_COMPLETE_EVT.

Public Members

uint16_t **net_idx**

NetKey Index

uint8_t **net_key**[16]
NetKey

uint16_t **addr**
Primary address

uint8_t **flags**
Flags

uint32_t **iv_index**
IV Index

struct ble_mesh_provision_reset_param
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_NODE_PROV_RESET_EVT.

struct ble_mesh_provisioner_add_local_app_key_comp_param
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROVISIONER_ADD_LOCAL_APP_KEY_COMP_

Public Members

int **err_code**
Indicate the result of adding local AppKey by the Provisioner

uint16_t **app_idx**
AppKey Index

struct ble_mesh_provisioner_add_local_net_key_comp_param
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROVISIONER_ADD_LOCAL_NET_KEY_COMP_

Public Members

int **err_code**
Indicate the result of adding local NetKey by the Provisioner

uint16_t **net_idx**
NetKey Index

struct ble_mesh_provisioner_add_unprov_dev_comp_param
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROVISIONER_ADD_UNPROV_DEV_COMP_EVT

Public Members

int **err_code**
Indicate the result of adding device into queue by the Provisioner

struct ble_mesh_provisioner_bind_local_mod_app_comp_param
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROVISIONER_BIND_APP_KEY_TO_MODEL_CO

Public Members

int **err_code**

Indicate the result of binding AppKey with model by the Provisioner

uint16_t **element_addr**

Element address

uint16_t **app_idx**

AppKey Index

uint16_t **company_id**

Company ID

uint16_t **model_id**

Model ID

struct ble_mesh_provisioner_delete_dev_comp_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROVISIONER_DELETE_DEV_COMP_EVT.

Public Members

int **err_code**

Indicate the result of deleting device by the Provisioner

struct ble_mesh_provisioner_delete_node_with_addr_comp_data_comp_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROVISIONER_DELETE_NODE_WITH_ADDR_C

Public Members

int **err_code**

Indicate the result of deleting node with unicast address by the Provisioner

uint16_t **unicast_addr**

Node unicast address

struct ble_mesh_provisioner_delete_node_with_uuid_comp_data_comp_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROVISIONER_DELETE_NODE_WITH_UUID_CO

Public Members

int **err_code**

Indicate the result of deleting node with uuid by the Provisioner

uint8_t **uuid**[16]

Node device uuid

struct ble_mesh_provisioner_link_close_evt_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROVISIONER_PROV_LINK_CLOSE_EVT.

Public Members

esp_ble_mesh_prov_bearer_t **bearer**

Type of the bearer used when Provisioner link is closed

uint8_t **reason**

Reason of the closed provisioning link

struct ble_mesh_provisioner_link_open_evt_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROVISIONER_PROV_LINK_OPEN_EVT.

Public Members

esp_ble_mesh_prov_bearer_t **bearer**

Type of the bearer used when Provisioner link is opened

struct ble_mesh_provisioner_prov_comp_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROVISIONER_PROV_COMPLETE_EVT.

Public Members

uint16_t **node_idx**

Index of the provisioned device

esp_ble_mesh_octet16_t **device_uuid**

Device UUID of the provisioned device

uint16_t **unicast_addr**

Primary address of the provisioned device

uint8_t **element_num**

Element count of the provisioned device

uint16_t **netkey_idx**

NetKey Index of the provisioned device

struct ble_mesh_provisioner_prov_dev_with_addr_comp_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROVISIONER_PROV_DEV_WITH_ADDR_COMP

Public Members

int **err_code**

Indicate the result of Provisioner starting to provision a device

struct ble_mesh_provisioner_prov_disable_comp_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROVISIONER_PROV_DISABLE_COMP_EVT.

Public Members

int **err_code**

Indicate the result of disabling BLE Mesh Provisioner

struct ble_mesh_provisioner_prov_enable_comp_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROVISIONER_PROV_ENABLE_COMP_EVT.

Public Members

int **err_code**

Indicate the result of enabling BLE Mesh Provisioner

struct ble_mesh_provisioner_prov_input_evt_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROVISIONER_PROV_INPUT_EVT.

Public Members

esp_ble_mesh_oob_method_t **method**

Method of device Output OOB Authentication

esp_ble_mesh_output_action_t **action**

Action of device Output OOB Authentication

uint8_t **size**

Size of device Output OOB Authentication

uint8_t **link_idx**

Index of the provisioning link

struct ble_mesh_provisioner_prov_input_num_comp_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROVISIONER_PROV_INPUT_NUMBER_COMP_E

Public Members

int **err_code**

Indicate the result of inputting number by the Provisioner

struct ble_mesh_provisioner_prov_input_str_comp_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROVISIONER_PROV_INPUT_STRING_COMP_EV

Public Members

int **err_code**

Indicate the result of inputting string by the Provisioner

struct ble_mesh_provisioner_prov_output_evt_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROVISIONER_PROV_OUTPUT_EVT.

Public Members

esp_ble_mesh_oob_method_t **method**

Method of device Input OOB Authentication

esp_ble_mesh_input_action_t **action**

Action of device Input OOB Authentication

uint8_t **size**

Size of device Input OOB Authentication

uint8_t **link_idx**

Index of the provisioning link

char **string**[8]

String output by the Provisioner

uint32_t **number**

Number output by the Provisioner

union *esp_ble_mesh_prov_cb_param_t*::*ble_mesh_provisioner_prov_output_evt_param*::[**anonymous**] [**anon**

struct ble_mesh_provisioner_prov_read_oob_pub_key_comp_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROVISIONER_PROV_READ_OOB_PUB_KEY_C

Public Members

int **err_code**

Indicate the result of setting OOB Public Key by the Provisioner

struct ble_mesh_provisioner_prov_read_oob_pub_key_evt_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROVISIONER_PROV_READ_OOB_PUB_KEY_E

Public Members

uint8_t **link_idx**

Index of the provisioning link

struct ble_mesh_provisioner_recv_unprov_adv_pkt_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROVISIONER_RECV_UNPROV_ADV_PKT_EVT.

Public Members

uint8_t **dev_uuid**[16]

Device UUID of the unprovisioned device

esp_ble_mesh_bd_addr_t **addr**

Device address of the unprovisioned device

esp_ble_mesh_addr_type_t **addr_type**

Device address type

uint16_t **oob_info**

OOB Info of the unprovisioned device

uint8_t **adv_type**

Avertising type of the unprovisioned device

esp_ble_mesh_prov_bearer_t **bearer**

Bearer of the unprovisioned device

int8_t **rssi**

RSSI of the received advertising packet

struct ble_mesh_provisioner_set_dev_uuid_match_comp_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROVISIONER_SET_DEV_UUID_MATCH_COMP_

Public Members

int **err_code**

Indicate the result of setting Device UUID match value by the Provisioner

struct ble_mesh_provisioner_set_node_name_comp_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROVISIONER_SET_NODE_NAME_COMP_EVT.

Public Members

int **err_code**

Indicate the result of setting provisioned device name by the Provisioner

uint16_t **node_index**

Index of the provisioned device

struct ble_mesh_provisioner_set_primary_elem_addr_comp_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROVISIONER_SET_PRIMARY_ELEM_ADDR_CO

Public Members

int **err_code**

Indicate the result of setting unicast address of primary element by the Provisioner

struct ble_mesh_provisioner_set_prov_data_info_comp_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROVISIONER_SET_PROV_DATA_INFO_COMP_

Public Members

int **err_code**

Indicate the result of setting provisioning info by the Provisioner

struct ble_mesh_provisioner_set_static_oob_val_comp_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROVISIONER_SET_STATIC_OOB_VALUE_COMP

Public Members

int **err_code**

Indicate the result of setting static oob value by the Provisioner

struct ble_mesh_provisioner_store_node_comp_data_comp_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROVISIONER_STORE_NODE_COMP_DATA_CO

Public Members

int **err_code**

Indicate the result of storing node composition data by the Provisioner

uint16_t **addr**

Node element address

struct ble_mesh_provisioner_update_local_app_key_comp_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROVISIONER_UPDATE_LOCAL_APP_KEY_CO

Public Members

int **err_code**

Indicate the result of updating local AppKey by the Provisioner

uint16_t **net_idx**

NetKey Index

uint16_t **app_idx**

AppKey Index

struct ble_mesh_provisioner_update_local_net_key_comp_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROVISIONER_UPDATE_LOCAL_NET_KEY_COM

Public Members

int **err_code**

Indicate the result of updating local NetKey by the Provisioner

uint16_t **net_idx**

NetKey Index

struct ble_mesh_proxy_client_add_filter_addr_comp_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROXY_CLIENT_ADD_FILTER_ADDR_COMP_EV

Public Members

int **err_code**

Indicate the result of Proxy Client add filter address

uint8_t **conn_handle**

Proxy connection handle

uint16_t **net_idx**

Corresponding NetKey Index

struct ble_mesh_proxy_client_connect_comp_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROXY_CLIENT_CONNECT_COMP_EVT.

Public Members

int **err_code**

Indicate the result of Proxy Client connect

esp_ble_mesh_bd_addr_t **addr**

Device address of the Proxy Server

esp_ble_mesh_addr_type_t **addr_type**

Device address type

uint16_t **net_idx**

Corresponding NetKey Index

struct ble_mesh_proxy_client_connected_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROXY_CLIENT_CONNECTED_EVT.

Public Members

esp_ble_mesh_bd_addr_t **addr**
Device address of the Proxy Server

esp_ble_mesh_addr_type_t **addr_type**
Device address type

uint8_t **conn_handle**
Proxy connection handle

uint16_t **net_idx**
Corresponding NetKey Index

struct ble_mesh_proxy_client_disconnect_comp_param
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROXY_CLIENT_DISCONNECT_COMP_EVT.

Public Members

int **err_code**
Indicate the result of Proxy Client disconnect

uint8_t **conn_handle**
Proxy connection handle

struct ble_mesh_proxy_client_disconnected_param
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROXY_CLIENT_DISCONNECTED_EVT.

Public Members

esp_ble_mesh_bd_addr_t **addr**
Device address of the Proxy Server

esp_ble_mesh_addr_type_t **addr_type**
Device address type

uint8_t **conn_handle**
Proxy connection handle

uint16_t **net_idx**
Corresponding NetKey Index

uint8_t **reason**
Proxy disconnect reason

struct ble_mesh_proxy_client_recv_adv_pkt_param
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROXY_CLIENT_RECV_ADV_PKT_EVT.

Public Members

esp_ble_mesh_bd_addr_t **addr**

Device address

esp_ble_mesh_addr_type_t **addr_type**

Device address type

uint16_t **net_idx**

Network ID related NetKey Index

uint8_t **net_id**[8]

Network ID contained in the advertising packet

int8_t **rssi**

RSSI of the received advertising packet

struct ble_mesh_proxy_client_recv_filter_status_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROXY_CLIENT_RECV_FILTER_STATUS_EVT.

Public Members

uint8_t **conn_handle**

Proxy connection handle

uint16_t **server_addr**

Proxy Server primary element address

uint16_t **net_idx**

Corresponding NetKey Index

uint8_t **filter_type**

Proxy Server filter type(whitelist or blacklist)

uint16_t **list_size**

Number of addresses in the Proxy Server filter list

struct ble_mesh_proxy_client_remove_filter_addr_comp_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROXY_CLIENT_REMOVE_FILTER_ADDR_COM

Public Members

int **err_code**

Indicate the result of Proxy Client remove filter address

uint8_t **conn_handle**

Proxy connection handle

uint16_t **net_idx**

Corresponding NetKey Index

struct ble_mesh_proxy_client_set_filter_type_comp_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_PROXY_CLIENT_SET_FILTER_TYPE_COMP_EV

Public Members

int **err_code**

Indicate the result of Proxy Client set filter type

uint8_t **conn_handle**

Proxy connection handle

uint16_t **net_idx**

Corresponding NetKey Index

struct ble_mesh_proxy_gatt_disable_comp_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_NODE_PROXY_GATT_DISABLE_COMP_EVT.

Public Members

int **err_code**

Indicate the result of disabling Mesh Proxy Service

struct ble_mesh_proxy_gatt_enable_comp_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_NODE_PROXY_GATT_ENABLE_COMP_EVT.

Public Members

int **err_code**

Indicate the result of enabling Mesh Proxy Service

struct ble_mesh_proxy_identity_enable_comp_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_NODE_PROXY_IDENTITY_ENABLE_COMP_EVT

Public Members

int **err_code**

Indicate the result of enabling Mesh Proxy advertising

struct ble_mesh_set_fast_prov_action_comp_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_SET_FAST_PROV_ACTION_COMP_EVT.

Public Members

uint8_t **status_action**

Indicate the result of setting action of fast provisioning

struct ble_mesh_set_fast_prov_info_comp_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_SET_FAST_PROV_INFO_COMP_EVT.

Public Members

uint8_t **status_unicast**

Indicate the result of setting unicast address range of fast provisioning

uint8_t **status_net_idx**

Indicate the result of setting NetKey Index of fast provisioning

uint8_t **status_match**

Indicate the result of setting matching Device UUID of fast provisioning

struct ble_mesh_set_oob_pub_key_comp_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_NODE_PROV_SET_OOB_PUB_KEY_COMP_EVT

Public Members

int **err_code**

Indicate the result of setting OOB Public Key

struct ble_mesh_set_unprov_dev_name_comp_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_NODE_SET_UNPROV_DEV_NAME_COMP_EVT.

Public Members

int **err_code**

Indicate the result of setting BLE Mesh device name

struct ble_mesh_start_ble_advertising_comp_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_START_BLE_ADVERTISING_COMP_EVT.

Public Members

int **err_code**

Indicate the result of starting BLE advertising

uint8_t **index**

Index of the BLE advertising

struct ble_mesh_stop_ble_advertising_comp_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_STOP_BLE_ADVERTISING_COMP_EVT.

Public Members

int **err_code**

Indicate the result of stopping BLE advertising

uint8_t **index**

Index of the BLE advertising

union esp_ble_mesh_server_state_value_t

#include <esp_ble_mesh_defs.h> Server model state value union.

Public Members

uint8_t **onoff**

The value of the Generic OnOff state

The value of the Light LC Light OnOff state

struct *esp_ble_mesh_server_state_value_t*::[**anonymous**] **gen_onoff**

The Generic OnOff state

int16_t **level**

The value of the Generic Level state

struct *esp_ble_mesh_server_state_value_t*::[**anonymous**] **gen_level**

The Generic Level state

uint8_t **onpowerup**

The value of the Generic OnPowerUp state

struct *esp_ble_mesh_server_state_value_t*::[**anonymous**] **gen_onpowerup**

The Generic OnPowerUp state

uint16_t **power**

The value of the Generic Power Actual state

struct *esp_ble_mesh_server_state_value_t*::[**anonymous**] **gen_power_actual**

The Generic Power Actual state

uint16_t **lightness**

The value of the Light Lightness Actual state

The value of the Light Lightness Linear state

The value of the Light CTL Lightness state

The value of the Light HSL Lightness state

The value of the Light xyL Lightness state

struct *esp_ble_mesh_server_state_value_t*::[**anonymous**] **light_lightness_actual**

The Light Lightness Actual state

struct *esp_ble_mesh_server_state_value_t*::[**anonymous**] **light_lightness_linear**

The Light Lightness Linear state

struct *esp_ble_mesh_server_state_value_t*::[**anonymous**] **light_ctl_lightness**

The Light CTL Lightness state

uint16_t **temperature**

The value of the Light CTL Temperature state

int16_t **delta_uv**

The value of the Light CTL Delta UV state

struct *esp_ble_mesh_server_state_value_t*::[**anonymous**] **light_ctl_temp_delta_uv**

The Light CTL Temperature & Delta UV states

struct *esp_ble_mesh_server_state_value_t*::[**anonymous**] **light_hsl_lightness**

The Light HSL Lightness state

uint16_t **hue**

The value of the Light HSL Hue state

struct *esp_ble_mesh_server_state_value_t*::[**anonymous**] **light_hsl_hue**

The Light HSL Hue state

uint16_t **saturation**

The value of the Light HSL Saturation state

struct *esp_ble_mesh_server_state_value_t*::[**anonymous**] **light_hsl_saturation**

The Light HSL Saturation state

struct *esp_ble_mesh_server_state_value_t*::[**anonymous**] **light_xyl_lightness**

The Light xyL Lightness state

struct *esp_ble_mesh_server_state_value_t*::[**anonymous**] **light_lc_light_onoff**

The Light LC Light OnOff state

union esp_ble_mesh_model_cb_param_t

#include <esp_ble_mesh_defs.h> BLE Mesh model callback parameters union.

Public Members

struct *esp_ble_mesh_model_cb_param_t*::*ble_mesh_model_operation_evt_param* **model_operation**

Event parameter of ESP_BLE_MESH_MODEL_OPERATION_EVT

struct *esp_ble_mesh_model_cb_param_t*::*ble_mesh_model_send_comp_param* **model_send_comp**

Event parameter of ESP_BLE_MESH_MODEL_SEND_COMP_EVT

struct *esp_ble_mesh_model_cb_param_t*::*ble_mesh_model_publish_comp_param* **model_publish_comp**
Event parameter of ESP_BLE_MESH_MODEL_PUBLISH_COMP_EVT

struct *esp_ble_mesh_model_cb_param_t*::*ble_mesh_mod_recv_publish_msg_param* **client_recv_publish_msg**
Event parameter of ESP_BLE_MESH_CLIENT_MODEL_RECV_PUBLISH_MSG_EVT

struct *esp_ble_mesh_model_cb_param_t*::*ble_mesh_client_model_send_timeout_param* **client_send_timeout**
Event parameter of ESP_BLE_MESH_CLIENT_MODEL_SEND_TIMEOUT_EVT

struct *esp_ble_mesh_model_cb_param_t*::*ble_mesh_model_publish_update_evt_param* **model_publish_update**
Event parameter of ESP_BLE_MESH_MODEL_PUBLISH_UPDATE_EVT

struct *esp_ble_mesh_model_cb_param_t*::*ble_mesh_server_model_update_state_comp_param* **server_model_upd**
Event parameter of ESP_BLE_MESH_SERVER_MODEL_UPDATE_STATE_COMP_EVT

struct ble_mesh_client_model_send_timeout_param
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_CLIENT_MODEL_SEND_TIMEOUT_EVT.

Public Members

uint32_t **opcode**
Opcode of the previously sent message

esp_ble_mesh_model_t ***model**
Pointer to the model which sends the previous message

esp_ble_mesh_msg_ctx_t ***ctx**
Pointer to the context of the previous message

struct ble_mesh_mod_recv_publish_msg_param
#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_CLIENT_MODEL_RECV_PUBLISH_MSG_EVT.

Public Members

uint32_t **opcode**
Opcode of the unsolicited received message

esp_ble_mesh_model_t ***model**
Pointer to the model which receives the message

esp_ble_mesh_msg_ctx_t ***ctx**
Pointer to the context of the message

uint16_t **length**
Length of the received message

uint8_t ***msg**
Value of the received message

struct ble_mesh_model_operation_evt_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_MODEL_OPERATION_EVT.

Public Members

uint32_t **opcode**

Opcode of the received message

esp_ble_mesh_model_t ***model**

Pointer to the model which receives the message

esp_ble_mesh_msg_ctx_t ***ctx**

Pointer to the context of the received message

uint16_t **length**

Length of the received message

uint8_t ***msg**

Value of the received message

struct ble_mesh_model_publish_comp_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_MODEL_PUBLISH_COMP_EVT.

Public Members

int **err_code**

Indicate the result of publishing a message

esp_ble_mesh_model_t ***model**

Pointer to the model which publishes the message

struct ble_mesh_model_publish_update_evt_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_MODEL_PUBLISH_UPDATE_EVT.

Public Members

esp_ble_mesh_model_t ***model**

Pointer to the model which is going to update its publish message

struct ble_mesh_model_send_comp_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_MODEL_SEND_COMP_EVT.

Public Members

int **err_code**

Indicate the result of sending a message

uint32_t **opcode**

Opcode of the message

esp_ble_mesh_model_t ***model**

Pointer to the model which sends the message

esp_ble_mesh_msg_ctx_t ***ctx**

Context of the message

struct ble_mesh_server_model_update_state_comp_param

#include <esp_ble_mesh_defs.h> ESP_BLE_MESH_SERVER_MODEL_UPDATE_STATE_COMP_EVT.

Public Members

int **err_code**

Indicate the result of updating server model state

esp_ble_mesh_model_t ***model**

Pointer to the server model which state value is updated

esp_ble_mesh_server_state_type_t **type**

Type of the updated server state

Structures

struct esp_ble_mesh_deinit_param_t

BLE Mesh deinit parameters

Public Members

bool **erase_flash**

Indicate if erasing flash when deinit mesh stack

struct esp_ble_mesh_elem_t

Abstraction that describes a BLE Mesh Element. This structure is associated with struct bt_mesh_elem in mesh_access.h

Public Members

uint16_t **element_addr**

Element Address, assigned during provisioning.

const uint16_t **location**

Location Descriptor (GATT Bluetooth Namespace Descriptors)

const uint8_t **sig_model_count**

SIG Model count

const uint8_t **vnd_model_count**

Vendor Model count

esp_ble_mesh_model_t ***sig_models**

SIG Models

esp_ble_mesh_model_t ***vnd_models**

Vendor Models

struct esp_ble_mesh_model_pub_t

Abstraction that describes a model publication context. This structure is associated with struct bt_mesh_model_pub in mesh_access.h

Public Members

esp_ble_mesh_model_t ***model**

Pointer to the model to which the context belongs. Initialized by the stack.

uint16_t **publish_addr**

Publish Address.

uint16_t **app_idx**

Publish AppKey Index.

uint16_t **cred**

Friendship Credentials Flag.

uint16_t **send_rel**

Force reliable sending (segment acks)

uint8_t **ttl**

Publish Time to Live.

uint8_t **retransmit**

Retransmit Count & Interval Steps.

uint8_t **period**

Publish Period.

uint8_t **period_div**

Divisor for the Period.

uint8_t **fast_period**

Use FastPeriodDivisor

uint8_t **count**

Retransmissions left.

uint32_t **period_start**

Start of the current period.

struct net_buf_simple ***msg**

Publication buffer, containing the publication message.

This will get correctly created when the publication context has been defined using the ESP_BLE_MESH_MODEL_PUB_DEFINE macro.

ESP_BLE_MESH_MODEL_PUB_DEFINE(name, size);

esp_ble_mesh_cb_t **update**

Callback used to update publish message. Initialized by the stack.

struct k_delayed_work **timer**

Publish Period Timer. Initialized by the stack.

uint8_t **dev_role**

Role of the device that is going to publish messages

struct esp_ble_mesh_model_op_t

Abstraction that describes a model operation context. This structure is associated with struct bt_mesh_model_op in mesh_access.h

Public Members

const uint32_t **opcode**

Message opcode

const size_t **min_len**

Message minimum length

esp_ble_mesh_cb_t **param_cb**

Callback used to handle message. Initialized by the stack.

struct esp_ble_mesh_model

Abstraction that describes a Mesh Model instance. This structure is associated with struct bt_mesh_model in mesh_access.h

Public Members

union *esp_ble_mesh_model*::[**anonymous**] [**anonymous**]

Model ID

uint8_t **element_idx**

Internal information, mainly for persistent storage Belongs to Nth element

uint8_t **model_idx**

Is the Nth model in the element

uint16_t **flags**

Information about what has changed

esp_ble_mesh_elem_t ***element**

The Element to which this Model belongs

esp_ble_mesh_model_pub_t ***const pub**

Model Publication

uint16_t **keys**[CONFIG_BLE_MESH_MODEL_KEY_COUNT]

AppKey List

uint16_t **groups**[CONFIG_BLE_MESH_MODEL_GROUP_COUNT]

Subscription List (group or virtual addresses)

esp_ble_mesh_model_op_t ***op**

Model operation context

void ***user_data**

Model-specific user data

struct esp_ble_mesh_msg_ctx_t

Message sending context. This structure is associated with struct bt_mesh_msg_ctx in mesh_access.h

Public Members

uint16_t **net_idx**

NetKey Index of the subnet through which to send the message.

uint16_t **app_idx**

AppKey Index for message encryption.

uint16_t **addr**

Remote address.

uint16_t **recv_dst**

Destination address of a received message. Not used for sending.

int8_t **recv_rssi**

RSSI of received packet. Not used for sending.

uint8_t **recv_ttl**

Received TTL value. Not used for sending.

uint8_t **send_rel**

Force sending reliably by using segment acknowledgement

uint8_t **send_ttl**

TTL, or BLE_MESH_TTL_DEFAULT for default TTL.

uint32_t **recv_op**

Opcode of a received message. Not used for sending message.

esp_ble_mesh_model_t ***model**

Model corresponding to the message, no need to be initialized before sending message

bool **srv_send**

Indicate if the message is sent by a node server model, no need to be initialized before sending message

struct esp_ble_mesh_prov_t

Provisioning properties & capabilities. This structure is associated with struct bt_mesh_prov in mesh_access.h

struct esp_ble_mesh_comp_t

Node Composition data context. This structure is associated with struct bt_mesh_comp in mesh_access.h

Public Members

uint16_t **cid**

16-bit SIG-assigned company identifier

uint16_t **pid**

16-bit vendor-assigned product identifier

uint16_t **vid**

16-bit vendor-assigned product version identifier

size_t **element_count**

Element count

esp_ble_mesh_elem_t ***elements**

A sequence of elements

struct esp_ble_mesh_unprov_dev_add_t

Information of the device which is going to be added for provisioning.

Public Members

esp_ble_mesh_bd_addr_t **addr**

Device address

esp_ble_mesh_addr_type_t **addr_type**

Device address type

uint8_t **uuid**[16]

Device UUID

uint16_t **oob_info**

Device OOB Info ADD_DEV_START_PROV_NOW_FLAG shall not be set if the bearer has both PB-ADV and PB-GATT enabled

esp_ble_mesh_prov_bearer_t **bearer**

Provisioning Bearer

struct esp_ble_mesh_device_delete_t

Information of the device which is going to be deleted.

Public Members

esp_ble_mesh_bd_addr_t **addr**

Device address

esp_ble_mesh_addr_type_t **addr_type**

Device address type

uint8_t **uuid**[16]

Device UUID

uint8_t **flag**

BIT0: device address; BIT1: device UUID

struct esp_ble_mesh_prov_data_info_t

Information of the provisioner which is going to be updated.

Public Members

uint16_t **net_idx**

NetKey Index

uint8_t **flags**

Flags

uint32_t **iv_index**

IV Index

uint8_t **flag**

BIT0: net_idx; BIT1: flags; BIT2: iv_index

struct esp_ble_mesh_node_t

Information of the provisioned node

Public Members

esp_ble_mesh_bd_addr_t **addr**

Node device address

esp_ble_mesh_addr_type_t **addr_type**
Node device address type

uint8_t **dev_uuid**[16]
Device UUID

uint16_t **oob_info**
Node OOB information

uint16_t **unicast_addr**
Node unicast address

uint8_t **element_num**
Node element number

uint16_t **net_idx**
Node NetKey Index

uint8_t **flags**
Node key refresh flag and iv update flag

uint32_t **iv_index**
Node IV Index

uint8_t **dev_key**[16]
Node device key

char **name**[**ESP_BLE_MESH_NODE_NAME_MAX_LEN** + 1]
Node name

uint16_t **comp_length**
Length of Composition Data

uint8_t ***comp_data**
Value of Composition Data

struct esp_ble_mesh_fast_prov_info_t
Context of fast provisioning which need to be set.

Public Members

uint16_t **unicast_min**
Minimum unicast address used for fast provisioning

uint16_t **unicast_max**
Maximum unicast address used for fast provisioning

uint16_t **net_idx**
Netkey index used for fast provisioning

uint8_t **flags**
Flags used for fast provisioning

uint32_t **iv_index**
IV Index used for fast provisioning

uint8_t **offset**
Offset of the UUID to be compared

uint8_t **match_len**
Length of the UUID to be compared

uint8_t **match_val**[16]
Value of UUID to be compared

struct esp_ble_mesh_ble_adv_param_t
Context of BLE advertising parameters.

Public Members

uint16_t **interval**
BLE advertising interval

uint8_t **adv_type**
BLE advertising type

uint8_t **own_addr_type**
Own address type

uint8_t **peer_addr_type**
Peer address type

uint8_t **peer_addr**[**BD_ADDR_LEN**]
Peer address

uint16_t **duration**
Duration is milliseconds

uint16_t **period**
Period in milliseconds

uint16_t **count**
Number of advertising duration

uint8_t **priority**
Priority of BLE advertising packet

struct esp_ble_mesh_ble_adv_data_t
Context of BLE advertising data.

Public Members

uint8_t **adv_data_len**
Advertising data length

uint8_t **adv_data**[31]
Advertising data

uint8_t **scan_rsp_data_len**
Scan response data length

uint8_t **scan_rsp_data**[31]
Scan response data

struct esp_ble_mesh_client_op_pair_t
BLE Mesh client models related definitions.

Client model Get/Set message opcode and corresponding Status message opcode

Public Members

uint32_t **cli_op**
The client message opcode

uint32_t **status_op**
The server status opcode corresponding to the client message opcode

struct esp_ble_mesh_client_t
Client Model user data context.

Public Members

esp_ble_mesh_model_t ***model**
Pointer to the client model. Initialized by the stack.

int **op_pair_size**
Size of the op_pair

const *esp_ble_mesh_client_op_pair_t* ***op_pair**
Table containing get/set message opcode and corresponding status message opcode

uint32_t **publish_status**
Callback used to handle the received unsolicited message. Initialized by the stack.

void ***internal_data**
Pointer to the internal data of client model

uint8_t **msg_role**
Role of the device (Node/Provisioner) that is going to send messages

struct esp_ble_mesh_client_common_param_t

Common parameters of the messages sent by Client Model.

Public Members

esp_ble_mesh_opcode_t **opcode**

Message opcode

esp_ble_mesh_model_t ***model**

Pointer to the client model structure

esp_ble_mesh_msg_ctx_t **ctx**

The context used to send message

int32_t **msg_timeout**

Timeout value (ms) to get response to the sent message Note: if using default timeout value in menuconfig, make sure to set this value to 0

uint8_t **msg_role**

Role of the device - Node/Provisioner

struct esp_ble_mesh_state_transition_t

Parameters of the server model state transition

Public Functions

BLE_MESH_ATOMIC_DEFINE(flag, *ESP_BLE_MESH_SERVER_FLAG_MAX*)

Flag used to indicate if the transition timer has been started internally.

If the model which contains *esp_ble_mesh_state_transition_t* sets "set_auto_rsp" to ESP_BLE_MESH_SERVER_RSP_BY_APP, the handler of the timer shall be initialized by the users.

And users can use this flag to indicate whether the timer is started or not.

Public Members

bool **just_started**

Indicate if the state transition has just started

uint8_t **trans_time**

State transition time

uint8_t **remain_time**

Remaining time of state transition

uint8_t **delay**

Delay before starting state transition

uint32_t **quo_tt**

Duration of each divided transition step

uint32_t **counter**

Number of steps which the transition duration is divided

uint32_t **total_duration**

State transition total duration

int64_t **start_timestamp**

Time when the state transition is started

struct k_delayed_work **timer**

Timer used for state transition

struct esp_ble_mesh_last_msg_info_t

Parameters of the server model received last same set message.

Public Members

uint8_t **tid**

Transaction number of the last message

uint16_t **src**

Source address of the last message

uint16_t **dst**

Destination address of the last message

int64_t **timestamp**

Time when the last message is received

struct esp_ble_mesh_server_rsp_ctrl_t

Parameters of the Server Model response control

Public Members

uint8_t **get_auto_rsp**

BLE Mesh Server Response Option.

1. If get_auto_rsp is set to ESP_BLE_MESH_SERVER_RSP_BY_APP, then the response of Client Get messages need to be replied by the application;
2. If get_auto_rsp is set to ESP_BLE_MESH_SERVER_AUTO_RSP, then the response of Client Get messages will be replied by the server models;

3. If set_auto_rsp is set to ESP_BLE_MESH_SERVER_RSP_BY_APP, then the response of Client Set messages need to be replied by the application;
4. If set_auto_rsp is set to ESP_BLE_MESH_SERVER_AUTO_RSP, then the response of Client Set messages will be replied by the server models;
5. If status_auto_rsp is set to ESP_BLE_MESH_SERVER_RSP_BY_APP, then the response of Server Status messages need to be replied by the application;
6. If status_auto_rsp is set to ESP_BLE_MESH_SERVER_AUTO_RSP, then the response of Server Status messages will be replied by the server models;Response control for Client Get messages

uint8_t **set_auto_rsp**

Response control for Client Set messages

uint8_t **status_auto_rsp**

Response control for Server Status messages

Macros

ESP_BLE_HOST_STATUS_ENABLED

ESP_BLE_HOST_STATUS_CHECK(status)

The maximum length of a BLE Mesh message, including Opcode, Payload and TransMIC

ESP_BLE_MESH_SDU_MAX_LEN

Length of a short Mesh MIC.

ESP_BLE_MESH_MIC_SHORT

Length of a long Mesh MIC.

ESP_BLE_MESH_MIC_LONG

The maximum length of a BLE Mesh provisioned node name

ESP_BLE_MESH_NODE_NAME_MAX_LEN

The maximum length of a BLE Mesh unprovisioned device name

ESP_BLE_MESH_DEVICE_NAME_MAX_LEN

Define the BLE Mesh octet 16 bytes size

ESP_BLE_MESH_OCTET16_LEN

ESP_BLE_MESH_OCTET8_LEN

ESP_BLE_MESH_ADDR_UNASSIGNED

ESP_BLE_MESH_ADDR_ALL_NODES

ESP_BLE_MESH_ADDR_PROXIES

ESP_BLE_MESH_ADDR_FRIENDS

ESP_BLE_MESH_ADDR_RELAYS

ESP_BLE_MESH_KEY_UNUSED

ESP_BLE_MESH_KEY_DEV

ESP_BLE_MESH_KEY_PRIMARY

ESP_BLE_MESH_KEY_ANY
: Primary Network Key index

ESP_BLE_MESH_NET_PRIMARY
: Relay state value

ESP_BLE_MESH_RELAY_DISABLED

ESP_BLE_MESH_RELAY_ENABLED

ESP_BLE_MESH_RELAY_NOT_SUPPORTED
: Beacon state value

ESP_BLE_MESH_BEACON_DISABLED

ESP_BLE_MESH_BEACON_ENABLED
: GATT Proxy state value

ESP_BLE_MESH_GATT_PROXY_DISABLED

ESP_BLE_MESH_GATT_PROXY_ENABLED

ESP_BLE_MESH_GATT_PROXY_NOT_SUPPORTED
: Friend state value

ESP_BLE_MESH_FRIEND_DISABLED

ESP_BLE_MESH_FRIEND_ENABLED

ESP_BLE_MESH_FRIEND_NOT_SUPPORTED
: Node identity state value

ESP_BLE_MESH_NODE_IDENTITY_STOPPED

ESP_BLE_MESH_NODE_IDENTITY_RUNNING

ESP_BLE_MESH_NODE_IDENTITY_NOT_SUPPORTED
: Supported features

ESP_BLE_MESH_FEATURE_RELAY

ESP_BLE_MESH_FEATURE_PROXY

ESP_BLE_MESH_FEATURE_FRIEND

ESP_BLE_MESH_FEATURE_LOW_POWER

ESP_BLE_MESH_FEATURE_ALL_SUPPORTED

ESP_BLE_MESH_ADDR_IS_UNICAST(addr)

ESP_BLE_MESH_ADDR_IS_GROUP(addr)

ESP_BLE_MESH_ADDR_IS_VIRTUAL(addr)

ESP_BLE_MESH_ADDR_IS_RFU(addr)

ESP_BLE_MESH_INVALID_NODE_INDEX

ESP_BLE_MESH_TRANSMIT(count, int_ms)

Encode transmission count & interval steps.

Note For example, ESP_BLE_MESH_TRANSMIT(2, 20) means that the message will be sent about 90ms(count is 3, step is 1, interval is 30 ms which includes 10ms of advertising interval random delay).

Return BLE Mesh transmit value that can be used e.g. for the default values of the Configuration Model data.

Parameters

- `count`: Number of retransmissions (first transmission is excluded).
- `int_ms`: Interval steps in milliseconds. Must be greater than 0 and a multiple of 10.

ESP_BLE_MESH_GET_TRANSMIT_COUNT(transmit)

Decode transmit count from a transmit value.

Return Transmission count (actual transmissions equal to N + 1).

Parameters

- `transmit`: Encoded transmit count & interval value.

ESP_BLE_MESH_GET_TRANSMIT_INTERVAL(transmit)

Decode transmit interval from a transmit value.

Return Transmission interval in milliseconds.

Parameters

- `transmit`: Encoded transmit count & interval value.

ESP_BLE_MESH_PUBLISH_TRANSMIT(count, int_ms)

Encode Publish Retransmit count & interval steps.

Return BLE Mesh transmit value that can be used e.g. for the default values of the Configuration Model data.

Parameters

- `count`: Number of retransmissions (first transmission is excluded).
- `int_ms`: Interval steps in milliseconds. Must be greater than 0 and a multiple of 50.

`ESP_BLE_MESH_GET_PUBLISH_TRANSMIT_COUNT`(transmit)

Decode Publish Retransmit count from a given value.

Return Retransmission count (actual transmissions equal to N + 1).

Parameters

- `transmit`: Encoded Publish Retransmit count & interval value.

`ESP_BLE_MESH_GET_PUBLISH_TRANSMIT_INTERVAL`(transmit)

Decode Publish Retransmit interval from a given value.

Callbacks which are not needed to be initialized by users (set with 0 and will be initialized internally)

Return Transmission interval in milliseconds.

Parameters

- `transmit`: Encoded Publish Retransmit count & interval value.

`ESP_BLE_MESH_PROV_STATIC_OOB_MAX_LEN`

Maximum length of string used by Output OOB authentication

`ESP_BLE_MESH_PROV_OUTPUT_OOB_MAX_LEN`

Maximum length of string used by Output OOB authentication

`ESP_BLE_MESH_PROV_INPUT_OOB_MAX_LEN`

Macros used to define message opcode

`ESP_BLE_MESH_MODEL_OP_1`(b0)

`ESP_BLE_MESH_MODEL_OP_2`(b0, b1)

`ESP_BLE_MESH_MODEL_OP_3`(b0, cid)

This macro is associated with BLE_MESH_MODEL in mesh_access.h

`ESP_BLE_MESH_SIG_MODEL`(_id, _op, _pub, _user_data)

This macro is associated with BLE_MESH_MODEL_VND in mesh_access.h

`ESP_BLE_MESH_VENDOR_MODEL`(_company, _id, _op, _pub, _user_data)

`ESP_BLE_MESH_ELEMENT`(_loc, _mods, _vnd_mods)

Helper to define a BLE Mesh element within an array.

In case the element has no SIG or Vendor models, the helper macro ESP_BLE_MESH_MODEL_NONE can be given instead.

Note This macro is associated with BLE_MESH_ELEM in mesh_access.h

Parameters

- `_loc`: Location Descriptor.
- `_mods`: Array of SIG models.
- `_vnd_mods`: Array of vendor models.

`ESP_BLE_MESH_PROV`(uuid, sta_val, sta_val_len, out_size, out_act, in_size, in_act)

`BT_OCTET32_LEN`

`BD_ADDR_LEN`

`ESP_BLE_MESH_ADDR_TYPE_PUBLIC`

`ESP_BLE_MESH_ADDR_TYPE_RANDOM`

`ESP_BLE_MESH_ADDR_TYPE_RPA_PUBLIC`

`ESP_BLE_MESH_ADDR_TYPE_RPA_RANDOM`

`ESP_BLE_MESH_MODEL_PUB_DEFINE`(_name, _msg_len, _role)

Define a model publication context.

Parameters

- `_name`: Variable name given to the context.
- `_msg_len`: Length of the publication message.
- `_role`: Role of the device which contains the model.

`ESP_BLE_MESH_MODEL_OP`(_opcode, _min_len)

Define a model operation context.

Parameters

- `_opcode`: Message opcode.
- `_min_len`: Message minimum length.

`ESP_BLE_MESH_MODEL_OP_END`

Define the terminator for the model operation table. Each model operation struct array must use this terminator as the end tag of the operation unit.

`ESP_BLE_MESH_MODEL_NONE`

Helper to define an empty model array. This structure is associated with BLE_MESH_MODEL_NONE in mesh_access.h

`ADD_DEV_RM_AFTER_PROV_FLAG`

Device will be removed from queue after provisioned successfully

ADD_DEV_START_PROV_NOW_FLAG

Start provisioning device immediately

ADD_DEV_FLUSHABLE_DEV_FLAG

Device can be remove when queue is full and new device is going to added

DEL_DEV_ADDR_FLAG

DEL_DEV_UUID_FLAG

PROV_DATA_NET_IDX_FLAG

PROV_DATA_FLAGS_FLAG

PROV_DATA_IV_INDEX_FLAG

ESP_BLE_MESH_BLE_ADV_INFINITE

Count for sending BLE advertising packet infinitely This enum value is the priority of BLE advertising packet

ESP_BLE_MESH_MODEL_ID_CONFIG_SRV

BLE Mesh models related Model ID and Opcode definitions.

< Foundation Models

ESP_BLE_MESH_MODEL_ID_CONFIG_CLI

ESP_BLE_MESH_MODEL_ID_HEALTH_SRV

ESP_BLE_MESH_MODEL_ID_HEALTH_CLI

Models from the Mesh Model Specification

ESP_BLE_MESH_MODEL_ID_GEN_ONOFF_SRV

ESP_BLE_MESH_MODEL_ID_GEN_ONOFF_CLI

ESP_BLE_MESH_MODEL_ID_GEN_LEVEL_SRV

ESP_BLE_MESH_MODEL_ID_GEN_LEVEL_CLI

ESP_BLE_MESH_MODEL_ID_GEN_DEF_TRANS_TIME_SRV

ESP_BLE_MESH_MODEL_ID_GEN_DEF_TRANS_TIME_CLI

ESP_BLE_MESH_MODEL_ID_GEN_POWER_ONOFF_SRV

ESP_BLE_MESH_MODEL_ID_GEN_POWER_ONOFF_SETUP_SRV

ESP_BLE_MESH_MODEL_ID_GEN_POWER_ONOFF_CLI

ESP_BLE_MESH_MODEL_ID_GEN_POWER_LEVEL_SRV

ESP_BLE_MESH_MODEL_ID_GEN_POWER_LEVEL_SETUP_SRV

ESP_BLE_MESH_MODEL_ID_GEN_POWER_LEVEL_CLI

ESP_BLE_MESH_MODEL_ID_GEN_BATTERY_SRV

ESP_BLE_MESH_MODEL_ID_GEN_BATTERY_CLI

ESP_BLE_MESH_MODEL_ID_GEN_LOCATION_SRV

ESP_BLE_MESH_MODEL_ID_GEN_LOCATION_SETUP_SRV

ESP_BLE_MESH_MODEL_ID_GEN_LOCATION_CLI

ESP_BLE_MESH_MODEL_ID_GEN_ADMIN_PROP_SRV

ESP_BLE_MESH_MODEL_ID_GEN_MANUFACTURER_PROP_SRV

ESP_BLE_MESH_MODEL_ID_GEN_USER_PROP_SRV

ESP_BLE_MESH_MODEL_ID_GEN_CLIENT_PROP_SRV

ESP_BLE_MESH_MODEL_ID_GEN_PROP_CLI

ESP_BLE_MESH_MODEL_ID_SENSOR_SRV

ESP_BLE_MESH_MODEL_ID_SENSOR_SETUP_SRV

ESP_BLE_MESH_MODEL_ID_SENSOR_CLI

ESP_BLE_MESH_MODEL_ID_TIME_SRV

ESP_BLE_MESH_MODEL_ID_TIME_SETUP_SRV

ESP_BLE_MESH_MODEL_ID_TIME_CLI

ESP_BLE_MESH_MODEL_ID_SCENE_SRV

ESP_BLE_MESH_MODEL_ID_SCENE_SETUP_SRV

ESP_BLE_MESH_MODEL_ID_SCENE_CLI

ESP_BLE_MESH_MODEL_ID_SCHEDULER_SRV

ESP_BLE_MESH_MODEL_ID_SCHEDULER_SETUP_SRV

ESP_BLE_MESH_MODEL_ID_SCHEDULER_CLI

ESP_BLE_MESH_MODEL_ID_LIGHT_LIGHTNESS_SRV

ESP_BLE_MESH_MODEL_ID_LIGHT_LIGHTNESS_SETUP_SRV

ESP_BLE_MESH_MODEL_ID_LIGHT_LIGHTNESS_CLI

ESP_BLE_MESH_MODEL_ID_LIGHT_CTL_SRV

ESP_BLE_MESH_MODEL_ID_LIGHT_CTL_SETUP_SRV

ESP_BLE_MESH_MODEL_ID_LIGHT_CTL_CLI

ESP_BLE_MESH_MODEL_ID_LIGHT_CTL_TEMP_SRV

ESP_BLE_MESH_MODEL_ID_LIGHT_HSL_SRV

ESP_BLE_MESH_MODEL_ID_LIGHT_HSL_SETUP_SRV

ESP_BLE_MESH_MODEL_ID_LIGHT_HSL_CLI

ESP_BLE_MESH_MODEL_ID_LIGHT_HSL_HUE_SRV

ESP_BLE_MESH_MODEL_ID_LIGHT_HSL_SAT_SRV

ESP_BLE_MESH_MODEL_ID_LIGHT_XYL_SRV

ESP_BLE_MESH_MODEL_ID_LIGHT_XYL_SETUP_SRV

ESP_BLE_MESH_MODEL_ID_LIGHT_XYL_CLI

ESP_BLE_MESH_MODEL_ID_LIGHT_LC_SRV

ESP_BLE_MESH_MODEL_ID_LIGHT_LC_SETUP_SRV

ESP_BLE_MESH_MODEL_ID_LIGHT_LC_CLI

ESP_BLE_MESH_MODEL_OP_BEACON_GET
: Config Beacon Get

ESP_BLE_MESH_MODEL_OP_COMPOSITION_DATA_GET
: Config Composition Data Get

ESP_BLE_MESH_MODEL_OP_DEFAULT_TTL_GET
: Config Default TTL Get

ESP_BLE_MESH_MODEL_OP_GATT_PROXY_GET
: Config GATT Proxy Get

ESP_BLE_MESH_MODEL_OP_RELAY_GET
: Config Relay Get

ESP_BLE_MESH_MODEL_OP_MODEL_PUB_GET
: Config Model Publication Get

ESP_BLE_MESH_MODEL_OP_FRIEND_GET
: Config Friend Get

ESP_BLE_MESH_MODEL_OP_HEARTBEAT_PUB_GET
: Config Heartbeat Publication Get

ESP_BLE_MESH_MODEL_OP_HEARTBEAT_SUB_GET
: Config Heartbeat Subscription Get

ESP_BLE_MESH_MODEL_OP_NET_KEY_GET
: Config NetKey Get

ESP_BLE_MESH_MODEL_OP_APP_KEY_GET
: Config AppKey Get

ESP_BLE_MESH_MODEL_OP_NODE_IDENTITY_GET
: Config Node Identity Get

ESP_BLE_MESH_MODEL_OP_SIG_MODEL_SUB_GET
: Config SIG Model Subscription Get

ESP_BLE_MESH_MODEL_OP_VENDOR_MODEL_SUB_GET
: Config Vendor Model Subscription Get

ESP_BLE_MESH_MODEL_OP_SIG_MODEL_APP_GET
: Config SIG Model App Get

ESP_BLE_MESH_MODEL_OP_VENDOR_MODEL_APP_GET
: Config Vendor Model App Get

ESP_BLE_MESH_MODEL_OP_KEY_REFRESH_PHASE_GET
: Config Key Refresh Phase Get

ESP_BLE_MESH_MODEL_OP_LPN_POLLTIMEOUT_GET
: Config Low Power Node PollTimeout Get

ESP_BLE_MESH_MODEL_OP_NETWORK_TRANSMIT_GET
: Config Network Transmit Get

ESP_BLE_MESH_MODEL_OP_BEACON_SET
: Config Beacon Set

ESP_BLE_MESH_MODEL_OP_DEFAULT_TTL_SET
: Config Default TTL Set

ESP_BLE_MESH_MODEL_OP_GATT_PROXY_SET
: Config GATT Proxy Set

ESP_BLE_MESH_MODEL_OP_RELAY_SET
: Config Relay Set

ESP_BLE_MESH_MODEL_OP_MODEL_PUB_SET
: Config Model Publication Set

ESP_BLE_MESH_MODEL_OP_MODEL_SUB_ADD
: Config Model Subscription Add

ESP_BLE_MESH_MODEL_OP_MODEL_SUB_VIRTUAL_ADDR_ADD
: Config Model Subscription Virtual Address Add

ESP_BLE_MESH_MODEL_OP_MODEL_SUB_DELETE
: Config Model Subscription Delete

ESP_BLE_MESH_MODEL_OP_MODEL_SUB_VIRTUAL_ADDR_DELETE
: Config Model Subscription Virtual Address Delete

ESP_BLE_MESH_MODEL_OP_MODEL_SUB_OVERWRITE
: Config Model Subscription Overwrite

ESP_BLE_MESH_MODEL_OP_MODEL_SUB_VIRTUAL_ADDR_OVERWRITE
Config Model Subscription Virtual Address Overwrite

ESP_BLE_MESH_MODEL_OP_NET_KEY_ADD
Config NetKey Add

ESP_BLE_MESH_MODEL_OP_APP_KEY_ADD
Config AppKey Add

ESP_BLE_MESH_MODEL_OP_MODEL_APP_BIND
Config Model App Bind

ESP_BLE_MESH_MODEL_OP_NODE_RESET
Config Node Reset

ESP_BLE_MESH_MODEL_OP_FRIEND_SET
Config Friend Set

ESP_BLE_MESH_MODEL_OP_HEARTBEAT_PUB_SET
Config Heartbeat Publication Set

ESP_BLE_MESH_MODEL_OP_HEARTBEAT_SUB_SET
Config Heartbeat Subscription Set

ESP_BLE_MESH_MODEL_OP_NET_KEY_UPDATE
Config NetKey Update

ESP_BLE_MESH_MODEL_OP_NET_KEY_DELETE
Config NetKey Delete

ESP_BLE_MESH_MODEL_OP_APP_KEY_UPDATE
Config AppKey Update

ESP_BLE_MESH_MODEL_OP_APP_KEY_DELETE
Config AppKey Delete

ESP_BLE_MESH_MODEL_OP_NODE_IDENTITY_SET
Config Node Identity Set

ESP_BLE_MESH_MODEL_OP_KEY_REFRESH_PHASE_SET
Config Key Refresh Phase Set

ESP_BLE_MESH_MODEL_OP_MODEL_PUB_VIRTUAL_ADDR_SET
Config Model Publication Virtual Address Set

ESP_BLE_MESH_MODEL_OP_MODEL_SUB_DELETE_ALL
Config Model Subscription Delete All

ESP_BLE_MESH_MODEL_OP_MODEL_APP_UNBIND
Config Model App Unbind

ESP_BLE_MESH_MODEL_OP_NETWORK_TRANSMIT_SET
Config Network Transmit Set

ESP_BLE_MESH_MODEL_OP_BEACON_STATUS

ESP_BLE_MESH_MODEL_OP_COMPOSITION_DATA_STATUS

ESP_BLE_MESH_MODEL_OP_DEFAULT_TTL_STATUS

ESP_BLE_MESH_MODEL_OP_GATT_PROXY_STATUS

ESP_BLE_MESH_MODEL_OP_RELAY_STATUS

ESP_BLE_MESH_MODEL_OP_MODEL_PUB_STATUS

ESP_BLE_MESH_MODEL_OP_MODEL_SUB_STATUS

ESP_BLE_MESH_MODEL_OP_SIG_MODEL_SUB_LIST

ESP_BLE_MESH_MODEL_OP_VENDOR_MODEL_SUB_LIST

ESP_BLE_MESH_MODEL_OP_NET_KEY_STATUS

ESP_BLE_MESH_MODEL_OP_NET_KEY_LIST

ESP_BLE_MESH_MODEL_OP_APP_KEY_STATUS

ESP_BLE_MESH_MODEL_OP_APP_KEY_LIST

ESP_BLE_MESH_MODEL_OP_NODE_IDENTITY_STATUS

ESP_BLE_MESH_MODEL_OP_MODEL_APP_STATUS

ESP_BLE_MESH_MODEL_OP_SIG_MODEL_APP_LIST

ESP_BLE_MESH_MODEL_OP_VENDOR_MODEL_APP_LIST

ESP_BLE_MESH_MODEL_OP_NODE_RESET_STATUS

ESP_BLE_MESH_MODEL_OP_FRIEND_STATUS

ESP_BLE_MESH_MODEL_OP_KEY_REFRESH_PHASE_STATUS

ESP_BLE_MESH_MODEL_OP_HEARTBEAT_PUB_STATUS

ESP_BLE_MESH_MODEL_OP_HEARTBEAT_SUB_STATUS

ESP_BLE_MESH_MODEL_OP_LPN_POLLTIMEOUT_STATUS

ESP_BLE_MESH_MODEL_OP_NETWORK_TRANSMIT_STATUS

ESP_BLE_MESH_CFG_STATUS_SUCCESS

ESP_BLE_MESH_CFG_STATUS_INVALID_ADDRESS

ESP_BLE_MESH_CFG_STATUS_INVALID_MODEL

ESP_BLE_MESH_CFG_STATUS_INVALID_APPKEY

ESP_BLE_MESH_CFG_STATUS_INVALID_NETKEY

ESP_BLE_MESH_CFG_STATUS_INSUFFICIENT_RESOURCES

ESP_BLE_MESH_CFG_STATUS_KEY_INDEX_ALREADY_STORED

ESP_BLE_MESH_CFG_STATUS_INVALID_PUBLISH_PARAMETERS

ESP_BLE_MESH_CFG_STATUS_NOT_A_SUBSCRIBE_MODEL

ESP_BLE_MESH_CFG_STATUS_STORAGE_FAILURE

ESP_BLE_MESH_CFG_STATUS_FEATURE_NOT_SUPPORTED

ESP_BLE_MESH_CFG_STATUS_CANNOT_UPDATE

ESP_BLE_MESH_CFG_STATUS_CANNOT_REMOVE

ESP_BLE_MESH_CFG_STATUS_CANNOT_BIND

ESP_BLE_MESH_CFG_STATUS_TEMP_UNABLE_TO_CHANGE_STATE

ESP_BLE_MESH_CFG_STATUS_CANNOT_SET

ESP_BLE_MESH_CFG_STATUS_UNSPECIFIED_ERROR

ESP_BLE_MESH_CFG_STATUS_INVALID_BINDING

ESP_BLE_MESH_MODEL_OP_HEALTH_FAULT_GET
: Health Fault Get

ESP_BLE_MESH_MODEL_OP_HEALTH_PERIOD_GET
: Health Period Get

ESP_BLE_MESH_MODEL_OP_ATTENTION_GET
: Health Attention Get

ESP_BLE_MESH_MODEL_OP_HEALTH_FAULT_CLEAR
: Health Fault Clear

ESP_BLE_MESH_MODEL_OP_HEALTH_FAULT_CLEAR_UNACK
: Health Fault Clear Unacknowledged

ESP_BLE_MESH_MODEL_OP_HEALTH_FAULT_TEST
: Health Fault Test

ESP_BLE_MESH_MODEL_OP_HEALTH_FAULT_TEST_UNACK
: Health Fault Test Unacknowledged

ESP_BLE_MESH_MODEL_OP_HEALTH_PERIOD_SET
: Health Period Set

ESP_BLE_MESH_MODEL_OP_HEALTH_PERIOD_SET_UNACK
: Health Period Set Unacknowledged

ESP_BLE_MESH_MODEL_OP_ATTENTION_SET
Health Attention Set

ESP_BLE_MESH_MODEL_OP_ATTENTION_SET_UNACK
Health Attention Set Unacknowledged

ESP_BLE_MESH_MODEL_OP_HEALTH_CURRENT_STATUS

ESP_BLE_MESH_MODEL_OP_HEALTH_FAULT_STATUS

ESP_BLE_MESH_MODEL_OP_HEALTH_PERIOD_STATUS

ESP_BLE_MESH_MODEL_OP_ATTENTION_STATUS

ESP_BLE_MESH_MODEL_OP_GEN_ONOFF_GET

ESP_BLE_MESH_MODEL_OP_GEN_ONOFF_SET

ESP_BLE_MESH_MODEL_OP_GEN_ONOFF_SET_UNACK

ESP_BLE_MESH_MODEL_OP_GEN_ONOFF_STATUS
Generic Level Message Opcode

ESP_BLE_MESH_MODEL_OP_GEN_LEVEL_GET

ESP_BLE_MESH_MODEL_OP_GEN_LEVEL_SET

ESP_BLE_MESH_MODEL_OP_GEN_LEVEL_SET_UNACK

ESP_BLE_MESH_MODEL_OP_GEN_LEVEL_STATUS

ESP_BLE_MESH_MODEL_OP_GEN_DELTA_SET

ESP_BLE_MESH_MODEL_OP_GEN_DELTA_SET_UNACK

ESP_BLE_MESH_MODEL_OP_GEN_MOVE_SET

ESP_BLE_MESH_MODEL_OP_GEN_MOVE_SET_UNACK
Generic Default Transition Time Message Opcode

ESP_BLE_MESH_MODEL_OP_GEN_DEF_TRANS_TIME_GET

ESP_BLE_MESH_MODEL_OP_GEN_DEF_TRANS_TIME_SET

ESP_BLE_MESH_MODEL_OP_GEN_DEF_TRANS_TIME_SET_UNACK

ESP_BLE_MESH_MODEL_OP_GEN_DEF_TRANS_TIME_STATUS
Generic Power OnOff Message Opcode

ESP_BLE_MESH_MODEL_OP_GEN_ONPOWERUP_GET

ESP_BLE_MESH_MODEL_OP_GEN_ONPOWERUP_STATUS
Generic Power OnOff Setup Message Opcode

ESP_BLE_MESH_MODEL_OP_GEN_ONPOWERUP_SET

ESP_BLE_MESH_MODEL_OP_GEN_ONPOWERUP_SET_UNACK
Generic Power Level Message Opcode

ESP_BLE_MESH_MODEL_OP_GEN_POWER_LEVEL_GET

ESP_BLE_MESH_MODEL_OP_GEN_POWER_LEVEL_SET

ESP_BLE_MESH_MODEL_OP_GEN_POWER_LEVEL_SET_UNACK

ESP_BLE_MESH_MODEL_OP_GEN_POWER_LEVEL_STATUS

ESP_BLE_MESH_MODEL_OP_GEN_POWER_LAST_GET

ESP_BLE_MESH_MODEL_OP_GEN_POWER_LAST_STATUS

ESP_BLE_MESH_MODEL_OP_GEN_POWER_DEFAULT_GET

ESP_BLE_MESH_MODEL_OP_GEN_POWER_DEFAULT_STATUS

ESP_BLE_MESH_MODEL_OP_GEN_POWER_RANGE_GET

ESP_BLE_MESH_MODEL_OP_GEN_POWER_RANGE_STATUS
Generic Power Level Setup Message Opcode

ESP_BLE_MESH_MODEL_OP_GEN_POWER_DEFAULT_SET

ESP_BLE_MESH_MODEL_OP_GEN_POWER_DEFAULT_SET_UNACK

ESP_BLE_MESH_MODEL_OP_GEN_POWER_RANGE_SET

ESP_BLE_MESH_MODEL_OP_GEN_POWER_RANGE_SET_UNACK
Generic Battery Message Opcode

ESP_BLE_MESH_MODEL_OP_GEN_BATTERY_GET

ESP_BLE_MESH_MODEL_OP_GEN_BATTERY_STATUS
Generic Location Message Opcode

ESP_BLE_MESH_MODEL_OP_GEN_LOC_GLOBAL_GET

ESP_BLE_MESH_MODEL_OP_GEN_LOC_GLOBAL_STATUS

ESP_BLE_MESH_MODEL_OP_GEN_LOC_LOCAL_GET

ESP_BLE_MESH_MODEL_OP_GEN_LOC_LOCAL_STATUS
Generic Location Setup Message Opcode

ESP_BLE_MESH_MODEL_OP_GEN_LOC_GLOBAL_SET

ESP_BLE_MESH_MODEL_OP_GEN_LOC_GLOBAL_SET_UNACK

ESP_BLE_MESH_MODEL_OP_GEN_LOC_LOCAL_SET

ESP_BLE_MESH_MODEL_OP_GEN_LOC_LOCAL_SET_UNACK
Generic Manufacturer Property Message Opcode

ESP_BLE_MESH_MODEL_OP_GEN_MANUFACTURER_PROPERTIES_GET

ESP_BLE_MESH_MODEL_OP_GEN_MANUFACTURER_PROPERTIES_STATUS

ESP_BLE_MESH_MODEL_OP_GEN_MANUFACTURER_PROPERTY_GET

ESP_BLE_MESH_MODEL_OP_GEN_MANUFACTURER_PROPERTY_SET

ESP_BLE_MESH_MODEL_OP_GEN_MANUFACTURER_PROPERTY_SET_UNACK

ESP_BLE_MESH_MODEL_OP_GEN_MANUFACTURER_PROPERTY_STATUS
Generic Admin Property Message Opcode

ESP_BLE_MESH_MODEL_OP_GEN_ADMIN_PROPERTIES_GET

ESP_BLE_MESH_MODEL_OP_GEN_ADMIN_PROPERTIES_STATUS

ESP_BLE_MESH_MODEL_OP_GEN_ADMIN_PROPERTY_GET

ESP_BLE_MESH_MODEL_OP_GEN_ADMIN_PROPERTY_SET

ESP_BLE_MESH_MODEL_OP_GEN_ADMIN_PROPERTY_SET_UNACK

ESP_BLE_MESH_MODEL_OP_GEN_ADMIN_PROPERTY_STATUS
Generic User Property Message Opcode

ESP_BLE_MESH_MODEL_OP_GEN_USER_PROPERTIES_GET

ESP_BLE_MESH_MODEL_OP_GEN_USER_PROPERTIES_STATUS

ESP_BLE_MESH_MODEL_OP_GEN_USER_PROPERTY_GET

ESP_BLE_MESH_MODEL_OP_GEN_USER_PROPERTY_SET

ESP_BLE_MESH_MODEL_OP_GEN_USER_PROPERTY_SET_UNACK

ESP_BLE_MESH_MODEL_OP_GEN_USER_PROPERTY_STATUS
Generic Client Property Message Opcode

ESP_BLE_MESH_MODEL_OP_GEN_CLIENT_PROPERTIES_GET

ESP_BLE_MESH_MODEL_OP_GEN_CLIENT_PROPERTIES_STATUS

ESP_BLE_MESH_MODEL_OP_SENSOR_DESCRIPTOR_GET

ESP_BLE_MESH_MODEL_OP_SENSOR_DESCRIPTOR_STATUS

ESP_BLE_MESH_MODEL_OP_SENSOR_GET

ESP_BLE_MESH_MODEL_OP_SENSOR_STATUS

ESP_BLE_MESH_MODEL_OP_SENSOR_COLUMN_GET

ESP_BLE_MESH_MODEL_OP_SENSOR_COLUMN_STATUS

ESP_BLE_MESH_MODEL_OP_SENSOR_SERIES_GET

ESP_BLE_MESH_MODEL_OP_SENSOR_SERIES_STATUS
Sensor Setup Message Opcode

ESP_BLE_MESH_MODEL_OP_SENSOR_CADENCE_GET

ESP_BLE_MESH_MODEL_OP_SENSOR_CADENCE_SET

ESP_BLE_MESH_MODEL_OP_SENSOR_CADENCE_SET_UNACK

ESP_BLE_MESH_MODEL_OP_SENSOR_CADENCE_STATUS

ESP_BLE_MESH_MODEL_OP_SENSOR_SETTINGS_GET

ESP_BLE_MESH_MODEL_OP_SENSOR_SETTINGS_STATUS

ESP_BLE_MESH_MODEL_OP_SENSOR_SETTING_GET

ESP_BLE_MESH_MODEL_OP_SENSOR_SETTING_SET

ESP_BLE_MESH_MODEL_OP_SENSOR_SETTING_SET_UNACK

ESP_BLE_MESH_MODEL_OP_SENSOR_SETTING_STATUS

ESP_BLE_MESH_MODEL_OP_TIME_GET

ESP_BLE_MESH_MODEL_OP_TIME_SET

ESP_BLE_MESH_MODEL_OP_TIME_STATUS

ESP_BLE_MESH_MODEL_OP_TIME_ROLE_GET

ESP_BLE_MESH_MODEL_OP_TIME_ROLE_SET

ESP_BLE_MESH_MODEL_OP_TIME_ROLE_STATUS

ESP_BLE_MESH_MODEL_OP_TIME_ZONE_GET

ESP_BLE_MESH_MODEL_OP_TIME_ZONE_SET

ESP_BLE_MESH_MODEL_OP_TIME_ZONE_STATUS

ESP_BLE_MESH_MODEL_OP_TAI_UTC_DELTA_GET

ESP_BLE_MESH_MODEL_OP_TAI_UTC_DELTA_SET

ESP_BLE_MESH_MODEL_OP_TAI_UTC_DELTA_STATUS

Scene Message Opcode

ESP_BLE_MESH_MODEL_OP_SCENE_GET

ESP_BLE_MESH_MODEL_OP_SCENE_RECALL

ESP_BLE_MESH_MODEL_OP_SCENE_RECALL_UNACK

ESP_BLE_MESH_MODEL_OP_SCENE_STATUS

ESP_BLE_MESH_MODEL_OP_SCENE_REGISTER_GET

ESP_BLE_MESH_MODEL_OP_SCENE_REGISTER_STATUS

Scene Setup Message Opcode

ESP_BLE_MESH_MODEL_OP_SCENE_STORE

ESP_BLE_MESH_MODEL_OP_SCENE_STORE_UNACK

ESP_BLE_MESH_MODEL_OP_SCENE_DELETE

ESP_BLE_MESH_MODEL_OP_SCENE_DELETE_UNACK

Scheduler Message Opcode

ESP_BLE_MESH_MODEL_OP_SCHEDULER_ACT_GET

ESP_BLE_MESH_MODEL_OP_SCHEDULER_ACT_STATUS

ESP_BLE_MESH_MODEL_OP_SCHEDULER_GET

ESP_BLE_MESH_MODEL_OP_SCHEDULER_STATUS

Scheduler Setup Message Opcode

ESP_BLE_MESH_MODEL_OP_SCHEDULER_ACT_SET

ESP_BLE_MESH_MODEL_OP_SCHEDULER_ACT_SET_UNACK

ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_GET

ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_SET

ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_SET_UNACK

ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_STATUS

ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_LINEAR_GET

ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_LINEAR_SET

ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_LINEAR_SET_UNACK

ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_LINEAR_STATUS

ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_LAST_GET

ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_LAST_STATUS

ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_DEFAULT_GET

ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_DEFAULT_STATUS

ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_RANGE_GET

ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_RANGE_STATUS

Light Lightness Setup Message Opcode

ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_DEFAULT_SET

ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_DEFAULT_SET_UNACK

ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_RANGE_SET

ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_RANGE_SET_UNACK

Light CTL Message Opcode

ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_GET

ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_SET

ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_SET_UNACK

ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_STATUS

ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_TEMPERATURE_GET

ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_TEMPERATURE_RANGE_GET

ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_TEMPERATURE_RANGE_STATUS

ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_TEMPERATURE_SET

ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_TEMPERATURE_SET_UNACK

ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_TEMPERATURE_STATUS

ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_DEFAULT_GET

ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_DEFAULT_STATUS
Light CTL Setup Message Opcode

ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_DEFAULT_SET

ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_DEFAULT_SET_UNACK

ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_TEMPERATURE_RANGE_SET

ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_TEMPERATURE_RANGE_SET_UNACK
Light HSL Message Opcode

ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_GET

ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_HUE_GET

ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_HUE_SET

ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_HUE_SET_UNACK

ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_HUE_STATUS

ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_SATURATION_GET

ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_SATURATION_SET

ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_SATURATION_SET_UNACK

ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_SATURATION_STATUS

ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_SET

ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_SET_UNACK

ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_STATUS

ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_TARGET_GET

ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_TARGET_STATUS

ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_DEFAULT_GET

ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_DEFAULT_STATUS

ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_RANGE_GET

ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_RANGE_STATUS
Light HSL Setup Message Opcode

ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_DEFAULT_SET

ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_DEFAULT_SET_UNACK

ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_RANGE_SET

ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_RANGE_SET_UNACK
Light xyL Message Opcode

ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_GET

ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_SET

ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_SET_UNACK

ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_STATUS

ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_TARGET_GET

ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_TARGET_STATUS

ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_DEFAULT_GET

ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_DEFAULT_STATUS

ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_RANGE_GET

ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_RANGE_STATUS
Light xyL Setup Message Opcode

ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_DEFAULT_SET

ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_DEFAULT_SET_UNACK

ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_RANGE_SET

ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_RANGE_SET_UNACK
Light Control Message Opcode

ESP_BLE_MESH_MODEL_OP_LIGHT_LC_MODE_GET

ESP_BLE_MESH_MODEL_OP_LIGHT_LC_MODE_SET

ESP_BLE_MESH_MODEL_OP_LIGHT_LC_MODE_SET_UNACK

ESP_BLE_MESH_MODEL_OP_LIGHT_LC_MODE_STATUS

ESP_BLE_MESH_MODEL_OP_LIGHT_LC_OM_GET

ESP_BLE_MESH_MODEL_OP_LIGHT_LC_OM_SET

ESP_BLE_MESH_MODEL_OP_LIGHT_LC_OM_SET_UNACK

ESP_BLE_MESH_MODEL_OP_LIGHT_LC_OM_STATUS

ESP_BLE_MESH_MODEL_OP_LIGHT_LC_LIGHT_ONOFF_GET

ESP_BLE_MESH_MODEL_OP_LIGHT_LC_LIGHT_ONOFF_SET

ESP_BLE_MESH_MODEL_OP_LIGHT_LC_LIGHT_ONOFF_SET_UNACK

ESP_BLE_MESH_MODEL_OP_LIGHT_LC_LIGHT_ONOFF_STATUS

ESP_BLE_MESH_MODEL_OP_LIGHT_LC_PROPERTY_GET

ESP_BLE_MESH_MODEL_OP_LIGHT_LC_PROPERTY_SET

ESP_BLE_MESH_MODEL_OP_LIGHT_LC_PROPERTY_SET_UNACK

ESP_BLE_MESH_MODEL_OP_LIGHT_LC_PROPERTY_STATUS

ESP_BLE_MESH_MODEL_STATUS_SUCCESS

ESP_BLE_MESH_MODEL_STATUS_CANNOT_SET_RANGE_MIN

ESP_BLE_MESH_MODEL_STATUS_CANNOT_SET_RANGE_MAX

ESP_BLE_MESH_SERVER_RSP_BY_APP
: Response will be sent internally

ESP_BLE_MESH_SERVER_AUTO_RSP
: Response need to be sent in the application

Type Definitions

typedef uint8_t esp_ble_mesh_octet16_t[ESP_BLE_MESH_OCTET16_LEN]
: Define the BLE Mesh octet 8 bytes size

typedef uint8_t esp_ble_mesh_octet8_t[ESP_BLE_MESH_OCTET8_LEN]

typedef uint32_t esp_ble_mesh_cb_t

typedef uint8_t UINT8

typedef uint16_t UINT16

typedef uint32_t UINT32

typedef uint64_t UINT64

typedef *UINT8* BT_OCTET32[BT_OCTET32_LEN]

typedef uint8_t BD_ADDR[BD_ADDR_LEN]

typedef uint8_t **esp_ble_mesh_bd_addr_t**[**BD_ADDR_LEN**]

typedef uint8_t **esp_ble_mesh_addr_type_t**

BLE device address type.

typedef struct *esp_ble_mesh_model* **esp_ble_mesh_model_t**

typedef uint8_t **esp_ble_mesh_dev_add_flag_t**

typedef uint32_t **esp_ble_mesh_opcode_config_client_get_t**

esp_ble_mesh_opcode_config_client_get_t belongs to esp_ble_mesh_opcode_t, this typedef is only used to locate the opcodes used by esp_ble_mesh_config_client_get_state. The following opcodes will only be used in the esp_ble_mesh_config_client_get_state function.

typedef uint32_t **esp_ble_mesh_opcode_config_client_set_t**

esp_ble_mesh_opcode_config_client_set_t belongs to esp_ble_mesh_opcode_t, this typedef is only used to locate the opcodes used by esp_ble_mesh_config_client_set_state. The following opcodes will only be used in the esp_ble_mesh_config_client_set_state function.

typedef uint32_t **esp_ble_mesh_opcode_config_status_t**

esp_ble_mesh_opcode_config_status_t belongs to esp_ble_mesh_opcode_t, this typedef is only used to locate the opcodes used by the Config Model messages The following opcodes are used by the BLE Mesh Config Server Model internally to respond to the Config Client Model' s request messages.

typedef uint8_t **esp_ble_mesh_cfg_status_t**

This typedef is only used to indicate the status code contained in some of the Configuration Server Model status message.

typedef uint32_t **esp_ble_mesh_opcode_health_client_get_t**

esp_ble_mesh_opcode_health_client_get_t belongs to esp_ble_mesh_opcode_t, this typedef is only used to locate the opcodes used by esp_ble_mesh_health_client_get_state. The following opcodes will only be used in the esp_ble_mesh_health_client_get_state function.

typedef uint32_t **esp_ble_mesh_opcode_health_client_set_t**

esp_ble_mesh_opcode_health_client_set_t belongs to esp_ble_mesh_opcode_t, this typedef is only used to locate the opcodes used by esp_ble_mesh_health_client_set_state. The following opcodes will only be used in the esp_ble_mesh_health_client_set_state function.

typedef uint32_t **esp_ble_mesh_health_model_status_t**

esp_ble_mesh_health_model_status_t belongs to esp_ble_mesh_opcode_t, this typedef is only used to locate the opcodes used by the Health Model messages. The following opcodes are used by the BLE Mesh Health Server Model internally to respond to the Health Client Model' s request messages.

typedef uint32_t **esp_ble_mesh_generic_message_opcode_t**

esp_ble_mesh_generic_message_opcode_t belongs to esp_ble_mesh_opcode_t, this typedef is only used to locate the opcodes used by functions esp_ble_mesh_generic_client_get_state & esp_ble_mesh_generic_client_set_state.Generic OnOff Message Opcode

typedef uint32_t **esp_ble_mesh_sensor_message_opcode_t**

esp_ble_mesh_sensor_message_opcode_t belongs to esp_ble_mesh_opcode_t, this typedef is only used to locate the opcodes used by functions esp_ble_mesh_sensor_client_get_state & esp_ble_mesh_sensor_client_set_state.Sensor Message Opcode

typedef uint32_t **esp_ble_mesh_time_scene_message_opcode_t**

esp_ble_mesh_time_scene_message_opcode_t belongs to esp_ble_mesh_opcode_t, this typedef is only used to locate the opcodes used by functions esp_ble_mesh_time_scene_client_get_state & esp_ble_mesh_time_scene_client_set_state.Time Message Opcode

typedef uint32_t **esp_ble_mesh_light_message_opcode_t**

esp_ble_mesh_light_message_opcode_t belongs to esp_ble_mesh_opcode_t, this typedef is only used to locate the opcodes used by functions esp_ble_mesh_light_client_get_state & esp_ble_mesh_light_client_set_state.Light Lightness Message Opcode

typedef uint32_t **esp_ble_mesh_opcode_t**

End of defines of esp_ble_mesh_opcode_t

typedef uint8_t **esp_ble_mesh_model_status_t**

This typedef is only used to indicate the status code contained in some of the server models (e.g. Generic Server Model) status message.

Enumerations

enum esp_ble_mesh_cb_type_t

Values:

ESP_BLE_MESH_TYPE_PROV_CB

ESP_BLE_MESH_TYPE_OUTPUT_NUM_CB

ESP_BLE_MESH_TYPE_OUTPUT_STR_CB

ESP_BLE_MESH_TYPE_INTPUT_CB

ESP_BLE_MESH_TYPE_LINK_OPEN_CB

ESP_BLE_MESH_TYPE_LINK_CLOSE_CB

ESP_BLE_MESH_TYPE_COMPLETE_CB

ESP_BLE_MESH_TYPE_RESET_CB

enum esp_ble_mesh_oob_method_t

Values:

ESP_BLE_MESH_NO_OOB

ESP_BLE_MESH_STATIC_OOB

ESP_BLE_MESH_OUTPUT_OOB

ESP_BLE_MESH_INPUT_OOB

enum esp_ble_mesh_output_action_t

Values:

ESP_BLE_MESH_NO_OUTPUT = 0

ESP_BLE_MESH_BLINK = BIT(0)

ESP_BLE_MESH_BEEP = BIT(1)

ESP_BLE_MESH_VIBRATE = BIT(2)

ESP_BLE_MESH_DISPLAY_NUMBER = BIT(3)

ESP_BLE_MESH_DISPLAY_STRING = BIT(4)

enum esp_ble_mesh_input_action_t

Values:

ESP_BLE_MESH_NO_INPUT = 0

ESP_BLE_MESH_PUSH = BIT(0)

ESP_BLE_MESH_TWIST = BIT(1)

ESP_BLE_MESH_ENTER_NUMBER = BIT(2)

ESP_BLE_MESH_ENTER_STRING = BIT(3)

enum esp_ble_mesh_prov_bearer_t

Values:

ESP_BLE_MESH_PROV_ADV = BIT(0)

ESP_BLE_MESH_PROV_GATT = BIT(1)

enum esp_ble_mesh_prov_oob_info_t

Values:

ESP_BLE_MESH_PROV_OOB_OTHER = BIT(0)

ESP_BLE_MESH_PROV_OOB_URI = BIT(1)

ESP_BLE_MESH_PROV_OOB_2D_CODE = BIT(2)

ESP_BLE_MESH_PROV_OOB_BAR_CODE = BIT(3)

ESP_BLE_MESH_PROV_OOB_NFC = BIT(4)

ESP_BLE_MESH_PROV_OOB_NUMBER = BIT(5)

ESP_BLE_MESH_PROV_OOB_STRING = BIT(6)

ESP_BLE_MESH_PROV_OOB_ON_BOX = BIT(11)

ESP_BLE_MESH_PROV_OOB_IN_BOX = BIT(12)

ESP_BLE_MESH_PROV_OOB_ON_PAPER = BIT(13)

ESP_BLE_MESH_PROV_OOB_IN_MANUAL = BIT(14)

ESP_BLE_MESH_PROV_OOB_ON_DEV = BIT(15)

enum esp_ble_mesh_dev_role_t

Values:

ROLE_NODE = 0

ROLE_PROVISIONER

ROLE_FAST_PROV

enum esp_ble_mesh_fast_prov_action_t

Values:

FAST_PROV_ACT_NONE

FAST_PROV_ACT_ENTER

FAST_PROV_ACT_SUSPEND

FAST_PROV_ACT_EXIT

FAST_PROV_ACT_MAX

enum esp_ble_mesh_proxy_filter_type_t

Values:

PROXY_FILTER_WHITELIST

PROXY_FILTER_BLACKLIST

enum esp_ble_mesh_ble_adv_priority_t

Values:

ESP_BLE_MESH_BLE_ADV_PRIO_LOW

ESP_BLE_MESH_BLE_ADV_PRIO_HIGH

enum esp_ble_mesh_prov_cb_event_t

Values:

ESP_BLE_MESH_PROV_REGISTER_COMP_EVT

Initialize BLE Mesh provisioning capabilities and internal data information completion event

ESP_BLE_MESH_NODE_SET_UNPROV_DEV_NAME_COMP_EVT

Set the unprovisioned device name completion event

ESP_BLE_MESH_NODE_PROV_ENABLE_COMP_EVT

Enable node provisioning functionality completion event

ESP_BLE_MESH_NODE_PROV_DISABLE_COMP_EVT

Disable node provisioning functionality completion event

ESP_BLE_MESH_NODE_PROV_LINK_OPEN_EVT
: Establish a BLE Mesh link event

ESP_BLE_MESH_NODE_PROV_LINK_CLOSE_EVT
: Close a BLE Mesh link event

ESP_BLE_MESH_NODE_PROV_OOB_PUB_KEY_EVT
: Generate Node input OOB public key event

ESP_BLE_MESH_NODE_PROV_OUTPUT_NUMBER_EVT
: Generate Node Output Number event

ESP_BLE_MESH_NODE_PROV_OUTPUT_STRING_EVT
: Generate Node Output String event

ESP_BLE_MESH_NODE_PROV_INPUT_EVT
: Event requiring the user to input a number or string

ESP_BLE_MESH_NODE_PROV_COMPLETE_EVT
: Provisioning done event

ESP_BLE_MESH_NODE_PROV_RESET_EVT
: Provisioning reset event

ESP_BLE_MESH_NODE_PROV_SET_OOB_PUB_KEY_COMP_EVT
: Node set oob public key completion event

ESP_BLE_MESH_NODE_PROV_INPUT_NUMBER_COMP_EVT
: Node input number completion event

ESP_BLE_MESH_NODE_PROV_INPUT_STRING_COMP_EVT
: Node input string completion event

ESP_BLE_MESH_NODE_PROXY_IDENTITY_ENABLE_COMP_EVT
: Enable BLE Mesh Proxy Identity advertising completion event

ESP_BLE_MESH_NODE_PROXY_GATT_ENABLE_COMP_EVT
: Enable BLE Mesh GATT Proxy Service completion event

ESP_BLE_MESH_NODE_PROXY_GATT_DISABLE_COMP_EVT
: Disable BLE Mesh GATT Proxy Service completion event

ESP_BLE_MESH_PROVISIONER_PROV_ENABLE_COMP_EVT
: Provisioner enable provisioning functionality completion event

ESP_BLE_MESH_PROVISIONER_PROV_DISABLE_COMP_EVT
: Provisioner disable provisioning functionality completion event

ESP_BLE_MESH_PROVISIONER_RECV_UNPROV_ADV_PKT_EVT
: Provisioner receives unprovisioned device beacon event

ESP_BLE_MESH_PROVISIONER_PROV_READ_OOB_PUB_KEY_EVT
: Provisioner read unprovisioned device OOB public key event

ESP_BLE_MESH_PROVISIONER_PROV_INPUT_EVT
: Provisioner input value for provisioning procedure event

ESP_BLE_MESH_PROVISIONER_PROV_OUTPUT_EVT
: Provisioner output value for provisioning procedure event

ESP_BLE_MESH_PROVISIONER_PROV_LINK_OPEN_EVT
: Provisioner establish a BLE Mesh link event

ESP_BLE_MESH_PROVISIONER_PROV_LINK_CLOSE_EVT
: Provisioner close a BLE Mesh link event

ESP_BLE_MESH_PROVISIONER_PROV_COMPLETE_EVT
: Provisioner provisioning done event

ESP_BLE_MESH_PROVISIONER_ADD_UNPROV_DEV_COMP_EVT
: Provisioner add a device to the list which contains devices that are waiting/going to be provisioned completion event

ESP_BLE_MESH_PROVISIONER_PROV_DEV_WITH_ADDR_COMP_EVT
: Provisioner start to provision an unprovisioned device completion event

ESP_BLE_MESH_PROVISIONER_DELETE_DEV_COMP_EVT
: Provisioner delete a device from the list, close provisioning link with the device if it exists and remove the device from network completion event

ESP_BLE_MESH_PROVISIONER_SET_DEV_UUID_MATCH_COMP_EVT
: Provisioner set the value to be compared with part of the unprovisioned device UUID completion event

ESP_BLE_MESH_PROVISIONER_SET_PROV_DATA_INFO_COMP_EVT
: Provisioner set net_idx/flags/iv_index used for provisioning completion event

ESP_BLE_MESH_PROVISIONER_SET_STATIC_OOB_VALUE_COMP_EVT
: Provisioner set static oob value used for provisioning completion event

ESP_BLE_MESH_PROVISIONER_SET_PRIMARY_ELEM_ADDR_COMP_EVT
: Provisioner set unicast address of primary element completion event

ESP_BLE_MESH_PROVISIONER_PROV_READ_OOB_PUB_KEY_COMP_EVT
: Provisioner read unprovisioned device OOB public key completion event

ESP_BLE_MESH_PROVISIONER_PROV_INPUT_NUMBER_COMP_EVT
: Provisioner input number completion event

ESP_BLE_MESH_PROVISIONER_PROV_INPUT_STRING_COMP_EVT
: Provisioner input string completion event

ESP_BLE_MESH_PROVISIONER_SET_NODE_NAME_COMP_EVT
: Provisioner set node name completion event

ESP_BLE_MESH_PROVISIONER_ADD_LOCAL_APP_KEY_COMP_EVT
: Provisioner add local app key completion event

ESP_BLE_MESH_PROVISIONER_UPDATE_LOCAL_APP_KEY_COMP_EVT
: Provisioner update local app key completion event

ESP_BLE_MESH_PROVISIONER_BIND_APP_KEY_TO_MODEL_COMP_EVT
: Provisioner bind local model with local app key completion event

ESP_BLE_MESH_PROVISIONER_ADD_LOCAL_NET_KEY_COMP_EVT
: Provisioner add local network key completion event

ESP_BLE_MESH_PROVISIONER_UPDATE_LOCAL_NET_KEY_COMP_EVT
: Provisioner update local network key completion event

ESP_BLE_MESH_PROVISIONER_STORE_NODE_COMP_DATA_COMP_EVT
: Provisioner store node composition data completion event

ESP_BLE_MESH_PROVISIONER_DELETE_NODE_WITH_UUID_COMP_EVT
: Provisioner delete node with uuid completion event

ESP_BLE_MESH_PROVISIONER_DELETE_NODE_WITH_ADDR_COMP_EVT
: Provisioner delete node with unicast address completion event

ESP_BLE_MESH_SET_FAST_PROV_INFO_COMP_EVT
: Set fast provisioning information (e.g. unicast address range, net_idx, etc.) completion event

ESP_BLE_MESH_SET_FAST_PROV_ACTION_COMP_EVT
: Set fast provisioning action completion event

ESP_BLE_MESH_HEARTBEAT_MESSAGE_RECV_EVT
: Receive Heartbeat message event

ESP_BLE_MESH_LPN_ENABLE_COMP_EVT
: Enable Low Power Node completion event

ESP_BLE_MESH_LPN_DISABLE_COMP_EVT
: Disable Low Power Node completion event

ESP_BLE_MESH_LPN_POLL_COMP_EVT
: Low Power Node send Friend Poll completion event

ESP_BLE_MESH_LPN_FRIENDSHIP_ESTABLISH_EVT
: Low Power Node establishes friendship event

ESP_BLE_MESH_LPN_FRIENDSHIP_TERMINATE_EVT
: Low Power Node terminates friendship event

ESP_BLE_MESH_FRIEND_FRIENDSHIP_ESTABLISH_EVT
Friend Node establishes friendship event

ESP_BLE_MESH_FRIEND_FRIENDSHIP_TERMINATE_EVT
Friend Node terminates friendship event

ESP_BLE_MESH_PROXY_CLIENT_RECV_ADV_PKT_EVT
Proxy Client receives Network ID advertising packet event

ESP_BLE_MESH_PROXY_CLIENT_CONNECTED_EVT
Proxy Client establishes connection successfully event

ESP_BLE_MESH_PROXY_CLIENT_DISCONNECTED_EVT
Proxy Client terminates connection successfully event

ESP_BLE_MESH_PROXY_CLIENT_RECV_FILTER_STATUS_EVT
Proxy Client receives Proxy Filter Status event

ESP_BLE_MESH_PROXY_CLIENT_CONNECT_COMP_EVT
Proxy Client connect completion event

ESP_BLE_MESH_PROXY_CLIENT_DISCONNECT_COMP_EVT
Proxy Client disconnect completion event

ESP_BLE_MESH_PROXY_CLIENT_SET_FILTER_TYPE_COMP_EVT
Proxy Client set filter type completion event

ESP_BLE_MESH_PROXY_CLIENT_ADD_FILTER_ADDR_COMP_EVT
Proxy Client add filter address completion event

ESP_BLE_MESH_PROXY_CLIENT_REMOVE_FILTER_ADDR_COMP_EVT
Proxy Client remove filter address completion event

ESP_BLE_MESH_START_BLE_ADVERTISING_COMP_EVT
Start BLE advertising completion event

ESP_BLE_MESH_STOP_BLE_ADVERTISING_COMP_EVT
Stop BLE advertising completion event

ESP_BLE_MESH_DEINIT_MESH_COMP_EVT
De-initialize BLE Mesh stack completion event

ESP_BLE_MESH_PROV_EVT_MAX

enum [anonymous]
BLE Mesh server models related definitions.

This enum value is the flag of transition timer operation

Values:

ESP_BLE_MESH_SERVER_TRANS_TIMER_START

ESP_BLE_MESH_SERVER_FLAG_MAX

enum esp_ble_mesh_server_state_type_t

This enum value is the type of server model states

Values:

ESP_BLE_MESH_GENERIC_ONOFF_STATE

ESP_BLE_MESH_GENERIC_LEVEL_STATE

ESP_BLE_MESH_GENERIC_ONPOWERUP_STATE

ESP_BLE_MESH_GENERIC_POWER_ACTUAL_STATE

ESP_BLE_MESH_LIGHT_LIGHTNESS_ACTUAL_STATE

ESP_BLE_MESH_LIGHT_LIGHTNESS_LINEAR_STATE

ESP_BLE_MESH_LIGHT_CTL_LIGHTNESS_STATE

ESP_BLE_MESH_LIGHT_CTL_TEMP_DELTA_UV_STATE

ESP_BLE_MESH_LIGHT_HSL_LIGHTNESS_STATE

ESP_BLE_MESH_LIGHT_HSL_HUE_STATE

ESP_BLE_MESH_LIGHT_HSL_SATURATION_STATE

ESP_BLE_MESH_LIGHT_XYL_LIGHTNESS_STATE

ESP_BLE_MESH_LIGHT_LC_LIGHT_ONOFF_STATE

ESP_BLE_MESH_SERVER_MODEL_STATE_MAX

enum esp_ble_mesh_model_cb_event_t

Values:

ESP_BLE_MESH_MODEL_OPERATION_EVT

User-defined models receive messages from peer devices (e.g. get, set, status, etc) event

ESP_BLE_MESH_MODEL_SEND_COMP_EVT

User-defined models send messages completion event

ESP_BLE_MESH_MODEL_PUBLISH_COMP_EVT

User-defined models publish messages completion event

ESP_BLE_MESH_CLIENT_MODEL_RECV_PUBLISH_MSG_EVT

User-defined client models receive publish messages event

ESP_BLE_MESH_CLIENT_MODEL_SEND_TIMEOUT_EVT

Timeout event for the user-defined client models that failed to receive response from peer server models

`ESP_BLE_MESH_MODEL_PUBLISH_UPDATE_EVT`
When a model is configured to publish messages periodically, this event will occur during every publish period

`ESP_BLE_MESH_SERVER_MODEL_UPDATE_STATE_COMP_EVT`
Server models update state value completion event

`ESP_BLE_MESH_MODEL_EVT_MAX`

ESP-BLE-MESH Core API Reference

This section contains ESP-BLE-MESH Core related APIs, which can be used to initialize ESP-BLE-MESH stack, provision, send/publish messages, etc.

This API reference covers six components:

- *ESP-BLE-MESH Stack Initialization*
- *Reading of Local Data Information*
- *Low Power Operation (Updating)*
- *Send/Publish Messages, add Local AppKey, etc.*
- *ESP-BLE-MESH Node/Provisioner Provisioning*
- *ESP-BLE-MESH GATT Proxy Server*

ESP-BLE-MESH Stack Initialization

Header File

- bt/esp_ble_mesh/api/core/include/esp_ble_mesh_common_api.h

Functions

esp_err_t **esp_ble_mesh_init**(*esp_ble_mesh_prov_t* *prov, *esp_ble_mesh_comp_t* *comp)
Initialize BLE Mesh module. This API initializes provisioning capabilities and composition data information.

Note After calling this API, the device needs to call esp_ble_mesh_prov_enable() to enable provisioning functionality again.

Return ESP_OK on success or error code otherwise.

Parameters

- `prov`: Pointer to the device provisioning capabilities. This pointer must remain valid during the lifetime of the BLE Mesh device.

- `comp`: Pointer to the device composition data information. This pointer must remain valid during the lifetime of the BLE Mesh device.

esp_err_t **esp_ble_mesh_deinit**(*esp_ble_mesh_deinit_param_t* **param*)

De-initialize BLE Mesh module.

Note This function shall be invoked after esp_ble_mesh_client_model_deinit().

Return ESP_OK on success or error code otherwise.

Parameters

- `param`: Pointer to the structure of BLE Mesh deinit parameters.

Reading of Local Data Information

Header File

- bt/esp_ble_mesh/api/core/include/esp_ble_mesh_local_data_operation_api.h

Functions

int32_t **esp_ble_mesh_get_model_publish_period**(*esp_ble_mesh_model_t* **model*)

Get the model publish period, the unit is ms.

Return Publish period value on success, 0 or (negative) error code from errno.h on failure.

Parameters

- `model`: Model instance pointer.

uint16_t **esp_ble_mesh_get_primary_element_address**(void)

Get the address of the primary element.

Return Address of the primary element on success, or ESP_BLE_MESH_ADDR_UNASSIGNED on failure which means the device has not been provisioned.

uint16_t ***esp_ble_mesh_is_model_subscribed_to_group**(*esp_ble_mesh_model_t* **model*, uint16_t *group_addr*)

Check if the model has subscribed to the given group address. Note: E.g., once a status message is received and the destination address is a group address, the model uses this API to check if it is successfully subscribed to the given group address.

Return Pointer to the group address within the Subscription List of the model on success, or NULL on failure which means the model has not subscribed to the given group address. Note: With

the pointer to the group address returned, you can reset the group address to 0x0000 in order to unsubscribe the model from the group.

Parameters

- `model`: Pointer to the model.
- `group_addr`: Group address.

esp_ble_mesh_elem_t ***esp_ble_mesh_find_element**(uint16_t *element_addr*)

Find the BLE Mesh element pointer via the element address.

Return Pointer to the element on success, or NULL on failure.

Parameters

- `element_addr`: Element address.

uint8_t **esp_ble_mesh_get_element_count**(void)

Get the number of elements that have been registered.

Return Number of elements.

esp_ble_mesh_model_t ***esp_ble_mesh_find_vendor_model**(const *esp_ble_mesh_elem_t* **element*, uint16_t *company_id*, uint16_t *model_id*)

Find the Vendor specific model with the given element, the company ID and the Vendor Model ID.

Return Pointer to the Vendor Model on success, or NULL on failure which means the Vendor Model is not found.

Parameters

- `element`: Element to which the model belongs.
- `company_id`: A 16-bit company identifier assigned by the Bluetooth SIG.
- `model_id`: A 16-bit vendor-assigned model identifier.

esp_ble_mesh_model_t ***esp_ble_mesh_find_sig_model**(const *esp_ble_mesh_elem_t* **element*, uint16_t *model_id*)

Find the SIG model with the given element and Model id.

Return Pointer to the SIG Model on success, or NULL on failure which means the SIG Model is not found.

Parameters

- `element`: Element to which the model belongs.
- `model_id`: SIG model identifier.

const *esp_ble_mesh_comp_t* ***esp_ble_mesh_get_composition_data**(void)

Get the Composition data which has been registered.

Return Pointer to the Composition data on success, or NULL on failure which means the Composition data is not initialized.

Low Power Operation (Updating)

Header File

- bt/esp_ble_mesh/api/core/include/esp_ble_mesh_low_power_api.h

Functions

esp_err_t **esp_ble_mesh_lpn_enable**(void)

Enable BLE Mesh device LPN functionality.

Note This API enables LPN functionality. Once called, the proper Friend Request will be sent.

Return ESP_OK on success or error code otherwise.

esp_err_t **esp_ble_mesh_lpn_disable**(bool *force*)

Disable BLE Mesh device LPN functionality.

Return ESP_OK on success or error code otherwise.

Parameters

- `force`: when disabling LPN functionality, use this flag to indicate whether directly clear corresponding information or just send friend clear to disable it if friendship has already been established.

esp_err_t **esp_ble_mesh_lpn_poll**(void)

LPN tries to poll messages from the Friend Node.

Note The Friend Poll message is sent by a Low Power node to ask the Friend node to send a message that it has stored for the Low Power node. Users can call this API to send Friend Poll message manually. If this API is not invoked, the bottom layer of the Low Power node will send Friend Poll before the PollTimeout timer expires. If the corresponding Friend Update is received and MD is set to 0, which means there are no messages for the Low Power node, then the Low Power node will stop scanning.

Return ESP_OK on success or error code otherwise.

Send/Publish Messages, add Local AppKey, etc.

Header File

- bt/esp_ble_mesh/api/core/include/esp_ble_mesh_networking_api.h

Functions

esp_err_t **esp_ble_mesh_register_custom_model_callback**(*esp_ble_mesh_model_cb_t callback*)

Register BLE Mesh callback for user-defined models' operations. This callback can report the following events generated for the user-defined models:

- Call back the messages received by user-defined client and server models to the application layer;
- If users call esp_ble_mesh_server/client_model_send, this callback notifies the application layer of the send_complete event;
- If user-defined client model sends a message that requires response, and the response message is received after the timer expires, the response message will be reported to the application layer as published by a peer device;
- If the user-defined client model fails to receive the response message during a specified period of time, a timeout event will be reported to the application layer.

Note The client models (i.e. Config Client model, Health Client model, Generic Client models, Sensor Client model, Scene Client model and Lighting Client models) that have been realized internally have their specific register functions. For example, esp_ble_mesh_register_config_client_callback is the register function for Config Client Model.

Return ESP_OK on success or error code otherwise.

Parameters

- `callback`: Pointer to the callback function.

esp_err_t **esp_ble_mesh_model_msg_opcode_init**(uint8_t **data*, uint32_t *opcode*)

Add the message opcode to the beginning of the model message before sending or publishing the model message.

Note This API is only used to set the opcode of the message.

Return ESP_OK on success or error code otherwise.

Parameters

- `data`: Pointer to the message data.
- `opcode`: The message opcode.

esp_err_t **esp_ble_mesh_client_model_init**(*esp_ble_mesh_model_t* **model*)

Initialize the user-defined client model. All user-defined client models shall call this function to initialize the client model internal data. Node: Before calling this API, the op_pair_size and op_pair variabled within the user_data(defined using esp_ble_mesh_client_t_) of the client model need to be initialized.

Return ESP_OK on success or error code otherwise.

Parameters

- `model`: BLE Mesh Client model to which the message belongs.

esp_err_t **esp_ble_mesh_client_model_deinit**(*esp_ble_mesh_model_t* **model*)

De-initialize the user-defined client model.

Note This function shall be invoked before esp_ble_mesh_deinit() is called.

Return ESP_OK on success or error code otherwise.

Parameters

- `model`: Pointer of the Client model.

esp_err_t **esp_ble_mesh_server_model_send_msg**(*esp_ble_mesh_model_t* **model*, *esp_ble_mesh_msg_ctx_t* **ctx*, uint32_t *opcode*, uint16_t *length*, uint8_t **data*)

Send server model messages(such as server model status messages).

Return ESP_OK on success or error code otherwise.

Parameters

- `model`: BLE Mesh Server Model to which the message belongs.
- `ctx`: Message context, includes keys, TTL, etc.
- `opcode`: Message opcode.
- `length`: Message length (exclude the message opcode).
- `data`: Parameters of Access Payload (exclude the message opcode) to be sent.

esp_err_t **esp_ble_mesh_client_model_send_msg**(*esp_ble_mesh_model_t* **model*, *esp_ble_mesh_msg_ctx_t* **ctx*, uint32_t *opcode*, uint16_t *length*, uint8_t **data*, int32_t *msg_timeout*, bool *need_rsp*, *esp_ble_mesh_dev_role_t* *device_role*)

Send client model message (such as model get, set, etc).

Return ESP_OK on success or error code otherwise.

Parameters

- `model`: BLE Mesh Client Model to which the message belongs.
- `ctx`: Message context, includes keys, TTL, etc.
- `opcode`: Message opcode.
- `length`: Message length (exclude the message opcode).
- `data`: Parameters of the Access Payload (exclude the message opcode) to be sent.
- `msg_timeout`: Time to get response to the message (in milliseconds).
- `need_rsp`: TRUE if the opcode requires the peer device to reply, FALSE otherwise.
- `device_role`: Role of the device (Node/Provisioner) that sends the message.

esp_err_t **esp_ble_mesh_model_publish**(*esp_ble_mesh_model_t* *model, uint32_t opcode, uint16_t length, uint8_t *data, *esp_ble_mesh_dev_role_t* device_role)

Send a model publication message.

Note Before calling this function, the user needs to ensure that the model publication message (*esp_ble_mesh_model_pub_t::msg*) contains a valid message to be sent. And if users want to update the publishing message, this API should be called in ESP_BLE_MESH_MODEL_PUBLISH_UPDATE_EVT with the message updated.

Return ESP_OK on success or error code otherwise.

Parameters

- `model`: Mesh (client) Model publishing the message.
- `opcode`: Message opcode.
- `length`: Message length (exclude the message opcode).
- `data`: Parameters of the Access Payload (exclude the message opcode) to be sent.
- `device_role`: Role of the device (node/provisioner) publishing the message of the type esp_ble_mesh_dev_role_t.

esp_err_t **esp_ble_mesh_server_model_update_state**(*esp_ble_mesh_model_t* *model, *esp_ble_mesh_server_state_type_t* type, *esp_ble_mesh_server_state_value_t* *value)

Update a server model state value. If the model publication state is set properly (e.g. publish address is set to a valid address), it will publish corresponding status message.

Note Currently this API is used to update bound state value, not for all server model states.

Return ESP_OK on success or error code otherwise.

Parameters

- `model`: Server model which is going to update the state.
- `type`: Server model state type.
- `value`: Server model state value.

esp_err_t **esp_ble_mesh_node_local_reset**(void)

Reset the provisioning procedure of the local BLE Mesh node.

Note All provisioning information in this node will be deleted and the node needs to be reprovisioned. The API function esp_ble_mesh_node_prov_enable() needs to be called to start a new provisioning procedure.

Return ESP_OK on success or error code otherwise.

esp_err_t **esp_ble_mesh_provisioner_set_node_name**(uint16_t *index*, **const** char **name*)

This function is called to set the node (provisioned device) name.

Note index is obtained from the parameters of ESP_BLE_MESH_PROVISIONER_PROV_COMPLETE_EVT.

Return ESP_OK on success or error code otherwise.

Parameters

- `index`: Index of the node in the node queue.
- `name`: Name (end by '\0') to be set for the node.

const char ***esp_ble_mesh_provisioner_get_node_name**(uint16_t *index*)

This function is called to get the node (provisioned device) name.

Note index is obtained from the parameters of ESP_BLE_MESH_PROVISIONER_PROV_COMPLETE_EVT.

Return Node name on success, or NULL on failure.

Parameters

- `index`: Index of the node in the node queue.

uint16_t **esp_ble_mesh_provisioner_get_node_index**(**const** char **name*)

This function is called to get the node (provisioned device) index.

Return Node index on success, or an invalid value (0xFFFF) on failure.

Parameters

- `name`: Name of the node (end by '\0').

esp_err_t **esp_ble_mesh_provisioner_store_node_comp_data**(uint16_t *unicast_addr*, uint8_t *data, uint16_t *length*)

This function is called to store the Composition Data of the node.

Return ESP_OK on success or error code otherwise.

Parameters

- `unicast_addr`: Element address of the node
- `data`: Pointer of Composition Data
- `length`: Length of Composition Data

esp_ble_mesh_node_t ***esp_ble_mesh_provisioner_get_node_with_uuid**(const uint8_t *uuid*[16])

This function is called to get the provisioned node information with the node device uuid.

Return Pointer of the node info struct or NULL on failure.

Parameters

- `uuid`: Device UUID of the node

esp_ble_mesh_node_t ***esp_ble_mesh_provisioner_get_node_with_addr**(uint16_t *unicast_addr*)

This function is called to get the provisioned node information with the node unicast address.

Return Pointer of the node info struct or NULL on failure.

Parameters

- `unicast_addr`: Unicast address of the node

esp_err_t **esp_ble_mesh_provisioner_delete_node_with_uuid**(const uint8_t *uuid*[16])

This function is called to delete the provisioned node information with the node device uuid.

Return ESP_OK on success or error code otherwise.

Parameters

- `uuid`: Device UUID of the node

esp_err_t **esp_ble_mesh_provisioner_delete_node_with_addr**(uint16_t *unicast_addr*)

This function is called to delete the provisioned node information with the node unicast address.

Return ESP_OK on success or error code otherwise.

Parameters

- `unicast_addr`: Unicast address of the node

esp_err_t **esp_ble_mesh_provisioner_add_local_app_key**(**const** uint8_t *app_key*[16], uint16_t *net_idx*, uint16_t *app_idx*)

This function is called to add a local AppKey for Provisioner.

Note app_key: If set to NULL, app_key will be generated internally. net_idx: Should be an existing one. app_idx: If it is going to be generated internally, it should be set to 0xFFFF, and the new app_idx will be reported via an event.

Return ESP_OK on success or error code otherwise.

Parameters

- `app_key`: The app key to be set for the local BLE Mesh stack.
- `net_idx`: The network key index.
- `app_idx`: The app key index.

esp_err_t **esp_ble_mesh_provisioner_update_local_app_key**(**const** uint8_t *app_key*[16], uint16_t *net_idx*, uint16_t *app_idx*)

This function is used to update a local AppKey for Provisioner.

Return ESP_OK on success or error code otherwise.

Parameters

- `app_key`: Value of the AppKey.
- `net_idx`: Corresponding NetKey Index.
- `app_idx`: The AppKey Index

const uint8_t ***esp_ble_mesh_provisioner_get_local_app_key**(uint16_t *net_idx*, uint16_t *app_idx*)

This function is called by Provisioner to get the local app key value.

Return App key on success, or NULL on failure.

Parameters

- `net_idx`: Network key index.
- `app_idx`: Application key index.

esp_err_t **esp_ble_mesh_provisioner_bind_app_key_to_local_model**(uint16_t *element_addr*, uint16_t *app_idx*, uint16_t *model_id*, uint16_t *company_id*)

This function is called by Provisioner to bind own model with proper app key.

Note company_id: If going to bind app_key with local vendor model, company_id should be set to 0xFFFF.

Return ESP_OK on success or error code otherwise.

Parameters

- `element_addr`: Provisioner local element address
- `app_idx`: Provisioner local appkey index
- `model_id`: Provisioner local model id
- `company_id`: Provisioner local company id

esp_err_t **esp_ble_mesh_provisioner_add_local_net_key**(**const** uint8_t *net_key*[16], uint16_t *net_idx*)

This function is called by Provisioner to add local network key.

Note net_key: If set to NULL, net_key will be generated internally. net_idx: If it is going to be generated internally, it should be set to 0xFFFF, and the new net_idx will be reported via an event.

Return ESP_OK on success or error code otherwise.

Parameters

- `net_key`: The network key to be added to the Provisioner local BLE Mesh stack.
- `net_idx`: The network key index.

esp_err_t **esp_ble_mesh_provisioner_update_local_net_key**(**const** uint8_t *net_key*[16], uint16_t *net_idx*)

This function is called by Provisioner to update a local network key.

Return ESP_OK on success or error code otherwise.

Parameters

- `net_key`: Value of the NetKey.
- `net_idx`: The NetKey Index.

const uint8_t ***esp_ble_mesh_provisioner_get_local_net_key**(uint16_t *net_idx*)

This function is called by Provisioner to get the local network key value.

Return Network key on success, or NULL on failure.

Parameters

- `net_idx`: Network key index.

uint16_t **esp_ble_mesh_provisioner_get_prov_node_count**(void)

This function is called by Provisioner to get provisioned node count.

Return Number of the provisioned nodes.

const uint8_t ***esp_ble_mesh_get_fast_prov_app_key**(uint16_t *net_idx*, uint16_t *app_idx*)

This function is called to get fast provisioning application key.

Return Application key on success, or NULL on failure.

Parameters

- `net_idx`: Network key index.
- `app_idx`: Application key index.

Type Definitions

typedef void (***esp_ble_mesh_model_cb_t**)(*esp_ble_mesh_model_cb_event_t* event, *esp_ble_mesh_model_cb_param_t* *param)

: event, event code of user-defined model events; param, parameters of user-defined model events

ESP-BLE-MESH Node/Provisioner Provisioning

Header File

- bt/esp_ble_mesh/api/core/include/esp_ble_mesh_provisioning_api.h

Functions

esp_err_t **esp_ble_mesh_register_prov_callback**(*esp_ble_mesh_prov_cb_t callback*)

Register BLE Mesh provisioning callback.

Return ESP_OK on success or error code otherwise.

Parameters

- `callback`: Pointer to the callback function.

bool **esp_ble_mesh_node_is_provisioned**(void)

Check if a device has been provisioned.

Return TRUE if the device is provisioned, FALSE if the device is unprovisioned.

esp_err_t **esp_ble_mesh_node_prov_enable**(*esp_ble_mesh_prov_bearer_t bearers*)

Enable specific provisioning bearers to get the device ready for provisioning.

Note PB-ADV: send unprovisioned device beacon. PB-GATT: send connectable advertising packets.

Return ESP_OK on success or error code otherwise.

Parameters

- `bearers`: Bit-wise OR of provisioning bearers.

esp_err_t **esp_ble_mesh_node_prov_disable**(*esp_ble_mesh_prov_bearer_t bearers*)

Disable specific provisioning bearers to make a device inaccessible for provisioning.

Return ESP_OK on success or error code otherwise.

Parameters

- `bearers`: Bit-wise OR of provisioning bearers.

esp_err_t **esp_ble_mesh_node_set_oob_pub_key**(uint8_t *pub_key_x*[32], uint8_t *pub_key_y*[32], uint8_t *private_key*[32])

Unprovisioned device set own oob public key & private key pair.

Return ESP_OK on success or error code otherwise.

Parameters

- `pub_key_x`: Unprovisioned device' s Public Key X
- `pub_key_y`: Unprovisioned device' s Public Key Y
- `private_key`: Unprovisioned device' s Private Key

esp_err_t **esp_ble_mesh_node_input_number**(uint32_t *number*)

Provide provisioning input OOB number.

Note This is intended to be called if the user has received ESP_BLE_MESH_NODE_PROV_INPUT_EVT with ESP_BLE_MESH_ENTER_NUMBER as the action.

Return ESP_OK on success or error code otherwise.

Parameters

- `number`: Number input by device.

esp_err_t **esp_ble_mesh_node_input_string**(**const** char **string*)

Provide provisioning input OOB string.

Note This is intended to be called if the user has received ESP_BLE_MESH_NODE_PROV_INPUT_EVT with ESP_BLE_MESH_ENTER_STRING as the action.

Return ESP_OK on success or error code otherwise.

Parameters

- `string`: String input by device.

esp_err_t **esp_ble_mesh_set_unprovisioned_device_name**(**const** char **name*)

Using this function, an unprovisioned device can set its own device name, which will be broadcasted in its advertising data.

Note This API applicable to PB-GATT mode only by setting the name to the scan response data, it doesn't apply to PB-ADV mode.

Return ESP_OK on success or error code otherwise.

Parameters

- `name`: Unprovisioned device name

esp_err_t **esp_ble_mesh_provisioner_read_oob_pub_key**(uint8_t *link_idx*, uint8_t *pub_key_x*[32], uint8_t *pub_key_y*[32])

Provisioner inputs unprovisioned device's oob public key.

Return ESP_OK on success or error code otherwise.

Parameters

- `link_idx`: The provisioning link index
- `pub_key_x`: Unprovisioned device's Public Key X
- `pub_key_y`: Unprovisioned device's Public Key Y

esp_err_t **esp_ble_mesh_provisioner_input_string**(**const** char **string*, uint8_t *link_idx*)

Provide provisioning input OOB string.

This is intended to be called after the *esp_ble_mesh_prov_t* prov_input_num callback has been called with ESP_BLE_MESH_ENTER_STRING as the action.

Return ESP_OK on success or error code otherwise.

Parameters

- `string`: String input by Provisioner.
- `link_idx`: The provisioning link index.

esp_err_t **esp_ble_mesh_provisioner_input_number**(uint32_t *number*, uint8_t *link_idx*)

Provide provisioning input OOB number.

This is intended to be called after the *esp_ble_mesh_prov_t* prov_input_num callback has been called with ESP_BLE_MESH_ENTER_NUMBER as the action.

Return ESP_OK on success or error code otherwise.

Parameters

- `number`: Number input by Provisioner.
- `link_idx`: The provisioning link index.

esp_err_t **esp_ble_mesh_provisioner_prov_enable**(*esp_ble_mesh_prov_bearer_t bearers*)

Enable one or more provisioning bearers.

Note PB-ADV: Enable BLE scan. PB-GATT: Initialize corresponding BLE Mesh Proxy info.

Return ESP_OK on success or error code otherwise.

Parameters

- `bearers`: Bit-wise OR of provisioning bearers.

esp_err_t **esp_ble_mesh_provisioner_prov_disable**(*esp_ble_mesh_prov_bearer_t bearers*)

Disable one or more provisioning bearers.

Note PB-ADV: Disable BLE scan. PB-GATT: Break any existing BLE Mesh Provisioning connections.

Return ESP_OK on success or error code otherwise.

Parameters

- `bearers`: Bit-wise OR of provisioning bearers.

esp_err_t **esp_ble_mesh_provisioner_add_unprov_dev**(*esp_ble_mesh_unprov_dev_add_t *add_dev*, *esp_ble_mesh_dev_add_flag_t flags*)

Add unprovisioned device info to the unprov_dev queue.

Return ESP_OK on success or error code otherwise.

Note : 1. Currently address type only supports public address and static random address.

1. If device UUID and/or device address as well as address type already exist in the device queue, but the bearer is different from the existing one, add operation will also be successful and it will update the provision bearer supported by the device.
2. For example, if the Provisioner wants to add an unprovisioned device info before receiving its unprovisioned device beacon or Mesh Provisioning advertising packets, the Provisioner can use this API to add the device info with each one or both of device UUID and device address added. When the Provisioner gets the device' s advertising packets, it will start provisioning the device internally.

- In this situation, the Provisioner can set bearers with each one or both of ESP_BLE_MESH_PROV_ADV and ESP_BLE_MESH_PROV_GATT enabled, and cannot set flags with ADD_DEV_START_PROV_NOW_FLAG enabled.

3. Another example is when the Provisioner receives the unprovisioned device' s beacon or Mesh Provisioning advertising packets, the advertising packets will be reported on to the application layer using the callback registered by the function esp_ble_mesh_register_prov_callback. And in the callback, the Provisioner can call this API to start provisioning the device.

   - If the Provisioner uses PB-ADV to provision, either one or both of device UUID and device address can be added, bearers shall be set with ESP_BLE_MESH_PROV_ADV enabled and the flags shall be set with ADD_DEV_START_PROV_NOW_FLAG enabled.
   - If the Provisioner uses PB-GATT to provision, both the device UUID and device address need to be added, bearers shall be set with ESP_BLE_MESH_PROV_GATT enabled, and the flags shall be set with ADD_DEV_START_PROV_NOW_FLAG enabled.
   - If the Provisioner just wants to store the unprovisioned device info when receiving its advertising packets and start to provision it the next time (e.g. after receiving its advertising packets again), then it can add the device info with either one or both of device UUID and device address included. Bearers can be set with either one or both of ESP_BLE_MESH_PROV_ADV and ESP_BLE_MESH_PROV_GATT enabled (recommend to enable the bearer which will receive its advertising packets, because if the other bearer is enabled, the Provisioner is not aware if the device supports the bearer), and flags cannot be set with ADD_DEV_START_PROV_NOW_FLAG enabled.
   - Note: ESP_BLE_MESH_PROV_ADV, ESP_BLE_MESH_PROV_GATT and ADD_DEV_START_PROV_NOW_FLAG can not be enabled at the same time.

Parameters

- `add_dev`: Pointer to a struct containing the device information
- `flags`: Flags indicate several operations on the device information
  - Remove device information from queue after device has been provisioned (BIT0)
  - Start provisioning immediately after device is added to queue (BIT1)
  - Device can be removed if device queue is full (BIT2)

esp_err_t **esp_ble_mesh_provisioner_prov_device_with_addr**(**const** uint8_t *uuid*[16], *esp_ble_mesh_bd_addr_t* *addr*, *esp_ble_mesh_addr_type_t* *addr_type*, *esp_ble_mesh_prov_bearer_t* *bearer*, uint16_t *oob_info*, uint16_t *unicast_addr*)

Provision an unprovisioned device and assign a fixed unicast address for it in advance.

Return Zero on success or (negative) error code otherwise.

Note : 1. Currently address type only supports public address and static random address.

1. Bearer must be equal to ESP_BLE_MESH_PROV_ADV or ESP_BLE_MESH_PROV_GATT, since Provisioner will start to provision a device immediately once this function is invoked. And the input bearer must be identical with the one within the parameters of the ESP_BLE_MESH_PROVISIONER_RECV_UNPROV_ADV_PKT_EVT event.
2. If this function is used by a Provisioner to provision devices, the application should take care of the assigned unicast address and avoid overlap of the unicast addresses of different nodes.
3. Recommend to use only one of the functions "esp_ble_mesh_provisioner_add_unprov_dev" and "esp_ble_mesh_provisioner_prov_device_with_addr" by a Provisioner.

Parameters

- `uuid`: Device UUID of the unprovisioned device
- `addr`: Device address of the unprovisioned device
- `addr_type`: Device address type of the unprovisioned device
- `bearer`: Provisioning bearer going to be used by Provisioner
- `oob_info`: OOB info of the unprovisioned device
- `unicast_addr`: Unicast address going to be allocated for the unprovisioned device

esp_err_t **esp_ble_mesh_provisioner_delete_dev**(*esp_ble_mesh_device_delete_t* **del_dev*)

Delete device from queue, reset current provisioning link and reset the node.

Note If the device is in the queue, remove it from the queue; if the device is being provisioned, terminate the provisioning procedure; if the device has already been provisioned, reset the device. And either one of the addr or device UUID can be input.

Return ESP_OK on success or error code otherwise.

Parameters

- `del_dev`: Pointer to a struct containing the device information.

esp_err_t **esp_ble_mesh_provisioner_set_dev_uuid_match**(**const** uint8_t **match_val*, uint8_t *match_len*, uint8_t *offset*, bool *prov_after_match*)

This function is called by Provisioner to set the part of the device UUID to be compared before starting to provision.

Return ESP_OK on success or error code otherwise.

Parameters

- `match_val`: Value to be compared with the part of the device UUID.
- `match_len`: Length of the compared match value.
- `offset`: Offset of the device UUID to be compared (based on zero).
- `prov_after_match`: Flag used to indicate whether provisioner should start to provision the device immediately if the part of the UUID matches.

esp_err_t **esp_ble_mesh_provisioner_set_prov_data_info**(*esp_ble_mesh_prov_data_info_t* **prov_data_info*)

This function is called by Provisioner to set provisioning data information before starting to provision.

Return ESP_OK on success or error code otherwise.

Parameters

- `prov_data_info`: Pointer to a struct containing net_idx or flags or iv_index.

esp_err_t **esp_ble_mesh_provisioner_set_static_oob_value**(const uint8_t **value*, uint8_t *length*)

This function is called by Provisioner to set static oob value used for provisioning.

Return ESP_OK on success or error code otherwise.

Parameters

- `value`: Pointer to the static oob value.
- `length`: Length of the static oob value.

esp_err_t **esp_ble_mesh_provisioner_set_primary_elem_addr**(uint16_t *addr*)

This function is called by Provisioner to set own Primary element address.

Note This API must be invoked when BLE Mesh initialization is completed successfully, and can be invoked before Provisioner functionality is enabled. Once this API is invoked successfully, the prov_unicast_addr value in the struct *esp_ble_mesh_prov_t* will be ignored, and Provisioner will use this address as its own primary element address. And if the unicast address going to assigned for the next unprovisioned device is smaller than the input address + element number of Provisioner, then the address for the next unprovisioned device will be recalculated internally.

Return ESP_OK on success or error code otherwise.

Parameters

- `addr`: Unicast address of the Primary element of Provisioner.

esp_err_t **esp_ble_mesh_set_fast_prov_info**(*esp_ble_mesh_fast_prov_info_t* **fast_prov_info*)

This function is called to set provisioning data information before starting fast provisioning.

Return ESP_OK on success or error code otherwise.

Parameters

- `fast_prov_info`: Pointer to a struct containing unicast address range, net_idx, etc.

esp_err_t **esp_ble_mesh_set_fast_prov_action**(*esp_ble_mesh_fast_prov_action_t action*)

This function is called to start/suspend/exit fast provisioning.

Return ESP_OK on success or error code otherwise.

Parameters

- `action`: fast provisioning action (i.e. enter, suspend, exit).

Type Definitions

typedef void (***esp_ble_mesh_prov_cb_t**)(*esp_ble_mesh_prov_cb_event_t* event, *esp_ble_mesh_prov_cb_param_t* *param)

: event, event code of provisioning events; param, parameters of provisioning events

typedef void (***esp_ble_mesh_prov_adv_cb_t**)(**const** *esp_ble_mesh_bd_addr_t* addr, **const** *esp_ble_mesh_addr_type_t* addr_type, **const** uint8_t adv_type, **const** uint8_t *dev_uuid, uint16_t oob_info, *esp_ble_mesh_prov_bearer_t* bearer)

Callback for Provisioner that received advertising packets from unprovisioned devices which are not in the unprovisioned device queue.

Report on the unprovisioned device beacon and mesh provisioning service adv data to application.

Parameters

- `addr`: Pointer to the unprovisioned device address.
- `addr_type`: Unprovisioned device address type.
- `adv_type`: Adv packet type(ADV_IND or ADV_NONCONN_IND).
- `dev_uuid`: Unprovisioned device UUID pointer.
- `oob_info`: OOB information of the unprovisioned device.
- `bearer`: Adv packet received from PB-GATT or PB-ADV bearer.

ESP-BLE-MESH GATT Proxy Server

Header File

- bt/esp_ble_mesh/api/core/include/esp_ble_mesh_proxy_api.h

Functions

esp_err_t **esp_ble_mesh_proxy_identity_enable**(void)

Enable advertising with Node Identity.

Note This API requires that GATT Proxy support be enabled. Once called, each subnet starts advertising using Node Identity for the next 60 seconds, and after 60s Network ID will be advertised. Under normal conditions, the BLE Mesh Proxy Node Identity and Network ID advertising will be enabled automatically by BLE Mesh stack after the device is provisioned.

Return ESP_OK on success or error code otherwise.

esp_err_t **esp_ble_mesh_proxy_gatt_enable**(void)

Enable BLE Mesh GATT Proxy Service.

Return ESP_OK on success or error code otherwise.

esp_err_t **esp_ble_mesh_proxy_gatt_disable**(void)

Disconnect the BLE Mesh GATT Proxy connection if there is any, and disable the BLE Mesh GATT Proxy Service.

Return ESP_OK on success or error code otherwise.

esp_err_t **esp_ble_mesh_proxy_client_connect**(*esp_ble_mesh_bd_addr_t* *addr*, *esp_ble_mesh_addr_type_t* *addr_type*, uint16_t *net_idx*)

Proxy Client creates a connection with the Proxy Server.

Return ESP_OK on success or error code otherwise.

Parameters

- `addr`: Device address of the Proxy Server.
- `addr_type`: Device address type(public or static random).
- `net_idx`: NetKey Index related with Network ID in the Mesh Proxy advertising packet.

esp_err_t **esp_ble_mesh_proxy_client_disconnect**(uint8_t *conn_handle*)

Proxy Client terminates a connection with the Proxy Server.

Return ESP_OK on success or error code otherwise.

Parameters

- `conn_handle`: Proxy connection handle.

esp_err_t **esp_ble_mesh_proxy_client_set_filter_type**(uint8_t *conn_handle*, uint16_t *net_idx*, *esp_ble_mesh_proxy_filter_type_t filter_type*)

Proxy Client sets the filter type of the Proxy Server.

Return ESP_OK on success or error code otherwise.

Parameters

- `conn_handle`: Proxy connection handle.
- `net_idx`: Corresponding NetKey Index.
- `filter_type`: whitelist or blacklist.

esp_err_t **esp_ble_mesh_proxy_client_add_filter_addr**(uint8_t *conn_handle*, uint16_t *net_idx*, uint16_t *addr, uint16_t *addr_num*)

Proxy Client adds address to the Proxy Server filter list.

Return ESP_OK on success or error code otherwise.

Parameters

- `conn_handle`: Proxy connection handle.
- `net_idx`: Corresponding NetKey Index.
- `addr`: Pointer to the filter address.
- `addr_num`: Number of the filter address.

esp_err_t **esp_ble_mesh_proxy_client_remove_filter_addr**(uint8_t *conn_handle*, uint16_t *net_idx*, uint16_t *addr, uint16_t *addr_num*)

Proxy Client removes address from the Proxy Server filter list.

Return ESP_OK on success or error code otherwise.

Parameters

- `conn_handle`: Proxy connection handle.
- `net_idx`: Corresponding NetKey Index.
- `addr`: Pointer to the filter address.
- `addr_num`: Number of the filter address.

ESP-BLE-MESH Models API Reference

This section contains ESP-BLE-MESH Model related APIs, event types, event parameters, etc.

There are six categories of models:

- *Configuration Client/Server Models*
- *Health Client/Server Models*
- *Generic Client/Server Models*
- *Sensor Client/Server Models*
- *Time and Scenes Client/Server Models*
- *Lighting Client/Server Models*

注解: Definitions related to Server Models are being updated, and will be released soon.

Configuration Client/Server Models

Header File

- bt/esp_ble_mesh/api/models/include/esp_ble_mesh_config_model_api.h

Functions

esp_err_t **esp_ble_mesh_register_config_client_callback**(*esp_ble_mesh_cfg_client_cb_t callback*)

Register BLE Mesh Config Client Model callback.

Return ESP_OK on success or error code otherwise.

Parameters

- `callback`: Pointer to the callback function.

esp_err_t **esp_ble_mesh_register_config_server_callback**(*esp_ble_mesh_cfg_server_cb_t callback*)

Register BLE Mesh Config Server Model callback.

Return ESP_OK on success or error code otherwise.

Parameters

- `callback`: Pointer to the callback function.

esp_err_t **esp_ble_mesh_config_client_get_state**(*esp_ble_mesh_client_common_param_t* **params*, *esp_ble_mesh_cfg_client_get_state_t* **get_state*)

Get the value of Config Server Model states using the Config Client Model get messages.

Note If you want to find the opcodes and corresponding meanings accepted by this API, please refer to esp_ble_mesh_opcode_config_client_get_t in esp_ble_mesh_defs.h

Return ESP_OK on success or error code otherwise.

Parameters

- `params`: Pointer to BLE Mesh common client parameters.
- `get_state`: Pointer to a union, each kind of opcode corresponds to one structure inside. Shall not be set to NULL.

esp_err_t **esp_ble_mesh_config_client_set_state**(*esp_ble_mesh_client_common_param_t* **params*, *esp_ble_mesh_cfg_client_set_state_t* **set_state*)

Set the value of the Configuration Server Model states using the Config Client Model set messages.

Note If you want to find the opcodes and corresponding meanings accepted by this API, please refer to esp_ble_mesh_opcode_config_client_set_t in esp_ble_mesh_defs.h

Return ESP_OK on success or error code otherwise.

Parameters

- `params`: Pointer to BLE Mesh common client parameters.
- `set_state`: Pointer to a union, each kind of opcode corresponds to one structure inside. Shall not be set to NULL.

Unions

union esp_ble_mesh_cfg_client_get_state_t

#include <esp_ble_mesh_config_model_api.h> For ESP_BLE_MESH_MODEL_OP_BEACON_GET ESP_BLE_MESH_MODEL_OP_COMPOSITION_DATA_GET ESP_BLE_MESH_MODEL_OP_DEFAULT_T ESP_BLE_MESH_MODEL_OP_GATT_PROXY_GET ESP_BLE_MESH_MODEL_OP_RELAY_GET ESP_BLE_MESH_MODEL_OP_MODEL_PUB_GET ESP_BLE_MESH_MODEL_OP_FRIEND_GET ESP_BLE_MESH_MODEL_OP_HEARTBEAT_PUB_GET ESP_BLE_MESH_MODEL_OP_HEARTBEAT_S the get_state parameter in the esp_ble_mesh_config_client_get_state function should not be set to NULL.

Public Members

esp_ble_mesh_cfg_model_pub_get_t **model_pub_get**

For ESP_BLE_MESH_MODEL_OP_MODEL_PUB_GET.

esp_ble_mesh_cfg_composition_data_get_t **comp_data_get**

For ESP_BLE_MESH_MODEL_OP_COMPOSITION_DATA_GET.

esp_ble_mesh_cfg_sig_model_sub_get_t **sig_model_sub_get**

For ESP_BLE_MESH_MODEL_OP_SIG_MODEL_SUB_GET

esp_ble_mesh_cfg_vnd_model_sub_get_t **vnd_model_sub_get**

For ESP_BLE_MESH_MODEL_OP_VENDOR_MODEL_SUB_GET

esp_ble_mesh_cfg_app_key_get_t **app_key_get**

For ESP_BLE_MESH_MODEL_OP_APP_KEY_GET.

esp_ble_mesh_cfg_node_identity_get_t **node_identity_get**

For ESP_BLE_MESH_MODEL_OP_NODE_IDENTITY_GET.

esp_ble_mesh_cfg_sig_model_app_get_t **sig_model_app_get**

For ESP_BLE_MESH_MODEL_OP_SIG_MODEL_APP_GET

esp_ble_mesh_cfg_vnd_model_app_get_t **vnd_model_app_get**

For ESP_BLE_MESH_MODEL_OP_VENDOR_MODEL_APP_GET

esp_ble_mesh_cfg_kr_phase_get_t **kr_phase_get**

For ESP_BLE_MESH_MODEL_OP_KEY_REFRESH_PHASE_GET

esp_ble_mesh_cfg_lpn_polltimeout_get_t **lpn_pollto_get**

For ESP_BLE_MESH_MODEL_OP_LPN_POLLTIMEOUT_GET

union esp_ble_mesh_cfg_client_set_state_t

#include <esp_ble_mesh_config_model_api.h> For ESP_BLE_MESH_MODEL_OP_BEACON_SET ESP_BLE_MESH_MODEL_OP_DEFAULT_TTL_SET ESP_BLE_MESH_MODEL_OP_GATT_PROXY_SET ESP_BLE_MESH_MODEL_OP_RELAY_SET ESP_BLE_MESH_MODEL_OP_MODEL_PUB_SET ESP_BLE_MESH_MODEL_OP_MODEL_SUB_ADD ESP_BLE_MESH_MODEL_OP_MODEL_SUB_VIRTU ESP_BLE_MESH_MODEL_OP_MODEL_SUB_DELETE ESP_BLE_MESH_MODEL_OP_MODEL_SUB_VI ESP_BLE_MESH_MODEL_OP_MODEL_SUB_OVERWRITE ESP_BLE_MESH_MODEL_OP_MODEL_SUB ESP_BLE_MESH_MODEL_OP_NET_KEY_ADD ESP_BLE_MESH_MODEL_OP_APP_KEY_ADD ESP_BLE_MESH_MODEL_OP_MODEL_APP_BIND ESP_BLE_MESH_MODEL_OP_NODE_RESET ESP_BLE_MESH_MODEL_OP_FRIEND_SET ESP_BLE_MESH_MODEL_OP_HEARTBEAT_PUB_SET ESP_BLE_MESH_MODEL_OP_HEARTBEAT_SUB_SET the set_state parameter in the esp_ble_mesh_config_client_set_state function should not be set to NULL.

Public Members

esp_ble_mesh_cfg_beacon_set_t **beacon_set**
For ESP_BLE_MESH_MODEL_OP_BEACON_SET

esp_ble_mesh_cfg_default_ttl_set_t **default_ttl_set**
For ESP_BLE_MESH_MODEL_OP_DEFAULT_TTL_SET

esp_ble_mesh_cfg_friend_set_t **friend_set**
For ESP_BLE_MESH_MODEL_OP_FRIEND_SET

esp_ble_mesh_cfg_gatt_proxy_set_t **gatt_proxy_set**
For ESP_BLE_MESH_MODEL_OP_GATT_PROXY_SET

esp_ble_mesh_cfg_relay_set_t **relay_set**
For ESP_BLE_MESH_MODEL_OP_RELAY_SET

esp_ble_mesh_cfg_net_key_add_t **net_key_add**
For ESP_BLE_MESH_MODEL_OP_NET_KEY_ADD

esp_ble_mesh_cfg_app_key_add_t **app_key_add**
For ESP_BLE_MESH_MODEL_OP_APP_KEY_ADD

esp_ble_mesh_cfg_model_app_bind_t **model_app_bind**
For ESP_BLE_MESH_MODEL_OP_MODEL_APP_BIND

esp_ble_mesh_cfg_model_pub_set_t **model_pub_set**
For ESP_BLE_MESH_MODEL_OP_MODEL_PUB_SET

esp_ble_mesh_cfg_model_sub_add_t **model_sub_add**
For ESP_BLE_MESH_MODEL_OP_MODEL_SUB_ADD

esp_ble_mesh_cfg_model_sub_delete_t **model_sub_delete**
For ESP_BLE_MESH_MODEL_OP_MODEL_SUB_DELETE

esp_ble_mesh_cfg_model_sub_overwrite_t **model_sub_overwrite**
For ESP_BLE_MESH_MODEL_OP_MODEL_SUB_OVERWRITE

esp_ble_mesh_cfg_model_sub_va_add_t **model_sub_va_add**
For ESP_BLE_MESH_MODEL_OP_MODEL_SUB_VIRTUAL_ADDR_ADD

esp_ble_mesh_cfg_model_sub_va_delete_t **model_sub_va_delete**
For ESP_BLE_MESH_MODEL_OP_MODEL_SUB_VIRTUAL_ADDR_DELETE

esp_ble_mesh_cfg_model_sub_va_overwrite_t **model_sub_va_overwrite**
For ESP_BLE_MESH_MODEL_OP_MODEL_SUB_VIRTUAL_ADDR_OVERWRITE

esp_ble_mesh_cfg_heartbeat_pub_set_t **heartbeat_pub_set**
For ESP_BLE_MESH_MODEL_OP_HEARTBEAT_PUB_SET

esp_ble_mesh_cfg_heartbeat_sub_set_t **heartbeat_sub_set**
For ESP_BLE_MESH_MODEL_OP_HEARTBEAT_SUB_SET

esp_ble_mesh_cfg_model_pub_va_set_t **model_pub_va_set**

For ESP_BLE_MESH_MODEL_OP_MODEL_PUB_VIRTUAL_ADDR_SET

esp_ble_mesh_cfg_model_sub_delete_all_t **model_sub_delete_all**

For ESP_BLE_MESH_MODEL_OP_MODEL_SUB_DELETE_ALL

esp_ble_mesh_cfg_net_key_update_t **net_key_update**

For ESP_BLE_MESH_MODEL_OP_NET_KEY_UPDATE

esp_ble_mesh_cfg_net_key_delete_t **net_key_delete**

For ESP_BLE_MESH_MODEL_OP_NET_KEY_DELETE

esp_ble_mesh_cfg_app_key_update_t **app_key_update**

For ESP_BLE_MESH_MODEL_OP_APP_KEY_UPDATE

esp_ble_mesh_cfg_app_key_delete_t **app_key_delete**

For ESP_BLE_MESH_MODEL_OP_APP_KEY_DELETE

esp_ble_mesh_cfg_node_identity_set_t **node_identity_set**

For ESP_BLE_MESH_MODEL_OP_NODE_IDENTITY_SET

esp_ble_mesh_cfg_model_app_unbind_t **model_app_unbind**

For ESP_BLE_MESH_MODEL_OP_MODEL_APP_UNBIND

esp_ble_mesh_cfg_kr_phase_set_t **kr_phase_set**

For ESP_BLE_MESH_MODEL_OP_KEY_REFRESH_PHASE_SET

esp_ble_mesh_cfg_net_transmit_set_t **net_transmit_set**

For ESP_BLE_MESH_MODEL_OP_NETWORK_TRANSMIT_SET

union esp_ble_mesh_cfg_client_common_cb_param_t

#include <esp_ble_mesh_config_model_api.h> Configuration Client Model received message union.

Public Members

esp_ble_mesh_cfg_beacon_status_cb_t **beacon_status**

The beacon status value

esp_ble_mesh_cfg_comp_data_status_cb_t **comp_data_status**

The composition data status value

esp_ble_mesh_cfg_default_ttl_status_cb_t **default_ttl_status**

The default_ttl status value

esp_ble_mesh_cfg_gatt_proxy_status_cb_t **gatt_proxy_status**

The gatt_proxy status value

esp_ble_mesh_cfg_relay_status_cb_t **relay_status**

The relay status value

esp_ble_mesh_cfg_model_pub_status_cb_t **model_pub_status**
The model publication status value

esp_ble_mesh_cfg_model_sub_status_cb_t **model_sub_status**
The model subscription status value

esp_ble_mesh_cfg_net_key_status_cb_t **netkey_status**
The netkey status value

esp_ble_mesh_cfg_app_key_status_cb_t **appkey_status**
The appkey status value

esp_ble_mesh_cfg_mod_app_status_cb_t **model_app_status**
The model app status value

esp_ble_mesh_cfg_friend_status_cb_t **friend_status**
The friend status value

esp_ble_mesh_cfg_hb_pub_status_cb_t **heartbeat_pub_status**
The heartbeat publication status value

esp_ble_mesh_cfg_hb_sub_status_cb_t **heartbeat_sub_status**
The heartbeat subscription status value

esp_ble_mesh_cfg_net_trans_status_cb_t **net_transmit_status**
The network transmit status value

esp_ble_mesh_cfg_model_sub_list_cb_t **model_sub_list**
The model subscription list value

esp_ble_mesh_cfg_net_key_list_cb_t **netkey_list**
The network key index list value

esp_ble_mesh_cfg_app_key_list_cb_t **appkey_list**
The application key index list value

esp_ble_mesh_cfg_node_id_status_cb_t **node_identity_status**
The node identity status value

esp_ble_mesh_cfg_model_app_list_cb_t **model_app_list**
The model application key index list value

esp_ble_mesh_cfg_kr_phase_status_cb_t **kr_phase_status**
The key refresh phase status value

esp_ble_mesh_cfg_lpn_pollto_status_cb_t **lpn_timeout_status**
The low power node poll timeout status value

union esp_ble_mesh_cfg_server_state_change_t
#include <esp_ble_mesh_config_model_api.h> Configuration Server model state change value union.

Public Members

esp_ble_mesh_state_change_cfg_mod_pub_set_t **mod_pub_set**

The recv_op in ctx can be used to decide which state is changed.Config Model Publication Set

esp_ble_mesh_state_change_cfg_model_sub_add_t **mod_sub_add**

Config Model Subscription Add

esp_ble_mesh_state_change_cfg_model_sub_delete_t **mod_sub_delete**

Config Model Subscription Delete

esp_ble_mesh_state_change_cfg_netkey_add_t **netkey_add**

Config NetKey Add

esp_ble_mesh_state_change_cfg_netkey_update_t **netkey_update**

Config NetKey Update

esp_ble_mesh_state_change_cfg_netkey_delete_t **netkey_delete**

Config NetKey Delete

esp_ble_mesh_state_change_cfg_appkey_add_t **appkey_add**

Config AppKey Add

esp_ble_mesh_state_change_cfg_appkey_update_t **appkey_update**

Config AppKey Update

esp_ble_mesh_state_change_cfg_appkey_delete_t **appkey_delete**

Config AppKey Delete

esp_ble_mesh_state_change_cfg_model_app_bind_t **mod_app_bind**

Config Model App Bind

esp_ble_mesh_state_change_cfg_model_app_unbind_t **mod_app_unbind**

Config Model App Unbind

esp_ble_mesh_state_change_cfg_kr_phase_set_t **kr_phase_set**

Config Key Refresh Phase Set

union esp_ble_mesh_cfg_server_cb_value_t

#include <esp_ble_mesh_config_model_api.h> Configuration Server model callback value union.

Public Members

esp_ble_mesh_cfg_server_state_change_t **state_change**

ESP_BLE_MESH_CFG_SERVER_STATE_CHANGE_EVT

Structures

struct esp_ble_mesh_cfg_srv

Configuration Server Model context

Public Members

esp_ble_mesh_model_t ***model**

Pointer to Configuration Server Model

uint8_t **net_transmit**

Network Transmit state

uint8_t **relay**

Relay Mode state

uint8_t **relay_retransmit**

Relay Retransmit state

uint8_t **beacon**

Secure Network Beacon state

uint8_t **gatt_proxy**

GATT Proxy state

uint8_t **friend_state**

Friend state

uint8_t **default_ttl**

Default TTL

struct k_delayed_work **timer**

Heartbeat Publication timer

uint16_t **dst**

Destination address for Heartbeat messages

uint16_t **count**

Number of Heartbeat messages to be sent

Number of Heartbeat messages received

uint8_t **period**

Period for sending Heartbeat messages

uint8_t **ttl**

TTL to be used when sending Heartbeat messages

uint16_t **feature**

Bit field indicating features that trigger Heartbeat messages when changed

uint16_t **net_idx**

NetKey Index used by Heartbeat Publication

struct *esp_ble_mesh_cfg_srv*::[**anonymous**] **heartbeat_pub**

Heartbeat Publication

int64_t **expiry**

Timestamp when Heartbeat subscription period is expired

uint16_t **src**

Source address for Heartbeat messages

uint8_t **min_hops**

Minimum hops when receiving Heartbeat messages

uint8_t **max_hops**

Maximum hops when receiving Heartbeat messages

esp_ble_mesh_cb_t **heartbeat_recv_cb**

Optional heartbeat subscription tracking function

struct *esp_ble_mesh_cfg_srv*::[**anonymous**] **heartbeat_sub**

Heartbeat Subscription

struct esp_ble_mesh_cfg_composition_data_get_t

Parameters of Config Composition Data Get.

Public Members

uint8_t **page**

Page number of the Composition Data.

struct esp_ble_mesh_cfg_model_pub_get_t

Parameters of Config Model Publication Get.

Public Members

uint16_t **element_addr**

The element address

uint16_t **model_id**

The model id

uint16_t **company_id**

The company id, if not a vendor model, shall set to 0xFFFF

struct esp_ble_mesh_cfg_sig_model_sub_get_t

Parameters of Config SIG Model Subscription Get.

Public Members

uint16_t **element_addr**
The element address

uint16_t **model_id**
The model id

struct esp_ble_mesh_cfg_vnd_model_sub_get_t
Parameters of Config Vendor Model Subscription Get.

Public Members

uint16_t **element_addr**
The element address

uint16_t **model_id**
The model id

uint16_t **company_id**
The company id, if not a vendor model, shall set to 0xFFFF

struct esp_ble_mesh_cfg_app_key_get_t
Parameters of Config AppKey Get.

Public Members

uint16_t **net_idx**
The network key index

struct esp_ble_mesh_cfg_node_identity_get_t
Parameters of Config Node Identity Get.

Public Members

uint16_t **net_idx**
The network key index

struct esp_ble_mesh_cfg_sig_model_app_get_t
Parameters of Config SIG Model App Get.

Public Members

uint16_t **element_addr**
The element address

uint16_t **model_id**

The model id

struct esp_ble_mesh_cfg_vnd_model_app_get_t

Parameters of Config Vendor Model App Get.

Public Members

uint16_t **element_addr**

The element address

uint16_t **model_id**

The model id

uint16_t **company_id**

The company id, if not a vendor model, shall set to 0xFFFF

struct esp_ble_mesh_cfg_kr_phase_get_t

Parameters of Config Key Refresh Phase Get.

Public Members

uint16_t **net_idx**

The network key index

struct esp_ble_mesh_cfg_lpn_polltimeout_get_t

Parameters of Config Low Power Node PollTimeout Get.

Public Members

uint16_t **lpn_addr**

The unicast address of the Low Power node

struct esp_ble_mesh_cfg_beacon_set_t

Parameters of Config Beacon Set.

Public Members

uint8_t **beacon**

New Secure Network Beacon state

struct esp_ble_mesh_cfg_default_ttl_set_t

Parameters of Config Default TTL Set.

Public Members

uint8_t **ttl**

The default TTL state value

struct esp_ble_mesh_cfg_friend_set_t

Parameters of Config Friend Set.

Public Members

uint8_t **friend_state**

The friend state value

struct esp_ble_mesh_cfg_gatt_proxy_set_t

Parameters of Config GATT Proxy Set.

Public Members

uint8_t **gatt_proxy**

The GATT Proxy state value

struct esp_ble_mesh_cfg_relay_set_t

Parameters of Config Relay Set.

Public Members

uint8_t **relay**

The relay value

uint8_t **relay_retransmit**

The relay retransmit value

struct esp_ble_mesh_cfg_net_key_add_t

Parameters of Config NetKey Add.

Public Members

uint16_t **net_idx**

The network key index

uint8_t **net_key**[16]

The network key value

struct esp_ble_mesh_cfg_app_key_add_t

Parameters of Config AppKey Add.

Public Members

uint16_t **net_idx**
The network key index

uint16_t **app_idx**
The app key index

uint8_t **app_key**[16]
The app key value

struct esp_ble_mesh_cfg_model_app_bind_t
Parameters of Config Model App Bind.

Public Members

uint16_t **element_addr**
The element address

uint16_t **model_app_idx**
Index of the app key to bind with the model

uint16_t **model_id**
The model id

uint16_t **company_id**
The company id, if not a vendor model, shall set to 0xFFFF

struct esp_ble_mesh_cfg_model_pub_set_t
Parameters of Config Model Publication Set.

Public Members

uint16_t **element_addr**
The element address

uint16_t **publish_addr**
Value of the publish address

uint16_t **publish_app_idx**
Index of the application key

bool **cred_flag**
Value of the Friendship Credential Flag

uint8_t **publish_ttl**
Default TTL value for the publishing messages

uint8_t **publish_period**

Period for periodic status publishing

uint8_t **publish_retransmit**

Number of retransmissions and number of 50-millisecond steps between retransmissions

uint16_t **model_id**

The model id

uint16_t **company_id**

The company id, if not a vendor model, shall set to 0xFFFF

struct esp_ble_mesh_cfg_model_sub_add_t

Parameters of Config Model Subscription Add.

Public Members

uint16_t **element_addr**

The element address

uint16_t **sub_addr**

The address to be added to the Subscription List

uint16_t **model_id**

The model id

uint16_t **company_id**

The company id, if not a vendor model, shall set to 0xFFFF

struct esp_ble_mesh_cfg_model_sub_delete_t

Parameters of Config Model Subscription Delete.

Public Members

uint16_t **element_addr**

The element address

uint16_t **sub_addr**

The address to be removed from the Subscription List

uint16_t **model_id**

The model id

uint16_t **company_id**

The company id, if not a vendor model, shall set to 0xFFFF

struct esp_ble_mesh_cfg_model_sub_overwrite_t

Parameters of Config Model Subscription Overwrite.

Public Members

uint16_t **element_addr**

The element address

uint16_t **sub_addr**

The address to be added to the Subscription List

uint16_t **model_id**

The model id

uint16_t **company_id**

The company id, if not a vendor model, shall set to 0xFFFF

struct esp_ble_mesh_cfg_model_sub_va_add_t

Parameters of Config Model Subscription Virtual Address Add.

Public Members

uint16_t **element_addr**

The element address

uint8_t **label_uuid**[16]

The Label UUID of the virtual address to be added to the Subscription List

uint16_t **model_id**

The model id

uint16_t **company_id**

The company id, if not a vendor model, shall set to 0xFFFF

struct esp_ble_mesh_cfg_model_sub_va_delete_t

Parameters of Config Model Subscription Virtual Address Delete.

Public Members

uint16_t **element_addr**

The element address

uint8_t **label_uuid**[16]

The Label UUID of the virtual address to be removed from the Subscription List

uint16_t **model_id**

The model id

uint16_t **company_id**

The company id, if not a vendor model, shall set to 0xFFFF

struct esp_ble_mesh_cfg_model_sub_va_overwrite_t

Parameters of Config Model Subscription Virtual Address Overwrite.

Public Members

uint16_t **element_addr**

The element address

uint8_t **label_uuid**[16]

The Label UUID of the virtual address to be added to the Subscription List

uint16_t **model_id**

The model id

uint16_t **company_id**

The company id, if not a vendor model, shall set to 0xFFFF

struct esp_ble_mesh_cfg_model_pub_va_set_t

Parameters of Config Model Publication Virtual Address Set.

Public Members

uint16_t **element_addr**

The element address

uint8_t **label_uuid**[16]

Value of the Label UUID publish address

uint16_t **publish_app_idx**

Index of the application key

bool **cred_flag**

Value of the Friendship Credential Flag

uint8_t **publish_ttl**

Default TTL value for the publishing messages

uint8_t **publish_period**

Period for periodic status publishing

uint8_t **publish_retransmit**

Number of retransmissions and number of 50-millisecond steps between retransmissions

uint16_t **model_id**

The model id

uint16_t **company_id**

The company id, if not a vendor model, shall set to 0xFFFF

struct esp_ble_mesh_cfg_model_sub_delete_all_t

Parameters of Config Model Subscription Delete All.

Public Members

uint16_t **element_addr**

The element address

uint16_t **model_id**

The model id

uint16_t **company_id**

The company id, if not a vendor model, shall set to 0xFFFF

struct esp_ble_mesh_cfg_net_key_update_t

Parameters of Config NetKey Update.

Public Members

uint16_t **net_idx**

The network key index

uint8_t **net_key**[16]

The network key value

struct esp_ble_mesh_cfg_net_key_delete_t

Parameters of Config NetKey Delete.

Public Members

uint16_t **net_idx**

The network key index

struct esp_ble_mesh_cfg_app_key_update_t

Parameters of Config AppKey Update.

Public Members

uint16_t **net_idx**

The network key index

uint16_t **app_idx**

The app key index

uint8_t **app_key**[16]

The app key value

struct esp_ble_mesh_cfg_app_key_delete_t

Parameters of Config AppKey Delete.

Public Members

uint16_t **net_idx**

The network key index

uint16_t **app_idx**

The app key index

struct esp_ble_mesh_cfg_node_identity_set_t

Parameters of Config Node Identity Set.

Public Members

uint16_t **net_idx**

The network key index

uint8_t **identity**

New Node Identity state

struct esp_ble_mesh_cfg_model_app_unbind_t

Parameters of Config Model App Unbind.

Public Members

uint16_t **element_addr**

The element address

uint16_t **model_app_idx**

Index of the app key to bind with the model

uint16_t **model_id**

The model id

uint16_t **company_id**

The company id, if not a vendor model, shall set to 0xFFFF

struct esp_ble_mesh_cfg_kr_phase_set_t

Parameters of Config Key Refresh Phase Set.

Public Members

uint16_t **net_idx**

The network key index

uint8_t **transition**

New Key Refresh Phase Transition

struct esp_ble_mesh_cfg_net_transmit_set_t

Parameters of Config Network Transmit Set.

Public Members

uint8_t **net_transmit**

Network Transmit State

struct esp_ble_mesh_cfg_heartbeat_pub_set_t

Parameters of Config Model Heartbeat Publication Set.

Public Members

uint16_t **dst**

Destination address for Heartbeat messages

uint8_t **count**

Number of Heartbeat messages to be sent

uint8_t **period**

Period for sending Heartbeat messages

uint8_t **ttl**

TTL to be used when sending Heartbeat messages

uint16_t **feature**

Bit field indicating features that trigger Heartbeat messages when changed

uint16_t **net_idx**

NetKey Index

struct esp_ble_mesh_cfg_heartbeat_sub_set_t

Parameters of Config Model Heartbeat Subscription Set.

Public Members

uint16_t **src**

Source address for Heartbeat messages

uint16_t **dst**

Destination address for Heartbeat messages

uint8_t **period**

Period for receiving Heartbeat messages

struct esp_ble_mesh_cfg_beacon_status_cb_t

Parameter of Config Beacon Status

Public Members

uint8_t **beacon**

Secure Network Beacon state value

struct esp_ble_mesh_cfg_comp_data_status_cb_t

Parameters of Config Composition Data Status

Public Members

uint8_t **page**

Page number of the Composition Data

struct net_buf_simple ***composition_data**

Pointer to Composition Data for the identified page

struct esp_ble_mesh_cfg_default_ttl_status_cb_t

Parameter of Config Default TTL Status

Public Members

uint8_t **default_ttl**

Default TTL state value

struct esp_ble_mesh_cfg_gatt_proxy_status_cb_t

Parameter of Config GATT Proxy Status

Public Members

uint8_t **gatt_proxy**

GATT Proxy state value

struct esp_ble_mesh_cfg_relay_status_cb_t

Parameters of Config Relay Status

Public Members

uint8_t **relay**

Relay state value

uint8_t **retransmit**

Relay retransmit value(number of retransmissions and number of 10-millisecond steps between retransmissions)

struct esp_ble_mesh_cfg_model_pub_status_cb_t

Parameters of Config Model Publication Status

Public Members

uint8_t **status**

Status Code for the request message

uint16_t **element_addr**

Address of the element

uint16_t **publish_addr**

Value of the publish address

uint16_t **app_idx**

Index of the application key

bool **cred_flag**

Value of the Friendship Credential Flag

uint8_t **ttl**

Default TTL value for the outgoing messages

uint8_t **period**

Period for periodic status publishing

uint8_t **transmit**

Number of retransmissions and number of 50-millisecond steps between retransmissions

uint16_t **company_id**

Company ID

uint16_t **model_id**

Model ID

struct esp_ble_mesh_cfg_model_sub_status_cb_t

Parameters of Config Model Subscription Status

Public Members

uint8_t **status**

Status Code for the request message

uint16_t **element_addr**

Address of the element

uint16_t **sub_addr**
Value of the address

uint16_t **company_id**
Company ID

uint16_t **model_id**
Model ID

struct esp_ble_mesh_cfg_net_key_status_cb_t
Parameters of Config NetKey Status

Public Members

uint8_t **status**
Status Code for the request message

uint16_t **net_idx**
Index of the NetKey

struct esp_ble_mesh_cfg_app_key_status_cb_t
Parameters of Config AppKey Status

Public Members

uint8_t **status**
Status Code for the request message

uint16_t **net_idx**
Index of the NetKey

uint16_t **app_idx**
Index of the application key

struct esp_ble_mesh_cfg_mod_app_status_cb_t
Parameters of Config Model App Status

Public Members

uint8_t **status**
Status Code for the request message

uint16_t **element_addr**
Address of the element

uint16_t **app_idx**
Index of the application key

uint16_t **company_id**

Company ID

uint16_t **model_id**

Model ID

struct esp_ble_mesh_cfg_friend_status_cb_t

Parameter of Config Friend Status

Public Members

uint8_t **friend_state**

Friend state value

struct esp_ble_mesh_cfg_hb_pub_status_cb_t

Parameters of Config Heartbeat Publication Status

Public Members

uint8_t **status**

Status Code for the request message

uint16_t **dst**

Destination address for Heartbeat messages

uint8_t **count**

Number of Heartbeat messages remaining to be sent

uint8_t **period**

Period for sending Heartbeat messages

uint8_t **ttl**

TTL to be used when sending Heartbeat messages

uint16_t **features**

Features that trigger Heartbeat messages when changed

uint16_t **net_idx**

Index of the NetKey

struct esp_ble_mesh_cfg_hb_sub_status_cb_t

Parameters of Config Heartbeat Subscription Status

Public Members

uint8_t **status**

Status Code for the request message

uint16_t **src**

Source address for Heartbeat messages

uint16_t **dst**

Destination address for Heartbeat messages

uint8_t **period**

Remaining Period for processing Heartbeat messages

uint8_t **count**

Number of Heartbeat messages received

uint8_t **min_hops**

Minimum hops when receiving Heartbeat messages

uint8_t **max_hops**

Maximum hops when receiving Heartbeat messages

struct esp_ble_mesh_cfg_net_trans_status_cb_t

Parameters of Config Network Transmit Status

Public Members

uint8_t **net_trans_count**

Number of transmissions for each Network PDU originating from the node

uint8_t **net_trans_step**

Maximum hops when receiving Heartbeat messages

struct esp_ble_mesh_cfg_model_sub_list_cb_t

Parameters of Config SIG/Vendor Subscription List

Public Members

uint8_t **status**

Status Code for the request message

uint16_t **element_addr**

Address of the element

uint16_t **company_id**

Company ID

uint16_t **model_id**

Model ID

struct net_buf_simple ***sub_addr**

A block of all addresses from the Subscription List

struct esp_ble_mesh_cfg_net_key_list_cb_t

Parameter of Config NetKey List

Public Members

struct net_buf_simple ***net_idx**

A list of NetKey Indexes known to the node

struct esp_ble_mesh_cfg_app_key_list_cb_t

Parameters of Config AppKey List

Public Members

uint8_t **status**

Status Code for the request message

uint16_t **net_idx**

NetKey Index of the NetKey that the AppKeys are bound to

struct net_buf_simple ***app_idx**

A list of AppKey indexes that are bound to the NetKey identified by NetKeyIndex

struct esp_ble_mesh_cfg_node_id_status_cb_t

Parameters of Config Node Identity Status

Public Members

uint8_t **status**

Status Code for the request message

uint16_t **net_idx**

Index of the NetKey

uint8_t **identity**

Node Identity state

struct esp_ble_mesh_cfg_model_app_list_cb_t

Parameters of Config SIG/Vendor Model App List

Public Members

uint8_t **status**

Status Code for the request message

uint16_t **element_addr**

Address of the element

uint16_t **company_id**

Company ID

uint16_t **model_id**

Model ID

struct net_buf_simple ***app_idx**

All AppKey indexes bound to the Model

struct esp_ble_mesh_cfg_kr_phase_status_cb_t

Parameters of Config Key Refresh Phase Status

Public Members

uint8_t **status**

Status Code for the request message

uint16_t **net_idx**

Index of the NetKey

uint8_t **phase**

Key Refresh Phase state

struct esp_ble_mesh_cfg_lpn_pollto_status_cb_t

Parameters of Config Low Power Node PollTimeout Status

Public Members

uint16_t **lpn_addr**

The unicast address of the Low Power node

int32_t **poll_timeout**

The current value of the PollTimeout timer of the Low Power node

struct esp_ble_mesh_cfg_client_cb_param_t

Configuration Client Model callback parameters

Public Members

int **error_code**

Appropriate error code

esp_ble_mesh_client_common_param_t ***params**

The client common parameters

esp_ble_mesh_cfg_client_common_cb_param_t **status_cb**

The config status message callback values

`struct esp_ble_mesh_state_change_cfg_mod_pub_set_t`

Configuration Server model related context.

Public Members

uint16_t **element_addr**

Element Address

uint16_t **pub_addr**

Publish Address

uint16_t **app_idx**

AppKey Index

bool **cred_flag**

Friendship Credential Flag

uint8_t **pub_ttl**

Publish TTL

uint8_t **pub_period**

Publish Period

uint8_t **pub_retransmit**

Publish Retransmit

uint16_t **company_id**

Company ID

uint16_t **model_id**

Model ID

`struct esp_ble_mesh_state_change_cfg_model_sub_add_t`

Parameters of Config Model Subscription Add

Public Members

uint16_t **element_addr**

Element Address

uint16_t **sub_addr**

Subscription Address

uint16_t **company_id**

Company ID

uint16_t **model_id**

Model ID

struct esp_ble_mesh_state_change_cfg_model_sub_delete_t

Parameters of Config Model Subscription Delete

Public Members

uint16_t **element_addr**

Element Address

uint16_t **sub_addr**

Subscription Address

uint16_t **company_id**

Company ID

uint16_t **model_id**

Model ID

struct esp_ble_mesh_state_change_cfg_netkey_add_t

Parameters of Config NetKey Add

Public Members

uint16_t **net_idx**

NetKey Index

uint8_t **net_key**[16]

NetKey

struct esp_ble_mesh_state_change_cfg_netkey_update_t

Parameters of Config NetKey Update

Public Members

uint16_t **net_idx**

NetKey Index

uint8_t **net_key**[16]

NetKey

struct esp_ble_mesh_state_change_cfg_netkey_delete_t

Parameter of Config NetKey Delete

Public Members

uint16_t **net_idx**

NetKey Index

struct esp_ble_mesh_state_change_cfg_appkey_add_t

Parameters of Config AppKey Add

Public Members

uint16_t **net_idx**

NetKey Index

uint16_t **app_idx**

AppKey Index

uint8_t **app_key**[16]

AppKey

struct esp_ble_mesh_state_change_cfg_appkey_update_t

Parameters of Config AppKey Update

Public Members

uint16_t **net_idx**

NetKey Index

uint16_t **app_idx**

AppKey Index

uint8_t **app_key**[16]

AppKey

struct esp_ble_mesh_state_change_cfg_appkey_delete_t

Parameters of Config AppKey Delete

Public Members

uint16_t **net_idx**

NetKey Index

uint16_t **app_idx**

AppKey Index

struct esp_ble_mesh_state_change_cfg_model_app_bind_t

Parameters of Config Model App Bind

Public Members

uint16_t **element_addr**

Element Address

uint16_t **app_idx**
AppKey Index

uint16_t **company_id**
Company ID

uint16_t **model_id**
Model ID

struct esp_ble_mesh_state_change_cfg_model_app_unbind_t
Parameters of Config Model App Unbind

Public Members

uint16_t **element_addr**
Element Address

uint16_t **app_idx**
AppKey Index

uint16_t **company_id**
Company ID

uint16_t **model_id**
Model ID

struct esp_ble_mesh_state_change_cfg_kr_phase_set_t
Parameters of Config Key Refresh Phase Set

Public Members

uint16_t **net_idx**
NetKey Index

uint8_t **kr_phase**
New Key Refresh Phase Transition

struct esp_ble_mesh_cfg_server_cb_param_t
Configuration Server model callback parameters

Public Members

esp_ble_mesh_model_t ***model**
Pointer to the server model structure

esp_ble_mesh_msg_ctx_t **ctx**
Context of the received message

esp_ble_mesh_cfg_server_cb_value_t **value**

Value of the received configuration messages

Macros

ESP_BLE_MESH_MODEL_CFG_SRV(srv_data)

Define a new Config Server Model.

Note The Config Server Model can only be included by a Primary Element.

Return New Config Server Model instance.

Parameters

- `srv_data`: Pointer to a unique Config Server Model user_data.

ESP_BLE_MESH_MODEL_CFG_CLI(cli_data)

Define a new Config Client Model.

Note The Config Client Model can only be included by a Primary Element.

Return New Config Client Model instance.

Parameters

- `cli_data`: Pointer to a unique struct *esp_ble_mesh_client_t*.

Type Definitions

typedef struct *esp_ble_mesh_cfg_srv* **esp_ble_mesh_cfg_srv_t**

Configuration Server Model context

typedef void (***esp_ble_mesh_cfg_client_cb_t**)(*esp_ble_mesh_cfg_client_cb_event_t* event, *esp_ble_mesh_cfg_client_cb_param_t* *param)

Bluetooth Mesh Config Client and Server Model functions.

Configuration Client Model callback function type

Parameters

- `event`: Event type
- `param`: Pointer to callback parameter

typedef void (***esp_ble_mesh_cfg_server_cb_t**)(*esp_ble_mesh_cfg_server_cb_event_t* event, *esp_ble_mesh_cfg_server_cb_param_t* *param)

Configuration Server Model callback function type.

Parameters

- `event`: Event type
- `param`: Pointer to callback parameter

Enumerations

enum esp_ble_mesh_cfg_client_cb_event_t

This enum value is the event of Configuration Client Model

Values:

ESP_BLE_MESH_CFG_CLIENT_GET_STATE_EVT

ESP_BLE_MESH_CFG_CLIENT_SET_STATE_EVT

ESP_BLE_MESH_CFG_CLIENT_PUBLISH_EVT

ESP_BLE_MESH_CFG_CLIENT_TIMEOUT_EVT

ESP_BLE_MESH_CFG_CLIENT_EVT_MAX

enum esp_ble_mesh_cfg_server_cb_event_t

This enum value is the event of Configuration Server model

Values:

ESP_BLE_MESH_CFG_SERVER_STATE_CHANGE_EVT

ESP_BLE_MESH_CFG_SERVER_EVT_MAX

Health Client/Server Models

Header File

- bt/esp_ble_mesh/api/models/include/esp_ble_mesh_generic_model_api.h

Functions

esp_err_t **esp_ble_mesh_register_generic_client_callback**(*esp_ble_mesh_generic_client_cb_t callback*)

Register BLE Mesh Generic Client Model callback.

Return ESP_OK on success or error code otherwise.

Parameters

- `callback`: Pointer to the callback function.

esp_err_t **esp_ble_mesh_generic_client_get_state**(*esp_ble_mesh_client_common_param_t* **params*, *esp_ble_mesh_generic_client_get_state_t* **get_state*)

Get the value of Generic Server Model states using the Generic Client Model get messages.

Note If you want to find the opcodes and corresponding meanings accepted by this API, please refer to esp_ble_mesh_generic_message_opcode_t in esp_ble_mesh_defs.h

Return ESP_OK on success or error code otherwise.

Parameters

- `params`: Pointer to BLE Mesh common client parameters.
- `get_state`: Pointer to generic get message value. Shall not be set to NULL.

esp_err_t **esp_ble_mesh_generic_client_set_state**(*esp_ble_mesh_client_common_param_t* **params*, *esp_ble_mesh_generic_client_set_state_t* **set_state*)

Set the value of Generic Server Model states using the Generic Client Model set messages.

Note If you want to find the opcodes and corresponding meanings accepted by this API, please refer to esp_ble_mesh_generic_message_opcode_t in esp_ble_mesh_defs.h

Return ESP_OK on success or error code otherwise.

Parameters

- `params`: Pointer to BLE Mesh common client parameters.
- `set_state`: Pointer to generic set message value. Shall not be set to NULL.

esp_err_t **esp_ble_mesh_register_generic_server_callback**(*esp_ble_mesh_generic_server_cb_t* *callback*)

Register BLE Mesh Generic Server Model callback.

Return ESP_OK on success or error code otherwise.

Parameters

- `callback`: Pointer to the callback function.

Unions

union esp_ble_mesh_generic_client_get_state_t

#include <esp_ble_mesh_generic_model_api.h> Generic Client Model get message union.

Public Members

esp_ble_mesh_gen_user_property_get_t **user_property_get**
For ESP_BLE_MESH_MODEL_OP_GEN_USER_PROPERTY_GET

esp_ble_mesh_gen_admin_property_get_t **admin_property_get**
For ESP_BLE_MESH_MODEL_OP_GEN_ADMIN_PROPERTY_GET

esp_ble_mesh_gen_manufacturer_property_get_t **manufacturer_property_get**
For ESP_BLE_MESH_MODEL_OP_GEN_MANUFACTURER_PROPERTY_SET

esp_ble_mesh_gen_client_properties_get_t **client_properties_get**
For ESP_BLE_MESH_MODEL_OP_GEN_CLIENT_PROPERTIES_GET

union esp_ble_mesh_generic_client_set_state_t
#include <esp_ble_mesh_generic_model_api.h> Generic Client Model set message union.

Public Members

esp_ble_mesh_gen_onoff_set_t **onoff_set**
For ESP_BLE_MESH_MODEL_OP_GEN_ONOFF_SET & ESP_BLE_MESH_MODEL_OP_GEN_ONOFF_SET_UNACK

esp_ble_mesh_gen_level_set_t **level_set**
For ESP_BLE_MESH_MODEL_OP_GEN_LEVEL_SET & ESP_BLE_MESH_MODEL_OP_GEN_LEVEL

esp_ble_mesh_gen_delta_set_t **delta_set**
For ESP_BLE_MESH_MODEL_OP_GEN_DELTA_SET & ESP_BLE_MESH_MODEL_OP_GEN_DELTA

esp_ble_mesh_gen_move_set_t **move_set**
For ESP_BLE_MESH_MODEL_OP_GEN_MOVE_SET & ESP_BLE_MESH_MODEL_OP_GEN_MOVE

esp_ble_mesh_gen_def_trans_time_set_t **def_trans_time_set**
For ESP_BLE_MESH_MODEL_OP_GEN_DEF_TRANS_TIME_SET & ESP_BLE_MESH_MODEL_OP_GEN_DEF_TRANS_TIME_SET_UNACK

esp_ble_mesh_gen_onpowerup_set_t **power_set**
For ESP_BLE_MESH_MODEL_OP_GEN_ONPOWERUP_SET & ESP_BLE_MESH_MODEL_OP_GEN_ONPOWERUP_SET_UNACK

esp_ble_mesh_gen_power_level_set_t **power_level_set**
For ESP_BLE_MESH_MODEL_OP_GEN_POWER_LEVEL_SET & ESP_BLE_MESH_MODEL_OP_GEN_POWER_LEVEL_SET_UNACK

esp_ble_mesh_gen_power_default_set_t **power_default_set**
For ESP_BLE_MESH_MODEL_OP_GEN_POWER_DEFAULT_SET & ESP_BLE_MESH_MODEL_OP_GEN_POWER_DEFAULT_SET_UNACK

esp_ble_mesh_gen_power_range_set_t **power_range_set**
For ESP_BLE_MESH_MODEL_OP_GEN_POWER_RANGE_SET & ESP_BLE_MESH_MODEL_OP_GEN_POWER_RANGE_SET_UNACK

esp_ble_mesh_gen_loc_global_set_t **loc_global_set**
For ESP_BLE_MESH_MODEL_OP_GEN_LOC_GLOBAL_SET & ESP_BLE_MESH_MODEL_OP_GEN_LOC_GLOBAL_SET_UNACK

esp_ble_mesh_gen_loc_local_set_t **loc_local_set**
For ESP_BLE_MESH_MODEL_OP_GEN_LOC_LOCAL_SET & ESP_BLE_MESH_MODEL_OP_GEN_LOC_LOCAL_SET_UNACK

esp_ble_mesh_gen_user_property_set_t **user_property_set**
For ESP_BLE_MESH_MODEL_OP_GEN_USER_PROPERTY_SET & ESP_BLE_MESH_MODEL_OP_GEN_USER_PROPERTY_SET_UNACK

esp_ble_mesh_gen_admin_property_set_t **admin_property_set**
For ESP_BLE_MESH_MODEL_OP_GEN_ADMIN_PROPERTY_SET & ESP_BLE_MESH_MODEL_OP_GEN_ADMIN_PROPERTY_SET_UNACK

esp_ble_mesh_gen_manufacturer_property_set_t **manufacturer_property_set**
For ESP_BLE_MESH_MODEL_OP_GEN_MANUFACTURER_PROPERTY_SET & ESP_BLE_MESH_MODEL_OP_GEN_MANUFACTURER_PROPERTY_SET_UNACK

union esp_ble_mesh_gen_client_status_cb_t

#include <esp_ble_mesh_generic_model_api.h> Generic Client Model received message union.

Public Members

esp_ble_mesh_gen_onoff_status_cb_t **onoff_status**
For ESP_BLE_MESH_MODEL_OP_GEN_ONOFF_STATUS

esp_ble_mesh_gen_level_status_cb_t **level_status**
For ESP_BLE_MESH_MODEL_OP_GEN_LEVEL_STATUS

esp_ble_mesh_gen_def_trans_time_status_cb_t **def_trans_time_status**
For ESP_BLE_MESH_MODEL_OP_GEN_DEF_TRANS_TIME_STATUS

esp_ble_mesh_gen_onpowerup_status_cb_t **onpowerup_status**
For ESP_BLE_MESH_MODEL_OP_GEN_ONPOWERUP_STATUS

esp_ble_mesh_gen_power_level_status_cb_t **power_level_status**
For ESP_BLE_MESH_MODEL_OP_GEN_POWER_LEVEL_STATUS

esp_ble_mesh_gen_power_last_status_cb_t **power_last_status**
For ESP_BLE_MESH_MODEL_OP_GEN_POWER_LAST_STATUS

esp_ble_mesh_gen_power_default_status_cb_t **power_default_status**
For ESP_BLE_MESH_MODEL_OP_GEN_POWER_DEFAULT_STATUS

esp_ble_mesh_gen_power_range_status_cb_t **power_range_status**
For ESP_BLE_MESH_MODEL_OP_GEN_POWER_RANGE_STATUS

esp_ble_mesh_gen_battery_status_cb_t **battery_status**
For ESP_BLE_MESH_MODEL_OP_GEN_BATTERY_STATUS

esp_ble_mesh_gen_loc_global_status_cb_t **location_global_status**
For ESP_BLE_MESH_MODEL_OP_GEN_LOC_GLOBAL_STATUS

esp_ble_mesh_gen_loc_local_status_cb_t **location_local_status**
ESP_BLE_MESH_MODEL_OP_GEN_LOC_LOCAL_STATUS

esp_ble_mesh_gen_user_properties_status_cb_t **user_properties_status**
ESP_BLE_MESH_MODEL_OP_GEN_USER_PROPERTIES_STATUS

esp_ble_mesh_gen_user_property_status_cb_t **user_property_status**
ESP_BLE_MESH_MODEL_OP_GEN_USER_PROPERTY_STATUS

esp_ble_mesh_gen_admin_properties_status_cb_t **admin_properties_status**
ESP_BLE_MESH_MODEL_OP_GEN_ADMIN_PROPERTIES_STATUS

esp_ble_mesh_gen_admin_property_status_cb_t **admin_property_status**
ESP_BLE_MESH_MODEL_OP_GEN_ADMIN_PROPERTY_STATUS

esp_ble_mesh_gen_manufacturer_properties_status_cb_t **manufacturer_properties_status**
ESP_BLE_MESH_MODEL_OP_GEN_MANUFACTURER_PROPERTIES_STATUS

esp_ble_mesh_gen_manufacturer_property_status_cb_t **manufacturer_property_status**
ESP_BLE_MESH_MODEL_OP_GEN_MANUFACTURER_PROPERTY_STATUS

esp_ble_mesh_gen_client_properties_status_cb_t **client_properties_status**
ESP_BLE_MESH_MODEL_OP_GEN_CLIENT_PROPERTIES_STATUS

union esp_ble_mesh_generic_server_state_change_t
#include <esp_ble_mesh_generic_model_api.h> Generic Server Model state change value union.

Public Members

esp_ble_mesh_state_change_gen_onoff_set_t **onoff_set**
The recv_op in ctx can be used to decide which state is changed.Generic OnOff Set

esp_ble_mesh_state_change_gen_level_set_t **level_set**
Generic Level Set

esp_ble_mesh_state_change_gen_delta_set_t **delta_set**
Generic Delta Set

esp_ble_mesh_state_change_gen_move_set_t **move_set**
Generic Move Set

esp_ble_mesh_state_change_gen_def_trans_time_set_t **def_trans_time_set**
Generic Default Transition Time Set

esp_ble_mesh_state_change_gen_onpowerup_set_t **onpowerup_set**
Generic OnPowerUp Set

esp_ble_mesh_state_change_gen_power_level_set_t **power_level_set**
Generic Power Level Set

esp_ble_mesh_state_change_gen_power_default_set_t **power_default_set**
Generic Power Default Set

esp_ble_mesh_state_change_gen_power_range_set_t **power_range_set**
Generic Power Range Set

esp_ble_mesh_state_change_gen_loc_global_set_t **loc_global_set**
Generic Location Global Set

esp_ble_mesh_state_change_gen_loc_local_set_t **loc_local_set**
Generic Location Local Set

esp_ble_mesh_state_change_gen_user_property_set_t **user_property_set**
Generic User Property Set

esp_ble_mesh_state_change_gen_admin_property_set_t **admin_property_set**
Generic Admin Property Set

esp_ble_mesh_state_change_gen_manu_property_set_t **manu_property_set**
Generic Manufacturer Property Set

union esp_ble_mesh_generic_server_recv_get_msg_t
#include <esp_ble_mesh_generic_model_api.h> Generic Server Model received get message union.

Public Members

esp_ble_mesh_server_recv_gen_user_property_get_t **user_property**
Generic User Property Get

esp_ble_mesh_server_recv_gen_admin_property_get_t **admin_property**
Generic Admin Property Get

esp_ble_mesh_server_recv_gen_manufacturer_property_get_t **manu_property**
Generic Manufacturer Property Get

esp_ble_mesh_server_recv_gen_client_properties_get_t **client_properties**
Generic Client Properties Get

union esp_ble_mesh_generic_server_recv_set_msg_t
#include <esp_ble_mesh_generic_model_api.h> Generic Server Model received set message union.

Public Members

esp_ble_mesh_server_recv_gen_onoff_set_t **onoff**

Generic OnOff Set/Generic OnOff Set Unack

esp_ble_mesh_server_recv_gen_level_set_t **level**

Generic Level Set/Generic Level Set Unack

esp_ble_mesh_server_recv_gen_delta_set_t **delta**

Generic Delta Set/Generic Delta Set Unack

esp_ble_mesh_server_recv_gen_move_set_t **move**

Generic Move Set/Generic Move Set Unack

esp_ble_mesh_server_recv_gen_def_trans_time_set_t **def_trans_time**

Generic Default Transition Time Set/Generic Default Transition Time Set Unack

esp_ble_mesh_server_recv_gen_onpowerup_set_t **onpowerup**

Generic OnPowerUp Set/Generic OnPowerUp Set Unack

esp_ble_mesh_server_recv_gen_power_level_set_t **power_level**

Generic Power Level Set/Generic Power Level Set Unack

esp_ble_mesh_server_recv_gen_power_default_set_t **power_default**

Generic Power Default Set/Generic Power Default Set Unack

esp_ble_mesh_server_recv_gen_power_range_set_t **power_range**

Generic Power Range Set/Generic Power Range Set Unack

esp_ble_mesh_server_recv_gen_loc_global_set_t **location_global**

Generic Location Global Set/Generic Location Global Set Unack

esp_ble_mesh_server_recv_gen_loc_local_set_t **location_local**

Generic Location Local Set/Generic Location Local Set Unack

esp_ble_mesh_server_recv_gen_user_property_set_t **user_property**

Generic User Property Set/Generic User Property Set Unack

esp_ble_mesh_server_recv_gen_admin_property_set_t **admin_property**

Generic Admin Property Set/Generic Admin Property Set Unack

esp_ble_mesh_server_recv_gen_manufacturer_property_set_t **manu_property**

Generic Manufacturer Property Set/Generic Manufacturer Property Set Unack

union esp_ble_mesh_generic_server_cb_value_t

#include <esp_ble_mesh_generic_model_api.h> Generic Server Model callback value union.

Public Members

esp_ble_mesh_generic_server_state_change_t **state_change**
ESP_BLE_MESH_GENERIC_SERVER_STATE_CHANGE_EVT

esp_ble_mesh_generic_server_recv_get_msg_t **get**
ESP_BLE_MESH_GENERIC_SERVER_RECV_GET_MSG_EVT

esp_ble_mesh_generic_server_recv_set_msg_t **set**
ESP_BLE_MESH_GENERIC_SERVER_RECV_SET_MSG_EVT

Structures

struct esp_ble_mesh_gen_onoff_set_t
Bluetooth Mesh Generic Client Model Get and Set parameters structure.

Parameters of Generic OnOff Set.

Public Members

bool **op_en**
Indicate if optional parameters are included

uint8_t **onoff**
Target value of Generic OnOff state

uint8_t **tid**
Transaction ID

uint8_t **trans_time**
Time to complete state transition (optional)

uint8_t **delay**
Indicate message execution delay (C.1)

struct esp_ble_mesh_gen_level_set_t
Parameters of Generic Level Set.

Public Members

bool **op_en**
Indicate if optional parameters are included

int16_t **level**
Target value of Generic Level state

uint8_t **tid**
Transaction ID

uint8_t **trans_time**

Time to complete state transition (optional)

uint8_t **delay**

Indicate message execution delay (C.1)

struct esp_ble_mesh_gen_delta_set_t

Parameters of Generic Delta Set.

Public Members

bool **op_en**

Indicate if optional parameters are included

int32_t **level**

Delta change of Generic Level state

uint8_t **tid**

Transaction ID

uint8_t **trans_time**

Time to complete state transition (optional)

uint8_t **delay**

Indicate message execution delay (C.1)

struct esp_ble_mesh_gen_move_set_t

Parameters of Generic Move Set.

Public Members

bool **op_en**

Indicate if optional parameters are included

int16_t **delta_level**

Delta Level step to calculate Move speed for Generic Level state

uint8_t **tid**

Transaction ID

uint8_t **trans_time**

Time to complete state transition (optional)

uint8_t **delay**

Indicate message execution delay (C.1)

struct esp_ble_mesh_gen_def_trans_time_set_t

Parameter of Generic Default Transition Time Set.

Public Members

uint8_t **trans_time**

The value of the Generic Default Transition Time state

struct esp_ble_mesh_gen_onpowerup_set_t

Parameter of Generic OnPowerUp Set.

Public Members

uint8_t **onpowerup**

The value of the Generic OnPowerUp state

struct esp_ble_mesh_gen_power_level_set_t

Parameters of Generic Power Level Set.

Public Members

bool **op_en**

Indicate if optional parameters are included

uint16_t **power**

Target value of Generic Power Actual state

uint8_t **tid**

Transaction ID

uint8_t **trans_time**

Time to complete state transition (optional)

uint8_t **delay**

Indicate message execution delay (C.1)

struct esp_ble_mesh_gen_power_default_set_t

Parameter of Generic Power Default Set.

Public Members

uint16_t **power**

The value of the Generic Power Default state

struct esp_ble_mesh_gen_power_range_set_t

Parameters of Generic Power Range Set.

Public Members

uint16_t **range_min**

Value of Range Min field of Generic Power Range state

uint16_t **range_max**

Value of Range Max field of Generic Power Range state

struct esp_ble_mesh_gen_loc_global_set_t

Parameters of Generic Location Global Set.

Public Members

int32_t **global_latitude**

Global Coordinates (Latitude)

int32_t **global_longitude**

Global Coordinates (Longitude)

int16_t **global_altitude**

Global Altitude

struct esp_ble_mesh_gen_loc_local_set_t

Parameters of Generic Location Local Set.

Public Members

int16_t **local_north**

Local Coordinates (North)

int16_t **local_east**

Local Coordinates (East)

int16_t **local_altitude**

Local Altitude

uint8_t **floor_number**

Floor Number

uint16_t **uncertainty**

Uncertainty

struct esp_ble_mesh_gen_user_property_get_t

Parameter of Generic User Property Get.

Public Members

uint16_t **property_id**

Property ID identifying a Generic User Property

struct esp_ble_mesh_gen_user_property_set_t

Parameters of Generic User Property Set.

Public Members

uint16_t **property_id**

Property ID identifying a Generic User Property

struct net_buf_simple ***property_value**

Raw value for the User Property

struct esp_ble_mesh_gen_admin_property_get_t

Parameter of Generic Admin Property Get.

Public Members

uint16_t **property_id**

Property ID identifying a Generic Admin Property

struct esp_ble_mesh_gen_admin_property_set_t

Parameters of Generic Admin Property Set.

Public Members

uint16_t **property_id**

Property ID identifying a Generic Admin Property

uint8_t **user_access**

Enumeration indicating user access

struct net_buf_simple ***property_value**

Raw value for the Admin Property

struct esp_ble_mesh_gen_manufacturer_property_get_t

Parameter of Generic Manufacturer Property Get.

Public Members

uint16_t **property_id**

Property ID identifying a Generic Manufacturer Property

struct esp_ble_mesh_gen_manufacturer_property_set_t

Parameters of Generic Manufacturer Property Set.

Public Members

uint16_t **property_id**

Property ID identifying a Generic Manufacturer Property

uint8_t **user_access**

Enumeration indicating user access

struct esp_ble_mesh_gen_client_properties_get_t

Parameter of Generic Client Properties Get.

Public Members

uint16_t **property_id**

A starting Client Property ID present within an element

struct esp_ble_mesh_gen_onoff_status_cb_t

Bluetooth Mesh Generic Client Model Get and Set callback parameters structure.

Parameters of Generic OnOff Status.

Public Members

bool **op_en**

Indicate if optional parameters are included

uint8_t **present_onoff**

Current value of Generic OnOff state

uint8_t **target_onoff**

Target value of Generic OnOff state (optional)

uint8_t **remain_time**

Time to complete state transition (C.1)

struct esp_ble_mesh_gen_level_status_cb_t

Parameters of Generic Level Status.

Public Members

bool **op_en**

Indicate if optional parameters are included

int16_t **present_level**

Current value of Generic Level state

int16_t **target_level**

Target value of the Generic Level state (optional)

uint8_t **remain_time**

Time to complete state transition (C.1)

struct esp_ble_mesh_gen_def_trans_time_status_cb_t

Parameter of Generic Default Transition Time Status.

Public Members

uint8_t **trans_time**

The value of the Generic Default Transition Time state

struct esp_ble_mesh_gen_onpowerup_status_cb_t

Parameter of Generic OnPowerUp Status.

Public Members

uint8_t **onpowerup**

The value of the Generic OnPowerUp state

struct esp_ble_mesh_gen_power_level_status_cb_t

Parameters of Generic Power Level Status.

Public Members

bool **op_en**

Indicate if optional parameters are included

uint16_t **present_power**

Current value of Generic Power Actual state

uint16_t **target_power**

Target value of Generic Power Actual state (optional)

uint8_t **remain_time**

Time to complete state transition (C.1)

struct esp_ble_mesh_gen_power_last_status_cb_t

Parameter of Generic Power Last Status.

Public Members

uint16_t **power**

The value of the Generic Power Last state

struct esp_ble_mesh_gen_power_default_status_cb_t

Parameter of Generic Power Default Status.

Public Members

uint16_t **power**

The value of the Generic Default Last state

struct esp_ble_mesh_gen_power_range_status_cb_t

Parameters of Generic Power Range Status.

Public Members

uint8_t **status_code**

Status Code for the request message

uint16_t **range_min**

Value of Range Min field of Generic Power Range state

uint16_t **range_max**

Value of Range Max field of Generic Power Range state

struct esp_ble_mesh_gen_battery_status_cb_t

Parameters of Generic Battery Status.

Public Members

u32_t **battery_level**

Value of Generic Battery Level state

u32_t **time_to_discharge**

Value of Generic Battery Time to Discharge state

u32_t **time_to_charge**

Value of Generic Battery Time to Charge state

u32_t **flags**

Value of Generic Battery Flags state

struct esp_ble_mesh_gen_loc_global_status_cb_t

Parameters of Generic Location Global Status.

Public Members

int32_t **global_latitude**
Global Coordinates (Latitude)

int32_t **global_longitude**
Global Coordinates (Longitude)

int16_t **global_altitude**
Global Altitude

struct esp_ble_mesh_gen_loc_local_status_cb_t
Parameters of Generic Location Local Status.

Public Members

int16_t **local_north**
Local Coordinates (North)

int16_t **local_east**
Local Coordinates (East)

int16_t **local_altitude**
Local Altitude

uint8_t **floor_number**
Floor Number

uint16_t **uncertainty**
Uncertainty

struct esp_ble_mesh_gen_user_properties_status_cb_t
Parameter of Generic User Properties Status.

Public Members

struct net_buf_simple ***property_ids**
Buffer contains a sequence of N User Property IDs

struct esp_ble_mesh_gen_user_property_status_cb_t
Parameters of Generic User Property Status.

Public Members

bool **op_en**
Indicate if optional parameters are included

uint16_t **property_id**

Property ID identifying a Generic User Property

uint8_t **user_access**

Enumeration indicating user access (optional)

struct net_buf_simple ***property_value**

Raw value for the User Property (C.1)

struct esp_ble_mesh_gen_admin_properties_status_cb_t

Parameter of Generic Admin Properties Status.

Public Members

struct net_buf_simple ***property_ids**

Buffer contains a sequence of N Admin Property IDs

struct esp_ble_mesh_gen_admin_property_status_cb_t

Parameters of Generic Admin Property Status.

Public Members

bool **op_en**

Indicate if optional parameters are included

uint16_t **property_id**

Property ID identifying a Generic Admin Property

uint8_t **user_access**

Enumeration indicating user access (optional)

struct net_buf_simple ***property_value**

Raw value for the Admin Property (C.1)

struct esp_ble_mesh_gen_manufacturer_properties_status_cb_t

Parameter of Generic Manufacturer Properties Status.

Public Members

struct net_buf_simple ***property_ids**

Buffer contains a sequence of N Manufacturer Property IDs

struct esp_ble_mesh_gen_manufacturer_property_status_cb_t

Parameters of Generic Manufacturer Property Status.

Public Members

bool **op_en**

Indicate if optional parameters are included

uint16_t **property_id**

Property ID identifying a Generic Manufacturer Property

uint8_t **user_access**

Enumeration indicating user access (optional)

struct net_buf_simple ***property_value**

Raw value for the Manufacturer Property (C.1)

struct esp_ble_mesh_gen_client_properties_status_cb_t

Parameter of Generic Client Properties Status.

Public Members

struct net_buf_simple ***property_ids**

Buffer contains a sequence of N Client Property IDs

struct esp_ble_mesh_generic_client_cb_param_t

Generic Client Model callback parameters

Public Members

int **error_code**

Appropriate error code

esp_ble_mesh_client_common_param_t ***params**

The client common parameters.

esp_ble_mesh_gen_client_status_cb_t **status_cb**

The generic status message callback values

struct esp_ble_mesh_gen_onoff_state_t

Parameters of Generic OnOff state

Public Members

uint8_t **onoff**

The present value of the Generic OnOff state

uint8_t **target_onoff**

The target value of the Generic OnOff state

struct esp_ble_mesh_gen_onoff_srv_t

User data of Generic OnOff Server Model

Public Members

esp_ble_mesh_model_t *model

Pointer to the Generic OnOff Server Model. Initialized internally.

esp_ble_mesh_server_rsp_ctrl_t rsp_ctrl

Response control of the server model received messages

esp_ble_mesh_gen_onoff_state_t state

Parameters of the Generic OnOff state

esp_ble_mesh_last_msg_info_t last

Parameters of the last received set message

esp_ble_mesh_state_transition_t transition

Parameters of state transition

struct esp_ble_mesh_gen_level_state_t

Parameters of Generic Level state

Public Members

int16_t level

The present value of the Generic Level state

int16_t target_level

The target value of the Generic Level state

int16_t last_level

When a new transaction starts, level should be set to last_last, and use "level + incoming delta" to calculate the target level. In another word, "last_level" is used to record "level" of the last transaction, and "last_delta" is used to record the previously received delta_level value.The last value of the Generic Level state

int32_t last_delta

The last delta change of the Generic Level state

bool move_start

Indicate if the transition of the Generic Level state has been started

bool positive

Indicate if the transition is positive or negative

struct esp_ble_mesh_gen_level_srv_t

User data of Generic Level Server Model

Public Members

esp_ble_mesh_model_t ***model**

Pointer to the Generic Level Server Model. Initialized internally.

esp_ble_mesh_server_rsp_ctrl_t **rsp_ctrl**

Response control of the server model received messages

esp_ble_mesh_gen_level_state_t **state**

Parameters of the Generic Level state

esp_ble_mesh_last_msg_info_t **last**

Parameters of the last received set message

esp_ble_mesh_state_transition_t **transition**

Parameters of state transition

int32_t **tt_delta_level**

Delta change value of level state transition

struct esp_ble_mesh_gen_def_trans_time_state_t

Parameter of Generic Default Transition Time state

Public Members

uint8_t **trans_time**

The value of the Generic Default Transition Time state

struct esp_ble_mesh_gen_def_trans_time_srv_t

User data of Generic Default Transition Time Server Model

Public Members

esp_ble_mesh_model_t ***model**

Pointer to the Generic Default Transition Time Server Model. Initialized internally.

esp_ble_mesh_server_rsp_ctrl_t **rsp_ctrl**

Response control of the server model received messages

esp_ble_mesh_gen_def_trans_time_state_t **state**

Parameters of the Generic Default Transition Time state

struct esp_ble_mesh_gen_onpowerup_state_t

Parameter of Generic OnPowerUp state

Public Members

uint8_t **onpowerup**

The value of the Generic OnPowerUp state

struct esp_ble_mesh_gen_power_onoff_srv_t

User data of Generic Power OnOff Server Model

Public Members

esp_ble_mesh_model_t ***model**

Pointer to the Generic Power OnOff Server Model. Initialized internally.

esp_ble_mesh_server_rsp_ctrl_t **rsp_ctrl**

Response control of the server model received messages

esp_ble_mesh_gen_onpowerup_state_t ***state**

Parameters of the Generic OnPowerUp state

struct esp_ble_mesh_gen_power_onoff_setup_srv_t

User data of Generic Power OnOff Setup Server Model

Public Members

esp_ble_mesh_model_t ***model**

Pointer to the Generic Power OnOff Setup Server Model. Initialized internally.

esp_ble_mesh_server_rsp_ctrl_t **rsp_ctrl**

Response control of the server model received messages

esp_ble_mesh_gen_onpowerup_state_t ***state**

Parameters of the Generic OnPowerUp state

struct esp_ble_mesh_gen_power_level_state_t

Parameters of Generic Power Level state

Public Members

uint16_t **power_actual**

The present value of the Generic Power Actual state

uint16_t **target_power_actual**

The target value of the Generic Power Actual state

uint16_t **power_last**

The value of the Generic Power Last state

uint16_t **power_default**

The value of the Generic Power Default state

uint8_t **status_code**

The status code of setting Generic Power Range state

uint16_t **power_range_min**

The minimum value of the Generic Power Range state

uint16_t **power_range_max**

The maximum value of the Generic Power Range state

struct esp_ble_mesh_gen_power_level_srv_t

User data of Generic Power Level Server Model

Public Members

esp_ble_mesh_model_t ***model**

Pointer to the Generic Power Level Server Model. Initialized internally.

esp_ble_mesh_server_rsp_ctrl_t **rsp_ctrl**

Response control of the server model received messages

esp_ble_mesh_gen_power_level_state_t ***state**

Parameters of the Generic Power Level state

esp_ble_mesh_last_msg_info_t **last**

Parameters of the last received set message

esp_ble_mesh_state_transition_t **transition**

Parameters of state transition

int32_t **tt_delta_level**

Delta change value of level state transition

struct esp_ble_mesh_gen_power_level_setup_srv_t

User data of Generic Power Level Setup Server Model

Public Members

esp_ble_mesh_model_t ***model**

Pointer to the Generic Power Level Setup Server Model. Initialized internally.

esp_ble_mesh_server_rsp_ctrl_t **rsp_ctrl**

Response control of the server model received messages

esp_ble_mesh_gen_power_level_state_t ***state**

Parameters of the Generic Power Level state

struct esp_ble_mesh_gen_battery_state_t

Parameters of Generic Battery state

Public Members

uint32_t **battery_level**

The value of the Generic Battery Level state

uint32_t **time_to_discharge**

The value of the Generic Battery Time to Discharge state

uint32_t **time_to_charge**

The value of the Generic Battery Time to Charge state

uint32_t **battery_flags**

The value of the Generic Battery Flags state

struct esp_ble_mesh_gen_battery_srv_t

User data of Generic Battery Server Model

Public Members

esp_ble_mesh_model_t ***model**

Pointer to the Generic Battery Server Model. Initialized internally.

esp_ble_mesh_server_rsp_ctrl_t **rsp_ctrl**

Response control of the server model received messages

esp_ble_mesh_gen_battery_state_t **state**

Parameters of the Generic Battery state

struct esp_ble_mesh_gen_location_state_t

Parameters of Generic Location state

Public Members

int32_t **global_latitude**

The value of the Global Latitude field

int32_t **global_longitude**

The value of the Global Longitude field

int16_t **global_altitude**

The value of the Global Altitude field

int16_t **local_north**

The value of the Local North field

int16_t **local_east**

The value of the Local East field

int16_t **local_altitude**

The value of the Local Altitude field

uint8_t **floor_number**

The value of the Floor Number field

uint16_t **uncertainty**

The value of the Uncertainty field

struct esp_ble_mesh_gen_location_srv_t

User data of Generic Location Server Model

Public Members

esp_ble_mesh_model_t ***model**

Pointer to the Generic Location Server Model. Initialized internally.

esp_ble_mesh_server_rsp_ctrl_t **rsp_ctrl**

Response control of the server model received messages

esp_ble_mesh_gen_location_state_t ***state**

Parameters of the Generic Location state

struct esp_ble_mesh_gen_location_setup_srv_t

User data of Generic Location Setup Server Model

Public Members

esp_ble_mesh_model_t ***model**

Pointer to the Generic Location Setup Server Model. Initialized internally.

esp_ble_mesh_server_rsp_ctrl_t **rsp_ctrl**

Response control of the server model received messages

esp_ble_mesh_gen_location_state_t ***state**

Parameters of the Generic Location state

struct esp_ble_mesh_generic_property_t

Parameters of Generic Property states

Public Members

uint16_t **id**

The value of User/Admin/Manufacturer Property ID

uint8_t **user_access**

The value of User Access field

uint8_t **admin_access**

The value of Admin Access field

uint8_t **manu_access**

The value of Manufacturer Access field

struct net_buf_simple ***val**

The value of User/Admin/Manufacturer Property

struct esp_ble_mesh_gen_user_prop_srv_t

User data of Generic User Property Server Model

Public Members

esp_ble_mesh_model_t ***model**

Pointer to the Generic User Property Server Model. Initialized internally.

esp_ble_mesh_server_rsp_ctrl_t **rsp_ctrl**

Response control of the server model received messages

uint8_t **property_count**

Generic User Property count

esp_ble_mesh_generic_property_t ***properties**

Parameters of the Generic User Property state

struct esp_ble_mesh_gen_admin_prop_srv_t

User data of Generic Admin Property Server Model

Public Members

esp_ble_mesh_model_t ***model**

Pointer to the Generic Admin Property Server Model. Initialized internally.

esp_ble_mesh_server_rsp_ctrl_t **rsp_ctrl**

Response control of the server model received messages

uint8_t **property_count**

Generic Admin Property count

esp_ble_mesh_generic_property_t ***properties**

Parameters of the Generic Admin Property state

struct esp_ble_mesh_gen_manu_prop_srv_t

User data of Generic Manufacturer Property Server Model

Public Members

esp_ble_mesh_model_t ***model**

Pointer to the Generic Manufacturer Property Server Model. Initialized internally.

esp_ble_mesh_server_rsp_ctrl_t **rsp_ctrl**

Response control of the server model received messages

uint8_t **property_count**

Generic Manufacturer Property count

esp_ble_mesh_generic_property_t ***properties**

Parameters of the Generic Manufacturer Property state

struct esp_ble_mesh_gen_client_prop_srv_t

User data of Generic Client Property Server Model

Public Members

esp_ble_mesh_model_t ***model**

Pointer to the Generic Client Property Server Model. Initialized internally.

esp_ble_mesh_server_rsp_ctrl_t **rsp_ctrl**

Response control of the server model received messages

uint8_t **id_count**

Generic Client Property ID count

uint16_t ***property_ids**

Parameters of the Generic Client Property state

struct esp_ble_mesh_state_change_gen_onoff_set_t

Parameter of Generic OnOff Set state change event

Public Members

uint8_t **onoff**

The value of Generic OnOff state

struct esp_ble_mesh_state_change_gen_level_set_t

Parameter of Generic Level Set state change event

Public Members

int16_t **level**

The value of Generic Level state

struct esp_ble_mesh_state_change_gen_delta_set_t

Parameter of Generic Delta Set state change event

Public Members

int16_t **level**

The value of Generic Level state

struct esp_ble_mesh_state_change_gen_move_set_t

Parameter of Generic Move Set state change event

Public Members

int16_t **level**

The value of Generic Level state

struct esp_ble_mesh_state_change_gen_def_trans_time_set_t

Parameter of Generic Default Transition Time Set state change event

Public Members

uint8_t **trans_time**

The value of Generic Default Transition Time state

struct esp_ble_mesh_state_change_gen_onpowerup_set_t

Parameter of Generic OnPowerUp Set state change event

Public Members

uint8_t **onpowerup**

The value of Generic OnPowerUp state

struct esp_ble_mesh_state_change_gen_power_level_set_t

Parameter of Generic Power Level Set state change event

Public Members

uint16_t **power**

The value of Generic Power Actual state

struct esp_ble_mesh_state_change_gen_power_default_set_t

Parameter of Generic Power Default Set state change event

Public Members

uint16_t **power**

The value of Generic Power Default state

struct esp_ble_mesh_state_change_gen_power_range_set_t

Parameters of Generic Power Range Set state change event

Public Members

uint16_t **range_min**

The minimum value of Generic Power Range state

uint16_t **range_max**

The maximum value of Generic Power Range state

struct esp_ble_mesh_state_change_gen_loc_global_set_t

Parameters of Generic Location Global Set state change event

Public Members

int32_t **latitude**

The Global Latitude value of Generic Location state

int32_t **longitude**

The Global Longitude value of Generic Location state

int16_t **altitude**

The Global Altitude value of Generic Location state

struct esp_ble_mesh_state_change_gen_loc_local_set_t

Parameters of Generic Location Local Set state change event

Public Members

int16_t **north**

The Local North value of Generic Location state

int16_t **east**

The Local East value of Generic Location state

int16_t **altitude**

The Local Altitude value of Generic Location state

uint8_t **floor_number**

The Floor Number value of Generic Location state

uint16_t **uncertainty**

The Uncertainty value of Generic Location state

struct esp_ble_mesh_state_change_gen_user_property_set_t

Parameters of Generic User Property Set state change event

Public Members

uint16_t **id**

The property id of Generic User Property state

struct net_buf_simple ***value**

The property value of Generic User Property state

struct esp_ble_mesh_state_change_gen_admin_property_set_t

Parameters of Generic Admin Property Set state change event

Public Members

uint16_t **id**

The property id of Generic Admin Property state

uint8_t **access**

The property access of Generic Admin Property state

struct net_buf_simple ***value**

The property value of Generic Admin Property state

struct esp_ble_mesh_state_change_gen_manu_property_set_t

Parameters of Generic Manufacturer Property Set state change event

Public Members

uint16_t **id**

The property id of Generic Manufacturer Property state

uint8_t **access**

The property value of Generic Manufacturer Property state

struct esp_ble_mesh_server_recv_gen_user_property_get_t

Context of the received Generic User Property Get message

Public Members

uint16_t **property_id**

Property ID identifying a Generic User Property

struct esp_ble_mesh_server_recv_gen_admin_property_get_t

Context of the received Generic Admin Property Get message

Public Members

uint16_t **property_id**

Property ID identifying a Generic Admin Property

struct esp_ble_mesh_server_recv_gen_manufacturer_property_get_t

Context of the received Generic Manufacturer Property message

Public Members

uint16_t **property_id**

Property ID identifying a Generic Manufacturer Property

struct esp_ble_mesh_server_recv_gen_client_properties_get_t

Context of the received Generic Client Properties Get message

Public Members

uint16_t **property_id**

A starting Client Property ID present within an element

struct esp_ble_mesh_server_recv_gen_onoff_set_t

Context of the received Generic OnOff Set message

Public Members

bool **op_en**

Indicate if optional parameters are included

uint8_t **onoff**

Target value of Generic OnOff state

uint8_t **tid**

Transaction ID

uint8_t **trans_time**

Time to complete state transition (optional)

uint8_t **delay**

Indicate message execution delay (C.1)

struct esp_ble_mesh_server_recv_gen_level_set_t

Context of the received Generic Level Set message

Public Members

bool **op_en**

Indicate if optional parameters are included

int16_t **level**

Target value of Generic Level state

uint8_t **tid**

Transaction ID

uint8_t **trans_time**

Time to complete state transition (optional)

uint8_t **delay**

Indicate message execution delay (C.1)

struct esp_ble_mesh_server_recv_gen_delta_set_t

Context of the received Generic Delta Set message

Public Members

bool **op_en**

Indicate if optional parameters are included

int32_t **delta_level**

Delta change of Generic Level state

uint8_t **tid**

Transaction ID

uint8_t **trans_time**

Time to complete state transition (optional)

uint8_t **delay**

Indicate message execution delay (C.1)

struct esp_ble_mesh_server_recv_gen_move_set_t

Context of the received Generic Move Set message

Public Members

bool **op_en**

Indicate if optional parameters are included

int16_t **delta_level**

Delta Level step to calculate Move speed for Generic Level state

uint8_t **tid**

Transaction ID

uint8_t **trans_time**

Time to complete state transition (optional)

uint8_t **delay**

Indicate message execution delay (C.1)

struct esp_ble_mesh_server_recv_gen_def_trans_time_set_t

Context of the received Generic Default Transition Time Set message

Public Members

uint8_t **trans_time**

The value of the Generic Default Transition Time state

struct esp_ble_mesh_server_recv_gen_onpowerup_set_t

Context of the received Generic OnPowerUp Set message

Public Members

uint8_t **onpowerup**

The value of the Generic OnPowerUp state

struct esp_ble_mesh_server_recv_gen_power_level_set_t

Context of the received Generic Power Level Set message

Public Members

bool **op_en**

Indicate if optional parameters are included

uint16_t **power**

Target value of Generic Power Actual state

uint8_t **tid**

Transaction ID

uint8_t **trans_time**

Time to complete state transition (optional)

uint8_t **delay**

Indicate message execution delay (C.1)

struct esp_ble_mesh_server_recv_gen_power_default_set_t

Context of the received Generic Power Default Set message

Public Members

uint16_t **power**

The value of the Generic Power Default state

struct esp_ble_mesh_server_recv_gen_power_range_set_t

Context of the received Generic Power Range Set message

Public Members

uint16_t **range_min**

Value of Range Min field of Generic Power Range state

uint16_t **range_max**

Value of Range Max field of Generic Power Range state

struct esp_ble_mesh_server_recv_gen_loc_global_set_t

Context of the received Generic Location Global Set message

Public Members

int32_t **global_latitude**

Global Coordinates (Latitude)

int32_t **global_longitude**

Global Coordinates (Longitude)

int16_t **global_altitude**

Global Altitude

struct esp_ble_mesh_server_recv_gen_loc_local_set_t

Context of the received Generic Location Local Set message

Public Members

int16_t **local_north**

Local Coordinates (North)

int16_t **local_east**

Local Coordinates (East)

int16_t **local_altitude**

Local Altitude

uint8_t **floor_number**

Floor Number

uint16_t **uncertainty**

Uncertainty

struct esp_ble_mesh_server_recv_gen_user_property_set_t

Context of the received Generic User Property Set message

Public Members

uint16_t **property_id**

Property ID identifying a Generic User Property

struct net_buf_simple ***property_value**

Raw value for the User Property

struct esp_ble_mesh_server_recv_gen_admin_property_set_t

Context of the received Generic Admin Property Set message

Public Members

uint16_t **property_id**

Property ID identifying a Generic Admin Property

uint8_t **user_access**

Enumeration indicating user access

struct net_buf_simple ***property_value**

Raw value for the Admin Property

struct esp_ble_mesh_server_recv_gen_manufacturer_property_set_t

Context of the received Generic Manufacturer Property Set message

Public Members

uint16_t **property_id**

Property ID identifying a Generic Manufacturer Property

uint8_t **user_access**

Enumeration indicating user access

struct esp_ble_mesh_generic_server_cb_param_t

Generic Server Model callback parameters

Public Members

esp_ble_mesh_model_t ***model**

Pointer to Generic Server Models

esp_ble_mesh_msg_ctx_t **ctx**
Context of the received messages

esp_ble_mesh_generic_server_cb_value_t **value**
Value of the received Generic Messages

Macros

ESP_BLE_MESH_MODEL_GEN_ONOFF_CLI(cli_pub, cli_data)
Define a new Generic OnOff Client Model.

Note This API needs to be called for each element on which the application needs to have a Generic OnOff Client Model.

Return New Generic OnOff Client Model instance.

Parameters

- `cli_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- `cli_data`: Pointer to the unique struct *esp_ble_mesh_client_t*.

ESP_BLE_MESH_MODEL_GEN_LEVEL_CLI(cli_pub, cli_data)
Define a new Generic Level Client Model.

Note This API needs to be called for each element on which the application needs to have a Generic Level Client Model.

Return New Generic Level Client Model instance.

Parameters

- `cli_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- `cli_data`: Pointer to the unique struct *esp_ble_mesh_client_t*.

ESP_BLE_MESH_MODEL_GEN_DEF_TRANS_TIME_CLI(cli_pub, cli_data)
Define a new Generic Default Transition Time Client Model.

Note This API needs to be called for each element on which the application needs to have a Generic Default Transition Time Client Model.

Return New Generic Default Transition Time Client Model instance.

Parameters

- `cli_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- `cli_data`: Pointer to the unique struct *esp_ble_mesh_client_t*.

ESP_BLE_MESH_MODEL_GEN_POWER_ONOFF_CLI(cli_pub, cli_data)

Define a new Generic Power OnOff Client Model.

Note This API needs to be called for each element on which the application needs to have a Generic Power OnOff Client Model.

Return New Generic Power OnOff Client Model instance.

Parameters

- `cli_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- `cli_data`: Pointer to the unique struct *esp_ble_mesh_client_t*.

ESP_BLE_MESH_MODEL_GEN_POWER_LEVEL_CLI(cli_pub, cli_data)

Define a new Generic Power Level Client Model.

Note This API needs to be called for each element on which the application needs to have a Generic Power Level Client Model.

Return New Generic Power Level Client Model instance.

Parameters

- `cli_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- `cli_data`: Pointer to the unique struct *esp_ble_mesh_client_t*.

ESP_BLE_MESH_MODEL_GEN_BATTERY_CLI(cli_pub, cli_data)

Define a new Generic Battery Client Model.

Note This API needs to be called for each element on which the application needs to have a Generic Battery Client Model.

Return New Generic Battery Client Model instance.

Parameters

- `cli_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- `cli_data`: Pointer to the unique struct *esp_ble_mesh_client_t*.

ESP_BLE_MESH_MODEL_GEN_LOCATION_CLI(cli_pub, cli_data)

Define a new Generic Location Client Model.

Note This API needs to be called for each element on which the application needs to have a Generic Location Client Model.

Return New Generic Location Client Model instance.

Parameters

- `cli_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- `cli_data`: Pointer to the unique struct *esp_ble_mesh_client_t*.

`ESP_BLE_MESH_MODEL_GEN_PROPERTY_CLI`(cli_pub, cli_data)

Define a new Generic Property Client Model.

Note This API needs to be called for each element on which the application needs to have a Generic Property Client Model.

Return New Generic Location Client Model instance.

Parameters

- `cli_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- `cli_data`: Pointer to the unique struct *esp_ble_mesh_client_t*.

`ESP_BLE_MESH_MODEL_GEN_ONOFF_SRV`(srv_pub, srv_data)

Generic Server Models related context.

Define a new Generic OnOff Server Model.

Note 1. The Generic OnOff Server Model is a root model.

1. This model shall support model publication and model subscription.

Return New Generic OnOff Server Model instance.

Parameters

- `srv_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- `srv_data`: Pointer to the unique struct *esp_ble_mesh_gen_onoff_srv_t*.

`ESP_BLE_MESH_MODEL_GEN_LEVEL_SRV`(srv_pub, srv_data)

Define a new Generic Level Server Model.

Note 1. The Generic Level Server Model is a root model.

1. This model shall support model publication and model subscription.

Return New Generic Level Server Model instance.

Parameters

- `srv_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- `srv_data`: Pointer to the unique struct *esp_ble_mesh_gen_level_srv_t*.

`ESP_BLE_MESH_MODEL_GEN_DEF_TRANS_TIME_SRV`(srv_pub, srv_data)

Define a new Generic Default Transition Time Server Model.

Note 1. The Generic Default Transition Time Server Model is a root model.

1. This model shall support model publication and model subscription.

Return New Generic Default Transition Time Server Model instance.

Parameters

- `srv_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- `srv_data`: Pointer to the unique struct *esp_ble_mesh_gen_def_trans_time_srv_t*.

`ESP_BLE_MESH_MODEL_GEN_POWER_ONOFF_SRV`(srv_pub, srv_data)

Define a new Generic Power OnOff Server Model.

Note 1. The Generic Power OnOff Server model extends the Generic OnOff Server model. When this model is present on an element, the corresponding Generic Power OnOff Setup Server model shall also be present.

1. This model may be used to represent a variety of devices that do not fit any of the model descriptions that have been defined but support the generic properties of On/Off.
2. This model shall support model publication and model subscription.

Return New Generic Power OnOff Server Model instance.

Parameters

- `srv_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- `srv_data`: Pointer to the unique struct *esp_ble_mesh_gen_power_onoff_srv_t*.

`ESP_BLE_MESH_MODEL_GEN_POWER_ONOFF_SETUP_SRV`(srv_pub, srv_data)

Define a new Generic Power OnOff Setup Server Model.

Note 1. The Generic Power OnOff Setup Server model extends the Generic Power OnOff Server model and the Generic Default Transition Time Server model.

1. This model shall support model subscription.

Return New Generic Power OnOff Setup Server Model instance.

Parameters

- `srv_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- `srv_data`: Pointer to the unique struct *esp_ble_mesh_gen_power_onoff_setup_srv_t*.

`ESP_BLE_MESH_MODEL_GEN_POWER_LEVEL_SRV`(srv_pub, srv_data)

Define a new Generic Power Level Server Model.

Note 1. The Generic Power Level Server model extends the Generic Power OnOff Server model and the Generic Level Server model. When this model is present on an Element, the corresponding Generic Power Level Setup Server model shall also be present.

1. This model shall support model publication and model subscription.

Return New Generic Power Level Server Model instance.

Parameters

- `srv_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- `srv_data`: Pointer to the unique struct *esp_ble_mesh_gen_power_level_srv_t*.

`ESP_BLE_MESH_MODEL_GEN_POWER_LEVEL_SETUP_SRV`(srv_pub, srv_data)

Define a new Generic Power Level Setup Server Model.

Note 1. The Generic Power Level Setup Server model extends the Generic Power Level Server model and the Generic Power OnOff Setup Server model.

1. This model shall support model subscription.

Return New Generic Power Level Setup Server Model instance.

Parameters

- `srv_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- `srv_data`: Pointer to the unique struct *esp_ble_mesh_gen_power_level_setup_srv_t*.

`ESP_BLE_MESH_MODEL_GEN_BATTERY_SRV`(srv_pub, srv_data)

Define a new Generic Battery Server Model.

Note 1. The Generic Battery Server Model is a root model.

1. This model shall support model publication and model subscription.
2. The model may be used to represent an element that is powered by a battery.

Return New Generic Battery Server Model instance.

Parameters

- `srv_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- `srv_data`: Pointer to the unique struct *esp_ble_mesh_gen_battery_srv_t*.

`ESP_BLE_MESH_MODEL_GEN_LOCATION_SRV`(srv_pub, srv_data)

Define a new Generic Location Server Model.

Note 1. The Generic Location Server model is a root model. When this model is present on an Element, the corresponding Generic Location Setup Server model shall also be present.

1. This model shall support model publication and model subscription.
2. The model may be used to represent an element that knows its location (global or local).

Return New Generic Location Server Model instance.

Parameters

- `srv_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- `srv_data`: Pointer to the unique struct *esp_ble_mesh_gen_location_srv_t*.

`ESP_BLE_MESH_MODEL_GEN_LOCATION_SETUP_SRV`(srv_pub, srv_data)

Define a new Generic Location Setup Server Model.

Note 1. The Generic Location Setup Server model extends the Generic Location Server model.

1. This model shall support model subscription.

Return New Generic Location Setup Server Model instance.

Parameters

- `srv_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- `srv_data`: Pointer to the unique struct *esp_ble_mesh_gen_location_setup_srv_t*.

`ESP_BLE_MESH_MODEL_GEN_USER_PROP_SRV`(srv_pub, srv_data)

Define a new Generic User Property Server Model.

Note 1. The Generic User Property Server model is a root model.

1. This model shall support model publication and model subscription.

Return New Generic User Property Server Model instance.

Parameters

- `srv_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- `srv_data`: Pointer to the unique struct *esp_ble_mesh_gen_user_prop_srv_t*.

`ESP_BLE_MESH_MODEL_GEN_ADMIN_PROP_SRV`(srv_pub, srv_data)

Define a new Generic Admin Property Server Model.

Note 1. The Generic Admin Property Server model extends the Generic User Property Server model.

1. This model shall support model publication and model subscription.

Return New Generic Admin Property Server Model instance.

Parameters

- `srv_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.

- `srv_data`: Pointer to the unique struct *esp_ble_mesh_gen_admin_prop_srv_t*.

`ESP_BLE_MESH_MODEL_GEN_MANUFACTURER_PROP_SRV`(srv_pub, srv_data)

Define a new Generic Manufacturer Property Server Model.

Note 1. The Generic Manufacturer Property Server model extends the Generic User Property Server model.

1. This model shall support model publication and model subscription.

Return New Generic Manufacturer Property Server Model instance.

Parameters

- `srv_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- `srv_data`: Pointer to the unique struct *esp_ble_mesh_gen_manu_prop_srv_t*.

`ESP_BLE_MESH_MODEL_GEN_CLIENT_PROP_SRV`(srv_pub, srv_data)

Define a new Generic User Property Server Model.

Note 1. The Generic Client Property Server model is a root model.

1. This model shall support model publication and model subscription.

Return New Generic Client Property Server Model instance.

Parameters

- `srv_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- `srv_data`: Pointer to the unique struct *esp_ble_mesh_gen_client_prop_srv_t*.

Type Definitions

`typedef` void (*`esp_ble_mesh_generic_client_cb_t`)(*esp_ble_mesh_generic_client_cb_event_t* event, *esp_ble_mesh_generic_client_cb_param_t* *param)

Bluetooth Mesh Generic Client Model function.

Generic Client Model callback function type

Parameters

- `event`: Event type
- `param`: Pointer to callback parameter

`typedef` void (*`esp_ble_mesh_generic_server_cb_t`)(*esp_ble_mesh_generic_server_cb_event_t* event, *esp_ble_mesh_generic_server_cb_param_t* *param)

Bluetooth Mesh Generic Server Model function.

Generic Server Model callback function type

Parameters

- `event`: Event type
- `param`: Pointer to callback parameter

Enumerations

enum esp_ble_mesh_generic_client_cb_event_t

This enum value is the event of Generic Client Model

Values:

ESP_BLE_MESH_GENERIC_CLIENT_GET_STATE_EVT

ESP_BLE_MESH_GENERIC_CLIENT_SET_STATE_EVT

ESP_BLE_MESH_GENERIC_CLIENT_PUBLISH_EVT

ESP_BLE_MESH_GENERIC_CLIENT_TIMEOUT_EVT

ESP_BLE_MESH_GENERIC_CLIENT_EVT_MAX

enum esp_ble_mesh_gen_user_prop_access_t

This enum value is the access value of Generic User Property

Values:

ESP_BLE_MESH_GEN_USER_ACCESS_PROHIBIT

ESP_BLE_MESH_GEN_USER_ACCESS_READ

ESP_BLE_MESH_GEN_USER_ACCESS_WRITE

ESP_BLE_MESH_GEN_USER_ACCESS_READ_WRITE

enum esp_ble_mesh_gen_admin_prop_access_t

This enum value is the access value of Generic Admin Property

Values:

ESP_BLE_MESH_GEN_ADMIN_NOT_USER_PROP

ESP_BLE_MESH_GEN_ADMIN_ACCESS_READ

ESP_BLE_MESH_GEN_ADMIN_ACCESS_WRITE

ESP_BLE_MESH_GEN_ADMIN_ACCESS_READ_WRITE

enum esp_ble_mesh_gen_manu_prop_access_t

This enum value is the access value of Generic Manufacturer Property

Values:

ESP_BLE_MESH_GEN_MANU_NOT_USER_PROP

ESP_BLE_MESH_GEN_MANU_ACCESS_READ

enum esp_ble_mesh_generic_server_cb_event_t

This enum value is the event of Generic Server Model

Values:

ESP_BLE_MESH_GENERIC_SERVER_STATE_CHANGE_EVT

1. When get_auto_rsp is set to ESP_BLE_MESH_SERVER_AUTO_RSP, no event will be callback to the application layer when Generic Get messages are received.
2. When set_auto_rsp is set to ESP_BLE_MESH_SERVER_AUTO_RSP, this event will be callback to the application layer when Generic Set/Set Unack messages are received.

ESP_BLE_MESH_GENERIC_SERVER_RECV_GET_MSG_EVT

When get_auto_rsp is set to ESP_BLE_MESH_SERVER_RSP_BY_APP, this event will be callback to the application layer when Generic Get messages are received.

ESP_BLE_MESH_GENERIC_SERVER_RECV_SET_MSG_EVT

When set_auto_rsp is set to ESP_BLE_MESH_SERVER_RSP_BY_APP, this event will be callback to the application layer when Generic Set/Set Unack messages are received.

ESP_BLE_MESH_GENERIC_SERVER_EVT_MAX

Generic Client/Server Models

Header File

- bt/esp_ble_mesh/api/models/include/esp_ble_mesh_health_model_api.h

Functions

esp_err_t **esp_ble_mesh_register_health_client_callback**(*esp_ble_mesh_health_client_cb_t callback*)

Register BLE Mesh Health Model callback, the callback will report Health Client & Server Model events.

Return ESP_OK on success or error code otherwise.

Parameters

- `callback`: Pointer to the callback function.

esp_err_t **esp_ble_mesh_register_health_server_callback**(*esp_ble_mesh_health_server_cb_t callback*)

Register BLE Mesh Health Server Model callback.

Return ESP_OK on success or error code otherwise.

Parameters

- `callback`: Pointer to the callback function.

esp_err_t **esp_ble_mesh_health_client_get_state**(*esp_ble_mesh_client_common_param_t* **params*, *esp_ble_mesh_health_client_get_state_t* **get_state*)

This function is called to get the Health Server states using the Health Client Model get messages.

Note If you want to find the opcodes and corresponding meanings accepted by this API, please refer to esp_ble_mesh_opcode_health_client_get_t in esp_ble_mesh_defs.h

Return ESP_OK on success or error code otherwise.

Parameters

- `params`: Pointer to BLE Mesh common client parameters.
- `get_state`: Pointer to a union, each kind of opcode corresponds to one structure inside. Shall not be set to NULL.

esp_err_t **esp_ble_mesh_health_client_set_state**(*esp_ble_mesh_client_common_param_t* **params*, *esp_ble_mesh_health_client_set_state_t* **set_state*)

This function is called to set the Health Server states using the Health Client Model set messages.

Note If you want to find the opcodes and corresponding meanings accepted by this API, please refer to esp_ble_mesh_opcode_health_client_set_t in esp_ble_mesh_defs.h

Return ESP_OK on success or error code otherwise.

Parameters

- `params`: Pointer to BLE Mesh common client parameters.
- `set_state`: Pointer to a union, each kind of opcode corresponds to one structure inside. Shall not be set to NULL.

esp_err_t **esp_ble_mesh_health_server_fault_update**(*esp_ble_mesh_elem_t* **element*)

This function is called by the Health Server Model to update the context of its Health Current status.

Return ESP_OK on success or error code otherwise.

Parameters

- `element`: The element to which the Health Server Model belongs.

Unions

union esp_ble_mesh_health_client_get_state_t

#include <esp_ble_mesh_health_model_api.h> For ESP_BLE_MESH_MODEL_OP_HEALTH_FAULT_GET ESP_BLE_MESH_MODEL_OP_ATTENTION_GET ESP_BLE_MESH_MODEL_OP_HEALTH_PERIOD_GE the get_state parameter in the esp_ble_mesh_health_client_get_state function should not be set to NULL.

Public Members

esp_ble_mesh_health_fault_get_t **fault_get**

For ESP_BLE_MESH_MODEL_OP_HEALTH_FAULT_GET.

union esp_ble_mesh_health_client_set_state_t

#include <esp_ble_mesh_health_model_api.h> For ESP_BLE_MESH_MODEL_OP_HEALTH_FAULT_CLEAR ESP_BLE_MESH_MODEL_OP_HEALTH_FAULT_CLEAR_UNACK ESP_BLE_MESH_MODEL_OP_HEALTH_FAULT_TEST ESP_BLE_MESH_MODEL_OP_HEALTH_FAULT ESP_BLE_MESH_MODEL_OP_HEALTH_PERIOD_SET ESP_BLE_MESH_MODEL_OP_HEALTH_PERIO ESP_BLE_MESH_MODEL_OP_ATTENTION_SET ESP_BLE_MESH_MODEL_OP_ATTENTION_SET_UN the set_state parameter in the esp_ble_mesh_health_client_set_state function should not be set to NULL.

Public Members

esp_ble_mesh_health_attention_set_t **attention_set**

For ESP_BLE_MESH_MODEL_OP_ATTENTION_SET or ESP_BLE_MESH_MODEL_OP_ATTENTION

esp_ble_mesh_health_period_set_t **period_set**

For ESP_BLE_MESH_MODEL_OP_HEALTH_PERIOD_SET or ESP_BLE_MESH_MODEL_OP_HEALTH_PERIOD_SET_UNACK.

esp_ble_mesh_health_fault_test_t **fault_test**

For ESP_BLE_MESH_MODEL_OP_HEALTH_FAULT_TEST or ESP_BLE_MESH_MODEL_OP_HEALTH_FAULT_TEST_UNACK.

esp_ble_mesh_health_fault_clear_t **fault_clear**

For ESP_BLE_MESH_MODEL_OP_HEALTH_FAULT_CLEAR or ESP_BLE_MESH_MODEL_OP_HEALTH_FAULT_CLEAR_UNACK.

union esp_ble_mesh_health_client_common_cb_param_t

#include <esp_ble_mesh_health_model_api.h> Health Client Model received message union.

Public Members

esp_ble_mesh_health_current_status_cb_t **current_status**

The health current status value

esp_ble_mesh_health_fault_status_cb_t **fault_status**

The health fault status value

esp_ble_mesh_health_period_status_cb_t **period_status**

The health period status value

esp_ble_mesh_health_attention_status_cb_t **attention_status**

The health attention status value

union esp_ble_mesh_health_server_cb_param_t

#include <esp_ble_mesh_health_model_api.h> Health Server Model callback parameters union.

Public Members

esp_ble_mesh_health_fault_update_comp_cb_t **fault_update_comp**

ESP_BLE_MESH_HEALTH_SERVER_FAULT_UPDATE_COMP_EVT

esp_ble_mesh_health_fault_clear_cb_t **fault_clear**

ESP_BLE_MESH_HEALTH_SERVER_FAULT_CLEAR_EVT

esp_ble_mesh_health_fault_test_cb_t **fault_test**

ESP_BLE_MESH_HEALTH_SERVER_FAULT_TEST_EVT

esp_ble_mesh_health_attention_on_cb_t **attention_on**

ESP_BLE_MESH_HEALTH_SERVER_ATTENTION_ON_EVT

esp_ble_mesh_health_attention_off_cb_t **attention_off**

ESP_BLE_MESH_HEALTH_SERVER_ATTENTION_OFF_EVT

Structures

struct esp_ble_mesh_health_srv_cb_t

ESP BLE Mesh Health Server callback

Public Members

esp_ble_mesh_cb_t **fault_clear**

Clear health registered faults. Initialized by the stack.

esp_ble_mesh_cb_t **fault_test**

Run a specific health test. Initialized by the stack.

esp_ble_mesh_cb_t **attention_on**

Health attention on callback. Initialized by the stack.

esp_ble_mesh_cb_t **attention_off**

Health attention off callback. Initialized by the stack.

struct esp_ble_mesh_health_test_t

ESP BLE Mesh Health Server test Context

Public Members

uint8_t **id_count**

Number of Health self-test ID

const uint8_t ***test_ids**

Array of Health self-test IDs

uint16_t **company_id**

Company ID used to identify the Health Fault state

uint8_t **prev_test_id**

Current test ID of the health fault test

uint8_t **current_faults**[**ESP_BLE_MESH_HEALTH_FAULT_ARRAY_SIZE**]

Array of current faults

uint8_t **registered_faults**[**ESP_BLE_MESH_HEALTH_FAULT_ARRAY_SIZE**]

Array of registered faults

struct esp_ble_mesh_health_srv_t

ESP BLE Mesh Health Server Model Context

Public Members

esp_ble_mesh_model_t ***model**

Pointer to Health Server Model

esp_ble_mesh_health_srv_cb_t **health_cb**

Health callback struct

struct k_delayed_work **attention_timer**

Attention Timer state

bool **attention_timer_start**

Attention Timer start flag

esp_ble_mesh_health_test_t **health_test**

Health Server fault test

struct esp_ble_mesh_health_fault_get_t

Parameter of Health Fault Get

Public Members

uint16_t **company_id**

Bluetooth assigned 16-bit Company ID

struct esp_ble_mesh_health_attention_set_t

Parameter of Health Attention Set

Public Members

uint8_t **attention**

Value of the Attention Timer state

struct esp_ble_mesh_health_period_set_t

Parameter of Health Period Set

Public Members

uint8_t **fast_period_divisor**

Divider for the Publish Period

struct esp_ble_mesh_health_fault_test_t

Parameter of Health Fault Test

Public Members

uint16_t **company_id**

Bluetooth assigned 16-bit Company ID

uint8_t **test_id**

ID of a specific test to be performed

struct esp_ble_mesh_health_fault_clear_t

Parameter of Health Fault Clear

Public Members

uint16_t **company_id**

Bluetooth assigned 16-bit Company ID

struct esp_ble_mesh_health_current_status_cb_t

Parameters of Health Current Status

Public Members

uint8_t **test_id**

ID of a most recently performed test

uint16_t **company_id**

Bluetooth assigned 16-bit Company ID

struct net_buf_simple ***fault_array**

FaultArray field contains a sequence of 1-octet fault values

struct esp_ble_mesh_health_fault_status_cb_t

Parameters of Health Fault Status

Public Members

uint8_t **test_id**

ID of a most recently performed test

uint16_t **company_id**

Bluetooth assigned 16-bit Company ID

struct net_buf_simple ***fault_array**

FaultArray field contains a sequence of 1-octet fault values

struct esp_ble_mesh_health_period_status_cb_t

Parameter of Health Period Status

Public Members

uint8_t **fast_period_divisor**

Divider for the Publish Period

struct esp_ble_mesh_health_attention_status_cb_t

Parameter of Health Attention Status

Public Members

uint8_t **attention**

Value of the Attention Timer state

struct esp_ble_mesh_health_client_cb_param_t

Health Client Model callback parameters

Public Members

int **error_code**

Appropriate error code

esp_ble_mesh_client_common_param_t ***params**

The client common parameters.

esp_ble_mesh_health_client_common_cb_param_t **status_cb**

The health message status callback values

struct esp_ble_mesh_health_fault_update_comp_cb_t

Parameter of publishing Health Current Status completion event

Public Members

int **error_code**

The result of publishing Health Current Status

esp_ble_mesh_elem_t ***element**

Pointer to the element which contains the Health Server Model

struct esp_ble_mesh_health_fault_clear_cb_t

Parameters of Health Fault Clear event

Public Members

esp_ble_mesh_model_t ***model**

Pointer to the Health Server Model

uint16_t **company_id**

Bluetooth assigned 16-bit Company ID

struct esp_ble_mesh_health_fault_test_cb_t

Parameters of Health Fault Test event

Public Members

esp_ble_mesh_model_t ***model**

Pointer to the Health Server Model

uint8_t **test_id**

ID of a specific test to be performed

uint16_t **company_id**

Bluetooth assigned 16-bit Company ID

struct esp_ble_mesh_health_attention_on_cb_t

Parameter of Health Attention On event

Public Members

esp_ble_mesh_model_t *model

Pointer to the Health Server Model

uint8_t time

Duration of attention timer on (in seconds)

struct esp_ble_mesh_health_attention_off_cb_t

Parameter of Health Attention Off event

Public Members

esp_ble_mesh_model_t *model

Pointer to the Health Server Model

Macros

ESP_BLE_MESH_MODEL_HEALTH_SRV(srv, pub)

Define a new Health Server Model.

Note The Health Server Model can only be included by a Primary Element.

Return New Health Server Model instance.

Parameters

- srv: Pointer to the unique struct *esp_ble_mesh_health_srv_t*.
- pub: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.

ESP_BLE_MESH_MODEL_HEALTH_CLI(cli_data)

Define a new Health Client Model.

Note This API needs to be called for each element on which the application needs to have a Health Client Model.

Return New Health Client Model instance.

Parameters

- cli_data: Pointer to the unique struct *esp_ble_mesh_client_t*.

ESP_BLE_MESH_HEALTH_PUB_DEFINE(_name, _max, _role)

A helper to define a health publication context

Parameters

- `_name`: Name given to the publication context variable.
- `_max`: Maximum number of faults the element can have.
- `_role`: Role of the device which contains the model.

`ESP_BLE_MESH_HEALTH_STANDARD_TEST`

SIG identifier of Health Fault Test. 0x01 ~ 0xFF: Vendor Specific Test.

`ESP_BLE_MESH_NO_FAULT`

Fault values of Health Fault Test. 0x33 ~ 0x7F: Reserved for Future Use. 0x80 ~ 0xFF: Vendor Specific Warning/Error.

`ESP_BLE_MESH_BATTERY_LOW_WARNING`

`ESP_BLE_MESH_BATTERY_LOW_ERROR`

`ESP_BLE_MESH_SUPPLY_VOLTAGE_TOO_LOW_WARNING`

`ESP_BLE_MESH_SUPPLY_VOLTAGE_TOO_LOW_ERROR`

`ESP_BLE_MESH_SUPPLY_VOLTAGE_TOO_HIGH_WARNING`

`ESP_BLE_MESH_SUPPLY_VOLTAGE_TOO_HIGH_ERROR`

`ESP_BLE_MESH_POWER_SUPPLY_INTERRUPTED_WARNING`

`ESP_BLE_MESH_POWER_SUPPLY_INTERRUPTED_ERROR`

`ESP_BLE_MESH_NO_LOAD_WARNING`

`ESP_BLE_MESH_NO_LOAD_ERROR`

`ESP_BLE_MESH_OVERLOAD_WARNING`

`ESP_BLE_MESH_OVERLOAD_ERROR`

`ESP_BLE_MESH_OVERHEAT_WARNING`

`ESP_BLE_MESH_OVERHEAT_ERROR`

`ESP_BLE_MESH_CONDENSATION_WARNING`

`ESP_BLE_MESH_CONDENSATION_ERROR`

`ESP_BLE_MESH_VIBRATION_WARNING`

`ESP_BLE_MESH_VIBRATION_ERROR`

`ESP_BLE_MESH_CONFIGURATION_WARNING`

`ESP_BLE_MESH_CONFIGURATION_ERROR`

`ESP_BLE_MESH_ELEMENT_NOT_CALIBRATED_WARNING`

`ESP_BLE_MESH_ELEMENT_NOT_CALIBRATED_ERROR`

ESP_BLE_MESH_MEMORY_WARNING

ESP_BLE_MESH_MEMORY_ERROR

ESP_BLE_MESH_SELF_TEST_WARNING

ESP_BLE_MESH_SELF_TEST_ERROR

ESP_BLE_MESH_INPUT_TOO_LOW_WARNING

ESP_BLE_MESH_INPUT_TOO_LOW_ERROR

ESP_BLE_MESH_INPUT_TOO_HIGH_WARNING

ESP_BLE_MESH_INPUT_TOO_HIGH_ERROR

ESP_BLE_MESH_INPUT_NO_CHANGE_WARNING

ESP_BLE_MESH_INPUT_NO_CHANGE_ERROR

ESP_BLE_MESH_ACTUATOR_BLOCKED_WARNING

ESP_BLE_MESH_ACTUATOR_BLOCKED_ERROR

ESP_BLE_MESH_HOUSING_OPENED_WARNING

ESP_BLE_MESH_HOUSING_OPENED_ERROR

ESP_BLE_MESH_TAMPER_WARNING

ESP_BLE_MESH_TAMPER_ERROR

ESP_BLE_MESH_DEVICE_MOVED_WARNING

ESP_BLE_MESH_DEVICE_MOVED_ERROR

ESP_BLE_MESH_DEVICE_DROPPED_WARNING

ESP_BLE_MESH_DEVICE_DROPPED_ERROR

ESP_BLE_MESH_OVERFLOW_WARNING

ESP_BLE_MESH_OVERFLOW_ERROR

ESP_BLE_MESH_EMPTY_WARNING

ESP_BLE_MESH_EMPTY_ERROR

ESP_BLE_MESH_INTERNAL_BUS_WARNING

ESP_BLE_MESH_INTERNAL_BUS_ERROR

ESP_BLE_MESH_MECHANISM_JAMMED_WARNING

ESP_BLE_MESH_MECHANISM_JAMMED_ERROR

ESP_BLE_MESH_HEALTH_FAULT_ARRAY_SIZE

Type Definitions

typedef void (*esp_ble_mesh_health_client_cb_t)(*esp_ble_mesh_health_client_cb_event_t* event, *esp_ble_mesh_health_client_cb_param_t* *param)

Bluetooth Mesh Health Client and Server Model function.

Health Client Model callback function type

Parameters

- `event`: Event type
- `param`: Pointer to callback parameter

typedef void (*esp_ble_mesh_health_server_cb_t)(*esp_ble_mesh_health_server_cb_event_t* event, *esp_ble_mesh_health_server_cb_param_t* *param)

Health Server Model callback function type.

Parameters

- `event`: Event type
- `param`: Pointer to callback parameter

Enumerations

enum esp_ble_mesh_health_client_cb_event_t

This enum value is the event of Health Client Model

Values:

ESP_BLE_MESH_HEALTH_CLIENT_GET_STATE_EVT

ESP_BLE_MESH_HEALTH_CLIENT_SET_STATE_EVT

ESP_BLE_MESH_HEALTH_CLIENT_PUBLISH_EVT

ESP_BLE_MESH_HEALTH_CLIENT_TIMEOUT_EVT

ESP_BLE_MESH_HEALTH_CLIENT_EVT_MAX

enum esp_ble_mesh_health_server_cb_event_t

This enum value is the event of Health Server Model

Values:

ESP_BLE_MESH_HEALTH_SERVER_FAULT_UPDATE_COMP_EVT

ESP_BLE_MESH_HEALTH_SERVER_FAULT_CLEAR_EVT

ESP_BLE_MESH_HEALTH_SERVER_FAULT_TEST_EVT

ESP_BLE_MESH_HEALTH_SERVER_ATTENTION_ON_EVT

ESP_BLE_MESH_HEALTH_SERVER_ATTENTION_OFF_EVT

ESP_BLE_MESH_HEALTH_SERVER_EVT_MAX

Sensor Client/Server Models

Header File

- bt/esp_ble_mesh/api/models/include/esp_ble_mesh_lighting_model_api.h

Functions

esp_err_t **esp_ble_mesh_register_light_client_callback**(*esp_ble_mesh_light_client_cb_t callback*)

Register BLE Mesh Light Client Model callback.

Return ESP_OK on success or error code otherwise.

Parameters

- `callback`: pointer to the callback function.

esp_err_t **esp_ble_mesh_light_client_get_state**(*esp_ble_mesh_client_common_param_t* **params*, *esp_ble_mesh_light_client_get_state_t* **get_state*)

Get the value of Light Server Model states using the Light Client Model get messages.

Note If you want to know the opcodes and corresponding meanings accepted by this API, please refer to esp_ble_mesh_light_message_opcode_t in esp_ble_mesh_defs.h

Return ESP_OK on success or error code otherwise.

Parameters

- `params`: Pointer to BLE Mesh common client parameters.
- `get_state`: Pointer of light get message value. Shall not be set to NULL.

esp_err_t **esp_ble_mesh_light_client_set_state**(*esp_ble_mesh_client_common_param_t* **params*, *esp_ble_mesh_light_client_set_state_t* **set_state*)

Set the value of Light Server Model states using the Light Client Model set messages.

Note If you want to know the opcodes and corresponding meanings accepted by this API, please refer to esp_ble_mesh_light_message_opcode_t in esp_ble_mesh_defs.h

Return ESP_OK on success or error code otherwise.

Parameters

- `params`: Pointer to BLE Mesh common client parameters.
- `set_state`: Pointer of light set message value. Shall not be set to NULL.

esp_err_t **esp_ble_mesh_register_lighting_server_callback**(*esp_ble_mesh_lighting_server_cb_t callback*)

Register BLE Mesh Lighting Server Model callback.

Return ESP_OK on success or error code otherwise.

Parameters

- `callback`: Pointer to the callback function.

Unions

union esp_ble_mesh_light_client_get_state_t

#include <esp_ble_mesh_lighting_model_api.h> Lighting Client Model get message union.

Public Members

esp_ble_mesh_light_lc_property_get_t **lc_property_get**

For ESP_BLE_MESH_MODEL_OP_LIGHT_LC_PROPERTY_GET

union esp_ble_mesh_light_client_set_state_t

#include <esp_ble_mesh_lighting_model_api.h> Lighting Client Model set message union.

Public Members

esp_ble_mesh_light_lightness_set_t **lightness_set**

For ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_SET & ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_SET_UNACK

esp_ble_mesh_light_lightness_linear_set_t **lightness_linear_set**

For ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_LINEAR_SET & ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_LINEAR_SET_UNACK

esp_ble_mesh_light_lightness_default_set_t **lightness_default_set**

For ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_DEFAULT_SET & ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_DEFAULT_SET_UNACK

esp_ble_mesh_light_lightness_range_set_t **lightness_range_set**

For ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_RANGE_SET & ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_RANGE_SET_UNACK

esp_ble_mesh_light_ctl_set_t **ctl_set**

For ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_SET & ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_

esp_ble_mesh_light_ctl_temperature_set_t **ctl_temperature_set**

For ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_TEMPERATURE_SET & ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_TEMPERATURE_SET_UNACK

esp_ble_mesh_light_ctl_temperature_range_set_t **ctl_temperature_range_set**

For ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_TEMPERATURE_RANGE_SET & ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_TEMPERATURE_RANGE_SET_UNACK

esp_ble_mesh_light_ctl_default_set_t **ctl_default_set**

For ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_DEFAULT_SET & ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_DEFAULT_SET_UNACK

esp_ble_mesh_light_hsl_set_t **hsl_set**

For ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_SET & ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_

esp_ble_mesh_light_hsl_hue_set_t **hsl_hue_set**

For ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_HUE_SET & ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_HUE_SET_UNACK

esp_ble_mesh_light_hsl_saturation_set_t **hsl_saturation_set**

For ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_SATURATION_SET & ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_SATURATION_SET_UNACK

esp_ble_mesh_light_hsl_default_set_t **hsl_default_set**

For ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_DEFAULT_SET & ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_DEFAULT_SET_UNACK

esp_ble_mesh_light_hsl_range_set_t **hsl_range_set**

For ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_RANGE_SET & ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_RANGE_SET_UNACK

esp_ble_mesh_light_xyl_set_t **xyl_set**

For ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_SET & ESP_BLE_MESH_MODEL_OP_LIGHT_XYL

esp_ble_mesh_light_xyl_default_set_t **xyl_default_set**

For ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_DEFAULT_SET & ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_DEFAULT_SET_UNACK

esp_ble_mesh_light_xyl_range_set_t **xyl_range_set**

For ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_RANGE_SET & ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_RANGE_SET_UNACK

esp_ble_mesh_light_lc_mode_set_t **lc_mode_set**

For ESP_BLE_MESH_MODEL_OP_LIGHT_LC_MODE_SET & ESP_BLE_MESH_MODEL_OP_LIGHT_LC_MODE_SET_UNACK

esp_ble_mesh_light_lc_om_set_t **lc_om_set**

For ESP_BLE_MESH_MODEL_OP_LIGHT_LC_OM_SET & ESP_BLE_MESH_MODEL_OP_LIGHT_LC_OM_SET_UNACK

esp_ble_mesh_light_lc_light_onoff_set_t **lc_light_onoff_set**

For ESP_BLE_MESH_MODEL_OP_LIGHT_LC_LIGHT_ONOFF_SET & ESP_BLE_MESH_MODEL_OP_LIGHT_LC_LIGHT_ONOFF_SET_UNACK

esp_ble_mesh_light_lc_property_set_t **lc_property_set**

For ESP_BLE_MESH_MODEL_OP_LIGHT_LC_PROPERTY_SET & ESP_BLE_MESH_MODEL_OP_LIGHT_LC_PROPERTY_SET_UNACK

union esp_ble_mesh_light_client_status_cb_t

#include <esp_ble_mesh_lighting_model_api.h> Lighting Client Model received message union.

Public Members

esp_ble_mesh_light_lightness_status_cb_t **lightness_status**

For ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_STATUS

esp_ble_mesh_light_lightness_linear_status_cb_t **lightness_linear_status**

For ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_LINEAR_STATUS

esp_ble_mesh_light_lightness_last_status_cb_t **lightness_last_status**

For ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_LAST_STATUS

esp_ble_mesh_light_lightness_default_status_cb_t **lightness_default_status**

For ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_DEFAULT_STATUS

esp_ble_mesh_light_lightness_range_status_cb_t **lightness_range_status**

For ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_RANGE_STATUS

esp_ble_mesh_light_ctl_status_cb_t **ctl_status**

For ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_STATUS

esp_ble_mesh_light_ctl_temperature_status_cb_t **ctl_temperature_status**

For ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_TEMPERATURE_STATUS

esp_ble_mesh_light_ctl_temperature_range_status_cb_t **ctl_temperature_range_status**

For ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_TEMPERATURE_RANGE_STATUS

esp_ble_mesh_light_ctl_default_status_cb_t **ctl_default_status**

For ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_DEFAULT_STATUS

esp_ble_mesh_light_hsl_status_cb_t **hsl_status**

For ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_STATUS

esp_ble_mesh_light_hsl_target_status_cb_t **hsl_target_status**

For ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_TARGET_STATUS

esp_ble_mesh_light_hsl_hue_status_cb_t **hsl_hue_status**
For ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_HUE_STATUS

esp_ble_mesh_light_hsl_saturation_status_cb_t **hsl_saturation_status**
For ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_SATURATION_STATUS

esp_ble_mesh_light_hsl_default_status_cb_t **hsl_default_status**
For ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_DEFAULT_STATUS

esp_ble_mesh_light_hsl_range_status_cb_t **hsl_range_status**
For ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_RANGE_STATUS

esp_ble_mesh_light_xyl_status_cb_t **xyl_status**
For ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_STATUS

esp_ble_mesh_light_xyl_target_status_cb_t **xyl_target_status**
For ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_TARGET_STATUS

esp_ble_mesh_light_xyl_default_status_cb_t **xyl_default_status**
For ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_DEFAULT_STATUS

esp_ble_mesh_light_xyl_range_status_cb_t **xyl_range_status**
For ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_RANGE_STATUS

esp_ble_mesh_light_lc_mode_status_cb_t **lc_mode_status**
For ESP_BLE_MESH_MODEL_OP_LIGHT_LC_MODE_STATUS

esp_ble_mesh_light_lc_om_status_cb_t **lc_om_status**
For ESP_BLE_MESH_MODEL_OP_LIGHT_LC_OM_STATUS

esp_ble_mesh_light_lc_light_onoff_status_cb_t **lc_light_onoff_status**
For ESP_BLE_MESH_MODEL_OP_LIGHT_LC_LIGHT_ONOFF_STATUS

esp_ble_mesh_light_lc_property_status_cb_t **lc_property_status**
For ESP_BLE_MESH_MODEL_OP_LIGHT_LC_PROPERTY_STATUS

union esp_ble_mesh_lighting_server_state_change_t
#include <esp_ble_mesh_lighting_model_api.h> Lighting Server Model state change value union.

Public Members

esp_ble_mesh_state_change_light_lightness_set_t **lightness_set**
The recv_op in ctx can be used to decide which state is changed.Light Lightness Set

esp_ble_mesh_state_change_light_lightness_linear_set_t **lightness_linear_set**
Light Lightness Linear Set

esp_ble_mesh_state_change_light_lightness_default_set_t **lightness_default_set**
Light Lightness Default Set

esp_ble_mesh_state_change_light_lightness_range_set_t **lightness_range_set**
Light Lightness Range Set

esp_ble_mesh_state_change_light_ctl_set_t **ctl_set**
Light CTL Set

esp_ble_mesh_state_change_light_ctl_temperature_set_t **ctl_temp_set**
Light CTL Temperature Set

esp_ble_mesh_state_change_light_ctl_temperature_range_set_t **ctl_temp_range_set**
Light CTL Temperature Range Set

esp_ble_mesh_state_change_light_ctl_default_set_t **ctl_default_set**
Light CTL Default Set

esp_ble_mesh_state_change_light_hsl_set_t **hsl_set**
Light HSL Set

esp_ble_mesh_state_change_light_hsl_hue_set_t **hsl_hue_set**
Light HSL Hue Set

esp_ble_mesh_state_change_light_hsl_saturation_set_t **hsl_saturation_set**
Light HSL Saturation Set

esp_ble_mesh_state_change_light_hsl_default_set_t **hsl_default_set**
Light HSL Default Set

esp_ble_mesh_state_change_light_hsl_range_set_t **hsl_range_set**
Light HSL Range Set

esp_ble_mesh_state_change_light_xyl_set_t **xyl_set**
Light xyL Set

esp_ble_mesh_state_change_light_xyl_default_set_t **xyl_default_set**
Light xyL Default Set

esp_ble_mesh_state_change_light_xyl_range_set_t **xyl_range_set**
Light xyL Range Set

esp_ble_mesh_state_change_light_lc_mode_set_t **lc_mode_set**
Light LC Mode Set

esp_ble_mesh_state_change_light_lc_om_set_t **lc_om_set**
Light LC Occupancy Mode Set

esp_ble_mesh_state_change_light_lc_light_onoff_set_t **lc_light_onoff_set**
Light LC Light OnOff Set

esp_ble_mesh_state_change_light_lc_property_set_t **lc_property_set**
Light LC Property Set

esp_ble_mesh_state_change_sensor_status_t **sensor_status**

Sensor Status

union esp_ble_mesh_lighting_server_recv_get_msg_t

#include <esp_ble_mesh_lighting_model_api.h> Lighting Server Model received get message union.

Public Members

esp_ble_mesh_server_recv_light_lc_property_get_t **lc_property**

Light LC Property Get

union esp_ble_mesh_lighting_server_recv_set_msg_t

#include <esp_ble_mesh_lighting_model_api.h> Lighting Server Model received set message union.

Public Members

esp_ble_mesh_server_recv_light_lightness_set_t **lightness**

Light Lightness Set/Light Lightness Set Unack

esp_ble_mesh_server_recv_light_lightness_linear_set_t **lightness_linear**

Light Lightness Linear Set/Light Lightness Linear Set Unack

esp_ble_mesh_server_recv_light_lightness_default_set_t **lightness_default**

Light Lightness Default Set/Light Lightness Default Set Unack

esp_ble_mesh_server_recv_light_lightness_range_set_t **lightness_range**

Light Lightness Range Set/Light Lightness Range Set Unack

esp_ble_mesh_server_recv_light_ctl_set_t **ctl**

Light CTL Set/Light CTL Set Unack

esp_ble_mesh_server_recv_light_ctl_temperature_set_t **ctl_temp**

Light CTL Temperature Set/Light CTL Temperature Set Unack

esp_ble_mesh_server_recv_light_ctl_temperature_range_set_t **ctl_temp_range**

Light CTL Temperature Range Set/Light CTL Temperature Range Set Unack

esp_ble_mesh_server_recv_light_ctl_default_set_t **ctl_default**

Light CTL Default Set/Light CTL Default Set Unack

esp_ble_mesh_server_recv_light_hsl_set_t **hsl**

Light HSL Set/Light HSL Set Unack

esp_ble_mesh_server_recv_light_hsl_hue_set_t **hsl_hue**

Light HSL Hue Set/Light HSL Hue Set Unack

esp_ble_mesh_server_recv_light_hsl_saturation_set_t **hsl_saturation**

Light HSL Saturation Set/Light HSL Saturation Set Unack

esp_ble_mesh_server_recv_light_hsl_default_set_t **hsl_default**
Light HSL Default Set/Light HSL Default Set Unack

esp_ble_mesh_server_recv_light_hsl_range_set_t **hsl_range**
Light HSL Range Set/Light HSL Range Set Unack

esp_ble_mesh_server_recv_light_xyl_set_t **xyl**
Light xyL Set/Light xyL Set Unack

esp_ble_mesh_server_recv_light_xyl_default_set_t **xyl_default**
Light xyL Default Set/Light xyL Default Set Unack

esp_ble_mesh_server_recv_light_xyl_range_set_t **xyl_range**
Light xyL Range Set/Light xyL Range Set Unack

esp_ble_mesh_server_recv_light_lc_mode_set_t **lc_mode**
Light LC Mode Set/Light LC Mode Set Unack

esp_ble_mesh_server_recv_light_lc_om_set_t **lc_om**
Light LC OM Set/Light LC OM Set Unack

esp_ble_mesh_server_recv_light_lc_light_onoff_set_t **lc_light_onoff**
Light LC Light OnOff Set/Light LC Light OnOff Set Unack

esp_ble_mesh_server_recv_light_lc_property_set_t **lc_property**
Light LC Property Set/Light LC Property Set Unack

union esp_ble_mesh_lighting_server_recv_status_msg_t
#include <esp_ble_mesh_lighting_model_api.h> Lighting Server Model received status message union.

Public Members

esp_ble_mesh_server_recv_sensor_status_t **sensor_status**
Sensor Status

union esp_ble_mesh_lighting_server_cb_value_t
#include <esp_ble_mesh_lighting_model_api.h> Lighting Server Model callback value union.

Public Members

esp_ble_mesh_lighting_server_state_change_t **state_change**
ESP_BLE_MESH_LIGHTING_SERVER_STATE_CHANGE_EVT

esp_ble_mesh_lighting_server_recv_get_msg_t **get**
ESP_BLE_MESH_LIGHTING_SERVER_RECV_GET_MSG_EVT

esp_ble_mesh_lighting_server_recv_set_msg_t **set**
ESP_BLE_MESH_LIGHTING_SERVER_RECV_SET_MSG_EVT

esp_ble_mesh_lighting_server_recv_status_msg_t **status**
ESP_BLE_MESH_LIGHTING_SERVER_RECV_STATUS_MSG_EVT

Structures

struct esp_ble_mesh_light_lightness_set_t
Bluetooth Mesh Light Lightness Client Model Get and Set parameters structure.

Parameters of Light Lightness Set

Public Members

bool **op_en**
Indicate if optional parameters are included

uint16_t **lightness**
Target value of light lightness actual state

uint8_t **tid**
Transaction ID

uint8_t **trans_time**
Time to complete state transition (optional)

uint8_t **delay**
Indicate message execution delay (C.1)

struct esp_ble_mesh_light_lightness_linear_set_t
Parameters of Light Lightness Linear Set

Public Members

bool **op_en**
Indicate if optional parameters are included

uint16_t **lightness**
Target value of light lightness linear state

uint8_t **tid**
Transaction ID

uint8_t **trans_time**
Time to complete state transition (optional)

uint8_t **delay**
Indicate message execution delay (C.1)

struct esp_ble_mesh_light_lightness_default_set_t

Parameter of Light Lightness Default Set

Public Members

uint16_t **lightness**

The value of the Light Lightness Default state

struct esp_ble_mesh_light_lightness_range_set_t

Parameters of Light Lightness Range Set

Public Members

uint16_t **range_min**

Value of range min field of light lightness range state

uint16_t **range_max**

Value of range max field of light lightness range state

struct esp_ble_mesh_light_ctl_set_t

Parameters of Light CTL Set

Public Members

bool **op_en**

Indicate if optional parameters are included

uint16_t **ctl_lightness**

Target value of light ctl lightness state

uint16_t **ctl_temperatrue**

Target value of light ctl temperature state

int16_t **ctl_delta_uv**

Target value of light ctl delta UV state

uint8_t **tid**

Transaction ID

uint8_t **trans_time**

Time to complete state transition (optional)

uint8_t **delay**

Indicate message execution delay (C.1)

struct esp_ble_mesh_light_ctl_temperature_set_t

Parameters of Light CTL Temperature Set

Public Members

bool **op_en**

Indicate if optional parameters are included

uint16_t **ctl_temperatrue**

Target value of light ctl temperature state

int16_t **ctl_delta_uv**

Target value of light ctl delta UV state

uint8_t **tid**

Transaction ID

uint8_t **trans_time**

Time to complete state transition (optional)

uint8_t **delay**

Indicate message execution delay (C.1)

struct esp_ble_mesh_light_ctl_temperature_range_set_t

Parameters of Light CTL Temperature Range Set

Public Members

uint16_t **range_min**

Value of temperature range min field of light ctl temperature range state

uint16_t **range_max**

Value of temperature range max field of light ctl temperature range state

struct esp_ble_mesh_light_ctl_default_set_t

Parameters of Light CTL Default Set

Public Members

uint16_t **lightness**

Value of light lightness default state

uint16_t **temperature**

Value of light temperature default state

int16_t **delta_uv**

Value of light delta UV default state

struct esp_ble_mesh_light_hsl_set_t

Parameters of Light HSL Set

Public Members

bool **op_en**

Indicate if optional parameters are included

uint16_t **hsl_lightness**

Target value of light hsl lightness state

uint16_t **hsl_hue**

Target value of light hsl hue state

uint16_t **hsl_saturation**

Target value of light hsl saturation state

uint8_t **tid**

Transaction ID

uint8_t **trans_time**

Time to complete state transition (optional)

uint8_t **delay**

Indicate message execution delay (C.1)

struct esp_ble_mesh_light_hsl_hue_set_t

Parameters of Light HSL Hue Set

Public Members

bool **op_en**

Indicate if optional parameters are included

uint16_t **hue**

Target value of light hsl hue state

uint8_t **tid**

Transaction ID

uint8_t **trans_time**

Time to complete state transition (optional)

uint8_t **delay**

Indicate message execution delay (C.1)

struct esp_ble_mesh_light_hsl_saturation_set_t

Parameters of Light HSL Saturation Set

Public Members

bool **op_en**

Indicate if optional parameters are included

uint16_t **saturation**

Target value of light hsl hue state

uint8_t **tid**

Transaction ID

uint8_t **trans_time**

Time to complete state transition (optional)

uint8_t **delay**

Indicate message execution delay (C.1)

struct esp_ble_mesh_light_hsl_default_set_t

Parameters of Light HSL Default Set

Public Members

uint16_t **lightness**

Value of light lightness default state

uint16_t **hue**

Value of light hue default state

uint16_t **saturation**

Value of light saturation default state

struct esp_ble_mesh_light_hsl_range_set_t

Parameters of Light HSL Range Set

Public Members

uint16_t **hue_range_min**

Value of hue range min field of light hsl hue range state

uint16_t **hue_range_max**

Value of hue range max field of light hsl hue range state

uint16_t **saturation_range_min**

Value of saturation range min field of light hsl saturation range state

uint16_t **saturation_range_max**

Value of saturation range max field of light hsl saturation range state

struct esp_ble_mesh_light_xyl_set_t

Parameters of Light xyL Set

Public Members

bool **op_en**

Indicate whether optional parameters included

uint16_t **xyl_lightness**

The target value of the Light xyL Lightness state

uint16_t **xyl_x**

The target value of the Light xyL x state

uint16_t **xyl_y**

The target value of the Light xyL y state

uint8_t **tid**

Transaction Identifier

uint8_t **trans_time**

Time to complete state transition (optional)

uint8_t **delay**

Indicate message execution delay (C.1)

struct esp_ble_mesh_light_xyl_default_set_t

Parameters of Light xyL Default Set

Public Members

uint16_t **lightness**

The value of the Light Lightness Default state

uint16_t **xyl_x**

The value of the Light xyL x Default state

uint16_t **xyl_y**

The value of the Light xyL y Default state

struct esp_ble_mesh_light_xyl_range_set_t

Parameters of Light xyL Range Set

Public Members

uint16_t **xyl_x_range_min**

The value of the xyL x Range Min field of the Light xyL x Range state

uint16_t **xyl_x_range_max**

The value of the xyL x Range Max field of the Light xyL x Range state

uint16_t **xyl_y_range_min**

The value of the xyL y Range Min field of the Light xyL y Range state

uint16_t **xyl_y_range_max**

The value of the xyL y Range Max field of the Light xyL y Range state

struct esp_ble_mesh_light_lc_mode_set_t

Parameter of Light LC Mode Set

Public Members

uint8_t **mode**

The target value of the Light LC Mode state

struct esp_ble_mesh_light_lc_om_set_t

Parameter of Light LC OM Set

Public Members

uint8_t **mode**

The target value of the Light LC Occupancy Mode state

struct esp_ble_mesh_light_lc_light_onoff_set_t

Parameters of Light LC Light OnOff Set

Public Members

bool **op_en**

Indicate whether optional parameters included

uint8_t **light_onoff**

The target value of the Light LC Light OnOff state

uint8_t **tid**

Transaction Identifier

uint8_t **trans_time**

Time to complete state transition (optional)

uint8_t **delay**

Indicate message execution delay (C.1)

struct esp_ble_mesh_light_lc_property_get_t

Parameter of Light LC Property Get

Public Members

uint16_t **property_id**

Property ID identifying a Light LC Property

struct esp_ble_mesh_light_lc_property_set_t

Parameters of Light LC Property Set

Public Members

uint16_t **property_id**

Property ID identifying a Light LC Property

struct net_buf_simple ***property_value**

Raw value for the Light LC Property

struct esp_ble_mesh_light_lightness_status_cb_t

Bluetooth Mesh Light Lightness Client Model Get and Set callback parameters structure.

Parameters of Light Lightness Status

Public Members

bool **op_en**

Indicate if optional parameters are included

uint16_t **present_lightness**

Current value of light lightness actual state

uint16_t **target_lightness**

Target value of light lightness actual state (optional)

uint8_t **remain_time**

Time to complete state transition (C.1)

struct esp_ble_mesh_light_lightness_linear_status_cb_t

Parameters of Light Lightness Linear Status

Public Members

bool **op_en**

Indicate if optional parameters are included

uint16_t **present_lightness**

Current value of light lightness linear state

uint16_t **target_lightness**

Target value of light lightness linear state (optional)

uint8_t **remain_time**

Time to complete state transition (C.1)

struct esp_ble_mesh_light_lightness_last_status_cb_t

Parameter of Light Lightness Last Status

Public Members

uint16_t **lightness**

The value of the Light Lightness Last state

struct esp_ble_mesh_light_lightness_default_status_cb_t

Parameter of Light Lightness Default Status

Public Members

uint16_t **lightness**

The value of the Light Lightness default State

struct esp_ble_mesh_light_lightness_range_status_cb_t

Parameters of Light Lightness Range Status

Public Members

uint8_t **status_code**

Status Code for the request message

uint16_t **range_min**

Value of range min field of light lightness range state

uint16_t **range_max**

Value of range max field of light lightness range state

struct esp_ble_mesh_light_ctl_status_cb_t

Parameters of Light CTL Status

Public Members

bool **op_en**

Indicate if optional parameters are included

uint16_t **present_ctl_lightness**

Current value of light ctl lightness state

uint16_t **present_ctl_temperature**

Current value of light ctl temperature state

uint16_t **target_ctl_lightness**

Target value of light ctl lightness state (optional)

uint16_t **target_ctl_temperature**

Target value of light ctl temperature state (C.1)

uint8_t **remain_time**

Time to complete state transition (C.1)

struct esp_ble_mesh_light_ctl_temperature_status_cb_t

Parameters of Light CTL Temperature Status

Public Members

bool **op_en**

Indicate if optional parameters are included

uint16_t **present_ctl_temperature**

Current value of light ctl temperature state

uint16_t **present_ctl_delta_uv**

Current value of light ctl delta UV state

uint16_t **target_ctl_temperature**

Target value of light ctl temperature state (optional)

uint16_t **target_ctl_delta_uv**

Target value of light ctl delta UV state (C.1)

uint8_t **remain_time**

Time to complete state transition (C.1)

struct esp_ble_mesh_light_ctl_temperature_range_status_cb_t

Parameters of Light CTL Temperature Range Status

Public Members

uint8_t **status_code**

Status code for the request message

uint16_t **range_min**

Value of temperature range min field of light ctl temperature range state

uint16_t **range_max**

Value of temperature range max field of light ctl temperature range state

struct esp_ble_mesh_light_ctl_default_status_cb_t

Parameters of Light CTL Default Status

Public Members

uint16_t **lightness**

Value of light lightness default state

uint16_t **temperature**

Value of light temperature default state

int16_t **delta_uv**

Value of light delta UV default state

struct esp_ble_mesh_light_hsl_status_cb_t

Parameters of Light HSL Status

Public Members

bool **op_en**

Indicate if optional parameters are included

uint16_t **hsl_lightness**

Current value of light hsl lightness state

uint16_t **hsl_hue**

Current value of light hsl hue state

uint16_t **hsl_saturation**

Current value of light hsl saturation state

uint8_t **remain_time**

Time to complete state transition (optional)

struct esp_ble_mesh_light_hsl_target_status_cb_t

Parameters of Light HSL Target Status

Public Members

bool **op_en**

Indicate if optional parameters are included

uint16_t **hsl_lightness_target**

Target value of light hsl lightness state

uint16_t **hsl_hue_target**

Target value of light hsl hue state

uint16_t **hsl_saturation_target**

Target value of light hsl saturation state

uint8_t **remain_time**

Time to complete state transition (optional)

struct esp_ble_mesh_light_hsl_hue_status_cb_t

Parameters of Light HSL Hue Status

Public Members

bool **op_en**

Indicate if optional parameters are included

uint16_t **present_hue**

Current value of light hsl hue state

uint16_t **target_hue**

Target value of light hsl hue state (optional)

uint8_t **remain_time**

Time to complete state transition (C.1)

struct esp_ble_mesh_light_hsl_saturation_status_cb_t

Parameters of Light HSL Saturation Status

Public Members

bool **op_en**

Indicate if optional parameters are included

uint16_t **present_saturation**

Current value of light hsl saturation state

uint16_t **target_saturation**

Target value of light hsl saturation state (optional)

uint8_t **remain_time**

Time to complete state transition (C.1)

struct esp_ble_mesh_light_hsl_default_status_cb_t

Parameters of Light HSL Default Status

Public Members

uint16_t **lightness**

Value of light lightness default state

uint16_t **hue**

Value of light hue default state

uint16_t **saturation**

Value of light saturation default state

struct esp_ble_mesh_light_hsl_range_status_cb_t

Parameters of Light HSL Range Status

Public Members

uint8_t **status_code**

Status code for the request message

uint16_t **hue_range_min**

Value of hue range min field of light hsl hue range state

uint16_t **hue_range_max**

Value of hue range max field of light hsl hue range state

uint16_t **saturation_range_min**

Value of saturation range min field of light hsl saturation range state

uint16_t **saturation_range_max**

Value of saturation range max field of light hsl saturation range state

struct esp_ble_mesh_light_xyl_status_cb_t

Parameters of Light xyL Status

Public Members

bool **op_en**

Indicate whether optional parameters included

uint16_t **xyl_lightness**

The present value of the Light xyL Lightness state

uint16_t **xyl_x**

The present value of the Light xyL x state

uint16_t **xyl_y**

The present value of the Light xyL y state

uint8_t **remain_time**

Time to complete state transition (optional)

struct esp_ble_mesh_light_xyl_target_status_cb_t

Parameters of Light xyL Target Status

Public Members

bool **op_en**

Indicate whether optional parameters included

uint16_t **target_xyl_lightness**

The target value of the Light xyL Lightness state

uint16_t **target_xyl_x**

The target value of the Light xyL x state

uint16_t **target_xyl_y**

The target value of the Light xyL y state

uint8_t **remain_time**

Time to complete state transition (optional)

struct esp_ble_mesh_light_xyl_default_status_cb_t

Parameters of Light xyL Default Status

Public Members

uint16_t **lightness**

The value of the Light Lightness Default state

uint16_t **xyl_x**

The value of the Light xyL x Default state

uint16_t **xyl_y**

The value of the Light xyL y Default state

struct esp_ble_mesh_light_xyl_range_status_cb_t

Parameters of Light xyL Range Status

Public Members

uint8_t **status_code**

Status Code for the requesting message

uint16_t **xyl_x_range_min**

The value of the xyL x Range Min field of the Light xyL x Range state

uint16_t **xyl_x_range_max**

The value of the xyL x Range Max field of the Light xyL x Range state

uint16_t **xyl_y_range_min**

The value of the xyL y Range Min field of the Light xyL y Range state

uint16_t **xyl_y_range_max**

The value of the xyL y Range Max field of the Light xyL y Range state

struct esp_ble_mesh_light_lc_mode_status_cb_t

Parameter of Light LC Mode Status

Public Members

uint8_t **mode**

The present value of the Light LC Mode state

struct esp_ble_mesh_light_lc_om_status_cb_t

Parameter of Light LC OM Status

Public Members

uint8_t **mode**

The present value of the Light LC Occupancy Mode state

struct esp_ble_mesh_light_lc_light_onoff_status_cb_t

Parameters of Light LC Light OnOff Status

Public Members

bool **op_en**

Indicate whether optional parameters included

uint8_t **present_light_onoff**

The present value of the Light LC Light OnOff state

uint8_t **target_light_onoff**

The target value of the Light LC Light OnOff state (Optional)

uint8_t **remain_time**

Time to complete state transition (C.1)

struct esp_ble_mesh_light_lc_property_status_cb_t

Parameters of Light LC Property Status

Public Members

uint16_t **property_id**

Property ID identifying a Light LC Property

struct net_buf_simple ***property_value**

Raw value for the Light LC Property

struct esp_ble_mesh_light_client_cb_param_t

Lighting Client Model callback parameters

Public Members

int **error_code**

Appropriate error code

esp_ble_mesh_client_common_param_t ***params**

The client common parameters.

esp_ble_mesh_light_client_status_cb_t **status_cb**

The light status message callback values

struct esp_ble_mesh_light_lightness_state_t

Parameters of Light Lightness state

Public Members

uint16_t **lightness_linear**

The present value of Light Lightness Linear state

uint16_t **target_lightness_linear**

The target value of Light Lightness Linear state

uint16_t **lightness_actual**

The present value of Light Lightness Actual state

uint16_t **target_lightness_actual**

The target value of Light Lightness Actual state

uint16_t **lightness_last**

The value of Light Lightness Last state

uint16_t **lightness_default**

The value of Light Lightness Default state

uint8_t **status_code**

The status code of setting Light Lightness Range state

uint16_t **lightness_range_min**

The minimum value of Light Lightness Range state

uint16_t **lightness_range_max**

The maximum value of Light Lightness Range state

struct esp_ble_mesh_light_lightness_srv_t

User data of Light Lightness Server Model

Public Members

esp_ble_mesh_model_t ***model**

Pointer to the Lighting Lightness Server Model. Initialized internally.

esp_ble_mesh_server_rsp_ctrl_t **rsp_ctrl**

Response control of the server model received messages

esp_ble_mesh_light_lightness_state_t ***state**

Parameters of the Light Lightness state

esp_ble_mesh_last_msg_info_t **last**

Parameters of the last received set message

esp_ble_mesh_state_transition_t **actual_transition**

Parameters of state transition

esp_ble_mesh_state_transition_t **linear_transition**

Parameters of state transition

int32_t **tt_delta_lightness_actual**

Delta change value of lightness actual state transition

int32_t **tt_delta_lightness_linear**

Delta change value of lightness linear state transition

struct esp_ble_mesh_light_lightness_setup_srv_t

User data of Light Lightness Setup Server Model

Public Members

esp_ble_mesh_model_t ***model**

Pointer to the Lighting Lightness Setup Server Model. Initialized internally.

esp_ble_mesh_server_rsp_ctrl_t **rsp_ctrl**

Response control of the server model received messages

esp_ble_mesh_light_lightness_state_t ***state**

Parameters of the Light Lightness state

struct esp_ble_mesh_light_ctl_state_t

Parameters of Light CTL state

Public Members

uint16_t **lightness**

The present value of Light CTL Lightness state

uint16_t **target_lightness**
: The target value of Light CTL Lightness state

uint16_t **temperature**
: The present value of Light CTL Temperature state

uint16_t **target_temperature**
: The target value of Light CTL Temperature state

int16_t **delta_uv**
: The present value of Light CTL Delta UV state

int16_t **target_delta_uv**
: The target value of Light CTL Delta UV state

uint8_t **status_code**
: The statue code of setting Light CTL Temperature Range state

uint16_t **temperature_range_min**
: The minimum value of Light CTL Temperature Range state

uint16_t **temperature_range_max**
: The maximum value of Light CTL Temperature Range state

uint16_t **lightness_default**
: The value of Light Lightness Default state

uint16_t **temperature_default**
: The value of Light CTL Temperature Default state

int16_t **delta_uv_default**
: The value of Light CTL Delta UV Default state

struct esp_ble_mesh_light_ctl_srv_t
: User data of Light CTL Server Model

Public Members

esp_ble_mesh_model_t ***model**
: Pointer to the Lighting CTL Server Model. Initialized internally.

esp_ble_mesh_server_rsp_ctrl_t **rsp_ctrl**
: Response control of the server model received messages

esp_ble_mesh_light_ctl_state_t ***state**
: Parameters of the Light CTL state

esp_ble_mesh_last_msg_info_t **last**
: Parameters of the last received set message

esp_ble_mesh_state_transition_t **transition**

Parameters of state transition

int32_t **tt_delta_lightness**

Delta change value of lightness state transition

int32_t **tt_delta_temperature**

Delta change value of temperature state transition

int32_t **tt_delta_delta_uv**

Delta change value of delta uv state transition

struct esp_ble_mesh_light_ctl_setup_srv_t

User data of Light CTL Setup Server Model

Public Members

esp_ble_mesh_model_t ***model**

Pointer to the Lighting CTL Setup Server Model. Initialized internally.

esp_ble_mesh_server_rsp_ctrl_t **rsp_ctrl**

Response control of the server model received messages

esp_ble_mesh_light_ctl_state_t ***state**

Parameters of the Light CTL state

struct esp_ble_mesh_light_ctl_temp_srv_t

User data of Light CTL Temperature Server Model

Public Members

esp_ble_mesh_model_t ***model**

Pointer to the Lighting CTL Temperature Server Model. Initialized internally.

esp_ble_mesh_server_rsp_ctrl_t **rsp_ctrl**

Response control of the server model received messages

esp_ble_mesh_light_ctl_state_t ***state**

Parameters of the Light CTL state

esp_ble_mesh_last_msg_info_t **last**

Parameters of the last received set message

esp_ble_mesh_state_transition_t **transition**

Parameters of state transition

int32_t **tt_delta_temperature**

Delta change value of temperature state transition

int32_t **tt_delta_delta_uv**

Delta change value of delta uv state transition

struct esp_ble_mesh_light_hsl_state_t

Parameters of Light HSL state

Public Members

uint16_t **lightness**

The present value of Light HSL Lightness state

uint16_t **target_lightness**

The target value of Light HSL Lightness state

uint16_t **hue**

The present value of Light HSL Hue state

uint16_t **target_hue**

The target value of Light HSL Hue state

uint16_t **saturation**

The present value of Light HSL Saturation state

uint16_t **target_saturation**

The target value of Light HSL Saturation state

uint16_t **lightness_default**

The value of Light Lightness Default state

uint16_t **hue_default**

The value of Light HSL Hue Default state

uint16_t **saturation_default**

The value of Light HSL Saturation Default state

uint8_t **status_code**

The status code of setting Light HSL Hue & Saturation Range state

uint16_t **hue_range_min**

The minimum value of Light HSL Hue Range state

uint16_t **hue_range_max**

The maximum value of Light HSL Hue Range state

uint16_t **saturation_range_min**

The minimum value of Light HSL Saturation state

uint16_t **saturation_range_max**

The maximum value of Light HSL Saturation state

struct esp_ble_mesh_light_hsl_srv_t

User data of Light HSL Server Model

Public Members

esp_ble_mesh_model_t ***model**

Pointer to the Lighting HSL Server Model. Initialized internally.

esp_ble_mesh_server_rsp_ctrl_t **rsp_ctrl**

Response control of the server model received messages

esp_ble_mesh_light_hsl_state_t ***state**

Parameters of the Light HSL state

esp_ble_mesh_last_msg_info_t **last**

Parameters of the last received set message

esp_ble_mesh_state_transition_t **transition**

Parameters of state transition

int32_t **tt_delta_lightness**

Delta change value of lightness state transition

int32_t **tt_delta_hue**

Delta change value of hue state transition

int32_t **tt_delta_saturation**

Delta change value of saturation state transition

struct esp_ble_mesh_light_hsl_setup_srv_t

User data of Light HSL Setup Server Model

Public Members

esp_ble_mesh_model_t ***model**

Pointer to the Lighting HSL Setup Server Model. Initialized internally.

esp_ble_mesh_server_rsp_ctrl_t **rsp_ctrl**

Response control of the server model received messages

esp_ble_mesh_light_hsl_state_t ***state**

Parameters of the Light HSL state

struct esp_ble_mesh_light_hsl_hue_srv_t

User data of Light HSL Hue Server Model

Public Members

esp_ble_mesh_model_t ***model**

Pointer to the Lighting HSL Hue Server Model. Initialized internally.

esp_ble_mesh_server_rsp_ctrl_t **rsp_ctrl**

Response control of the server model received messages

esp_ble_mesh_light_hsl_state_t ***state**

Parameters of the Light HSL state

esp_ble_mesh_last_msg_info_t **last**

Parameters of the last received set message

esp_ble_mesh_state_transition_t **transition**

Parameters of state transition

int32_t **tt_delta_hue**

Delta change value of hue state transition

struct esp_ble_mesh_light_hsl_sat_srv_t

User data of Light HSL Saturation Server Model

Public Members

esp_ble_mesh_model_t ***model**

Pointer to the Lighting HSL Saturation Server Model. Initialized internally.

esp_ble_mesh_server_rsp_ctrl_t **rsp_ctrl**

Response control of the server model received messages

esp_ble_mesh_light_hsl_state_t ***state**

Parameters of the Light HSL state

esp_ble_mesh_last_msg_info_t **last**

Parameters of the last received set message

esp_ble_mesh_state_transition_t **transition**

Parameters of state transition

int32_t **tt_delta_saturation**

Delta change value of saturation state transition

struct esp_ble_mesh_light_xyl_state_t

Parameters of Light xyL state

Public Members

uint16_t **lightness**

The present value of Light xyL Lightness state

uint16_t **target_lightness**

The target value of Light xyL Lightness state

uint16_t **x**

The present value of Light xyL x state

uint16_t **target_x**

The target value of Light xyL x state

uint16_t **y**

The present value of Light xyL y state

uint16_t **target_y**

The target value of Light xyL y state

uint16_t **lightness_default**

The value of Light Lightness Default state

uint16_t **x_default**

The value of Light xyL x Default state

uint16_t **y_default**

The value of Light xyL y Default state

uint8_t **status_code**

The status code of setting Light xyL x & y Range state

uint16_t **x_range_min**

The minimum value of Light xyL x Range state

uint16_t **x_range_max**

The maximum value of Light xyL x Range state

uint16_t **y_range_min**

The minimum value of Light xyL y Range state

uint16_t **y_range_max**

The maximum value of Light xyL y Range state

struct esp_ble_mesh_light_xyl_srv_t

User data of Light xyL Server Model

Public Members

esp_ble_mesh_model_t ***model**

Pointer to the Lighting xyL Server Model. Initialized internally.

esp_ble_mesh_server_rsp_ctrl_t **rsp_ctrl**

Response control of the server model received messages

esp_ble_mesh_light_xyl_state_t ***state**

Parameters of the Light xyL state

esp_ble_mesh_last_msg_info_t **last**

Parameters of the last received set message

esp_ble_mesh_state_transition_t **transition**

Parameters of state transition

int32_t **tt_delta_lightness**

Delta change value of lightness state transition

int32_t **tt_delta_x**

Delta change value of x state transition

int32_t **tt_delta_y**

Delta change value of y state transition

struct esp_ble_mesh_light_xyl_setup_srv_t

User data of Light xyL Setup Server Model

Public Members

esp_ble_mesh_model_t ***model**

Pointer to the Lighting xyL Setup Server Model. Initialized internally.

esp_ble_mesh_server_rsp_ctrl_t **rsp_ctrl**

Response control of the server model received messages

esp_ble_mesh_light_xyl_state_t ***state**

Parameters of the Light xyL state

struct esp_ble_mesh_light_lc_state_t

Parameters of Light LC states

Public Members

uint32_t **mode**

0b0 The controller is turned off.

- The binding with the Light Lightness state is disabled. 0b1 The controller is turned on.

- The binding with the Light Lightness state is enabled.The value of Light LC Mode state

uint32_t **occupancy_mode**

The value of Light LC Occupancy Mode state

uint32_t **light_onoff**

The present value of Light LC Light OnOff state

uint32_t **target_light_onoff**

The target value of Light LC Light OnOff state

uint32_t **occupancy**

The value of Light LC Occupancy state

uint32_t **ambient_luxlevel**

The value of Light LC Ambient LuxLevel state

uint16_t **linear_output**

1. Light LC Linear Output = max((Lightness Out)^2/65535, Regulator Output)
2. If the Light LC Mode state is set to 0b1, the binding is enabled and upon a change of the Light LC Linear Output state, the following operation shall be performed: Light Lightness Linear = Light LC Linear Output
3. If the Light LC Mode state is set to 0b0, the binding is disabled (i.e., upon a change of the Light LC Linear Output state, no operation on the Light Lightness Linear state is performed).The value of Light LC Linear Output state

struct esp_ble_mesh_light_lc_property_state_t

Parameters of Light Property states. The Light LC Property states are read / write states that determine the configuration of a Light Lightness Controller. Each state is represented by a device property and is controlled by Light LC Property messages.

Public Members

uint32_t **time_occupancy_delay**

A timing state that determines the delay for changing the Light LC Occupancy state upon receiving a Sensor Status message from an occupancy sensor.The value of Light LC Time Occupancy Delay state

uint32_t **time_fade_on**

A timing state that determines the time the controlled lights fade to the level determined by the Light LC Lightness On state.The value of Light LC Time Fade On state

uint32_t **time_run_on**

A timing state that determines the time the controlled lights stay at the level determined by the Light LC Lightness On state.The value of Light LC Time Run On state

uint32_t **time_fade**

A timing state that determines the time the controlled lights fade from the level determined by the Light LC Lightness On state to the level determined by the Light Lightness Prolong state.The value of Light LC Time Fade state

uint32_t **time_prolong**

A timing state that determines the time the controlled lights stay at the level determined by the Light LC Lightness Prolong state.The value of Light LC Time Prolong state

uint32_t **time_fade_standby_auto**

A timing state that determines the time the controlled lights fade from the level determined by the Light LC Lightness Prolong state to the level determined by the Light LC Lightness Standby state when the transition is automatic.The value of Light LC Time Fade Standby Auto state

uint32_t **time_fade_standby_manual**

A timing state that determines the time the controlled lights fade from the level determined by the Light LC Lightness Prolong state to the level determined by the Light LC Lightness Standby state when the transition is triggered by a change in the Light LC Light OnOff state.The value of Light LC Time Fade Standby Manual state

uint16_t **lightness_on**

A lightness state that determines the perceptive light lightness at the Occupancy and Run internal controller states.The value of Light LC Lightness On state

uint16_t **lightness_prolong**

A lightness state that determines the light lightness at the Prolong internal controller state.The value of Light LC Lightness Prolong state

uint16_t **lightness_standby**

A lightness state that determines the light lightness at the Standby internal controller state.The value of Light LC Lightness Standby state

uint16_t **ambient_luxlevel_on**

A uint16 state representing the Ambient LuxLevel level that determines if the controller transitions from the Light Control Standby state.The value of Light LC Ambient LuxLevel On state

uint16_t **ambient_luxlevel_prolong**

A uint16 state representing the required Ambient LuxLevel level in the Prolong state.The value of Light LC Ambient LuxLevel Prolong state

uint16_t **ambient_luxlevel_standby**

A uint16 state representing the required Ambient LuxLevel level in the Standby state.The value of Light LC Ambient LuxLevel Standby state

float **regulator_kiu**

A float32 state representing the integral coefficient that determines the integral part of the equation defining the output of the Light LC PI Feedback Regulator, when Light LC Ambient LuxLevel

is less than LuxLevel Out. Valid range: 0.0 ~ 1000.0. The default value is 250.0.The value of Light LC Regulator Kiu state

float **regulator_kid**

A float32 state representing the integral coefficient that determines the integral part of the equation defining the output of the Light LC PI Feedback Regulator, when Light LC Ambient LuxLevel is greater than or equal to the value of the LuxLevel Out state. Valid range: 0.0 ~ 1000.0. The default value is 25.0.The value of Light LC Regulator Kid state

float **regulator_kpu**

A float32 state representing the proportional coefficient that determines the proportional part of the equation defining the output of the Light LC PI Feedback Regulator, when Light LC Ambient LuxLevel is less than the value of the LuxLevel Out state. Valid range: 0.0 ~ 1000.0. The default value is 80.0.The value of Light LC Regulator Kpu state

float **regulator_kpd**

A float32 state representing the proportional coefficient that determines the proportional part of the equation defining the output of the Light LC PI Feedback Regulator, when Light LC Ambient LuxLevel is greater than or equal to the value of the LuxLevel Out state. Valid range: 0.0 ~ 1000.0. The default value is 80.0.The value of Light LC Regulator Kpd state

int8_t **regulator_accuracy**

A int8 state representing the percentage accuracy of the Light LC PI Feedback Regulator. Valid range: 0.0 ~ 100.0. The default value is 2.0.The value of Light LC Regulator Accuracy state

uint32_t **set_occupancy_to_1_delay**

If the message Raw field contains a Raw Value for the Time Since Motion Sensed device property, which represents a value less than or equal to the value of the Light LC Occupancy Delay state, it shall delay setting the Light LC Occupancy state to 0b1 by the difference between the value of the Light LC Occupancy Delay state and the received Time Since Motion value. The value of the difference between value of the Light LC Occupancy Delay state and the received Time Since Motion value

struct esp_ble_mesh_light_lc_state_machine_t

Parameters of Light LC state machine

Public Members

uint8_t **fade_on**

The value of transition time of Light LC Time Fade On

uint8_t **fade**

The value of transition time of Light LC Time Fade

uint8_t **fade_standby_auto**

The value of transition time of Light LC Time Fade Standby Auto

uint8_t **fade_standby_manual**

The value of transition time of Light LC Time Fade Standby Manual

struct *esp_ble_mesh_light_lc_state_machine_t*::[**anonymous**] **trans_time**

The Fade On, Fade, Fade Standby Auto, and Fade Standby Manual states are transition states that define the transition of the Lightness Out and LuxLevel Out states. This transition can be started as a result of the Light LC State Machine change or as a result of receiving the Light LC Light OnOff Set or Light LC Light Set Unacknowledged message.The value of transition time

esp_ble_mesh_lc_state_t **state**

The value of Light LC state machine state

struct k_delayed_work **timer**

Timer of Light LC state machine

struct esp_ble_mesh_light_control_t

Parameters of Light Lightness controller

Public Members

esp_ble_mesh_light_lc_state_t **state**

Parameters of Light LC state

esp_ble_mesh_light_lc_property_state_t **prop_state**

Parameters of Light LC Property state

esp_ble_mesh_light_lc_state_machine_t **state_machine**

Parameters of Light LC state machine

struct esp_ble_mesh_light_lc_srv_t

User data of Light LC Server Model

Public Members

esp_ble_mesh_model_t ***model**

Pointer to the Lighting LC Server Model. Initialized internally.

esp_ble_mesh_server_rsp_ctrl_t **rsp_ctrl**

Response control of the server model received messages

esp_ble_mesh_light_control_t ***lc**

Parameters of the Light controller

esp_ble_mesh_last_msg_info_t **last**

Parameters of the last received set message

esp_ble_mesh_state_transition_t **transition**

Parameters of state transition

struct esp_ble_mesh_light_lc_setup_srv_t

User data of Light LC Setup Server Model

Public Members

esp_ble_mesh_model_t ***model**

Pointer to the Lighting LC Setup Server Model. Initialized internally.

esp_ble_mesh_server_rsp_ctrl_t **rsp_ctrl**

Response control of the server model received messages

esp_ble_mesh_light_control_t ***lc**

Parameters of the Light controller

struct esp_ble_mesh_state_change_light_lightness_set_t

Parameter of Light Lightness Actual state change event

Public Members

uint16_t **lightness**

The value of Light Lightness Actual state

struct esp_ble_mesh_state_change_light_lightness_linear_set_t

Parameter of Light Lightness Linear state change event

Public Members

uint16_t **lightness**

The value of Light Lightness Linear state

struct esp_ble_mesh_state_change_light_lightness_default_set_t

Parameter of Light Lightness Default state change event

Public Members

uint16_t **lightness**

The value of Light Lightness Default state

struct esp_ble_mesh_state_change_light_lightness_range_set_t

Parameters of Light Lightness Range state change event

Public Members

uint16_t **range_min**

The minimum value of Light Lightness Range state

uint16_t **range_max**

The maximum value of Light Lightness Range state

struct esp_ble_mesh_state_change_light_ctl_set_t

Parameters of Light CTL state change event

Public Members

uint16_t **lightness**

The value of Light CTL Lightness state

uint16_t **temperature**

The value of Light CTL Temperature state

int16_t **delta_uv**

The value of Light CTL Delta UV state

struct esp_ble_mesh_state_change_light_ctl_temperature_set_t

Parameters of Light CTL Temperature state change event

Public Members

uint16_t **temperature**

The value of Light CTL Temperature state

int16_t **delta_uv**

The value of Light CTL Delta UV state

struct esp_ble_mesh_state_change_light_ctl_temperature_range_set_t

Parameters of Light CTL Temperature Range state change event

Public Members

uint16_t **range_min**

The minimum value of Light CTL Temperature Range state

uint16_t **range_max**

The maximum value of Light CTL Temperature Range state

struct esp_ble_mesh_state_change_light_ctl_default_set_t

Parameters of Light CTL Default state change event

Public Members

uint16_t **lightness**

The value of Light Lightness Default state

uint16_t **temperature**

The value of Light CTL Temperature Default state

int16_t **delta_uv**

The value of Light CTL Delta UV Default state

struct esp_ble_mesh_state_change_light_hsl_set_t

Parameters of Light HSL state change event

Public Members

uint16_t **lightness**

The value of Light HSL Lightness state

uint16_t **hue**

The value of Light HSL Hue state

uint16_t **saturation**

The value of Light HSL Saturation state

struct esp_ble_mesh_state_change_light_hsl_hue_set_t

Parameter of Light HSL Hue state change event

Public Members

uint16_t **hue**

The value of Light HSL Hue state

struct esp_ble_mesh_state_change_light_hsl_saturation_set_t

Parameter of Light HSL Saturation state change event

Public Members

uint16_t **saturation**

The value of Light HSL Saturation state

struct esp_ble_mesh_state_change_light_hsl_default_set_t

Parameters of Light HSL Default state change event

Public Members

uint16_t **lightness**

The value of Light HSL Lightness Default state

uint16_t **hue**

The value of Light HSL Hue Default state

uint16_t **saturation**

The value of Light HSL Saturation Default state

struct esp_ble_mesh_state_change_light_hsl_range_set_t

Parameters of Light HSL Range state change event

Public Members

uint16_t **hue_range_min**

The minimum hue value of Light HSL Range state

uint16_t **hue_range_max**

The maximum hue value of Light HSL Range state

uint16_t **saturation_range_min**

The minimum saturation value of Light HSL Range state

uint16_t **saturation_range_max**

The maximum saturation value of Light HSL Range state

struct esp_ble_mesh_state_change_light_xyl_set_t

Parameters of Light xyL state change event

Public Members

uint16_t **lightness**

The value of Light xyL Lightness state

uint16_t **x**

The value of Light xyL x state

uint16_t **y**

The value of Light xyL y state

struct esp_ble_mesh_state_change_light_xyl_default_set_t

Parameters of Light xyL Default state change event

Public Members

uint16_t **lightness**

The value of Light Lightness Default state

uint16_t **x**

The value of Light xyL x Default state

uint16_t **y**

The value of Light xyL y Default state

struct esp_ble_mesh_state_change_light_xyl_range_set_t

Parameters of Light xyL Range state change event

Public Members

uint16_t **x_range_min**

The minimum value of Light xyL x Range state

uint16_t **x_range_max**

The maximum value of Light xyL x Range state

uint16_t **y_range_min**

The minimum value of Light xyL y Range state

uint16_t **y_range_max**

The maximum value of Light xyL y Range state

struct esp_ble_mesh_state_change_light_lc_mode_set_t

Parameter of Light LC Mode state change event

Public Members

uint8_t **mode**

The value of Light LC Mode state

struct esp_ble_mesh_state_change_light_lc_om_set_t

Parameter of Light LC Occupancy Mode state change event

Public Members

uint8_t **mode**

The value of Light LC Occupancy Mode state

struct esp_ble_mesh_state_change_light_lc_light_onoff_set_t

Parameter of Light LC Light OnOff state change event

Public Members

uint8_t **onoff**

The value of Light LC Light OnOff state

struct esp_ble_mesh_state_change_light_lc_property_set_t

Parameters of Light LC Property state change event

Public Members

uint16_t **property_id**

The property id of Light LC Property state

struct net_buf_simple ***property_value**

The property value of Light LC Property state

struct esp_ble_mesh_state_change_sensor_status_t

Parameters of Sensor Status state change event

Public Members

uint16_t **property_id**

The value of Sensor Property ID

uint8_t **occupancy**

The value of Light LC Occupancy state

uint32_t **set_occupancy_to_1_delay**

The value of Light LC Set Occupancy to 1 Delay state

uint32_t **ambient_luxlevel**

The value of Light LC Ambient Luxlevel state

union *esp_ble_mesh_state_change_sensor_status_t*::[**anonymous**] **state**

Parameters of Sensor Status related state

struct esp_ble_mesh_server_recv_light_lc_property_get_t

Context of the received Light LC Property Get message

Public Members

uint16_t **property_id**

Property ID identifying a Light LC Property

struct esp_ble_mesh_server_recv_light_lightness_set_t

Context of the received Light Lightness Set message

Public Members

bool **op_en**

Indicate if optional parameters are included

uint16_t **lightness**

Target value of light lightness actual state

uint8_t **tid**

Transaction ID

uint8_t **trans_time**

Time to complete state transition (optional)

uint8_t **delay**

Indicate message execution delay (C.1)

struct esp_ble_mesh_server_recv_light_lightness_linear_set_t

Context of the received Light Lightness Linear Set message

Public Members

bool **op_en**

Indicate if optional parameters are included

uint16_t **lightness**

Target value of light lightness linear state

uint8_t **tid**

Transaction ID

uint8_t **trans_time**

Time to complete state transition (optional)

uint8_t **delay**

Indicate message execution delay (C.1)

struct esp_ble_mesh_server_recv_light_lightness_default_set_t

Context of the received Light Lightness Default Set message

Public Members

uint16_t **lightness**

The value of the Light Lightness Default state

struct esp_ble_mesh_server_recv_light_lightness_range_set_t

Context of the received Light Lightness Range Set message

Public Members

uint16_t **range_min**

Value of range min field of light lightness range state

uint16_t **range_max**

Value of range max field of light lightness range state

struct esp_ble_mesh_server_recv_light_ctl_set_t

Context of the received Light CTL Set message

Public Members

bool **op_en**

Indicate if optional parameters are included

uint16_t **lightness**

Target value of light ctl lightness state

uint16_t **temperature**

Target value of light ctl temperature state

int16_t **delta_uv**

Target value of light ctl delta UV state

uint8_t **tid**

Transaction ID

uint8_t **trans_time**

Time to complete state transition (optional)

uint8_t **delay**

Indicate message execution delay (C.1)

struct esp_ble_mesh_server_recv_light_ctl_temperature_set_t

Context of the received Light CTL Temperature Set message

Public Members

bool **op_en**

Indicate if optional parameters are included

uint16_t **temperature**

Target value of light ctl temperature state

int16_t **delta_uv**

Target value of light ctl delta UV state

uint8_t **tid**

Transaction ID

uint8_t **trans_time**

Time to complete state transition (optional)

uint8_t **delay**

Indicate message execution delay (C.1)

struct esp_ble_mesh_server_recv_light_ctl_temperature_range_set_t

Context of the received Light CTL Temperature Range Set message

Public Members

uint16_t **range_min**

Value of temperature range min field of light ctl temperature range state

uint16_t **range_max**

Value of temperature range max field of light ctl temperature range state

struct esp_ble_mesh_server_recv_light_ctl_default_set_t

Context of the received Light CTL Default Set message

Public Members

uint16_t **lightness**

Value of light lightness default state

uint16_t **temperature**

Value of light temperature default state

int16_t **delta_uv**

Value of light delta UV default state

struct esp_ble_mesh_server_recv_light_hsl_set_t

Context of the received Light HSL Set message

Public Members

bool **op_en**

Indicate if optional parameters are included

uint16_t **lightness**

Target value of light hsl lightness state

uint16_t **hue**

Target value of light hsl hue state

uint16_t **saturation**

Target value of light hsl saturation state

uint8_t **tid**

Transaction ID

uint8_t **trans_time**

Time to complete state transition (optional)

uint8_t **delay**

Indicate message execution delay (C.1)

struct esp_ble_mesh_server_recv_light_hsl_hue_set_t

Context of the received Light HSL Hue Set message

Public Members

bool **op_en**

Indicate if optional parameters are included

uint16_t **hue**

Target value of light hsl hue state

uint8_t **tid**

Transaction ID

uint8_t **trans_time**

Time to complete state transition (optional)

uint8_t **delay**

Indicate message execution delay (C.1)

struct esp_ble_mesh_server_recv_light_hsl_saturation_set_t

Context of the received Light HSL Saturation Set message

Public Members

bool **op_en**

Indicate if optional parameters are included

uint16_t **saturation**

Target value of light hsl hue state

uint8_t **tid**

Transaction ID

uint8_t **trans_time**

Time to complete state transition (optional)

uint8_t **delay**

Indicate message execution delay (C.1)

struct esp_ble_mesh_server_recv_light_hsl_default_set_t

Context of the received Light HSL Default Set message

Public Members

uint16_t **lightness**

Value of light lightness default state

uint16_t **hue**

Value of light hue default state

uint16_t **saturation**

Value of light saturation default state

struct esp_ble_mesh_server_recv_light_hsl_range_set_t

Context of the received Light HSL Range Set message

Public Members

uint16_t **hue_range_min**

Value of hue range min field of light hsl hue range state

uint16_t **hue_range_max**

Value of hue range max field of light hsl hue range state

uint16_t **saturation_range_min**

Value of saturation range min field of light hsl saturation range state

uint16_t **saturation_range_max**

Value of saturation range max field of light hsl saturation range state

struct esp_ble_mesh_server_recv_light_xyl_set_t

Context of the received Light xyL Set message

Public Members

bool **op_en**

Indicate whether optional parameters included

uint16_t **lightness**

The target value of the Light xyL Lightness state

uint16_t **x**

The target value of the Light xyL x state

uint16_t **y**

The target value of the Light xyL y state

uint8_t **tid**

Transaction Identifier

uint8_t **trans_time**

Time to complete state transition (optional)

uint8_t **delay**

Indicate message execution delay (C.1)

struct esp_ble_mesh_server_recv_light_xyl_default_set_t

Context of the received Light xyL Default Set message

Public Members

uint16_t **lightness**

The value of the Light Lightness Default state

uint16_t **x**

The value of the Light xyL x Default state

uint16_t **y**

The value of the Light xyL y Default state

struct esp_ble_mesh_server_recv_light_xyl_range_set_t

Context of the received Light xyl Range Set message

Public Members

uint16_t **x_range_min**

The value of the xyL x Range Min field of the Light xyL x Range state

uint16_t **x_range_max**

The value of the xyL x Range Max field of the Light xyL x Range state

uint16_t **y_range_min**

The value of the xyL y Range Min field of the Light xyL y Range state

uint16_t **y_range_max**

The value of the xyL y Range Max field of the Light xyL y Range state

struct esp_ble_mesh_server_recv_light_lc_mode_set_t

Context of the received Light LC Mode Set message

Public Members

uint8_t **mode**

The target value of the Light LC Mode state

struct esp_ble_mesh_server_recv_light_lc_om_set_t

Context of the received Light OM Set message

Public Members

uint8_t **mode**

The target value of the Light LC Occupancy Mode state

struct esp_ble_mesh_server_recv_light_lc_light_onoff_set_t

Context of the received Light LC Light OnOff Set message

Public Members

bool **op_en**

Indicate whether optional parameters included

uint8_t **light_onoff**

The target value of the Light LC Light OnOff state

uint8_t **tid**

Transaction Identifier

uint8_t **trans_time**

Time to complete state transition (optional)

uint8_t **delay**

Indicate message execution delay (C.1)

struct esp_ble_mesh_server_recv_light_lc_property_set_t

Context of the received Light LC Property Set message

Public Members

uint16_t **property_id**

Property ID identifying a Light LC Property

struct net_buf_simple ***property_value**

Raw value for the Light LC Property

struct esp_ble_mesh_server_recv_sensor_status_t

Context of the received Sensor Status message

Public Members

struct net_buf_simple ***data**

Value of sensor data state (optional)

struct esp_ble_mesh_lighting_server_cb_param_t

Lighting Server Model callback parameters

Public Members

esp_ble_mesh_model_t ***model**

Pointer to Lighting Server Models

esp_ble_mesh_msg_ctx_t **ctx**

Context of the received messages

esp_ble_mesh_lighting_server_cb_value_t **value**

Value of the received Lighting Messages

Macros

ESP_BLE_MESH_MODEL_LIGHT_LIGHTNESS_CLI(cli_pub, cli_data)

Define a new Light Lightness Client Model.

Note This API needs to be called for each element on which the application needs to have a Light Lightness Client Model.

Return New Light Lightness Client Model instance.

Parameters

- `cli_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- `cli_data`: Pointer to the unique struct *esp_ble_mesh_client_t*.

ESP_BLE_MESH_MODEL_LIGHT_CTL_CLI(cli_pub, cli_data)

Define a new Light CTL Client Model.

Note This API needs to be called for each element on which the application needs to have a Light CTL Client Model.

Return New Light CTL Client Model instance.

Parameters

- `cli_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- `cli_data`: Pointer to the unique struct *esp_ble_mesh_client_t*.

ESP_BLE_MESH_MODEL_LIGHT_HSL_CLI(cli_pub, cli_data)

Define a new Light HSL Client Model.

Note This API needs to be called for each element on which the application needs to have a Light HSL Client Model.

Return New Light HSL Client Model instance.

Parameters

- `cli_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- `cli_data`: Pointer to the unique struct *esp_ble_mesh_client_t*.

`ESP_BLE_MESH_MODEL_LIGHT_XYL_CLI`(cli_pub, cli_data)

Define a new Light xyL Client Model.

Note This API needs to be called for each element on which the application needs to have a Light xyL Client Model.

Return New Light xyL Client Model instance.

Parameters

- `cli_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- `cli_data`: Pointer to the unique struct *esp_ble_mesh_client_t*.

`ESP_BLE_MESH_MODEL_LIGHT_LC_CLI`(cli_pub, cli_data)

Define a new Light LC Client Model.

Note This API needs to be called for each element on which the application needs to have a Light LC Client Model.

Return New Light LC Client Model instance.

Parameters

- `cli_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- `cli_data`: Pointer to the unique struct *esp_ble_mesh_client_t*.

`ESP_BLE_MESH_MODEL_LIGHT_LIGHTNESS_SRV`(srv_pub, srv_data)

Lighting Server Models related context.

Define a new Light Lightness Server Model.

Note 1. The Light Lightness Server model extends the Generic Power OnOff Server model and the Generic Level Server model. When this model is present on an Element, the corresponding Light Lightness Setup Server model shall also be present.

1. This model shall support model publication and model subscription.

Return New Light Lightness Server Model instance.

Parameters

- `srv_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- `srv_data`: Pointer to the unique struct *esp_ble_mesh_light_lightness_srv_t*.

ESP_BLE_MESH_MODEL_LIGHT_LIGHTNESS_SETUP_SRV(srv_pub, srv_data)

Define a new Light Lightness Setup Server Model.

Note 1. The Light Lightness Setup Server model extends the Light Lightness Server model and the Generic Power OnOff Setup Server model.

1. This model shall support model subscription.

Return New Light Lightness Setup Server Model instance.

Parameters

- `srv_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- `srv_data`: Pointer to the unique struct *esp_ble_mesh_light_lightness_setup_srv_t*.

ESP_BLE_MESH_MODEL_LIGHT_CTL_SRV(srv_pub, srv_data)

Define a new Light CTL Server Model.

Note 1. The Light CTL Server model extends the Light Lightness Server model. When this model is present on an Element, the corresponding Light CTL Temperature Server model and the corresponding Light CTL Setup Server model shall also be present.

1. This model shall support model publication and model subscription.
2. The model requires two elements: the main element and the Temperature element. The Temperature element contains the corresponding Light CTL Temperature Server model and an instance of a Generic Level state bound to the Light CTL Temperature state on the Temperature element. The Light CTL Temperature state on the Temperature element is bound to the Light CTL state on the main element.

Return New Light CTL Server Model instance.

Parameters

- `srv_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- `srv_data`: Pointer to the unique struct *esp_ble_mesh_light_ctl_srv_t*.

ESP_BLE_MESH_MODEL_LIGHT_CTL_SETUP_SRV(srv_pub, srv_data)

Define a new Light CTL Setup Server Model.

Note 1. The Light CTL Setup Server model extends the Light CTL Server and the Light Lightness Setup Server.

1. This model shall support model subscription.

Return New Light CTL Setup Server Model instance.

Parameters

- `srv_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.

- `srv_data`: Pointer to the unique struct *esp_ble_mesh_light_ctl_setup_srv_t*.

`ESP_BLE_MESH_MODEL_LIGHT_CTL_TEMP_SRV`(srv_pub, srv_data)

Define a new Light CTL Temperature Server Model.

Note 1. The Light CTL Temperature Server model extends the Generic Level Server model.

1. This model shall support model publication and model subscription.

Return New Light CTL Temperature Server Model instance.

Parameters

- `srv_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- `srv_data`: Pointer to the unique struct *esp_ble_mesh_light_ctl_temp_srv_t*.

`ESP_BLE_MESH_MODEL_LIGHT_HSL_SRV`(srv_pub, srv_data)

Define a new Light HSL Server Model.

Note 1. The Light HSL Server model extends the Light Lightness Server model. When this model is present on an Element, the corresponding Light HSL Hue Server model and the corresponding Light HSL Saturation Server model and the corresponding Light HSL Setup Server model shall also be present.

1. This model shall support model publication and model subscription.
2. The model requires three elements: the main element and the Hue element and the Saturation element. The Hue element contains the corresponding Light HSL Hue Server model and an instance of a Generic Level state bound to the Light HSL Hue state on the Hue element. The Saturation element contains the corresponding Light HSL Saturation Server model and an instance of a Generic Level state bound to the Light HSL Saturation state on the Saturation element. The Light HSL Hue state on the Hue element is bound to the Light HSL state on the main element and the Light HSL Saturation state on the Saturation element is bound to the Light HSL state on the main element.

Return New Light HSL Server Model instance.

Parameters

- `srv_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- `srv_data`: Pointer to the unique struct *esp_ble_mesh_light_hsl_srv_t*.

`ESP_BLE_MESH_MODEL_LIGHT_HSL_SETUP_SRV`(srv_pub, srv_data)

Define a new Light HSL Setup Server Model.

Note 1. The Light HSL Setup Server model extends the Light HSL Server and the Light Lightness Setup Server.

1. This model shall support model subscription.

Return New Light HSL Setup Server Model instance.

Parameters

- `srv_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- `srv_data`: Pointer to the unique struct *esp_ble_mesh_light_hsl_setup_srv_t*.

`ESP_BLE_MESH_MODEL_LIGHT_HSL_HUE_SRV`(srv_pub, srv_data)

Define a new Light HSL Hue Server Model.

Note 1. The Light HSL Hue Server model extends the Generic Level Server model. This model is associated with the Light HSL Server model.

1. This model shall support model publication and model subscription.

Return New Light HSL Hue Server Model instance.

Parameters

- `srv_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- `srv_data`: Pointer to the unique struct *esp_ble_mesh_light_hsl_hue_srv_t*.

`ESP_BLE_MESH_MODEL_LIGHT_HSL_SAT_SRV`(srv_pub, srv_data)

Define a new Light HSL Saturation Server Model.

Note 1. The Light HSL Saturation Server model extends the Generic Level Server model. This model is associated with the Light HSL Server model.

1. This model shall support model publication and model subscription.

Return New Light HSL Saturation Server Model instance.

Parameters

- `srv_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- `srv_data`: Pointer to the unique struct *esp_ble_mesh_light_hsl_sat_srv_t*.

`ESP_BLE_MESH_MODEL_LIGHT_XYL_SRV`(srv_pub, srv_data)

Define a new Light xyL Server Model.

Note 1. The Light xyL Server model extends the Light Lightness Server model. When this model is present on an Element, the corresponding Light xyL Setup Server model shall also be present.

1. This model shall support model publication and model subscription.

Return New Light xyL Server Model instance.

Parameters

- `srv_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- `srv_data`: Pointer to the unique struct *esp_ble_mesh_light_xyl_srv_t*.

`ESP_BLE_MESH_MODEL_LIGHT_XYL_SETUP_SRV`(srv_pub, srv_data)

Define a new Light xyL Setup Server Model.

Note 1. The Light xyL Setup Server model extends the Light xyL Server and the Light Lightness Setup Server.

1. This model shall support model subscription.

Return New Light xyL Setup Server Model instance.

Parameters

- `srv_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- `srv_data`: Pointer to the unique struct *esp_ble_mesh_light_xyl_setup_srv_t*.

`ESP_BLE_MESH_MODEL_LIGHT_LC_SRV`(srv_pub, srv_data)

Define a new Light LC Server Model.

Note 1. The Light LC (Lightness Control) Server model extends the Light Lightness Server model and the Generic OnOff Server model. When this model is present on an Element, the corresponding Light LC Setup Server model shall also be present.

1. This model shall support model publication and model subscription.
2. This model may be used to represent an element that is a client to a Sensor Server model and controls the Light Lightness Actual state via defined state bindings.

Return New Light LC Server Model instance.

Parameters

- `srv_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- `srv_data`: Pointer to the unique struct *esp_ble_mesh_light_lc_srv_t*.

`ESP_BLE_MESH_MODEL_LIGHT_LC_SETUP_SRV`(srv_pub, srv_data)

Define a new Light LC Setup Server Model.

Note 1. The Light LC (Lightness Control) Setup model extends the Light LC Server model.

1. This model shall support model publication and model subscription.
2. This model may be used to configure setup parameters for the Light LC Server model.

Return New Light LC Setup Server Model instance.

Parameters

- `srv_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- `srv_data`: Pointer to the unique struct *esp_ble_mesh_light_lc_setup_srv_t*.

Type Definitions

`typedef` void (*`esp_ble_mesh_light_client_cb_t`)(*esp_ble_mesh_light_client_cb_event_t* event, *esp_ble_mesh_light_client_cb_param_t* *param)

Bluetooth Mesh Light Client Model function.

Lighting Client Model callback function type

Parameters

- `event`: Event type
- `param`: Pointer to callback parameter

`typedef` void (*`esp_ble_mesh_lighting_server_cb_t`)(*esp_ble_mesh_lighting_server_cb_event_t* event, *esp_ble_mesh_lighting_server_cb_param_t* *param)

Bluetooth Mesh Lighting Server Model function.

Lighting Server Model callback function type

Parameters

- `event`: Event type
- `param`: Pointer to callback parameter

Enumerations

`enum esp_ble_mesh_light_client_cb_event_t`

This enum value is the event of Lighting Client Model

Values:

`ESP_BLE_MESH_LIGHT_CLIENT_GET_STATE_EVT`

`ESP_BLE_MESH_LIGHT_CLIENT_SET_STATE_EVT`

`ESP_BLE_MESH_LIGHT_CLIENT_PUBLISH_EVT`

`ESP_BLE_MESH_LIGHT_CLIENT_TIMEOUT_EVT`

`ESP_BLE_MESH_LIGHT_CLIENT_EVT_MAX`

`enum esp_ble_mesh_lc_state_t`

This enum value is the Light LC State Machine states

Values:

ESP_BLE_MESH_LC_OFF

ESP_BLE_MESH_LC_STANDBY

ESP_BLE_MESH_LC_FADE_ON

ESP_BLE_MESH_LC_RUN

ESP_BLE_MESH_LC_FADE

ESP_BLE_MESH_LC_PROLONG

ESP_BLE_MESH_LC_FADE_STANDBY_AUTO

ESP_BLE_MESH_LC_FADE_STANDBY_MANUAL

enum esp_ble_mesh_lighting_server_cb_event_t

This enum value is the event of Lighting Server Model

Values:

ESP_BLE_MESH_LIGHTING_SERVER_STATE_CHANGE_EVT

1. When get_auto_rsp is set to ESP_BLE_MESH_SERVER_AUTO_RSP, no event will be callback to the application layer when Lighting Get messages are received.
2. When set_auto_rsp is set to ESP_BLE_MESH_SERVER_AUTO_RSP, this event will be callback to the application layer when Lighting Set/Set Unack messages are received.

ESP_BLE_MESH_LIGHTING_SERVER_RECV_GET_MSG_EVT

When get_auto_rsp is set to ESP_BLE_MESH_SERVER_RSP_BY_APP, this event will be callback to the application layer when Lighting Get messages are received.

ESP_BLE_MESH_LIGHTING_SERVER_RECV_SET_MSG_EVT

When set_auto_rsp is set to ESP_BLE_MESH_SERVER_RSP_BY_APP, this event will be callback to the application layer when Lighting Set/Set Unack messages are received.

ESP_BLE_MESH_LIGHTING_SERVER_RECV_STATUS_MSG_EVT

When status_auto_rsp is set to ESP_BLE_MESH_SERVER_RSP_BY_APP, this event will be callback to the application layer when Sensor Status message is received.

ESP_BLE_MESH_LIGHTING_SERVER_EVT_MAX

Time and Scenes Client/Server Models

Header File

- bt/esp_ble_mesh/api/models/include/esp_ble_mesh_sensor_model_api.h

Functions

esp_err_t **esp_ble_mesh_register_sensor_client_callback**(*esp_ble_mesh_sensor_client_cb_t callback*)

Register BLE Mesh Sensor Client Model callback.

Return ESP_OK on success or error code otherwise.

Parameters

- `callback`: Pointer to the callback function.

esp_err_t **esp_ble_mesh_sensor_client_get_state**(*esp_ble_mesh_client_common_param_t* **params*, *esp_ble_mesh_sensor_client_get_state_t* **get_state*)

Get the value of Sensor Server Model states using the Sensor Client Model get messages.

Note If you want to know the opcodes and corresponding meanings accepted by this API, please refer to esp_ble_mesh_sensor_message_opcode_t in esp_ble_mesh_defs.h

Return ESP_OK on success or error code otherwise.

Parameters

- `params`: Pointer to BLE Mesh common client parameters.
- `get_state`: Pointer to sensor get message value. Shall not be set to NULL.

esp_err_t **esp_ble_mesh_sensor_client_set_state**(*esp_ble_mesh_client_common_param_t* **params*, *esp_ble_mesh_sensor_client_set_state_t* **set_state*)

Set the value of Sensor Server Model states using the Sensor Client Model set messages.

Note If you want to know the opcodes and corresponding meanings accepted by this API, please refer to esp_ble_mesh_sensor_message_opcode_t in esp_ble_mesh_defs.h

Return ESP_OK on success or error code otherwise.

Parameters

- `params`: Pointer to BLE Mesh common client parameters.
- `set_state`: Pointer to sensor set message value. Shall not be set to NULL.

esp_err_t **esp_ble_mesh_register_sensor_server_callback**(*esp_ble_mesh_sensor_server_cb_t callback*)

Register BLE Mesh Sensor Server Model callback.

Return ESP_OK on success or error code otherwise.

Parameters

- `callback`: Pointer to the callback function.

Unions

union esp_ble_mesh_sensor_client_get_state_t

#include <esp_ble_mesh_sensor_model_api.h> Sensor Client Model get message union.

Public Members

esp_ble_mesh_sensor_descriptor_get_t **descriptor_get**

For ESP_BLE_MESH_MODEL_OP_SENSOR_DESCRIPTOR_GET

esp_ble_mesh_sensor_cadence_get_t **cadence_get**

For ESP_BLE_MESH_MODEL_OP_SENSOR_CADENCE_GET

esp_ble_mesh_sensor_settings_get_t **settings_get**

For ESP_BLE_MESH_MODEL_OP_SENSOR_SETTINGS_GET

esp_ble_mesh_sensor_setting_get_t **setting_get**

For ESP_BLE_MESH_MODEL_OP_SENSOR_SETTING_GET

esp_ble_mesh_sensor_get_t **sensor_get**

For ESP_BLE_MESH_MODEL_OP_SENSOR_GET

esp_ble_mesh_sensor_column_get_t **column_get**

For ESP_BLE_MESH_MODEL_OP_SENSOR_COLUMN_GET

esp_ble_mesh_sensor_series_get_t **series_get**

For ESP_BLE_MESH_MODEL_OP_SENSOR_SERIES_GET

union esp_ble_mesh_sensor_client_set_state_t

#include <esp_ble_mesh_sensor_model_api.h> Sensor Client Model set message union.

Public Members

esp_ble_mesh_sensor_cadence_set_t **cadence_set**

For ESP_BLE_MESH_MODEL_OP_SENSOR_CADENCE_SET & ESP_BLE_MESH_MODEL_OP_SENSOR_CADENCE_SET_UNACK

esp_ble_mesh_sensor_setting_set_t **setting_set**

For ESP_BLE_MESH_MODEL_OP_SENSOR_SETTING_SET & ESP_BLE_MESH_MODEL_OP_SENSOR_SETTING_SET_UNACK

union esp_ble_mesh_sensor_client_status_cb_t

#include <esp_ble_mesh_sensor_model_api.h> Sensor Client Model received message union.

Public Members

esp_ble_mesh_sensor_descriptor_status_cb_t **descriptor_status**

For ESP_BLE_MESH_MODEL_OP_SENSOR_DESCRIPTOR_STATUS

esp_ble_mesh_sensor_cadence_status_cb_t **cadence_status**

For ESP_BLE_MESH_MODEL_OP_SENSOR_CADENCE_STATUS

esp_ble_mesh_sensor_settings_status_cb_t **settings_status**

For ESP_BLE_MESH_MODEL_OP_SENSOR_SETTINGS_STATUS

esp_ble_mesh_sensor_setting_status_cb_t **setting_status**

For ESP_BLE_MESH_MODEL_OP_SENSOR_SETTING_STATUS

esp_ble_mesh_sensor_status_cb_t **sensor_status**

For ESP_BLE_MESH_MODEL_OP_SENSOR_STATUS

esp_ble_mesh_sensor_column_status_cb_t **column_status**

For ESP_BLE_MESH_MODEL_OP_SENSOR_COLUMN_STATUS

esp_ble_mesh_sensor_series_status_cb_t **series_status**

For ESP_BLE_MESH_MODEL_OP_SENSOR_SERIES_STATUS

union esp_ble_mesh_sensor_server_state_change_t

#include <esp_ble_mesh_sensor_model_api.h> Sensor Server Model state change value union.

Public Members

esp_ble_mesh_state_change_sensor_cadence_set_t **sensor_cadence_set**

The recv_op in ctx can be used to decide which state is changed.Sensor Cadence Set

esp_ble_mesh_state_change_sensor_setting_set_t **sensor_setting_set**

Sensor Setting Set

union esp_ble_mesh_sensor_server_recv_get_msg_t

#include <esp_ble_mesh_sensor_model_api.h> Sensor Server Model received get message union.

Public Members

esp_ble_mesh_server_recv_sensor_descriptor_get_t **sensor_descriptor**

Sensor Descriptor Get

esp_ble_mesh_server_recv_sensor_cadence_get_t **sensor_cadence**

Sensor Cadence Get

esp_ble_mesh_server_recv_sensor_settings_get_t **sensor_settings**

Sensor Settings Get

esp_ble_mesh_server_recv_sensor_setting_get_t **sensor_setting**

Sensor Setting Get

esp_ble_mesh_server_recv_sensor_get_t **sensor_data**

Sensor Get

esp_ble_mesh_server_recv_sensor_column_get_t **sensor_column**

Sensor Column Get

esp_ble_mesh_server_recv_sensor_series_get_t **sensor_series**

Sensor Series Get

union esp_ble_mesh_sensor_server_recv_set_msg_t

#include <esp_ble_mesh_sensor_model_api.h> Sensor Server Model received set message union.

Public Members

esp_ble_mesh_server_recv_sensor_cadence_set_t **sensor_cadence**

Sensor Cadence Set

esp_ble_mesh_server_recv_sensor_setting_set_t **sensor_setting**

Sensor Setting Set

union esp_ble_mesh_sensor_server_cb_value_t

#include <esp_ble_mesh_sensor_model_api.h> Sensor Server Model callback value union.

Public Members

esp_ble_mesh_sensor_server_state_change_t **state_change**

ESP_BLE_MESH_SENSOR_SERVER_STATE_CHANGE_EVT

esp_ble_mesh_sensor_server_recv_get_msg_t **get**

ESP_BLE_MESH_SENSOR_SERVER_RECV_GET_MSG_EVT

esp_ble_mesh_sensor_server_recv_set_msg_t **set**

ESP_BLE_MESH_SENSOR_SERVER_RECV_SET_MSG_EVT

Structures

struct esp_ble_mesh_sensor_descriptor_get_t

Bluetooth Mesh Sensor Client Model Get and Set parameters structure.

Parameters of Sensor Descriptor Get

Public Members

bool **op_en**

Indicate if optional parameters are included

uint16_t **property_id**

Property ID of a sensor (optional)

struct esp_ble_mesh_sensor_cadence_get_t

Parameter of Sensor Cadence Get

Public Members

uint16_t **property_id**

Property ID of a sensor

struct esp_ble_mesh_sensor_cadence_set_t

Parameters of Sensor Cadence Set

Public Members

uint16_t **property_id**

Property ID for the sensor

uint8_t **fast_cadence_period_divisor**

Divisor for the publish period

uint8_t **status_trigger_type**

The unit and format of the Status Trigger Delta fields

struct net_buf_simple ***status_trigger_delta_down**

Delta down value that triggers a status message

struct net_buf_simple ***status_trigger_delta_up**

Delta up value that triggers a status message

uint8_t **status_min_interval**

Minimum interval between two consecutive Status messages

struct net_buf_simple ***fast_cadence_low**

Low value for the fast cadence range

struct net_buf_simple ***fast_cadence_high**

Fast value for the fast cadence range

struct esp_ble_mesh_sensor_settings_get_t

Parameter of Sensor Settings Get

Public Members

uint16_t **sensor_property_id**
Property ID of a sensor

struct esp_ble_mesh_sensor_setting_get_t
Parameters of Sensor Setting Get

Public Members

uint16_t **sensor_property_id**
Property ID of a sensor

uint16_t **sensor_setting_property_id**
Setting ID identifying a setting within a sensor

struct esp_ble_mesh_sensor_setting_set_t
Parameters of Sensor Setting Set

Public Members

uint16_t **sensor_property_id**
Property ID identifying a sensor

uint16_t **sensor_setting_property_id**
Setting ID identifying a setting within a sensor

struct net_buf_simple ***sensor_setting_raw**
Raw value for the setting

struct esp_ble_mesh_sensor_get_t
Parameters of Sensor Get

Public Members

bool **op_en**
Indicate if optional parameters are included

uint16_t **property_id**
Property ID for the sensor (optional)

struct esp_ble_mesh_sensor_column_get_t
Parameters of Sensor Column Get

Public Members

uint16_t **property_id**

Property identifying a sensor

struct net_buf_simple ***raw_value_x**

Raw value identifying a column

struct esp_ble_mesh_sensor_series_get_t

Parameters of Sensor Series Get

Public Members

bool **op_en**

Indicate if optional parameters are included

uint16_t **property_id**

Property identifying a sensor

struct net_buf_simple ***raw_value_x1**

Raw value identifying a starting column (optional)

struct net_buf_simple ***raw_value_x2**

Raw value identifying an ending column (C.1)

struct esp_ble_mesh_sensor_descriptor_status_cb_t

Bluetooth Mesh Sensor Client Model Get and Set callback parameters structure.

Parameter of Sensor Descriptor Status

Public Members

struct net_buf_simple ***descriptor**

Sequence of 8-octet sensor descriptors (optional)

struct esp_ble_mesh_sensor_cadence_status_cb_t

Parameters of Sensor Cadence Status

Public Members

uint16_t **property_id**

Property for the sensor

struct net_buf_simple ***sensor_cadence_value**

Value of sensor cadence state

struct esp_ble_mesh_sensor_settings_status_cb_t

Parameters of Sensor Settings Status

Public Members

uint16_t **sensor_property_id**
Property ID identifying a sensor

struct net_buf_simple ***sensor_setting_property_ids**
A sequence of N sensor setting property IDs (optional)

struct esp_ble_mesh_sensor_setting_status_cb_t
Parameters of Sensor Setting Status

Public Members

bool **op_en**
Indicate id optional parameters are included

uint16_t **sensor_property_id**
Property ID identifying a sensor

uint16_t **sensor_setting_property_id**
Setting ID identifying a setting within a sensor

uint8_t **sensor_setting_access**
Read/Write access rights for the setting (optional)

struct net_buf_simple ***sensor_setting_raw**
Raw value for the setting

struct esp_ble_mesh_sensor_status_cb_t
Parameter of Sensor Status

Public Members

struct net_buf_simple ***marshalled_sensor_data**
Value of sensor data state (optional)

struct esp_ble_mesh_sensor_column_status_cb_t
Parameters of Sensor Column Status

Public Members

uint16_t **property_id**
Property identifying a sensor and the Y axis

struct net_buf_simple ***sensor_column_value**
Left values of sensor column status

struct esp_ble_mesh_sensor_series_status_cb_t

Parameters of Sensor Series Status

Public Members

uint16_t **property_id**

Property identifying a sensor and the Y axis

struct net_buf_simple ***sensor_series_value**

Left values of sensor series status

struct esp_ble_mesh_sensor_client_cb_param_t

Sensor Client Model callback parameters

Public Members

int **error_code**

0: success, otherwise failure. For the error code values please refer to errno.h file. A negative sign is added to the standard error codes in errno.h.

esp_ble_mesh_client_common_param_t ***params**

The client common parameters.

esp_ble_mesh_sensor_client_status_cb_t **status_cb**

The sensor status message callback values

struct esp_ble_mesh_sensor_descriptor_t

Parameters of Sensor Descriptor state

Public Members

uint32_t **positive_tolerance**

The value of Sensor Positive Tolerance field

uint32_t **negative_tolerance**

The value of Sensor Negative Tolerance field

uint32_t **sampling_function**

The value of Sensor Sampling Function field

uint8_t **measure_period**

The value of Sensor Measurement Period field

uint8_t **update_interval**

The value of Sensor Update Interval field

struct esp_ble_mesh_sensor_setting_t

Parameters of Sensor Setting state

Public Members

uint16_t **property_id**

The value of Sensor Setting Property ID field

uint8_t **access**

The value of Sensor Setting Access field

struct net_buf_simple ***raw**

The value of Sensor Setting Raw field

struct esp_ble_mesh_sensor_cadence_t

Parameters of Sensor Cadence state

Public Members

uint8_t **period_divisor**

The value of Fast Cadence Period Divisor field

uint8_t **trigger_type**

The value of Status Trigger Type field

struct net_buf_simple ***trigger_delta_down**

Note: The parameter "size" in trigger_delta_down, trigger_delta_up, fast_cadence_low & fast_cadence_high indicates the exact length of these four parameters, and they are associated with the Sensor Property ID. Users need to initialize the "size" precisely.The value of Status Trigger Delta Down field

struct net_buf_simple ***trigger_delta_up**

The value of Status Trigger Delta Up field

uint8_t **min_interval**

The value of Status Min Interval field

struct net_buf_simple ***fast_cadence_low**

The value of Fast Cadence Low field

struct net_buf_simple ***fast_cadence_high**

The value of Fast Cadence High field

struct esp_ble_mesh_sensor_data_t

Parameters of Sensor Data state

Public Members

uint8_t **format**

Format A: The Length field is a 1-based uint4 value (valid range 0x0–0xF, representing range of

1 – 16). Format B: The Length field is a 1-based uint7 value (valid range 0x0–0x7F, representing range of 1 – 127). The value 0x7F represents a length of zero.The value of the Sensor Data format

uint8_t **length**

The value of the Sensor Data length

struct net_buf_simple ***raw_value**

The value of Sensor Data raw value

struct esp_ble_mesh_sensor_series_column_t

Parameters of Sensor Series Column state

Public Members

struct net_buf_simple ***raw_value_x**

The value of Sensor Raw Value X field

struct net_buf_simple ***column_width**

The value of Sensor Column Width field

struct net_buf_simple ***raw_value_y**

The value of Sensor Raw Value Y field

struct esp_ble_mesh_sensor_state_t

Parameters of Sensor states

Public Members

uint16_t **sensor_property_id**

The value of Sensor Property ID field

esp_ble_mesh_sensor_descriptor_t **descriptor**

Parameters of the Sensor Descriptor state

const uint8_t **setting_count**

Multiple Sensor Setting states may be present for each sensor. The Sensor Setting Property ID values shall be unique for each Sensor Property ID that identifies a sensor within an element.

esp_ble_mesh_sensor_setting_t ***settings**

Parameters of the Sensor Setting state

esp_ble_mesh_sensor_cadence_t ***cadence**

The Sensor Cadence state may be not supported by sensors based on device properties referencing "non-scalar characteristics" such as "histograms" or "composite characteristics" .Parameters of the Sensor Cadence state

esp_ble_mesh_sensor_data_t **sensor_data**

Parameters of the Sensor Data state

esp_ble_mesh_sensor_series_column_t **series_column**

Parameters of the Sensor Series Column state

struct esp_ble_mesh_sensor_srv_t

User data of Sensor Server Model

Public Members

esp_ble_mesh_model_t ***model**

Pointer to the Sensor Server Model. Initialized internally.

esp_ble_mesh_server_rsp_ctrl_t **rsp_ctrl**

Response control of the server model received messages

const uint8_t **state_count**

Sensor state count

esp_ble_mesh_sensor_state_t ***states**

Parameters of the Sensor states

struct esp_ble_mesh_sensor_setup_srv_t

User data of Sensor Setup Server Model

Public Members

esp_ble_mesh_model_t ***model**

Pointer to the Sensor Setup Server Model. Initialized internally.

esp_ble_mesh_server_rsp_ctrl_t **rsp_ctrl**

Response control of the server model received messages

const uint8_t **state_count**

Sensor state count

esp_ble_mesh_sensor_state_t ***states**

Parameters of the Sensor states

struct esp_ble_mesh_state_change_sensor_cadence_set_t

Parameters of Sensor Cadence Set state change event

Public Members

uint16_t **property_id**

The value of Sensor Property ID state

uint8_t **period_divisor**

The value of Fast Cadence Period Divisor state

uint8_t **trigger_type**
The value of Status Trigger Type state

struct net_buf_simple ***trigger_delta_down**
The value of Status Trigger Delta Down state

struct net_buf_simple ***trigger_delta_up**
The value of Status Trigger Delta Up state

uint8_t **min_interval**
The value of Status Min Interval state

struct net_buf_simple ***fast_cadence_low**
The value of Fast Cadence Low state

struct net_buf_simple ***fast_cadence_high**
The value of Fast Cadence High state

struct esp_ble_mesh_state_change_sensor_setting_set_t
Parameters of Sensor Setting Set state change event

Public Members

uint16_t **property_id**
The value of Sensor Property ID state

uint16_t **setting_property_id**
The value of Sensor Setting Property ID state

struct net_buf_simple ***setting_value**
The value of Sensor Property Value state

struct esp_ble_mesh_server_recv_sensor_descriptor_get_t
Context of the received Sensor Descriptor Get message

Public Members

bool **op_en**
Indicate if optional parameters are included

uint16_t **property_id**
Property ID of a sensor (optional)

struct esp_ble_mesh_server_recv_sensor_cadence_get_t
Context of the received Sensor Cadence Get message

Public Members

uint16_t **property_id**

Property ID of a sensor

struct esp_ble_mesh_server_recv_sensor_settings_get_t

Context of the received Sensor Settings Get message

Public Members

uint16_t **property_id**

Property ID of a sensor

struct esp_ble_mesh_server_recv_sensor_setting_get_t

Context of the received Sensor Setting Get message

Public Members

uint16_t **property_id**

Property ID of a sensor

uint16_t **setting_property_id**

Setting ID identifying a setting within a sensor

struct esp_ble_mesh_server_recv_sensor_get_t

Context of the received Sensor Get message

Public Members

bool **op_en**

Indicate if optional parameters are included

uint16_t **property_id**

Property ID for the sensor (optional)

struct esp_ble_mesh_server_recv_sensor_column_get_t

Context of the received Sensor Column Get message

Public Members

uint16_t **property_id**

Property identifying a sensor

struct net_buf_simple ***raw_value_x**

Raw value identifying a column

struct esp_ble_mesh_server_recv_sensor_series_get_t

Context of the received Sensor Series Get message

Public Members

bool **op_en**

Indicate if optional parameters are included

uint16_t **property_id**

Property identifying a sensor

struct net_buf_simple ***raw_value**

Raw value containing X1 and X2 (optional)

struct esp_ble_mesh_server_recv_sensor_cadence_set_t

Context of the received Sensor Cadence Set message

Public Members

uint16_t **property_id**

Property ID for the sensor

struct net_buf_simple ***cadence**

Value of Sensor Cadence state

struct esp_ble_mesh_server_recv_sensor_setting_set_t

Context of the received Sensor Setting Set message

Public Members

uint16_t **property_id**

Property ID identifying a sensor

uint16_t **setting_property_id**

Setting ID identifying a setting within a sensor

struct net_buf_simple ***setting_raw**

Raw value for the setting

struct esp_ble_mesh_sensor_server_cb_param_t

Sensor Server Model callback parameters

Public Members

esp_ble_mesh_model_t ***model**

Pointer to Sensor Server Models

esp_ble_mesh_msg_ctx_t **ctx**
Context of the received messages

esp_ble_mesh_sensor_server_cb_value_t **value**
Value of the received Sensor Messages

Macros

ESP_BLE_MESH_MODEL_SENSOR_CLI(cli_pub, cli_data)
Define a new Sensor Client Model.

Note This API needs to be called for each element on which the application needs to have a Sensor Client Model.

Return New Sensor Client Model instance.

Parameters

- **cli_pub**: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- **cli_data**: Pointer to the unique struct *esp_ble_mesh_client_t*.

ESP_BLE_MESH_MODEL_SENSOR_SRV(srv_pub, srv_data)
Sensor Server Models related context.

Define a new Sensor Server Model.

Note 1. The Sensor Server model is a root model. When this model is present on an element, the corresponding Sensor Setup Server model shall also be present.

1. This model shall support model publication and model subscription.

Return New Sensor Server Model instance.

Parameters

- **srv_pub**: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- **srv_data**: Pointer to the unique struct *esp_ble_mesh_sensor_srv_t*.

ESP_BLE_MESH_MODEL_SENSOR_SETUP_SRV(srv_pub, srv_data)
Define a new Sensor Setup Server Model.

Note 1. The Sensor Setup Server model extends the Sensor Server model.

1. This model shall support model publication and model subscription.

Return New Sensor Setup Server Model instance.

Parameters

- **srv_pub**: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.

- `srv_data`: Pointer to the unique struct *esp_ble_mesh_sensor_setup_srv_t*.

`ESP_BLE_MESH_INVALID_SENSOR_PROPERTY_ID`
Invalid Sensor Property ID

`ESP_BLE_MESH_SENSOR_PROPERTY_ID_LEN`
Length of Sensor Property ID

`ESP_BLE_MESH_SENSOR_DESCRIPTOR_LEN`
Length of Sensor Descriptor state

`ESP_BLE_MESH_SENSOR_UNSPECIFIED_POS_TOLERANCE`
Unspecified Sensor Positive Tolerance

`ESP_BLE_MESH_SENSOR_UNSPECIFIED_NEG_TOLERANCE`
Unspecified Sensor Negative Tolerance

`ESP_BLE_MESH_SENSOR_NOT_APPL_MEASURE_PERIOD`
Not applicable Sensor Measurement Period

`ESP_BLE_MESH_SENSOR_NOT_APPL_UPDATE_INTERVAL`
Not applicable Sensor Update Interval

`ESP_BLE_MESH_INVALID_SENSOR_SETTING_PROPERTY_ID`
Invalid Sensor Setting Property ID

`ESP_BLE_MESH_SENSOR_SETTING_PROPERTY_ID_LEN`
Length of Sensor Setting Property ID

`ESP_BLE_MESH_SENSOR_SETTING_ACCESS_LEN`
Length of Sensor Setting Access

`ESP_BLE_MESH_SENSOR_SETTING_ACCESS_READ`
Sensor Setting Access - Read

`ESP_BLE_MESH_SENSOR_SETTING_ACCESS_READ_WRITE`
Sensor Setting Access - Read & Write

`ESP_BLE_MESH_SENSOR_DIVISOR_TRIGGER_TYPE_LEN`
Length of Sensor Divisor Trigger Type

`ESP_BLE_MESH_SENSOR_STATUS_MIN_INTERVAL_LEN`
Length of Sensor Status Min Interval

`ESP_BLE_MESH_SENSOR_PERIOD_DIVISOR_MAX_VALUE`
Maximum value of Sensor Period Divisor

`ESP_BLE_MESH_SENSOR_STATUS_MIN_INTERVAL_MAX`
Maximum value of Sensor Status Min Interval

ESP_BLE_MESH_SENSOR_STATUS_TRIGGER_TYPE_CHAR
Sensor Status Trigger Type - Format Type of the characteristic that the Sensor Property ID state references

ESP_BLE_MESH_SENSOR_STATUS_TRIGGER_TYPE_UINT16
Sensor Status Trigger Type - Format Type "uint16"

ESP_BLE_MESH_SENSOR_DATA_FORMAT_A
Sensor Data Format A

ESP_BLE_MESH_SENSOR_DATA_FORMAT_B
Sensor Data Format B

ESP_BLE_MESH_SENSOR_DATA_FORMAT_A_MPID_LEN
MPID length of Sensor Data Format A

ESP_BLE_MESH_SENSOR_DATA_FORMAT_B_MPID_LEN
MPID length of Sensor Data Format B

ESP_BLE_MESH_SENSOR_DATA_ZERO_LEN
Zero length of Sensor Data.

Note: The Length field is a 1-based uint7 value (valid range 0x0–0x7F, representing range of 1–127). The value 0x7F represents a length of zero.

ESP_BLE_MESH_GET_SENSOR_DATA_FORMAT(_data)
Get format of the sensor data.

Note Multiple sensor data may be concatenated. Make sure the _data pointer is updated before getting the format of the corresponding sensor data.

Return Format of the sensor data.

Parameters

- `_data`: Pointer to the start of the sensor data.

ESP_BLE_MESH_GET_SENSOR_DATA_LENGTH(_data, _fmt)
Get length of the sensor data.

Note Multiple sensor data may be concatenated. Make sure the _data pointer is updated before getting the length of the corresponding sensor data.

Return Length (zero-based) of the sensor data.

Parameters

- `_data`: Pointer to the start of the sensor data.
- `_fmt`: Format of the sensor data.

ESP_BLE_MESH_GET_SENSOR_DATA_PROPERTY_ID(_data, _fmt)

Get Sensor Property ID of the sensor data.

Note Multiple sensor data may be concatenated. Make sure the _data pointer is updated before getting Sensor Property ID of the corresponding sensor data.

Return Sensor Property ID of the sensor data.

Parameters

- `_data`: Pointer to the start of the sensor data.
- `_fmt`: Format of the sensor data.

ESP_BLE_MESH_SENSOR_DATA_FORMAT_A_MPID(_len, _id)

Generate a MPID value for sensor data with Format A.

Note 1. The Format field is 0b0 and indicates that Format A is used.

1. The Length field is a 1-based uint4 value (valid range 0x0–0xF, representing range of 1–16).
2. The Property ID is an 11-bit bit field representing 11 LSb of a Property ID.
3. This format may be used for Property Values that are not longer than 16 octets and for Property IDs less than 0x0800.

Return 2-octet MPID value for sensor data with Format A.

Parameters

- `_len`: Length of Sensor Raw value.
- `_id`: Sensor Property ID.

ESP_BLE_MESH_SENSOR_DATA_FORMAT_B_MPID(_len, _id)

Generate a MPID value for sensor data with Format B.

Note 1. The Format field is 0b1 and indicates Format B is used.

1. The Length field is a 1-based uint7 value (valid range 0x0–0x7F, representing range of 1–127). The value 0x7F represents a length of zero.
2. The Property ID is a 16-bit bit field representing a Property ID.
3. This format may be used for Property Values not longer than 128 octets and for any Property IDs. Property values longer than 128 octets are not supported by the Sensor Status message.
4. Exclude the generated 1-octet value, the 2-octet Sensor Property ID

Return 3-octet MPID value for sensor data with Format B.

Parameters

- `_len`: Length of Sensor Raw value.

- `_id`: Sensor Property ID.

Type Definitions

`typedef` void (*`esp_ble_mesh_sensor_client_cb_t`)(*esp_ble_mesh_sensor_client_cb_event_t* event, *esp_ble_mesh_sensor_client_cb_param_t* *param)

Bluetooth Mesh Sensor Client Model function.

Sensor Client Model callback function type

Parameters

- `event`: Event type
- `param`: Pointer to callback parameter

`typedef` void (*`esp_ble_mesh_sensor_server_cb_t`)(*esp_ble_mesh_sensor_server_cb_event_t* event, *esp_ble_mesh_sensor_server_cb_param_t* *param)

Bluetooth Mesh Sensor Server Model function.

Sensor Server Model callback function type

Parameters

- `event`: Event type
- `param`: Pointer to callback parameter

Enumerations

`enum esp_ble_mesh_sensor_client_cb_event_t`

This enum value is the event of Sensor Client Model

Values:

`ESP_BLE_MESH_SENSOR_CLIENT_GET_STATE_EVT`

`ESP_BLE_MESH_SENSOR_CLIENT_SET_STATE_EVT`

`ESP_BLE_MESH_SENSOR_CLIENT_PUBLISH_EVT`

`ESP_BLE_MESH_SENSOR_CLIENT_TIMEOUT_EVT`

`ESP_BLE_MESH_SENSOR_CLIENT_EVT_MAX`

`enum esp_ble_mesh_sensor_sample_func`

This enum value is value of Sensor Sampling Function

Values:

`ESP_BLE_MESH_SAMPLE_FUNC_UNSPECIFIED`

ESP_BLE_MESH_SAMPLE_FUNC_INSTANTANEOUS

ESP_BLE_MESH_SAMPLE_FUNC_ARITHMETIC_MEAN

ESP_BLE_MESH_SAMPLE_FUNC_RMS

ESP_BLE_MESH_SAMPLE_FUNC_MAXIMUM

ESP_BLE_MESH_SAMPLE_FUNC_MINIMUM

ESP_BLE_MESH_SAMPLE_FUNC_ACCUMULATED

ESP_BLE_MESH_SAMPLE_FUNC_COUNT

enum esp_ble_mesh_sensor_server_cb_event_t

This enum value is the event of Sensor Server Model

Values:

ESP_BLE_MESH_SENSOR_SERVER_STATE_CHANGE_EVT

1. When get_auto_rsp is set to ESP_BLE_MESH_SERVER_AUTO_RSP, no event will be callback to the application layer when Sensor Get messages are received.
2. When set_auto_rsp is set to ESP_BLE_MESH_SERVER_AUTO_RSP, this event will be callback to the application layer when Sensor Set/Set Unack messages are received.

ESP_BLE_MESH_SENSOR_SERVER_RECV_GET_MSG_EVT

When get_auto_rsp is set to ESP_BLE_MESH_SERVER_RSP_BY_APP, this event will be callback to the application layer when Sensor Get messages are received.

ESP_BLE_MESH_SENSOR_SERVER_RECV_SET_MSG_EVT

When set_auto_rsp is set to ESP_BLE_MESH_SERVER_RSP_BY_APP, this event will be callback to the application layer when Sensor Set/Set Unack messages are received.

ESP_BLE_MESH_SENSOR_SERVER_EVT_MAX

Lighting Client/Server Models

Header File

- bt/esp_ble_mesh/api/models/include/esp_ble_mesh_time_scene_model_api.h

Functions

esp_err_t **esp_ble_mesh_register_time_scene_client_callback**(*esp_ble_mesh_time_scene_client_cb_t callback*)

Register BLE Mesh Time Scene Client Model callback.

Return ESP_OK on success or error code otherwise.

Parameters

- `callback`: Pointer to the callback function.

esp_err_t **esp_ble_mesh_time_scene_client_get_state**(*esp_ble_mesh_client_common_param_t* **params*, *esp_ble_mesh_time_scene_client_get_state_t* **get_state*)

Get the value of Time Scene Server Model states using the Time Scene Client Model get messages.

Note If you want to know the opcodes and corresponding meanings accepted by this API, please refer to esp_ble_mesh_time_scene_message_opcode_t in esp_ble_mesh_defs.h

Return ESP_OK on success or error code otherwise.

Parameters

- `params`: Pointer to BLE Mesh common client parameters.
- `get_state`: Pointer to time scene get message value. Shall not be set to NULL.

esp_err_t **esp_ble_mesh_time_scene_client_set_state**(*esp_ble_mesh_client_common_param_t* **params*, *esp_ble_mesh_time_scene_client_set_state_t* **set_state*)

Set the value of Time Scene Server Model states using the Time Scene Client Model set messages.

Note If you want to know the opcodes and corresponding meanings accepted by this API, please refer to esp_ble_mesh_time_scene_message_opcode_t in esp_ble_mesh_defs.h

Return ESP_OK on success or error code otherwise.

Parameters

- `params`: Pointer to BLE Mesh common client parameters.
- `set_state`: Pointer to time scene set message value. Shall not be set to NULL.

esp_err_t **esp_ble_mesh_register_time_scene_server_callback**(*esp_ble_mesh_time_scene_server_cb_t* *callback*)

Register BLE Mesh Time and Scenes Server Model callback.

Return ESP_OK on success or error code otherwise.

Parameters

- `callback`: Pointer to the callback function.

Unions

union esp_ble_mesh_time_scene_client_get_state_t

#include <esp_ble_mesh_time_scene_model_api.h> Time Scene Client Model get message union.

Public Members

esp_ble_mesh_scheduler_act_get_t **scheduler_act_get**

For ESP_BLE_MESH_MODEL_OP_SCHEDULER_ACT_GET

union esp_ble_mesh_time_scene_client_set_state_t

#include <esp_ble_mesh_time_scene_model_api.h> Time Scene Client Model set message union.

Public Members

esp_ble_mesh_time_set_t **time_set**

For ESP_BLE_MESH_MODEL_OP_TIME_SET

esp_ble_mesh_time_zone_set_t **time_zone_set**

For ESP_BLE_MESH_MODEL_OP_TIME_ZONE_SET

esp_ble_mesh_tai_utc_delta_set_t **tai_utc_delta_set**

For ESP_BLE_MESH_MODEL_OP_TAI_UTC_DELTA_SET

esp_ble_mesh_time_role_set_t **time_role_set**

For ESP_BLE_MESH_MODEL_OP_TIME_ROLE_SET

esp_ble_mesh_scene_store_t **scene_store**

For ESP_BLE_MESH_MODEL_OP_SCENE_STORE & ESP_BLE_MESH_MODEL_OP_SCENE_STORE

esp_ble_mesh_scene_recall_t **scene_recall**

For ESP_BLE_MESH_MODEL_OP_SCENE_RECALL & ESP_BLE_MESH_MODEL_OP_SCENE_RECA

esp_ble_mesh_scene_delete_t **scene_delete**

For ESP_BLE_MESH_MODEL_OP_SCENE_DELETE & ESP_BLE_MESH_MODEL_OP_SCENE_DELE

esp_ble_mesh_scheduler_act_set_t **scheduler_act_set**

For ESP_BLE_MESH_MODEL_OP_SCHEDULER_ACT_SET & ESP_BLE_MESH_MODEL_OP_SCHEDULER_ACT_SET_UNACK

union esp_ble_mesh_time_scene_client_status_cb_t

#include <esp_ble_mesh_time_scene_model_api.h> Time Scene Client Model received message union.

Public Members

esp_ble_mesh_time_status_cb_t **time_status**

For ESP_BLE_MESH_MODEL_OP_TIME_STATUS

esp_ble_mesh_time_zone_status_cb_t **time_zone_status**

For ESP_BLE_MESH_MODEL_OP_TIME_ZONE_STATUS

esp_ble_mesh_tai_utc_delta_status_cb_t **tai_utc_delta_status**

For ESP_BLE_MESH_MODEL_OP_TAI_UTC_DELTA_STATUS

esp_ble_mesh_time_role_status_cb_t **time_role_status**

For ESP_BLE_MESH_MODEL_OP_TIME_ROLE_STATUS

esp_ble_mesh_scene_status_cb_t **scene_status**

For ESP_BLE_MESH_MODEL_OP_SCENE_STATUS

esp_ble_mesh_scene_register_status_cb_t **scene_register_status**

For ESP_BLE_MESH_MODEL_OP_SCENE_REGISTER_STATUS

esp_ble_mesh_scheduler_status_cb_t **scheduler_status**

For ESP_BLE_MESH_MODEL_OP_SCHEDULER_STATUS

esp_ble_mesh_scheduler_act_status_cb_t **scheduler_act_status**

For ESP_BLE_MESH_MODEL_OP_SCHEDULER_ACT_STATUS

union esp_ble_mesh_time_scene_server_state_change_t

#include <esp_ble_mesh_time_scene_model_api.h> Time Scene Server Model state change value union.

Public Members

esp_ble_mesh_state_change_time_set_t **time_set**

The recv_op in ctx can be used to decide which state is changed.Time Set

esp_ble_mesh_state_change_time_status_t **time_status**

Time Status

esp_ble_mesh_state_change_time_zone_set_t **time_zone_set**

Time Zone Set

esp_ble_mesh_state_change_tai_utc_delta_set_t **tai_utc_delta_set**

TAI UTC Delta Set

esp_ble_mesh_state_change_time_role_set_t **time_role_set**

Time Role Set

esp_ble_mesh_state_change_scene_store_t **scene_store**

Scene Store

esp_ble_mesh_state_change_scene_recall_t **scene_recall**

Scene Recall

esp_ble_mesh_state_change_scene_delete_t **scene_delete**

Scene Delete

esp_ble_mesh_state_change_scheduler_act_set_t **scheduler_act_set**

Scheduler Action Set

union esp_ble_mesh_time_scene_server_recv_get_msg_t

#include <esp_ble_mesh_time_scene_model_api.h> Time Scene Server Model received get message union.

Public Members

esp_ble_mesh_server_recv_scheduler_act_get_t **scheduler_act**

Scheduler Action Get

union esp_ble_mesh_time_scene_server_recv_set_msg_t

#include <esp_ble_mesh_time_scene_model_api.h> Time Scene Server Model received set message union.

Public Members

esp_ble_mesh_server_recv_time_set_t **time**

Time Set

esp_ble_mesh_server_recv_time_zone_set_t **time_zone**

Time Zone Set

esp_ble_mesh_server_recv_tai_utc_delta_set_t **tai_utc_delta**

TAI-UTC Delta Set

esp_ble_mesh_server_recv_time_role_set_t **time_role**

Time Role Set

esp_ble_mesh_server_recv_scene_store_t **scene_store**

Scene Store/Scene Store Unack

esp_ble_mesh_server_recv_scene_recall_t **scene_recall**

Scene Recall/Scene Recall Unack

esp_ble_mesh_server_recv_scene_delete_t **scene_delete**

Scene Delete/Scene Delete Unack

esp_ble_mesh_server_recv_scheduler_act_set_t **scheduler_act**

Scheduler Action Set/Scheduler Action Set Unack

union esp_ble_mesh_time_scene_server_recv_status_msg_t

#include <esp_ble_mesh_time_scene_model_api.h> Time Scene Server Model received status message union.

Public Members

esp_ble_mesh_server_recv_time_status_t **time_status**

Time Status

union esp_ble_mesh_time_scene_server_cb_value_t

#include <esp_ble_mesh_time_scene_model_api.h> Time Scene Server Model callback value union.

Public Members

esp_ble_mesh_time_scene_server_state_change_t **state_change**

ESP_BLE_MESH_TIME_SCENE_SERVER_STATE_CHANGE_EVT

esp_ble_mesh_time_scene_server_recv_get_msg_t **get**

ESP_BLE_MESH_TIME_SCENE_SERVER_RECV_GET_MSG_EVT

esp_ble_mesh_time_scene_server_recv_set_msg_t **set**

ESP_BLE_MESH_TIME_SCENE_SERVER_RECV_SET_MSG_EVT

esp_ble_mesh_time_scene_server_recv_status_msg_t **status**

ESP_BLE_MESH_TIME_SCENE_SERVER_RECV_STATUS_MSG_EVT

Structures

struct esp_ble_mesh_time_set_t

Bluetooth Mesh Time Scene Client Model Get and Set parameters structure.

Parameters of Time Set

Public Members

uint8_t **tai_seconds**[5]

The current TAI time in seconds

uint8_t **sub_second**

The sub-second time in units of 1/256 second

uint8_t **uncertainty**

The estimated uncertainty in 10-millisecond steps

uint16_t **time_authority**

0 = No Time Authority, 1 = Time Authority

uint16_t **tai_utc_delta**

Current difference between TAI and UTC in seconds

uint8_t **time_zone_offset**

The local time zone offset in 15-minute increments

struct esp_ble_mesh_time_zone_set_t

Parameters of Time Zone Set

Public Members

uint8_t **time_zone_offset_new**

Upcoming local time zone offset

uint8_t **tai_zone_change**[5]

TAI Seconds time of the upcoming Time Zone Offset change

struct esp_ble_mesh_tai_utc_delta_set_t

Parameters of TAI-UTC Delta Set

Public Members

uint16_t **tai_utc_delta_new**

Upcoming difference between TAI and UTC in seconds

uint16_t **padding**

Always 0b0. Other values are Prohibited.

uint8_t **tai_delta_change**[5]

TAI Seconds time of the upcoming TAI-UTC Delta change

struct esp_ble_mesh_time_role_set_t

Parameter of Time Role Set

Public Members

uint8_t **time_role**

The Time Role for the element

struct esp_ble_mesh_scene_store_t

Parameter of Scene Store

Public Members

uint16_t **scene_number**

The number of scenes to be stored

struct esp_ble_mesh_scene_recall_t

Parameters of Scene Recall

Public Members

bool **op_en**

Indicate if optional parameters are included

uint16_t **scene_number**

The number of scenes to be recalled

uint8_t **tid**

Transaction ID

uint8_t **trans_time**

Time to complete state transition (optional)

uint8_t **delay**

Indicate message execution delay (C.1)

struct esp_ble_mesh_scene_delete_t

Parameter of Scene Delete

Public Members

uint16_t **scene_number**

The number of scenes to be deleted

struct esp_ble_mesh_scheduler_act_get_t

Parameter of Scheduler Action Get

Public Members

uint8_t **index**

Index of the Schedule Register entry to get

struct esp_ble_mesh_scheduler_act_set_t

Parameters of Scheduler Action Set

Public Members

uint64_t **index**

Index of the Schedule Register entry to set

uint64_t **year**

Scheduled year for the action

uint64_t **month**

Scheduled month for the action

uint64_t **day**

Scheduled day of the month for the action

uint64_t **hour**

Scheduled hour for the action

uint64_t **minute**

Scheduled minute for the action

uint64_t **second**

Scheduled second for the action

uint64_t **day_of_week**

Schedule days of the week for the action

uint64_t **action**

Action to be performed at the scheduled time

uint64_t **trans_time**

Transition time for this action

uint16_t **scene_number**

Transition time for this action

struct esp_ble_mesh_time_status_cb_t

Bluetooth Mesh Time Scene Client Model Get and Set callback parameters structure.

Parameters of Time Status

Public Members

uint8_t **tai_seconds**[5]

The current TAI time in seconds

uint8_t **sub_second**

The sub-second time in units of 1/256 second

uint8_t **uncertainty**

The estimated uncertainty in 10-millisecond steps

uint16_t **time_authority**

0 = No Time Authority, 1 = Time Authority

uint16_t **tai_utc_delta**

Current difference between TAI and UTC in seconds

uint8_t **time_zone_offset**

The local time zone offset in 15-minute increments

struct esp_ble_mesh_time_zone_status_cb_t

Parameters of Time Zone Status

Public Members

uint8_t **time_zone_offset_curr**

Current local time zone offset

uint8_t **time_zone_offset_new**

Upcoming local time zone offset

uint8_t **tai_zone_change**[5]

TAI Seconds time of the upcoming Time Zone Offset change

struct esp_ble_mesh_tai_utc_delta_status_cb_t

Parameters of TAI-UTC Delta Status

Public Members

uint16_t **tai_utc_delta_curr**

Current difference between TAI and UTC in seconds

uint16_t **padding_1**

Always 0b0. Other values are Prohibited.

uint16_t **tai_utc_delta_new**

Upcoming difference between TAI and UTC in seconds

uint16_t **padding_2**

Always 0b0. Other values are Prohibited.

uint8_t **tai_delta_change**[5]

TAI Seconds time of the upcoming TAI-UTC Delta change

struct esp_ble_mesh_time_role_status_cb_t

Parameter of Time Role Status

Public Members

uint8_t **time_role**

The Time Role for the element

struct esp_ble_mesh_scene_status_cb_t

Parameters of Scene Status

Public Members

bool **op_en**

Indicate if optional parameters are included

uint8_t **status_code**

Status code of the last operation

uint16_t **current_scene**

Scene Number of the current scene

uint16_t **target_scene**

Scene Number of the target scene (optional)

uint8_t **remain_time**

Time to complete state transition (C.1)

struct esp_ble_mesh_scene_register_status_cb_t

Parameters of Scene Register Status

Public Members

uint8_t **status_code**

Status code for the previous operation

uint16_t **current_scene**

Scene Number of the current scene

struct net_buf_simple ***scenes**

A list of scenes stored within an element

struct esp_ble_mesh_scheduler_status_cb_t

Parameter of Scheduler Status

Public Members

uint16_t **schedules**

Bit field indicating defined Actions in the Schedule Register

struct esp_ble_mesh_scheduler_act_status_cb_t

Parameters of Scheduler Action Status

Public Members

uint64_t **index**

Enumerates (selects) a Schedule Register entry

uint64_t **year**

Scheduled year for the action

uint64_t **month**

Scheduled month for the action

uint64_t **day**

Scheduled day of the month for the action

uint64_t **hour**

Scheduled hour for the action

uint64_t **minute**

Scheduled minute for the action

uint64_t **second**

Scheduled second for the action

uint64_t **day_of_week**

Schedule days of the week for the action

uint64_t **action**

Action to be performed at the scheduled time

uint64_t **trans_time**

Transition time for this action

uint16_t **scene_number**

Transition time for this action

struct esp_ble_mesh_time_scene_client_cb_param_t

Time Scene Client Model callback parameters

Public Members

int **error_code**

Appropriate error code

esp_ble_mesh_client_common_param_t ***params**

The client common parameters.

esp_ble_mesh_time_scene_client_status_cb_t **status_cb**

The scene status message callback values

struct esp_ble_mesh_time_state_t

Parameters of Time state

Public Members

uint8_t **tai_seconds**[5]

The value of the TAI Seconds state

uint8_t **subsecond**

The value of the Subsecond field

uint8_t **uncertainty**

The value of the Uncertainty field

uint8_t **time_zone_offset_curr**

The value of the Time Zone Offset Current field

uint8_t **time_zone_offset_new**

The value of the Time Zone Offset New state

uint8_t **tai_zone_change**[5]

The value of the TAI of Zone Chaneg field

uint16_t **time_authority**

The value of the Time Authority bit

uint16_t **tai_utc_delta_curr**

The value of the TAI-UTC Delta Current state

uint16_t **tai_utc_delta_new**

The value of the TAI-UTC Delta New state

uint8_t **tai_delta_change**[5]

The value of the TAI of Delta Change field

struct *esp_ble_mesh_time_state_t*::[**anonymous**] **time**

Parameters of the Time state

uint8_t **time_role**

The value of the Time Role state

struct esp_ble_mesh_time_srv_t

User data of Time Server Model

Public Members

esp_ble_mesh_model_t ***model**

Pointer to the Time Server Model. Initialized internally.

esp_ble_mesh_server_rsp_ctrl_t **rsp_ctrl**

Response control of the server model received messages

esp_ble_mesh_time_state_t ***state**

Parameters of the Time state

struct esp_ble_mesh_time_setup_srv_t

User data of Time Setup Server Model

Public Members

esp_ble_mesh_model_t ***model**

Pointer to the Time Setup Server Model. Initialized internally.

esp_ble_mesh_server_rsp_ctrl_t **rsp_ctrl**

Response control of the server model received messages

esp_ble_mesh_time_state_t ***state**

Parameters of the Time state

struct esp_ble_mesh_scene_register_t

1. Scene Store is an operation of storing values of a present state of an element.
2. The structure and meaning of the stored state is determined by a model. States to be stored are specified by each model.
3. The Scene Store operation shall persistently store all values of all states marked as Stored with Scene for all models present on all elements of a node.
4. If a model is extending another model, the extending model shall determine the Stored with Scene behavior of that model.Parameters of Scene Register state

Public Members

uint16_t **scene_number**

The value of the Scene Number

uint8_t **scene_type**

The value of the Scene Type

struct net_buf_simple ***scene_value**

Scene value may use a union to represent later, the union contains structures of all the model states which can be stored in a scene.The value of the Scene Value

struct esp_ble_mesh_scenes_state_t

Parameters of Scenes state.

Scenes serve as memory banks for storage of states (e.g., a power level or a light level/color). Values of states of an element can be stored as a scene and can be recalled later from the scene memory.

A scene is represented by a Scene Number, which is a 16-bit non-zero, mesh-wide value. (There can be a maximum of 65535 scenes in a mesh network.) The meaning of a scene, as well as the state storage container associated with it, are determined by a model.

The Scenes state change may start numerous parallel model transitions. In that case, each individual model handles the transition internally.

The scene transition is defined as a group of individual model transitions started by a Scene Recall operation. The scene transition is in progress when at least one transition from the group of individual model transitions is in progress.

Public Members

const uint16_t **scene_count**

The Scenes state’ s scene count

esp_ble_mesh_scene_register_t ***scenes**

Parameters of the Scenes state

uint16_t **current_scene**

The Current Scene state is a 16-bit value that contains either the Scene Number of the currently active scene or a value of 0x0000 when no scene is active.

When a Scene Store operation or a Scene Recall operation completes with success, the Current Scene state value shall be to the Scene Number used during that operation.

When the Current Scene Number is deleted from a Scene Register state as a result of Scene Delete operation, the Current Scene state shall be set to 0x0000.

When any of the element' s state that is marked as "Stored with Scene" has changed not as a result of a Scene Recall operation, the value of the Current Scene state shall be set to 0x0000.

When a scene transition is in progress, the value of the Current Scene state shall be set to 0x0000.The value of the Current Scene state

uint16_t **target_scene**

The Target Scene state is a 16-bit value that contains the target Scene Number when a scene transition is in progress.

When the scene transition is in progress and the target Scene Number is deleted from a Scene Register state as a result of Scene Delete operation, the Target Scene state shall be set to 0x0000.

When the scene transition is in progress and a new Scene Number is stored in the Scene Register as a result of Scene Store operation, the Target Scene state shall be set to the new Scene Number.

When the scene transition is not in progress, the value of the Target Scene state shall be set to 0x0000.The value of the Target Scene state

uint8_t **status_code**

The status code of the last scene operation

bool **in_progress**

Indicate if the scene transition is in progress

struct esp_ble_mesh_scene_srv_t

User data of Scene Server Model

Public Members

esp_ble_mesh_model_t ***model**

Pointer to the Scene Server Model. Initialized internally.

esp_ble_mesh_server_rsp_ctrl_t **rsp_ctrl**

Response control of the server model received messages

esp_ble_mesh_scenes_state_t ***state**

Parameters of the Scenes state

esp_ble_mesh_last_msg_info_t **last**

Parameters of the last received set message

esp_ble_mesh_state_transition_t **transition**

Parameters of state transition

struct esp_ble_mesh_scene_setup_srv_t

User data of Scene Setup Server Model

Public Members

esp_ble_mesh_model_t ***model**

Pointer to the Scene Setup Server Model. Initialized internally.

esp_ble_mesh_server_rsp_ctrl_t **rsp_ctrl**

Response control of the server model received messages

esp_ble_mesh_scenes_state_t ***state**

Parameters of the Scenes state

struct esp_ble_mesh_schedule_register_t

Parameters of Scheduler Register state

Public Members

bool **in_use**

Indicate if the registered schedule is in use

uint64_t **year**

The value of Scheduled year for the action

uint64_t **month**

The value of Scheduled month for the action

uint64_t **day**

The value of Scheduled day of the month for the action

uint64_t **hour**

The value of Scheduled hour for the action

uint64_t **minute**

The value of Scheduled minute for the action

uint64_t **second**

The value of Scheduled second for the action

uint64_t **day_of_week**

The value of Schedule days of the week for the action

uint64_t **action**

The value of Action to be performed at the scheduled time

uint64_t **trans_time**

The value of Transition time for this action

uint16_t **scene_number**

The value of Scene Number to be used for some actions

struct esp_ble_mesh_scheduler_state_t

Parameters of Scheduler state

Public Members

const uint8_t **schedule_count**

Scheduler count

esp_ble_mesh_schedule_register_t ***schedules**

Up to 16 scheduled entries

struct esp_ble_mesh_scheduler_srv_t

User data of Scheduler Server Model

Public Members

esp_ble_mesh_model_t ***model**

Pointer to the Scheduler Server Model. Initialized internally.

esp_ble_mesh_server_rsp_ctrl_t **rsp_ctrl**

Response control of the server model received messages

esp_ble_mesh_scheduler_state_t ***state**

Parameters of the Scheduler state

struct esp_ble_mesh_scheduler_setup_srv_t

User data of Scheduler Setup Server Model

Public Members

esp_ble_mesh_model_t ***model**

Pointer to the Scheduler Setup Server Model. Initialized internally.

esp_ble_mesh_server_rsp_ctrl_t **rsp_ctrl**

Response control of the server model received messages

esp_ble_mesh_scheduler_state_t ***state**

Parameters of the Scheduler state

struct esp_ble_mesh_state_change_time_set_t

Parameters of Time Set state change event

Public Members

uint8_t **tai_seconds**[5]

The current TAI time in seconds

uint8_t **subsecond**

The sub-second time in units of 1/256 second

uint8_t **uncertainty**

The estimated uncertainty in 10-millisecond steps

uint16_t **time_authority**

0 = No Time Authority, 1 = Time Authority

uint16_t **tai_utc_delta_curr**

Current difference between TAI and UTC in seconds

uint8_t **time_zone_offset_curr**

The local time zone offset in 15-minute increments

struct esp_ble_mesh_state_change_time_status_t

Parameters of Time Status state change event

Public Members

uint8_t **tai_seconds**[5]

The current TAI time in seconds

uint8_t **subsecond**

The sub-second time in units of 1/256 second

uint8_t **uncertainty**

The estimated uncertainty in 10-millisecond steps

uint16_t **time_authority**

0 = No Time Authority, 1 = Time Authority

uint16_t **tai_utc_delta_curr**

Current difference between TAI and UTC in seconds

uint8_t **time_zone_offset_curr**

The local time zone offset in 15-minute increments

struct esp_ble_mesh_state_change_time_zone_set_t

Parameters of Time Zone Set state change event

Public Members

uint8_t **time_zone_offset_new**

Upcoming local time zone offset

uint8_t **tai_zone_change**[5]

TAI Seconds time of the upcoming Time Zone Offset change

struct esp_ble_mesh_state_change_tai_utc_delta_set_t

Parameters of TAI UTC Delta Set state change event

Public Members

uint16_t **tai_utc_delta_new**

Upcoming difference between TAI and UTC in seconds

uint8_t **tai_delta_change**[5]

TAI Seconds time of the upcoming TAI-UTC Delta change

struct esp_ble_mesh_state_change_time_role_set_t

Parameter of Time Role Set state change event

Public Members

uint8_t **time_role**

The Time Role for the element

struct esp_ble_mesh_state_change_scene_store_t

Parameter of Scene Store state change event

Public Members

uint16_t **scene_number**

The number of scenes to be stored

struct esp_ble_mesh_state_change_scene_recall_t

Parameter of Scene Recall state change event

Public Members

uint16_t **scene_number**

The number of scenes to be recalled

struct esp_ble_mesh_state_change_scene_delete_t

Parameter of Scene Delete state change event

Public Members

uint16_t **scene_number**

The number of scenes to be deleted

struct esp_ble_mesh_state_change_scheduler_act_set_t

Parameter of Scheduler Action Set state change event

Public Members

uint64_t **index**

Index of the Schedule Register entry to set

uint64_t **year**

Scheduled year for the action

uint64_t **month**

Scheduled month for the action

uint64_t **day**

Scheduled day of the month for the action

uint64_t **hour**

Scheduled hour for the action

uint64_t **minute**

Scheduled minute for the action

uint64_t **second**

Scheduled second for the action

uint64_t **day_of_week**

Schedule days of the week for the action

uint64_t **action**

Action to be performed at the scheduled time

uint64_t **trans_time**

Transition time for this action

uint16_t **scene_number**

Scene number to be used for some actions

struct esp_ble_mesh_server_recv_scheduler_act_get_t

Context of the received Scheduler Action Get message

Public Members

uint8_t **index**

Index of the Schedule Register entry to get

struct esp_ble_mesh_server_recv_time_set_t

Context of the received Time Set message

Public Members

uint8_t **tai_seconds**[5]

The current TAI time in seconds

uint8_t **subsecond**

The sub-second time in units of 1/256 second

uint8_t **uncertainty**

The estimated uncertainty in 10-millisecond steps

uint16_t **time_authority**

0 = No Time Authority, 1 = Time Authority

uint16_t **tai_utc_delta**

Current difference between TAI and UTC in seconds

uint8_t **time_zone_offset**

The local time zone offset in 15-minute increments

struct esp_ble_mesh_server_recv_time_zone_set_t

Context of the received Time Zone Set message

Public Members

uint8_t **time_zone_offset_new**

Upcoming local time zone offset

uint8_t **tai_zone_change**[5]

TAI Seconds time of the upcoming Time Zone Offset change

struct esp_ble_mesh_server_recv_tai_utc_delta_set_t

Context of the received TAI UTC Delta Set message

Public Members

uint16_t **tai_utc_delta_new**

Upcoming difference between TAI and UTC in seconds

uint16_t **padding**

Always 0b0. Other values are Prohibited.

uint8_t **tai_delta_change**[5]

TAI Seconds time of the upcoming TAI-UTC Delta change

struct esp_ble_mesh_server_recv_time_role_set_t

Context of the received Time Role Set message

Public Members

uint8_t **time_role**

The Time Role for the element

struct esp_ble_mesh_server_recv_scene_store_t

Context of the received Scene Store message

Public Members

uint16_t **scene_number**

The number of scenes to be stored

struct esp_ble_mesh_server_recv_scene_recall_t

Context of the received Scene Recall message

Public Members

bool **op_en**

Indicate if optional parameters are included

uint16_t **scene_number**

The number of scenes to be recalled

uint8_t **tid**

Transaction ID

uint8_t **trans_time**

Time to complete state transition (optional)

uint8_t **delay**

Indicate message execution delay (C.1)

struct esp_ble_mesh_server_recv_scene_delete_t

Context of the received Scene Delete message

Public Members

uint16_t **scene_number**

The number of scenes to be deleted

struct esp_ble_mesh_server_recv_scheduler_act_set_t

Context of the received Scheduler Action Set message

Public Members

uint64_t **index**

Index of the Schedule Register entry to set

uint64_t **year**

Scheduled year for the action

uint64_t **month**

Scheduled month for the action

uint64_t **day**

Scheduled day of the month for the action

uint64_t **hour**

Scheduled hour for the action

uint64_t **minute**

Scheduled minute for the action

uint64_t **second**

Scheduled second for the action

uint64_t **day_of_week**

Schedule days of the week for the action

uint64_t **action**

Action to be performed at the scheduled time

uint64_t **trans_time**

Transition time for this action

uint16_t **scene_number**

Scene number to be used for some actions

struct esp_ble_mesh_server_recv_time_status_t

Context of the received Time Status message

Public Members

uint8_t **tai_seconds**[5]
The current TAI time in seconds

uint8_t **subsecond**
The sub-second time in units of 1/256 second

uint8_t **uncertainty**
The estimated uncertainty in 10-millisecond steps

uint16_t **time_authority**
0 = No Time Authority, 1 = Time Authority

uint16_t **tai_utc_delta**
Current difference between TAI and UTC in seconds

uint8_t **time_zone_offset**
The local time zone offset in 15-minute increments

struct esp_ble_mesh_time_scene_server_cb_param_t
Time Scene Server Model callback parameters

Public Members

esp_ble_mesh_model_t ***model**
Pointer to Time and Scenes Server Models

esp_ble_mesh_msg_ctx_t **ctx**
Context of the received messages

esp_ble_mesh_time_scene_server_cb_value_t **value**
Value of the received Time and Scenes Messages

Macros

ESP_BLE_MESH_MODEL_TIME_CLI(cli_pub, cli_data)
Define a new Time Client Model.

Note This API needs to be called for each element on which the application needs to have a Time Client Model.

Return New Time Client Model instance.

Parameters

- `cli_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- `cli_data`: Pointer to the unique struct *esp_ble_mesh_client_t*.

ESP_BLE_MESH_MODEL_SCENE_CLI(cli_pub, cli_data)

Define a new Scene Client Model.

Note This API needs to be called for each element on which the application needs to have a Scene Client Model.

Return New Scene Client Model instance.

Parameters

- `cli_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- `cli_data`: Pointer to the unique struct *esp_ble_mesh_client_t*.

ESP_BLE_MESH_MODEL_SCHEDULER_CLI(cli_pub, cli_data)

Define a new Scheduler Client Model.

Note This API needs to be called for each element on which the application needs to have a Scheduler Client Model.

Return New Scheduler Client Model instance.

Parameters

- `cli_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- `cli_data`: Pointer to the unique struct *esp_ble_mesh_client_t*.

ESP_BLE_MESH_MODEL_TIME_SRV(srv_pub, srv_data)

Time Scene Server Models related context.

Define a new Time Server Model.

Note 1. The Time Server model is a root model. When this model is present on an Element, the corresponding Time Setup Server model shall also be present.

1. This model shall support model publication and model subscription.

Return New Time Server Model instance.

Parameters

- `srv_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- `srv_data`: Pointer to the unique struct *esp_ble_mesh_time_srv_t*.

ESP_BLE_MESH_MODEL_TIME_SETUP_SRV(srv_data)

Define a new Time Setup Server Model.

Note 1. The Time Setup Server model extends the Time Server model. Time is sensitive information that is propagated across a mesh network.

1. Only an authorized Time Client should be allowed to change the Time and Time Role states. A dedicated application key Bluetooth SIG Proprietary should be used on the Time Setup Server to restrict access to the server to only authorized Time Clients.
2. This model does not support subscribing nor publishing.

Return New Time Setup Server Model instance.

Parameters

- `srv_data`: Pointer to the unique struct *esp_ble_mesh_time_setup_srv_t*.

`ESP_BLE_MESH_MODEL_SCENE_SRV`(srv_pub, srv_data)

Define a new Scene Server Model.

Note 1. The Scene Server model is a root model. When this model is present on an Element, the corresponding Scene Setup Server model shall also be present.

1. This model shall support model publication and model subscription.
2. The model may be present only on the Primary element of a node.

Return New Scene Server Model instance.

Parameters

- `srv_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- `srv_data`: Pointer to the unique struct *esp_ble_mesh_scene_srv_t*.

`ESP_BLE_MESH_MODEL_SCENE_SETUP_SRV`(srv_pub, srv_data)

Define a new Scene Setup Server Model.

Note 1. The Scene Setup Server model extends the Scene Server model and the Generic Default Transition Time Server model.

1. This model shall support model subscription.
2. The model may be present only on the Primary element of a node.

Return New Scene Setup Server Model instance.

Parameters

- `srv_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- `srv_data`: Pointer to the unique struct *esp_ble_mesh_scene_setup_srv_t*.

`ESP_BLE_MESH_MODEL_SCHEDULER_SRV`(srv_pub, srv_data)

Define a new Scheduler Server Model.

Note 1. The Scheduler Server model extends the Scene Server model. When this model is present on an Element, the corresponding Scheduler Setup Server model shall also be present.

1. This model shall support model publication and model subscription.
2. The model may be present only on the Primary element of a node.
3. The model requires the Time Server model shall be present on the element.

Return New Scheduler Server Model instance.

Parameters

- `srv_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- `srv_data`: Pointer to the unique struct *esp_ble_mesh_scheduler_srv_t*.

`ESP_BLE_MESH_MODEL_SCHEDULER_SETUP_SRV`(srv_pub, srv_data)

Define a new Scheduler Setup Server Model.

Note 1. The Scheduler Setup Server model extends the Scheduler Server and the Scene Setup Server models.

1. This model shall support model subscription.
2. The model may be present only on the Primary element of a node.

Return New Scheduler Setup Server Model instance.

Parameters

- `srv_pub`: Pointer to the unique struct *esp_ble_mesh_model_pub_t*.
- `srv_data`: Pointer to the unique struct *esp_ble_mesh_scheduler_setup_srv_t*.

`ESP_BLE_MESH_UNKNOWN_TAI_SECONDS`

Unknown TAI Seconds

`ESP_BLE_MESH_UNKNOWN_TAI_ZONE_CHANGE`

Unknown TAI of Zone Change

`ESP_BLE_MESH_UNKNOWN_TAI_DELTA_CHANGE`

Unknown TAI of Delta Change

`ESP_BLE_MESH_TAI_UTC_DELTA_MAX_VALUE`

Maximum TAI-UTC Delta value

`ESP_BLE_MESH_TAI_SECONDS_LEN`

Length of TAI Seconds

`ESP_BLE_MESH_TAI_OF_ZONE_CHANGE_LEN`

Length of TAI of Zone Change

`ESP_BLE_MESH_TAI_OF_DELTA_CHANGE_LEN`

Length of TAI of Delta Change

ESP_BLE_MESH_INVALID_SCENE_NUMBER
: Invalid Scene Number

ESP_BLE_MESH_SCENE_NUMBER_LEN
: Length of the Scene Number

ESP_BLE_MESH_SCHEDULE_YEAR_ANY_YEAR
: Any year of the Scheduled year

ESP_BLE_MESH_SCHEDULE_DAY_ANY_DAY
: Any day of the Scheduled day

ESP_BLE_MESH_SCHEDULE_HOUR_ANY_HOUR
: Any hour of the Scheduled hour

ESP_BLE_MESH_SCHEDULE_HOUR_ONCE_A_DAY
: Any hour of the Scheduled Day

ESP_BLE_MESH_SCHEDULE_SEC_ANY_OF_HOUR
: Any minute of the Scheduled hour

ESP_BLE_MESH_SCHEDULE_SEC_EVERY_15_MIN
: Every 15 minutes of the Scheduled hour

ESP_BLE_MESH_SCHEDULE_SEC_EVERY_20_MIN
: Every 20 minutes of the Scheduled hour

ESP_BLE_MESH_SCHEDULE_SEC_ONCE_AN_HOUR
: Once of the Scheduled hour

ESP_BLE_MESH_SCHEDULE_SEC_ANY_OF_MIN
: Any second of the Scheduled minute

ESP_BLE_MESH_SCHEDULE_SEC_EVERY_15_SEC
: Every 15 seconds of the Scheduled minute

ESP_BLE_MESH_SCHEDULE_SEC_EVERY_20_SEC
: Every 20 seconds of the Scheduled minute

ESP_BLE_MESH_SCHEDULE_SEC_ONCE_AN_MIN
: Once of the Scheduled minute

ESP_BLE_MESH_SCHEDULE_ACT_TURN_OFF
: Scheduled Action - Turn Off

ESP_BLE_MESH_SCHEDULE_ACT_TURN_ON
: Scheduled Action - Turn On

ESP_BLE_MESH_SCHEDULE_ACT_SCENE_RECALL
: Scheduled Action - Scene Recall

ESP_BLE_MESH_SCHEDULE_ACT_NO_ACTION
: Scheduled Action - No Action

ESP_BLE_MESH_SCHEDULE_SCENE_NO_SCENE
: Scheduled Scene - No Scene

ESP_BLE_MESH_SCHEDULE_ENTRY_MAX_INDEX
: Maximum number of Scheduled entries

ESP_BLE_MESH_TIME_NONE
: Time Role - None

ESP_BLE_MESH_TIME_AUTHORITY
: Time Role - Mesh Time Authority

ESP_BLE_MESH_TIME_RELAY
: Time Role - Mesh Time Relay

ESP_BLE_MESH_TIME_CLINET
: Time Role - Mesh Time Client

ESP_BLE_MESH_SCENE_SUCCESS
: Scene operation - Success

ESP_BLE_MESH_SCENE_REG_FULL
: Scene operation - Scene Register Full

ESP_BLE_MESH_SCENE_NOT_FOUND
: Scene operation - Scene Not Found

Type Definitions

typedef void (*esp_ble_mesh_time_scene_client_cb_t)(*esp_ble_mesh_time_scene_client_cb_event_t* event, *esp_ble_mesh_time_scene_client_cb_param_t* *param)

Bluetooth Mesh Time Scene Client Model function.

Time Scene Client Model callback function type

Parameters

- event: Event type
- param: Pointer to callback parameter

typedef void (*esp_ble_mesh_time_scene_server_cb_t)(*esp_ble_mesh_time_scene_server_cb_event_t* event, *esp_ble_mesh_time_scene_server_cb_param_t* *param)

Bluetooth Mesh Time and Scenes Server Model function.

Time Scene Server Model callback function type

Parameters

- `event`: Event type
- `param`: Pointer to callback parameter

Enumerations

enum esp_ble_mesh_time_scene_client_cb_event_t

This enum value is the event of Time Scene Client Model

Values:

ESP_BLE_MESH_TIME_SCENE_CLIENT_GET_STATE_EVT

ESP_BLE_MESH_TIME_SCENE_CLIENT_SET_STATE_EVT

ESP_BLE_MESH_TIME_SCENE_CLIENT_PUBLISH_EVT

ESP_BLE_MESH_TIME_SCENE_CLIENT_TIMEOUT_EVT

ESP_BLE_MESH_TIME_SCENE_CLIENT_EVT_MAX

enum esp_ble_mesh_time_scene_server_cb_event_t

This enum value is the event of Time Scene Server Model

Values:

ESP_BLE_MESH_TIME_SCENE_SERVER_STATE_CHANGE_EVT

1. When get_auto_rsp is set to ESP_BLE_MESH_SERVER_AUTO_RSP, no event will be callback to the application layer when Time Scene Get messages are received.
2. When set_auto_rsp is set to ESP_BLE_MESH_SERVER_AUTO_RSP, this event will be callback to the application layer when Time Scene Set/Set Unack messages are received.

ESP_BLE_MESH_TIME_SCENE_SERVER_RECV_GET_MSG_EVT

When get_auto_rsp is set to ESP_BLE_MESH_SERVER_RSP_BY_APP, this event will be callback to the application layer when Time Scene Get messages are received.

ESP_BLE_MESH_TIME_SCENE_SERVER_RECV_SET_MSG_EVT

When set_auto_rsp is set to ESP_BLE_MESH_SERVER_RSP_BY_APP, this event will be callback to the application layer when Time Scene Set/Set Unack messages are received.

ESP_BLE_MESH_TIME_SCENE_SERVER_RECV_STATUS_MSG_EVT

When status_auto_rsp is set to ESP_BLE_MESH_SERVER_RSP_BY_APP, this event will be callback to the application layer when TIme Status message is received.

ESP_BLE_MESH_TIME_SCENE_SERVER_EVT_MAX

请点击下方链接，查看 ESP32 蓝牙架构：

ESP32 蓝牙架构 (PDF)

蓝牙 API 的示例代码存放于 ESP-IDF bluetooth/bluedroid 示例目录下，请查看。

下面的示例给出了详细介绍：

- GATT 客户端示例
- GATT 服务端服务表格示例
- GATT 服务端示例
- GATT 客户端安全性示例
- GATT 服务端安全性示例
- GATT 客户端多连接示例

2.2 连网 API

[English]

2.2.1 Wi-Fi

Wi-Fi 库

[English]

概述

Wi-Fi 库支持配置及监控 ESP32 Wi-Fi 连网功能。

支持配置：

- 基站模式（即 STA 模式或 Wi-Fi 客户端模式），此时 ESP32 连接到接入点 (AP)。
- AP 模式（即 Soft-AP 模式或接入点模式），此时基站连接到 ESP32。
- AP-STA 共存模式（ESP32 既是接入点，同时又作为基站连接到另外一个接入点）。
- 上述模式的各种安全模式（WPA、WPA2 及 WEP 等）。
- 扫描接入点（包括主动扫描及被动扫描）。
- 使用混杂模式监控 IEEE802.11 Wi-Fi 数据包。

应用示例

ESP-IDF 示例项目的 wifi 目录下包含以下应用程序：

- Wi-Fi 示例代码；

- 另外一个简单的应用程序 esp-idf-template，演示了如何将 ESP32 模组连接到 AP。

API 参考

Header File

- esp_wifi/include/esp_wifi.h

Functions

esp_err_t **esp_wifi_init**(**const** *wifi_init_config_t* **config*)

Init WiFi Alloc resource for WiFi driver, such as WiFi control structure, RX/TX buffer, WiFi NVS structure etc, this WiFi also start WiFi task.

Attention 1. This API must be called before all other WiFi API can be called

Attention 2. Always use WIFI_INIT_CONFIG_DEFAULT macro to init the config to default values, this can guarantee all the fields got correct value when more fields are added into *wifi_init_config_t* in future release. If you want to set your owner initial values, overwrite the default values which are set by WIFI_INIT_CONFIG_DEFAULT, please be notified that the field 'magic' of *wifi_init_config_t* should always be WIFI_INIT_CONFIG_MAGIC!

Return

- ESP_OK: succeed
- ESP_ERR_NO_MEM: out of memory
- others: refer to error code esp_err.h

Parameters

- **config**: pointer to WiFi init configuration structure; can point to a temporary variable.

esp_err_t **esp_wifi_deinit**(void)

Deinit WiFi Free all resource allocated in esp_wifi_init and stop WiFi task.

Attention 1. This API should be called if you want to remove WiFi driver from the system

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init

esp_err_t **esp_wifi_set_mode**(*wifi_mode_t mode*)

Set the WiFi operating mode.

Set the WiFi operating mode as station, soft-AP or station+soft-AP, The default mode is soft-AP mode.

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_INVALID_ARG: invalid argument
- others: refer to error code in esp_err.h

Parameters

- `mode`: WiFi operating mode

esp_err_t **esp_wifi_get_mode**(*wifi_mode_t* **mode*)

Get current operating mode of WiFi.

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_INVALID_ARG: invalid argument

Parameters

- `mode`: store current WiFi mode

esp_err_t **esp_wifi_start**(void)

Start WiFi according to current configuration If mode is WIFI_MODE_STA, it create station control block and start station If mode is WIFI_MODE_AP, it create soft-AP control block and start soft-AP If mode is WIFI_MODE_APSTA, it create soft-AP and station control block and start soft-AP and station.

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_INVALID_ARG: invalid argument
- ESP_ERR_NO_MEM: out of memory
- ESP_ERR_WIFI_CONN: WiFi internal error, station or soft-AP control block wrong
- ESP_FAIL: other WiFi internal errors

esp_err_t **esp_wifi_stop**(void)

Stop WiFi If mode is WIFI_MODE_STA, it stop station and free station control block If mode is WIFI_MODE_AP, it stop soft-AP and free soft-AP control block If mode is WIFI_MODE_APSTA, it stop station/soft-AP and free station/soft-AP control block.

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init

esp_err_t **esp_wifi_restore**(void)

Restore WiFi stack persistent settings to default values.

This function will reset settings made using the following APIs:

- esp_wifi_get_auto_connect,
- esp_wifi_set_protocol,
- esp_wifi_set_config related
- esp_wifi_set_mode

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init

esp_err_t **esp_wifi_connect**(void)

Connect the ESP32 WiFi station to the AP.

Attention 1. This API only impact WIFI_MODE_STA or WIFI_MODE_APSTA mode

Attention 2. If the ESP32 is connected to an AP, call esp_wifi_disconnect to disconnect.

Attention 3. The scanning triggered by esp_wifi_start_scan() will not be effective until connection between ESP32 and the AP is established. If ESP32 is scanning and connecting at the same time, ESP32 will abort scanning and return a warning message and error number ESP_ERR_WIFI_STATE. If you want to do reconnection after ESP32 received disconnect event, remember to add the maximum retry time, otherwise the called scan will not work. This is especially true when the AP doesn' t exist, and you still try reconnection after ESP32 received disconnect event with the reason code WIFI_REASON_NO_AP_FOUND.

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_NOT_STARTED: WiFi is not started by esp_wifi_start

- ESP_ERR_WIFI_CONN: WiFi internal error, station or soft-AP control block wrong
- ESP_ERR_WIFI_SSID: SSID of AP which station connects is invalid

esp_err_t **esp_wifi_disconnect**(void)

Disconnect the ESP32 WiFi station from the AP.

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi was not initialized by esp_wifi_init
- ESP_ERR_WIFI_NOT_STARTED: WiFi was not started by esp_wifi_start
- ESP_FAIL: other WiFi internal errors

esp_err_t **esp_wifi_clear_fast_connect**(void)

Currently this API is just an stub API.

Return

- ESP_OK: succeed
- others: fail

esp_err_t **esp_wifi_deauth_sta**(uint16_t *aid*)

deauthenticate all stations or associated id equals to aid

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_NOT_STARTED: WiFi was not started by esp_wifi_start
- ESP_ERR_INVALID_ARG: invalid argument
- ESP_ERR_WIFI_MODE: WiFi mode is wrong

Parameters

- **aid**: when aid is 0, deauthenticate all stations, otherwise deauthenticate station whose associated id is aid

esp_err_t **esp_wifi_scan_start**(**const** *wifi_scan_config_t* **config*, bool *block*)

Scan all available APs.

Attention If this API is called, the found APs are stored in WiFi driver dynamic allocated memory and the will be freed in esp_wifi_scan_get_ap_records, so generally, call esp_wifi_scan_get_ap_records to cause the memory to be freed once the scan is done

Attention The values of maximum active scan time and passive scan time per channel are limited to 1500 milliseconds. Values above 1500ms may cause station to disconnect from AP and are not recommended.

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_NOT_STARTED: WiFi was not started by esp_wifi_start
- ESP_ERR_WIFI_TIMEOUT: blocking scan is timeout
- ESP_ERR_WIFI_STATE: wifi still connecting when invoke esp_wifi_scan_start
- others: refer to error code in esp_err.h

Parameters

- `config`: configuration of scanning
- `block`: if block is true, this API will block the caller until the scan is done, otherwise it will return immediately

esp_err_t **esp_wifi_scan_stop**(void)

Stop the scan in process.

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_NOT_STARTED: WiFi is not started by esp_wifi_start

esp_err_t **esp_wifi_scan_get_ap_num**(uint16_t **number*)

Get number of APs found in last scan.

Attention This API can only be called when the scan is completed, otherwise it may get wrong value.

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_NOT_STARTED: WiFi is not started by esp_wifi_start
- ESP_ERR_INVALID_ARG: invalid argument

Parameters

- `number`: store number of APIs found in last scan

esp_err_t **esp_wifi_scan_get_ap_records**(uint16_t **number*, *wifi_ap_record_t* **ap_records*)

Get AP list found in last scan.

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_NOT_STARTED: WiFi is not started by esp_wifi_start
- ESP_ERR_INVALID_ARG: invalid argument
- ESP_ERR_NO_MEM: out of memory

Parameters

- `number`: As input param, it stores max AP number ap_records can hold. As output param, it receives the actual AP number this API returns.
- `ap_records`: *wifi_ap_record_t* array to hold the found APs

esp_err_t **esp_wifi_sta_get_ap_info**(*wifi_ap_record_t* **ap_info*)

Get information of AP which the ESP32 station is associated with.

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_CONN: The station interface don't initialized
- ESP_ERR_WIFI_NOT_CONNECT: The station is in disconnect status

Parameters

- `ap_info`: the *wifi_ap_record_t* to hold AP information sta can get the connected ap's phy mode info through the struct member phy_11b, phy_11g, phy_11n, phy_lr in the *wifi_ap_record_t* struct. For example, phy_11b = 1 imply that ap support 802.11b mode

esp_err_t **esp_wifi_set_ps**(*wifi_ps_type_t type*)

Set current WiFi power save type.

Attention Default power save type is WIFI_PS_MIN_MODEM.

Return ESP_OK: succeed

Parameters

- `type`: power save type

esp_err_t **esp_wifi_get_ps**(*wifi_ps_type_t* **type*)

Get current WiFi power save type.

Attention Default power save type is WIFI_PS_MIN_MODEM.

Return ESP_OK: succeed

Parameters

- `type`: store current power save type

esp_err_t **esp_wifi_set_protocol**(*wifi_interface_t ifx*, uint8_t *protocol_bitmap*)

Set protocol type of specified interface The default protocol is (WIFI_PROTOCOL_11B|WIFI_PROTOCOL_11G|WIFI_PROTOCOL_11N)

Attention Currently we only support 802.11b or 802.11bg or 802.11bgn mode

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_IF: invalid interface
- others: refer to error codes in esp_err.h

Parameters

- `ifx`: interfaces
- `protocol_bitmap`: WiFi protocol bitmap

esp_err_t **esp_wifi_get_protocol**(*wifi_interface_t ifx*, uint8_t **protocol_bitmap*)

Get the current protocol bitmap of the specified interface.

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_IF: invalid interface
- ESP_ERR_INVALID_ARG: invalid argument
- others: refer to error codes in esp_err.h

Parameters

- `ifx`: interface
- `protocol_bitmap`: store current WiFi protocol bitmap of interface ifx

esp_err_t **esp_wifi_set_bandwidth**(*wifi_interface_t ifx*, *wifi_bandwidth_t bw*)

Set the bandwidth of ESP32 specified interface.

Attention 1. API return false if try to configure an interface that is not enabled

Attention 2. WIFI_BW_HT40 is supported only when the interface support 11N

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_IF: invalid interface
- ESP_ERR_INVALID_ARG: invalid argument
- others: refer to error codes in esp_err.h

Parameters

- `ifx`: interface to be configured
- `bw`: bandwidth

esp_err_t **esp_wifi_get_bandwidth**(*wifi_interface_t ifx*, *wifi_bandwidth_t* **bw*)

Get the bandwidth of ESP32 specified interface.

Attention 1. API return false if try to get a interface that is not enable

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_IF: invalid interface
- ESP_ERR_INVALID_ARG: invalid argument

Parameters

- `ifx`: interface to be configured
- `bw`: store bandwidth of interface ifx

esp_err_t **esp_wifi_set_channel**(uint8_t *primary*, *wifi_second_chan_t second*)

Set primary/secondary channel of ESP32.

Attention 1. This API should be called after esp_wifi_start()

Attention 2. When ESP32 is in STA mode, this API should not be called when STA is scanning or connecting to an external AP

Attention 3. When ESP32 is in softAP mode, this API should not be called when softAP has connected to external STAs

Attention 4. When ESP32 is in STA+softAP mode, this API should not be called when in the scenarios described above

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_IF: invalid interface
- ESP_ERR_INVALID_ARG: invalid argument

Parameters

- `primary`: for HT20, primary is the channel number, for HT40, primary is the primary channel
- `second`: for HT20, second is ignored, for HT40, second is the second channel

esp_err_t **esp_wifi_get_channel**(uint8_t *primary, *wifi_second_chan_t* *second)

Get the primary/secondary channel of ESP32.

Attention 1. API return false if try to get a interface that is not enable

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_INVALID_ARG: invalid argument

Parameters

- `primary`: store current primary channel
- `second`: store current second channel

esp_err_t **esp_wifi_set_country**(**const** *wifi_country_t* *country)

configure country info

Attention 1. The default country is {.cc=" CN" , .schan=1, .nchan=13, policy=WIFI_COUNTRY_POLICY_AUTO}

Attention 2. When the country policy is WIFI_COUNTRY_POLICY_AUTO, the country info of the AP to which the station is connected is used. E.g. if the configured country info is {.cc=" USA" , .schan=1, .nchan=11} and the country info of the AP to which the station is connected is {.cc=" JP" , .schan=1, .nchan=14} then the country info that will be used is {.cc=" JP" , .schan=1, .nchan=14}. If the station disconnected from the AP the country info is set back back to the country info of the station automatically, {.cc=" US" , .schan=1, .nchan=11} in the example.

Attention 3. When the country policy is WIFI_COUNTRY_POLICY_MANUAL, always use the configured country info.

Attention 4. When the country info is changed because of configuration or because the station connects to a different external AP, the country IE in probe response/beacon of the soft-AP is changed also.

Attention 5. The country configuration is not stored into flash

Attention 6. This API doesn't validate the per-country rules, it's up to the user to fill in all fields according to local regulations.

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_INVALID_ARG: invalid argument

Parameters

- `country`: the configured country info

esp_err_t **esp_wifi_get_country**(*wifi_country_t* **country*)

get the current country info

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_INVALID_ARG: invalid argument

Parameters

- `country`: country info

esp_err_t **esp_wifi_set_mac**(*wifi_interface_t* *ifx*, **const** uint8_t *mac*[6])

Set MAC address of the ESP32 WiFi station or the soft-AP interface.

Attention 1. This API can only be called when the interface is disabled

Attention 2. ESP32 soft-AP and station have different MAC addresses, do not set them to be the same.

Attention 3. The bit 0 of the first byte of ESP32 MAC address can not be 1. For example, the MAC address can set to be "1a:XX:XX:XX:XX:XX", but can not be "15:XX:XX:XX:XX:XX".

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_INVALID_ARG: invalid argument

- ESP_ERR_WIFI_IF: invalid interface
- ESP_ERR_WIFI_MAC: invalid mac address
- ESP_ERR_WIFI_MODE: WiFi mode is wrong
- others: refer to error codes in esp_err.h

Parameters

- `ifx`: interface
- `mac`: the MAC address

esp_err_t **esp_wifi_get_mac**(*wifi_interface_t ifx*, uint8_t *mac*[6])

Get mac of specified interface.

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_INVALID_ARG: invalid argument
- ESP_ERR_WIFI_IF: invalid interface

Parameters

- `ifx`: interface
- `mac`: store mac of the interface ifx

esp_err_t **esp_wifi_set_promiscuous_rx_cb**(*wifi_promiscuous_cb_t cb*)

Register the RX callback function in the promiscuous mode.

Each time a packet is received, the registered callback function will be called.

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init

Parameters

- `cb`: callback

esp_err_t **esp_wifi_set_promiscuous**(bool *en*)

Enable the promiscuous mode.

Return

- ESP_OK: succeed

- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init

Parameters

- `en`: false - disable, true - enable

esp_err_t **esp_wifi_get_promiscuous**(bool *en)

Get the promiscuous mode.

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_INVALID_ARG: invalid argument

Parameters

- `en`: store the current status of promiscuous mode

esp_err_t **esp_wifi_set_promiscuous_filter**(const *wifi_promiscuous_filter_t* *filter)

Enable the promiscuous mode packet type filter.

Note The default filter is to filter all packets except WIFI_PKT_MISC

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init

Parameters

- `filter`: the packet type filtered in promiscuous mode.

esp_err_t **esp_wifi_get_promiscuous_filter**(*wifi_promiscuous_filter_t* *filter)

Get the promiscuous filter.

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_INVALID_ARG: invalid argument

Parameters

- `filter`: store the current status of promiscuous filter

esp_err_t **esp_wifi_set_promiscuous_ctrl_filter**(const *wifi_promiscuous_filter_t* *filter)

Enable subtype filter of the control packet in promiscuous mode.

Note The default filter is to filter none control packet.

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init

Parameters

- `filter`: the subtype of the control packet filtered in promiscuous mode.

esp_err_t **esp_wifi_get_promiscuous_ctrl_filter**(*wifi_promiscuous_filter_t *filter*)

Get the subtype filter of the control packet in promiscuous mode.

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_ARG: invalid argument

Parameters

- `filter`: store the current status of subtype filter of the control packet in promiscuous mode

esp_err_t **esp_wifi_set_config**(*wifi_interface_t interface*, *wifi_config_t *conf*)

Set the configuration of the ESP32 STA or AP.

Attention 1. This API can be called only when specified interface is enabled, otherwise, API fail

Attention 2. For station configuration, bssid_set needs to be 0; and it needs to be 1 only when users need to check the MAC address of the AP.

Attention 3. ESP32 is limited to only one channel, so when in the soft-AP+station mode, the soft-AP will adjust its channel automatically to be the same as the channel of the ESP32 station.

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_INVALID_ARG: invalid argument
- ESP_ERR_WIFI_IF: invalid interface
- ESP_ERR_WIFI_MODE: invalid mode
- ESP_ERR_WIFI_PASSWORD: invalid password
- ESP_ERR_WIFI_NVS: WiFi internal NVS error
- others: refer to the erro code in esp_err.h

Parameters

- `interface`: interface
- `conf`: station or soft-AP configuration

esp_err_t **esp_wifi_get_config**(*wifi_interface_t interface*, *wifi_config_t* **conf*)

Get configuration of specified interface.

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_INVALID_ARG: invalid argument
- ESP_ERR_WIFI_IF: invalid interface

Parameters

- `interface`: interface
- `conf`: station or soft-AP configuration

esp_err_t **esp_wifi_ap_get_sta_list**(*wifi_sta_list_t* **sta*)

Get STAs associated with soft-AP.

Attention SSC only API

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_INVALID_ARG: invalid argument
- ESP_ERR_WIFI_MODE: WiFi mode is wrong
- ESP_ERR_WIFI_CONN: WiFi internal error, the station/soft-AP control block is invalid

Parameters

- `sta`: station list ap can get the connected sta' s phy mode info through the struct member phy_11b, phy_11g, phy_11n, phy_lr in the *wifi_sta_info_t* struct. For example, phy_11b = 1 imply that sta support 802.11b mode

esp_err_t **esp_wifi_ap_get_sta_aid**(**const** uint8_t *mac*[6], uint16_t **aid*)

Get AID of STA connected with soft-AP.

Return

- ESP_OK: succeed

- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_INVALID_ARG: invalid argument
- ESP_ERR_NOT_FOUND: Requested resource not found
- ESP_ERR_WIFI_MODE: WiFi mode is wrong
- ESP_ERR_WIFI_CONN: WiFi internal error, the station/soft-AP control block is invalid

Parameters

- `mac`: STA' s mac address
- `aid`: Store the AID corresponding to STA mac

esp_err_t **esp_wifi_set_storage**(*wifi_storage_t storage*)

Set the WiFi API configuration storage type.

Attention 1. The default value is WIFI_STORAGE_FLASH

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_INVALID_ARG: invalid argument

Parameters

- `storage`: : storage type

esp_err_t **esp_wifi_set_vendor_ie**(bool *enable*, *wifi_vendor_ie_type_t type*, *wifi_vendor_ie_id_t idx*, `const` void **vnd_ie*)

Set 802.11 Vendor-Specific Information Element.

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init()
- ESP_ERR_INVALID_ARG: Invalid argument, including if first byte of vnd_ie is not WIFI_VENDOR_IE_ELEMENT_ID (0xDD) or second byte is an invalid length.
- ESP_ERR_NO_MEM: Out of memory

Parameters

- `enable`: If true, specified IE is enabled. If false, specified IE is removed.
- `type`: Information Element type. Determines the frame type to associate with the IE.
- `idx`: Index to set or clear. Each IE type can be associated with up to two elements (indices 0 & 1).

- `vnd_ie`: Pointer to vendor specific element data. First 6 bytes should be a header with fields matching *vendor_ie_data_t*. If enable is false, this argument is ignored and can be NULL. Data does not need to remain valid after the function returns.

esp_err_t **esp_wifi_set_vendor_ie_cb**(*esp_vendor_ie_cb_t cb*, void **ctx*)

Register Vendor-Specific Information Element monitoring callback.

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init

Parameters

- `cb`: Callback function
- `ctx`: Context argument, passed to callback function.

esp_err_t **esp_wifi_set_max_tx_power**(int8_t *power*)

Set maximum transmitting power after WiFi start.

Attention 1. Maximum power before wifi startup is limited by PHY init data bin.

Attention 2. The value set by this API will be mapped to the max_tx_power of the structure *wifi_country_t* variable.

Attention 3. Mapping Table {Power, max_tx_power} = {{8, 2}, {20, 5}, {28, 7}, {34, 8}, {44, 11}, {52, 13}, {56, 14}, {60, 15}, {66, 16}, {72, 18}, {78, 20}}.

Attention 4. Param power unit is 0.25dBm, range is [8, 78] corresponding to 2dBm - 20dBm.

Attention 5. Relationship between set value and actual value. As follows: +———+———+ | set value | actual value | +==========+==========+ | [8, 19] | 8 | +———+———+ | [20, 27] | 20 | +———+———+ | [28, 33] | 28 | +———+———+ | [34, 43] | 34 | +———+———+ | [44, 51] | 44 | +———+———+ | [52, 55] | 52 | +———+———+ | [56, 59] | 56 | +———+———+ | [60, 65] | 60 | +———+———+ | [66, 71] | 66 | +———+———+ | [72, 77] | 72 | +———+———+ | 78 | 78 | +———+———+

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_NOT_START: WiFi is not started by esp_wifi_start
- ESP_ERR_WIFI_ARG: invalid argument, e.g. parameter is out of range

Parameters

- `power`: Maximum WiFi transmitting power.

esp_err_t **esp_wifi_get_max_tx_power**(int8_t *power)

Get maximum transmiting power after WiFi start.

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_NOT_START: WiFi is not started by esp_wifi_start
- ESP_ERR_WIFI_ARG: invalid argument

Parameters

- `power`: Maximum WiFi transmitting power, unit is 0.25dBm.

esp_err_t **esp_wifi_set_event_mask**(uint32_t *mask*)

Set mask to enable or disable some WiFi events.

Attention 1. Mask can be created by logical OR of various WIFI_EVENT_MASK_ constants. Events which have corresponding bit set in the mask will not be delivered to the system event handler.

Attention 2. Default WiFi event mask is WIFI_EVENT_MASK_AP_PROBEREQRECVED.

Attention 3. There may be lots of stations sending probe request data around. Don't unmask this event unless you need to receive probe request data.

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init

Parameters

- `mask`: WiFi event mask.

esp_err_t **esp_wifi_get_event_mask**(uint32_t *mask)

Get mask of WiFi events.

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_ARG: invalid argument

Parameters

- `mask`: WiFi event mask.

esp_err_t **esp_wifi_80211_tx**(*wifi_interface_t ifx*, **const** void **buffer*, int *len*, bool *en_sys_seq*)

Send raw ieee80211 data.

Attention Currently only support for sending beacon/probe request/probe response/action and non-QoS data frame

Return

- ESP_OK: success
- ESP_ERR_WIFI_IF: Invalid interface
- ESP_ERR_INVALID_ARG: Invalid parameter
- ESP_ERR_WIFI_NO_MEM: out of memory

Parameters

- `ifx`: interface if the Wi-Fi mode is Station, the ifx should be WIFI_IF_STA. If the Wi-Fi mode is SoftAP, the ifx should be WIFI_IF_AP. If the Wi-Fi mode is Station+SoftAP, the ifx should be WIFI_IF_STA or WIFI_IF_AP. If the ifx is wrong, the API returns ESP_ERR_WIFI_IF.
- `buffer`: raw ieee80211 buffer
- `len`: the length of raw buffer, the len must be <= 1500 Bytes and >= 24 Bytes
- `en_sys_seq`: indicate whether use the internal sequence number. If en_sys_seq is false, the sequence in raw buffer is unchanged, otherwise it will be overwritten by WiFi driver with the system sequence number. Generally, if esp_wifi_80211_tx is called before the Wi-Fi connection has been set up, both en_sys_seq==true and en_sys_seq==false are fine. However, if the API is called after the Wi-Fi connection has been set up, en_sys_seq must be true, otherwise ESP_ERR_WIFI_ARG is returned.

esp_err_t **esp_wifi_set_csi_rx_cb**(*wifi_csi_cb_t cb*, void **ctx*)

Register the RX callback function of CSI data.

Each time a CSI data is received, the callback function will be called.

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init

Parameters

- `cb`: callback
- `ctx`: context argument, passed to callback function

esp_err_t **esp_wifi_set_csi_config**(const *wifi_csi_config_t* **config*)

Set CSI data configuration.

return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_NOT_START: WiFi is not started by esp_wifi_start or promiscuous mode is not enabled
- ESP_ERR_INVALID_ARG: invalid argument

Parameters

- `config`: configuration

esp_err_t **esp_wifi_set_csi**(bool *en*)

Enable or disable CSI.

return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_NOT_START: WiFi is not started by esp_wifi_start or promiscuous mode is not enabled
- ESP_ERR_INVALID_ARG: invalid argument

Parameters

- `en`: true - enable, false - disable

esp_err_t **esp_wifi_set_ant_gpio**(const *wifi_ant_gpio_config_t* **config*)

Set antenna GPIO configuration.

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_ARG: Invalid argument, e.g. parameter is NULL, invalid GPIO number etc

Parameters

- `config`: Antenna GPIO configuration.

esp_err_t **esp_wifi_get_ant_gpio**(*wifi_ant_gpio_config_t* **config*)

Get current antenna GPIO configuration.

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_ARG: invalid argument, e.g. parameter is NULL

Parameters

- `config`: Antenna GPIO configuration.

esp_err_t **esp_wifi_set_ant**(**const** *wifi_ant_config_t* **config*)

Set antenna configuration.

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_ARG: Invalid argument, e.g. parameter is NULL, invalid antenna mode or invalid GPIO number

Parameters

- `config`: Antenna configuration.

esp_err_t **esp_wifi_get_ant**(*wifi_ant_config_t* **config*)

Get current antenna configuration.

Return

- ESP_OK: succeed
- ESP_ERR_WIFI_NOT_INIT: WiFi is not initialized by esp_wifi_init
- ESP_ERR_WIFI_ARG: invalid argument, e.g. parameter is NULL

Parameters

- `config`: Antenna configuration.

Structures

struct wifi_init_config_t

WiFi stack configuration parameters passed to esp_wifi_init call.

Public Members

system_event_handler_t **event_handler**
WiFi event handler

wifi_osi_funcs_t ***osi_funcs**
WiFi OS functions

wpa_crypto_funcs_t **wpa_crypto_funcs**
WiFi station crypto functions when connect

int **static_rx_buf_num**
WiFi static RX buffer number

int **dynamic_rx_buf_num**
WiFi dynamic RX buffer number

int **tx_buf_type**
WiFi TX buffer type

int **static_tx_buf_num**
WiFi static TX buffer number

int **dynamic_tx_buf_num**
WiFi dynamic TX buffer number

int **csi_enable**
WiFi channel state information enable flag

int **ampdu_rx_enable**
WiFi AMPDU RX feature enable flag

int **ampdu_tx_enable**
WiFi AMPDU TX feature enable flag

int **nvs_enable**
WiFi NVS flash enable flag

int **nano_enable**
Nano option for printf/scan family enable flag

int **tx_ba_win**
WiFi Block Ack TX window size

int **rx_ba_win**
WiFi Block Ack RX window size

int **wifi_task_core_id**
WiFi Task Core ID

int **beacon_max_len**
WiFi softAP maximum length of the beacon

int **mgmt_sbuf_num**

WiFi management short buffer number, the minimum value is 6, the maximum value is 32

uint64_t **feature_caps**

Enables additional WiFi features and capabilities

int **magic**

WiFi init magic number, it should be the last field

Macros

ESP_ERR_WIFI_NOT_INIT

WiFi driver was not installed by esp_wifi_init

ESP_ERR_WIFI_NOT_STARTED

WiFi driver was not started by esp_wifi_start

ESP_ERR_WIFI_NOT_STOPPED

WiFi driver was not stopped by esp_wifi_stop

ESP_ERR_WIFI_IF

WiFi interface error

ESP_ERR_WIFI_MODE

WiFi mode error

ESP_ERR_WIFI_STATE

WiFi internal state error

ESP_ERR_WIFI_CONN

WiFi internal control block of station or soft-AP error

ESP_ERR_WIFI_NVS

WiFi internal NVS module error

ESP_ERR_WIFI_MAC

MAC address is invalid

ESP_ERR_WIFI_SSID

SSID is invalid

ESP_ERR_WIFI_PASSWORD

Password is invalid

ESP_ERR_WIFI_TIMEOUT

Timeout error

ESP_ERR_WIFI_WAKE_FAIL

WiFi is in sleep state(RF closed) and wakeup fail

ESP_ERR_WIFI_WOULD_BLOCK
: The caller would block

ESP_ERR_WIFI_NOT_CONNECT
: Station still in disconnect status

ESP_ERR_WIFI_POST
: Failed to post the event to WiFi task

ESP_ERR_WIFI_INIT_STATE
: Invalod WiFi state when init/deinit is called

ESP_ERR_WIFI_STOP_STATE
: Returned when WiFi is stopping

WIFI_STATIC_TX_BUFFER_NUM

WIFI_DYNAMIC_TX_BUFFER_NUM

WIFI_CSI_ENABLED

WIFI_AMPDU_RX_ENABLED

WIFI_AMPDU_TX_ENABLED

WIFI_NVS_ENABLED

WIFI_NANO_FORMAT_ENABLED

WIFI_INIT_CONFIG_MAGIC

WIFI_DEFAULT_TX_BA_WIN

WIFI_DEFAULT_RX_BA_WIN

WIFI_TASK_CORE_ID

WIFI_SOFTAP_BEACON_MAX_LEN

WIFI_MGMT_SBUF_NUM

CONFIG_FEATURE_WPA3_SAE_BIT

WIFI_INIT_CONFIG_DEFAULT()

Type Definitions

typedef void (*__wifi_promiscuous_cb_t__)(void *buf, *wifi_promiscuous_pkt_type_t* type)
: The RX callback function in the promiscuous mode. Each time a packet is received, the callback function will be called.

Parameters

- `buf`: Data received. Type of data in buffer (*wifi_promiscuous_pkt_t* or *wifi_pkt_rx_ctrl_t*) indicated by 'type' parameter.
- `type`: promiscuous packet type.

`typedef` void (*`esp_vendor_ie_cb_t`)(void *ctx, *wifi_vendor_ie_type_t* type, `const` uint8_t sa[6], `const` *vendor_ie_data_t* *vnd_ie, int rssi)

Function signature for received Vendor-Specific Information Element callback.

Parameters

- `ctx`: Context argument, as passed to esp_wifi_set_vendor_ie_cb() when registering callback.
- `type`: Information element type, based on frame type received.
- `sa`: Source 802.11 address.
- `vnd_ie`: Pointer to the vendor specific element data received.
- `rssi`: Received signal strength indication.

`typedef` void (*`wifi_csi_cb_t`)(void *ctx, *wifi_csi_info_t* *data)

The RX callback function of Channel State Information(CSI) data.

Each time a CSI data is received, the callback function will be called.

Parameters

- `ctx`: context argument, passed to esp_wifi_set_csi_rx_cb() when registering callback function.
- `data`: CSI data received. The memory that it points to will be deallocated after callback function returns.

Header File

- esp_wifi/include/esp_wifi_types.h

Unions

`union wifi_config_t`

#include <esp_wifi_types.h> Configuration data for ESP32 AP or STA.

The usage of this union (for ap or sta configuration) is determined by the accompanying interface argument passed to esp_wifi_set_config() or esp_wifi_get_config()

Public Members

wifi_ap_config_t **ap**
configuration of AP

wifi_sta_config_t **sta**
configuration of STA

Structures

struct wifi_country_t
Structure describing WiFi country-based regional restrictions.

Public Members

char **cc**[3]
country code string

uint8_t **schan**
start channel

uint8_t **nchan**
total channel number

int8_t **max_tx_power**
This field is used for getting WiFi maximum transmitting power, call esp_wifi_set_max_tx_power to set the maximum transmitting power.

wifi_country_policy_t **policy**
country policy

struct wifi_active_scan_time_t
Range of active scan times per channel.

Public Members

uint32_t **min**
minimum active scan time per channel, units: millisecond

uint32_t **max**
maximum active scan time per channel, units: millisecond, values above 1500ms may cause station to disconnect from AP and are not recommended.

struct wifi_scan_time_t
Aggregate of active & passive scan time per channel.

Public Members

wifi_active_scan_time_t **active**

active scan time per channel, units: millisecond.

uint32_t **passive**

passive scan time per channel, units: millisecond, values above 1500ms may cause station to disconnect from AP and are not recommended.

struct wifi_scan_config_t

Parameters for an SSID scan.

Public Members

uint8_t ***ssid**

SSID of AP

uint8_t ***bssid**

MAC address of AP

uint8_t **channel**

channel, scan the specific channel

bool **show_hidden**

enable to scan AP whose SSID is hidden

wifi_scan_type_t **scan_type**

scan type, active or passive

wifi_scan_time_t **scan_time**

scan time per channel

struct wifi_ap_record_t

Description of a WiFi AP.

Public Members

uint8_t **bssid**[6]

MAC address of AP

uint8_t **ssid**[33]

SSID of AP

uint8_t **primary**

channel of AP

wifi_second_chan_t **second**

secondary channel of AP

int8_t **rssi**
signal strength of AP

wifi_auth_mode_t **authmode**
authmode of AP

wifi_cipher_type_t **pairwise_cipher**
pairwise cipher of AP

wifi_cipher_type_t **group_cipher**
group cipher of AP

wifi_ant_t **ant**
antenna used to receive beacon from AP

uint32_t **phy_11b**
bit: 0 flag to identify if 11b mode is enabled or not

uint32_t **phy_11g**
bit: 1 flag to identify if 11g mode is enabled or not

uint32_t **phy_11n**
bit: 2 flag to identify if 11n mode is enabled or not

uint32_t **phy_lr**
bit: 3 flag to identify if low rate is enabled or not

uint32_t **wps**
bit: 4 flag to identify if WPS is supported or not

uint32_t **reserved**
bit: 5..31 reserved

wifi_country_t **country**
country information of AP

struct wifi_scan_threshold_t
Structure describing parameters for a WiFi fast scan.

Public Members

int8_t **rssi**
The minimum rssi to accept in the fast scan mode

wifi_auth_mode_t **authmode**
The weakest authmode to accept in the fast scan mode

struct wifi_pmf_config_t
Configuration structure for Protected Management Frame

Public Members

bool **capable**

Advertizes support for Protected Management Frame. Device will prefer to connect in PMF mode if other device also advertizes PMF capability.

bool **required**

Advertizes that Protected Management Frame is required. Device will not associate to non-PMF capable devices.

struct wifi_ap_config_t

Soft-AP configuration settings for the ESP32.

Public Members

uint8_t **ssid**[32]

SSID of ESP32 soft-AP. If ssid_len field is 0, this must be a Null terminated string. Otherwise, length is set according to ssid_len.

uint8_t **password**[64]

Password of ESP32 soft-AP. Null terminated string.

uint8_t **ssid_len**

Optional length of SSID field.

uint8_t **channel**

Channel of ESP32 soft-AP

wifi_auth_mode_t **authmode**

Auth mode of ESP32 soft-AP. Do not support AUTH_WEP in soft-AP mode

uint8_t **ssid_hidden**

Broadcast SSID or not, default 0, broadcast the SSID

uint8_t **max_connection**

Max number of stations allowed to connect in, default 4, max 10

uint16_t **beacon_interval**

Beacon interval, 100 ~ 60000 ms, default 100 ms

struct wifi_sta_config_t

STA configuration settings for the ESP32.

Public Members

uint8_t **ssid**[32]

SSID of target AP. Null terminated string.

uint8_t **password**[64]

Password of target AP. Null terminated string.

wifi_scan_method_t **scan_method**

do all channel scan or fast scan

bool **bssid_set**

whether set MAC address of target AP or not. Generally, station_config.bssid_set needs to be 0; and it needs to be 1 only when users need to check the MAC address of the AP.

uint8_t **bssid**[6]

MAC address of target AP

uint8_t **channel**

channel of target AP. Set to 1~13 to scan starting from the specified channel before connecting to AP. If the channel of AP is unknown, set it to 0.

uint16_t **listen_interval**

Listen interval for ESP32 station to receive beacon when WIFI_PS_MAX_MODEM is set. Units: AP beacon intervals. Defaults to 3 if set to 0.

wifi_sort_method_t **sort_method**

sort the connect AP in the list by rssi or security mode

wifi_scan_threshold_t **threshold**

When sort_method is set, only APs which have an auth mode that is more secure than the selected auth mode and a signal stronger than the minimum RSSI will be used.

wifi_pmf_config_t **pmf_cfg**

Configuration for Protected Management Frame. Will be advertized in RSN Capabilities in RSN IE.

struct wifi_sta_info_t

Description of STA associated with AP.

Public Members

uint8_t **mac**[6]

mac address

int8_t **rssi**

current average rssi of sta connected

uint32_t **phy_11b**

bit: 0 flag to identify if 11b mode is enabled or not

uint32_t **phy_11g**

bit: 1 flag to identify if 11g mode is enabled or not

uint32_t **phy_11n**

bit: 2 flag to identify if 11n mode is enabled or not

uint32_t **phy_lr**

bit: 3 flag to identify if low rate is enabled or not

uint32_t **reserved**

bit: 4..31 reserved

struct wifi_sta_list_t

List of stations associated with the ESP32 Soft-AP.

Public Members

wifi_sta_info_t **sta**[**ESP_WIFI_MAX_CONN_NUM**]

station list

int **num**

number of stations in the list (other entries are invalid)

struct vendor_ie_data_t

Vendor Information Element header.

The first bytes of the Information Element will match this header. Payload follows.

Public Members

uint8_t **element_id**

Should be set to WIFI_VENDOR_IE_ELEMENT_ID (0xDD)

uint8_t **length**

Length of all bytes in the element data following this field. Minimum 4.

uint8_t **vendor_oui**[3]

Vendor identifier (OUI).

uint8_t **vendor_oui_type**

Vendor-specific OUI type.

uint8_t **payload**[0]

Payload. Length is equal to value in 'length' field, minus 4.

struct wifi_pkt_rx_ctrl_t

Received packet radio metadata header, this is the common header at the beginning of all promiscuous mode RX callback buffers.

Public Members

signed **rssi**

Received Signal Strength Indicator(RSSI) of packet. unit: dBm

unsigned **rate**

PHY rate encoding of the packet. Only valid for non HT(11bg) packet

unsigned **__pad0__**

reserve

unsigned **sig_mode**

0: non HT(11bg) packet; 1: HT(11n) packet; 3: VHT(11ac) packet

unsigned **__pad1__**

reserve

unsigned **mcs**

Modulation Coding Scheme. If is HT(11n) packet, shows the modulation, range from 0 to 76(MSC0 ~ MCS76)

unsigned **cwb**

Channel Bandwidth of the packet. 0: 20MHz; 1: 40MHz

unsigned **__pad2__**

reserve

unsigned **smoothing**

reserve

unsigned **not_sounding**

reserve

unsigned **__pad3__**

reserve

unsigned **aggregation**

Aggregation. 0: MPDU packet; 1: AMPDU packet

unsigned **stbc**

Space Time Block Code(STBC). 0: non STBC packet; 1: STBC packet

unsigned **fec_coding**

Flag is set for 11n packets which are LDPC

unsigned **sgi**

Short Guide Interval(SGI). 0: Long GI; 1: Short GI

unsigned **ampdu_cnt**

ampdu cnt

unsigned **channel**

primary channel on which this packet is received

unsigned **secondary_channel**

secondary channel on which this packet is received. 0: none; 1: above; 2: below

unsigned **__pad4__**

reserve

unsigned **timestamp**

timestamp. The local time when this packet is received. It is precise only if modem sleep or light sleep is not enabled. unit: microsecond

unsigned **__pad5__**

reserve

unsigned **__pad6__**

reserve

unsigned **ant**

antenna number from which this packet is received. 0: WiFi antenna 0; 1: WiFi antenna 1

unsigned **sig_len**

length of packet including Frame Check Sequence(FCS)

unsigned **__pad7__**

reserve

unsigned **rx_state**

state of the packet. 0: no error; others: error numbers which are not public

struct wifi_promiscuous_pkt_t

Payload passed to 'buf' parameter of promiscuous mode RX callback.

Public Members

wifi_pkt_rx_ctrl_t **rx_ctrl**

metadata header

uint8_t **payload**[0]

Data or management payload. Length of payload is described by rx_ctrl.sig_len. Type of content determined by packet type argument of callback.

struct wifi_promiscuous_filter_t

Mask for filtering different packet types in promiscuous mode.

Public Members

uint32_t **filter_mask**

OR of one or more filter values WIFI_PROMIS_FILTER_*

struct wifi_csi_config_t

Channel state information(CSI) configuration type.

Public Members

bool **lltf_en**

enable to receive legacy long training field(lltf) data. Default enabled

bool **htltf_en**

enable to receive HT long training field(htltf) data. Default enabled

bool **stbc_htltf2_en**

enable to receive space time block code HT long training field(stbc-htltf2) data. Default enabled

bool **ltf_merge_en**

enable to generate htlft data by averaging lltf and ht_ltf data when receiving HT packet. Otherwise, use ht_ltf data directly. Default enabled

bool **channel_filter_en**

enable to turn on channel filter to smooth adjacent sub-carrier. Disable it to keep independence of adjacent sub-carrier. Default enabled

bool **manu_scale**

manually scale the CSI data by left shifting or automatically scale the CSI data. If set true, please set the shift bits. false: automatically. true: manually. Default false

uint8_t **shift**

manually left shift bits of the scale of the CSI data. The range of the left shift bits is 0~15

struct wifi_csi_info_t

CSI data type.

Public Members

wifi_pkt_rx_ctrl_t **rx_ctrl**

received packet radio metadata header of the CSI data

uint8_t **mac**[6]

source MAC address of the CSI data

bool **first_word_invalid**

first four bytes of the CSI data is invalid or not

int8_t *buf
: buffer of CSI data

uint16_t len
: length of CSI data

struct wifi_ant_gpio_t
: WiFi GPIO configuration for antenna selection.

Public Members

uint8_t gpio_select
: Whether this GPIO is connected to external antenna switch

uint8_t gpio_num
: The GPIO number that connects to external antenna switch

struct wifi_ant_gpio_config_t
: WiFi GPIOs configuration for antenna selection.

Public Members

wifi_ant_gpio_t gpio_cfg[4]
: The configurations of GPIOs that connect to external antenna switch

struct wifi_ant_config_t
: WiFi antenna configuration.

Public Members

wifi_ant_mode_t rx_ant_mode
: WiFi antenna mode for receiving

wifi_ant_t rx_ant_default
: Default antenna mode for receiving, it's ignored if rx_ant_mode is not WIFI_ANT_MODE_AUTO

wifi_ant_mode_t tx_ant_mode
: WiFi antenna mode for transmission, it can be set to WIFI_ANT_MODE_AUTO only if rx_ant_mode is set to WIFI_ANT_MODE_AUTO

uint8_t enabled_ant0
: Index (in antenna GPIO configuration) of enabled WIFI_ANT_MODE_ANT0

uint8_t enabled_ant1
: Index (in antenna GPIO configuration) of enabled WIFI_ANT_MODE_ANT1

struct wifi_event_sta_scan_done_t

Argument structure for WIFI_EVENT_SCAN_DONE event

Public Members

uint32_t **status**

status of scanning APs: 0 —success, 1 - failure

uint8_t **number**

number of scan results

uint8_t **scan_id**

scan sequence number, used for block scan

struct wifi_event_sta_connected_t

Argument structure for WIFI_EVENT_STA_CONNECTED event

Public Members

uint8_t **ssid**[32]

SSID of connected AP

uint8_t **ssid_len**

SSID length of connected AP

uint8_t **bssid**[6]

BSSID of connected AP

uint8_t **channel**

channel of connected AP

wifi_auth_mode_t **authmode**

authentication mode used by AP

struct wifi_event_sta_disconnected_t

Argument structure for WIFI_EVENT_STA_DISCONNECTED event

Public Members

uint8_t **ssid**[32]

SSID of disconnected AP

uint8_t **ssid_len**

SSID length of disconnected AP

uint8_t **bssid**[6]

BSSID of disconnected AP

uint8_t **reason**

reason of disconnection

struct wifi_event_sta_authmode_change_t

Argument structure for WIFI_EVENT_STA_AUTHMODE_CHANGE event

Public Members

wifi_auth_mode_t **old_mode**

the old auth mode of AP

wifi_auth_mode_t **new_mode**

the new auth mode of AP

struct wifi_event_sta_wps_er_pin_t

Argument structure for WIFI_EVENT_STA_WPS_ER_PIN event

Public Members

uint8_t **pin_code**[8]

PIN code of station in enrollee mode

struct wifi_event_ap_staconnected_t

Argument structure for WIFI_EVENT_AP_STACONNECTED event

Public Members

uint8_t **mac**[6]

MAC address of the station connected to ESP32 soft-AP

uint8_t **aid**

the aid that ESP32 soft-AP gives to the station connected to

struct wifi_event_ap_stadisconnected_t

Argument structure for WIFI_EVENT_AP_STADISCONNECTED event

Public Members

uint8_t **mac**[6]

MAC address of the station disconnects to ESP32 soft-AP

uint8_t **aid**

the aid that ESP32 soft-AP gave to the station disconnects to

struct wifi_event_ap_probe_req_rx_t

Argument structure for WIFI_EVENT_AP_PROBEREQRECVED event

Public Members

int **rssi**

Received probe request signal strength

uint8_t **mac**[6]

MAC address of the station which send probe request

Macros

WIFI_IF_STA

WIFI_IF_AP

WIFI_PROTOCOL_11B

WIFI_PROTOCOL_11G

WIFI_PROTOCOL_11N

WIFI_PROTOCOL_LR

ESP_WIFI_MAX_CONN_NUM

max number of stations which can connect to ESP32 soft-AP

WIFI_VENDOR_IE_ELEMENT_ID

WIFI_PROMIS_FILTER_MASK_ALL

filter all packets

WIFI_PROMIS_FILTER_MASK_MGMT

filter the packets with type of WIFI_PKT_MGMT

WIFI_PROMIS_FILTER_MASK_CTRL

filter the packets with type of WIFI_PKT_CTRL

WIFI_PROMIS_FILTER_MASK_DATA

filter the packets with type of WIFI_PKT_DATA

WIFI_PROMIS_FILTER_MASK_MISC

filter the packets with type of WIFI_PKT_MISC

WIFI_PROMIS_FILTER_MASK_DATA_MPDU

filter the MPDU which is a kind of WIFI_PKT_DATA

WIFI_PROMIS_FILTER_MASK_DATA_AMPDU

filter the AMPDU which is a kind of WIFI_PKT_DATA

WIFI_PROMIS_CTRL_FILTER_MASK_ALL

filter all control packets

WIFI_PROMIS_CTRL_FILTER_MASK_WRAPPER
: filter the control packets with subtype of Control Wrapper

WIFI_PROMIS_CTRL_FILTER_MASK_BAR
: filter the control packets with subtype of Block Ack Request

WIFI_PROMIS_CTRL_FILTER_MASK_BA
: filter the control packets with subtype of Block Ack

WIFI_PROMIS_CTRL_FILTER_MASK_PSPOLL
: filter the control packets with subtype of PS-Poll

WIFI_PROMIS_CTRL_FILTER_MASK_RTS
: filter the control packets with subtype of RTS

WIFI_PROMIS_CTRL_FILTER_MASK_CTS
: filter the control packets with subtype of CTS

WIFI_PROMIS_CTRL_FILTER_MASK_ACK
: filter the control packets with subtype of ACK

WIFI_PROMIS_CTRL_FILTER_MASK_CFEND
: filter the control packets with subtype of CF-END

WIFI_PROMIS_CTRL_FILTER_MASK_CFENDACK
: filter the control packets with subtype of CF-END+CF-ACK

WIFI_EVENT_MASK_ALL
: mask all WiFi events

WIFI_EVENT_MASK_NONE
: mask none of the WiFi events

WIFI_EVENT_MASK_AP_PROBEREQRECVED
: mask SYSTEM_EVENT_AP_PROBEREQRECVED event

Type Definitions

typedef esp_interface_t **wifi_interface_t**

Enumerations

enum wifi_mode_t

Values:

WIFI_MODE_NULL = 0
: null mode

`WIFI_MODE_STA`
WiFi station mode

`WIFI_MODE_AP`
WiFi soft-AP mode

`WIFI_MODE_APSTA`
WiFi station + soft-AP mode

`WIFI_MODE_MAX`

`enum wifi_country_policy_t`
Values:

`WIFI_COUNTRY_POLICY_AUTO`
Country policy is auto, use the country info of AP to which the station is connected

`WIFI_COUNTRY_POLICY_MANUAL`
Country policy is manual, always use the configured country info

`enum wifi_auth_mode_t`
Values:

`WIFI_AUTH_OPEN` = 0
authenticate mode : open

`WIFI_AUTH_WEP`
authenticate mode : WEP

`WIFI_AUTH_WPA_PSK`
authenticate mode : WPA_PSK

`WIFI_AUTH_WPA2_PSK`
authenticate mode : WPA2_PSK

`WIFI_AUTH_WPA_WPA2_PSK`
authenticate mode : WPA_WPA2_PSK

`WIFI_AUTH_WPA2_ENTERPRISE`
authenticate mode : WPA2_ENTERPRISE

`WIFI_AUTH_WPA3_PSK`
authenticate mode : WPA3_PSK

`WIFI_AUTH_WPA2_WPA3_PSK`
authenticate mode : WPA2_WPA3_PSK

`WIFI_AUTH_MAX`

`enum wifi_err_reason_t`
Values:

`WIFI_REASON_UNSPECIFIED` = 1

WIFI_REASON_AUTH_EXPIRE = 2

WIFI_REASON_AUTH_LEAVE = 3

WIFI_REASON_ASSOC_EXPIRE = 4

WIFI_REASON_ASSOC_TOOMANY = 5

WIFI_REASON_NOT_AUTHED = 6

WIFI_REASON_NOT_ASSOCED = 7

WIFI_REASON_ASSOC_LEAVE = 8

WIFI_REASON_ASSOC_NOT_AUTHED = 9

WIFI_REASON_DISASSOC_PWRCAP_BAD = 10

WIFI_REASON_DISASSOC_SUPCHAN_BAD = 11

WIFI_REASON_IE_INVALID = 13

WIFI_REASON_MIC_FAILURE = 14

WIFI_REASON_4WAY_HANDSHAKE_TIMEOUT = 15

WIFI_REASON_GROUP_KEY_UPDATE_TIMEOUT = 16

WIFI_REASON_IE_IN_4WAY_DIFFERS = 17

WIFI_REASON_GROUP_CIPHER_INVALID = 18

WIFI_REASON_PAIRWISE_CIPHER_INVALID = 19

WIFI_REASON_AKMP_INVALID = 20

WIFI_REASON_UNSUPP_RSN_IE_VERSION = 21

WIFI_REASON_INVALID_RSN_IE_CAP = 22

WIFI_REASON_802_1X_AUTH_FAILED = 23

WIFI_REASON_CIPHER_SUITE_REJECTED = 24

WIFI_REASON_INVALID_PMKID = 53

WIFI_REASON_BEACON_TIMEOUT = 200

WIFI_REASON_NO_AP_FOUND = 201

WIFI_REASON_AUTH_FAIL = 202

WIFI_REASON_ASSOC_FAIL = 203

WIFI_REASON_HANDSHAKE_TIMEOUT = 204

WIFI_REASON_CONNECTION_FAIL = 205

enum wifi_second_chan_t

Values:

WIFI_SECOND_CHAN_NONE = 0

the channel width is HT20

WIFI_SECOND_CHAN_ABOVE

the channel width is HT40 and the secondary channel is above the primary channel

WIFI_SECOND_CHAN_BELOW

the channel width is HT40 and the secondary channel is below the primary channel

enum wifi_scan_type_t

Values:

WIFI_SCAN_TYPE_ACTIVE = 0

active scan

WIFI_SCAN_TYPE_PASSIVE

passive scan

enum wifi_cipher_type_t

Values:

WIFI_CIPHER_TYPE_NONE = 0

the cipher type is none

WIFI_CIPHER_TYPE_WEP40

the cipher type is WEP40

WIFI_CIPHER_TYPE_WEP104

the cipher type is WEP104

WIFI_CIPHER_TYPE_TKIP

the cipher type is TKIP

WIFI_CIPHER_TYPE_CCMP

the cipher type is CCMP

WIFI_CIPHER_TYPE_TKIP_CCMP

the cipher type is TKIP and CCMP

WIFI_CIPHER_TYPE_AES_CMAC128

the cipher type is AES-CMAC-128

WIFI_CIPHER_TYPE_UNKNOWN

the cipher type is unknown

enum wifi_ant_t

WiFi antenna.

Values:

WIFI_ANT_ANT0

WiFi antenna 0

WIFI_ANT_ANT1

WiFi antenna 1

WIFI_ANT_MAX

Invalid WiFi antenna

enum wifi_scan_method_t

Values:

WIFI_FAST_SCAN = 0

Do fast scan, scan will end after find SSID match AP

WIFI_ALL_CHANNEL_SCAN

All channel scan, scan will end after scan all the channel

enum wifi_sort_method_t

Values:

WIFI_CONNECT_AP_BY_SIGNAL = 0

Sort match AP in scan list by RSSI

WIFI_CONNECT_AP_BY_SECURITY

Sort match AP in scan list by security mode

enum wifi_ps_type_t

Values:

WIFI_PS_NONE

No power save

WIFI_PS_MIN_MODEM

Minimum modem power saving. In this mode, station wakes up to receive beacon every DTIM period

WIFI_PS_MAX_MODEM

Maximum modem power saving. In this mode, interval to receive beacons is determined by the listen_interval parameter in *wifi_sta_config_t*

enum wifi_bandwidth_t

Values:

WIFI_BW_HT20 = 1

WIFI_BW_HT40

enum wifi_storage_t

Values:

`WIFI_STORAGE_FLASH`
all configuration will store in both memory and flash

`WIFI_STORAGE_RAM`
all configuration will only store in the memory

enum wifi_vendor_ie_type_t
Vendor Information Element type.

Determines the frame type that the IE will be associated with.

Values:

`WIFI_VND_IE_TYPE_BEACON`

`WIFI_VND_IE_TYPE_PROBE_REQ`

`WIFI_VND_IE_TYPE_PROBE_RESP`

`WIFI_VND_IE_TYPE_ASSOC_REQ`

`WIFI_VND_IE_TYPE_ASSOC_RESP`

enum wifi_vendor_ie_id_t
Vendor Information Element index.

Each IE type can have up to two associated vendor ID elements.

Values:

`WIFI_VND_IE_ID_0`

`WIFI_VND_IE_ID_1`

enum wifi_promiscuous_pkt_type_t
Promiscuous frame type.

Passed to promiscuous mode RX callback to indicate the type of parameter in the buffer.

Values:

`WIFI_PKT_MGMT`
Management frame, indicates ‘buf’ argument is *wifi_promiscuous_pkt_t*

`WIFI_PKT_CTRL`
Control frame, indicates ‘buf’ argument is *wifi_promiscuous_pkt_t*

`WIFI_PKT_DATA`
Data frame, indiciates ‘buf’ argument is *wifi_promiscuous_pkt_t*

`WIFI_PKT_MISC`
Other type, such as MIMO etc. ‘buf’ argument is *wifi_promiscuous_pkt_t* but the payload is zero length.

enum wifi_ant_mode_t

WiFi antenna mode.

Values:

WIFI_ANT_MODE_ANT0

Enable WiFi antenna 0 only

WIFI_ANT_MODE_ANT1

Enable WiFi antenna 1 only

WIFI_ANT_MODE_AUTO

Enable WiFi antenna 0 and 1, automatically select an antenna

WIFI_ANT_MODE_MAX

Invalid WiFi enabled antenna

enum wifi_phy_rate_t

WiFi PHY rate encodings.

Values:

WIFI_PHY_RATE_1M_L = 0x00

1 Mbps with long preamble

WIFI_PHY_RATE_2M_L = 0x01

2 Mbps with long preamble

WIFI_PHY_RATE_5M_L = 0x02

5.5 Mbps with long preamble

WIFI_PHY_RATE_11M_L = 0x03

11 Mbps with long preamble

WIFI_PHY_RATE_2M_S = 0x05

2 Mbps with short preamble

WIFI_PHY_RATE_5M_S = 0x06

5.5 Mbps with short preamble

WIFI_PHY_RATE_11M_S = 0x07

11 Mbps with short preamble

WIFI_PHY_RATE_48M = 0x08

48 Mbps

WIFI_PHY_RATE_24M = 0x09

24 Mbps

WIFI_PHY_RATE_12M = 0x0A

12 Mbps

`WIFI_PHY_RATE_6M` = 0x0B
6 Mbps

`WIFI_PHY_RATE_54M` = 0x0C
54 Mbps

`WIFI_PHY_RATE_36M` = 0x0D
36 Mbps

`WIFI_PHY_RATE_18M` = 0x0E
18 Mbps

`WIFI_PHY_RATE_9M` = 0x0F
9 Mbps

`WIFI_PHY_RATE_MCS0_LGI` = 0x10
MCS0 with long GI, 6.5 Mbps for 20MHz, 13.5 Mbps for 40MHz

`WIFI_PHY_RATE_MCS1_LGI` = 0x11
MCS1 with long GI, 13 Mbps for 20MHz, 27 Mbps for 40MHz

`WIFI_PHY_RATE_MCS2_LGI` = 0x12
MCS2 with long GI, 19.5 Mbps for 20MHz, 40.5 Mbps for 40MHz

`WIFI_PHY_RATE_MCS3_LGI` = 0x13
MCS3 with long GI, 26 Mbps for 20MHz, 54 Mbps for 40MHz

`WIFI_PHY_RATE_MCS4_LGI` = 0x14
MCS4 with long GI, 39 Mbps for 20MHz, 81 Mbps for 40MHz

`WIFI_PHY_RATE_MCS5_LGI` = 0x15
MCS5 with long GI, 52 Mbps for 20MHz, 108 Mbps for 40MHz

`WIFI_PHY_RATE_MCS6_LGI` = 0x16
MCS6 with long GI, 58.5 Mbps for 20MHz, 121.5 Mbps for 40MHz

`WIFI_PHY_RATE_MCS7_LGI` = 0x17
MCS7 with long GI, 65 Mbps for 20MHz, 135 Mbps for 40MHz

`WIFI_PHY_RATE_MCS0_SGI` = 0x18
MCS0 with short GI, 7.2 Mbps for 20MHz, 15 Mbps for 40MHz

`WIFI_PHY_RATE_MCS1_SGI` = 0x19
MCS1 with short GI, 14.4 Mbps for 20MHz, 30 Mbps for 40MHz

`WIFI_PHY_RATE_MCS2_SGI` = 0x1A
MCS2 with short GI, 21.7 Mbps for 20MHz, 45 Mbps for 40MHz

`WIFI_PHY_RATE_MCS3_SGI` = 0x1B
MCS3 with short GI, 28.9 Mbps for 20MHz, 60 Mbps for 40MHz

WIFI_PHY_RATE_MCS4_SGI = 0x1C

MCS4 with short GI, 43.3 Mbps for 20MHz, 90 Mbps for 40MHz

WIFI_PHY_RATE_MCS5_SGI = 0x1D

MCS5 with short GI, 57.8 Mbps for 20MHz, 120 Mbps for 40MHz

WIFI_PHY_RATE_MCS6_SGI = 0x1E

MCS6 with short GI, 65 Mbps for 20MHz, 135 Mbps for 40MHz

WIFI_PHY_RATE_MCS7_SGI = 0x1F

MCS7 with short GI, 72.2 Mbps for 20MHz, 150 Mbps for 40MHz

WIFI_PHY_RATE_LORA_250K = 0x29

250 Kbps

WIFI_PHY_RATE_LORA_500K = 0x2A

500 Kbps

WIFI_PHY_RATE_MAX

enum wifi_event_t

WiFi event declarations

Values:

WIFI_EVENT_WIFI_READY = 0

ESP32 WiFi ready

WIFI_EVENT_SCAN_DONE

ESP32 finish scanning AP

WIFI_EVENT_STA_START

ESP32 station start

WIFI_EVENT_STA_STOP

ESP32 station stop

WIFI_EVENT_STA_CONNECTED

ESP32 station connected to AP

WIFI_EVENT_STA_DISCONNECTED

ESP32 station disconnected from AP

WIFI_EVENT_STA_AUTHMODE_CHANGE

the auth mode of AP connected by ESP32 station changed

WIFI_EVENT_STA_WPS_ER_SUCCESS

ESP32 station wps succeeds in enrollee mode

WIFI_EVENT_STA_WPS_ER_FAILED

ESP32 station wps fails in enrollee mode

`WIFI_EVENT_STA_WPS_ER_TIMEOUT`
ESP32 station wps timeout in enrollee mode

`WIFI_EVENT_STA_WPS_ER_PIN`
ESP32 station wps pin code in enrollee mode

`WIFI_EVENT_STA_WPS_ER_PBC_OVERLAP`
ESP32 station wps overlap in enrollee mode

`WIFI_EVENT_AP_START`
ESP32 soft-AP start

`WIFI_EVENT_AP_STOP`
ESP32 soft-AP stop

`WIFI_EVENT_AP_STACONNECTED`
a station connected to ESP32 soft-AP

`WIFI_EVENT_AP_STADISCONNECTED`
a station disconnected from ESP32 soft-AP

`WIFI_EVENT_AP_PROBEREQRECVED`
Receive probe request packet in soft-AP interface

`WIFI_EVENT_MAX`
Invalid WiFi event ID

`enum wifi_event_sta_wps_fail_reason_t`
Argument structure for WIFI_EVENT_STA_WPS_ER_FAILED event

Values:

`WPS_FAIL_REASON_NORMAL` = 0
ESP32 WPS normal fail reason

`WPS_FAIL_REASON_RECV_M2D`
ESP32 WPS receive M2D frame

`WPS_FAIL_REASON_MAX`

SmartConfig

[English]

API 参考

Header File

- esp_wifi/include/esp_smartconfig.h

Functions

const char *esp_smartconfig_get_version(void)

Get the version of SmartConfig.

Return

- SmartConfig version const char.

esp_err_t esp_smartconfig_start(const *smartconfig_start_config_t* *config)

Start SmartConfig, config ESP device to connect AP. You need to broadcast information by phone APP. Device sniffer special packets from the air that containing SSID and password of target AP.

Attention 1. This API can be called in station or softAP-station mode.

Attention 2. Can not call esp_smartconfig_start twice before it finish, please call esp_smartconfig_stop first.

Return

- ESP_OK: succeed
- others: fail

Parameters

- config: pointer to smartconfig start configure structure

esp_err_t esp_smartconfig_stop(void)

Stop SmartConfig, free the buffer taken by esp_smartconfig_start.

Attention Whether connect to AP succeed or not, this API should be called to free memory taken by smartconfig_start.

Return

- ESP_OK: succeed
- others: fail

esp_err_t esp_esptouch_set_timeout(uint8_t *time_s*)

Set timeout of SmartConfig process.

Attention Timing starts from SC_STATUS_FIND_CHANNEL status. SmartConfig will restart if timeout.

Return

- ESP_OK: succeed
- others: fail

Parameters

- `time_s`: range 15s~255s, offset:45s.

esp_err_t **esp_smartconfig_set_type**(*smartconfig_type_t type*)

Set protocol type of SmartConfig.

Attention If users need to set the SmartConfig type, please set it before calling esp_smartconfig_start.

Return

- ESP_OK: succeed
- others: fail

Parameters

- `type`: Choose from the smartconfig_type_t.

esp_err_t **esp_smartconfig_fast_mode**(bool *enable*)

Set mode of SmartConfig. default normal mode.

Attention 1. Please call it before API esp_smartconfig_start.

Attention 2. Fast mode have corresponding APP(phone).

Attention 3. Two mode is compatible.

Return

- ESP_OK: succeed
- others: fail

Parameters

- `enable`: false-disable(default); true-enable;

Structures

struct smartconfig_event_got_ssid_pswd_t

Argument structure for SC_EVENT_GOT_SSID_PSWD event

Public Members

uint8_t **ssid**[32]

SSID of the AP. Null terminated string.

uint8_t **password**[64]

Password of the AP. Null terminated string.

bool **bssid_set**

whether set MAC address of target AP or not.

uint8_t **bssid**[6]

MAC address of target AP.

smartconfig_type_t **type**

Type of smartconfig(ESPTouch or AirKiss).

uint8_t **token**

Token from cellphone which is used to send ACK to cellphone.

uint8_t **cellphone_ip**[4]

IP address of cellphone.

struct smartconfig_start_config_t

Configure structure for esp_smartconfig_start

Public Members

bool **enable_log**

Enable smartconfig logs.

Macros

SMARTCONFIG_START_CONFIG_DEFAULT()

Enumerations

enum smartconfig_type_t

Values:

SC_TYPE_ESPTOUCH = 0

protocol: ESPTouch

SC_TYPE_AIRKISS

protocol: AirKiss

SC_TYPE_ESPTOUCH_AIRKISS

protocol: ESPTouch and AirKiss

enum smartconfig_event_t

Smartconfig event declarations

Values:

SC_EVENT_SCAN_DONE

ESP32 station smartconfig has finished to scan for APs

`SC_EVENT_FOUND_CHANNEL`
ESP32 station smartconfig has found the channel of the target AP

`SC_EVENT_GOT_SSID_PSWD`
ESP32 station smartconfig got the SSID and password

`SC_EVENT_SEND_ACK_DONE`
ESP32 station smartconfig has sent ACK to cellphone

ESP-NOW

[英文]

概述

ESP-NOW 是一种由乐鑫公司定义的无连接 Wi-Fi 通信协议。在 ESP-NOW 中，应用程序数据被封装在各个供应商的动作帧中，然后在无连接的情况下，从一个 Wi-Fi 设备传输到另一个 Wi-Fi 设备。CTR 与 CBC-MAC 协议 (CCMP) 可用来保护动作帧的安全。ESP-NOW 广泛应用于智能照明、远程控制、传感器等领域。

帧格式

ESP-NOW 使用各个供应商的动作帧传输数据，默认比特率为 1 Mbps。各个供应商的动作帧格式为：

MAC 报头	分类代码	组织标识符	随机值	供应商特定内容	FCS
24 字节	1 字节	3 字节	4 字节	7~255 字节	4 字节

- 分类代码：分类代码字段可用于指示各个供应商的类别（比如 127）。
- 组织标识符：组织标识符包含一个唯一标识符 (比如 0x18fe34)，为乐鑫指定的 MAC 地址的前三个字节。
- 随机值：防止重放攻击。
- 供应商特定内容：供应商特定内容包含供应商特定字段，如下所示：

元素 ID	长度	组织标识符	类型	版本	正文
1 字节	1 字节	3 字节	1 字节	1 字节	0 ~ 250 字节

- 元素 ID：元素 ID 字段可用于指示特定于供应商的元素。

- 长度：长度是组织标识符、类型、版本和正文的总长度。
- 组织标识符：组织标识符包含一个唯一标识符 (比如 0x18fe34)，为乐鑫指定的 MAC 地址的前三个字节。
- 类型：类型字段设置为 4，代表 ESP-NOW。
- 版本：版本字段设置为 ESP-NOW 的版本。
- 正文：正文包含 ESP-NOW 数据。

由于 ESP-NOW 是无连接的，因此 MAC 报头与标准帧略有不同。FrameControl 字段的 FromDS 和 ToDS 位均为 0。第一个地址字段用于配置目标地址。第二个地址字段用于配置源地址。第三个地址字段用于配置广播地址 (0xff:0xff:0xff:0xff:0xff:0xff)。

安全

ESP-NOW 采用 CCMP 方法保护供应商特定动作帧的安全，具体可参考 IEEE Std. 802.11-2012。Wi-Fi 设备维护一

- PMK 可使用 AES-128 算法加密 LMK。请调用 `esp_now_set_pmk()` 设置 PMK。如果未设置 PMK，将使用默认 PMK。
- LMK 可通过 CCMP 方法对供应商特定的动作帧进行加密，最多拥有 6 个不同的 LMK。如果未设置配对设备的 LMK，则动作帧不进行加密。

目前，不支持加密组播供应商特定的动作帧。

初始化和反初始化

调用 `esp_now_init()` 初始化 ESP-NOW，调用 `esp_now_deinit()` 反初始化 ESP-NOW。ESP-NOW 数据必须在 Wi-Fi 启动后传输，因此建议在初始化 ESP-NOW 之前启动 Wi-Fi，并在反初始化 ESP-NOW 之后停止 Wi-Fi。当调用 `esp_now_deinit()` 时，配对设备的所有信息都将被删除。

添加配对设备

在将数据发送到其他设备之前，请先调用 `esp_now_add_peer()` 将其添加到配对设备列表中。配对设备的最大数量是 20。如果启用了加密，则必须设置 LMK。ESP-NOW 数据可以从 Station 或 Softap 接口发送。确保在发送 ESP-NOW 数据之前已启用该接口。在发送广播数据之前必须添加具有广播 MAC 地址的设备。配对设备的信道范围是从 0 ~ 14。如果信道设置为 0，数据将在当前信道上发送。否则，必须使用本地设备所在的通道。

发送 ESP-NOW 数据

调用 `esp_now_send()` 发送 ESP-NOW 数据，调用 `esp_now_register_send_cb` 注册发送回调函数。如果 MAC 层成功接收到数据，则该函数将返回 *ESP_NOW_SEND_SUCCESS* 事件。否则，它将返回 *ESP_NOW_SEND_FAIL*。ESP-NOW 数据发送失败可能有几种原因，比如目标设备不存在、设备的信道不相同、动作帧在传输过程中丢失等。应用层并不一定可以总能接收到数据。如果需要，应用层可在接收 ESP-NOW 数据时发回一个应答 (ACK) 数据。如果接收 ACK 数据超时，则将重新传输 ESP-NOW 数据。可以为 ESP-NOW 数据设置序列号，从而删除重复的数据。

如果有大量 ESP-NOW 数据要发送，则调用 `esp_now_send()` 一次性发送不大于 250 字节的数据。请注意，两个 ESP-NOW 数据包的发送间隔太短可能导致回调函数返回混乱。因此，建议在等到上一次回调函数返回 ACK 后再发送下一个 ESP-NOW 数据。发送回调函数从高优先级的 Wi-Fi 任务中运行。因此，不要在回调函数中执行冗长的操作。相反，将必要的数据发布到队列，并交给优先级较低的任务处理。

接收 ESP-NOW 数据

调用 `esp_now_register_recv_cb` 注册接收回调函数。当接收 ESP-NOW 数据时，需要调用接收回调函数。接收回调函数也在 Wi-Fi 任务任务中运行。因此，不要在回调函数中执行冗长的操作。相反，将必要的数据发布到队列，并交给优先级较低的任务处理。

API 参考

Header File

- esp_wifi/include/esp_now.h

Functions

esp_err_t **esp_now_init**(void)

Initialize ESPNOW function.

Return

- ESP_OK : succeed
- ESP_ERR_ESPNOW_INTERNAL : Internal error

esp_err_t **esp_now_deinit**(void)

De-initialize ESPNOW function.

Return

- ESP_OK : succeed

esp_err_t **esp_now_get_version**(uint32_t **version*)

Get the version of ESPNOW.

Return

- ESP_OK : succeed
- ESP_ERR_ESPNOW_ARG : invalid argument

Parameters

- `version`: ESPNOW version

esp_err_t **esp_now_register_recv_cb**(*esp_now_recv_cb_t cb*)

Register callback function of receiving ESPNOW data.

Return

- ESP_OK : succeed
- ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized
- ESP_ERR_ESPNOW_INTERNAL : internal error

Parameters

- `cb`: callback function of receiving ESPNOW data

esp_err_t **esp_now_unregister_recv_cb**(void)

Unregister callback function of receiving ESPNOW data.

Return

- ESP_OK : succeed
- ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized

esp_err_t **esp_now_register_send_cb**(*esp_now_send_cb_t cb*)

Register callback function of sending ESPNOW data.

Return

- ESP_OK : succeed
- ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized
- ESP_ERR_ESPNOW_INTERNAL : internal error

Parameters

- `cb`: callback function of sending ESPNOW data

esp_err_t **esp_now_unregister_send_cb**(void)

Unregister callback function of sending ESPNOW data.

Return

- ESP_OK : succeed
- ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized

esp_err_t **esp_now_send**(**const** uint8_t **peer_addr*, **const** uint8_t **data*, size_t *len*)

Send ESPNOW data.

Attention 1. If peer_addr is not NULL, send data to the peer whose MAC address matches peer_addr

Attention 2. If peer_addr is NULL, send data to all of the peers that are added to the peer list

Attention 3. The maximum length of data must be less than ESP_NOW_MAX_DATA_LEN

Attention 4. The buffer pointed to by data argument does not need to be valid after esp_now_send returns

Return

- ESP_OK : succeed
- ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized
- ESP_ERR_ESPNOW_ARG : invalid argument
- ESP_ERR_ESPNOW_INTERNAL : internal error
- ESP_ERR_ESPNOW_NO_MEM : out of memory
- ESP_ERR_ESPNOW_NOT_FOUND : peer is not found
- ESP_ERR_ESPNOW_IF : current WiFi interface doesn't match that of peer

Parameters

- `peer_addr`: peer MAC address
- `data`: data to send
- `len`: length of data

esp_err_t **esp_now_add_peer**(**const** *esp_now_peer_info_t* **peer*)

Add a peer to peer list.

Return

- ESP_OK : succeed
- ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized
- ESP_ERR_ESPNOW_ARG : invalid argument

- ESP_ERR_ESPNOW_FULL : peer list is full
- ESP_ERR_ESPNOW_NO_MEM : out of memory
- ESP_ERR_ESPNOW_EXIST : peer has existed

Parameters

- `peer`: peer information

esp_err_t **esp_now_del_peer**(**const** uint8_t **peer_addr*)

Delete a peer from peer list.

Return

- ESP_OK : succeed
- ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized
- ESP_ERR_ESPNOW_ARG : invalid argument
- ESP_ERR_ESPNOW_NOT_FOUND : peer is not found

Parameters

- `peer_addr`: peer MAC address

esp_err_t **esp_now_mod_peer**(**const** *esp_now_peer_info_t* **peer*)

Modify a peer.

Return

- ESP_OK : succeed
- ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized
- ESP_ERR_ESPNOW_ARG : invalid argument
- ESP_ERR_ESPNOW_FULL : peer list is full

Parameters

- `peer`: peer information

esp_err_t **esp_now_get_peer**(**const** uint8_t **peer_addr*, *esp_now_peer_info_t* **peer*)

Get a peer whose MAC address matches peer_addr from peer list.

Return

- ESP_OK : succeed
- ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized
- ESP_ERR_ESPNOW_ARG : invalid argument

- ESP_ERR_ESPNOW_NOT_FOUND : peer is not found

Parameters

- `peer_addr`: peer MAC address
- `peer`: peer information

esp_err_t **esp_now_fetch_peer**(bool *from_head*, *esp_now_peer_info_t* **peer*)

Fetch a peer from peer list.

Return

- ESP_OK : succeed
- ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized
- ESP_ERR_ESPNOW_ARG : invalid argument
- ESP_ERR_ESPNOW_NOT_FOUND : peer is not found

Parameters

- `from_head`: fetch from head of list or not
- `peer`: peer information

bool **esp_now_is_peer_exist**(**const** uint8_t **peer_addr*)

Peer exists or not.

Return

- true : peer exists
- false : peer not exists

Parameters

- `peer_addr`: peer MAC address

esp_err_t **esp_now_get_peer_num**(*esp_now_peer_num_t* **num*)

Get the number of peers.

Return

- ESP_OK : succeed
- ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized
- ESP_ERR_ESPNOW_ARG : invalid argument

Parameters

- `num`: number of peers

esp_err_t **esp_now_set_pmk**(**const** uint8_t *pmk)

Set the primary master key.

Attention 1. primary master key is used to encrypt local master key

Return

- ESP_OK : succeed
- ESP_ERR_ESPNOW_NOT_INIT : ESPNOW is not initialized
- ESP_ERR_ESPNOW_ARG : invalid argument

Parameters

- `pmk`: primary master key

Structures

struct esp_now_peer_info

ESPNOW peer information parameters.

Public Members

uint8_t **peer_addr**[**ESP_NOW_ETH_ALEN**]

ESPNOW peer MAC address that is also the MAC address of station or softap

uint8_t **lmk**[**ESP_NOW_KEY_LEN**]

ESPNOW peer local master key that is used to encrypt data

uint8_t **channel**

Wi-Fi channel that peer uses to send/receive ESPNOW data. If the value is 0, use the current channel which station or softap is on. Otherwise, it must be set as the channel that station or softap is on.

wifi_interface_t **ifidx**

Wi-Fi interface that peer uses to send/receive ESPNOW data

bool **encrypt**

ESPNOW data that this peer sends/receives is encrypted or not

void ***priv**

ESPNOW peer private data

struct esp_now_peer_num

Number of ESPNOW peers which exist currently.

Public Members

int **total_num**

Total number of ESPNOW peers, maximum value is ESP_NOW_MAX_TOTAL_PEER_NUM

int **encrypt_num**

Number of encrypted ESPNOW peers, maximum value is ESP_NOW_MAX_ENCRYPT_PEER_NUM

Macros

ESP_ERR_ESPNOW_BASE

ESPNOW error number base.

ESP_ERR_ESPNOW_NOT_INIT

ESPNOW is not initialized.

ESP_ERR_ESPNOW_ARG

Invalid argument

ESP_ERR_ESPNOW_NO_MEM

Out of memory

ESP_ERR_ESPNOW_FULL

ESPNOW peer list is full

ESP_ERR_ESPNOW_NOT_FOUND

ESPNOW peer is not found

ESP_ERR_ESPNOW_INTERNAL

Internal error

ESP_ERR_ESPNOW_EXIST

ESPNOW peer has existed

ESP_ERR_ESPNOW_IF

Interface error

ESP_NOW_ETH_ALEN

Length of ESPNOW peer MAC address

ESP_NOW_KEY_LEN

Length of ESPNOW peer local master key

ESP_NOW_MAX_TOTAL_PEER_NUM

Maximum number of ESPNOW total peers

ESP_NOW_MAX_ENCRYPT_PEER_NUM

Maximum number of ESPNOW encrypted peers

ESP_NOW_MAX_DATA_LEN

Maximum length of ESPNOW data which is sent very time

Type Definitions

typedef struct *esp_now_peer_info* esp_now_peer_info_t

ESPNOW peer information parameters.

typedef struct *esp_now_peer_num* esp_now_peer_num_t

Number of ESPNOW peers which exist currently.

typedef void (*esp_now_recv_cb_t)(const uint8_t *mac_addr, const uint8_t *data, int data_len)

Callback function of receiving ESPNOW data.

Parameters

- mac_addr: peer MAC address
- data: received data
- data_len: length of received data

typedef void (*esp_now_send_cb_t)(const uint8_t *mac_addr, *esp_now_send_status_t* status)

Callback function of sending ESPNOW data.

Parameters

- mac_addr: peer MAC address
- status: status of sending ESPNOW data (succeed or fail)

Enumerations

enum esp_now_send_status_t

Status of sending ESPNOW data .

Values:

ESP_NOW_SEND_SUCCESS = 0

Send ESPNOW data successfully

ESP_NOW_SEND_FAIL

Send ESPNOW data fail

ESP-MESH 编程指南

[English]

这是 ESP-MESH 的编程指南，包括 API 参考和编码示例。本指南分为以下部分：

1. *ESP-MESH 编程模型*
2. *编写 ESP-MESH 应用程序*
3. *自组网*
4. *应用实例*
5. *API 参考*

有关 ESP-MESH 协议的文档，请见*ESP-MESH API 指南*。有关 ESP-MESH 开发框架的更多内容，请见 ESP-MESH 开发框架。

ESP-MESH 编程模型

软件栈

ESP-MESH 软件栈基于 Wi-Fi 驱动程序和 FreeRTOS 构建，某些情况下（如根节点）也会使用 LwIP 软件栈。下图展示了 ESP-MESH 软件栈。

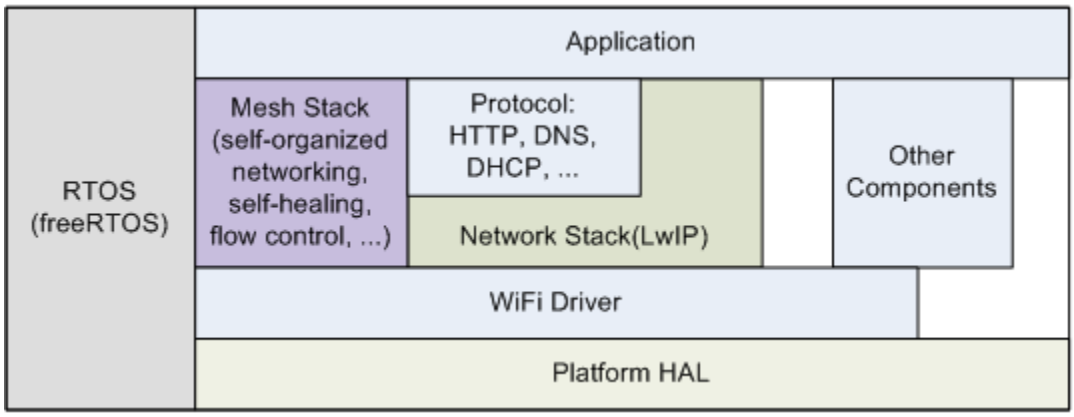


图 2: ESP-MESH 软件栈

系统事件

应用程序可通过 **ESP-MESH 事件**与 ESP-MESH 交互。由于 ESP-MESH 构建在 Wi-Fi 软件栈之上，因此也可以通过 **Wi-Fi 事件任务**与 Wi-Fi 驱动程序进行交互。下图展示了 ESP-MESH 应用程序中各种系统事件的接口。

mesh_event_id_t 定义了所有可能的 ESP-MESH 事件，并且可以指示父节点和子节点的连接或断开等事件。应用程序如需使用 ESP-MESH 事件，则必须通过*esp_event_handler_register()* 将 **Mesh 事件处理程序**注册在默认事件任务中。注册完成后，ESP-MESH 事件将包含与应用程序所有相关事件相关的处理程序。

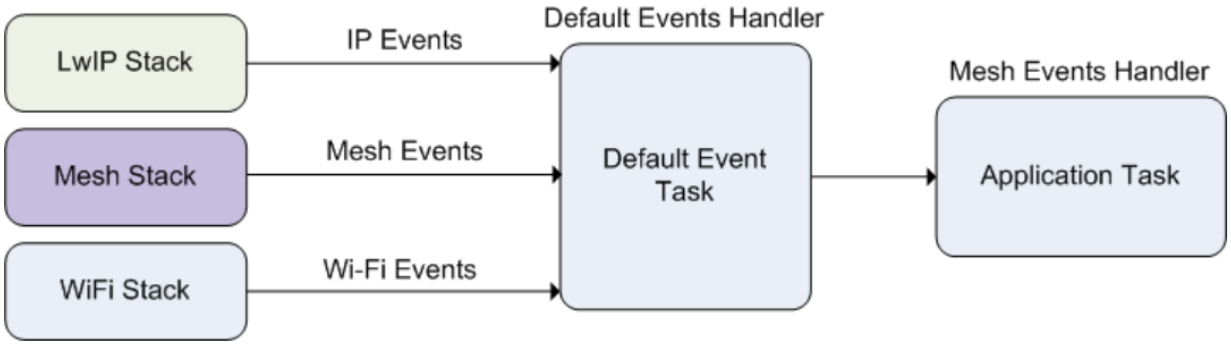


图 3: ESP-MESH 系统事件交付

Mesh 事件的典型应用场景包括：使用*MESH_EVENT_PARENT_CONNECTED* 和*MESH_EVENT_CHILD_CONNECTED* 事件来指示节点何时可以分别开始传输上行和下行的数据。同样，也可以使用 IP_EVENT_STA_GOT_IP 和 IP_EVENT_STA_LOST_IP 事件来指示根节点何时可以向外部 IP 网络传输数据。

警告： 在自组网模式下使用 ESP-MESH 时，用户必须确保不得调用 Wi-Fi API。原因在于：自组网模式将在内部调用 Wi-Fi API 实现连接/断开/扫描等操作。**此时，如果外部应用程序调用 Wi-Fi API（包括来自回调函数和 Wi-Fi 事件处理程序的调用）都可能会干扰 ESP-MESH 的自组网行为**。因此，用户不应该在*esp_mesh_start()* 和*esp_mesh_stop()* 之间调用 Wi-Fi API。

LwIP & ESP-MESH

应用程序无需通过 LwIP 层便可直接访问 ESP-MESH 软件栈，LwIP 层仅在根节点和外部 IP 网络的数据发送与接收时会用到。但是，由于每个节点都有可能成为根节点（由于自动根节点选择机制的存在），每个节点仍必须初始化 LwIP 软件栈。

每个节点都需要通过调用 tcpip_adapter_init() 初始化 LwIP 软件栈。为了防止非根节点访问 LwIP，应用程序应该在 LwIP 初始化完成后停止以下服务：

- SoftAP 接口上的 DHCP 服务器服务。
- Station 接口上的 DHCP 客户端服务。

下方代码片段展示如何为 ESP-MESH 应用程序进行 LwIP 初始化。

```
/*  tcpip 初始化 */
tcpip_adapter_init();
/*
* 对于 MESH
* 默认情况下，在 SoftAP 接口上停止 DHCP 服务器
* 默认情况下，在 Station 接口上停止 DHCP 客户端
```

(下页继续)

(续上页)

```
*/
ESP_ERROR_CHECK(tcpip_adapter_dhcps_stop(TCPIP_ADAPTER_IF_AP));
ESP_ERROR_CHECK(tcpip_adapter_dhcpc_stop(TCPIP_ADAPTER_IF_STA));
```

注解： ESP-MESH 的根节点必须与路由器连接。因此，当一个节点成为根节点时，**该节点对应的处理程序必须启动 DHCP 客户端服务并立即获取 IP 地址**。这样做将允许其他节点开始向/从外部 IP 网络发送/接收数据包。但是，如果使用静态 IP 设置，则不需要执行此步骤。

编写 ESP-MESH 应用程序

ESP-MESH 在正常启动前必须先初始化 LwIP 和 Wi-Fi 软件栈。下方代码展示了 ESP-MESH 在开始自身初始化前必须完成的步骤。

```
tcpip_adapter_init();
/*
* 对于 MESH
* 默认情况下，在 SoftAP 接口上停止 DHCP 服务器
* 默认情况下，在 Station 接口上停止 DHCP 客户端
*/
ESP_ERROR_CHECK(tcpip_adapter_dhcps_stop(TCPIP_ADAPTER_IF_AP));
ESP_ERROR_CHECK(tcpip_adapter_dhcpc_stop(TCPIP_ADAPTER_IF_STA));

/* 事件初始化 */
ESP_ERROR_CHECK(esp_event_loop_create_default());

/*Wi-Fi 初始化 */
wifi_init_config_t config = WIFI_INIT_CONFIG_DEFAULT();
ESP_ERROR_CHECK(esp_wifi_init(&config));
/* 注册 IP 事件处理程序 */
ESP_ERROR_CHECK(esp_event_handler_register(IP_EVENT, IP_EVENT_STA_GOT_IP, &ip_event_
↪handler, NULL));
ESP_ERROR_CHECK(esp_wifi_set_storage(WIFI_STORAGE_FLASH));
ESP_ERROR_CHECK(esp_wifi_start());
```

在完成 LwIP 和 Wi-Fi 的初始化后，需完成以下三个步骤以启动并运行 ESP-MESH。

1. *初始化 Mesh*
2. *配置 ESP-MESH 网络*
3. *启动 Mesh*

初始化 Mesh

下方代码片段展示如何初始化 ESP-MESH。

```
/*Mesh 初始化 */
ESP_ERROR_CHECK(esp_mesh_init());
/* 注册 mesh 事件处理程序 */
ESP_ERROR_CHECK(esp_event_handler_register(MESH_EVENT, ESP_EVENT_ANY_ID, &mesh_event_
↪handler, NULL));
```

配置 ESP-MESH 网络

ESP-MESH 可通过*esp_mesh_set_config()* 进行配置，并使用*mesh_cfg_t* 结构体传递参数。该结构体包含以下 ESP-MESH 的配置参数：

参数	描述
Channel（信道）	1 到 14 信道
Mesh ID	ESP-MESH 网络的 ID，见*mesh_addr_t*。
Router（路由器）	路由器配置，见*mesh_router_t*。
Mesh AP	Mesh AP 配置，见*mesh_ap_cfg_t*
Crypto Functions（加密函数）	Mesh IE 的加密函数，见 mesh_crypto_funcs_t。

下方代码片段展示如何配置 ESP-MESH。

```
/* 默认启用 MESH IE 加密 */
mesh_cfg_t cfg = MESH_INIT_CONFIG_DEFAULT();
/* Mesh ID */
memcpy((uint8_t *) &cfg.mesh_id, MESH_ID, 6);
/* 信道（需与路由器信道匹配）*/
cfg.channel = CONFIG_MESH_CHANNEL;
/* 路由器 */
cfg.router.ssid_len = strlen(CONFIG_MESH_ROUTER_SSID);
memcpy((uint8_t *) &cfg.router.ssid, CONFIG_MESH_ROUTER_SSID, cfg.router.ssid_len);
memcpy((uint8_t *) &cfg.router.password, CONFIG_MESH_ROUTER_PASSWD,
    strlen(CONFIG_MESH_ROUTER_PASSWD));
/* Mesh softAP */
cfg.mesh_ap.max_connection = CONFIG_MESH_AP_CONNECTIONS;
memcpy((uint8_t *) &cfg.mesh_ap.password, CONFIG_MESH_AP_PASSWD,
    strlen(CONFIG_MESH_AP_PASSWD));
ESP_ERROR_CHECK(esp_mesh_set_config(&cfg));
```

启动 Mesh

下方代码片段展示如何启动 ESP-MESH。

```
/* 启动 Mesh */
ESP_ERROR_CHECK(esp_mesh_start());
```

启动 ESP-MESH 后，应用程序应检查 ESP-MESH 事件，以确定它是何时连接到网络的。连接后，应用程序可使用*esp_mesh_send()* 和*esp_mesh_recv()* 在 ESP-MESH 网络中发送、接收数据包。

自组网

自组网是 ESP-MESH 的功能之一，允许节点自动扫描/选择/连接/重新连接到其他节点和路由器。此功能允许 ESP-MESH 网络具有很高的自主性，可适应变化的动态网络拓扑结构和环境。启用自组网功能后，ESP-MESH 网络中的节点能够自主完成以下操作：

- 选择或选举根节点（见*Building a Network* 中的 **自动根节点选择**）
- 选择首选的父节点（见*Building a Network* 中的 **父节点选择**）
- 网络断开时自动重新连接（见*Managing a Network* 中的 **中间父节点失败**）

启用自组网功能后，ESP-MESH 软件栈将内部调用 Wi-Fi API。因此，**在启用自组网功能时，应用层不得调用 Wi-Fi API，否则会干扰 ESP-MESH 的工作**。

开关自组网

应用程序可以在运行时通过调用*esp_mesh_set_self_organized()* 函数，启用或禁用自组网功能。该函数具有以下两个参数：

- `bool enable` 指定启用或禁用自组网功能。
- `bool select_parent` 指定在启用自组网功能时是否应选择新的父节点。根据节点类型和节点当前状态，选择新的父节点具有不同的作用。在禁用自组网功能时，此参数不使用。

禁用自组网

下方代码片段展示了如何禁用自组网功能。

```
//禁用自组网
esp_mesh_set_self_organized(false, false);
```

ESP-MESH 将在禁用自组网时尝试维护节点的当前 Wi-Fi 状态。

- 如果节点先前已连接到其他节点，则将保持连接。

- 如果节点先前已断开连接并且正在扫描父节点或路由器，则将停止扫描。
- 如果节点以前尝试重新连接到父节点或路由器，则将停止重新连接。

启用自组网

ESP-MESH 将尝试在启用自组网时保持节点的当前 Wi-Fi 状态。但是，根据节点类型以及是否选择了新的父节点，节点的 Wi-Fi 状态可能会发生变化。下表显示了启用自组网的效果。

是否选择父节点	是否为根结点	作用
N	N	已连接到父节点的节点将保持连接。
		之前扫描父节点的节点将停止扫描。调用 *esp_mesh_connect()* 重新启动。
	Y	已连接到路由器的根节点将保持连接。
		从路由器断开的根结点需调用 *esp_mesh_connect()* 进行重连。
Y	N	没有父节点的节点将自动选择首选父节点并连接。
		已连接到父节点的节点将断开连接，重新选择首选父节点并进行重连。
	Y	根结点在连接至父节点前必须放弃“根结点”的角色。因此，根节点将断开与路由器和所有子节点的连接，选择首选父节点并进行连接。

下方代码片段展示了如何启用自组网功能。

```
//启用自组网，并选择一个新的父节点
esp_mesh_set_self_organized(true, true);

...

//启用自组网并手动重新连接
esp_mesh_set_self_organized(true, false);
esp_mesh_connect();
```

调用 Wi-Fi API

在有些情况下，应用程序可能希望在使用 ESP-MESH 期间调用 Wi-Fi API。例如，应用程序可能需要手动扫描邻近的接入点 (AP)。**但在应用程序调用任何 Wi-Fi API 之前，必须先禁用自组网**。否则，ESP-MESH 软件栈可能会同时调用 Wi-Fi API，进而影响应用程序的正常调用。

应用程序不应在 *esp_mesh_set_self_organized()* 之间调用 Wi-Fi API。下方代码片段展示了应用程序如何在 ESP-MESH 运行期间安全地调用 *esp_wifi_scan_start()*。

```
//禁用自组网
esp_mesh_set_self_organized(0, 0);

//停止任何正在进行的扫描
esp_wifi_scan_stop();
//手动启动扫描运行完成时自动停止
esp_wifi_scan_start();

//进程扫描结果

...

//如果仍为连接状态，则重新启用自组网
esp_mesh_set_self_organized(1, 0);

...

//如果不为根节点且未连接，则重新启用自组网
esp_mesh_set_self_organized(1, 1);

...

//如果为根节点且未连接，则重新启用
esp_mesh_set_self_organized(1, 0);  //不选择新的父节点
esp_mesh_connect();                 //手动重新连接到路由器
```

应用实例

ESP-IDF 包含以下 ESP-MESH 示例项目：

内部通信示例 展示了如何搭建 ESP-MESH 网络，并让根节点向网络中的每个节点发送数据包。

手动连网示例 展示了如何在禁用自组网功能的情况下使用 ESP-MESH。此示例展示了如何对节点进行编程，以手动扫描潜在父节点的列表，并根据自定义标准选择父节点。

API 参考

Header File

- esp_wifi/include/esp_mesh.h

Functions

esp_err_t **esp_mesh_init**(void)

Mesh initialization.

- Check whether Wi-Fi is started.
- Initialize mesh global variables with default values.

Attention This API shall be called after Wi-Fi is started.

Return

- ESP_OK
- ESP_FAIL

esp_err_t **esp_mesh_deinit**(void)

Mesh de-initialization.

- Release resources and stop the mesh

Return

- ESP_OK
- ESP_FAIL

esp_err_t **esp_mesh_start**(void)

Start mesh.

- Initialize mesh IE.
- Start mesh network management service.
- Create TX and RX queues according to the configuration.
- Register mesh packets receive callback.

Attention This API shall be called after mesh initialization and configuration.

Return

- ESP_OK
- ESP_FAIL
- ESP_ERR_MESH_NOT_INIT
- ESP_ERR_MESH_NOT_CONFIG
- ESP_ERR_MESH_NO_MEMORY

esp_err_t **esp_mesh_stop**(void)

Stop mesh.

- Deinitialize mesh IE.
- Disconnect with current parent.
- Disassociate all currently associated children.
- Stop mesh network management service.
- Unregister mesh packets receive callback.
- Delete TX and RX queues.
- Release resources.
- Restore Wi-Fi softAP to default settings if Wi-Fi dual mode is enabled.

Return

- ESP_OK
- ESP_FAIL

esp_err_t **esp_mesh_send**(**const** *mesh_addr_t* *to, **const** *mesh_data_t* *data, int *flag*, **const** *mesh_opt_t* *opt*[], int *opt_count*)

Send a packet over the mesh network.

- Send a packet to any device in the mesh network.
- Send a packet to external IP network.

Attention This API is not reentrant.

Return

- ESP_OK
- ESP_FAIL
- ESP_ERR_MESH_ARGUMENT
- ESP_ERR_MESH_NOT_START
- ESP_ERR_MESH_DISCONNECTED
- ESP_ERR_MESH_OPT_UNKNOWN
- ESP_ERR_MESH_EXCEED_MTU
- ESP_ERR_MESH_NO_MEMORY
- ESP_ERR_MESH_TIMEOUT
- ESP_ERR_MESH_QUEUE_FULL

- ESP_ERR_MESH_NO_ROUTE_FOUND
- ESP_ERR_MESH_DISCARD

Parameters

- `to`: the address of the final destination of the packet
  - If the packet is to the root, set this parameter to NULL.
  - If the packet is to an external IP network, set this parameter to the IPv4:PORT combination. This packet will be delivered to the root firstly, then the root will forward this packet to the final IP server address.
- `data`: pointer to a sending mesh packet
  - Field size should not exceed MESH_MPS. Note that the size of one mesh packet should not exceed MESH_MTU.
  - Field proto should be set to data protocol in use (default is MESH_PROTO_BIN for binary).
  - Field tos should be set to transmission tos (type of service) in use (default is MESH_TOS_P2P for point-to-point reliable).
- `flag`: bitmap for data sent
  - Speed up the route search
    * If the packet is to the root and "to" parameter is NULL, set this parameter to 0.
    * If the packet is to an internal device, MESH_DATA_P2P should be set.
    * If the packet is to the root ("to" parameter isn't NULL) or to external IP network, MESH_DATA_TODS should be set.
    * If the packet is from the root to an internal device, MESH_DATA_FROMDS should be set.
  - Specify whether this API is block or non-block, block by default
    * If needs non-block, MESH_DATA_NONBLOCK should be set.
  - In the situation of the root change, MESH_DATA_DROP identifies this packet can be dropped by the new root for upstream data to external IP network, we try our best to avoid data loss caused by the root change, but there is a risk that the new root is running out of memory because most of memory is occupied by the pending data which isn't read out in time by esp_mesh_recv_toDS().

    Generally, we suggest esp_mesh_recv_toDS() is called after a connection with IP network is created. Thus data outgoing to external IP network via socket is just from reading esp_mesh_recv_toDS() which avoids unnecessary memory copy.
- `opt`: options

  - In case of sending a packet to a certain group, MESH_OPT_SEND_GROUP is a good choice. In this option, the value field should be set to the target receiver addresses in this group.
  - Root sends a packet to an internal device, this packet is from external IP network in case the receiver device responds this packet, MESH_OPT_RECV_DS_ADDR is required to attach the target DS address.

- `opt_count`: option count
  - Currently, this API only takes one option, so opt_count is only supported to be 1.

esp_err_t **esp_mesh_recv**(*mesh_addr_t* **from*, *mesh_data_t* **data*, int *timeout_ms*, int **flag*, *mesh_opt_t opt*[], int *opt_count*)

Receive a packet targeted to self over the mesh network.

flag could be MESH_DATA_FROMDS or MESH_DATA_TODS.

Attention Mesh RX queue should be checked regularly to avoid running out of memory.

- Use esp_mesh_get_rx_pending() to check the number of packets available in the queue waiting to be received by applications.

Return

- ESP_OK
- ESP_ERR_MESH_ARGUMENT
- ESP_ERR_MESH_NOT_START
- ESP_ERR_MESH_TIMEOUT
- ESP_ERR_MESH_DISCARD

Parameters

- `from`: the address of the original source of the packet
- `data`: pointer to the received mesh packet
  - Field proto is the data protocol in use. Should follow it to parse the received data.
  - Field tos is the transmission tos (type of service) in use.
- `timeout_ms`: wait time if a packet isn't immediately available (0:no wait, portMAX_DELAY:wait forever)
- `flag`: bitmap for data received
  - MESH_DATA_FROMDS represents data from external IP network
  - MESH_DATA_TODS represents data directed upward within the mesh network

Parameters

- `opt`: options desired to receive
  - MESH_OPT_RECV_DS_ADDR attaches the DS address
- `opt_count`: option count desired to receive
  - Currently, this API only takes one option, so opt_count is only supported to be 1.

esp_err_t **esp_mesh_recv_toDS**(*mesh_addr_t *from*, *mesh_addr_t *to*, *mesh_data_t *data*, int *timeout_ms*, int *flag*, *mesh_opt_t opt*[], int *opt_count*)

Receive a packet targeted to external IP network.

- Root uses this API to receive packets destined to external IP network
- Root forwards the received packets to the final destination via socket.
- If no socket connection is ready to send out the received packets and this esp_mesh_recv_toDS() hasn' t been called by applications, packets from the whole mesh network will be pending in toDS queue.

Use esp_mesh_get_rx_pending() to check the number of packets available in the queue waiting to be received by applications in case of running out of memory in the root.

Using esp_mesh_set_xon_qsize() users may configure the RX queue size, default:32. If this size is too large, and esp_mesh_recv_toDS() isn' t called in time, there is a risk that a great deal of memory is occupied by the pending packets. If this size is too small, it will impact the efficiency on upstream. How to decide this value depends on the specific application scenarios.

flag could be MESH_DATA_TODS.

Attention This API is only called by the root.

Return

- ESP_OK
- ESP_ERR_MESH_ARGUMENT
- ESP_ERR_MESH_NOT_START
- ESP_ERR_MESH_TIMEOUT
- ESP_ERR_MESH_DISCARD

Parameters

- `from`: the address of the original source of the packet
- `to`: the address contains remote IP address and port (IPv4:PORT)
- `data`: pointer to the received packet
  - Contain the protocol and applications should follow it to parse the data.

- `timeout_ms`: wait time if a packet isn't immediately available (0:no wait, portMAX_DELAY:wait forever)
- `flag`: bitmap for data received
  - MESH_DATA_TODS represents the received data target to external IP network. Root shall forward this data to external IP network via the association with router.

Parameters

- `opt`: options desired to receive
- `opt_count`: option count desired to receive

esp_err_t **esp_mesh_set_config(const** *mesh_cfg_t* ***config)**

Set mesh stack configuration.

- Use MESH_INIT_CONFIG_DEFAULT() to initialize the default values, mesh IE is encrypted by default.
- Mesh network is established on a fixed channel (1-14).
- Mesh event callback is mandatory.
- Mesh ID is an identifier of an MBSS. Nodes with the same mesh ID can communicate with each other.
- Regarding to the router configuration, if the router is hidden, BSSID field is mandatory.

If BSSID field isn't set and there exists more than one router with same SSID, there is a risk that more roots than one connected with different BSSID will appear. It means more than one mesh network is established with the same mesh ID.

Root conflict function could eliminate redundant roots connected with the same BSSID, but couldn't handle roots connected with different BSSID. Because users might have such requirements of setting up routers with same SSID for the future replacement. But in that case, if the above situations happen, please make sure applications implement forward functions on the root to guarantee devices in different mesh networks can communicate with each other. max_connection of mesh softAP is limited by the max number of Wi-Fi softAP supported (max:10).

Attention This API shall be called before mesh is started after mesh is initialized.

Return

- ESP_OK
- ESP_ERR_MESH_ARGUMENT
- ESP_ERR_MESH_NOT_ALLOWED

Parameters

- `config`: pointer to mesh stack configuration

esp_err_t **esp_mesh_get_config**(*mesh_cfg_t* **config*)

Get mesh stack configuration.

Return

- ESP_OK
- ESP_ERR_MESH_ARGUMENT

Parameters

- `config`: pointer to mesh stack configuration

esp_err_t **esp_mesh_set_router**(**const** *mesh_router_t* **router*)

Get router configuration.

Attention This API is used to dynamically modify the router configuration after mesh is configured.

Return

- ESP_OK
- ESP_ERR_MESH_ARGUMENT

Parameters

- `router`: pointer to router configuration

esp_err_t **esp_mesh_get_router**(*mesh_router_t* **router*)

Get router configuration.

Return

- ESP_OK
- ESP_ERR_MESH_ARGUMENT

Parameters

- `router`: pointer to router configuration

esp_err_t **esp_mesh_set_id**(**const** *mesh_addr_t* **id*)

Set mesh network ID.

Attention This API is used to dynamically modify the mesh network ID.

Return

- ESP_OK
- ESP_ERR_MESH_ARGUMENT: invalid argument

Parameters

- `id`: pointer to mesh network ID

esp_err_t **esp_mesh_get_id**(*mesh_addr_t* **id*)

Get mesh network ID.

Return

- ESP_OK
- ESP_ERR_MESH_ARGUMENT

Parameters

- `id`: pointer to mesh network ID

esp_err_t **esp_mesh_set_type**(*mesh_type_t type*)

Designate device type over the mesh network.

- MESH_ROOT: designates the root node for a mesh network
- MESH_LEAF: designates a device as a standalone Wi-Fi station

Return

- ESP_OK
- ESP_ERR_MESH_NOT_ALLOWED

Parameters

- `type`: device type

mesh_type_t **esp_mesh_get_type**(void)

Get device type over mesh network.

Attention This API shall be called after having received the event MESH_EVENT_PARENT_CONNECTED.

Return mesh type

esp_err_t **esp_mesh_set_max_layer**(int *max_layer*)

Set network max layer value (max:25, default:25)

- Network max layer limits the max hop count.

Attention This API shall be called before mesh is started.

Return

- ESP_OK
- ESP_ERR_MESH_ARGUMENT

- ESP_ERR_MESH_NOT_ALLOWED

Parameters

- `max_layer`: max layer value

int **esp_mesh_get_max_layer**(void)

Get max layer value.

Return max layer value

esp_err_t **esp_mesh_set_ap_password**(**const** uint8_t **pwd*, int *len*)

Set mesh softAP password.

Attention This API shall be called before mesh is started.

Return

- ESP_OK
- ESP_ERR_MESH_ARGUMENT
- ESP_ERR_MESH_NOT_ALLOWED

Parameters

- `pwd`: pointer to the password
- `len`: password length

esp_err_t **esp_mesh_set_ap_authmode**(*wifi_auth_mode_t authmode*)

Set mesh softAP authentication mode.

Attention This API shall be called before mesh is started.

Return

- ESP_OK
- ESP_ERR_MESH_ARGUMENT
- ESP_ERR_MESH_NOT_ALLOWED

Parameters

- `authmode`: authentication mode

wifi_auth_mode_t **esp_mesh_get_ap_authmode**(void)

Get mesh softAP authentication mode.

Return authentication mode

esp_err_t **esp_mesh_set_ap_connections**(int *connections*)

Set mesh softAP max connection value.

Attention This API shall be called before mesh is started.

Return

- ESP_OK
- ESP_ERR_MESH_ARGUMENT

Parameters

- `connections`: the number of max connections

int **esp_mesh_get_ap_connections**(void)

Get mesh softAP max connection configuration.

Return the number of max connections

int **esp_mesh_get_layer**(void)

Get current layer value over the mesh network.

Attention This API shall be called after having received the event MESH_EVENT_PARENT_CONNECTED.

Return layer value

esp_err_t **esp_mesh_get_parent_bssid**(*mesh_addr_t* **bssid*)

Get the parent BSSID.

Attention This API shall be called after having received the event MESH_EVENT_PARENT_CONNECTED.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `bssid`: pointer to parent BSSID

bool **esp_mesh_is_root**(void)

Return whether the device is the root node of the network.

Return true/false

esp_err_t **esp_mesh_set_self_organized**(bool *enable*, bool *select_parent*)

Enable/disable self-organized networking.

- Self-organized networking has three main functions: select the root node; find a preferred parent; initiate reconnection if a disconnection is detected.
- Self-organized networking is enabled by default.
- If self-organized is disabled, users should set a parent for the device via esp_mesh_set_parent().

Attention This API is used to dynamically modify whether to enable the self organizing.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `enable`: enable or disable self-organized networking
- `select_parent`: Only valid when self-organized networking is enabled.
  - if select_parent is set to true, the root will give up its mesh root status and search for a new parent like other non-root devices.

bool **esp_mesh_get_self_organized**(void)

Return whether enable self-organized networking or not.

Return true/false

esp_err_t **esp_mesh_waive_root**(**const** *mesh_vote_t* **vote*, int *reason*)

Cause the root device to give up (waive) its mesh root status.

- A device is elected root primarily based on RSSI from the external router.
- If external router conditions change, users can call this API to perform a root switch.
- In this API, users could specify a desired root address to replace itself or specify an attempts value to ask current root to initiate a new round of voting. During the voting, a better root candidate would be expected to find to replace the current one.
- If no desired root candidate, the vote will try a specified number of attempts (at least 15). If no better root candidate is found, keep the current one. If a better candidate is found, the new better one will send a root switch request to the current root, current root will respond with a root switch acknowledgment.
- After that, the new candidate will connect to the router to be a new root, the previous root will disconnect with the router and choose another parent instead.

Root switch is completed with minimal disruption to the whole mesh network.

Attention This API is only called by the root.

Return

- ESP_OK
- ESP_ERR_MESH_QUEUE_FULL
- ESP_ERR_MESH_DISCARD
- ESP_FAIL

Parameters

- `vote`: vote configuration
  - If this parameter is set NULL, the vote will perform the default 15 times.
  - Field percentage threshold is 0.9 by default.
  - Field is_rc_specified shall be false.
  - Field attempts shall be at least 15 times.
- `reason`: only accept MESH_VOTE_REASON_ROOT_INITIATED for now

esp_err_t **esp_mesh_set_vote_percentage**(float *percentage*)

Set vote percentage threshold for approval of being a root.

- During the networking, only obtaining vote percentage reaches this threshold, the device could be a root.

Attention This API shall be called before mesh is started.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `percentage`: vote percentage threshold

float **esp_mesh_get_vote_percentage**(void)

Get vote percentage threshold for approval of being a root.

Return percentage threshold

esp_err_t **esp_mesh_set_ap_assoc_expire**(int *seconds*)

Set mesh softAP associate expired time (default:10 seconds)

- If mesh softAP hasn't received any data from an associated child within this time, mesh softAP will take this child inactive and disassociate it.
- If mesh softAP is encrypted, this value should be set a greater value, such as 30 seconds.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `seconds`: the expired time

int **esp_mesh_get_ap_assoc_expire**(void)

Get mesh softAP associate expired time.

Return seconds

int **esp_mesh_get_total_node_num**(void)

Get total number of devices in current network (including the root)

Attention The returned value might be incorrect when the network is changing.

Return total number of devices (including the root)

int **esp_mesh_get_routing_table_size**(void)

Get the number of devices in this device's sub-network (including self)

Return the number of devices over this device's sub-network (including self)

esp_err_t **esp_mesh_get_routing_table**(*mesh_addr_t* *mac, int *len*, int *size)

Get routing table of this device's sub-network (including itself)

Return

- ESP_OK
- ESP_ERR_MESH_ARGUMENT

Parameters

- `mac`: pointer to routing table
- `len`: routing table size(in bytes)
- `size`: pointer to the number of devices in routing table (including itself)

esp_err_t **esp_mesh_post_toDS_state**(bool *reachable*)

Post the toDS state to the mesh stack.

Attention This API is only for the root.

Return

- ESP_OK

- ESP_FAIL

Parameters

- `reachable`: this state represents whether the root is able to access external IP network

esp_err_t **esp_mesh_get_tx_pending**(*mesh_tx_pending_t* **pending*)

Return the number of packets pending in the queue waiting to be sent by the mesh stack.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `pending`: pointer to the TX pending

esp_err_t **esp_mesh_get_rx_pending**(*mesh_rx_pending_t* **pending*)

Return the number of packets available in the queue waiting to be received by applications.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `pending`: pointer to the RX pending

int **esp_mesh_available_txupQ_num**(**const** *mesh_addr_t* **addr*, uint32_t **xseqno_in*)

Return the number of packets could be accepted from the specified address.

Return the number of upQ for a certain address

Parameters

- `addr`: self address or an associate children address
- `xseqno_in`: sequence number of the last received packet from the specified address

esp_err_t **esp_mesh_set_xon_qsize**(int *qsize*)

Set the number of queue.

Attention This API shall be called before mesh is started.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `qsize`: default:32 (min:16)

int **esp_mesh_get_xon_qsize**(void)

Get queue size.

Return the number of queue

esp_err_t **esp_mesh_allow_root_conflicts**(bool *allowed*)

Set whether allow more than one root existing in one network.

Return

- ESP_OK
- ESP_WIFI_ERR_NOT_INIT
- ESP_WIFI_ERR_NOT_START

Parameters

- `allowed`: allow or not

bool **esp_mesh_is_root_conflicts_allowed**(void)

Check whether allow more than one root to exist in one network.

Return true/false

esp_err_t **esp_mesh_set_group_id**(**const** *mesh_addr_t* **addr*, int *num*)

Set group ID addresses.

Return

- ESP_OK
- ESP_MESH_ERR_ARGUMENT

Parameters

- `addr`: pointer to new group ID addresses
- `num`: the number of group ID addresses

esp_err_t **esp_mesh_delete_group_id**(**const** *mesh_addr_t* **addr*, int *num*)

Delete group ID addresses.

Return

- ESP_OK
- ESP_MESH_ERR_ARGUMENT

Parameters

- `addr`: pointer to deleted group ID address
- `num`: the number of group ID addresses

int **esp_mesh_get_group_num**(void)

Get the number of group ID addresses.

Return the number of group ID addresses

esp_err_t **esp_mesh_get_group_list**(*mesh_addr_t* **addr*, int *num*)

Get group ID addresses.

Return

- ESP_OK
- ESP_MESH_ERR_ARGUMENT

Parameters

- `addr`: pointer to group ID addresses
- `num`: the number of group ID addresses

bool **esp_mesh_is_my_group**(**const** *mesh_addr_t* **addr*)

Check whether the specified group address is my group.

Return true/false

esp_err_t **esp_mesh_set_capacity_num**(int *num*)

Set mesh network capacity (max:1000, default:300)

Attention This API shall be called before mesh is started.

Return

- ESP_OK
- ESP_ERR_MESH_NOT_ALLOWED
- ESP_MESH_ERR_ARGUMENT

Parameters

- `num`: mesh network capacity

int **esp_mesh_get_capacity_num**(void)

Get mesh network capacity.

Return mesh network capacity

esp_err_t **esp_mesh_set_ie_crypto_funcs**(**const** mesh_crypto_funcs_t *_crypto_funcs*)

Set mesh IE crypto functions.

Attention This API can be called at any time after mesh is initialized.

Return

- ESP_OK

Parameters

- `crypto_funcs`: crypto functions for mesh IE
  - If crypto_funcs is set to NULL, mesh IE is no longer encrypted.

esp_err_t **esp_mesh_set_ie_crypto_key**(**const** char *_key_, int *_len_)

Set mesh IE crypto key.

Attention This API can be called at any time after mesh is initialized.

Return

- ESP_OK
- ESP_MESH_ERR_ARGUMENT

Parameters

- `key`: ASCII crypto key
- `len`: length in bytes, range:8~64

esp_err_t **esp_mesh_get_ie_crypto_key**(char *_key_, int *_len_)

Get mesh IE crypto key.

Return

- ESP_OK
- ESP_MESH_ERR_ARGUMENT

Parameters

- `key`: ASCII crypto key
- `len`: length in bytes, range:8~64

esp_err_t **esp_mesh_set_root_healing_delay**(int *_delay_ms_)

Set delay time before starting root healing.

Return

- ESP_OK

Parameters

- `delay_ms`: delay time in milliseconds

int **esp_mesh_get_root_healing_delay**(void)

Get delay time before network starts root healing.

Return delay time in milliseconds

esp_err_t **esp_mesh_fix_root**(bool *enable*)

Enable network Fixed Root Setting.

- Enabling fixed root disables automatic election of the root node via voting.
- All devices in the network shall use the same Fixed Root Setting (enabled or disabled).
- If Fixed Root is enabled, users should make sure a root node is designated for the network.

Return

- ESP_OK

Parameters

- `enable`: enable or not

bool **esp_mesh_is_root_fixed**(void)

Check whether network Fixed Root Setting is enabled.

- Enable/disable network Fixed Root Setting by API esp_mesh_fix_root().
- Network Fixed Root Setting also changes with the "flag" value in parent networking IE.

Return true/false

esp_err_t **esp_mesh_set_parent**(const *wifi_config_t* *parent, const *mesh_addr_t* *parent_mesh_id, *mesh_type_t* *my_type*, int *my_layer*)

Set a specified parent for the device.

Attention This API can be called at any time after mesh is configured.

Return

- ESP_OK
- ESP_ERR_ARGUMENT
- ESP_ERR_MESH_NOT_CONFIG

Parameters

- `parent`: parent configuration, the SSID and the channel of the parent are mandatory.

  - If the BSSID is set, make sure that the SSID and BSSID represent the same parent, otherwise the device will never find this specified parent.

- `parent_mesh_id`: parent mesh ID,
  - If this value is not set, the original mesh ID is used.
- `my_type`: mesh type
  - If the parent set for the device is the same as the router in the network configuration, then my_type shall set MESH_ROOT and my_layer shall set MESH_ROOT_LAYER.
- `my_layer`: mesh layer
  - my_layer of the device may change after joining the network.
  - If my_type is set MESH_NODE, my_layer shall be greater than MESH_ROOT_LAYER.
  - If my_type is set MESH_LEAF, the device becomes a standalone Wi-Fi station and no longer has the ability to extend the network.

esp_err_t **esp_mesh_scan_get_ap_ie_len**(int **len*)

Get mesh networking IE length of one AP.

Return

- ESP_OK
- ESP_ERR_WIFI_NOT_INIT
- ESP_ERR_WIFI_ARG
- ESP_ERR_WIFI_FAIL

Parameters

- `len`: mesh networking IE length

esp_err_t **esp_mesh_scan_get_ap_record**(*wifi_ap_record_t* **ap_record*, void **buffer*)

Get AP record.

Attention Different from esp_wifi_scan_get_ap_records(), this API only gets one of APs scanned each time. See "manual_networking" example.

Return

- ESP_OK
- ESP_ERR_WIFI_NOT_INIT
- ESP_ERR_WIFI_ARG
- ESP_ERR_WIFI_FAIL

Parameters

- `ap_record`: pointer to one AP record
- `buffer`: pointer to the mesh networking IE of this AP

esp_err_t **esp_mesh_flush_upstream_packets**(void)

Flush upstream packets pending in to_parent queue and to_parent_p2p queue.

Return

- ESP_OK

esp_err_t **esp_mesh_get_subnet_nodes_num**(**const** *mesh_addr_t* **child_mac*, int **nodes_num*)

Get the number of nodes in the subnet of a specific child.

Return

- ESP_OK
- ESP_ERR_MESH_NOT_START
- ESP_ERR_MESH_ARGUMENT

Parameters

- `child_mac`: an associated child address of this device
- `nodes_num`: pointer to the number of nodes in the subnet of a specific child

esp_err_t **esp_mesh_get_subnet_nodes_list**(**const** *mesh_addr_t* **child_mac*, *mesh_addr_t* **nodes*, int *nodes_num*)

Get nodes in the subnet of a specific child.

Return

- ESP_OK
- ESP_ERR_MESH_NOT_START
- ESP_ERR_MESH_ARGUMENT

Parameters

- `child_mac`: an associated child address of this device
- `nodes`: pointer to nodes in the subnet of a specific child
- `nodes_num`: the number of nodes in the subnet of a specific child

esp_err_t **esp_mesh_disconnect**(void)

Disconnect from current parent.

Return

- ESP_OK

esp_err_t **esp_mesh_connect**(void)

Connect to current parent.

Return

- ESP_OK

esp_err_t **esp_mesh_flush_scan_result**(void)

Flush scan result.

Return

- ESP_OK

esp_err_t **esp_mesh_switch_channel**(**const** uint8_t **new_bssid*, int *csa_newchan*, int *csa_count*)

Cause the root device to add Channel Switch Announcement Element (CSA IE) to beacon.

- Set the new channel
- Set how many beacons with CSA IE will be sent before changing a new channel
- Enable the channel switch function

Attention This API is only called by the root.

Return

- ESP_OK

Parameters

- `new_bssid`: the new router BSSID if the router changes
- `csa_newchan`: the new channel number to which the whole network is moving
- `csa_count`: channel switch period(beacon count), unit is based on beacon interval of its softAP, the default value is 15.

esp_err_t **esp_mesh_get_router_bssid**(uint8_t **router_bssid*)

Get the router BSSID.

Return

- ESP_OK
- ESP_ERR_WIFI_NOT_INIT
- ESP_ERR_WIFI_ARG

Parameters

- `router_bssid`: pointer to the router BSSID

int64_t **esp_mesh_get_tsf_time**(void)

Get the TSF time.

Return the TSF time

Unions

union mesh_addr_t

#include <esp_mesh.h> Mesh address.

Public Members

uint8_t **addr**[6]

mac address

mip_t **mip**

mip address

union mesh_event_info_t

#include <esp_mesh.h> Mesh event information.

Public Members

mesh_event_channel_switch_t **channel_switch**

channel switch

mesh_event_child_connected_t **child_connected**

child connected

mesh_event_child_disconnected_t **child_disconnected**

child disconnected

mesh_event_routing_table_change_t **routing_table**

routing table change

mesh_event_connected_t **connected**

parent connected

mesh_event_disconnected_t **disconnected**

parent disconnected

mesh_event_no_parent_found_t **no_parent**

no parent found

mesh_event_layer_change_t **layer_change**

layer change

mesh_event_toDS_state_t **toDS_state**

toDS state, devices shall check this state firstly before trying to send packets to external IP network. This state indicates right now whether the root is capable of sending packets out. If not, devices had better to wait until this state changes to be MESH_TODS_REACHABLE.

mesh_event_vote_started_t **vote_started**

vote started

mesh_event_root_address_t **root_addr**

root address

mesh_event_root_switch_req_t **switch_req**

root switch request

mesh_event_root_conflict_t **root_conflict**

other powerful root

mesh_event_root_fixed_t **root_fixed**

fixed root

mesh_event_scan_done_t **scan_done**

scan done

mesh_event_network_state_t **network_state**

network state, such as whether current mesh network has a root.

mesh_event_find_network_t **find_network**

network found that can join

mesh_event_router_switch_t **router_switch**

new router information

union mesh_rc_config_t

#include <esp_mesh.h> Vote address configuration.

Public Members

int **attempts**

max vote attempts before a new root is elected automatically by mesh network. (min:15, 15 by default)

mesh_addr_t **rc_addr**

a new root address specified by users for API esp_mesh_waive_root()

Structures

struct mip_t

IP address and port.

Public Members

ip4_addr_t **ip4**

IP address

uint16_t **port**

port

struct mesh_event_channel_switch_t

Channel switch information.

Public Members

uint8_t **channel**

new channel

struct mesh_event_connected_t

Parent connected information.

Public Members

wifi_event_sta_connected_t **connected**

parent information, same as Wi-Fi event SYSTEM_EVENT_STA_CONNECTED does

uint8_t **self_layer**

layer

struct mesh_event_no_parent_found_t

No parent found information.

Public Members

int **scan_times**

scan times being through

struct mesh_event_layer_change_t

Layer change information.

Public Members

uint8_t **new_layer**

new layer

struct mesh_event_vote_started_t

vote started information

Public Members

int **reason**

vote reason, vote could be initiated by children or by the root itself

int **attempts**

max vote attempts before stopped

mesh_addr_t **rc_addr**

root address specified by users via API esp_mesh_waive_root()

struct mesh_event_find_network_t

find a mesh network that this device can join

Public Members

uint8_t **channel**

channel number of the new found network

uint8_t **router_bssid**[6]

router BSSID

struct mesh_event_root_switch_req_t

Root switch request information.

Public Members

int **reason**

root switch reason, generally root switch is initialized by users via API esp_mesh_waive_root()

mesh_addr_t **rc_addr**

the address of root switch requester

struct mesh_event_root_conflict_t

Other powerful root address.

Public Members

int8_t **rssi**
: rssi with router

uint16_t **capacity**
: the number of devices in current network

uint8_t **addr**[6]
: other powerful root address

struct mesh_event_routing_table_change_t
: Routing table change.

Public Members

uint16_t **rt_size_new**
: the new value

uint16_t **rt_size_change**
: the changed value

struct mesh_event_root_fixed_t
: Root fixed.

Public Members

bool **is_fixed**
: status

struct mesh_event_scan_done_t
: Scan done event information.

Public Members

uint8_t **number**
: the number of APs scanned

struct mesh_event_network_state_t
: Network state information.

Public Members

bool **is_rootless**
: whether current mesh network has a root

struct mesh_opt_t

Mesh option.

Public Members

uint8_t **type**

option type

uint16_t **len**

option length

uint8_t ***val**

option value

struct mesh_data_t

Mesh data for esp_mesh_send() and esp_mesh_recv()

Public Members

uint8_t ***data**

data

uint16_t **size**

data size

mesh_proto_t **proto**

data protocol

mesh_tos_t **tos**

data type of service

struct mesh_router_t

Router configuration.

Public Members

uint8_t **ssid**[32]

SSID

uint8_t **ssid_len**

length of SSID

uint8_t **bssid**[6]

BSSID, if this value is specified, users should also specify "allow_router_switch".

uint8_t **password**[64]

password

bool **allow_router_switch**

if the BSSID is specified and this value is also set, when the router of this specified BSSID fails to be found after "fail" (mesh_attempts_t) times, the whole network is allowed to switch to another router with the same SSID. The new router might also be on a different channel. The default value is false. There is a risk that if the password is different between the new switched router and the previous one, the mesh network could be established but the root will never connect to the new switched router.

struct mesh_ap_cfg_t

Mesh softAP configuration.

Public Members

uint8_t **password**[64]

mesh softAP password

uint8_t **max_connection**

max number of stations allowed to connect in, max 10

struct mesh_cfg_t

Mesh initialization configuration.

Public Members

uint8_t **channel**

channel, the mesh network on

bool **allow_channel_switch**

if this value is set, when "fail" (mesh_attempts_t) times is reached, device will change to a full channel scan for a network that could join. The default value is false.

mesh_addr_t **mesh_id**

mesh network identification

mesh_router_t **router**

router configuration

mesh_ap_cfg_t **mesh_ap**

mesh softAP configuration

const mesh_crypto_funcs_t ***crypto_funcs**

crypto functions

struct mesh_vote_t

Vote.

Public Members

float **percentage**

vote percentage threshold for approval of being a root

bool **is_rc_specified**

if true, rc_addr shall be specified (Unimplemented). if false, attempts value shall be specified to make network start root election.

mesh_rc_config_t **config**

vote address configuration

struct mesh_tx_pending_t

The number of packets pending in the queue waiting to be sent by the mesh stack.

Public Members

int **to_parent**

to parent queue

int **to_parent_p2p**

to parent (P2P) queue

int **to_child**

to child queue

int **to_child_p2p**

to child (P2P) queue

int **mgmt**

management queue

int **broadcast**

broadcast and multicast queue

struct mesh_rx_pending_t

The number of packets available in the queue waiting to be received by applications.

Public Members

int **toDS**

to external DS

int **toSelf**

to self

Macros

MESH_ROOT_LAYER
: root layer value

MESH_MTU
: max transmit unit(in bytes)

MESH_MPS
: max payload size(in bytes)

ESP_ERR_MESH_WIFI_NOT_START
: Mesh error code definition.

  Wi-Fi isn’t started

ESP_ERR_MESH_NOT_INIT
: mesh isn’t initialized

ESP_ERR_MESH_NOT_CONFIG
: mesh isn’t configured

ESP_ERR_MESH_NOT_START
: mesh isn’t started

ESP_ERR_MESH_NOT_SUPPORT
: not supported yet

ESP_ERR_MESH_NOT_ALLOWED
: operation is not allowed

ESP_ERR_MESH_NO_MEMORY
: out of memory

ESP_ERR_MESH_ARGUMENT
: illegal argument

ESP_ERR_MESH_EXCEED_MTU
: packet size exceeds MTU

ESP_ERR_MESH_TIMEOUT
: timeout

ESP_ERR_MESH_DISCONNECTED
: disconnected with parent on station interface

ESP_ERR_MESH_QUEUE_FAIL
: queue fail

ESP_ERR_MESH_QUEUE_FULL
: queue full

ESP_ERR_MESH_NO_PARENT_FOUND
: no parent found to join the mesh network

ESP_ERR_MESH_NO_ROUTE_FOUND
: no route found to forward the packet

ESP_ERR_MESH_OPTION_NULL
: no option found

ESP_ERR_MESH_OPTION_UNKNOWN
: unknown option

ESP_ERR_MESH_XON_NO_WINDOW
: no window for software flow control on upstream

ESP_ERR_MESH_INTERFACE
: low-level Wi-Fi interface error

ESP_ERR_MESH_DISCARD_DUPLICATE
: discard the packet due to the duplicate sequence number

ESP_ERR_MESH_DISCARD
: discard the packet

ESP_ERR_MESH_VOTING
: vote in progress

MESH_DATA_ENC
: Flags bitmap for esp_mesh_send() and esp_mesh_recv()

  data encrypted (Unimplemented)

MESH_DATA_P2P
: point-to-point delivery over the mesh network

MESH_DATA_FROMDS
: receive from external IP network

MESH_DATA_TODS
: identify this packet is target to external IP network

MESH_DATA_NONBLOCK
: esp_mesh_send() non-block

MESH_DATA_DROP
: in the situation of the root having been changed, identify this packet can be dropped by new root

MESH_DATA_GROUP
: identify this packet is target to a group address

MESH_OPT_SEND_GROUP
: Option definitions for esp_mesh_send() and esp_mesh_recv()

data transmission by group; used with esp_mesh_send() and shall have payload

`MESH_OPT_RECV_DS_ADDR`
return a remote IP address; used with esp_mesh_send() and esp_mesh_recv()

`MESH_ASSOC_FLAG_VOTE_IN_PROGRESS`
Flag of mesh networking IE.

vote in progress

`MESH_ASSOC_FLAG_NETWORK_FREE`
no root in current network

`MESH_ASSOC_FLAG_ROOTS_FOUND`
root conflict is found

`MESH_ASSOC_FLAG_ROOT_FIXED`
fixed root

`MESH_INIT_CONFIG_DEFAULT()`

Type Definitions

`typedef` *mesh_addr_t* **mesh_event_root_address_t**
Root address.

`typedef` *wifi_event_sta_disconnected_t* **mesh_event_disconnected_t**
Parent disconnected information.

`typedef` *wifi_event_ap_staconnected_t* **mesh_event_child_connected_t**
Child connected information.

`typedef` *wifi_event_ap_stadisconnected_t* **mesh_event_child_disconnected_t**
Child disconnected information.

`typedef` *wifi_event_sta_connected_t* **mesh_event_router_switch_t**
New router information.

Enumerations

`enum mesh_event_id_t`
Enumerated list of mesh event id.

Values:

`MESH_EVENT_STARTED`
mesh is started

`MESH_EVENT_STOPPED`
mesh is stopped

`MESH_EVENT_CHANNEL_SWITCH`
channel switch

`MESH_EVENT_CHILD_CONNECTED`
a child is connected on softAP interface

`MESH_EVENT_CHILD_DISCONNECTED`
a child is disconnected on softAP interface

`MESH_EVENT_ROUTING_TABLE_ADD`
routing table is changed by adding newly joined children

`MESH_EVENT_ROUTING_TABLE_REMOVE`
routing table is changed by removing leave children

`MESH_EVENT_PARENT_CONNECTED`
parent is connected on station interface

`MESH_EVENT_PARENT_DISCONNECTED`
parent is disconnected on station interface

`MESH_EVENT_NO_PARENT_FOUND`
no parent found

`MESH_EVENT_LAYER_CHANGE`
layer changes over the mesh network

`MESH_EVENT_TODS_STATE`
state represents whether the root is able to access external IP network

`MESH_EVENT_VOTE_STARTED`
the process of voting a new root is started either by children or by the root

`MESH_EVENT_VOTE_STOPPED`
the process of voting a new root is stopped

`MESH_EVENT_ROOT_ADDRESS`
the root address is obtained. It is posted by mesh stack automatically.

`MESH_EVENT_ROOT_SWITCH_REQ`
root switch request sent from a new voted root candidate

`MESH_EVENT_ROOT_SWITCH_ACK`
root switch acknowledgment responds the above request sent from current root

`MESH_EVENT_ROOT_ASKED_YIELD`
the root is asked yield by a more powerful existing root. If self organized is disabled and this device is specified to be a root by users, users should set a new parent for this device. if self organized is enabled, this device will find a new parent by itself, users could ignore this event.

MESH_EVENT_ROOT_FIXED
when devices join a network, if the setting of Fixed Root for one device is different from that of its parent, the device will update the setting the same as its parent' s. Fixed Root Setting of each device is variable as that setting changes of the root.

MESH_EVENT_SCAN_DONE
if self-organized networking is disabled, user can call esp_wifi_scan_start() to trigger this event, and add the corresponding scan done handler in this event.

MESH_EVENT_NETWORK_STATE
network state, such as whether current mesh network has a root.

MESH_EVENT_STOP_RECONNECTION
the root stops reconnecting to the router and non-root devices stop reconnecting to their parents.

MESH_EVENT_FIND_NETWORK
when the channel field in mesh configuration is set to zero, mesh stack will perform a full channel scan to find a mesh network that can join, and return the channel value after finding it.

MESH_EVENT_ROUTER_SWITCH
if users specify BSSID of the router in mesh configuration, when the root connects to another router with the same SSID, this event will be posted and the new router information is attached.

MESH_EVENT_MAX

enum mesh_type_t
Device type.

Values:

MESH_IDLE
hasn' t joined the mesh network yet

MESH_ROOT
the only sink of the mesh network. Has the ability to access external IP network

MESH_NODE
intermediate device. Has the ability to forward packets over the mesh network

MESH_LEAF
has no forwarding ability

enum mesh_proto_t
Protocol of transmitted application data.

Values:

MESH_PROTO_BIN
binary

MESH_PROTO_HTTP
HTTP protocol

MESH_PROTO_JSON
JSON format

MESH_PROTO_MQTT
MQTT protocol

enum mesh_tos_t
For reliable transmission, mesh stack provides three type of services.

Values:

MESH_TOS_P2P
provide P2P (point-to-point) retransmission on mesh stack by default

MESH_TOS_E2E
provide E2E (end-to-end) retransmission on mesh stack (Unimplemented)

MESH_TOS_DEF
no retransmission on mesh stack

enum mesh_vote_reason_t
Vote reason.

Values:

MESH_VOTE_REASON_ROOT_INITIATED = 1
vote is initiated by the root

MESH_VOTE_REASON_CHILD_INITIATED
vote is initiated by children

enum mesh_disconnect_reason_t
Mesh disconnect reason code.

Values:

MESH_REASON_CYCLIC = 100
cyclic is detected

MESH_REASON_PARENT_IDLE
parent is idle

MESH_REASON_LEAF
the connected device is changed to a leaf

MESH_REASON_DIFF_ID
in different mesh ID

MESH_REASON_ROOTS
root conflict is detected

`MESH_REASON_PARENT_STOPPED`
parent has stopped the mesh

`MESH_REASON_SCAN_FAIL`
scan fail

`MESH_REASON_IE_UNKNOWN`
unknown IE

`MESH_REASON_WAIVE_ROOT`
waive root

`MESH_REASON_PARENT_WORSE`
parent with very poor RSSI

`MESH_REASON_EMPTY_PASSWORD`
use an empty password to connect to an encrypted parent

`MESH_REASON_PARENT_UNENCRYPTED`
connect to an unencrypted parent/router

enum mesh_event_toDS_state_t
The reachability of the root to a DS (distribute system)

Values:

`MESH_TODS_UNREACHABLE`
the root isn't able to access external IP network

`MESH_TODS_REACHABLE`
the root is able to access external IP network

本部分的 Wi-Fi API 示例代码存放在 ESP-IDF 示例项目的 wifi 目录下。

ESP-MESH 的示例代码存放在 ESP-IDF 示例项目的 mesh 目录下。

2.2.2 以太网

以太网

[英文]

应用示例

- 以太网基本示例：ethernet/ethernet.
- 以太网 iperf 示例：ethernet/iperf.

以太网驱动程序模型

- esp_eth/include/esp_eth.h

以太网通用接口

- esp_eth/include/esp_eth_com.h

以太网 MAC 接口

- esp_eth/include/esp_eth_mac.h

以太网 PHY 接口

- esp_eth/include/esp_eth_phy.h

以太网 PHY 公共寄存器

- esp_eth/include/eth_phy_regs_struct.h

API 参考 – 驱动程序模型

Header File

- esp_eth/include/esp_eth.h

Functions

esp_err_t **esp_eth_driver_install**(**const** *esp_eth_config_t* **config*, *esp_eth_handle_t* **out_hdl*)

Install Ethernet driver.

Return

- ESP_OK: install esp_eth driver successfully
- ESP_ERR_INVALID_ARG: install esp_eth driver failed because of some invalid argument
- ESP_ERR_NO_MEM: install esp_eth driver failed because there’ s no memory for driver
- ESP_FAIL: install esp_eth driver failed because some other error occurred

Parameters

- `config`: configuration of the Ethernet driver

- `out_hdl`: handle of Ethernet driver

esp_err_t **esp_eth_driver_uninstall**(*esp_eth_handle_t hdl*)

Uninstall Ethernet driver.

Note It' s not recommended to uninstall Ethernet driver unless it won' t get used any more in application code. To uninstall Ethernet driver, you have to make sure, all references to the driver are released. Ethernet driver can only be uninstalled successfully when reference counter equals to one.

Return

- ESP_OK: uninstall esp_eth driver successfully
- ESP_ERR_INVALID_ARG: uninstall esp_eth driver failed because of some invalid argument
- ESP_ERR_INVALID_STATE: uninstall esp_eth driver failed because it has more than one reference
- ESP_FAIL: uninstall esp_eth driver failed because some other error occurred

Parameters

- `hdl`: handle of Ethernet driver

esp_err_t **esp_eth_start**(*esp_eth_handle_t hdl*)

Start Ethernet driver **ONLY** in standalone mode (i.e. without TCP/IP stack)

Note This API will start driver state machine and internal software timer (for checking link status).

Return

- ESP_OK: start esp_eth driver successfully
- ESP_ERR_INVALID_ARG: start esp_eth driver failed because of some invalid argument
- ESP_ERR_INVALID_STATE: start esp_eth driver failed because driver has started already
- ESP_FAIL: start esp_eth driver failed because some other error occurred

Parameters

- `hdl`: handle of Ethernet driver

esp_err_t **esp_eth_stop**(*esp_eth_handle_t hdl*)

Stop Ethernet driver.

Note This function does the oppsite operation of `esp_eth_start`.

Return

- ESP_OK: stop esp_eth driver successfully

- ESP_ERR_INVALID_ARG: stop esp_eth driver failed because of some invalid argument
- ESP_ERR_INVALID_STATE: stop esp_eth driver failed because driver has not started yet
- ESP_FAIL: stop esp_eth driver failed because some other error occurred

Parameters

- `hdl`: handle of Ethernet driver

esp_err_t **esp_eth_update_input_path**(*esp_eth_handle_t hdl*, *esp_err_t* (**stack_input*))*esp_eth_handle_t* hdl, uint8_t *buffer, uint32_t length, void *priv , void **priv*Update Ethernet data input path (i.e. specify where to pass the input buffer)

Note After install driver, Ethernet still don't know where to deliver the input buffer. In fact, this API registers a callback function which get invoked when Ethernet received new packets.

Return

- ESP_OK: update input path successfully
- ESP_ERR_INVALID_ARG: update input path failed because of some invalid argument
- ESP_FAIL: update input path failed because some other error occurred

Parameters

- `hdl`: handle of Ethernet driver
- `stack_input`: function pointer, which does the actual process on incoming packets
- `priv`: private resource, which gets passed to `stack_input` callback without any modification

esp_err_t **esp_eth_transmit**(*esp_eth_handle_t hdl*, void **buf*, uint32_t *length*)

General Transmit.

Return

- ESP_OK: transmit frame buffer successfully
- ESP_ERR_INVALID_ARG: transmit frame buffer failed because of some invalid argument
- ESP_FAIL: transmit frame buffer failed because some other error occurred

Parameters

- `hdl`: handle of Ethernet driver
- `buf`: buffer of the packet to transfer
- `length`: length of the buffer to transfer

esp_err_t **esp_eth_receive**(*esp_eth_handle_t hdl*, uint8_t **buf*, uint32_t **length*)

General Receive.

Note Before this function got invoked, the value of "length" should set by user, equals the size of buffer. After the function returned, the value of "length" means the real length of received data.

Return

- ESP_OK: receive frame buffer successfully
- ESP_ERR_INVALID_ARG: receive frame buffer failed because of some invalid argument
- ESP_ERR_INVALID_SIZE: input buffer size is not enough to hold the incoming data. in this case, value of returned "length" indicates the real size of incoming data.
- ESP_FAIL: receive frame buffer failed because some other error occurred

Parameters

- `hdl`: handle of Ethernet driver
- `buf`: buffer to preserve the received packet
- `length`: length of the received packet

esp_err_t **esp_eth_ioctl**(*esp_eth_handle_t hdl*, *esp_eth_io_cmd_t cmd*, void **data*)

Misc IO function of Etherent driver.

Return

- ESP_OK: process io command successfully
- ESP_ERR_INVALID_ARG: process io command failed because of some invalid argument
- ESP_FAIL: process io command failed because some other error occurred

Parameters

- `hdl`: handle of Ethernet driver
- `cmd`: IO control command
- `data`: specificed data for command

esp_err_t **esp_eth_increase_reference**(*esp_eth_handle_t hdl*)

Increase Ethernet driver reference.

Note Ethernet driver handle can be obtained by os timer, netif, etc. It's dangerous when thread A is using Ethernet but thread B uninstall the driver. Using reference counter can prevent such risk, but care should be taken, when you obtain Ethernet driver, this API must be invoked so that the driver won't be uninstalled during your using time.

Return

- ESP_OK: increase reference successfully
- ESP_ERR_INVALID_ARG: increase reference failed because of some invalid argument

Parameters

- `hdl`: handle of Ethernet driver

esp_err_t **esp_eth_decrease_reference**(*esp_eth_handle_t hdl*)

Decrease Ethernet driver reference.

Return

- ESP_OK: increase reference successfully
- ESP_ERR_INVALID_ARG: increase reference failed because of some invalid argument

Parameters

- `hdl`: handle of Ethernet driver

Structures

struct esp_eth_config_t

Configuration of Ethernet driver.

Public Members

esp_eth_mac_t ***mac**

Ethernet MAC object.

esp_eth_phy_t ***phy**

Ethernet PHY object.

uint32_t **check_link_period_ms**

Period time of checking Ethernet link status.

esp_err_t (***stack_input**)(*esp_eth_handle_t* eth_handle, uint8_t *buffer, uint32_t length, void *priv)

Input frame buffer to user's stack.

Return

- ESP_OK: input frame buffer to upper stack successfully
- ESP_FAIL: error occurred when inputting buffer to upper stack

Parameters

- `eth_handle`: handle of Ethernet driver
- `buffer`: frame buffer that will get input to upper stack
- `length`: length of the frame buffer

esp_err_t (*`on_lowlevel_init_done`)(*esp_eth_handle_t* eth_handle)
Callback function invoked when lowlevel initialization is finished.

Return

- ESP_OK: process extra lowlevel initialization successfully
- ESP_FAIL: error occurred when processing extra lowlevel initialization

Parameters

- `eth_handle`: handle of Ethernet driver

esp_err_t (*`on_lowlevel_deinit_done`)(*esp_eth_handle_t* eth_handle)
Callback function invoked when lowlevel deinitialization is finished.

Return

- ESP_OK: process extra lowlevel deinitialization successfully
- ESP_FAIL: error occurred when processing extra lowlevel deinitialization

Parameters

- `eth_handle`: handle of Ethernet driver

Macros

`ETH_DEFAULT_CONFIG`(emac, ephy)
Default configuration for Ethernet driver.

Type Definitions

`typedef` void *`esp_eth_handle_t`
Handle of Ethernet driver.

API 参考 – 通用接口

Header File

- esp_eth/include/esp_eth_com.h

Functions

esp_err_t `esp_eth_detect_phy_addr`(*esp_eth_mediator_t* **eth*, uint32_t **detected_addr*)
Detect PHY address.

Return

- ESP_OK: detect phy address successfully
- ESP_ERR_INVALID_ARG: invalid parameter
- ESP_ERR_NOT_FOUND: can’t detect any PHY device
- ESP_FAIL: detect phy address failed because some error occurred

Parameters

- `eth`: mediator of Ethernet driver
- `detected_addr`: a valid address after detection

Structures

struct esp_eth_mediator_s

Ethernet mediator.

Public Members

esp_err_t (***phy_reg_read**)(*esp_eth_mediator_t* *eth, uint32_t phy_addr, uint32_t phy_reg, uint32_t *reg_value)

Read PHY register.

Return

- ESP_OK: read PHY register successfully
- ESP_FAIL: read PHY register failed because some error occurred

Parameters

- `eth`: mediator of Ethernet driver
- `phy_addr`: PHY Chip address (0~31)
- `phy_reg`: PHY register index code
- `reg_value`: PHY register value

esp_err_t (***phy_reg_write**)(*esp_eth_mediator_t* *eth, uint32_t phy_addr, uint32_t phy_reg, uint32_t reg_value)

Write PHY register.

Return

- ESP_OK: write PHY register successfully
- ESP_FAIL: write PHY register failed because some error occurred

Parameters

- `eth`: mediator of Ethernet driver
- `phy_addr`: PHY Chip address (0~31)
- `phy_reg`: PHY register index code
- `reg_value`: PHY register value

esp_err_t (***stack_input**)(*esp_eth_mediator_t* *eth, uint8_t *buffer, uint32_t length)

Deliver packet to upper stack.

Return

- ESP_OK: deliver packet to upper stack successfully
- ESP_FAIL: deliver packet failed because some error occurred

Parameters

- `eth`: mediator of Ethernet driver
- `buffer`: packet buffer
- `length`: length of the packet

esp_err_t (***on_state_changed**)(*esp_eth_mediator_t* *eth, *esp_eth_state_t* state, void *args)

Callback on Ethernet state changed.

Return

- ESP_OK: process the new state successfully
- ESP_FAIL: process the new state failed because some error occurred

Parameters

- `eth`: mediator of Ethernet driver
- `state`: new state
- `args`: optional argument for the new state

Macros

`ETH_MAX_PAYLOAD_LEN`

Maximum Ethernet payload size.

`ETH_MIN_PAYLOAD_LEN`

Minimum Ethernet payload size.

ETH_HEADER_LEN
: Ethernet frame header size: Dest addr(6 Bytes) + Src addr(6 Bytes) + length/type(2 Bytes)

ETH_CRC_LEN
: Ethernet frame CRC length.

ETH_VLAN_TAG_LEN
: Optional 802.1q VLAN Tag length.

ETH_JUMBO_FRAME_PAYLOAD_LEN
: Jumbo frame payload size.

ETH_MAX_PACKET_SIZE
: Maximum frame size (1522 Bytes)

ETH_MIN_PACKET_SIZE
: Minimum frame size (64 Bytes)

Type Definitions

typedef struct *esp_eth_mediator_s* esp_eth_mediator_t
: Ethernet mediator.

Enumerations

enum esp_eth_state_t
: Ethernet driver state.

Values:

ETH_STATE_LLINIT
: Lowlevel init done

ETH_STATE_DEINIT
: Deinit done

ETH_STATE_LINK
: Link status changed

ETH_STATE_SPEED
: Speed updated

ETH_STATE_DUPLEX
: Duplex updated

enum esp_eth_io_cmd_t
: Command list for ioctl API.

Values:

`ETH_CMD_G_MAC_ADDR`
Get MAC address

`ETH_CMD_S_MAC_ADDR`
Set MAC address

`ETH_CMD_G_PHY_ADDR`
Get PHY address

`ETH_CMD_S_PHY_ADDR`
Set PHY address

`ETH_CMD_G_SPEED`
Get Speed

`ETH_CMD_S_PROMISCUOUS`
Set promiscuous mode

enum eth_link_t
Ethernet link status.

Values:

`ETH_LINK_UP`
Ethernet link is up

`ETH_LINK_DOWN`
Ethernet link is down

enum eth_speed_t
Ethernet speed.

Values:

`ETH_SPEED_10M`
Ethernet speed is 10Mbps

`ETH_SPEED_100M`
Ethernet speed is 100Mbps

enum eth_duplex_t
Ethernet duplex mode.

Values:

`ETH_DUPLEX_HALF`
Ethernet is in half duplex

`ETH_DUPLEX_FULL`
Ethernet is in full duplex

enum eth_event_t
Ethernet event declarations.

Values:

`ETHERNET_EVENT_START`
Ethernet driver start

`ETHERNET_EVENT_STOP`
Ethernet driver stop

`ETHERNET_EVENT_CONNECTED`
Ethernet got a valid link

`ETHERNET_EVENT_DISCONNECTED`
Ethernet lost a valid link

API 参考 – MAC 接口

Header File

- esp_eth/include/esp_eth_mac.h

Structures

`struct esp_eth_mac_s`
Ethernet MAC.

Public Members

esp_err_t (*`set_mediator`)(*esp_eth_mac_t* *mac, *esp_eth_mediator_t* *eth)
Set mediator for Ethernet MAC.

Return

- ESP_OK: set mediator for Ethernet MAC successfully
- ESP_ERR_INVALID_ARG: set mediator for Ethernet MAC failed because of invalid argument

Parameters

- `mac`: Ethernet MAC instance
- `eth`: Ethernet mediator

esp_err_t (*`init`)(*esp_eth_mac_t* *mac)
Initialize Ethernet MAC.

Return

- ESP_OK: initialize Ethernet MAC successfully
- ESP_ERR_TIMEOUT: initialize Ethernet MAC failed because of timeout
- ESP_FAIL: initialize Ethernet MAC failed because some other error occurred

Parameters

- `mac`: Ethernet MAC instance

esp_err_t (*`deinit`)(*esp_eth_mac_t* *mac)

Deinitialize Ethernet MAC.

Return

- ESP_OK: deinitialize Ethernet MAC successfully
- ESP_FAIL: deinitialize Ethernet MAC failed because some error occurred

Parameters

- `mac`: Ethernet MAC instance

esp_err_t (*`start`)(*esp_eth_mac_t* *mac)

Start Ethernet MAC.

Return

- ESP_OK: start Ethernet MAC successfully
- ESP_FAIL: start Ethernet MAC failed because some other error occurred

Parameters

- `mac`: Ethernet MAC instance

esp_err_t (*`stop`)(*esp_eth_mac_t* *mac)

Stop Ethernet MAC.

Return

- ESP_OK: stop Ethernet MAC successfully
- ESP_FAIL: stop Ethernet MAC failed because some error occurred

Parameters

- `mac`: Ethernet MAC instance

esp_err_t (*`transmit`)(*esp_eth_mac_t* *mac, uint8_t *buf, uint32_t length)

Transmit packet from Ethernet MAC.

Return

- ESP_OK: transmit packet successfully
- ESP_ERR_INVALID_ARG: transmit packet failed because of invalid argument
- ESP_ERR_INVALID_STATE: transmit packet failed because of wrong state of MAC
- ESP_FAIL: transmit packet failed because some other error occurred

Parameters

- `mac`: Ethernet MAC instance
- `buf`: packet buffer to transmit
- `length`: length of packet

esp_err_t (***receive**)(*esp_eth_mac_t* *mac, uint8_t *buf, uint32_t *length)

Receive packet from Ethernet MAC.

Note Memory of buf is allocated in the Layer2, make sure it get free after process.

Note Before this function got invoked, the value of "length" should set by user, equals the size of buffer. After the function returned, the value of "length" means the real length of received data.

Return

- ESP_OK: receive packet successfully
- ESP_ERR_INVALID_ARG: receive packet failed because of invalid argument
- ESP_ERR_INVALID_SIZE: input buffer size is not enough to hold the incoming data. in this case, value of returned "length" indicates the real size of incoming data.
- ESP_FAIL: receive packet failed because some other error occurred

Parameters

- `mac`: Ethernet MAC instance
- `buf`: packet buffer which will preserve the received frame
- `length`: length of the received packet

esp_err_t (***read_phy_reg**)(*esp_eth_mac_t* *mac, uint32_t phy_addr, uint32_t phy_reg, uint32_t *reg_value)

Read PHY register.

Return

- ESP_OK: read PHY register successfully
- ESP_ERR_INVALID_ARG: read PHY register failed because of invalid argument
- ESP_ERR_INVALID_STATE: read PHY register failed because of wrong state of MAC

- ESP_ERR_TIMEOUT: read PHY register failed because of timeout
- ESP_FAIL: read PHY register failed because some other error occurred

Parameters

- `mac`: Ethernet MAC instance
- `phy_addr`: PHY chip address (0~31)
- `phy_reg`: PHY register index code
- `reg_value`: PHY register value

esp_err_t (***write_phy_reg**)(*esp_eth_mac_t* *mac, uint32_t phy_addr, uint32_t phy_reg, uint32_t reg_value)

Write PHY register.

Return

- ESP_OK: write PHY register successfully
- ESP_ERR_INVALID_STATE: write PHY register failed because of wrong state of MAC
- ESP_ERR_TIMEOUT: write PHY register failed because of timeout
- ESP_FAIL: write PHY register failed because some other error occurred

Parameters

- `mac`: Ethernet MAC instance
- `phy_addr`: PHY chip address (0~31)
- `phy_reg`: PHY register index code
- `reg_value`: PHY register value

esp_err_t (***set_addr**)(*esp_eth_mac_t* *mac, uint8_t *addr)

Set MAC address.

Return

- ESP_OK: set MAC address successfully
- ESP_ERR_INVALID_ARG: set MAC address failed because of invalid argument
- ESP_FAIL: set MAC address failed because some other error occurred

Parameters

- `mac`: Ethernet MAC instance
- `addr`: MAC address

esp_err_t (*`get_addr`)(*esp_eth_mac_t* *mac, uint8_t *addr)
Get MAC address.

Return

- ESP_OK: get MAC address successfully
- ESP_ERR_INVALID_ARG: get MAC address failed because of invalid argument
- ESP_FAIL: get MAC address failed because some other error occurred

Parameters

- `mac`: Ethernet MAC instance
- `addr`: MAC address

esp_err_t (*`set_speed`)(*esp_eth_mac_t* *mac, *eth_speed_t* speed)
Set speed of MAC.

Return

- ESP_OK: set MAC speed successfully
- ESP_ERR_INVALID_ARG: set MAC speed failed because of invalid argument
- ESP_FAIL: set MAC speed failed because some other error occurred

Parameters

- `ma:c`: Ethernet MAC instance
- `speed`: MAC speed

esp_err_t (*`set_duplex`)(*esp_eth_mac_t* *mac, *eth_duplex_t* duplex)
Set duplex mode of MAC.

Return

- ESP_OK: set MAC duplex mode successfully
- ESP_ERR_INVALID_ARG: set MAC duplex failed because of invalid argument
- ESP_FAIL: set MAC duplex failed because some other error occurred

Parameters

- `mac`: Ethernet MAC instance
- `duplex`: MAC duplex

esp_err_t (*`set_link`)(*esp_eth_mac_t* *mac, *eth_link_t* link)
Set link status of MAC.

Return

- ESP_OK: set link status successfully
- ESP_ERR_INVALID_ARG: set link status failed because of invalid argument
- ESP_FAIL: set link status failed because some other error occurred

Parameters

- `mac`: Ethernet MAC instance
- `link`: Link status

esp_err_t (***set_promiscuous**)(*esp_eth_mac_t* *mac, bool enable)

Set promiscuous of MAC.

Return

- ESP_OK: set promiscuous mode successfully
- ESP_FAIL: set promiscuous mode failed because some error occurred

Parameters

- `mac`: Ethernet MAC instance
- `enable`: set true to enable promiscuous mode; set false to disable promiscuous mode

esp_err_t (***del**)(*esp_eth_mac_t* *mac)

Free memory of Ethernet MAC.

Return

- ESP_OK: free Ethernet MAC instance successfully
- ESP_FAIL: free Ethernet MAC instance failed because some error occurred

Parameters

- `mac`: Ethernet MAC instance

struct eth_mac_config_t

Configuration of Ethernet MAC object.

Public Members

uint32_t **sw_reset_timeout_ms**

Software reset timeout value (Unit: ms)

uint32_t **rx_task_stack_size**

Stack size of the receive task

uint32_t **rx_task_prio**
Priority of the receive task

int **smi_mdc_gpio_num**
SMI MDC GPIO number

int **smi_mdio_gpio_num**
SMI MDIO GPIO number

uint32_t **flags**
Flags that specify extra capability for mac driver

Macros

ETH_MAC_FLAG_WORK_WITH_CACHE_DISABLE
MAC driver can work when cache is disabled

ETH_MAC_DEFAULT_CONFIG()
Default configuration for Ethernet MAC object.

Type Definitions

typedef struct *esp_eth_mac_s* **esp_eth_mac_t**
Ethernet MAC.

API 参考 – PHY 接口

Header File

- esp_eth/include/esp_eth_phy.h

Functions

esp_eth_phy_t ***esp_eth_phy_new_ip101(const** *eth_phy_config_t* ***config)**
Create a PHY instance of IP101.

Return

- instance: create PHY instance successfully
- NULL: create PHY instance failed because some error occurred

Parameters

- `config`: configuration of PHY

esp_eth_phy_t ***esp_eth_phy_new_rtl8201**(**const** *eth_phy_config_t* **config*)

Create a PHY instance of RTL8201.

Return

- instance: create PHY instance successfully
- NULL: create PHY instance failed because some error occurred

Parameters

- `config`: configuration of PHY

esp_eth_phy_t ***esp_eth_phy_new_lan8720**(**const** *eth_phy_config_t* **config*)

Create a PHY instance of LAN8720.

Return

- instance: create PHY instance successfully
- NULL: create PHY instance failed because some error occurred

Parameters

- `config`: configuration of PHY

esp_eth_phy_t ***esp_eth_phy_new_dp83848**(**const** *eth_phy_config_t* **config*)

Create a PHY instance of DP83848.

Return

- instance: create PHY instance successfully
- NULL: create PHY instance failed because some error occurred

Parameters

- `config`: configuration of PHY

Structures

struct esp_eth_phy_s

Ethernet PHY.

Public Members

esp_err_t (***set_mediator**)(*esp_eth_phy_t* *phy, *esp_eth_mediator_t* *mediator)

Set mediator for PHY.

Return

- ESP_OK: set mediator for Ethernet PHY instance successfully
- ESP_ERR_INVALID_ARG: set mediator for Ethernet PHY instance failed because of some invalid arguments

Parameters

- `phy`: Ethernet PHY instance
- `mediator`: mediator of Ethernet driver

esp_err_t (***reset**)(*esp_eth_phy_t* *phy)

Software Reset Ethernet PHY.

Return

- ESP_OK: reset Ethernet PHY successfully
- ESP_FAIL: reset Ethernet PHY failed because some error occurred

Parameters

- `phy`: Ethernet PHY instance

esp_err_t (***reset_hw**)(*esp_eth_phy_t* *phy)

Hardware Reset Ethernet PHY.

Note Hardware reset is mostly done by pull down and up PHY' s nRST pin

Return

- ESP_OK: reset Ethernet PHY successfully
- ESP_FAIL: reset Ethernet PHY failed because some error occurred

Parameters

- `phy`: Ethernet PHY instance

esp_err_t (***init**)(*esp_eth_phy_t* *phy)

Initialize Ethernet PHY.

Return

- ESP_OK: initialize Ethernet PHY successfully
- ESP_FAIL: initialize Ethernet PHY failed because some error occurred

Parameters

- `phy`: Ethernet PHY instance

esp_err_t (***deinit**)(*esp_eth_phy_t* *phy)

Deinitialize Ethernet PHY.

Return

- ESP_OK: deinitialize Ethernet PHY successfully
- ESP_FAIL: deinitialize Ethernet PHY failed because some error occurred

Parameters

- `phyL`: Ethernet PHY instance

esp_err_t (***negotiate**)(*esp_eth_phy_t* *phy)

Start auto negotiation.

Return

- ESP_OK: restart auto negotiation successfully
- ESP_FAIL: restart auto negotiation failed because some error occurred

Parameters

- `phy`: Ethernet PHY instance

esp_err_t (***get_link**)(*esp_eth_phy_t* *phy)

Get Ethernet PHY link status.

Return

- ESP_OK: get Ethernet PHY link status successfully
- ESP_FAIL: get Ethernet PHY link status failed because some error occurred

Parameters

- `phy`: Ethernet PHY instance

esp_err_t (***pwrctl**)(*esp_eth_phy_t* *phy, bool enable)

Power control of Ethernet PHY.

Return

- ESP_OK: control Ethernet PHY power successfully
- ESP_FAIL: control Ethernet PHY power failed because some error occurred

Parameters

- `phy`: Ethernet PHY instance
- `enable`: set true to power on Ethernet PHY; ser false to power off Ethernet PHY

esp_err_t (***set_addr**)(*esp_eth_phy_t* *phy, uint32_t addr)

Set PHY chip address.

Return

- ESP_OK: set Ethernet PHY address successfully
- ESP_FAIL: set Ethernet PHY address failed because some error occurred

Parameters

- `phy`: Ethernet PHY instance
- `addr`: PHY chip address

esp_err_t (*`get_addr`)(*esp_eth_phy_t* *phy, uint32_t *addr)

Get PHY chip address.

Return

- ESP_OK: get Ethernet PHY address successfully
- ESP_ERR_INVALID_ARG: get Ethernet PHY address failed because of invalid argument

Parameters

- `phy`: Ethernet PHY instance
- `addr`: PHY chip address

esp_err_t (*`del`)(*esp_eth_phy_t* *phy)

Free memory of Ethernet PHY instance.

Return

- ESP_OK: free PHY instance successfully
- ESP_FAIL: free PHY instance failed because some error occurred

Parameters

- `phy`: Ethernet PHY instance

`struct eth_phy_config_t`

Ethernet PHY configuration.

Public Members

int32_t `phy_addr`

PHY address, set -1 to enable PHY address detection at initialization stage

uint32_t `reset_timeout_ms`

Reset timeout value (Unit: ms)

uint32_t **autonego_timeout_ms**
: Auto-negotiation timeout value (Unit: ms)

int **reset_gpio_num**
: Reset GPIO number, -1 means no hardware reset

Macros

`ESP_ETH_PHY_ADDR_AUTO`

`ETH_PHY_DEFAULT_CONFIG()`
: Default configuration for Ethernet PHY object.

Type Definitions

typedef struct *esp_eth_phy_s* **esp_eth_phy_t**
: Ethernet PHY.

API 参考 – esp_netif 相关使用

Header File

- esp_eth/include/esp_eth_netif_glue.h

Functions

void ***esp_eth_new_netif_glue**(*esp_eth_handle_t eth_hdl*)
: Create a netif glue for Ethernet driver.

Note netif glue is used to attach io driver to TCP/IP netif

Return glue object, which inherits esp_netif_driver_base_t

Parameters

- `eth_hdl`: Ethernet driver handle

esp_err_t **esp_eth_del_netif_glue**(void **glue*)
: Delete netif glue of Ethernet driver.

Return -ESP_OK: delete netif glue successfully

Parameters

- `glue`: netif glue

esp_err_t **esp_eth_set_default_handlers**(void **esp_netif*)

Register default IP layer handlers for Ethernet.

Note : Ethernet handle might not yet properly initialized when setting up these default handlers

Return

- ESP_ERR_INVALID_ARG: invalid parameter (esp_netif is NULL)
- ESP_OK: set default IP layer handlers successfully
- others: other failure occurred during register esp_event handler

Parameters

- **esp_netif**: esp network interface handle created for Ethernet driver

esp_err_t **esp_eth_clear_default_handlers**(void **esp_netif*)

Unregister default IP layer handlers for Ethernet.

Return

- ESP_ERR_INVALID_ARG: invalid parameter (esp_netif is NULL)
- ESP_OK: clear default IP layer handlers successfully
- others: other failure occurred during unregister esp_event handler

Parameters

- **esp_netif**: esp network interface handle created for Ethernet driver

本部分的以太网 API 示例代码存放在 ESP-IDF 示例项目的 ethernet 目录下。

2.2.3 IP 网络层协议

ESP-NETIF

The purpose of ESP-NETIF library is twofold:

- It provides an abstraction layer for the application on top of the TCP/IP stack. This will allow applications to choose between IP stacks in the future.
- The APIs it provides are thread safe, even if the underlying TCP/IP stack APIs are not.

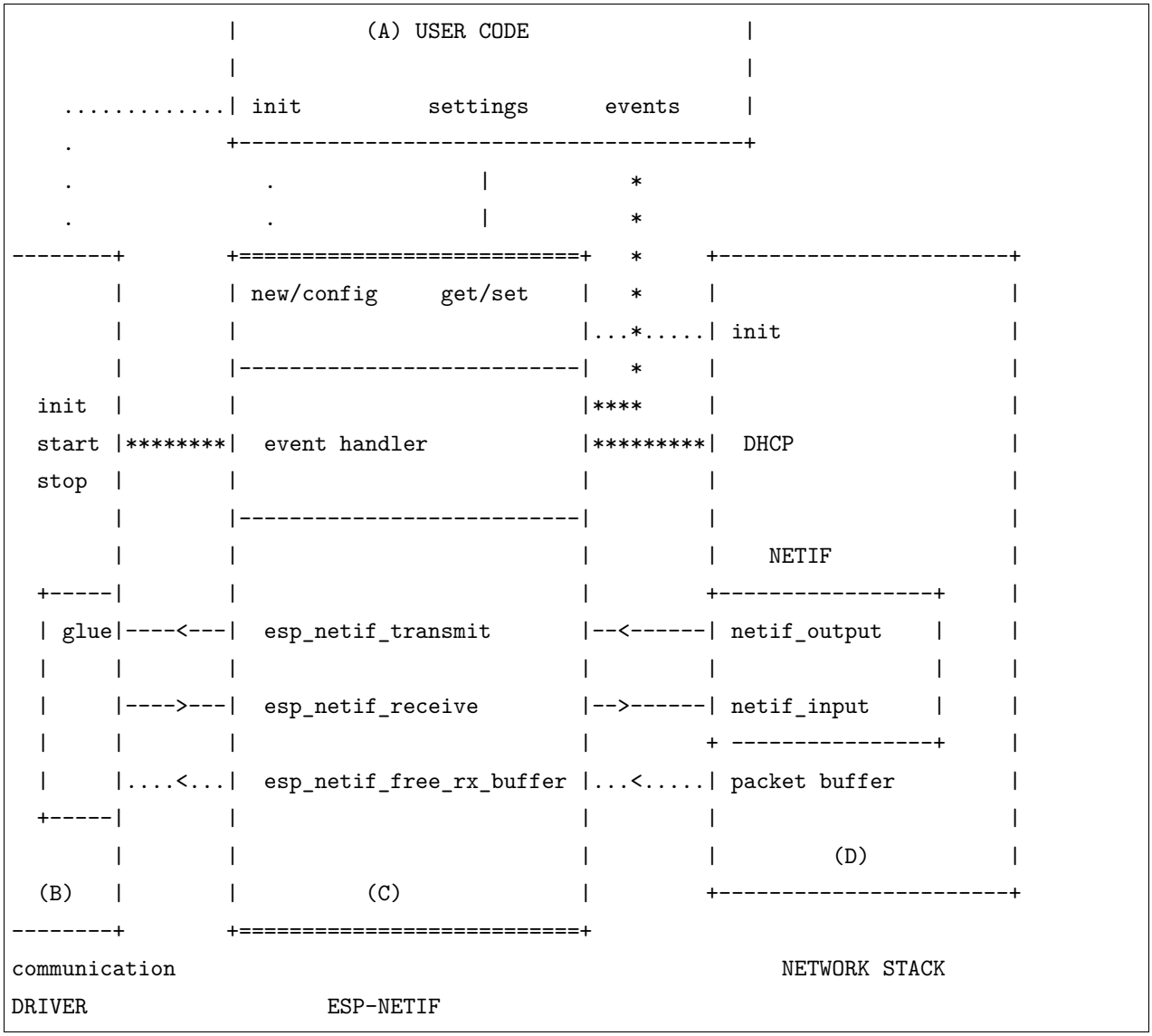
ESP-IDF currently implements ESP-NETIF for the lwIP TCP/IP stack only. However, the adapter itself is TCP/IP implementation agnostic and different implementations are possible.

Some ESP-NETIF API functions are intended to be called by application code, for example to get/set interface IP addresses, configure DHCP. Other functions are intended for internal ESP-IDF use by the network driver layer.

In many cases, applications do not need to call ESP-NETIF APIs directly as they are called from the default network event handlers.

ESP-NETIF component is a successor of the tcpip_adapter, former network interface abstraction, which has become deprecated since IDF v4.1. Please refer to the *TCP/IP 适配器迁移指南* section in case existing applications to be ported to use the esp-netif API instead.

ESP-NETIF architecture



Data and event flow in the diagram

- ..... Initialization line from user code to ESP-NETIF and communication driver

- `--<---->--` Data packets going from communication media to TCP/IP stack and back
- `********` Events aggregated in ESP-NETIF propagates to driver, user code and network stack
- `|` User settings and runtime configuration

ESP-NETIF interaction

A) User code, boiler plate

Overall application interaction with a specific IO driver for communication media and configured TCP/IP network stack is abstracted using ESP-NETIF APIs and outlined as below:

A) Initialization code

1) Initializes IO driver
2) Creates a new instance of ESP-NETIF and configure with
   - ESP-NETIF specific options (flags, behaviour, name)
   - Network stack options (netif init and input functions, not publicly available)
   - IO driver specific options (transmit, free rx buffer functions, IO driver handle)
3) Attaches the IO driver handle to the ESP-NETIF instance created in the above steps
4) Configures event handlers
   - use default handlers for common interfaces defined in IO drivers; or define a specific handlers for customised behaviour/new interfaces
   - register handlers for app related events (such as IP lost/acquired)

B) Interaction with network interfaces using ESP-NETIF API

- Getting and setting TCP/IP related parameters (DHCP, IP, etc)
- Receiving IP events (connect/disconnect)
- Controlling application lifecycle (set interface up/down)

B) Communication driver, IO driver, media driver

Communication driver plays these two important roles in relation with ESP-NETIF:

1) Event handlers: Define behaviour patterns of interaction with ESP-NETIF (for example: ethernet link-up -> turn netif on)
2) Glue IO layer: Adapts the input/output functions to use ESP-NETIF transmit, receive and free receive buffer

- Installs driver_transmit to appropriate ESP-NETIF object, so that outgoing packets from network stack are passed to the IO driver
- Calls *esp_netif_receive()* to pass incoming data to network stack

C) ESP-NETIF, former tcpip_adapter

ESP-NETIF is an intermediary between an IO driver and a network stack, connecting packet data path between these two. As that it provides a set of interfaces for attaching a driver to ESP-NETIF object (runtime) and configuring a network stack (compile time). In addition to that a set of API is provided to control network interface lifecycle and its TCP/IP properties. As an overview, the ESP-NETIF public interface could be divided into these 6 groups:

1) Initialization APIs (to create and configure ESP-NETIF instance)

2) Input/Output API (for passing data between IO driver and network stack)

3) Event or Action API

- Used for network interface lifecycle management
- ESP-NETIF provides building blocks for designing event handlers

4) Setters and Getters for basic network interface properties

5) Network stack abstraction: enabling user interaction with TCP/IP stack

- Set interface up or down
- DHCP server and client API
- DNS API

6) Driver conversion utilities

D) Network stack

Network stack has no public interaction with application code with regard to public interfaces and shall be fully abstracted by ESP-NETIF API.

ESP-NETIF programmer's manual

Please refer to the example section for basic initialization of default interfaces:

- WiFi Station: examples/wifi/getting_started/station/main/station_example_main.c
- WiFi Access Point: examples/wifi/getting_started/softAP/main/softap_example_main.c
- Ethernet examples/ethernet/basic/main/ethernet_example_main.c

For more specific cases please consult this guide: *ESP-NETIF Custom I/O Driver*.

WiFi default initialization

The initialization code as well as registering event handlers for default interfaces, such as softAP and station, are provided in two separate APIs to facilitate simple startup code for most applications:

- *esp_netif_create_default_wifi_ap()*
- *esp_netif_create_default_wifi_sta()*

Please note that these functions return the `esp_netif` handle, i.e. a pointer to a network interface object allocated and configured with default settings, which as a consequence, means that:

- The created object has to be destroyed if a network de-initialization is provided by an application.
- These *default* interfaces must not be created multiple times, unless the created handle is deleted using *esp_netif_destroy()*.
- When using Wifi in `AP+STA` mode, both these interfaces has to be created.

API Reference

Header File

- esp_netif/include/esp_netif.h

Functions

esp_err_t **esp_netif_init**(void)

Initialize the underlying TCP/IP stack.

Return

- ESP_OK on success
- ESP_FAIL if initializing failed

Note This function should be called exactly once from application code, when the application starts up.

esp_err_t **esp_netif_deinit**(void)

Deinitialize the esp-netif component (and the underlying TCP/IP stack)

Note: Deinitialization is not supported yet

Return

- ESP_ERR_INVALID_STATE if esp_netif not initialized
- ESP_ERR_NOT_SUPPORTED otherwise

esp_netif_t *esp_netif_new(const esp_netif_config_t *esp_netif_config)

Creates an instance of new esp-netif object based on provided config.

Return

- pointer to esp-netif object on success
- NULL otherwise

Parameters

- esp_netif_config: pointer esp-netif configuration

void esp_netif_destroy(esp_netif_t *esp_netif)

Destroys the esp_netif object.

Parameters

- esp_netif: pointer to the object to be deleted

esp_err_t esp_netif_set_driver_config(esp_netif_t *esp_netif, const esp_netif_driver_ifconfig_t *driver_config)

Configures driver related options of esp_netif object.

Return

- ESP_OK on success
- ESP_ERR_ESP_NETIF_INVALID_PARAMS if invalid parameters provided

Parameters

- esp_netif: pointer to the object to be configured
- driver_config: pointer esp-netif io driver related configuration

esp_err_t esp_netif_attach(esp_netif_t *esp_netif, esp_netif_iodriver_handle driver_handle)

Attaches esp_netif instance to the io driver handle.

Calling this function enables connecting specific esp_netif object with already initialized io driver to update esp_netif object with driver specific configuration (i.e. calls post_attach callback, which typically sets io driver callbacks to esp_netif instance and starts the driver)

Return

- ESP_OK on success
- ESP_ERR_ESP_NETIF_DRIVER_ATTACH_FAILED if driver' s pot_attach callback failed

Parameters

- esp_netif: pointer to esp_netif object to be attached

- `driver_handle`: pointer to the driver handle

esp_err_t **esp_netif_receive**(esp_netif_t *esp_netif, void *buffer, size_t len, void *eb)

Passes the raw packets from communication media to the appropriate TCP/IP stack.

This function is called from the configured (peripheral) driver layer. The data are then forwarded as frames to the TCP/IP stack.

Return

- ESP_OK

Parameters

- `esp_netif`: Handle to esp-netif instance
- `buffer`: Received data
- `len`: Length of the data frame
- `eb`: Pointer to internal buffer (used in Wi-Fi driver)

void **esp_netif_action_start**(void *esp_netif, *esp_event_base_t* base, int32_t event_id, void *data)

Default building block for network interface action upon IO driver start event Creates network interface, if AUTOUP enabled turns the interface on, if DHCPS enabled starts dhcp server.

Note This API can be directly used as event handler

Parameters

- `esp_netif`: Handle to esp-netif instance
- `base`:
- `event_id`:
- `data`:

void **esp_netif_action_stop**(void *esp_netif, *esp_event_base_t* base, int32_t event_id, void *data)

Default building block for network interface action upon IO driver stop event.

Note This API can be directly used as event handler

Parameters

- `esp_netif`: Handle to esp-netif instance
- `base`:
- `event_id`:
- `data`:

void **esp_netif_action_connected**(void *esp_netif, esp_event_base_t base, int32_t event_id, void *data)

Default building block for network interface action upon IO driver connected event.

Note This API can be directly used as event handler

Parameters

- `esp_netif`: Handle to esp-netif instance
- `base`:
- `event_id`:
- `data`:

void **esp_netif_action_disconnected**(void *esp_netif, esp_event_base_t base, int32_t event_id, void *data)

Default building block for network interface action upon IO driver disconnected event.

Note This API can be directly used as event handler

Parameters

- `esp_netif`: Handle to esp-netif instance
- `base`:
- `event_id`:
- `data`:

void **esp_netif_action_got_ip**(void *esp_netif, esp_event_base_t base, int32_t event_id, void *data)

Default building block for network interface action upon network got IP event.

Note This API can be directly used as event handler

Parameters

- `esp_netif`: Handle to esp-netif instance
- `base`:
- `event_id`:
- `data`:

esp_err_t **esp_netif_set_mac**(esp_netif_t *esp_netif, uint8_t mac[])

Set the mac address for the interface instance.

Return ESP_OK

Parameters

- `esp_netif`: Handle to esp-netif instance
- `mac`: Desired mac address for the related network interface

esp_err_t **esp_netif_set_hostname**(esp_netif_t *_esp_netif_, **const** char *_hostname_)

Set the hostname of an interface.

Return

- ESP_OK - success
- ESP_ERR_ESP_NETIF_IF_NOT_READY - interface status error
- ESP_ERR_ESP_NETIF_INVALID_PARAMS - parameter error

Parameters

- `esp_netif`: Handle to esp-netif instance
- `hostname`: New hostname for the interface. Maximum length 32 bytes.

esp_err_t **esp_netif_get_hostname**(esp_netif_t *_esp_netif_, **const** char **_hostname_)

Get interface hostname.

Return

- ESP_OK - success
- ESP_ERR_ESP_NETIF_IF_NOT_READY - interface status error
- ESP_ERR_ESP_NETIF_INVALID_PARAMS - parameter error

Parameters

- `esp_netif`: Handle to esp-netif instance
- `hostname`: Returns a pointer to the hostname. May be NULL if no hostname is set. If set non-NULL, pointer remains valid (and string may change if the hostname changes).

bool **esp_netif_is_netif_up**(esp_netif_t *_esp_netif_)

Test if supplied interface is up or down.

Return

- true - Interface is up
- false - Interface is down

Parameters

- `esp_netif`: Handle to esp-netif instance

esp_err_t **esp_netif_get_ip_info**(esp_netif_t *_esp_netif_, esp_netif_ip_info_t *_ip_info_)
Get interface' s IP address information.

If the interface is up, IP information is read directly from the TCP/IP stack. If the interface is down, IP information is read from a copy kept in the ESP-NETIF instance

Return

- ESP_OK
- ESP_ERR_ESP_NETIF_INVALID_PARAMS

Parameters

- `esp_netif`: Handle to esp-netif instance
- `ip_info`: If successful, IP information will be returned in this argument.

esp_err_t **esp_netif_get_old_ip_info**(esp_netif_t *_esp_netif_, esp_netif_ip_info_t *_ip_info_)
Get interface' s old IP information.

Returns an "old" IP address previously stored for the interface when the valid IP changed.

If the IP lost timer has expired (meaning the interface was down for longer than the configured interval) then the old IP information will be zero.

Return

- ESP_OK
- ESP_ERR_ESP_NETIF_INVALID_PARAMS

Parameters

- `esp_netif`: Handle to esp-netif instance
- `ip_info`: If successful, IP information will be returned in this argument.

esp_err_t **esp_netif_set_ip_info**(esp_netif_t *_esp_netif_, **const** esp_netif_ip_info_t *_ip_info_)
Set interface' s IP address information.

This function is mainly used to set a static IP on an interface.

If the interface is up, the new IP information is set directly in the TCP/IP stack.

The copy of IP information kept in the ESP-NETIF instance is also updated (this copy is returned if the IP is queried while the interface is still down.)

Note DHCP client/server must be stopped (if enabled for this interface) before setting new IP information.

Note Calling this interface for may generate a SYSTEM_EVENT_STA_GOT_IP or SYSTEM_EVENT_ETH_GOT_IP event.

Return

- ESP_OK
- ESP_ERR_ESP_NETIF_INVALID_PARAMS
- ESP_ERR_ESP_NETIF_DHCP_NOT_STOPPED If DHCP server or client is still running

Parameters

- `esp_netif`: Handle to esp-netif instance
- `ip_info`: IP information to set on the specified interface

esp_err_t **esp_netif_set_old_ip_info**(esp_netif_t **esp_netif*, **const** esp_netif_ip_info_t **ip_info*)

Set interface old IP information.

This function is called from the DHCP client (if enabled), before a new IP is set. It is also called from the default handlers for the SYSTEM_EVENT_STA_CONNECTED and SYSTEM_EVENT_ETH_CONNECTED events.

Calling this function stores the previously configured IP, which can be used to determine if the IP changes in the future.

If the interface is disconnected or down for too long, the "IP lost timer" will expire (after the configured interval) and set the old IP information to zero.

Return

- ESP_OK
- ESP_ERR_ESP_NETIF_INVALID_PARAMS

Parameters

- `esp_netif`: Handle to esp-netif instance
- `ip_info`: Store the old IP information for the specified interface

int **esp_netif_get_netif_impl_index**(esp_netif_t **esp_netif*)

Get net interface index from network stack implementation.

Note This index could be used in `setsockopt()` to bind socket with multicast interface

Return implementation specific index of interface represented with supplied esp_netif

Parameters

- `esp_netif`: Handle to esp-netif instance

esp_err_t **esp_netif_dhcps_option**(esp_netif_t **esp_netif*, esp_netif_dhcp_option_mode_t *opt_op*, esp_netif_dhcp_option_id_t *opt_id*, void **opt_val*, uint32_t *opt_len*)

Set or Get DHCP server option.

Return

- ESP_OK
- ESP_ERR_ESP_NETIF_INVALID_PARAMS
- ESP_ERR_ESP_NETIF_DHCP_ALREADY_STOPPED
- ESP_ERR_ESP_NETIF_DHCP_ALREADY_STARTED

Parameters

- `esp_netif`: Handle to esp-netif instance
- `opt_op`: ESP_NETIF_OP_SET to set an option, ESP_NETIF_OP_GET to get an option.
- `opt_id`: Option index to get or set, must be one of the supported enum values.
- `opt_val`: Pointer to the option parameter.
- `opt_len`: Length of the option parameter.

esp_err_t **esp_netif_dhcpc_option**(esp_netif_t **esp_netif*, esp_netif_dhcp_option_mode_t *opt_op*, esp_netif_dhcp_option_id_t *opt_id*, void **opt_val*, uint32_t *opt_len*)

Set or Get DHCP client option.

Return

- ESP_OK
- ESP_ERR_ESP_NETIF_INVALID_PARAMS
- ESP_ERR_ESP_NETIF_DHCP_ALREADY_STOPPED
- ESP_ERR_ESP_NETIF_DHCP_ALREADY_STARTED

Parameters

- `esp_netif`: Handle to esp-netif instance
- `opt_op`: ESP_NETIF_OP_SET to set an option, ESP_NETIF_OP_GET to get an option.
- `opt_id`: Option index to get or set, must be one of the supported enum values.
- `opt_val`: Pointer to the option parameter.
- `opt_len`: Length of the option parameter.

esp_err_t **esp_netif_dhcpc_start**(esp_netif_t **esp_netif*)

Start DHCP client (only if enabled in interface object)

Note The default event handlers for the SYSTEM_EVENT_STA_CONNECTED and SYSTEM_EVENT_ETH_CONNECTED events call this function.

Return

- ESP_OK
- ESP_ERR_ESP_NETIF_INVALID_PARAMS
- ESP_ERR_ESP_NETIF_DHCP_ALREADY_STARTED
- ESP_ERR_ESP_NETIF_DHCPC_START_FAILED

Parameters

- `esp_netif`: Handle to esp-netif instance

esp_err_t **esp_netif_dhcpc_stop**(esp_netif_t **esp_netif*)

Stop DHCP client (only if enabled in interface object)

Note Calling action_netif_stop() will also stop the DHCP Client if it is running.

Return

- ESP_OK
- ESP_ERR_ESP_NETIF_INVALID_PARAMS
- ESP_ERR_ESP_NETIF_DHCP_ALREADY_STOPPED
- ESP_ERR_ESP_NETIF_IF_NOT_READY

Parameters

- `esp_netif`: Handle to esp-netif instance

esp_err_t **esp_netif_dhcpc_get_status**(esp_netif_t **esp_netif*, esp_netif_dhcp_status_t **status*)

Get DHCP client status.

Return

- ESP_OK

Parameters

- `esp_netif`: Handle to esp-netif instance
- `status`: If successful, the status of DHCP client will be returned in this argument.

esp_err_t **esp_netif_dhcps_get_status**(esp_netif_t **esp_netif*, esp_netif_dhcp_status_t **status*)

Get DHCP Server status.

Return

- ESP_OK

Parameters

- `esp_netif`: Handle to esp-netif instance
- `status`: If successful, the status of the DHCP server will be returned in this argument.

esp_err_t **esp_netif_dhcps_start**(esp_netif_t **esp_netif*)

Start DHCP server (only if enabled in interface object)

Return

- ESP_OK
- ESP_ERR_ESP_NETIF_INVALID_PARAMS
- ESP_ERR_ESP_NETIF_DHCP_ALREADY_STARTED

Parameters

- `esp_netif`: Handle to esp-netif instance

esp_err_t **esp_netif_dhcps_stop**(esp_netif_t **esp_netif*)

Stop DHCP server (only if enabled in interface object)

Return

- ESP_OK
- ESP_ERR_ESP_NETIF_INVALID_PARAMS
- ESP_ERR_ESP_NETIF_DHCP_ALREADY_STOPPED
- ESP_ERR_ESP_NETIF_IF_NOT_READY

Parameters

- `esp_netif`: Handle to esp-netif instance

esp_err_t **esp_netif_set_dns_info**(esp_netif_t **esp_netif*, esp_netif_dns_type_t *type*, esp_netif_dns_info_t **dns*)

Set DNS Server information.

This function behaves differently if DHCP server or client is enabled

If DHCP client is enabled, main and backup DNS servers will be updated automatically from the DHCP lease if the relevant DHCP options are set. Fallback DNS Server is never updated from the DHCP lease and is designed to be set via this API. If DHCP client is disabled, all DNS server types can be set via this API only.

If DHCP server is enabled, the Main DNS Server setting is used by the DHCP server to provide a DNS Server option to DHCP clients (Wi-Fi stations).

- The default Main DNS server is typically the IP of the Wi-Fi AP interface itself.
- This function can override it by setting server type ESP_NETIF_DNS_MAIN.
- Other DNS Server types are not supported for the Wi-Fi AP interface.

Return

- ESP_OK on success
- ESP_ERR_ESP_NETIF_INVALID_PARAMS invalid params

Parameters

- `esp_netif`: Handle to esp-netif instance
- `type`: Type of DNS Server to set: ESP_NETIF_DNS_MAIN, ESP_NETIF_DNS_BACKUP, ESP_NETIF_DNS_FALLBACK
- `dns`: DNS Server address to set

esp_err_t **esp_netif_get_dns_info**(esp_netif_t **esp_netif*, esp_netif_dns_type_t *type*, esp_netif_dns_info_t **dns*)

Get DNS Server information.

Return the currently configured DNS Server address for the specified interface and Server type.

This may be result of a previous call to *esp_netif_set_dns_info()*. If the interface's DHCP client is enabled, the Main or Backup DNS Server may be set by the current DHCP lease.

Return

- ESP_OK on success
- ESP_ERR_ESP_NETIF_INVALID_PARAMS invalid params

Parameters

- `esp_netif`: Handle to esp-netif instance
- `type`: Type of DNS Server to get: ESP_NETIF_DNS_MAIN, ESP_NETIF_DNS_BACKUP, ESP_NETIF_DNS_FALLBACK
- `dns`: DNS Server result is written here on success

esp_err_t **esp_netif_create_ip6_linklocal**(esp_netif_t **esp_netif*)

Create interface link-local IPv6 address.

Cause the TCP/IP stack to create a link-local IPv6 address for the specified interface.

This function also registers a callback for the specified interface, so that if the link-local address becomes verified as the preferred address then a SYSTEM_EVENT_GOT_IP6 event will be sent.

Return

- ESP_OK
- ESP_ERR_ESP_NETIF_INVALID_PARAMS

Parameters

- `esp_netif`: Handle to esp-netif instance

esp_err_t **esp_netif_get_ip6_linklocal**(esp_netif_t **esp_netif*, esp_ip6_addr_t **if_ip6*)

Get interface link-local IPv6 address.

If the specified interface is up and a preferred link-local IPv6 address has been created for the interface, return a copy of it.

Return

- ESP_OK
- ESP_FAIL If interface is down, does not have a link-local IPv6 address, or the link-local IPv6 address is not a preferred address.

Parameters

- `esp_netif`: Handle to esp-netif instance
- `if_ip6`: IPv6 information will be returned in this argument if successful.

esp_err_t **esp_netif_get_ip6_global**(esp_netif_t **esp_netif*, esp_ip6_addr_t **if_ip6*)

Get interface global IPv6 address.

If the specified interface is up and a preferred global IPv6 address has been created for the interface, return a copy of it.

Return

- ESP_OK
- ESP_FAIL If interface is down, does not have a global IPv6 address, or the global IPv6 address is not a preferred address.

Parameters

- `esp_netif`: Handle to esp-netif instance
- `if_ip6`: IPv6 information will be returned in this argument if successful.

void **esp_netif_set_ip4_addr**(esp_ip4_addr_t **addr*, uint8_t *a*, uint8_t *b*, uint8_t *c*, uint8_t *d*)

Sets IPv4 address to the specified octets.

Parameters

- `addr`: IP address to be set

- a: the first octet (127 for IP 127.0.0.1)
- b:
- c:
- d:

char *esp_ip4addr_ntoa(const esp_ip4_addr_t *addr, char *buf, int buflen)

Converts numeric IP address into decimal dotted ASCII representation.

Return either pointer to buf which now holds the ASCII representation of addr or NULL if buf was too small

Parameters

- addr: ip address in network order to convert
- buf: target buffer where the string is stored
- buflen: length of buf

uint32_t esp_ip4addr_aton(const char *addr)

Ascii internet address interpretation routine The value returned is in network order.

Return ip address in network order

Parameters

- addr: IP address in ascii representation (e.g. "127.0.0.1")

esp_netif_iodriver_handle esp_netif_get_io_driver(esp_netif_t *esp_netif)

Gets media driver handle for this esp-netif instance.

Return opaque pointer of related IO driver

Parameters

- esp_netif: Handle to esp-netif instance

esp_netif_t *esp_netif_get_handle_from_ifkey(const char *if_key)

Searches over a list of created objects to find an instance with supplied if key.

Return Handle to esp-netif instance

Parameters

- if_key: Textual description of network interface

esp_netif_flags_t esp_netif_get_flags(esp_netif_t *esp_netif)

Returns configured flags for this interface.

Return Configuration flags

Parameters

- `esp_netif`: Handle to esp-netif instance

`const` char *`esp_netif_get_ifkey`(esp_netif_t *_esp_netif_)

Returns configured interface key for this esp-netif instance.

Return Textual description of related interface

Parameters

- `esp_netif`: Handle to esp-netif instance

`const` char *`esp_netif_get_desc`(esp_netif_t *_esp_netif_)

Returns configured interface type for this esp-netif instance.

Return Enumerated type of this interface, such as station, AP, ethernet

Parameters

- `esp_netif`: Handle to esp-netif instance

int32_t `esp_netif_get_event_id`(esp_netif_t *_esp_netif_, esp_netif_ip_event_type_t _event_type_)

Returns configured event for this esp-netif instance and supplied event type.

Return specific event id which is configured to be raised if the interface lost or acquired IP address -1 if supplied event_type is not known

Parameters

- `esp_netif`: Handle to esp-netif instance
- `event_type`: (either get or lost IP)

esp_netif_t *`esp_netif_next`(esp_netif_t *_esp_netif_)

Iterates over list of interfaces. Returns first netif if NULL given as parameter.

Return First netif from the list if supplied parameter is NULL, next one otherwise

Parameters

- `esp_netif`: Handle to esp-netif instance

size_t `esp_netif_get_nr_of_ifs`(void)

Returns number of registered esp_netif objects.

Return Number of esp_netifs

WiFi default API reference

Header File

- esp_wifi/include/esp_wifi_default.h

Functions

esp_err_t **esp_netif_attach_wifi_station**(esp_netif_t *esp_netif)

Attaches wifi station interface to supplied netif.

Return

- ESP_OK on success
- ESP_FAIL if attach failed

Parameters

- **esp_netif**: instance to attach the wifi station to

esp_err_t **esp_netif_attach_wifi_ap**(esp_netif_t *esp_netif)

Attaches wifi soft AP interface to supplied netif.

Return

- ESP_OK on success
- ESP_FAIL if attach failed

Parameters

- **esp_netif**: instance to attach the wifi AP to

esp_err_t **esp_wifi_set_default_wifi_sta_handlers**(void)

Sets default wifi event handlers for STA interface.

Return

- ESP_OK on success, error returned from esp_event_handler_register if failed

esp_err_t **esp_wifi_set_default_wifi_ap_handlers**(void)

Sets default wifi event handlers for STA interface.

Return

- ESP_OK on success, error returned from esp_event_handler_register if failed

esp_err_t **esp_wifi_clear_default_wifi_driver_and_handlers**(void **esp_netif*)

Clears default wifi event handlers for supplied network interface.

Return

- ESP_OK on success, error returned from esp_event_handler_register if failed

Parameters

- **esp_netif**: instance of corresponding if object

esp_netif_t ***esp_netif_create_default_wifi_ap**(void)

Creates default WIFI AP. In case of any init error this API aborts.

Return pointer to esp-netif instance

esp_netif_t ***esp_netif_create_default_wifi_sta**(void)

Creates default WIFI STA. In case of any init error this API aborts.

Return pointer to esp-netif instance

esp_err_t **esp_netif_create_default_wifi_mesh_netifs**(esp_netif_t ***p_netif_sta*, esp_netif_t ***p_netif_ap*)

Creates default STA and AP network interfaces for esp-mesh.

Both netifs are almost identical to the default station and softAP, but with DHCP client and server disabled. Please note that the DHCP client is typically enabled only if the device is promoted to a root node.

Returns created interfaces which could be ignored setting parameters to NULL if an application code does not need to save the interface instances for further processing.

Return ESP_OK on success

Parameters

- **p_netif_sta**: pointer where the resultant STA interface is saved (if non NULL)
- **p_netif_ap**: pointer where the resultant AP interface is saved (if non NULL)

TCP/IP 适配器迁移指南

[英文]

TCP/IP 适配器是在 IDF V4.1 之前使用的网络接口抽象组件。本文档概述了从 tcpip_adapter 移出至其后继者*ESP-NETIF* 的过程。

更新网络连接代码

网络软件栈初始化

只需将 `tcpip_adapter_init()` 替换位 `esp_netif_init()`。请注意，*ESP-NETIF* 初始化 API 可返回标准错误代码，还可以使用 `esp_netif_deinit()` 进行去初始化。

此外，还需将 `#include "tcpip_adapter.h"` 替换为 `#include "esp_netif.h"`。

创建网络接口

TCP/IP 适配器静态定义了三个接口：

- Wi-Fi Station
- Wi-Fi AP
- 以太网

网络接口的设计应严格参考*ESP-NETIF*，以使其能够连接到 TCP/IP 软件栈。例如，在 TCP/IP 软件栈和事件循环初始化完成后，Wi-Fi 的初始化代码必须显示调用 `esp_netif_create_default_wifi_sta();` 或 `esp_netif_create_default_wifi_ap();` 。请参阅这三个接口的初始化代码示例：

- Wi-Fi Station：examples/wifi/getting_started/station/main/station_example_main.c
- Wi-Fi AP：examples/wifi/getting_started/softAP/main/softap_example_main.c
- 以太网 examples/ethernet/basic/main/ethernet_example_main.c

更换其他 tcpip_adapter API

所有 tcpip_adapter 函数都有对应的 esp-netif。具体请见 esp_netif 的内容：

- Setters/Getters
- DHCP
- DNS
- IP address

默认事件处理程序

事件处理程序已经从 tcpip_adapter 移动到相应的驱动程序代码。从应用程序的角度来看，这不会带来任何影响，所有事件仍以相同的方式处理。请注意，在与 IP 相关的事件处理程序中，应用程序代码通常以 esp-netif 结构体的形式接收 IP 地址（不是 LwIP 结构，但兼容二进制格式）。这是打印地址的首选方式：

```
ESP_LOGI(TAG, "got ip:" IPSTR "\n", IP2STR(&event->ip_info.ip));
```

而不是

```
ESP_LOGI(TAG, "got ip:%s\n", ip4addr_ntoa(&event->ip_info.ip));
```

由于 `ip4addr_ntoa()` 为 LwIP API，因此 esp-netif 还提供了替代函数 `esp_ip4addr_ntoa()`，但整体而言仍推荐上述方法。

IP 地址

推荐使用 esp-netif 定义的 IP 结构。请注意，在启用默认兼容性时，LwIP 结构体仍然可以工作。* esp-netif IP address definitions

下一步

为了移植应用程序使其可以使用*ESP-NETIF* 还需完成的步骤包括：在组件配置中禁用 tcpip_adapter 兼容层。方法为：`ESP NETIF Adapter -> Enable backward compatible tcpip_adapter interface`，并检查工程是否编译成功。TCP/IP 适配器涉及大量依赖项，这一步可能有助于将应用程序与使用特定 TCP/IP 软件栈的 API 分离开来。

ESP-NETIF Custom I/O Driver

This section outlines implementing a new I/O driver with esp-netif connection capabilities. By convention the I/O driver has to register itself as an esp-netif driver and thus holds a dependency on esp-netif component and is responsible for providing data path functions, post-attach callback and in most cases also default event handlers to define network interface actions based on driver's lifecycle transitions.

Packet input/output

As shown in the diagram, the following three API functions for the packet data path must be defined for connecting with esp-netif:

- *esp_netif_transmit()*
- *esp_netif_free_rx_buffer()*
- *esp_netif_receive()*

The first two functions for transmitting and freeing the rx buffer are provided as callbacks, i.e. they get called from esp-netif (and its underlying TCP/IP stack) and I/O driver provides their implementation.

The receiving function on the other hand gets called from the I/O driver, so that the driver's code simply calls *esp_netif_receive()* on a new data received event.

Post attach callback

A final part of the network interface initialization consists of attaching the esp-netif instance to the I/O driver, by means of calling the following API:

```
esp_err_t esp_netif_attach(esp_netif_t *esp_netif, esp_netif_iodriver_handle driver_
↪handle);
```

It is assumed that the `esp_netif_iodriver_handle` is a pointer to driver's object, a struct derived from `struct esp_netif_driver_base_s`, so that the first member of I/O driver structure must be this base structure with pointers to

- post-attach function callback
- related esp-netif instance

As a consequence the I/O driver has to create an instance of the struct per below:

```
typedef struct my_netif_driver_s {
        esp_netif_driver_base_t base;           /*!< base structure reserved as esp-
↪netif driver */
        driver_impl             *h;             /*!< handle of driver implementation */
    } my_netif_driver_t;
```

with actual values of `my_netif_driver_t::base.post_attach` and the actual drivers handle `my_netif_driver_t::h`. So when the *esp_netif_attach()* gets called from the initialization code, the post-attach callback from I/O driver's code gets executed to mutually register callbacks between esp-netif and I/O driver instances. Typically the driver is started as well in the post-attach callback. An example of a simple post-attach callback is outlined below:

```
static esp_err_t my_post_attach_start(esp_netif_t * esp_netif, void * args)
{
    my_netif_driver_t *driver = args;
    const esp_netif_driver_ifconfig_t driver_ifconfig = {
            .driver_free_rx_buffer = my_free_rx_buf,
            .transmit = my_transmit,
            .handle = driver->driver_impl
    };
    driver->base.netif = esp_netif;
    ESP_ERROR_CHECK(esp_netif_set_driver_config(esp_netif, &driver_ifconfig));
    my_driver_start(driver->driver_impl);
    return ESP_OK;
}
```

Default handlers

I/O drivers also typically provide default definitions of lifecycle behaviour of related network interfaces based on state transitions of I/O drivers. For example *driver start* -> *network start*, etc. An example of such a default handler is provided below:

```
esp_err_t my_driver_netif_set_default_handlers(my_netif_driver_t *driver, esp_netif_t *␣
↪esp_netif)
{
    driver_set_event_handler(driver->driver_impl, esp_netif_action_start, MY_DRV_EVENT_
↪START, esp_netif);
    driver_set_event_handler(driver->driver_impl, esp_netif_action_stop, MY_DRV_EVENT_
↪STOP, esp_netif);
    return ESP_OK;
}
```

Network stack connection

The packet data path functions for transmitting and freeing the rx buffer (defined in the I/O driver) are called from the esp-netif, specifically from its TCP/IP stack connecting layer. The following API reference outlines these network stack interaction with the esp-netif.

Header File

- esp_netif/include/esp_netif_net_stack.h

Functions

esp_netif_t *__esp_netif_get_handle_from_netif_impl__(void *_dev_)

Returns esp-netif handle.

Return handle to related esp-netif instance

Parameters

- `dev`: opaque ptr to network interface of specific TCP/IP stack

void *__esp_netif_get_netif_impl__(esp_netif_t *_esp_netif_)

Returns network stack specific implementation handle (if supported)

Note that it is not supported to acquire PPP netif impl pointer and this function will return NULL for esp_netif instances configured to PPP mode

Return handle to related network stack netif handle

Parameters

- `esp_netif`: Handle to esp-netif instance

esp_err_t **esp_netif_transmit**(esp_netif_t *_esp_netif_, void *_data_, size_t _len_)

Outputs packets from the TCP/IP stack to the media to be transmitted.

This function gets called from network stack to output packets to IO driver.

Return ESP_OK on success, an error passed from the I/O driver otherwise

Parameters

- `esp_netif`: Handle to esp-netif instance
- `data`: Data to be transmitted
- `len`: Length of the data frame

void **esp_netif_free_rx_buffer**(void *_esp_netif_, void *_buffer_)

Free the rx buffer allocated by the media driver.

This function gets called from network stack when the rx buffer to be freed in IO driver context, i.e. to deallocate a buffer owned by io driver (when data packets were passed to higher levels to avoid copying)

Parameters

- `esp_netif`: Handle to esp-netif instance
- `buffer`: Rx buffer pointer

TCP/IP 套接字 API 的示例代码存放在 ESP-IDF 示例项目的 protocols/sockets 目录下。

2.2.4 应用层协议

应用层网络协议（IP 网络层协议之上）的相关文档存放在应用层协议。

2.3 外设 API

[English]

2.3.1 Analog to Digital Converter

Overview

The ESP32 integrates two 12-bit SAR (Successive Approximation Register) ADCs supporting a total of 18 measurement channels (analog enabled pins).

The ADC driver API supports ADC1 (8 channels, attached to GPIOs 32 - 39), and ADC2 (10 channels, attached to GPIOs 0, 2, 4, 12 - 15 and 25 - 27). However, the usage of ADC2 has some restrictions for the application:

1. ADC2 is used by the Wi-Fi driver. Therefore the application can only use ADC2 when the Wi-Fi driver has not started.
2. Some of the ADC2 pins are used as strapping pins (GPIO 0, 2, 15) thus cannot be used freely. Such is the case in the following official Development Kits:

- *ESP32 DevKitC*: GPIO 0 cannot be used due to external auto program circuits.
- *ESP-WROVER-KIT*: GPIO 0, 2, 4 and 15 cannot be used due to external connections for different purposes.

Configuration and Reading ADC

The ADC should be configured before reading is taken.

- For ADC1, configure desired precision and attenuation by calling functions *adc1_config_width()* and *adc1_config_channel_atten()*.
- For ADC2, configure the attenuation by *adc2_config_channel_atten()*. The reading width of ADC2 is configured every time you take the reading.

Attenuation configuration is done per channel, see *adc1_channel_t* and *adc2_channel_t*, set as a parameter of above functions.

Then it is possible to read ADC conversion result with *adc1_get_raw()* and *adc2_get_raw()*. Reading width of ADC2 should be set as a parameter of *adc2_get_raw()* instead of in the configuration functions.

注解: Since the ADC2 is shared with the WIFI module, which has higher priority, reading operation of *adc2_get_raw()* will fail between *esp_wifi_start()* and *esp_wifi_stop()*. Use the return code to see whether the reading is successful.

It is also possible to read the internal hall effect sensor via ADC1 by calling dedicated function *hall_sensor_read()*. Note that even the hall sensor is internal to ESP32, reading from it uses channels 0 and 3 of ADC1 (GPIO 36 and 39). Do not connect anything else to these pins and do not change their configuration. Otherwise it may affect the measurement of low value signal from the sensor.

This API provides convenient way to configure ADC1 for reading from *ULP*. To do so, call function *adc1_ulp_enable()* and then set precision and attenuation as discussed above.

There is another specific function *adc2_vref_to_gpio()* used to route internal reference voltage to a GPIO pin. It comes handy to calibrate ADC reading and this is discussed in section *Minimizing Noise*.

Application Examples

Reading voltage on ADC1 channel 0 (GPIO 36):

```
#include <driver/adc.h>

...

    adc1_config_width(ADC_WIDTH_BIT_12);
    adc1_config_channel_atten(ADC1_CHANNEL_0,ADC_ATTEN_DB_0);
    int val = adc1_get_raw(ADC1_CHANNEL_0);
```

The input voltage in above example is from 0 to 1.1V (0 dB attenuation). The input range can be extended by setting higher attenuation, see *adc_atten_t*. An example using the ADC driver including calibration (discussed below) is available in esp-idf: peripherals/adc

Reading voltage on ADC2 channel 7 (GPIO 27):

```
#include <driver/adc.h>

...

    int read_raw;
    adc2_config_channel_atten( ADC2_CHANNEL_7, ADC_ATTEN_0db );

    esp_err_t r = adc2_get_raw( ADC2_CHANNEL_7, ADC_WIDTH_12Bit, &read_raw);
    if ( r == ESP_OK ) {
        printf("%d\n", read_raw );
    } else if ( r == ESP_ERR_TIMEOUT ) {
        printf("ADC2 used by Wi-Fi.\n");
    }
```

The reading may fail due to collision with Wi-Fi, should check it. An example using the ADC2 driver to read the output of DAC is available in esp-idf: peripherals/adc2

Reading the internal hall effect sensor:

```
#include <driver/adc.h>

...

    adc1_config_width(ADC_WIDTH_BIT_12);
    int val = hall_sensor_read();
```

The value read in both these examples is 12 bits wide (range 0-4095).

Minimizing Noise

The ESP32 ADC can be sensitive to noise leading to large discrepancies in ADC readings. To minimize noise, users may connect a 0.1uF capacitor to the ADC input pad in use. Multisampling may also be used to further mitigate the effects of noise.

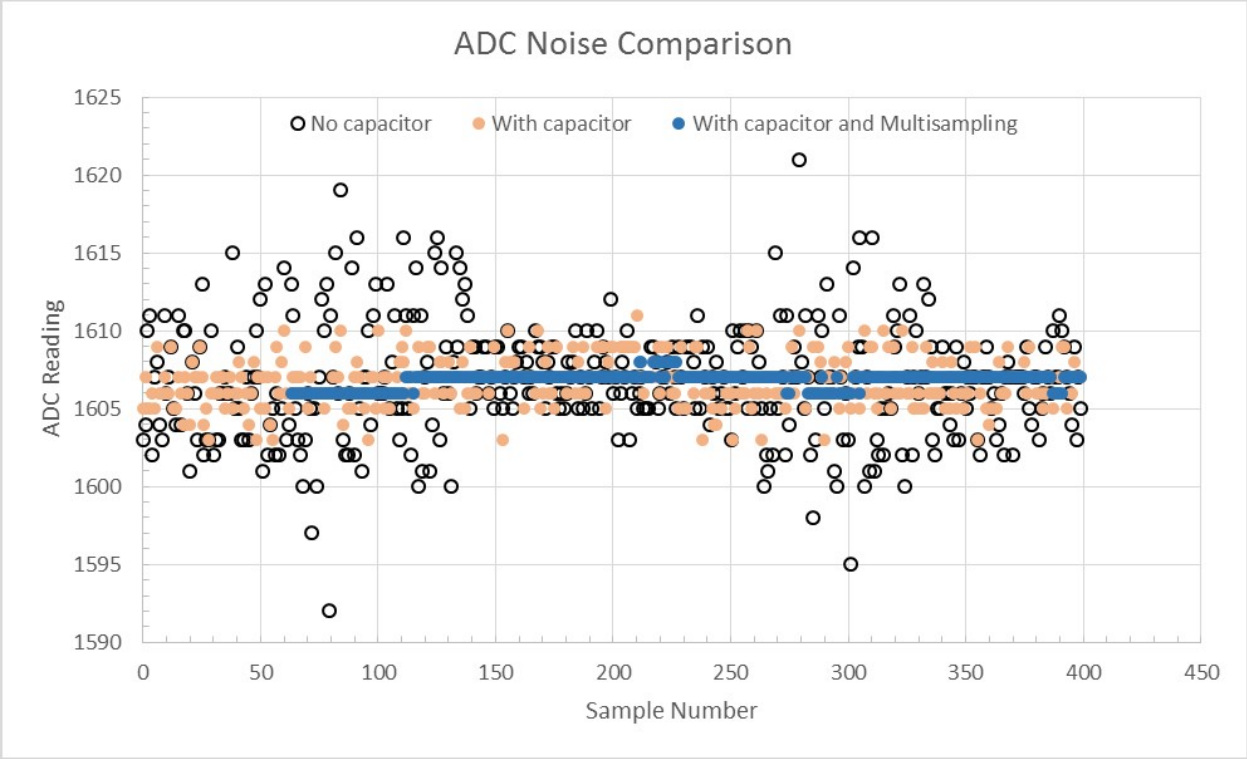


图 4: Graph illustrating noise mitigation using capacitor and multisampling of 64 samples.

ADC Calibration

The esp_adc_cal/include/esp_adc_cal.h API provides functions to correct for differences in measured voltages caused by variation of ADC reference voltages (Vref) between chips. Per design the ADC reference

voltage is 1100mV, however the true reference voltage can range from 1000mV to 1200mV amongst different ESP32s.

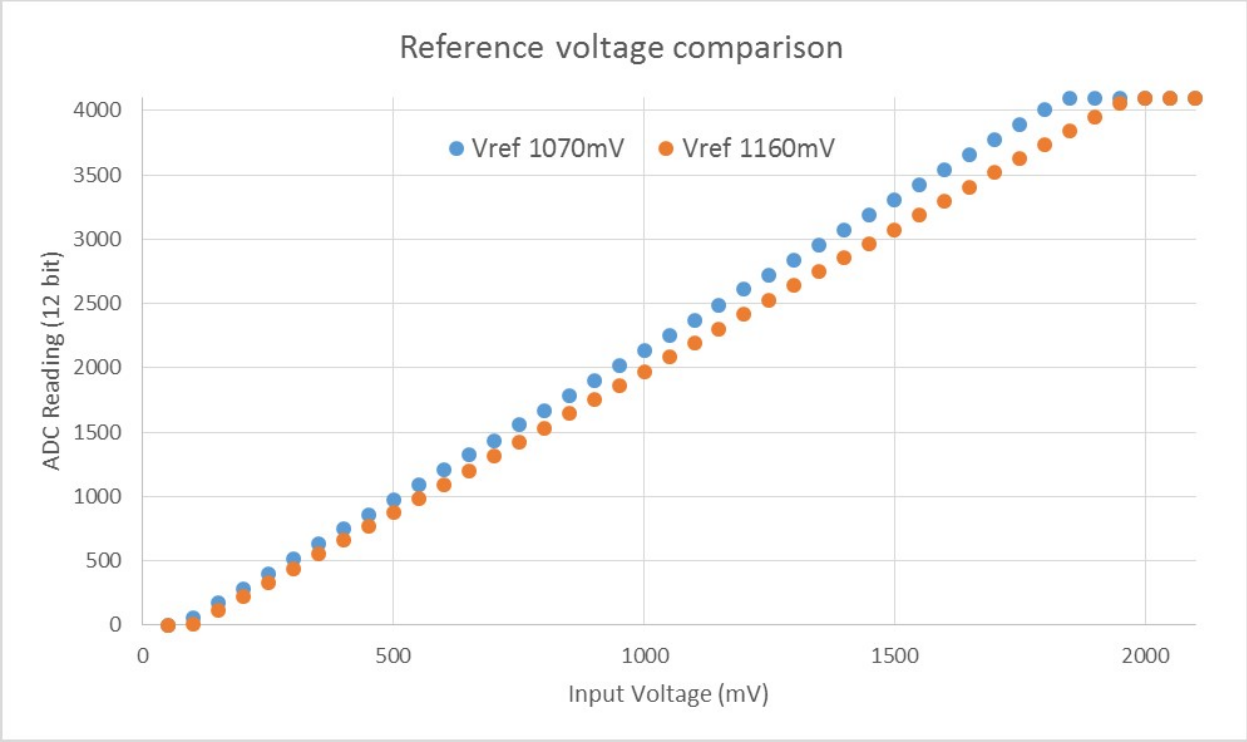


图 5: Graph illustrating effect of differing reference voltages on the ADC voltage curve.

Correcting ADC readings using this API involves characterizing one of the ADCs at a given attenuation to obtain a characteristics curve (ADC-Voltage curve) that takes into account the difference in ADC reference voltage. The characteristics curve is in the form of `y = coeff_a * x + coeff_b` and is used to convert ADC readings to voltages in mV. Calculation of the characteristics curve is based on calibration values which can be stored in eFuse or provided by the user.

Calibration Values

Calibration values are used to generate characteristic curves that account for the unique ADC reference voltage of a particular ESP32. There are currently three sources of calibration values. The availability of these calibration values will depend on the type and production date of the ESP32 chip/module.

- **Two Point** values represent each of the ADCs' readings at 150mV and 850mV. To obtain more accurate calibration results these values should be measured by user and burned into eFuse `BLOCK3`.
- **eFuse Vref** represents the true ADC reference voltage. This value is measured and burned into eFuse `BLOCK0` during factory calibration.
- **Default Vref** is an estimate of the ADC reference voltage provided by the user as a parameter during characterization. If Two Point or eFuse Vref values are unavailable, **Default Vref** will be used.

Individual measurement and burning of the **eFuse Vref** has been applied to ESP32-D0WD and ESP32-D0WDQ6 chips produced on/after the 1st week of 2018. Such chips may be recognized by date codes on/later than 012018 (see Line 4 on figure below).

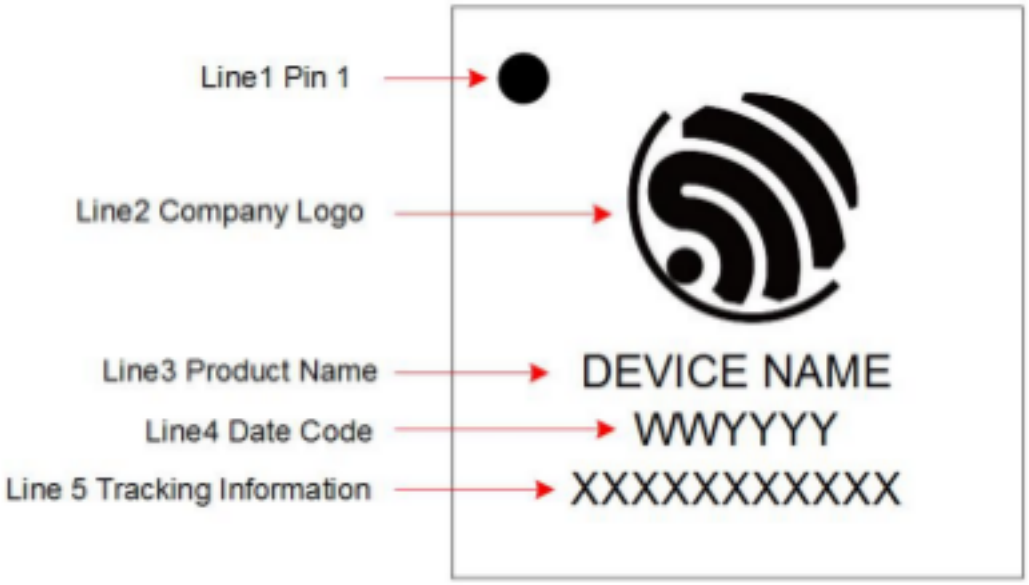


图 6: ESP32 Chip Surface Marking

If you would like to purchase chips or modules with calibration, double check with distributor or Espressif directly.

If you are unable to check the date code (i.e. the chip may be enclosed inside a canned module, etc.), you can still verify if **eFuse Vref** is present by running espefuse.py tool with `adc_info` parameter

```
$IDF_PATH/components/esptool_py/esptool/espefuse.py --port /dev/ttyUSB0 adc_info
```

Replace `/dev/ttyUSB0` with ESP32 board' s port name.

A chip that has specific **eFuse Vref** value programmed (in this case 1093mV) will be reported as follows:

```
ADC VRef calibration: 1093mV
```

In another example below the **eFuse Vref** is not programmed:

```
ADC VRef calibration: None (1100mV nominal)
```

For a chip with two point calibration the message will look similar to:

```
ADC VRef calibration: 1149mV
ADC readings stored in efuse BLK3:
    ADC1 Low reading  (150mV): 306
```

(下页继续)

(续上页)

```
    ADC1 High reading (850mV): 3153
    ADC2 Low reading  (150mV): 389
    ADC2 High reading (850mV): 3206
```

Application Example

For a full example see esp-idf: peripherals/adc

Characterizing an ADC at a particular attenuation:

```
#include "driver/adc.h"
#include "esp_adc_cal.h"

...

    //Characterize ADC at particular atten
    esp_adc_cal_characteristics_t *adc_chars = calloc(1, sizeof(esp_adc_cal_
↪characteristics_t));
    esp_adc_cal_value_t val_type = esp_adc_cal_characterize(unit, atten, ADC_WIDTH_BIT_
↪12, DEFAULT_VREF, adc_chars);
    //Check type of calibration value used to characterize ADC
    if (val_type == ESP_ADC_CAL_VAL_EFUSE_VREF) {
        printf("eFuse Vref");
    } else if (val_type == ESP_ADC_CAL_VAL_EFUSE_TP) {
        printf("Two Point");
    } else {
        printf("Default");
    }
```

Reading an ADC then converting the reading to a voltage:

```
#include "driver/adc.h"
#include "esp_adc_cal.h"

...
    uint32_t reading =  adc1_get_raw(ADC1_CHANNEL_5);
    uint32_t voltage = esp_adc_cal_raw_to_voltage(reading, adc_chars);
```

Routing ADC reference voltage to GPIO, so it can be manually measured (for **Default Vref**):

```
#include "driver/adc.h"

...

    esp_err_t status = adc2_vref_to_gpio(GPIO_NUM_25);
    if (status == ESP_OK) {
        printf("v_ref routed to GPIO\n");
    } else {
        printf("failed to route v_ref\n");
    }
```

GPIO Lookup Macros

There are macros available to specify the GPIO number of a ADC channel, or vice versa. e.g.

1. `ADC1_CHANNEL_0_GPIO_NUM` is the GPIO number of ADC1 channel 0 (36);
2. `ADC1_GPIO32_CHANNEL` is the ADC1 channel number of GPIO 32 (ADC1 channel 4).

API Reference

This reference covers three components:

- *ADC driver*
- *ADC Calibration*
- *GPIO Lookup Macros*

ADC driver

Header File

- driver/include/driver/adc.h

Functions

esp_err_t **adc1_pad_get_io_num**(*adc1_channel_t channel*, *gpio_num_t *gpio_num*)

Get the GPIO number of a specific ADC1 channel.

Return

- ESP_OK if success

- ESP_ERR_INVALID_ARG if channel not valid

Parameters

- `channel`: Channel to get the GPIO number
- `gpio_num`: output buffer to hold the GPIO number

esp_err_t **adc1_config_width**(*adc_bits_width_t width_bit*)

Configure ADC1 capture width, meanwhile enable output invert for ADC1. The configuration is for all channels of ADC1.

Return

- ESP_OK success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `width_bit`: Bit capture width for ADC1

esp_err_t **adc_set_data_width**(*adc_unit_t adc_unit*, *adc_bits_width_t width_bit*)

Configure ADC capture width.

Return

- ESP_OK success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `adc_unit`: ADC unit index
- `width_bit`: Bit capture width for ADC unit.

esp_err_t **adc1_config_channel_atten**(*adc1_channel_t channel*, *adc_atten_t atten*)

Set the attenuation of a particular channel on ADC1, and configure its associated GPIO pin mux.

The default ADC full-scale voltage is 1.1 V. To read higher voltages (up to the pin maximum voltage, usually 3.3 V) requires setting >0 dB signal attenuation for that ADC channel.

Note For any given channel, this function must be called before the first time adc1_get_raw() is called for that channel.

Note This function can be called multiple times to configure multiple ADC channels simultaneously. adc1_get_raw() can then be called for any configured channel.

When VDD_A is 3.3 V:

- 0 dB attenuation (ADC_ATTEN_DB_0) gives full-scale voltage 1.1 V

- 2.5 dB attenuation (ADC_ATTEN_DB_2_5) gives full-scale voltage 1.5 V
- 6 dB attenuation (ADC_ATTEN_DB_6) gives full-scale voltage 2.2 V
- 11 dB attenuation (ADC_ATTEN_DB_11) gives full-scale voltage 3.9 V (see note below)

Due to ADC characteristics, most accurate results are obtained within the following approximate voltage ranges:

Note The full-scale voltage is the voltage corresponding to a maximum reading (depending on ADC1 configured bit width, this value is: 4095 for 12-bits, 2047 for 11-bits, 1023 for 10-bits, 511 for 9 bits.)

Note At 11 dB attenuation the maximum voltage is limited by VDD_A, not the full scale voltage.

- 0 dB attenuation (ADC_ATTEN_DB_0) between 100 and 950 mV
- 2.5 dB attenuation (ADC_ATTEN_DB_2_5) between 100 and 1250 mV
- 6 dB attenuation (ADC_ATTEN_DB_6) between 150 to 1750 mV
- 11 dB attenuation (ADC_ATTEN_DB_11) between 150 to 2450 mV

For maximum accuracy, use the ADC calibration APIs and measure voltages within these recommended ranges.

Return

- ESP_OK success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `channel`: ADC1 channel to configure
- `atten`: Attenuation level

int **adc1_get_raw**(*adc1_channel_t channel*)

Take an ADC1 reading from a single channel.

Note When the power switch of SARADC1, SARADC2, HALL sensor and AMP sensor is turned on, the input of GPIO36 and GPIO39 will be pulled down for about 80ns. When enabling power for any of these peripherals, ignore input from GPIO36 and GPIO39. Please refer to section 3.11 of 'ECO_and_Workarounds_for_Bugs_in_ESP32' for the description of this issue.

Note Call adc1_config_width() before the first time this function is called.

Note For any given channel, adc1_config_channel_atten(channel) must be called before the first time this function is called. Configuring a new channel does not prevent a previously configured channel from being read.

Return

- -1: Parameter error
- Other: ADC1 channel reading.

Parameters

- `channel`: ADC1 channel to read

void **adc_power_on**(void)

Enable ADC power.

void **adc_power_off**(void)

Power off SAR ADC This function will force power down for ADC.

esp_err_t **adc_gpio_init**(*adc_unit_t adc_unit*, *adc_channel_t channel*)

Initialize ADC pad.

Return

- ESP_OK success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `adc_unit`: ADC unit index
- `channel`: ADC channel index

esp_err_t **adc_set_data_inv**(*adc_unit_t adc_unit*, bool *inv_en*)

Set ADC data invert.

Return

- ESP_OK success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `adc_unit`: ADC unit index
- `inv_en`: whether enable data invert

esp_err_t **adc_set_clk_div**(uint8_t *clk_div*)

Set ADC source clock.

Return

- ESP_OK success

Parameters

- `clk_div`: ADC clock divider, ADC clock is divided from APB clock

esp_err_t **adc_set_i2s_data_source**(*adc_i2s_source_t src*)

Set I2S data source.

Return

- ESP_OK success

Parameters

- `src`: I2S DMA data source, I2S DMA can get data from digital signals or from ADC.

esp_err_t **adc_i2s_mode_init**(*adc_unit_t adc_unit*, *adc_channel_t channel*)

Initialize I2S ADC mode.

Return

- ESP_OK success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `adc_unit`: ADC unit index
- `channel`: ADC channel index

void **adc1_ulp_enable**(void)

Configure ADC1 to be usable by the ULP.

This function reconfigures ADC1 to be controlled by the ULP. Effect of this function can be reverted using adc1_get_raw function.

Note that adc1_config_channel_atten, adc1_config_width functions need to be called to configure ADC1 channels, before ADC1 is used by the ULP.

int **hall_sensor_read**(void)

Read Hall Sensor.

Note When the power switch of SARADC1, SARADC2, HALL sensor and AMP sensor is turned on, the input of GPIO36 and GPIO39 will be pulled down for about 80ns. When enabling power for any of these peripherals, ignore input from GPIO36 and GPIO39. Please refer to section 3.11 of 'ECO_and_Workarounds_for_Bugs_in_ESP32' for the description of this issue.

Note The Hall Sensor uses channels 0 and 3 of ADC1. Do not configure these channels for use as ADC channels.

Note The ADC1 module must be enabled by calling adc1_config_width() before calling hall_sensor_read(). ADC1 should be configured for 12 bit readings, as the hall sensor readings are low values and do not cover the full range of the ADC.

Return The hall sensor reading.

esp_err_t **adc2_pad_get_io_num**(*adc2_channel_t channel*, *gpio_num_t *gpio_num*)

Get the GPIO number of a specific ADC2 channel.

Return

- ESP_OK if success
- ESP_ERR_INVALID_ARG if channel not valid

Parameters

- `channel`: Channel to get the GPIO number
- `gpio_num`: output buffer to hold the GPIO number

esp_err_t **adc2_config_channel_atten**(*adc2_channel_t channel*, *adc_atten_t atten*)

Configure the ADC2 channel, including setting attenuation.

The default ADC full-scale voltage is 1.1 V. To read higher voltages (up to the pin maximum voltage, usually 3.3 V) requires setting >0 dB signal attenuation for that ADC channel.

Note This function also configures the input GPIO pin mux to connect it to the ADC2 channel. It must be called before calling `adc2_get_raw()` for this channel.

When VDD_A is 3.3 V:

- 0 dB attenuation (ADC_ATTEN_0db) gives full-scale voltage 1.1 V
- 2.5 dB attenuation (ADC_ATTEN_2_5db) gives full-scale voltage 1.5 V
- 6 dB attenuation (ADC_ATTEN_6db) gives full-scale voltage 2.2 V
- 11 dB attenuation (ADC_ATTEN_11db) gives full-scale voltage 3.9 V (see note below)

Note The full-scale voltage is the voltage corresponding to a maximum reading (depending on ADC2 configured bit width, this value is: 4095 for 12-bits, 2047 for 11-bits, 1023 for 10-bits, 511 for 9 bits.)

Note At 11 dB attenuation the maximum voltage is limited by VDD_A, not the full scale voltage.

Return

- ESP_OK success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `channel`: ADC2 channel to configure
- `atten`: Attenuation level

esp_err_t **adc2_get_raw**(*adc2_channel_t channel*, *adc_bits_width_t width_bit*, int *raw_out)

Take an ADC2 reading on a single channel.

Note When the power switch of SARADC1, SARADC2, HALL sensor and AMP sensor is turned on, the input of GPIO36 and GPIO39 will be pulled down for about 80ns. When enabling power for any of these peripherals, ignore input from GPIO36 and GPIO39. Please refer to section 3.11 of 'ECO_and_Workarounds_for_Bugs_in_ESP32' for the description of this issue.

Note For a given channel, `adc2_config_channel_atten()` must be called before the first time this function is called. If Wi-Fi is started via `esp_wifi_start()`, this function will always fail with `ESP_ERR_TIMEOUT`.

Return

- ESP_OK if success
- ESP_ERR_TIMEOUT the WIFI is started, using the ADC2

Parameters

- `channel`: ADC2 channel to read
- `width_bit`: Bit capture width for ADC2
- `raw_out`: the variable to hold the output data.

esp_err_t **adc2_vref_to_gpio**(*gpio_num_t gpio*)

Output ADC2 reference voltage to GPIO 25 or 26 or 27.

This function utilizes the testing mux exclusive to ADC 2 to route the reference voltage one of ADC2's channels. Supported GPIOs are GPIOs 25, 26, and 27. This refernce voltage can be manually read from the pin and used in the esp_adc_cal component.

Return

- ESP_OK: v_ref successfully routed to selected GPIO
- ESP_ERR_INVALID_ARG: Unsupported GPIO

Parameters

- `gpio`: GPIO number (GPIOs 25, 26 and 27 are supported)

Macros

`ADC_ATTEN_0db`

`ADC_ATTEN_2_5db`

`ADC_ATTEN_6db`

`ADC_ATTEN_11db`

ADC_WIDTH_9Bit

ADC_WIDTH_10Bit

ADC_WIDTH_11Bit

ADC_WIDTH_12Bit

Enumerations

enum adc1_channel_t

Values:

ADC1_CHANNEL_0 = 0

ADC1 channel 0 is GPIO36 (ESP32), GPIO1 (ESP32-S2)

ADC1_CHANNEL_1

ADC1 channel 1 is GPIO37 (ESP32), GPIO2 (ESP32-S2)

ADC1_CHANNEL_2

ADC1 channel 2 is GPIO38 (ESP32), GPIO3 (ESP32-S2)

ADC1_CHANNEL_3

ADC1 channel 3 is GPIO39 (ESP32), GPIO4 (ESP32-S2)

ADC1_CHANNEL_4

ADC1 channel 4 is GPIO32 (ESP32), GPIO5 (ESP32-S2)

ADC1_CHANNEL_5

ADC1 channel 5 is GPIO33 (ESP32), GPIO6 (ESP32-S2)

ADC1_CHANNEL_6

ADC1 channel 6 is GPIO34 (ESP32), GPIO7 (ESP32-S2)

ADC1_CHANNEL_7

ADC1 channel 7 is GPIO35 (ESP32), GPIO8 (ESP32-S2)

enum adc2_channel_t

Values:

ADC2_CHANNEL_0 = 0

ADC2 channel 0 is GPIO4 (ESP32), GPIO11 (ESP32-S2)

ADC2_CHANNEL_1

ADC2 channel 1 is GPIO0 (ESP32), GPIO12 (ESP32-S2)

ADC2_CHANNEL_2

ADC2 channel 2 is GPIO2 (ESP32), GPIO13 (ESP32-S2)

ADC2_CHANNEL_3

ADC2 channel 3 is GPIO15 (ESP32), GPIO14 (ESP32-S2)

ADC2_CHANNEL_4
: ADC2 channel 4 is GPIO13 (ESP32), GPIO15 (ESP32-S2)

ADC2_CHANNEL_5
: ADC2 channel 5 is GPIO12 (ESP32), GPIO16 (ESP32-S2)

ADC2_CHANNEL_6
: ADC2 channel 6 is GPIO14 (ESP32), GPIO17 (ESP32-S2)

ADC2_CHANNEL_7
: ADC2 channel 7 is GPIO27 (ESP32), GPIO18 (ESP32-S2)

ADC2_CHANNEL_8
: ADC2 channel 8 is GPIO25 (ESP32), GPIO19 (ESP32-S2)

ADC2_CHANNEL_9
: ADC2 channel 9 is GPIO26 (ESP32), GPIO20 (ESP32-S2)

ADC2_CHANNEL_MAX

enum adc_unit_t

Values:

ADC_UNIT_1 = 1
: SAR ADC 1

ADC_UNIT_2 = 2
: SAR ADC 2, not supported yet

ADC_UNIT_BOTH = 3
: SAR ADC 1 and 2, not supported yet

ADC_UNIT_ALTER = 7
: SAR ADC 1 and 2 alternative mode, not supported yet

ADC_UNIT_MAX

enum adc_i2s_encode_t

Values:

ADC_ENCODE_12BIT
: ADC to I2S data format, [15:12]-channel [11:0]-12 bits ADC data

ADC_ENCODE_11BIT
: ADC to I2S data format, [15]-1 [14:11]-channel [10:0]-11 bits ADC data

ADC_ENCODE_MAX

Header File

- soc/include/hal/adc_types.h

Enumerations

enum adc_channel_t

Values:

ADC_CHANNEL_0 = 0

ADC channel

ADC_CHANNEL_1

ADC channel

ADC_CHANNEL_2

ADC channel

ADC_CHANNEL_3

ADC channel

ADC_CHANNEL_4

ADC channel

ADC_CHANNEL_5

ADC channel

ADC_CHANNEL_6

ADC channel

ADC_CHANNEL_7

ADC channel

ADC_CHANNEL_8

ADC channel

ADC_CHANNEL_9

ADC channel

ADC_CHANNEL_MAX

enum adc_atten_t

Values:

ADC_ATTEN_DB_0 = 0

The input voltage of ADC will be reduced to about 1/1

ADC_ATTEN_DB_2_5 = 1

The input voltage of ADC will be reduced to about 1/1.34

ADC_ATTEN_DB_6 = 2

The input voltage of ADC will be reduced to about 1/2

ADC_ATTEN_DB_11 = 3

The input voltage of ADC will be reduced to about 1/3.6

ADC_ATTEN_MAX

enum adc_i2s_source_t

Values:

ADC_I2S_DATA_SRC_IO_SIG = 0

I2S data from GPIO matrix signal

ADC_I2S_DATA_SRC_ADC = 1

I2S data from ADC

ADC_I2S_DATA_SRC_MAX

enum adc_bits_width_t

Values:

ADC_WIDTH_BIT_9 = 0

ADC capture width is 9Bit

ADC_WIDTH_BIT_10 = 1

ADC capture width is 10Bit

ADC_WIDTH_BIT_11 = 2

ADC capture width is 11Bit

ADC_WIDTH_BIT_12 = 3

ADC capture width is 12Bit

ADC_WIDTH_MAX

ADC Calibration

Header File

- esp_adc_cal/include/esp_adc_cal.h

Functions

esp_err_t **esp_adc_cal_check_efuse**(*esp_adc_cal_value_t value_type*)

Checks if ADC calibration values are burned into eFuse.

This function checks if ADC reference voltage or Two Point values have been burned to the eFuse of the current ESP32

Return

- ESP_OK: The calibration mode is supported in eFuse
- ESP_ERR_NOT_SUPPORTED: Error, eFuse values are not burned

- ESP_ERR_INVALID_ARG: Error, invalid argument (ESP_ADC_CAL_VAL_DEFAULT_VREF)

Parameters

- `value_type`: Type of calibration value (ESP_ADC_CAL_VAL_EFUSE_VREF or ESP_ADC_CAL_VAL_EFUSE_TP)

esp_adc_cal_value_t **esp_adc_cal_characterize**(*adc_unit_t adc_num*, *adc_atten_t atten*, *adc_bits_width_t bit_width*, uint32_t *default_vref*, *esp_adc_cal_characteristics_t* **chars*)

Characterize an ADC at a particular attenuation.

This function will characterize the ADC at a particular attenuation and generate the ADC-Voltage curve in the form of [y = coeff_a * x + coeff_b]. Characterization can be based on Two Point values, eFuse Vref, or default Vref and the calibration values will be prioritized in that order.

Note Two Point values and eFuse Vref can be enabled/disabled using menuconfig.

Return

- ESP_ADC_CAL_VAL_EFUSE_VREF: eFuse Vref used for characterization
- ESP_ADC_CAL_VAL_EFUSE_TP: Two Point value used for characterization (only in Linear Mode)
- ESP_ADC_CAL_VAL_DEFAULT_VREF: Default Vref used for characterization

Parameters

- `adc_num`: ADC to characterize (ADC_UNIT_1 or ADC_UNIT_2)
- `atten`: Attenuation to characterize
- `bit_width`: Bit width configuration of ADC
- `default_vref`: Default ADC reference voltage in mV (used if eFuse values is not available)
- `chars`: Pointer to empty structure used to store ADC characteristics

uint32_t **esp_adc_cal_raw_to_voltage**(uint32_t *adc_reading*, **const** *esp_adc_cal_characteristics_t* **chars*)

Convert an ADC reading to voltage in mV.

This function converts an ADC reading to a voltage in mV based on the ADC' s characteristics.

Note Characteristics structure must be initialized before this function is called (call esp_adc_cal_characterize())

Return Voltage in mV

Parameters

- `adc_reading`: ADC reading

- `chars`: Pointer to initialized structure containing ADC characteristics

esp_err_t **esp_adc_cal_get_voltage**(*adc_channel_t channel*, **const** *esp_adc_cal_characteristics_t* **chars*, uint32_t **voltage*)

Reads an ADC and converts the reading to a voltage in mV.

This function reads an ADC then converts the raw reading to a voltage in mV based on the characteristics provided. The ADC that is read is also determined by the characteristics.

Note The Characteristics structure must be initialized before this function is called (call esp_adc_cal_characterize())

Return

- ESP_OK: ADC read and converted to mV
- ESP_ERR_TIMEOUT: Error, timed out attempting to read ADC
- ESP_ERR_INVALID_ARG: Error due to invalid arguments

Parameters

- `channel`: ADC Channel to read
- `chars`: Pointer to initialized ADC characteristics structure
- `voltage`: Pointer to store converted voltage

Structures

struct esp_adc_cal_characteristics_t

Structure storing characteristics of an ADC.

Note Call esp_adc_cal_characterize() to initialize the structure

Public Members

adc_unit_t **adc_num**

ADC number

adc_atten_t **atten**

ADC attenuation

adc_bits_width_t **bit_width**

ADC bit width

uint32_t **coeff_a**

Gradient of ADC-Voltage curve

uint32_t **coeff_b**

Offset of ADC-Voltage curve

uint32_t **vref**

Vref used by lookup table

const uint32_t ***low_curve**

Pointer to low Vref curve of lookup table (NULL if unused)

const uint32_t ***high_curve**

Pointer to high Vref curve of lookup table (NULL if unused)

Enumerations

enum esp_adc_cal_value_t

Type of calibration value used in characterization.

Values:

ESP_ADC_CAL_VAL_EFUSE_VREF = 0

Characterization based on reference voltage stored in eFuse

ESP_ADC_CAL_VAL_EFUSE_TP = 1

Characterization based on Two Point values stored in eFuse

ESP_ADC_CAL_VAL_DEFAULT_VREF = 2

Characterization based on default reference voltage

GPIO Lookup Macros

Header File

- soc/esp32/include/soc/adc_channel.h

Macros

ADC1_GPIO36_CHANNEL

ADC1_CHANNEL_0_GPIO_NUM

ADC1_GPIO37_CHANNEL

ADC1_CHANNEL_1_GPIO_NUM

ADC1_GPIO38_CHANNEL

ADC1_CHANNEL_2_GPIO_NUM

ADC1_GPIO39_CHANNEL

ADC1_CHANNEL_3_GPIO_NUM

ADC1_GPIO32_CHANNEL

ADC1_CHANNEL_4_GPIO_NUM

ADC1_GPIO33_CHANNEL

ADC1_CHANNEL_5_GPIO_NUM

ADC1_GPIO34_CHANNEL

ADC1_CHANNEL_6_GPIO_NUM

ADC1_GPIO35_CHANNEL

ADC1_CHANNEL_7_GPIO_NUM

ADC2_GPIO4_CHANNEL

ADC2_CHANNEL_0_GPIO_NUM

ADC2_GPIO0_CHANNEL

ADC2_CHANNEL_1_GPIO_NUM

ADC2_GPIO2_CHANNEL

ADC2_CHANNEL_2_GPIO_NUM

ADC2_GPIO15_CHANNEL

ADC2_CHANNEL_3_GPIO_NUM

ADC2_GPIO13_CHANNEL

ADC2_CHANNEL_4_GPIO_NUM

ADC2_GPIO12_CHANNEL

ADC2_CHANNEL_5_GPIO_NUM

ADC2_GPIO14_CHANNEL

ADC2_CHANNEL_6_GPIO_NUM

ADC2_GPIO27_CHANNEL

ADC2_CHANNEL_7_GPIO_NUM

ADC2_GPIO25_CHANNEL

ADC2_CHANNEL_8_GPIO_NUM

ADC2_GPIO26_CHANNEL

ADC2_CHANNEL_9_GPIO_NUM

2.3.2 Controller Area Network (CAN)

Overview

The ESP32' s peripherals contains a CAN Controller that supports Standard Frame Format (11-bit ID) and Extended Frame Format (29-bit ID) of the CAN2.0B specification.

警告: The ESP32 CAN controller is not compatible with CAN FD frames and will interpret such frames as errors.

This programming guide is split into the following sections:

1. *Basic CAN Concepts*
2. *Signals Lines and Transceiver*
3. *Configuration*
4. *Driver Operation*
5. *Examples*

Basic CAN Concepts

注解: The following section only covers the basic aspects of CAN. For full details, see the CAN2.0B specification

---

The CAN protocol is a multi-master, multi-cast communication protocol with error detection/signalling and inbuilt message prioritization. The CAN protocol is commonly used as a communication bus in automotive applications.

Multi-master: Any node in a CAN bus is allowed initiate the transfer of data.

Multi-cast: When a node transmits a message, all nodes are able to receive the message (broadcast). However some nodes can selective choose which messages to accept via the use of acceptance filtering (multi-cast).

Error Detection and Signalling: Every CAN node will constantly monitor the CAN bus. When any node detects an error, it will signal the error by transmitting an error frame. Other nodes will receive the error frame and transmit their own error frames in response. This will result in an error detection being propagated to all nodes on the bus.

Message Priorities: If two nodes attempt to transmit simultaneously, the node transmitting the message with the lower ID will win arbitration. All other nodes will become receivers ensuring there is at most one transmitter at any time.

CAN Message Frames

The CAN2.0B specification contains two frame formats known as **Extended Frame** and **Standard Frame** which contain 29-bit IDs and 11-bit IDs respectively. A CAN message consists of the following components

- 29-bit or 11-bit ID
- Data Length Code (DLC) between 0 to 8
- Up to 8 bytes of data (should match DLC)

Error States and Counters

The CAN2.0B specification implements fault confinement by requiring every CAN node to maintain two internal error counters known as the **Transmit Error Counter (TEC)** and the **Receive Error Counter (REC)**. The two error counters are used to determine a CAN node' s **error state**, and the counters are incremented and decremented following a set of rules (see CAN2.0B specification). These error states are known as **Error Active**, **Error Passive**, and **Bus-Off**.

Error Active: A CAN node is Error Active when **both TEC and REC are less than 128** and indicates a CAN node is operating normally. Error Active nodes are allowed to participate in CAN bus activities, and will actively signal any error conditions it detects by transmitting an **Active Error Flag** over the CAN bus.

Error Passive: A CAN node is Error Passive when **either the TEC or REC becomes greater than or equal to 128**. Error Passive nodes are still able to take part in CAN bus activities, but will instead transmit a **Passive Error Flag** upon detection of an error.

Bus-Off: A CAN node becomes Bus-Off when the **TEC becomes greater than or equal to 256**. A Bus-Off node is unable take part in CAN bus activity and will remain so until it undergoes bus recovery.

Signals Lines and Transceiver

The CAN controller does not contain a internal transceiver and therefore **requires an external transceiver** to operate. The type of external transceiver will depend on the application' s physical layer specification (e.g. using SN65HVD23X transceivers for ISO 11898-2 compatibility).

The CAN controller' s interface consists of 4 signal lines known as **TX, RX, BUS-OFF, and CLKOUT**. These four signal lines can be routed through the GPIO Matrix to GPIOs.

TX and RX: The TX and RX signal lines are required to interface with an external CAN transceiver. Both signal lines represent/interpret a dominant bit as a low logic level (0V), and a recessive bit as a high logic level (3.3V).

BUS-OFF: The BUS-OFF signal line is **optional** and is set to a low logic level (0V) whenever the CAN controller reaches a bus-off state. The BUS-OFF signal line is set to a high logic level (3.3V) otherwise.

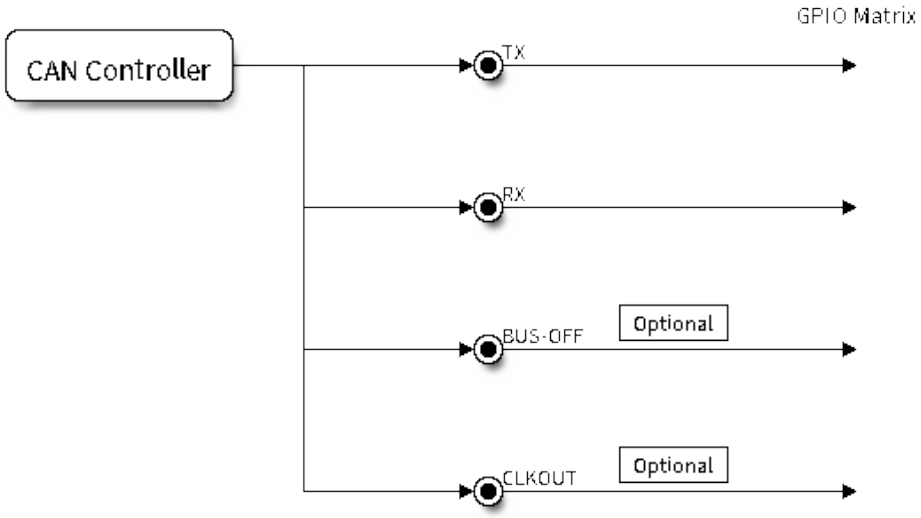


图 7: Signal lines of the CAN controller

CLKOUT: The CLKOUT signal line is **optional** and outputs a prescaled version of the CAN controller's source clock (APB Clock).

注解: An external transceiver **must internally tie the TX input and the RX output** such that a change in logic level to the TX signal line can be observed on the RX line. Failing to do so will cause the CAN controller to interpret differences in logic levels between the two signal lines as a lost in arbitration or a bit error.

Configuration

Operating Modes

The CAN driver supports the following modes of operations:

Normal Mode: The normal operating mode allows the CAN controller to take part in bus activities such as transmitting and receiving messages/error frames. Acknowledgement from another CAN node is required when transmitting message frames.

No Ack Mode: The No Acknowledgement mode is similar to normal mode, however acknowledgements are not required when transmitting message frames. This mode is useful when self testing the CAN controller.

Listen Only Mode: This mode will prevent the CAN controller from taking part in bus activities. Therefore transmissions of messages/acknowledgement/error frames will be disabled. However the the CAN controller

will still be able to receive messages (without acknowledging). This mode is suited for applications such as CAN bus monitoring.

Alerts

The CAN driver contains an alert feature which is used to notify the application level of certain CAN driver events. Alerts are selectively enabled when the CAN driver is installed, but can be reconfigured during runtime by calling *can_reconfigure_alerts()*. The application can then wait for any enabled alerts to occur by calling *can_read_alerts()*. The CAN driver supports the following alerts:

Alert	Description
CAN_ALERT_TX_IDLE	No more messages queued for transmission
CAN_ALERT_TX_SUCCESS	The previous transmission was successful
CAN_ALERT_BELOW_ERR_WARN	Both error counters have dropped below error warning limit
CAN_ALERT_ERR_ACTIVE	CAN controller has become error active
CAN_ALERT_RECOVERY_IN_PROGRESS	CAN controller is undergoing bus recovery
CAN_ALERT_BUS_RECOVERED	CAN controller has successfully completed bus recovery
CAN_ALERT_ARB_LOST	The previous transmission lost arbitration
CAN_ALERT_ABOVE_ERR_WARN	One of the error counters have exceeded the error warning limit
CAN_ALERT_BUS_ERROR	A (Bit, Stuff, CRC, Form, ACK) error has occurred on the bus
CAN_ALERT_TX_FAILED	The previous transmission has failed
CAN_ALERT_RX_QUEUE_FULL	The RX queue is full causing a received frame to be lost
CAN_ALERT_ERR_PASS	CAN controller has become error passive
CAN_ALERT_BUS_OFF	Bus-off condition occurred. CAN controller can no longer influence bus

注解: The **error warning limit** can be used to preemptively warn the application of bus errors before the error passive state is reached. By default the CAN driver sets the **error warning limit** to **96**. The CAN_ALERT_ABOVE_ERR_WARN is raised when the TEC or REC becomes larger then or equal to the error warning limit. The CAN_ALERT_BELOW_ERR_WARN is raised when both TEC and REC return back to values below **96**.

注解: When enabling alerts, the CAN_ALERT_AND_LOG flag can be used to cause the CAN driver to log any raised alerts to UART. The CAN_ALERT_ALL and CAN_ALERT_NONE macros can also be used to enable/disable all alerts during configuration/reconfiguration.

Bit Timing

The operating bit rate of the CAN controller is configured using the *can_timing_config_t* structure. The period of each bit is made up of multiple **time quanta**, and the period of a **time quanta** is determined by a prescaled version of the CAN controller' s source clock. A single bit contains the following segments in the following order:

1. The **Synchronization Segment** consists of a single time quanta
2. **Timing Segment 1** consists of 1 to 16 time quanta before sample point
3. **Timing Segment 2** consists of 1 to 8 time quanta after sample point

The **Baudrate Prescaler** is used to determine the period of each time quanta by dividing the CAN controller' s source clock (80 MHz APB clock). The `brp` can be **any even number from 2 to 128**. If the ESP32 is a revision 2 or later chip, the `brp` will also support **any multiple of 4 from 132 to 256**, and can be enabled by setting the *CONFIG_ESP32_REV_MIN* to revision 2 or higher.

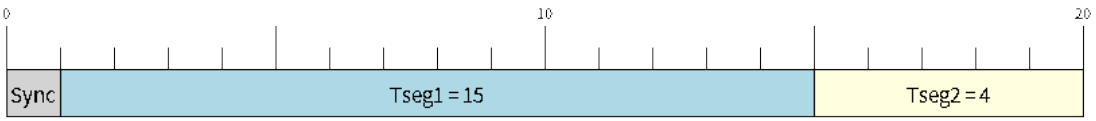


图 8: Bit timing configuration for 500kbit/s given BRP = 8

The sample point of a bit is located on the intersection of Timing Segment 1 and 2. Enabling **Triple Sampling** will cause 3 time quanta to be sampled per bit instead of 1 (extra samples are located at the tail end of Timing Segment 1).

The **Synchronization Jump Width** is used to determined the maximum number of time quanta a single bit time can be lengthened/shortened for synchronization purposes. `sjw` can **range from 1 to 4**.

注解: Multiple combinations of `brp`, `tseg_1`, `tseg_2`, and `sjw` can achieve the same bit rate. Users should tune these values to the physical characteristics of their CAN bus by taking into account factors such as **propagation delay, node information processing time, and phase errors**.

Bit timing **macro initializers** are also available for commonly used CAN bus bit rates. The following macro initializers are provided by the CAN driver.

- `CAN_TIMING_CONFIG_12_5KBITS()`
- `CAN_TIMING_CONFIG_16KBITS()`
- `CAN_TIMING_CONFIG_20KBITS()`
- `CAN_TIMING_CONFIG_25KBITS()`

- CAN_TIMING_CONFIG_50KBITS()
- CAN_TIMING_CONFIG_100KBITS()
- CAN_TIMING_CONFIG_125KBITS()
- CAN_TIMING_CONFIG_250KBITS()
- CAN_TIMING_CONFIG_500KBITS()
- CAN_TIMING_CONFIG_800KBITS()
- CAN_TIMING_CONFIG_1MBITS()

注解: The macro initializers for 12.5K, 16K, and 20K bit rates are only available for ESP32 revision 2 or later.

Acceptance Filter

The CAN controller contains a hardware acceptance filter which can be used to filter CAN messages of a particular ID. A node that filters out a message **will not receive the message, but will still acknowledge it**. Acceptances filters can make a node more efficient by filtering out messages sent over the CAN bus that are irrelevant to the CAN node in question. The CAN controller' s acceptance filter is configured using two 32-bit values within *can_filter_config_t* known as the **acceptance code** and the **acceptance mask**.

The **acceptance code** specifies the bit sequence which a message' s ID, RTR, and data bytes must match in order for the message to be received by the CAN controller. The **acceptance mask** is a bit sequence specifying which bits of the acceptance code can be ignored. This allows for a messages of different IDs to be accepted by a single acceptance code.

The acceptance filter can be used under **Single or Dual Filter Mode**. Single Filter Mode will use the acceptance code and mask to define a single filter. This allows for the first two data bytes of a standard frame to be filtered, or the entirety of an extended frame' s 29-bit ID. The following diagram illustrates how the 32-bit acceptance code and mask will be interpreted under Single Filter Mode (Note: The yellow and blue fields represent standard and extended CAN frames respectively).

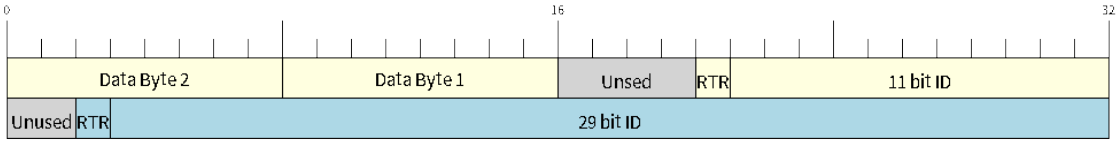


图 9: Bit layout of single filter mode (Right side MSBit)

Dual Filter Mode will use the acceptance code and mask to define two separate filters allowing for increased

flexibility of ID' s to accept, but does not allow for all 29-bits of an extended ID to be filtered. The following diagram illustrates how the 32-bit acceptance code and mask will be interpreted under **Dual Filter Mode** (Note: The yellow and blue fields represent standard and extended CAN frames respectively).

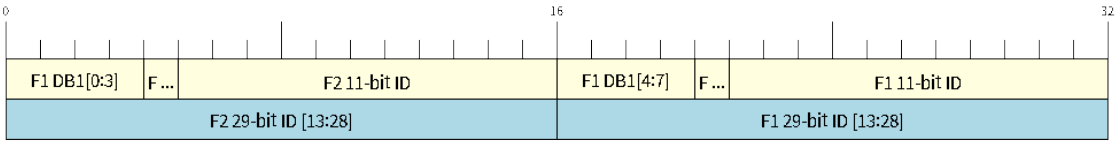


图 10: Bit layout of dual filter mode (Right side MSBit)

Disabling TX Queue

The TX queue can be disabled during configuration by setting the `tx_queue_len` member of *`can_general_config_t`* to `0`. This will allow applications that do not require message transmission to save a small amount of memory when using the CAN driver.

Driver Operation

The CAN driver is designed with distinct states and strict rules regarding the functions or conditions that trigger a state transition. The following diagram illustrates the various states and their transitions.

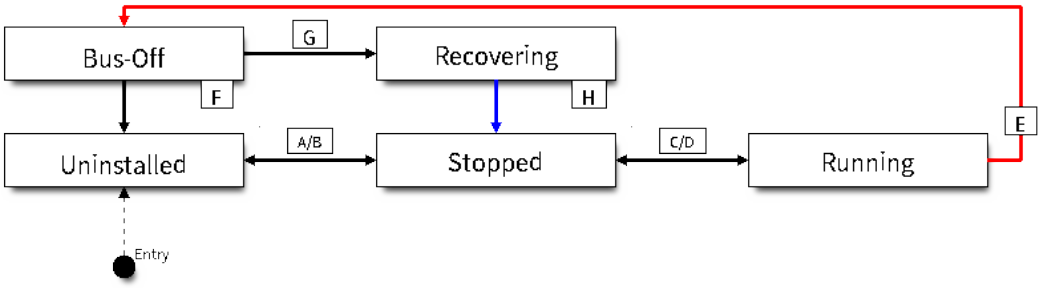


图 11: State transition diagram of the CAN driver (see table below)

Label	Transition	Action/Condition
A	Uninstalled -> Stopped	*can_driver_install()*
B	Stopped -> Uninstalled	*can_driver_uninstall()*
C	Stopped -> Running	*can_start()*
D	Running -> Stopped	*can_stop()*
E	Running -> Bus-Off	Transmit Error Counter >= 256
F	Bus-Off -> Uninstalled	*can_driver_uninstall()*
G	Bus-Off -> Recovering	*can_initiate_recovery()*
H	Recovering -> Stopped	128 occurrences of bus-free signal

Driver States

Uninstalled: In the uninstalled state, no memory is allocated for the driver and the CAN controller is powered OFF.

Stopped: In this state, the CAN controller is powered ON and the CAN driver has been installed. However the CAN controller will be unable to take part in any CAN bus activities such as transmitting, receiving, or acknowledging messages.

Running: In the running state, the CAN controller is able to take part in bus activities. Therefore messages can be transmitted/received/acknowledged. Furthermore the CAN controller will be able to transmit error frames upon detection of errors on the CAN bus.

Bus-Off: The bus-off state is automatically entered when the CAN controller' s Transmit Error Counter becomes greater than or equal to 256 (see CAN2.0B specification regarding error counter rules). The bus-off state indicates the occurrence of severe errors on the CAN bus or in the CAN controller. Whilst in the bus-off state, the CAN controller will be unable to take part in any CAN bus activities. To exit the bus-off state, the CAN controller must undergo the bus recovery process.

Recovering: The recovering state is entered when the CAN driver undergoes bus recovery. The CAN driver/controller will remain in the recovering state until the 128 occurrences of the bus-free signal (see CAN2.0B specification) is observed on the CAN bus.

Message Flags

The CAN driver distinguishes different types of CAN messages by using the various bit field members of the *can_message_t* structure. These bit field members help distinguish whether a message is in standard or extended format, an RTR, and the type of transmission to use when transmitting such a message. These bit field members can also be toggled using the the *flags* member of *can_message_t* and the following message flags:

Flag	Description
CAN_MSG_FLAG_EXTD	Message is in Extended Frame Format (29bit ID)
CAN_MSG_FLAG_RTR	Message is a Remote Transmit Request
CAN_MSG_FLAG_SS	Transmit message using Single Shot Transmission (Message will not be retransmitted upon error or loss of arbitration). Unused for received message.
CAN_MSG_FLAG_SELF	Transmit message using Self Reception Request (Transmitted message will also received by the same node). Unused for received message.
CAN_MSG_FLAG_DLC_NON_COMP	Message's Data length code is larger than 8. This will break compliance with CAN2.0B
CAN_MSG_FLAG_NONE	Clears all bit fields. Equivalent to a Standard Frame Format (11bit ID) Data Frame.

Examples

Configuration & Installation

The following code snippet demonstrates how to configure, install, and start the CAN driver via the use of the various configuration structures, macro initializers, the *can_driver_install()* function, and the *can_start()* function.

```
#include "driver/gpio.h"
#include "driver/can.h"

void app_main()
{
    //Initialize configuration structures using macro initializers
    can_general_config_t g_config = CAN_GENERAL_CONFIG_DEFAULT(GPIO_NUM_21, GPIO_NUM_22,␣
↪CAN_MODE_NORMAL);
    can_timing_config_t t_config = CAN_TIMING_CONFIG_500KBITS();
    can_filter_config_t f_config = CAN_FILTER_CONFIG_ACCEPT_ALL();

    //Install CAN driver
    if (can_driver_install(&g_config, &t_config, &f_config) == ESP_OK) {
        printf("Driver installed\n");
    } else {
        printf("Failed to install driver\n");
        return;
    }

    //Start CAN driver
    if (can_start() == ESP_OK) {
```

(下页继续)

(续上页)

```
        printf("Driver started\n");
    } else {
        printf("Failed to start driver\n");
        return;
    }

    ...

}
```

The usage of macro initializers are not mandatory and each of the configuration structures can be manually.

Message Transmission

The following code snippet demonstrates how to transmit a message via the usage of the *can_message_t* type and *can_transmit()* function.

```
#include "driver/can.h"

...

//Configure message to transmit
can_message_t message;
message.identifier = 0xAAAA;
message.extd = 1;
message.data_length_code = 4;
for (int i = 0; i < 4; i++) {
    message.data[i] = 0;
}

//Queue message for transmission
if (can_transmit(&message, pdMS_TO_TICKS(1000)) == ESP_OK) {
    printf("Message queued for transmission\n");
} else {
    printf("Failed to queue message for transmission\n");
}
```

Message Reception

The following code snippet demonstrates how to receive a message via the usage of the *can_message_t* type and *can_receive()* function.

```
#include "driver/can.h"

...

//Wait for message to be received
can_message_t message;
if (can_receive(&message, pdMS_TO_TICKS(10000)) == ESP_OK) {
    printf("Message received\n");
} else {
    printf("Failed to receive message\n");
    return;
}

//Process received message
if (message.extd) {
    printf("Message is in Extended Format\n");
} else {
    printf("Message is in Standard Format\n");
}
printf("ID is %d\n", message.identifier);
if (!(message.rtr)) {
    for (int i = 0; i < message.data_length_code; i++) {
        printf("Data byte %d = %d\n", i, message.data[i]);
    }
}
```

Reconfiguring and Reading Alerts

The following code snippet demonstrates how to reconfigure and read CAN driver alerts via the use of the *can_reconfigure_alerts()* and *can_read_alerts()* functions.

```
#include "driver/can.h"

...

//Reconfigure alerts to detect Error Passive and Bus-Off error states
```

(下页继续)

(续上页)

```
uint32_t alerts_to_enable = CAN_ALERT_ERR_PASS | CAN_ALERT_BUS_OFF;
if (can_reconfigure_alerts(alerts_to_enable, NULL) == ESP_OK) {
    printf("Alerts reconfigured\n");
} else {
    printf("Failed to reconfigure alerts");
}

//Block indefinitely until an alert occurs
uint32_t alerts_triggered;
can_read_alerts(&alerts_triggered, portMAX_DELAY);
```

Stop and Uninstall

The following code demonstrates how to stop and uninstall the CAN driver via the use of the *can_stop()* and *can_driver_uninstall()* functions.

```
#include "driver/can.h"

...

//Stop the CAN driver
if (can_stop() == ESP_OK) {
    printf("Driver stopped\n");
} else {
    printf("Failed to stop driver\n");
    return;
}

//Uninstall the CAN driver
if (can_driver_uninstall() == ESP_OK) {
    printf("Driver uninstalled\n");
} else {
    printf("Failed to uninstall driver\n");
    return;
}
```

Multiple ID Filter Configuration

The acceptance mask in *can_filter_config_t* can be configured such that two or more IDs will be accepted for a single filter. For a particular filter to accept multiple IDs, the conflicting bit positions amongst the IDs must be set in the acceptance mask. The acceptance code can be set to any one of the IDs.

The following example shows how the calculate the acceptance mask given multiple IDs:

```
ID1 =  11'b101 1010 0000
ID2 =  11'b101 1010 0001
ID3 =  11'b101 1010 0100
ID4 =  11'b101 1010 1000
//Acceptance Mask
MASK = 11'b000 0000 1101
```

Application Examples

Network Example: The CAN Network example demonstrates communication between two ESP32s using the CAN driver API. One CAN node acts as a network master initiate and ceasing the transfer of a data from another CAN node acting as a network slave. The example can be found via peripherals/can/can_network.

Alert and Recovery Example: This example demonstrates how to use the CAN driver' s alert and bus recovery API. The example purposely introduces errors on the CAN bus to put the CAN controller into the Bus-Off state. An alert is used to detect the Bus-Off state and trigger the bus recovery process. The example can be found via peripherals/can/can_alert_and_recovery.

Self Test Example: This example uses the No Acknowledge Mode and Self Reception Request to cause the CAN controller to send and simultaneously receive a series of messages. This example can be used to verify if the connections between the CAN controller and the external transceiver are working correctly. The example can be found via peripherals/can/can_self_test.

API Reference

Header File

- soc/include/hal/can_types.h

Structures

`struct can_message_t`

Structure to store a CAN message.

Note

Note The flags member is deprecated

Public Members

uint32_t **extd**

Extended Frame Format (29bit ID)

uint32_t **rtr**

Message is a Remote Transmit Request

uint32_t **ss**

Transmit as a Single Shot Transmission. Unused for received.

uint32_t **self**

Transmit as a Self Reception Request. Unused for received.

uint32_t **dlc_non_comp**

Message' s Data length code is larger than 8. This will break compliance with CAN2.0B.

uint32_t **reserved**

Reserved bits

uint32_t **flags**

Alternate way to set message flags using message flag macros (see documentation)

uint32_t **identifier**

11 or 29 bit identifier

uint8_t **data_length_code**

Data length code

uint8_t **data**[**CAN_FRAME_MAX_DLC**]

Data bytes (not relevant in RTR frame)

struct can_timing_config_t

Structure for bit timing configuration of the CAN driver.

Note Macro initializers are available for this structure

Public Members

uint32_t **brp**

Baudrate prescaler (i.e., APB clock divider) can be any even number from 2 to 128. For ESP32 Rev 2 or later, multiples of 4 from 132 to 256 are also supported

uint8_t **tseg_1**

Timing segment 1 (Number of time quanta, between 1 to 16)

uint8_t **tseg_2**

Timing segment 2 (Number of time quanta, 1 to 8)

uint8_t **sjw**

Synchronization Jump Width (Max time quanta jump for synchronize from 1 to 4)

bool **triple_sampling**

Enables triple sampling when the CAN controller samples a bit

struct can_filter_config_t

Structure for acceptance filter configuration of the CAN driver (see documentation)

Note Macro initializers are available for this structure

Public Members

uint32_t **acceptance_code**

32-bit acceptance code

uint32_t **acceptance_mask**

32-bit acceptance mask

bool **single_filter**

Use Single Filter Mode (see documentation)

Macros

CAN_EXTD_ID_MASK

CAN2.0B Constants.

Bit mask for 29 bit Extended Frame Format ID

CAN_STD_ID_MASK

Bit mask for 11 bit Standard Frame Format ID

CAN_FRAME_MAX_DLC

Max data bytes allowed in CAN2.0

CAN_FRAME_EXTD_ID_LEN_BYTES

EFF ID requires 4 bytes (29bit)

CAN_FRAME_STD_ID_LEN_BYTES

SFF ID requires 2 bytes (11bit)

CAN_ERR_PASS_THRESH

Error counter threshold for error passive

Enumerations

enum can_mode_t

CAN Controller operating modes.

Values:

CAN_MODE_NORMAL

Normal operating mode where CAN controller can send/receive/acknowledge messages

CAN_MODE_NO_ACK

Transmission does not require acknowledgment. Use this mode for self testing

CAN_MODE_LISTEN_ONLY

The CAN controller will not influence the bus (No transmissions or acknowledgments) but can receive messages

Header File

- driver/include/driver/can.h

Functions

esp_err_t **can_driver_install**(**const** *can_general_config_t* **g_config*, **const** *can_timing_config_t* **t_config*, **const** *can_filter_config_t* **f_config*)

Install CAN driver.

This function installs the CAN driver using three configuration structures. The required memory is allocated and the CAN driver is placed in the stopped state after running this function.

Note Macro initializers are available for the configuration structures (see documentation)

Note To reinstall the CAN driver, call can_driver_uninstall() first

Return

- ESP_OK: Successfully installed CAN driver
- ESP_ERR_INVALID_ARG: Arguments are invalid
- ESP_ERR_NO_MEM: Insufficient memory
- ESP_ERR_INVALID_STATE: Driver is already installed

Parameters

- `g_config`: General configuration structure
- `t_config`: Timing configuration structure

- `f_config`: Filter configuration structure

esp_err_t **can_driver_uninstall**(void)

Uninstall the CAN driver.

This function uninstalls the CAN driver, freeing the memory utilized by the driver. This function can only be called when the driver is in the stopped state or the bus-off state.

Warning The application must ensure that no tasks are blocked on TX/RX queues or alerts when this function is called.

Return

- ESP_OK: Successfully uninstalled CAN driver
- ESP_ERR_INVALID_STATE: Driver is not in stopped/bus-off state, or is not installed

esp_err_t **can_start**(void)

Start the CAN driver.

This function starts the CAN driver, putting the CAN driver into the running state. This allows the CAN driver to participate in CAN bus activities such as transmitting/receiving messages. The RX queue is reset in this function, clearing any unread messages. This function can only be called when the CAN driver is in the stopped state.

Return

- ESP_OK: CAN driver is now running
- ESP_ERR_INVALID_STATE: Driver is not in stopped state, or is not installed

esp_err_t **can_stop**(void)

Stop the CAN driver.

This function stops the CAN driver, preventing any further message from being transmitted or received until can_start() is called. Any messages in the TX queue are cleared. Any messages in the RX queue should be read by the application after this function is called. This function can only be called when the CAN driver is in the running state.

Warning A message currently being transmitted/received on the CAN bus will be ceased immediately. This may lead to other CAN nodes interpreting the unfinished message as an error.

Return

- ESP_OK: CAN driver is now Stopped
- ESP_ERR_INVALID_STATE: Driver is not in running state, or is not installed

esp_err_t **can_transmit**(**const** *can_message_t* **message*, TickType_t *ticks_to_wait*)

Transmit a CAN message.

This function queues a CAN message for transmission. Transmission will start immediately if no other messages are queued for transmission. If the TX queue is full, this function will block until more space becomes available or until it timesout. If the TX queue is disabled (TX queue length = 0 in configuration), this function will return immediately if another message is undergoing transmission. This function can only be called when the CAN driver is in the running state and cannot be called under Listen Only Mode.

Note This function does not guarantee that the transmission is successful. The TX_SUCCESS/TX_FAILED alert can be enabled to alert the application upon the success/failure of a transmission.

Note The TX_IDLE alert can be used to alert the application when no other messages are awaiting transmission.

Return

- ESP_OK: Transmission successfully queued/initiated
- ESP_ERR_INVALID_ARG: Arguments are invalid
- ESP_ERR_TIMEOUT: Timed out waiting for space on TX queue
- ESP_FAIL: TX queue is disabled and another message is currently transmitting
- ESP_ERR_INVALID_STATE: CAN driver is not in running state, or is not installed
- ESP_ERR_NOT_SUPPORTED: Listen Only Mode does not support transmissions

Parameters

- `message`: Message to transmit
- `ticks_to_wait`: Number of FreeRTOS ticks to block on the TX queue

esp_err_t **can_receive**(*can_message_t* **message*, TickType_t *ticks_to_wait*)

Receive a CAN message.

This function receives a message from the RX queue. The flags field of the message structure will indicate the type of message received. This function will block if there are no messages in the RX queue

Warning The flags field of the received message should be checked to determine if the received message contains any data bytes.

Return

- ESP_OK: Message successfully received from RX queue
- ESP_ERR_TIMEOUT: Timed out waiting for message
- ESP_ERR_INVALID_ARG: Arguments are invalid
- ESP_ERR_INVALID_STATE: CAN driver is not installed

Parameters

- `message`: Received message
- `ticks_to_wait`: Number of FreeRTOS ticks to block on RX queue

esp_err_t **can_read_alerts**(uint32_t **alerts*, TickType_t *ticks_to_wait*)

Read CAN driver alerts.

This function will read the alerts raised by the CAN driver. If no alert has been when this function is called, this function will block until an alert occurs or until it timeouts.

Note Multiple alerts can be raised simultaneously. The application should check for all alerts that have been enabled.

Return

- ESP_OK: Alerts read
- ESP_ERR_TIMEOUT: Timed out waiting for alerts
- ESP_ERR_INVALID_ARG: Arguments are invalid
- ESP_ERR_INVALID_STATE: CAN driver is not installed

Parameters

- `alerts`: Bit field of raised alerts (see documentation for alert flags)
- `ticks_to_wait`: Number of FreeRTOS ticks to block for alert

esp_err_t **can_reconfigure_alerts**(uint32_t *alerts_enabled*, uint32_t **current_alerts*)

Reconfigure which alerts are enabled.

This function reconfigures which alerts are enabled. If there are alerts which have not been read whilst reconfiguring, this function can read those alerts.

Return

- ESP_OK: Alerts reconfigured
- ESP_ERR_INVALID_STATE: CAN driver is not installed

Parameters

- `alerts_enabled`: Bit field of alerts to enable (see documentation for alert flags)
- `current_alerts`: Bit field of currently raised alerts. Set to NULL if unused

esp_err_t **can_initiate_recovery**(void)

Start the bus recovery process.

This function initiates the bus recovery process when the CAN driver is in the bus-off state. Once initiated, the CAN driver will enter the recovering state and wait for 128 occurrences of the bus-free

signal on the CAN bus before returning to the stopped state. This function will reset the TX queue, clearing any messages pending transmission.

Note The BUS_RECOVERED alert can be enabled to alert the application when the bus recovery process completes.

Return

- ESP_OK: Bus recovery started
- ESP_ERR_INVALID_STATE: CAN driver is not in the bus-off state, or is not installed

esp_err_t **can_get_status_info**(*can_status_info_t* **status_info*)

Get current status information of the CAN driver.

Return

- ESP_OK: Status information retrieved
- ESP_ERR_INVALID_ARG: Arguments are invalid
- ESP_ERR_INVALID_STATE: CAN driver is not installed

Parameters

- `status_info`: Status information

esp_err_t **can_clear_transmit_queue**(void)

Clear the transmit queue.

This function will clear the transmit queue of all messages.

Note The transmit queue is automatically cleared when can_stop() or can_initiate_recovery() is called.

Return

- ESP_OK: Transmit queue cleared
- ESP_ERR_INVALID_STATE: CAN driver is not installed or TX queue is disabled

esp_err_t **can_clear_receive_queue**(void)

Clear the receive queue.

This function will clear the receive queue of all messages.

Note The receive queue is automatically cleared when can_start() is called.

Return

- ESP_OK: Transmit queue cleared
- ESP_ERR_INVALID_STATE: CAN driver is not installed

Structures

struct can_general_config_t

Structure for general configuration of the CAN driver.

Note Macro initializers are available for this structure

Public Members

can_mode_t **mode**

Mode of CAN controller

gpio_num_t **tx_io**

Transmit GPIO number

gpio_num_t **rx_io**

Receive GPIO number

gpio_num_t **clkout_io**

CLKOUT GPIO number (optional, set to -1 if unused)

gpio_num_t **bus_off_io**

Bus off indicator GPIO number (optional, set to -1 if unused)

uint32_t **tx_queue_len**

Number of messages TX queue can hold (set to 0 to disable TX Queue)

uint32_t **rx_queue_len**

Number of messages RX queue can hold

uint32_t **alerts_enabled**

Bit field of alerts to enable (see documentation)

uint32_t **clkout_divider**

CLKOUT divider. Can be 1 or any even number from 2 to 14 (optional, set to 0 if unused)

struct can_status_info_t

Structure to store status information of CAN driver.

Public Members

can_state_t **state**

Current state of CAN controller (Stopped/Running/Bus-Off/Recovery)

uint32_t **msgs_to_tx**

Number of messages queued for transmission or awaiting transmission completion

uint32_t **msgs_to_rx**

Number of messages in RX queue waiting to be read

uint32_t **tx_error_counter**

Current value of Transmit Error Counter

uint32_t **rx_error_counter**

Current value of Receive Error Counter

uint32_t **tx_failed_count**

Number of messages that failed transmissions

uint32_t **rx_missed_count**

Number of messages that were lost due to a full RX queue

uint32_t **arb_lost_count**

Number of instances arbitration was lost

uint32_t **bus_error_count**

Number of instances a bus error has occurred

Macros

CAN_IO_UNUSED

Marks GPIO as unused in CAN configuration

Enumerations

enum can_state_t

CAN driver states.

Values:

CAN_STATE_STOPPED

Stopped state. The CAN controller will not participate in any CAN bus activities

CAN_STATE_RUNNING

Running state. The CAN controller can transmit and receive messages

CAN_STATE_BUS_OFF

Bus-off state. The CAN controller cannot participate in bus activities until it has recovered

CAN_STATE_RECOVERING

Recovering state. The CAN controller is undergoing bus recovery

2.3.3 Digital To Analog Converter

Overview

ESP32 has two 8-bit DAC (digital to analog converter) channels, connected to GPIO25 (Channel 1) and GPIO26 (Channel 2).

The DAC driver allows these channels to be set to arbitrary voltages.

The DAC channels can also be driven with DMA-style written sample data, via the *I2S driver* when using the "built-in DAC mode".

For other analog output options, see the *Sigma-delta Modulation module* and the *LED Control module*. Both these modules produce high frequency PWM output, which can be hardware low-pass filtered in order to generate a lower frequency analog output.

Application Example

Setting DAC channel 1 (GPIO 25) voltage to approx 0.78 of VDD_A voltage (VDD * 200 / 255). For VDD_A 3.3V, this is 2.59V:

```
#include <driver/dac.h>

...

    dac_output_enable(DAC_CHANNEL_1);
    dac_output_voltage(DAC_CHANNEL_1, 200);
```

API Reference

Header File

- driver/include/driver/dac.h

Functions

esp_err_t **dac_pad_get_io_num**(*dac_channel_t channel*, *gpio_num_t *gpio_num*)

Get the gpio number of a specific DAC channel.

Return

- ESP_OK if success

Parameters

- `channel`: Channel to get the gpio number
- `gpio_num`: output buffer to hold the gpio number

esp_err_t **dac_output_voltage**(*dac_channel_t channel*, uint8_t *dac_value*)

Set DAC output voltage. DAC output is 8-bit. Maximum (255) corresponds to VDD3P3_RTC.

Note Need to configure DAC pad before calling this function. DAC channel 1 is attached to GPIO25, DAC channel 2 is attached to GPIO26

Return

- ESP_OK success

Parameters

- `channel`: DAC channel
- `dac_value`: DAC output value

esp_err_t **dac_output_enable**(*dac_channel_t channel*)

DAC pad output enable.

Note DAC channel 1 is attached to GPIO25, DAC channel 2 is attached to GPIO26 I2S left channel will be mapped to DAC channel 2 I2S right channel will be mapped to DAC channel 1

Parameters

- `channel`: DAC channel

esp_err_t **dac_output_disable**(*dac_channel_t channel*)

DAC pad output disable.

Note DAC channel 1 is attached to GPIO25, DAC channel 2 is attached to GPIO26

Return

- ESP_OK success

Parameters

- `channel`: DAC channel

esp_err_t **dac_i2s_enable**(void)

Enable DAC output data from I2S.

Return

- ESP_OK success

esp_err_t **dac_i2s_disable**(void)

Disable DAC output data from I2S.

Return

- ESP_OK success

esp_err_t **dac_cw_generator_enable**(void)

Enable cosine wave generator output.

Return

- ESP_OK success

esp_err_t **dac_cw_generator_disable**(void)

Disable cosine wave generator output.

Return

- ESP_OK success

esp_err_t **dac_cw_generator_config**(*dac_cw_config_t* *cw)

Config the cosine wave generator function in DAC module.

Return

- ESP_OK success

Parameters

- `cw`: Configuration.

GPIO Lookup Macros

Some useful macros can be used to specified the GPIO number of a DAC channel, or vice versa. e.g.

1. `DAC_CHANNEL_1_GPIO_NUM` is the GPIO number of channel 1 (25);
2. `DAC_GPIO26_CHANNEL` is the channel number of GPIO 26 (channel 2).

Header File

- soc/esp32/include/soc/dac_channel.h

Macros

`DAC_GPIO25_CHANNEL`

`DAC_CHANNEL_1_GPIO_NUM`

`DAC_GPIO26_CHANNEL`

`DAC_CHANNEL_2_GPIO_NUM`

Header File

- soc/include/hal/dac_types.h

Structures

struct dac_cw_config_t

Config the cosine wave generator function in DAC module.

Public Members

dac_channel_t **en_ch**

Enable the cosine wave generator of DAC channel.

dac_cw_scale_t **scale**

Set the amplitude of the cosine wave generator output.

dac_cw_phase_t **phase**

Set the phase of the cosine wave generator output.

uint32_t **freq**

Set frequency of cosine wave generator output. Range: 130(130Hz) ~ 55000(100KHz).

int8_t **offset**

Set the voltage value of the DC component of the cosine wave generator output. Note: Unreasonable settings can cause waveform to be oversaturated. Range: -128 ~ 127.

Enumerations

enum dac_channel_t

Values:

DAC_CHANNEL_1 = 0

DAC channel 1 is GPIO25(ESP32) / GPIO17(ESP32S2BETA)

DAC_CHANNEL_2 = 1

DAC channel 2 is GPIO26(ESP32) / GPIO18(ESP32S2BETA)

DAC_CHANNEL_MAX

enum dac_cw_scale_t

The multiple of the amplitude of the cosine wave generator. The max amplitude is VDD3P3_RTC.

Values:

DAC_CW_SCALE_1 = 0x0

1/1. Default.

`DAC_CW_SCALE_2` = 0x1
1/2.

`DAC_CW_SCALE_4` = 0x2
1/4.

`DAC_CW_SCALE_8` = 0x3
1/8.

`enum dac_cw_phase_t`
Set the phase of the cosine wave generator output.

Values:

`DAC_CW_PHASE_0` = 0x2
Phase shift +0°

`DAC_CW_PHASE_180` = 0x3
Phase shift +180°

2.3.4 GPIO & RTC GPIO

Overview

The ESP32 chip features 40 physical GPIO pads. Some GPIO pads cannot be used or do not have the corresponding pin on the chip package(refer to technical reference manual). Each pad can be used as a general purpose I/O or can be connected to an internal peripheral signal.

- Note that GPIO6-11 are usually used for SPI flash.
- GPIO34-39 can only be set as input mode and do not have software pullup or pulldown functions.

There is also separate "RTC GPIO" support, which functions when GPIOs are routed to the "RTC" low-power and analog subsystem. These pin functions can be used when in deep sleep, when the *Ultra Low Power co-processor* is running, or when analog functions such as ADC/DAC/etc are in use.

Application Example

GPIO output and input interrupt example: peripherals/gpio.

API Reference - Normal GPIO

Header File

- driver/include/driver/gpio.h

Functions

esp_err_t **gpio_config**(**const** *gpio_config_t* **pGPIOConfig*)

GPIO common configuration.

Configure GPIO' s Mode,pull-up,PullDown,IntrType

Return

- ESP_OK success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `pGPIOConfig`: Pointer to GPIO configure struct

esp_err_t **gpio_reset_pin**(*gpio_num_t gpio_num*)

Reset an gpio to default state (select gpio function, enable pullup and disable input and output).

Note This function also configures the IOMUX for this pin to the GPIO function, and disconnects any other peripheral output configured via GPIO Matrix.

Return Always return ESP_OK.

Parameters

- `gpio_num`: GPIO number.

esp_err_t **gpio_set_intr_type**(*gpio_num_t gpio_num*, *gpio_int_type_t intr_type*)

GPIO set interrupt trigger type.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `gpio_num`: GPIO number. If you want to set the trigger type of e.g. of GPIO16, gpio_num should be GPIO_NUM_16 (16);
- `intr_type`: Interrupt type, select from gpio_int_type_t

esp_err_t **gpio_intr_enable**(*gpio_num_t gpio_num*)

Enable GPIO module interrupt signal.

Note Please do not use the interrupt of GPIO36 and GPIO39 when using ADC. Please refer to the comments of `adc1_get_raw`. Please refer to section 3.11 of 'ECO_and_Workarounds_for_Bugs_in_ESP32' for the description of this issue.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `gpio_num`: GPIO number. If you want to enable an interrupt on e.g. GPIO16, gpio_num should be GPIO_NUM_16 (16);

esp_err_t **gpio_intr_disable**(*gpio_num_t gpio_num*)

Disable GPIO module interrupt signal.

Return

- ESP_OK success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `gpio_num`: GPIO number. If you want to disable the interrupt of e.g. GPIO16, gpio_num should be GPIO_NUM_16 (16);

esp_err_t **gpio_set_level**(*gpio_num_t gpio_num*, uint32_t *level*)

GPIO set output level.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG GPIO number error

Parameters

- `gpio_num`: GPIO number. If you want to set the output level of e.g. GPIO16, gpio_num should be GPIO_NUM_16 (16);
- `level`: Output level. 0: low ; 1: high

int **gpio_get_level**(*gpio_num_t gpio_num*)

GPIO get input level.

Warning If the pad is not configured for input (or input and output) the returned value is always 0.

Return

- 0 the GPIO input level is 0
- 1 the GPIO input level is 1

Parameters

- `gpio_num`: GPIO number. If you want to get the logic level of e.g. pin GPIO16, gpio_num should be GPIO_NUM_16 (16);

esp_err_t **gpio_set_direction**(*gpio_num_t gpio_num*, *gpio_mode_t mode*)

GPIO set direction.

Configure GPIO direction,such as output_only,input_only,output_and_input

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG GPIO error

Parameters

- `gpio_num`: Configure GPIO pins number, it should be GPIO number. If you want to set direction of e.g. GPIO16, gpio_num should be GPIO_NUM_16 (16);
- `mode`: GPIO direction

esp_err_t **gpio_set_pull_mode**(*gpio_num_t gpio_num*, *gpio_pull_mode_t pull*)

Configure GPIO pull-up/pull-down resistors.

Only pins that support both input & output have integrated pull-up and pull-down resistors. Input-only GPIOs 34-39 do not.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG : Parameter error

Parameters

- `gpio_num`: GPIO number. If you want to set pull up or down mode for e.g. GPIO16, gpio_num should be GPIO_NUM_16 (16);
- `pull`: GPIO pull up/down mode.

esp_err_t **gpio_wakeup_enable**(*gpio_num_t gpio_num*, *gpio_int_type_t intr_type*)

Enable GPIO wake-up function.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `gpio_num`: GPIO number.

- `intr_type`: GPIO wake-up type. Only GPIO_INTR_LOW_LEVEL or GPIO_INTR_HIGH_LEVEL can be used.

esp_err_t **gpio_wakeup_disable**(*gpio_num_t gpio_num*)

Disable GPIO wake-up function.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `gpio_num`: GPIO number

esp_err_t **gpio_isr_register**(void (**fn*))void *

, void **arg*, int *intr_alloc_flags*, *gpio_isr_handle_t* **handle*Register GPIO interrupt handler, the handler is an ISR. The handler will be attached to the same CPU core that this function is running on.

This ISR function is called whenever any GPIO interrupt occurs. See the alternative gpio_install_isr_service() and gpio_isr_handler_add() API in order to have the driver support per-GPIO ISRs.

To disable or remove the ISR, pass the returned handle to the *interrupt allocation functions*.

Parameters

- `fn`: Interrupt handler function.
- `intr_alloc_flags`: Flags used to allocate the interrupt. One or multiple (ORred) ESP_INTR_FLAG_* values. See esp_intr_alloc.h for more info.
- `arg`: Parameter for handler function
- `handle`: Pointer to return handle. If non-NULL, a handle for the interrupt will be returned here.

Return

- ESP_OK Success ;
- ESP_ERR_INVALID_ARG GPIO error
- ESP_ERR_NOT_FOUND No free interrupt found with the specified flags

esp_err_t **gpio_pullup_en**(*gpio_num_t gpio_num*)

Enable pull-up on GPIO.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- gpio_num: GPIO number

esp_err_t **gpio_pullup_dis**(*gpio_num_t gpio_num*)

Disable pull-up on GPIO.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- gpio_num: GPIO number

esp_err_t **gpio_pulldown_en**(*gpio_num_t gpio_num*)

Enable pull-down on GPIO.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- gpio_num: GPIO number

esp_err_t **gpio_pulldown_dis**(*gpio_num_t gpio_num*)

Disable pull-down on GPIO.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- gpio_num: GPIO number

esp_err_t **gpio_install_isr_service**(int *intr_alloc_flags*)

Install the driver's GPIO ISR handler service, which allows per-pin GPIO interrupt handlers.

This function is incompatible with gpio_isr_register() - if that function is used, a single global ISR is registered for all GPIO interrupts. If this function is used, the ISR service provides a global GPIO ISR and individual pin handlers are registered via the gpio_isr_handler_add() function.

Return

- ESP_OK Success
- ESP_ERR_NO_MEM No memory to install this service
- ESP_ERR_INVALID_STATE ISR service already installed.
- ESP_ERR_NOT_FOUND No free interrupt found with the specified flags
- ESP_ERR_INVALID_ARG GPIO error

Parameters

- `intr_alloc_flags`: Flags used to allocate the interrupt. One or multiple (ORred) ESP_INTR_FLAG_* values. See esp_intr_alloc.h for more info.

void **gpio_uninstall_isr_service**(void)

Uninstall the driver's GPIO ISR service, freeing related resources.

esp_err_t **gpio_isr_handler_add**(*gpio_num_t gpio_num*, *gpio_isr_t isr_handler*, void **args*)

Add ISR handler for the corresponding GPIO pin.

Call this function after using gpio_install_isr_service() to install the driver's GPIO ISR handler service.

The pin ISR handlers no longer need to be declared with IRAM_ATTR, unless you pass the ESP_INTR_FLAG_IRAM flag when allocating the ISR in gpio_install_isr_service().

This ISR handler will be called from an ISR. So there is a stack size limit (configurable as "ISR stack size" in menuconfig). This limit is smaller compared to a global GPIO interrupt handler due to the additional level of indirection.

Return

- ESP_OK Success
- ESP_ERR_INVALID_STATE Wrong state, the ISR service has not been initialized.
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `gpio_num`: GPIO number
- `isr_handler`: ISR handler function for the corresponding GPIO number.
- `args`: parameter for ISR handler.

esp_err_t **gpio_isr_handler_remove**(*gpio_num_t gpio_num*)

Remove ISR handler for the corresponding GPIO pin.

Return

- ESP_OK Success
- ESP_ERR_INVALID_STATE Wrong state, the ISR service has not been initialized.
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `gpio_num`: GPIO number

esp_err_t **gpio_set_drive_capability**(*gpio_num_t gpio_num*, *gpio_drive_cap_t strength*)

Set GPIO pad drive capability.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `gpio_num`: GPIO number, only support output GPIOs
- `strength`: Drive capability of the pad

esp_err_t **gpio_get_drive_capability**(*gpio_num_t gpio_num*, *gpio_drive_cap_t *strength*)

Get GPIO pad drive capability.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `gpio_num`: GPIO number, only support output GPIOs
- `strength`: Pointer to accept drive capability of the pad

esp_err_t **gpio_hold_en**(*gpio_num_t gpio_num*)

Enable gpio pad hold function.

The gpio pad hold function works in both input and output modes, but must be output-capable gpios. If pad hold enabled: in output mode: the output level of the pad will be force locked and can not be changed. in input mode: the input value read will not change, regardless the changes of input signal.

The state of digital gpio cannot be held during Deep-sleep, and it will resume the hold function when the chip wakes up from Deep-sleep. If the digital gpio also needs to be held during Deep-sleep, `gpio_deep_sleep_hold_en` should also be called.

Power down or call gpio_hold_dis will disable this function.

Return

- ESP_OK Success
- ESP_ERR_NOT_SUPPORTED Not support pad hold function

Parameters

- `gpio_num`: GPIO number, only support output-capable GPIOs

esp_err_t **gpio_hold_dis**(*gpio_num_t gpio_num*)

Disable gpio pad hold function.

When the chip is woken up from Deep-sleep, the gpio will be set to the default mode, so, the gpio will output the default level if this function is called. If you don't want the level changes, the gpio should be configured to a known state before this function is called. e.g. If you hold gpio18 high during Deep-sleep, after the chip is woken up and `gpio_hold_dis` is called, gpio18 will output low level(because gpio18 is input mode by default). If you don't want this behavior, you should configure gpio18 as output mode and set it to hight level before calling `gpio_hold_dis`.

Return

- ESP_OK Success
- ESP_ERR_NOT_SUPPORTED Not support pad hold function

Parameters

- `gpio_num`: GPIO number, only support output-capable GPIOs

void **gpio_deep_sleep_hold_en**(void)

Enable all digital gpio pad hold function during Deep-sleep.

When the chip is in Deep-sleep mode, all digital gpio will hold the state before sleep, and when the chip is woken up, the status of digital gpio will not be held. Note that the pad hold feature only works when the chip is in Deep-sleep mode, when not in sleep mode, the digital gpio state can be changed even you have called this function.

Power down or call gpio_hold_dis will disable this function, otherwise, the digital gpio hold feature works as long as the chip enter Deep-sleep.

void **gpio_deep_sleep_hold_dis**(void)

Disable all digital gpio pad hold function during Deep-sleep.

void **gpio_iomux_in**(uint32_t *gpio_num*, uint32_t *signal_idx*)

Set pad input to a peripheral signal through the IOMUX.

Parameters

- `gpio_num`: GPIO number of the pad.

- `signal_idx`: Peripheral signal id to input. One of the `*_IN_IDX` signals in `soc/gpio_sig_map.h`.

void **gpio_iomux_out**(uint8_t *gpio_num*, int *func*, bool *oen_inv*)

Set peripheral output to an GPIO pad through the IOMUX.

Parameters

- `gpio_num`: gpio_num GPIO number of the pad.
- `func`: The function number of the peripheral pin to output pin. One of the `FUNC_X_*` of specified pin (X) in `soc/io_mux_reg.h`.
- `oen_inv`: True if the output enable needs to be inverted, otherwise False.

Type Definitions

typedef *intr_handle_t* **gpio_isr_handle_t**

Header File

- soc/include/hal/gpio_types.h

Structures

struct gpio_config_t

Configuration parameters of GPIO pad for gpio_config function.

Public Members

uint64_t **pin_bit_mask**

GPIO pin: set with bit mask, each bit maps to a GPIO

gpio_mode_t **mode**

GPIO mode: set input/output mode

gpio_pullup_t **pull_up_en**

GPIO pull-up

gpio_pulldown_t **pull_down_en**

GPIO pull-down

gpio_int_type_t **intr_type**

GPIO interrupt type

Type Definitions

typedef void (*gpio_isr_t)(void *)

Enumerations

enum gpio_port_t

Values:

GPIO_PORT_0 = 0

GPIO_PORT_MAX

enum gpio_num_t

Values:

GPIO_NUM_NC = -1

Use to signal not connected to S/W

GPIO_NUM_0 = 0

GPIO0, input and output

GPIO_NUM_1 = 1

GPIO1, input and output

GPIO_NUM_2 = 2

GPIO2, input and output

GPIO_NUM_3 = 3

GPIO3, input and output

GPIO_NUM_4 = 4

GPIO4, input and output

GPIO_NUM_5 = 5

GPIO5, input and output

GPIO_NUM_6 = 6

GPIO6, input and output

GPIO_NUM_7 = 7

GPIO7, input and output

GPIO_NUM_8 = 8

GPIO8, input and output

GPIO_NUM_9 = 9

GPIO9, input and output

GPIO_NUM_10 = 10
GPIO10, input and output

GPIO_NUM_11 = 11
GPIO11, input and output

GPIO_NUM_12 = 12
GPIO12, input and output

GPIO_NUM_13 = 13
GPIO13, input and output

GPIO_NUM_14 = 14
GPIO14, input and output

GPIO_NUM_15 = 15
GPIO15, input and output

GPIO_NUM_16 = 16
GPIO16, input and output

GPIO_NUM_17 = 17
GPIO17, input and output

GPIO_NUM_18 = 18
GPIO18, input and output

GPIO_NUM_19 = 19
GPIO19, input and output

GPIO_NUM_20 = 20
GPIO20, input and output

GPIO_NUM_21 = 21
GPIO21, input and output

GPIO_NUM_26 = 26
GPIO26, input and output

GPIO_NUM_27 = 27
GPIO27, input and output

GPIO_NUM_28 = 28
GPIO28, input and output

GPIO_NUM_29 = 29
GPIO29, input and output

GPIO_NUM_30 = 30
GPIO30, input and output

`GPIO_NUM_31` = 31
GPIO31, input and output

`GPIO_NUM_32` = 32
GPIO32, input and output

`GPIO_NUM_33` = 33
GPIO33, input and output

`GPIO_NUM_34` = 34
GPIO34, input mode only(ESP32) / input and output(ESP32-S2)

`GPIO_NUM_35` = 35
GPIO35, input mode only(ESP32) / input and output(ESP32-S2)

`GPIO_NUM_36` = 36
GPIO36, input mode only(ESP32) / input and output(ESP32-S2)

`GPIO_NUM_37` = 37
GPIO37, input mode only(ESP32) / input and output(ESP32-S2)

`GPIO_NUM_38` = 38
GPIO38, input mode only(ESP32) / input and output(ESP32-S2)

`GPIO_NUM_39` = 39
GPIO39, input mode only(ESP32) / input and output(ESP32-S2)

`GPIO_NUM_MAX`

`enum gpio_int_type_t`
Values:

`GPIO_INTR_DISABLE` = 0
Disable GPIO interrupt

`GPIO_INTR_POSEDGE` = 1
GPIO interrupt type : rising edge

`GPIO_INTR_NEGEDGE` = 2
GPIO interrupt type : falling edge

`GPIO_INTR_ANYEDGE` = 3
GPIO interrupt type : both rising and falling edge

`GPIO_INTR_LOW_LEVEL` = 4
GPIO interrupt type : input low level trigger

`GPIO_INTR_HIGH_LEVEL` = 5
GPIO interrupt type : input high level trigger

`GPIO_INTR_MAX`

enum gpio_mode_t

Values:

GPIO_MODE_DISABLE = GPIO_MODE_DEF_DISABLE

GPIO mode : disable input and output

GPIO_MODE_INPUT = GPIO_MODE_DEF_INPUT

GPIO mode : input only

GPIO_MODE_OUTPUT = GPIO_MODE_DEF_OUTPUT

GPIO mode : output only mode

GPIO_MODE_OUTPUT_OD = ((GPIO_MODE_DEF_OUTPUT) | (GPIO_MODE_DEF_OD))

GPIO mode : output only with open-drain mode

GPIO_MODE_INPUT_OUTPUT_OD = ((GPIO_MODE_DEF_INPUT) | (GPIO_MODE_DEF_OUTPUT) | (GPIO_MO

GPIO mode : output and input with open-drain mode

GPIO_MODE_INPUT_OUTPUT = ((GPIO_MODE_DEF_INPUT) | (GPIO_MODE_DEF_OUTPUT))

GPIO mode : output and input mode

enum gpio_pullup_t

Values:

GPIO_PULLUP_DISABLE = 0x0

Disable GPIO pull-up resistor

GPIO_PULLUP_ENABLE = 0x1

Enable GPIO pull-up resistor

enum gpio_pulldown_t

Values:

GPIO_PULLDOWN_DISABLE = 0x0

Disable GPIO pull-down resistor

GPIO_PULLDOWN_ENABLE = 0x1

Enable GPIO pull-down resistor

enum gpio_pull_mode_t

Values:

GPIO_PULLUP_ONLY

Pad pull up

GPIO_PULLDOWN_ONLY

Pad pull down

GPIO_PULLUP_PULLDOWN

Pad pull up + pull down

`GPIO_FLOATING`
Pad floating

`enum gpio_drive_cap_t`
Values:

`GPIO_DRIVE_CAP_0` = 0
Pad drive capability: weak

`GPIO_DRIVE_CAP_1` = 1
Pad drive capability: stronger

`GPIO_DRIVE_CAP_2` = 2
Pad drive capability: medium

`GPIO_DRIVE_CAP_DEFAULT` = 2
Pad drive capability: medium

`GPIO_DRIVE_CAP_3` = 3
Pad drive capability: strongest

`GPIO_DRIVE_CAP_MAX`

API Reference - RTC GPIO

Header File

- driver/include/driver/rtc_io.h

Functions

`static` bool `rtc_gpio_is_valid_gpio`(*gpio_num_t gpio_num*)
Determine if the specified GPIO is a valid RTC GPIO.

Return true if GPIO is valid for RTC GPIO use. false otherwise.

Parameters

- `gpio_num`: GPIO number

`static` int `rtc_io_number_get`(*gpio_num_t gpio_num*)
Get RTC IO index number by gpio number.

Return >=0: Index of rtcio. -1 : The gpio is not rtcio.

Parameters

- `gpio_num`: GPIO number

esp_err_t **rtc_gpio_init**(*gpio_num_t gpio_num*)

Init a GPIO as RTC GPIO.

This function must be called when initializing a pad for an analog function.

Return

- ESP_OK success
- ESP_ERR_INVALID_ARG GPIO is not an RTC IO

Parameters

- **gpio_num**: GPIO number (e.g. GPIO_NUM_12)

esp_err_t **rtc_gpio_deinit**(*gpio_num_t gpio_num*)

Init a GPIO as digital GPIO.

Return

- ESP_OK success
- ESP_ERR_INVALID_ARG GPIO is not an RTC IO

Parameters

- **gpio_num**: GPIO number (e.g. GPIO_NUM_12)

uint32_t **rtc_gpio_get_level**(*gpio_num_t gpio_num*)

Get the RTC IO input level.

Return

- 1 High level
- 0 Low level
- ESP_ERR_INVALID_ARG GPIO is not an RTC IO

Parameters

- **gpio_num**: GPIO number (e.g. GPIO_NUM_12)

esp_err_t **rtc_gpio_set_level**(*gpio_num_t gpio_num*, uint32_t *level*)

Set the RTC IO output level.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG GPIO is not an RTC IO

Parameters

- `gpio_num`: GPIO number (e.g. GPIO_NUM_12)
- `level`: output level

esp_err_t **rtc_gpio_set_direction**(*gpio_num_t gpio_num*, *rtc_gpio_mode_t mode*)

RTC GPIO set direction.

Configure RTC GPIO direction, such as output only, input only, output and input.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG GPIO is not an RTC IO

Parameters

- `gpio_num`: GPIO number (e.g. GPIO_NUM_12)
- `mode`: GPIO direction

esp_err_t **rtc_gpio_set_direction_in_sleep**(*gpio_num_t gpio_num*, *rtc_gpio_mode_t mode*)

RTC GPIO set direction in deep sleep mode or disable sleep status (default). In some application scenarios, IO needs to have another states during deep sleep.

NOTE: ESP32 support INPUT_ONLY mode. ESP32S2 support INPUT_ONLY, OUTPUT_ONLY, INPUT_OUTPUT mode.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG GPIO is not an RTC IO

Parameters

- `gpio_num`: GPIO number (e.g. GPIO_NUM_12)
- `mode`: GPIO direction

esp_err_t **rtc_gpio_pullup_en**(*gpio_num_t gpio_num*)

RTC GPIO pullup enable.

This function only works for RTC IOs. In general, call gpio_pullup_en, which will work both for normal GPIOs and RTC IOs.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG GPIO is not an RTC IO

Parameters

- gpio_num: GPIO number (e.g. GPIO_NUM_12)

esp_err_t **rtc_gpio_pulldown_en**(*gpio_num_t gpio_num*)

RTC GPIO pulldown enable.

This function only works for RTC IOs. In general, call gpio_pulldown_en, which will work both for normal GPIOs and RTC IOs.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG GPIO is not an RTC IO

Parameters

- gpio_num: GPIO number (e.g. GPIO_NUM_12)

esp_err_t **rtc_gpio_pullup_dis**(*gpio_num_t gpio_num*)

RTC GPIO pullup disable.

This function only works for RTC IOs. In general, call gpio_pullup_dis, which will work both for normal GPIOs and RTC IOs.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG GPIO is not an RTC IO

Parameters

- gpio_num: GPIO number (e.g. GPIO_NUM_12)

esp_err_t **rtc_gpio_pulldown_dis**(*gpio_num_t gpio_num*)

RTC GPIO pulldown disable.

This function only works for RTC IOs. In general, call gpio_pulldown_dis, which will work both for normal GPIOs and RTC IOs.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG GPIO is not an RTC IO

Parameters

- gpio_num: GPIO number (e.g. GPIO_NUM_12)

esp_err_t **rtc_gpio_hold_en**(*gpio_num_t gpio_num*)

Enable hold function on an RTC IO pad.

Enabling HOLD function will cause the pad to latch current values of input enable, output enable, output value, function, drive strength values. This function is useful when going into light or deep sleep mode to prevent the pin configuration from changing.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG GPIO is not an RTC IO

Parameters

- `gpio_num`: GPIO number (e.g. GPIO_NUM_12)

esp_err_t **rtc_gpio_hold_dis**(*gpio_num_t gpio_num*)

Disable hold function on an RTC IO pad.

Disabling hold function will allow the pad receive the values of input enable, output enable, output value, function, drive strength from RTC_IO peripheral.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG GPIO is not an RTC IO

Parameters

- `gpio_num`: GPIO number (e.g. GPIO_NUM_12)

esp_err_t **rtc_gpio_isolate**(*gpio_num_t gpio_num*)

Helper function to disconnect internal circuits from an RTC IO This function disables input, output, pullup, pulldown, and enables hold feature for an RTC IO. Use this function if an RTC IO needs to be disconnected from internal circuits in deep sleep, to minimize leakage current.

In particular, for ESP32-WROVER module, call rtc_gpio_isolate(GPIO_NUM_12) before entering deep sleep, to reduce deep sleep current.

Return

- ESP_OK on success
- ESP_ERR_INVALID_ARG if GPIO is not an RTC IO

Parameters

- `gpio_num`: GPIO number (e.g. GPIO_NUM_12).

esp_err_t **rtc_gpio_force_hold_all**(void)

Enable force hold signal for all RTC IOs.

Each RTC pad has a "force hold" input signal from the RTC controller. If this signal is set, pad latches current values of input enable, function, output enable, and other signals which come from the RTC mux. Force hold signal is enabled before going into deep sleep for pins which are used for EXT1 wakeup.

esp_err_t **rtc_gpio_force_hold_dis_all**(void)

Disable force hold signal for all RTC IOs.

esp_err_t **rtc_gpio_set_drive_capability**(*gpio_num_t gpio_num*, *gpio_drive_cap_t strength*)

Set RTC GPIO pad drive capability.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `gpio_num`: GPIO number, only support output GPIOs
- `strength`: Drive capability of the pad

esp_err_t **rtc_gpio_get_drive_capability**(*gpio_num_t gpio_num*, *gpio_drive_cap_t *strength*)

Get RTC GPIO pad drive capability.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `gpio_num`: GPIO number, only support output GPIOs
- `strength`: Pointer to accept drive capability of the pad

esp_err_t **rtc_gpio_wakeup_enable**(*gpio_num_t gpio_num*, *gpio_int_type_t intr_type*)

Enable wakeup from sleep mode using specific GPIO.

Return

- ESP_OK on success
- ESP_ERR_INVALID_ARG if gpio_num is not an RTC IO, or intr_type is not one of GPIO_INTR_HIGH_LEVEL, GPIO_INTR_LOW_LEVEL.

Parameters

- `gpio_num`: GPIO number

- `intr_type`: Wakeup on high level (GPIO_INTR_HIGH_LEVEL) or low level (GPIO_INTR_LOW_LEVEL)

esp_err_t **rtc_gpio_wakeup_disable**(*gpio_num_t gpio_num*)

Disable wakeup from sleep mode using specific GPIO.

Return

- ESP_OK on success
- ESP_ERR_INVALID_ARG if gpio_num is not an RTC IO

Parameters

- `gpio_num`: GPIO number

Macros

`RTC_GPIO_IS_VALID_GPIO`(gpio_num)

Header File

- soc/include/hal/rtc_io_types.h

Enumerations

`enum rtc_gpio_mode_t`

RTCIO output/input mode type.

Values:

`RTC_GPIO_MODE_INPUT_ONLY`

Pad input

`RTC_GPIO_MODE_OUTPUT_ONLY`

Pad output

`RTC_GPIO_MODE_INPUT_OUTPUT`

Pad input + output

`RTC_GPIO_MODE_DISABLED`

Pad (output + input) disable

`RTC_GPIO_MODE_OUTPUT_OD`

Pad open-drain output

`RTC_GPIO_MODE_INPUT_OUTPUT_OD`

Pad input + open-drain output

2.3.5 I2C Driver

Overview

I2C is a serial, synchronous, half-duplex communication protocol that allows co-existence of multiple masters and slaves on the same bus. The I2C bus consists of two lines: serial data line (SDA) and serial clock (SCL). Both lines require pull-up resistors.

With such advantages as simplicity and low manufacturing cost, I2C is mostly used for communication of low-speed peripheral devices over short distances (within one foot).

ESP32 has two I2C controllers (also referred to as ports) which are responsible for handling communications on two I2C buses. Each I2C controller can operate as master or slave. As an example, one controller can act as a master and the other as a slave at the same time.

Driver Features

I2C driver governs communications of devices over the I2C bus. The driver supports the following features:

- Reading and writing bytes in Master mode
- Slave mode
- Reading and writing to registers which are in turn read/written by the master

Driver Usage

The following sections describe typical steps of configuring and operating the I2C driver:

1. *Configuration* - set the initialization parameters (master or slave mode, GPIO pins for SDA and SCL, clock speed, etc.)
2. *Install Driver*- activate the driver on one of the two I2C controllers as a master or slave
3. Depending on whether you configure the driver for a master or slave, choose the appropriate item
   a) *Communication as Master* - handle communications (master)
   b) *Communication as Slave* - respond to messages from the master (slave)
4. *Interrupt Handling* - configure and service I2C interrupts
5. *Customized Configuration* - adjust default I2C communication parameters (timings, bit order, etc.)
6. *Error Handling* - how to recognize and handle driver configuration and communication errors
7. *Delete Driver*- release resources used by the I2C driver when communication ends

Configuration

To establish I2C communication, start by configuring the driver. This is done by setting the parameters of the structure *i2c_config_t*:

- Set I2C **mode of operation** - slave or master from *i2c_mode_t*
- Configure **communication pins**
  - Assign GPIO pins for SDA and SCL signals
  - Set whether to enable ESP32' s internal pull-ups
- (Master only) Set I2C **clock speed**
- (Slave only) Configure the following
  - Whether to enable **10 bit address mode**
  - Define **slave address**

After that, initialize the configuration for a given I2C port. For this, call the function *i2c_param_config()* and pass to it the port number and the structure *i2c_config_t*.

At this stage, *i2c_param_config()* also sets a few other I2C configuration parameters to default values that are defined by the I2C specification. For more details on the values and how to modify them, see *Customized Configuration*.

Install Driver

After the I2C driver is configured, install it by calling the function *i2c_driver_install()* with the following parameters:

- Port number, one of the two port numbers from *i2c_port_t*
- Master or slave, selected from *i2c_mode_t*
- (Slave only) Size of buffers to allocate for sending and receiving data. As I2C is a master-centric bus, data can only go from the slave to the master at the master' s request. Therefore, the slave will usually have a send buffer where the slave application writes data. The data remains in the send buffer to be read by the master at the master' s own discretion.
- Flags for allocating the interrupt (see ESP_INTR_FLAG_* values in esp32/include/esp_intr_alloc.h)

Communication as Master

After installing the I2C driver, ESP32 is ready to communicate with other I2C devices.

ESP32' s I2C controller operating as master is responsible for establishing communication with I2C slave devices and sending commands to trigger a slave to action, for example, to take a measurement and send the readings back to the master.

For better process organization, the driver provides a container, called a "command link", that should be populated with a sequence of commands and then passed to the I2C controller for execution.

Master Write

The example below shows how to build a command link for an I2C master to send n bytes to a slave.

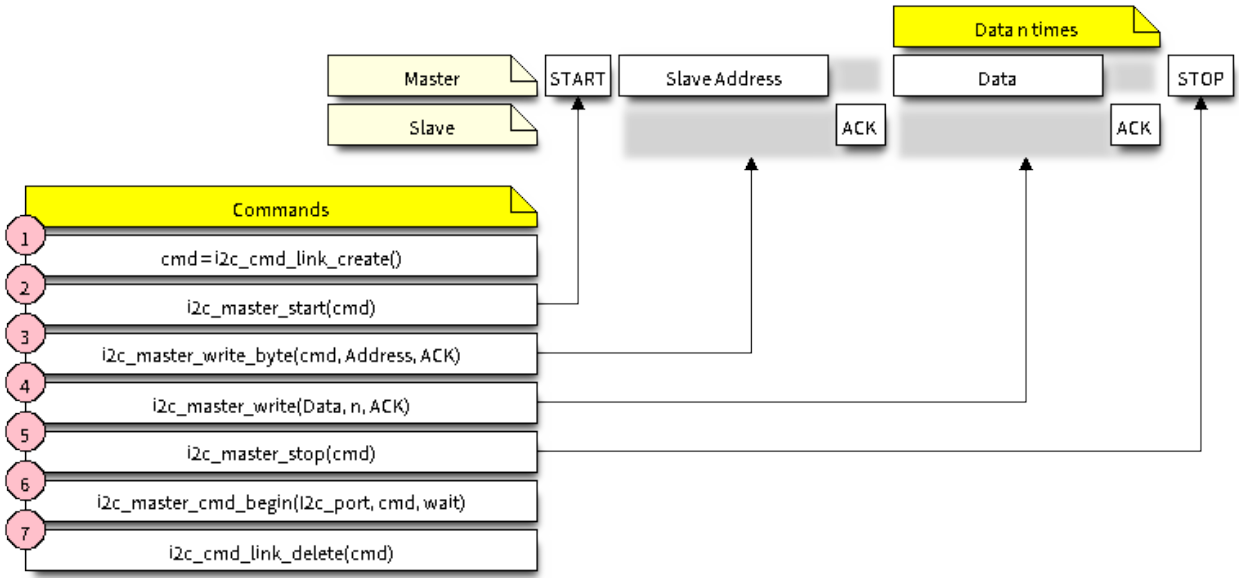


图 12: I2C command link - master write example

The following describes how a command link for a "master write" is set up and what comes inside:

1. Create a command link with *i2c_cmd_link_create()*.

   Then, populate it with the series of data to be sent to the slave:

   a) **Start bit** - *i2c_master_start()*

   b) **Slave address** - *i2c_master_write_byte()*. The single byte address is provided as an argument of this function call.

   c) **Data** - One or more bytes as an argument of *i2c_master_write()*

   d) **Stop bit** - *i2c_master_stop()*

   Both functions *i2c_master_write_byte()* and *i2c_master_write()* have an additional argument specifying whether the master should ensure that it has received the ACK bit.

2. Trigger the execution of the command link by I2C controller by calling *i2c_master_cmd_begin()*. Once the execution is triggered, the command link cannot be modified.

3. After the commands are transmitted, release the resources used by the command link by calling *i2c_cmd_link_delete()*.

Master Read

The example below shows how to build a command link for an I2C master to read n bytes from a slave.

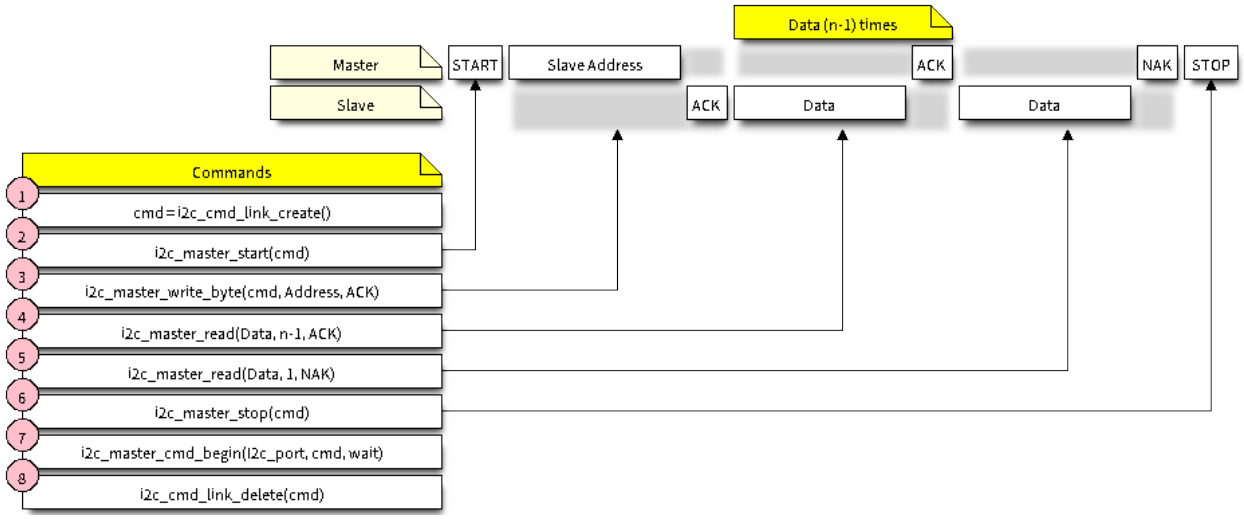


图 13: I2C command link - master read example

Compared to writing data, the command link is populated in Step 4 not with `i2c_master_write...` functions but with *i2c_master_read_byte()* and / or *i2c_master_read()*. Also, the last read in Step 5 is configured so that the master does not provide the ACK bit.

Indicating Write or Read

After sending a slave address (see Step 3 on both diagrams above), the master either writes or reads from the slave.

The information on what the master will actually do is hidden in the least significant bit of the slave' s address.

For this reason, the command link sent by the master to write data to the slave contains the address `(ESP_SLAVE_ADDR << 1) | I2C_MASTER_WRITE` and looks as follows:

```
i2c_master_write_byte(cmd, (ESP_SLAVE_ADDR << 1) | I2C_MASTER_WRITE, ACK_EN);
```

Likewise, the command link to read from the slave looks as follows:

```
i2c_master_write_byte(cmd, (ESP_SLAVE_ADDR << 1) | I2C_MASTER_READ, ACK_EN);
```

Communication as Slave

After installing the I2C driver, ESP32 is ready to communicate with other I2C devices.

The API provides the following functions for slaves

- *i2c_slave_read_buffer()*

  Whenever the master writes data to the slave, the slave will automatically store it in the receive buffer. This allows the slave application to call the function *i2c_slave_read_buffer()* at its own discretion. This function also has a parameter to specify block time if no data is in the receive buffer. This will allow the slave application to wait with a specified timeout for data to arrive to the buffer.

- *i2c_slave_write_buffer()*

  The send buffer is used to store all the data that the slave wants to send to the master in FIFO order. The data stays there until the master requests for it. The function *i2c_slave_write_buffer()* has a parameter to specify block time if the send buffer is full. This will allow the slave application to wait with a specified timeout for the adequate amount of space to become available in the send buffer.

A code example showing how to use these functions can be found in peripherals/i2c.

Interrupt Handling

During driver installation, an interrupt handler is installed by default. However, you can register your own interrupt handler instead of the default one by calling the function *i2c_isr_register()*. When implementing your own interrupt handler, refer to the ESP32 Technical Reference Manual for the description of interrupts triggered by the I2C controller.

To delete an interrupt handler, call *i2c_isr_free()*.

Customized Configuration

As mentioned at the end of Section *Configuration*, when the function *i2c_param_config()* initializes the driver configuration for an I2C port, it also sets several I2C communication parameters to default values defined in the I2C specification. Some other related parameters are pre-configured in registers of the I2C controller.

All these parameters can be changed to user-defined values by calling dedicated functions given in the table below. Please note that the timing values are defined in APB clock cycles. The frequency of APB is specified in `I2C_APB_CLK_FREQ`.

表 1: Other Configurable I2C Communication Parameters

Parameters to Change	Function
High time and low time for SCL pulses	*i2c_set_period()*
SCL and SDA signal timing used during generation of **start** signals	*i2c_set_start_timing()*
SCL and SDA signal timing used during generation of **stop** signals	*i2c_set_stop_timing()*
Timing relationship between SCL and SDA signals when slave samples, as well as when master toggles	*i2c_set_data_timing()*
I2C timeout	*i2c_set_timeout()*
Choice between transmitting / receiving the LSB or MSB first, choose one of the modes defined in *i2c_trans_mode_t*	*i2c_set_data_mode()*

Each of the above functions has a *_get_* counterpart to check the currently set value. For example, to check the I2C timeout value, call *i2c_get_timeout()*.

To check the default parameter values which are set during the driver configuration process, please refer to the file driver/i2c.c and look for defines with the suffix `_DEFAULT`.

You can also select different pins for SDA and SCL signals and alter the configuration of pull-ups with the function *i2c_set_pin()*. If you want to modify already entered values, use the function *i2c_param_config()*.

注解: ESP32’ s internal pull-ups are in the range of tens of kOhm, which is, in most cases, insufficient for use as I2C pull-ups. Users are advised to use external pull-ups with values described in the I2C specification.

Error Handling

The majority of I2C driver functions either return `ESP_OK` on successful completion or a specific error code on failure. It is a good practice to always check the returned values and implement error handling. The driver also prints out log messages that contain error details, e.g., when checking the validity of entered configuration. For details please refer to the file driver/i2c.c and look for defines with the suffix `_ERR_STR`.

Use dedicated interrupts to capture communication failures. For instance, if a slave stretches the clock for too long while preparing the data to send back to master, the interrupt `I2C_TIME_OUT_INT` will be triggered. For detailed information, see *Interrupt Handling*.

In case of a communication failure, you can reset the internal hardware buffers by calling the functions *i2c_reset_tx_fifo()* and *i2c_reset_rx_fifo()* for the send and receive buffers respectively.

Delete Driver

When the I2C communication is established with the function *i2c_driver_install()* and is not required for some substantial amount of time, the driver may be deinitialized to release allocated resources by calling *i2c_driver_delete()*.

Application Example

I2C master and slave example: peripherals/i2c.

API Reference

Header File

- driver/include/driver/i2c.h

Functions

esp_err_t **i2c_driver_install**(*i2c_port_t i2c_num*, *i2c_mode_t mode*, size_t *slv_rx_buf_len*, size_t *slv_tx_buf_len*, int *intr_alloc_flags*)

I2C driver install.

Note Only slave mode will use this value, driver will ignore this value in master mode.

Note Only slave mode will use this value, driver will ignore this value in master mode.

Note In master mode, if the cache is likely to be disabled(such as write flash) and the slave is time-sensitive, `ESP_INTR_FLAG_IRAM` is suggested to be used. In this case, please use the memory allocated from internal RAM in i2c read and write function, because we can not access the psram(if psram is enabled) in interrupt handle function when cache is disabled.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_FAIL Driver install error

Parameters

- `i2c_num`: I2C port number
- `mode`: I2C mode(master or slave)
- `slv_rx_buf_len`: receiving buffer size for slave mode

Parameters

- `slv_tx_buf_len`: sending buffer size for slave mode

Parameters

- `intr_alloc_flags`: Flags used to allocate the interrupt. One or multiple (ORred) ESP_INTR_FLAG_* values. See esp_intr_alloc.h for more info.

esp_err_t **i2c_driver_delete**(*i2c_port_t i2c_num*)

I2C driver delete.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `i2c_num`: I2C port number

esp_err_t **i2c_param_config**(*i2c_port_t i2c_num*, **const** *i2c_config_t* **i2c_conf*)

I2C parameter initialization.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `i2c_num`: I2C port number
- `i2c_conf`: pointer to I2C parameter settings

esp_err_t **i2c_reset_tx_fifo**(*i2c_port_t i2c_num*)

reset I2C tx hardware fifo

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `i2c_num`: I2C port number

esp_err_t **i2c_reset_rx_fifo**(*i2c_port_t i2c_num*)

reset I2C rx fifo

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `i2c_num`: I2C port number

esp_err_t **i2c_isr_register**(*i2c_port_t i2c_num*, void (**fn*))void *
, void **arg*, int *intr_alloc_flags*, *intr_handle_t* **handle*I2C isr handler register.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `i2c_num`: I2C port number
- `fn`: isr handler function
- `arg`: parameter for isr handler function
- `intr_alloc_flags`: Flags used to allocate the interrupt. One or multiple (ORred) ESP_INTR_FLAG_* values. See esp_intr_alloc.h for more info.
- `handle`: handle return from esp_intr_alloc.

esp_err_t **i2c_isr_free**(*intr_handle_t handle*)
to delete and free I2C isr.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `handle`: handle of isr.

esp_err_t **i2c_set_pin**(*i2c_port_t i2c_num*, int *sda_io_num*, int *scl_io_num*, bool *sda_pullup_en*, bool *scl_pullup_en*, *i2c_mode_t mode*)
Configure GPIO signal for I2C sck and sda.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `i2c_num`: I2C port number
- `sda_io_num`: GPIO number for I2C sda signal
- `scl_io_num`: GPIO number for I2C scl signal
- `sda_pullup_en`: Whether to enable the internal pullup for sda pin
- `scl_pullup_en`: Whether to enable the internal pullup for scl pin
- `mode`: I2C mode

i2c_cmd_handle_t **i2c_cmd_link_create**(void)

Create and init I2C command link.

Note Before we build I2C command link, we need to call i2c_cmd_link_create() to create a command link. After we finish sending the commands, we need to call i2c_cmd_link_delete() to release and return the resources.

Return i2c command link handler

void **i2c_cmd_link_delete**(*i2c_cmd_handle_t cmd_handle*)

Free I2C command link.

Note Before we build I2C command link, we need to call i2c_cmd_link_create() to create a command link. After we finish sending the commands, we need to call i2c_cmd_link_delete() to release and return the resources.

Parameters

- `cmd_handle`: I2C command handle

esp_err_t **i2c_master_start**(*i2c_cmd_handle_t cmd_handle*)

Queue command for I2C master to generate a start signal.

Note Only call this function in I2C master mode Call i2c_master_cmd_begin() to send all queued commands

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `cmd_handle`: I2C cmd link

esp_err_t **i2c_master_write_byte**(*i2c_cmd_handle_t cmd_handle*, uint8_t *data*, bool *ack_en*)

Queue command for I2C master to write one byte to I2C bus.

Note Only call this function in I2C master mode Call i2c_master_cmd_begin() to send all queued commands

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `cmd_handle`: I2C cmd link
- `data`: I2C one byte command to write to bus
- `ack_en`: enable ack check for master

esp_err_t **i2c_master_write**(*i2c_cmd_handle_t cmd_handle*, uint8_t **data*, size_t *data_len*, bool *ack_en*)

Queue command for I2C master to write buffer to I2C bus.

Note Only call this function in I2C master mode Call i2c_master_cmd_begin() to send all queued commands

Note If the psram is enabled and intr_flag is `ESP_INTR_FLAG_IRAM`, please use the memory allocated from internal RAM.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `cmd_handle`: I2C cmd link
- `data`: data to send

Parameters

- `data_len`: data length
- `ack_en`: enable ack check for master

esp_err_t **i2c_master_read_byte**(*i2c_cmd_handle_t cmd_handle*, uint8_t **data*, *i2c_ack_type_t ack*)

Queue command for I2C master to read one byte from I2C bus.

Note Only call this function in I2C master mode Call i2c_master_cmd_begin() to send all queued commands

Note If the psram is enabled and intr_flag is `ESP_INTR_FLAG_IRAM`, please use the memory allocated from internal RAM.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `cmd_handle`: I2C cmd link
- `data`: pointer accept the data byte

Parameters

- `ack`: ack value for read command

esp_err_t **i2c_master_read**(*i2c_cmd_handle_t cmd_handle*, uint8_t **data*, size_t *data_len*, *i2c_ack_type_t ack*)

Queue command for I2C master to read data from I2C bus.

Note Only call this function in I2C master mode Call i2c_master_cmd_begin() to send all queued commands

Note If the psram is enabled and intr_flag is `ESP_INTR_FLAG_IRAM`, please use the memory allocated from internal RAM.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `cmd_handle`: I2C cmd link
- `data`: data buffer to accept the data from bus

Parameters

- `data_len`: read data length
- `ack`: ack value for read command

esp_err_t **i2c_master_stop**(*i2c_cmd_handle_t cmd_handle*)

Queue command for I2C master to generate a stop signal.

Note Only call this function in I2C master mode Call i2c_master_cmd_begin() to send all queued commands

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `cmd_handle`: I2C cmd link

esp_err_t **i2c_master_cmd_begin**(*i2c_port_t i2c_num*, *i2c_cmd_handle_t cmd_handle*, TickType_t *ticks_to_wait*)

I2C master send queued commands. This function will trigger sending all queued commands. The task will be blocked until all the commands have been sent out. The I2C APIs are not thread-safe, if you want to use one I2C port in different tasks, you need to take care of the multi-thread issue.

Note Only call this function in I2C master mode

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_FAIL Sending command error, slave doesn't ACK the transfer.
- ESP_ERR_INVALID_STATE I2C driver not installed or not in master mode.
- ESP_ERR_TIMEOUT Operation timeout because the bus is busy.

Parameters

- `i2c_num`: I2C port number
- `cmd_handle`: I2C command handler
- `ticks_to_wait`: maximum wait ticks.

int **i2c_slave_write_buffer**(*i2c_port_t i2c_num*, uint8_t **data*, int *size*, TickType_t *ticks_to_wait*)

I2C slave write data to internal ringbuffer, when tx fifo empty, isr will fill the hardware fifo from the internal ringbuffer.

Note Only call this function in I2C slave mode

Return

- ESP_FAIL(-1) Parameter error
- Others(>=0) The number of data bytes that pushed to the I2C slave buffer.

Parameters

- `i2c_num`: I2C port number
- `data`: data pointer to write into internal buffer
- `size`: data size
- `ticks_to_wait`: Maximum waiting ticks

int **i2c_slave_read_buffer**(*i2c_port_t i2c_num*, uint8_t **data*, size_t *max_size*, TickType_t *ticks_to_wait*)

I2C slave read data from internal buffer. When I2C slave receive data, isr will copy received data from hardware rx fifo to internal ringbuffer. Then users can read from internal ringbuffer.

Note Only call this function in I2C slave mode

Return

- ESP_FAIL(-1) Parameter error
- Others(>=0) The number of data bytes that read from I2C slave buffer.

Parameters

- `i2c_num`: I2C port number
- `data`: data pointer to accept data from internal buffer
- `max_size`: Maximum data size to read
- `ticks_to_wait`: Maximum waiting ticks

esp_err_t **i2c_set_period**(*i2c_port_t i2c_num*, int *high_period*, int *low_period*)

set I2C master clock period

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `i2c_num`: I2C port number
- `high_period`: clock cycle number during SCL is high level, high_period is a 14 bit value
- `low_period`: clock cycle number during SCL is low level, low_period is a 14 bit value

esp_err_t **i2c_get_period**(*i2c_port_t i2c_num*, int **high_period*, int **low_period*)

get I2C master clock period

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `i2c_num`: I2C port number
- `high_period`: pointer to get clock cycle number during SCL is high level, will get a 14 bit value

- `low_period`: pointer to get clock cycle number during SCL is low level, will get a 14 bit value

esp_err_t **i2c_filter_enable**(*i2c_port_t i2c_num*, uint8_t *cyc_num*)

enable hardware filter on I2C bus Sometimes the I2C bus is disturbed by high frequency noise(about 20ns), or the rising edge of the SCL clock is very slow, these may cause the master state machine broken. enable hardware filter can filter out high frequency interference and make the master more stable.

Note Enable filter will slow the SCL clock.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `i2c_num`: I2C port number
- `cyc_num`: the APB cycles need to be filtered(0<= cyc_num <=7). When the period of a pulse is less than cyc_num * APB_cycle, the I2C controller will ignore this pulse.

esp_err_t **i2c_filter_disable**(*i2c_port_t i2c_num*)

disable filter on I2C bus

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `i2c_num`: I2C port number

esp_err_t **i2c_set_start_timing**(*i2c_port_t i2c_num*, int *setup_time*, int *hold_time*)

set I2C master start signal timing

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `i2c_num`: I2C port number
- `setup_time`: clock number between the falling-edge of SDA and rising-edge of SCL for start mark, it’ s a 10-bit value.

- `hold_time`: clock num between the falling-edge of SDA and falling-edge of SCL for start mark, it’s a 10-bit value.

esp_err_t **i2c_get_start_timing**(*i2c_port_t i2c_num*, int **setup_time*, int **hold_time*)

get I2C master start signal timing

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `i2c_num`: I2C port number
- `setup_time`: pointer to get setup time
- `hold_time`: pointer to get hold time

esp_err_t **i2c_set_stop_timing**(*i2c_port_t i2c_num*, int *setup_time*, int *hold_time*)

set I2C master stop signal timing

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `i2c_num`: I2C port number
- `setup_time`: clock num between the rising-edge of SCL and the rising-edge of SDA, it’s a 10-bit value.
- `hold_time`: clock number after the STOP bit’s rising-edge, it’s a 14-bit value.

esp_err_t **i2c_get_stop_timing**(*i2c_port_t i2c_num*, int **setup_time*, int **hold_time*)

get I2C master stop signal timing

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `i2c_num`: I2C port number
- `setup_time`: pointer to get setup time.
- `hold_time`: pointer to get hold time.

esp_err_t **i2c_set_data_timing**(*i2c_port_t i2c_num*, int *sample_time*, int *hold_time*)

set I2C data signal timing

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `i2c_num`: I2C port number
- `sample_time`: clock number I2C used to sample data on SDA after the rising-edge of SCL, it's a 10-bit value
- `hold_time`: clock number I2C used to hold the data after the falling-edge of SCL, it's a 10-bit value

esp_err_t **i2c_get_data_timing**(*i2c_port_t i2c_num*, int **sample_time*, int **hold_time*)

get I2C data signal timing

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `i2c_num`: I2C port number
- `sample_time`: pointer to get sample time
- `hold_time`: pointer to get hold time

esp_err_t **i2c_set_timeout**(*i2c_port_t i2c_num*, int *timeout*)

set I2C timeout value

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `i2c_num`: I2C port number
- `timeout`: timeout value for I2C bus (unit: APB 80Mhz clock cycle)

esp_err_t **i2c_get_timeout**(*i2c_port_t i2c_num*, int **timeout*)

get I2C timeout value

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `i2c_num`: I2C port number
- `timeout`: pointer to get timeout value

esp_err_t **i2c_set_data_mode**(*i2c_port_t i2c_num*, *i2c_trans_mode_t tx_trans_mode*, *i2c_trans_mode_t rx_trans_mode*)

set I2C data transfer mode

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `i2c_num`: I2C port number
- `tx_trans_mode`: I2C sending data mode
- `rx_trans_mode`: I2C receving data mode

esp_err_t **i2c_get_data_mode**(*i2c_port_t i2c_num*, *i2c_trans_mode_t *tx_trans_mode*, *i2c_trans_mode_t *rx_trans_mode*)

get I2C data transfer mode

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `i2c_num`: I2C port number
- `tx_trans_mode`: pointer to get I2C sending data mode
- `rx_trans_mode`: pointer to get I2C receiving data mode

Macros

`I2C_APB_CLK_FREQ`

I2C source clock is APB clock, 80MHz

`I2C_NUM_0`
: I2C port 0

`I2C_NUM_1`
: I2C port 1

`I2C_NUM_MAX`
: I2C port max

Type Definitions

`typedef void *i2c_cmd_handle_t`
: I2C command handle

Header File

- soc/include/hal/i2c_types.h

Structures

`struct i2c_config_t`
: I2C initialization parameters.

Public Members

i2c_mode_t **mode**
: I2C mode

int **sda_io_num**
: GPIO number for I2C sda signal

int **scl_io_num**
: GPIO number for I2C scl signal

bool **sda_pullup_en**
: Internal GPIO pull mode for I2C sda signal

bool **scl_pullup_en**
: Internal GPIO pull mode for I2C scl signal

uint32_t **clk_speed**
: I2C clock frequency for master mode, (no higher than 1MHz for now)

uint8_t **addr_10bit_en**
: I2C 10bit address mode enable for slave mode

uint16_t **slave_addr**

I2C address for slave mode

Type Definitions

typedef int **i2c_port_t**

I2C port number, can be I2C_NUM_0 ~ (I2C_NUM_MAX-1).

Enumerations

enum i2c_mode_t

Values:

I2C_MODE_SLAVE = 0

I2C slave mode

I2C_MODE_MASTER

I2C master mode

I2C_MODE_MAX

enum i2c_rw_t

Values:

I2C_MASTER_WRITE = 0

I2C write data

I2C_MASTER_READ

I2C read data

enum i2c_opmode_t

Values:

I2C_CMD_RESTART = 0

I2C restart command

I2C_CMD_WRITE

I2C write command

I2C_CMD_READ

I2C read command

I2C_CMD_STOP

I2C stop command

I2C_CMD_END

I2C end command

enum i2c_trans_mode_t

Values:

I2C_DATA_MODE_MSB_FIRST = 0

I2C data msb first

I2C_DATA_MODE_LSB_FIRST = 1

I2C data lsb first

I2C_DATA_MODE_MAX

enum i2c_addr_mode_t

Values:

I2C_ADDR_BIT_7 = 0

I2C 7bit address for slave mode

I2C_ADDR_BIT_10

I2C 10bit address for slave mode

I2C_ADDR_BIT_MAX

enum i2c_ack_type_t

Values:

I2C_MASTER_ACK = 0x0

I2C ack for each byte read

I2C_MASTER_NACK = 0x1

I2C nack for each byte read

I2C_MASTER_LAST_NACK = 0x2

I2C nack for the last byte

I2C_MASTER_ACK_MAX

enum i2c_sclk_t

Values:

I2C_SCLK_REF_TICK

I2C source clock from REF_TICK

I2C_SCLK_APB

I2C source clock from APB

2.3.6 I2S

Overview

I2S (Inter-IC Sound) is a serial, synchronous communication protocol that is usually used for transmitting audio data between two digital audio devices.

ESP32 integrates two I2S controllers, referred to as I2S0 and I2S1, both of which can be used for streaming audio and video digital data.

An I2S bus consists of the following lines:

- Bit clock line
- Channel select line
- Serial data line

Each I2S controller has the following features that can be configured using the I2S driver:

- Operation as system master or slave
- Capable of acting as transmitter or receiver
- Dedicated DMA controller that allows for streaming sample data without requiring the CPU to copy each data sample

Each controller can operate in half-duplex communication mode. Thus, the two controllers can be combined to establish full-duplex communication.

I2S0 output can be routed directly to the digital-to-analog converter' s (DAC) output channels (GPIO 25 & GPIO 26) to produce direct analog output without involving any external I2S codecs. I2S0 can also be used for transmitting PDM (Pulse-density modulation) signals.

The I2S peripherals also support LCD mode for communicating data over a parallel bus, as used by some LCD displays and camera modules. LCD mode has the following operational modes:

- LCD master transmitting mode
- Camera slave receiving mode
- ADC/DAC mode

For more information, see the ESP32 Technical Reference Manual.

注解: For high accuracy clock applications, use the APLL_CLK clock source, which has the frequency range of 16 ~ 128 MHz. You can enable the APLL_CLK clock source by setting *i2s_config_t::use_apll* to `TRUE`.

If *i2s_config_t::use_apll* = `TRUE` and *i2s_config_t::fixed_mclk* > `0`, then the master clock output frequency for I2S will be equal to the value of *i2s_config_t::fixed_mclk*, which means that the mclk frequency is provided by the user, instead of being calculated by the driver.

The clock rate of the word select line, which is called audio left-right clock rate (LRCK) here, is always the divisor of the master clock output frequency and for which the following is always true: 0 < MCLK/LRCK/channels/bits_per_sample < 64.

Functional Overview

Installing the Driver

Install the I2S driver by calling the function :cpp:func‘i2s_driver_install‘ and passing the following arguments:

- Port number
- The structure *i2s_config_t* with defined communication parameters
- Event queue size and handle

Configuration example:

```
static const int i2s_num = 0; // i2s port number

static const i2s_config_t i2s_config = {
    .mode = I2S_MODE_MASTER | I2S_MODE_TX,
    .sample_rate = 44100,
    .bits_per_sample = 16,
    .channel_format = I2S_CHANNEL_FMT_RIGHT_LEFT,
    .communication_format = I2S_COMM_FORMAT_I2S | I2S_COMM_FORMAT_I2S_MSB,
    .intr_alloc_flags = 0, // default interrupt priority
    .dma_buf_count = 8,
    .dma_buf_len = 64,
    .use_apll = false
};

i2s_driver_install(I2S_NUM, &i2s_config, 0, NULL);
```

Setting Communication Pins

Once the driver is installed, configure physical GPIO pins to which signals will be routed. For this, call the function :cpp:func‘i2s_set_pin‘ and pass the following arguments to it:

- Port number
- The structure *i2s_pin_config_t* defining the GPIO pin numbers to which the driver should route the BCK, WS, DATA out, and DATA in signals. If you want to keep a currently allocated pin number for a specific signal, or if this signal is unused, then pass the macro *I2S_PIN_NO_CHANGE*. See the example below.

```
static const i2s_pin_config_t pin_config = {
    .bck_io_num = 26,
    .ws_io_num = 25,
    .data_out_num = 22,
    .data_in_num = I2S_PIN_NO_CHANGE
};

i2s_set_pin(i2s_num, &pin_config);
```

Running I2S Communication

To perform a transmission:

- Prepare the data for sending
- Call the function *i2s_write()* and pass the data buffer address and data length to it

The function will write the data to the I2S DMA Tx buffer, and then the data will be transmitted automatically.

```
i2s_write(I2S_NUM, samples_data, ((bits+8)/16)*SAMPLE_PER_CYCLE*4, &i2s_bytes_write,␣
↪100);
```

To retrieve received data, use the function *i2s_read()*. It will retrieve the data from the I2S DMA Rx buffer, once the data is received by the I2S controller.

You can temporarily stop the I2S driver by calling the function *i2s_stop()*, which will disable the I2S Tx/Rx units until the function *i2s_start()* is called. If the function :cpp:func‘i2s_driver_install‘ is used, the driver will start up automatically eliminating the need to call *i2s_start()*.

Deleting the Driver

If the established communication is no longer required, the driver can be removed to free allocated resources by calling *i2s_driver_uninstall()*.

Application Example

A code example for the I2S driver can be found in the directory peripherals/i2s.

In addition, there are two short configuration examples for the I2S driver.

I2S configuration

```
#include "driver/i2s.h"
#include "freertos/queue.h"

static const int i2s_num = 0; // i2s port number

static const i2s_config_t i2s_config = {
    .mode = I2S_MODE_MASTER | I2S_MODE_TX,
    .sample_rate = 44100,
    .bits_per_sample = 16,
    .channel_format = I2S_CHANNEL_FMT_RIGHT_LEFT,
    .communication_format = I2S_COMM_FORMAT_I2S | I2S_COMM_FORMAT_I2S_MSB,
    .intr_alloc_flags = 0, // default interrupt priority
    .dma_buf_count = 8,
    .dma_buf_len = 64,
    .use_apll = false
};

static const i2s_pin_config_t pin_config = {
    .bck_io_num = 26,
    .ws_io_num = 25,
    .data_out_num = 22,
    .data_in_num = I2S_PIN_NO_CHANGE
};

...

    i2s_driver_install(i2s_num, &i2s_config, 0, NULL);   //install and start i2s driver

    i2s_set_pin(i2s_num, &pin_config);

    i2s_set_sample_rates(i2s_num, 22050); //set sample rates

    i2s_driver_uninstall(i2s_num); //stop & destroy i2s driver
```

Configuring I2S to use internal DAC for analog output

```
#include "driver/i2s.h"
```

(下页继续)

(续上页)

```
#include "freertos/queue.h"

static const int i2s_num = 0; // i2s port number

static const i2s_config_t i2s_config = {
    .mode = I2S_MODE_MASTER | I2S_MODE_TX | I2S_MODE_DAC_BUILT_IN,
    .sample_rate = 44100,
    .bits_per_sample = 16, /* the DAC module will only take the 8bits from MSB */
    .channel_format = I2S_CHANNEL_FMT_RIGHT_LEFT,
    .communication_format = I2S_COMM_FORMAT_I2S_MSB,
    .intr_alloc_flags = 0, // default interrupt priority
    .dma_buf_count = 8,
    .dma_buf_len = 64,
    .use_apll = false
};

...

    i2s_driver_install(i2s_num, &i2s_config, 0, NULL);   //install and start i2s driver

    i2s_set_pin(i2s_num, NULL); //for internal DAC, this will enable both of the␣
↪internal channels

    //You can call i2s_set_dac_mode to set built-in DAC output mode.
    //i2s_set_dac_mode(I2S_DAC_CHANNEL_BOTH_EN);

    i2s_set_sample_rates(i2s_num, 22050); //set sample rates

    i2s_driver_uninstall(i2s_num); //stop & destroy i2s driver
```

API Reference

Header File

- driver/include/driver/i2s.h

Functions

esp_err_t **i2s_set_pin**(*i2s_port_t i2s_num*, **const** *i2s_pin_config_t* **pin*)

Set I2S pin number.

Inside the pin configuration structure, set I2S_PIN_NO_CHANGE for any pin where the current configuration should not be changed.

Note The I2S peripheral output signals can be connected to multiple GPIO pads. However, the I2S peripheral input signal can only be connected to one GPIO pad.

Parameters

- `i2s_num`: I2S_NUM_0 or I2S_NUM_1
- `pin`: I2S Pin structure, or NULL to set 2-channel 8-bit internal DAC pin configuration (GPIO25 & GPIO26)

Note if *pin is set as NULL, this function will initialize both of the built-in DAC channels by default. if you don't want this to happen and you want to initialize only one of the DAC channels, you can call i2s_set_dac_mode instead.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_FAIL IO error

esp_err_t **i2s_set_dac_mode**(*i2s_dac_mode_t dac_mode*)

Set I2S dac mode, I2S built-in DAC is disabled by default.

Note Built-in DAC functions are only supported on I2S0 for current ESP32 chip. If either of the built-in DAC channel are enabled, the other one can not be used as RTC DAC function at the same time.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `dac_mode`: DAC mode configurations - see i2s_dac_mode_t

esp_err_t **i2s_driver_install**(*i2s_port_t i2s_num*, **const** *i2s_config_t* **i2s_config*, int *queue_size*, void **i2s_queue*)

Install and start I2S driver.

This function must be called before any I2S driver read/write operations.

Parameters

- `i2s_num`: I2S_NUM_0, I2S_NUM_1
- `i2s_config`: I2S configurations - see *i2s_config_t* struct

- `queue_size`: I2S event queue size/depth.
- `i2s_queue`: I2S event queue handle, if set NULL, driver will not use an event queue.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_ERR_NO_MEM Out of memory

esp_err_t **i2s_driver_uninstall**(*i2s_port_t i2s_num*)

Uninstall I2S driver.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `i2s_num`: I2S_NUM_0, I2S_NUM_1

esp_err_t **i2s_write**(*i2s_port_t i2s_num*, **const** void **src*, size_t *size*, size_t **bytes_written*, TickType_t *ticks_to_wait*)

Write data to I2S DMA transmit buffer.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `i2s_num`: I2S_NUM_0, I2S_NUM_1
- `src`: Source address to write from
- `size`: Size of data in bytes
- `bytes_written`: Number of bytes written, if timeout, the result will be less than the size passed in.
- `ticks_to_wait`: TX buffer wait timeout in RTOS ticks. If this many ticks pass without space becoming available in the DMA transmit buffer, then the function will return (note that if the data is written to the DMA buffer in pieces, the overall operation may still take longer than this timeout.) Pass portMAX_DELAY for no timeout.

esp_err_t **i2s_write_expand**(*i2s_port_t i2s_num*, **const** void **src*, size_t *size*, size_t *src_bits*, size_t *aim_bits*, size_t **bytes_written*, TickType_t *ticks_to_wait*)

Write data to I2S DMA transmit buffer while expanding the number of bits per sample. For example, expanding 16-bit PCM to 32-bit PCM.

Format of the data in source buffer is determined by the I2S configuration (see *i2s_config_t*).

Parameters

- `i2s_num`: I2S_NUM_0, I2S_NUM_1
- `src`: Source address to write from
- `size`: Size of data in bytes
- `src_bits`: Source audio bit
- `aim_bits`: Bit wanted, no more than 32, and must be greater than src_bits
- `bytes_written`: Number of bytes written, if timeout, the result will be less than the size passed in.
- `ticks_to_wait`: TX buffer wait timeout in RTOS ticks. If this many ticks pass without space becoming available in the DMA transmit buffer, then the function will return (note that if the data is written to the DMA buffer in pieces, the overall operation may still take longer than this timeout.) Pass portMAX_DELAY for no timeout.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **i2s_read**(*i2s_port_t i2s_num*, void **dest*, size_t *size*, size_t **bytes_read*, TickType_t *ticks_to_wait*)

Read data from I2S DMA receive buffer.

Note If the built-in ADC mode is enabled, we should call i2s_adc_start and i2s_adc_stop around the whole reading process, to prevent the data getting corrupted.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `i2s_num`: I2S_NUM_0, I2S_NUM_1
- `dest`: Destination address to read into
- `size`: Size of data in bytes

- `bytes_read`: Number of bytes read, if timeout, bytes read will be less than the size passed in.
- `ticks_to_wait`: RX buffer wait timeout in RTOS ticks. If this many ticks pass without bytes becoming available in the DMA receive buffer, then the function will return (note that if data is read from the DMA buffer in pieces, the overall operation may still take longer than this timeout.) Pass portMAX_DELAY for no timeout.

esp_err_t **i2s_set_sample_rates**(*i2s_port_t i2s_num*, uint32_t *rate*)

Set sample rate used for I2S RX and TX.

The bit clock rate is determined by the sample rate and *i2s_config_t* configuration parameters (number of channels, bits_per_sample).

```
bit_clock = rate * (number of channels) * bits_per_sample
```

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_ERR_NO_MEM Out of memory

Parameters

- `i2s_num`: I2S_NUM_0, I2S_NUM_1
- `rate`: I2S sample rate (ex: 8000, 44100···)

esp_err_t **i2s_stop**(*i2s_port_t i2s_num*)

Stop I2S driver.

There is no need to call i2s_stop() before calling i2s_driver_uninstall().

Disables I2S TX/RX, until i2s_start() is called.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `i2s_num`: I2S_NUM_0, I2S_NUM_1

esp_err_t **i2s_start**(*i2s_port_t i2s_num*)

Start I2S driver.

It is not necessary to call this function after i2s_driver_install() (it is started automatically), however it is necessary to call it after i2s_stop().

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `i2s_num`: I2S_NUM_0, I2S_NUM_1

esp_err_t **i2s_zero_dma_buffer**(*i2s_port_t i2s_num*)

Zero the contents of the TX DMA buffer.

Pushes zero-byte samples into the TX DMA buffer, until it is full.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `i2s_num`: I2S_NUM_0, I2S_NUM_1

esp_err_t **i2s_set_clk**(*i2s_port_t i2s_num*, uint32_t *rate*, *i2s_bits_per_sample_t bits*, *i2s_channel_t ch*)

Set clock & bit width used for I2S RX and TX.

Similar to i2s_set_sample_rates(), but also sets bit width.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_ERR_NO_MEM Out of memory

Parameters

- `i2s_num`: I2S_NUM_0, I2S_NUM_1
- `rate`: I2S sample rate (ex: 8000, 44100···)
- `bits`: I2S bit width (I2S_BITS_PER_SAMPLE_16BIT, I2S_BITS_PER_SAMPLE_24BIT, I2S_BITS_PER_SAMPLE_32BIT)
- `ch`: I2S channel, (I2S_CHANNEL_MONO, I2S_CHANNEL_STEREO)

float **i2s_get_clk**(*i2s_port_t i2s_num*)

get clock set on particular port number.

Return

- actual clock set by i2s driver

Parameters

- `i2s_num`: I2S_NUM_0, I2S_NUM_1

esp_err_t **i2s_set_adc_mode**(*adc_unit_t adc_unit*, *adc1_channel_t adc_channel*)

Set built-in ADC mode for I2S DMA, this function will initialize ADC pad, and set ADC parameters.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `adc_unit`: SAR ADC unit index
- `adc_channel`: ADC channel index

esp_err_t **i2s_adc_enable**(*i2s_port_t i2s_num*)

Start to use I2S built-in ADC mode.

Note This function would acquire the lock of ADC to prevent the data getting corrupted during the I2S peripheral is being used to do fully continuous ADC sampling.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_ERR_INVALID_STATE Driver state error

Parameters

- `i2s_num`: i2s port index

esp_err_t **i2s_adc_disable**(*i2s_port_t i2s_num*)

Stop to use I2S built-in ADC mode.

Note This function would release the lock of ADC so that other tasks can use ADC.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_ERR_INVALID_STATE Driver state error

Parameters

- `i2s_num`: i2s port index

Type Definitions

typedef *intr_handle_t* **i2s_isr_handle_t**

Header File

- soc/include/hal/i2s_types.h

Structures

struct **i2s_config_t**

I2S configuration parameters for i2s_param_config function.

Public Members

i2s_mode_t **mode**

I2S work mode

int **sample_rate**

I2S sample rate

i2s_bits_per_sample_t **bits_per_sample**

I2S bits per sample

i2s_channel_fmt_t **channel_format**

I2S channel format

i2s_comm_format_t **communication_format**

I2S communication format

int **intr_alloc_flags**

Flags used to allocate the interrupt. One or multiple (ORred) ESP_INTR_FLAG_* values. See esp_intr_alloc.h for more info

int **dma_buf_count**

I2S DMA Buffer Count

int **dma_buf_len**

I2S DMA Buffer Length

bool **use_apll**

I2S using APLL as main I2S clock, enable it to get accurate clock

bool **tx_desc_auto_clear**

I2S auto clear tx descriptor if there is underflow condition (helps in avoiding noise in case of data unavailability)

int **fixed_mclk**

I2S using fixed MCLK output. If use_apll = true and fixed_mclk > 0, then the clock output for i2s is fixed and equal to the fixed_mclk value.

struct i2s_event_t

Event structure used in I2S event queue.

Public Members

i2s_event_type_t **type**

I2S event type

size_t **size**

I2S data size for I2S_DATA event

struct i2s_pin_config_t

I2S pin number for i2s_set_pin.

Public Members

int **bck_io_num**

BCK in out pin

int **ws_io_num**

WS in out pin

int **data_out_num**

DATA out pin

int **data_in_num**

DATA in pin

Macros

I2S_PIN_NO_CHANGE

Use in *i2s_pin_config_t* for pins which should not be changed

Type Definitions

typedef int **i2s_port_t**

I2S port number, the max port number is (I2S_NUM_MAX -1).

Enumerations

enum i2s_bits_per_sample_t

I2S bit width per sample.

Values:

I2S_BITS_PER_SAMPLE_8BIT = 8

I2S bits per sample: 8-bits

I2S_BITS_PER_SAMPLE_16BIT = 16

I2S bits per sample: 16-bits

I2S_BITS_PER_SAMPLE_24BIT = 24

I2S bits per sample: 24-bits

I2S_BITS_PER_SAMPLE_32BIT = 32

I2S bits per sample: 32-bits

enum i2s_channel_t

I2S channel.

Values:

I2S_CHANNEL_MONO = 1

I2S 1 channel (mono)

I2S_CHANNEL_STEREO = 2

I2S 2 channel (stereo)

enum i2s_comm_format_t

I2S communication standard format.

Values:

I2S_COMM_FORMAT_I2S = 0x01

I2S communication format I2S

I2S_COMM_FORMAT_I2S_MSB = 0x02

I2S format MSB

I2S_COMM_FORMAT_I2S_LSB = 0x04

I2S format LSB

I2S_COMM_FORMAT_PCM = 0x08

I2S communication format PCM

I2S_COMM_FORMAT_PCM_SHORT = 0x10

PCM Short

I2S_COMM_FORMAT_PCM_LONG = 0x20

PCM Long

enum i2s_channel_fmt_t

I2S channel format type.

Values:

I2S_CHANNEL_FMT_RIGHT_LEFT = 0x00

I2S_CHANNEL_FMT_ALL_RIGHT

I2S_CHANNEL_FMT_ALL_LEFT

I2S_CHANNEL_FMT_ONLY_RIGHT

I2S_CHANNEL_FMT_ONLY_LEFT

enum pdm_sample_rate_ratio_t

PDM sample rate ratio, measured in Hz.

Values:

PDM_SAMPLE_RATE_RATIO_64

PDM_SAMPLE_RATE_RATIO_128

enum pdm_pcm_conv_t

PDM PCM convter enable/disable.

Values:

PDM_PCM_CONV_ENABLE

PDM_PCM_CONV_DISABLE

enum i2s_mode_t

I2S Mode, defaut is I2S_MODE_MASTER | I2S_MODE_TX.

Note PDM and built-in DAC functions are only supported on I2S0 for current ESP32 chip.

Values:

I2S_MODE_MASTER = 1

I2S_MODE_SLAVE = 2

I2S_MODE_TX = 4

I2S_MODE_RX = 8

I2S_MODE_DAC_BUILT_IN = 16

Output I2S data to built-in DAC, no matter the data format is 16bit or 32 bit, the DAC module will only take the 8bits from MSB

I2S_MODE_ADC_BUILT_IN = 32

Input I2S data from built-in ADC, each data can be 12-bit width at most

enum i2s_event_type_t

I2S event types.

Values:

I2S_EVENT_DMA_ERROR

I2S_EVENT_TX_DONE

I2S DMA finish sent 1 buffer

I2S_EVENT_RX_DONE

I2S DMA finish received 1 buffer

I2S_EVENT_MAX

I2S event max index

enum i2s_dac_mode_t

I2S DAC mode for i2s_set_dac_mode.

Note PDM and built-in DAC functions are only supported on I2S0 for current ESP32 chip.

Values:

I2S_DAC_CHANNEL_DISABLE = 0

Disable I2S built-in DAC signals

I2S_DAC_CHANNEL_RIGHT_EN = 1

Enable I2S built-in DAC right channel, maps to DAC channel 1 on GPIO25

I2S_DAC_CHANNEL_LEFT_EN = 2

Enable I2S built-in DAC left channel, maps to DAC channel 2 on GPIO26

I2S_DAC_CHANNEL_BOTH_EN = 0x3

Enable both of the I2S built-in DAC channels.

I2S_DAC_CHANNEL_MAX = 0x4

I2S built-in DAC mode max index

enum i2s_clock_src_t

Values:

I2S_CLK_D2CLK = 0

I2S_CLK_APLL

2.3.7 LED Control

Introduction

The LED control (LEDC) peripheral is primarily designed to control the intensity of LEDs, although it can also be used to generate PWM signals for other purposes as well. It has 16 channels which can generate

independent waveforms that can be used, for example, to drive RGB LED devices.

A half of LEDC's channels operate in high speed mode. This mode is implemented in hardware and offers automatic and glitch-free changing of the PWM duty cycle. The other half of channels operate in low speed mode, where the moment of change depends on the application software. Each group of channels is also able to use different clock sources, but this feature is not yet supported in the LEDC driver.

The PWM controller can automatically increase or decrease the duty cycle gradually, allowing for fades without any processor interference.

Functionality Overview

Getting LEDC to work on a specific channel in either *high or low speed mode* is done in three steps:

1. *Configure Timer* by specifying the PWM signal's frequency and duty cycle resolution.
2. *Configure Channel* by associating it with the timer and GPIO to output the PWM signal.
3. *Change PWM Signal* that drives the output in order to change LED's intensity. This can be done under the full control of software or with hardware fading functions.

As an optional step, it is also possible to set up an interrupt on the fade end.

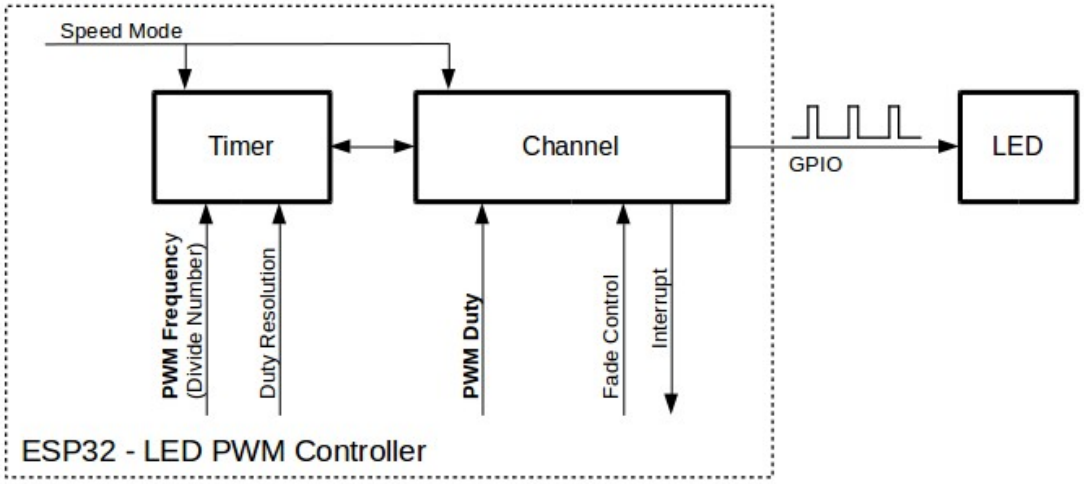


图 14: Key Settings of LED PWM Controller's API

Configure Timer

Setting the timer is done by calling the function *ledc_timer_config()* and passing to it a data structure *ledc_timer_config_t* that contains the following configuration settings:

- Timer number *ledc_timer_t*
- Speed mode *ledc_mode_t*
- PWM signal frequency
- Resolution of PWM duty

The frequency and the duty resolution are interdependent. The higher the PWM frequency, the lower duty resolution is available, and vice versa. This relationship might be important if you are planning to use this API for purposes other than changing the intensity of LEDs. For more details, see Section *Supported Range of Frequency and Duty Resolutions*.

Configure Channel

When the timer is set up, configure a selected channel (one out of *ledc_channel_t*). This is done by calling the function *ledc_channel_config()*.

Similar to the timer configuration, the channel setup function should be passed a structure *ledc_channel_config_t* that contains the channel' s configuration parameters.

At this point, the channel should start operating and generating the PWM signal on the selected GPIO, as configured in *ledc_channel_config_t*, with the frequency specified in the timer settings and the given duty cycle. The channel operation (signal generation) can be suspended at any time by calling the function *ledc_stop()*.

Change PWM Signal

Once the channel starts operating and generating the PWM signal with the constant duty cycle and frequency, there are a couple of ways to change this signal. When driving LEDs, primarily the duty cycle is changed to vary the light intensity.

The following two sections describe how to change the duty cycle using software and hardware fading. If required, the signal' s frequency can also be changed; it is covered in Section *Change PWM Frequency*.

Change PWM Duty Cycle Using Software

To set the duty cycle, use the dedicated function *ledc_set_duty()*. After that, call *ledc_update_duty()* to activeate the changes. To check the currently set value, use the corresponding `_get_` function *ledc_get_duty()*.

Another way to set the duty cycle, as well as some other channel parameters, is by calling *ledc_channel_config()* covered in Section *Configure Channel*.

The range of the duty cycle values passed to functions depends on selected `duty_resolution` and should be from `0` to `(2 ** duty_resolution) - 1`. For example, if the selected duty resolution is 10, then the duty

cycle values can range from 0 to 1023. This provides the resolution of ~0.1%.

Change PWM Duty Cycle using Hardware

The LEDC hardware provides the means to gradually transition from one duty cycle value to another. To use this functionality, enable fading with *ledc_fade_func_install()* and then configure it by calling one of the available fading functions:

- *ledc_set_fade_with_time()*
- *ledc_set_fade_with_step()*
- *ledc_set_fade()*

Finally start fading with *ledc_fade_start()*.

If not required anymore, fading and an associated interrupt can be disabled with *ledc_fade_func_uninstall()*.

Change PWM Frequency

The LEDC API provides several ways to change the PWM frequency "on the fly":

- Set the frequency by calling *ledc_set_freq()*. There is a corresponding function *ledc_get_freq()* to check the current frequency.
- Change the frequency and the duty resolution by calling *ledc_bind_channel_timer()* to bind some other timer to the channel.
- Change the channel's timer by calling *ledc_channel_config()*.

More Control Over PWM

There are several lower level timer-specific functions that can be used to change PWM settings:

- *ledc_timer_set()*
- *ledc_timer_rst()*
- *ledc_timer_pause()*
- *ledc_timer_resume()*

The first two functions are called "behind the scenes" by *ledc_channel_config()* to provide a "clean" startup of a timer after it is configured.

Use Interrupts

When configuring an LEDC channel, one of the parameters selected within *ledc_channel_config_t* is *ledc_intr_type_t* which triggers an interrupt on fade completion.

For registration of a handler to address this interrupt, call *ledc_isr_register()*.

LEDC High and Low Speed Mode

Of the total 8 timers and 16 channels available in the LED PWM Controller, half of them are dedicated to operation in high speed mode and the other half in low speed mode. Selection of a low or high speed timer or channel is done with the parameter *ledc_mode_t* that can be found in applicable function calls.

The advantage of high speed mode is hardware-supported, glitch-free changeover of the timer settings. This means that if the timer settings are modified, the changes will be applied automatically on the next overflow interrupt of the timer. In contrast, when updating the low-speed timer, the change of settings should be explicitly triggered by software. The LEDC driver handles it in the background, e.g., when *ledc_timer_config()* or *ledc_timer_set()* is called.

For additional details regarding speed modes, refer to ESP32 Technical Reference Manual (PDF). Please note that the support for `SLOW_CLOCK` mentioned in this manual is not yet supported in the LEDC driver.

Supported Range of Frequency and Duty Resolutions

The LED PWM Controller is designed primarily to drive LEDs. It provides a wide resolution for PWM duty cycle settings. For instance, the PWM frequency of 5 kHz can have the maximum duty resolution of 13 bits. It means that the duty can be set anywhere from 0 to 100% with a resolution of ~0.012% (2 ** 13 = 8192 discrete levels of the LED intensity).

The LEDC can be used for generating signals at much higher frequencies that are sufficient enough to clock other devices, e.g., a digital camera module. In this case, the maximum available frequency is 40 MHz with duty resolution of 1 bit. This means that the duty cycle is fixed at 50% and cannot be adjusted.

The LEDC API is designed to report an error when trying to set a frequency and a duty resolution that exceed the range of LEDC' s hardware. For example, an attempt to set the frequency to 20 MHz and the duty resolution to 3 bits will result in the following error reported on a serial monitor:

```
E (196) ledc: requested frequency and duty resolution cannot be achieved, try reducing␣
↪freq_hz or duty_resolution. div_param=128
```

In such a situation, either the duty resolution or the frequency must be reduced. For example, setting the duty resolution to 2 will resolve this issue and will make it possible to set the duty cycle at 25% steps, i.e., at 25%, 50% or 75%.

The LEDC driver will also capture and report attempts to configure frequency / duty resolution combinations that are below the supported minimum, e.g.:

```
E (196) ledc: requested frequency and duty resolution cannot be achieved, try increasing␣
↪freq_hz or duty_resolution. div_param=128000000
```

The duty resolution is normally set using *ledc_timer_bit_t*. This enumeration covers the range from 10 to 15 bits. If a smaller duty resolution is required (from 10 down to 1), enter the equivalent numeric values directly.

Application Example

The LEDC change duty cycle and fading control example: peripherals/ledc.

API Reference

Header File

- driver/include/driver/ledc.h

Functions

esp_err_t **ledc_channel_config**(**const** *ledc_channel_config_t* **ledc_conf*)

LEDC channel configuration Configure LEDC channel with the given channel/output gpio_num/interrupt/source timer/frequency(Hz)/LEDC duty resolution.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `ledc_conf`: Pointer of LEDC channel configure struct

esp_err_t **ledc_timer_config**(**const** *ledc_timer_config_t* **timer_conf*)

LEDC timer configuration Configure LEDC timer with the given source timer/frequency(Hz)/duty_resolution.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_FAIL Can not find a proper pre-divider number base on the given frequency and the current duty_resolution.

Parameters

- `timer_conf`: Pointer of LEDC timer configure struct

esp_err_t **ledc_update_duty**(*ledc_mode_t speed_mode*, *ledc_channel_t channel*)

LEDC update channel parameters.

Note Call this function to activate the LEDC updated parameters. After ledc_set_duty, we need to call this function to update the settings.

Note ledc_set_duty, ledc_set_duty_with_hpoint and ledc_update_duty are not thread-safe, do not call these functions to control one LEDC channel in different tasks at the same time. A thread-safe version of API is ledc_set_duty_and_update

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `speed_mode`: Select the LEDC speed_mode, high-speed mode and low-speed mode,
- `channel`: LEDC channel (0-7), select from ledc_channel_t

esp_err_t **ledc_set_pin**(int *gpio_num*, *ledc_mode_t speed_mode*, *ledc_channel_t ledc_channel*)

Set LEDC output gpio.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `gpio_num`: The LEDC output gpio
- `speed_mode`: Select the LEDC speed_mode, high-speed mode and low-speed mode
- `ledc_channel`: LEDC channel (0-7), select from ledc_channel_t

esp_err_t **ledc_stop**(*ledc_mode_t speed_mode*, *ledc_channel_t channel*, uint32_t *idle_level*)

LEDC stop. Disable LEDC output, and set idle level.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `speed_mode`: Select the LEDC speed_mode, high-speed mode and low-speed mode
- `channel`: LEDC channel (0-7), select from ledc_channel_t
- `idle_level`: Set output idle level after LEDC stops.

esp_err_t **ledc_set_freq**(*ledc_mode_t speed_mode*, *ledc_timer_t timer_num*, uint32_t *freq_hz*)

LEDC set channel frequency (Hz)

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_FAIL Can not find a proper pre-divider number base on the given frequency and the current duty_resolution.

Parameters

- `speed_mode`: Select the LEDC speed_mode, high-speed mode and low-speed mode
- `timer_num`: LEDC timer index (0-3), select from ledc_timer_t
- `freq_hz`: Set the LEDC frequency

uint32_t **ledc_get_freq**(*ledc_mode_t speed_mode*, *ledc_timer_t timer_num*)

LEDC get channel frequency (Hz)

Return

- 0 error
- Others Current LEDC frequency

Parameters

- `speed_mode`: Select the LEDC speed_mode, high-speed mode and low-speed mode
- `timer_num`: LEDC timer index (0-3), select from ledc_timer_t

esp_err_t **ledc_set_duty_with_hpoint**(*ledc_mode_t speed_mode*, *ledc_channel_t channel*, uint32_t *duty*, uint32_t *hpoint*)

LEDC set duty and hpoint value Only after calling ledc_update_duty will the duty update.

Note ledc_set_duty, ledc_set_duty_with_hpoint and ledc_update_duty are not thread-safe, do not call these functions to control one LEDC channel in different tasks at the same time. A thread-safe version of API is ledc_set_duty_and_update

Note If a fade operation is running in progress on that channel, the driver would not allow it to be stopped. Other duty operations will have to wait until the fade operation has finished.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `speed_mode`: Select the LEDC speed_mode, high-speed mode and low-speed mode
- `channel`: LEDC channel (0-7), select from ledc_channel_t
- `duty`: Set the LEDC duty, the range of duty setting is [0, (2**duty_resolution)]
- `hpoint`: Set the LEDC hpoint value(max: 0xfffff)

int **ledc_get_hpoint**(*ledc_mode_t speed_mode*, *ledc_channel_t channel*)

LEDC get hpoint value, the counter value when the output is set high level.

Return

- LEDC_ERR_VAL if parameter error
- Others Current hpoint value of LEDC channel

Parameters

- `speed_mode`: Select the LEDC speed_mode, high-speed mode and low-speed mode
- `channel`: LEDC channel (0-7), select from ledc_channel_t

esp_err_t **ledc_set_duty**(*ledc_mode_t speed_mode*, *ledc_channel_t channel*, uint32_t *duty*)

LEDC set duty This function do not change the hpoint value of this channel. if needed, please call ledc_set_duty_with_hpoint. only after calling ledc_update_duty will the duty update.

Note ledc_set_duty, ledc_set_duty_with_hpoint and ledc_update_duty are not thread-safe, do not call these functions to control one LEDC channel in different tasks at the same time. A thread-safe version of API is ledc_set_duty_and_update.

Note If a fade operation is running in progress on that channel, the driver would not allow it to be stopped. Other duty operations will have to wait until the fade operation has finished.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `speed_mode`: Select the LEDC speed_mode, high-speed mode and low-speed mode
- `channel`: LEDC channel (0-7), select from ledc_channel_t
- `duty`: Set the LEDC duty, the range of duty setting is [0, (2**duty_resolution)]

uint32_t **ledc_get_duty**(*ledc_mode_t speed_mode*, *ledc_channel_t channel*)

LEDC get duty.

Return

- LEDC_ERR_DUTY if parameter error
- Others Current LEDC duty

Parameters

- `speed_mode`: Select the LEDC speed_mode, high-speed mode and low-speed mode
- `channel`: LEDC channel (0-7), select from ledc_channel_t

esp_err_t **ledc_set_fade**(*ledc_mode_t speed_mode*, *ledc_channel_t channel*, uint32_t *duty*, *ledc_duty_direction_t fade_direction*, uint32_t *step_num*, uint32_t *duty_cycle_num*, uint32_t *duty_scale*)

LEDC set gradient Set LEDC gradient, After the function calls the ledc_update_duty function, the function can take effect.

Note If a fade operation is running in progress on that channel, the driver would not allow it to be stopped. Other duty operations will have to wait until the fade operation has finished.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `speed_mode`: Select the LEDC speed_mode, high-speed mode and low-speed mode
- `channel`: LEDC channel (0-7), select from ledc_channel_t
- `duty`: Set the start of the gradient duty, the range of duty setting is [0, (2**duty_resolution)]
- `fade_direction`: Set the direction of the gradient
- `step_num`: Set the number of the gradient
- `duty_cycle_num`: Set how many LEDC tick each time the gradient lasts
- `duty_scale`: Set gradient change amplitude

esp_err_t **ledc_isr_register**(void (**fn*)(void *, void **arg*, int *intr_alloc_flags*, *ledc_isr_handle_t* **handle*)Register LEDC interrupt handler, the handler is an ISR. The handler will be attached to the same CPU core that this function is running on.

Return

- ESP_OK Success

- ESP_ERR_INVALID_ARG Function pointer error.

Parameters

- `fn`: Interrupt handler function.
- `arg`: User-supplied argument passed to the handler function.
- `intr_alloc_flags`: Flags used to allocate the interrupt. One or multiple (ORred) ESP_INTR_FLAG_* values. See esp_intr_alloc.h for more info.
- `handle`: Pointer to return handle. If non-NULL, a handle for the interrupt will be returned here.

esp_err_t **ledc_timer_set**(*ledc_mode_t speed_mode*, *ledc_timer_t timer_sel*, uint32_t *clock_divider*, uint32_t *duty_resolution*, *ledc_clk_src_t clk_src*)

Configure LEDC settings.

Return

- (-1) Parameter error
- Other Current LEDC duty

Parameters

- `speed_mode`: Select the LEDC speed_mode, high-speed mode and low-speed mode
- `timer_sel`: Timer index (0-3), there are 4 timers in LEDC module
- `clock_divider`: Timer clock divide value, the timer clock is divided from the selected clock source
- `duty_resolution`: Resolution of duty setting in number of bits. The range of duty values is [0, (2**duty_resolution)]
- `clk_src`: Select LEDC source clock.

esp_err_t **ledc_timer_rst**(*ledc_mode_t speed_mode*, *ledc_timer_t timer_sel*)

Reset LEDC timer.

Return

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success

Parameters

- `speed_mode`: Select the LEDC speed_mode, high-speed mode and low-speed mode
- `timer_sel`: LEDC timer index (0-3), select from ledc_timer_t

esp_err_t **ledc_timer_pause**(*ledc_mode_t speed_mode*, *ledc_timer_t timer_sel*)

Pause LEDC timer counter.

Return

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success

Parameters

- **speed_mode**: Select the LEDC speed_mode, high-speed mode and low-speed mode
- **timer_sel**: LEDC timer index (0-3), select from ledc_timer_t

esp_err_t **ledc_timer_resume**(*ledc_mode_t speed_mode*, *ledc_timer_t timer_sel*)

Resume LEDC timer.

Return

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success

Parameters

- **speed_mode**: Select the LEDC speed_mode, high-speed mode and low-speed mode
- **timer_sel**: LEDC timer index (0-3), select from ledc_timer_t

esp_err_t **ledc_bind_channel_timer**(*ledc_mode_t speed_mode*, *ledc_channel_t channel*, *ledc_timer_t timer_sel*)

Bind LEDC channel with the selected timer.

Return

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success

Parameters

- **speed_mode**: Select the LEDC speed_mode, high-speed mode and low-speed mode
- **channel**: LEDC channel index (0-7), select from ledc_channel_t
- **timer_sel**: LEDC timer index (0-3), select from ledc_timer_t

esp_err_t **ledc_set_fade_with_step**(*ledc_mode_t speed_mode*, *ledc_channel_t channel*, uint32_t *target_duty*, uint32_t *scale*, uint32_t *cycle_num*)

Set LEDC fade function.

Note Call ledc_fade_func_install() once before calling this function. Call ledc_fade_start() after this to start fading.

Note ledc_set_fade_with_step, ledc_set_fade_with_time and ledc_fade_start are not thread-safe, do not call these functions to control one LEDC channel in different tasks at the same time. A thread-safe version of API is ledc_set_fade_step_and_start

Note If a fade operation is running in progress on that channel, the driver would not allow it to be stopped. Other duty operations will have to wait until the fade operation has finished.

Return

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success
- ESP_ERR_INVALID_STATE Fade function not installed.
- ESP_FAIL Fade function init error

Parameters

- `speed_mode`: Select the LEDC speed_mode, high-speed mode and low-speed mode,
- `channel`: LEDC channel index (0-7), select from ledc_channel_t
- `target_duty`: Target duty of fading [0, (2**duty_resolution) - 1]
- `scale`: Controls the increase or decrease step scale.
- `cycle_num`: increase or decrease the duty every cycle_num cycles

esp_err_t **ledc_set_fade_with_time**(*ledc_mode_t speed_mode*, *ledc_channel_t channel*, uint32_t *target_duty*, int *max_fade_time_ms*)

Set LEDC fade function, with a limited time.

Note Call ledc_fade_func_install() once before calling this function. Call ledc_fade_start() after this to start fading.

Note ledc_set_fade_with_step, ledc_set_fade_with_time and ledc_fade_start are not thread-safe, do not call these functions to control one LEDC channel in different tasks at the same time. A thread-safe version of API is ledc_set_fade_step_and_start

Note If a fade operation is running in progress on that channel, the driver would not allow it to be stopped. Other duty operations will have to wait until the fade operation has finished.

Return

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success
- ESP_ERR_INVALID_STATE Fade function not installed.
- ESP_FAIL Fade function init error

Parameters

- `speed_mode`: Select the LEDC speed_mode, high-speed mode and low-speed mode,
- `channel`: LEDC channel index (0-7), select from ledc_channel_t
- `target_duty`: Target duty of fading.(0 - (2 ** duty_resolution - 1)))
- `max_fade_time_ms`: The maximum time of the fading (ms).

esp_err_t **ledc_fade_func_install**(int *intr_alloc_flags*)

Install LEDC fade function. This function will occupy interrupt of LEDC module.

Return

- ESP_OK Success
- ESP_ERR_INVALID_STATE Fade function already installed.

Parameters

- `intr_alloc_flags`: Flags used to allocate the interrupt. One or multiple (ORred) ESP_INTR_FLAG_* values. See esp_intr_alloc.h for more info.

void **ledc_fade_func_uninstall**(void)

Uninstall LEDC fade function.

esp_err_t **ledc_fade_start**(*ledc_mode_t speed_mode*, *ledc_channel_t channel*, *ledc_fade_mode_t fade_mode*)

Start LEDC fading.

Note Call ledc_fade_func_install() once before calling this function. Call this API right after ledc_set_fade_with_time or ledc_set_fade_with_step before to start fading.

Note If a fade operation is running in progress on that channel, the driver would not allow it to be stopped. Other duty operations will have to wait until the fade operation has finished.

Return

- ESP_OK Success
- ESP_ERR_INVALID_STATE Fade function not installed.
- ESP_ERR_INVALID_ARG Parameter error.

Parameters

- `speed_mode`: Select the LEDC speed_mode, high-speed mode and low-speed mode
- `channel`: LEDC channel number
- `fade_mode`: Whether to block until fading done.

esp_err_t **ledc_set_duty_and_update**(*ledc_mode_t speed_mode*, *ledc_channel_t channel*, uint32_t *duty*, uint32_t *hpoint*)

A thread-safe API to set duty for LEDC channel and return when duty updated.

Note If a fade operation is running in progress on that channel, the driver would not allow it to be stopped. Other duty operations will have to wait until the fade operation has finished.

Parameters

- `speed_mode`: Select the LEDC speed_mode, high-speed mode and low-speed mode
- `channel`: LEDC channel (0-7), select from ledc_channel_t
- `duty`: Set the LEDC duty, the range of duty setting is [0, (2**duty_resolution)]
- `hpoint`: Set the LEDC hpoint value(max: 0xfffff)

esp_err_t **ledc_set_fade_time_and_start**(*ledc_mode_t speed_mode*, *ledc_channel_t channel*, uint32_t *target_duty*, uint32_t *max_fade_time_ms*, *ledc_fade_mode_t fade_mode*)

A thread-safe API to set and start LEDC fade function, with a limited time.

Note Call ledc_fade_func_install() once, before calling this function.

Note If a fade operation is running in progress on that channel, the driver would not allow it to be stopped. Other duty operations will have to wait until the fade operation has finished.

Return

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success
- ESP_ERR_INVALID_STATE Fade function not installed.
- ESP_FAIL Fade function init error

Parameters

- `speed_mode`: Select the LEDC speed_mode, high-speed mode and low-speed mode,
- `channel`: LEDC channel index (0-7), select from ledc_channel_t
- `target_duty`: Target duty of fading.(0 - (2 ** duty_resolution - 1)))
- `max_fade_time_ms`: The maximum time of the fading (ms).
- `fade_mode`: choose blocking or non-blocking mode

esp_err_t **ledc_set_fade_step_and_start**(*ledc_mode_t speed_mode*, *ledc_channel_t channel*, uint32_t *target_duty*, uint32_t *scale*, uint32_t *cycle_num*, *ledc_fade_mode_t fade_mode*)

A thread-safe API to set and start LEDC fade function.

Note Call ledc_fade_func_install() once before calling this function.

Note If a fade operation is running in progress on that channel, the driver would not allow it to be stopped. Other duty operations will have to wait until the fade operation has finished.

Return

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success
- ESP_ERR_INVALID_STATE Fade function not installed.
- ESP_FAIL Fade function init error

Parameters

- `speed_mode`: Select the LEDC speed_mode, high-speed mode and low-speed mode,
- `channel`: LEDC channel index (0-7), select from ledc_channel_t
- `target_duty`: Target duty of fading [0, (2**duty_resolution) - 1]
- `scale`: Controls the increase or decrease step scale.
- `cycle_num`: increase or decrease the duty every cycle_num cycles
- `fade_mode`: choose blocking or non-blocking mode

Macros

`LEDC_APB_CLK_HZ`

`LEDC_REF_CLK_HZ`

`LEDC_ERR_DUTY`

`LEDC_ERR_VAL`

Type Definitions

`typedef` *intr_handle_t* `ledc_isr_handle_t`

Header File

- soc/include/hal/ledc_types.h

Structures

`struct ledc_channel_config_t`

Configuration parameters of LEDC channel for ledc_channel_config function.

Public Members

int **gpio_num**

the LEDC output gpio_num, if you want to use gpio16, gpio_num = 16

ledc_mode_t **speed_mode**

LEDC speed speed_mode, high-speed mode or low-speed mode

ledc_channel_t **channel**

LEDC channel (0 - 7)

ledc_intr_type_t **intr_type**

configure interrupt, Fade interrupt enable or Fade interrupt disable

ledc_timer_t **timer_sel**

Select the timer source of channel (0 - 3)

uint32_t **duty**

LEDC channel duty, the range of duty setting is [0, (2**duty_resolution)]

int **hpoint**

LEDC channel hpoint value, the max value is 0xfffff

struct ledc_timer_config_t

Configuration parameters of LEDC Timer timer for ledc_timer_config function.

Public Members

ledc_mode_t **speed_mode**

LEDC speed speed_mode, high-speed mode or low-speed mode

ledc_timer_bit_t **duty_resolution**

LEDC channel duty resolution

ledc_timer_bit_t **bit_num**

Deprecated in ESP-IDF 3.0. This is an alias to 'duty_resolution' for backward compatibility with ESP-IDF 2.1

ledc_timer_t **timer_num**

The timer source of channel (0 - 3)

uint32_t **freq_hz**

LEDC timer frequency (Hz)

ledc_clk_cfg_t **clk_cfg**

Configure LEDC source clock. For low speed channels and high speed channels, you can specify the source clock using LEDC_USE_REF_TICK, LEDC_USE_APB_CLK or LEDC_AUTO_CLK. For low speed channels, you can also specify the source clock using

LEDC_USE_RTC8M_CLK, in this case, all low speed channel' s source clock must be RTC8M_CLK

Enumerations

enum ledc_mode_t

Values:

LEDC_LOW_SPEED_MODE

LEDC low speed speed_mode

LEDC_SPEED_MODE_MAX

LEDC speed limit

enum ledc_intr_type_t

Values:

LEDC_INTR_DISABLE = 0

Disable LEDC interrupt

LEDC_INTR_FADE_END

Enable LEDC interrupt

LEDC_INTR_MAX

enum ledc_duty_direction_t

Values:

LEDC_DUTY_DIR_DECREASE = 0

LEDC duty decrease direction

LEDC_DUTY_DIR_INCREASE = 1

LEDC duty increase direction

LEDC_DUTY_DIR_MAX

enum ledc_slow_clk_sel_t

Values:

LEDC_SLOW_CLK_RTC8M = 0

LEDC low speed timer clock source is 8MHz RTC clock

LEDC_SLOW_CLK_APB

LEDC low speed timer clock source is 80MHz APB clock

enum ledc_clk_cfg_t

Values:

LEDC_AUTO_CLK = 0

The driver will automatically select the source clock(REF_TICK or APB) based on the giving resolution and duty parameter when init the timer

`LEDC_USE_REF_TICK`
LEDC timer select REF_TICK clock as source clock

`LEDC_USE_APB_CLK`
LEDC timer select APB clock as source clock

`LEDC_USE_RTC8M_CLK`
LEDC timer select RTC8M_CLK as source clock. Only for low speed channels and this parameter must be the same for all low speed channels

`enum ledc_clk_src_t`
Values:

`LEDC_REF_TICK` = LEDC_USE_REF_TICK
LEDC timer clock divided from reference tick (1Mhz)

`LEDC_APB_CLK` = LEDC_USE_APB_CLK
LEDC timer clock divided from APB clock (80Mhz)

`enum ledc_timer_t`
Values:

`LEDC_TIMER_0` = 0
LEDC timer 0

`LEDC_TIMER_1`
LEDC timer 1

`LEDC_TIMER_2`
LEDC timer 2

`LEDC_TIMER_3`
LEDC timer 3

`LEDC_TIMER_MAX`

`enum ledc_channel_t`
Values:

`LEDC_CHANNEL_0` = 0
LEDC channel 0

`LEDC_CHANNEL_1`
LEDC channel 1

`LEDC_CHANNEL_2`
LEDC channel 2

`LEDC_CHANNEL_3`
LEDC channel 3

LEDC_CHANNEL_4
: LEDC channel 4

LEDC_CHANNEL_5
: LEDC channel 5

LEDC_CHANNEL_6
: LEDC channel 6

LEDC_CHANNEL_7
: LEDC channel 7

LEDC_CHANNEL_MAX

enum ledc_timer_bit_t

Values:

LEDC_TIMER_1_BIT = 1
: LEDC PWM duty resolution of 1 bits

LEDC_TIMER_2_BIT
: LEDC PWM duty resolution of 2 bits

LEDC_TIMER_3_BIT
: LEDC PWM duty resolution of 3 bits

LEDC_TIMER_4_BIT
: LEDC PWM duty resolution of 4 bits

LEDC_TIMER_5_BIT
: LEDC PWM duty resolution of 5 bits

LEDC_TIMER_6_BIT
: LEDC PWM duty resolution of 6 bits

LEDC_TIMER_7_BIT
: LEDC PWM duty resolution of 7 bits

LEDC_TIMER_8_BIT
: LEDC PWM duty resolution of 8 bits

LEDC_TIMER_9_BIT
: LEDC PWM duty resolution of 9 bits

LEDC_TIMER_10_BIT
: LEDC PWM duty resolution of 10 bits

LEDC_TIMER_11_BIT
: LEDC PWM duty resolution of 11 bits

LEDC_TIMER_12_BIT
: LEDC PWM duty resolution of 12 bits

LEDC_TIMER_13_BIT
LEDC PWM duty resolution of 13 bits

LEDC_TIMER_14_BIT
LEDC PWM duty resolution of 14 bits

LEDC_TIMER_15_BIT
LEDC PWM duty resolution of 15 bits

LEDC_TIMER_16_BIT
LEDC PWM duty resolution of 16 bits

LEDC_TIMER_17_BIT
LEDC PWM duty resolution of 17 bits

LEDC_TIMER_18_BIT
LEDC PWM duty resolution of 18 bits

LEDC_TIMER_19_BIT
LEDC PWM duty resolution of 19 bits

LEDC_TIMER_20_BIT
LEDC PWM duty resolution of 20 bits

LEDC_TIMER_BIT_MAX

enum ledc_fade_mode_t
Values:

LEDC_FADE_NO_WAIT = 0
LEDC fade function will return immediately

LEDC_FADE_WAIT_DONE
LEDC fade function will block until fading to the target duty

LEDC_FADE_MAX

2.3.8 MCPWM

ESP32 has two MCPWM units which can be used to control different types of motors. Each unit has three pairs of PWM outputs.

Further in documentation the outputs of a single unit are labeled `PWMxA` / `PWMxB`.

More detailed block diagram of the MCPWM unit is shown below. Each A/B pair may be clocked by any one of the three timers Timer 0, 1 and 2. The same timer may be used to clock more than one pair of PWM outputs. Each unit is also able to collect inputs such as `SYNC SIGNALS`, detect `FAULT SIGNALS` like motor overcurrent or overvoltage, as well as obtain feedback with `CAPTURE SIGNALS` on e.g. a rotor position.

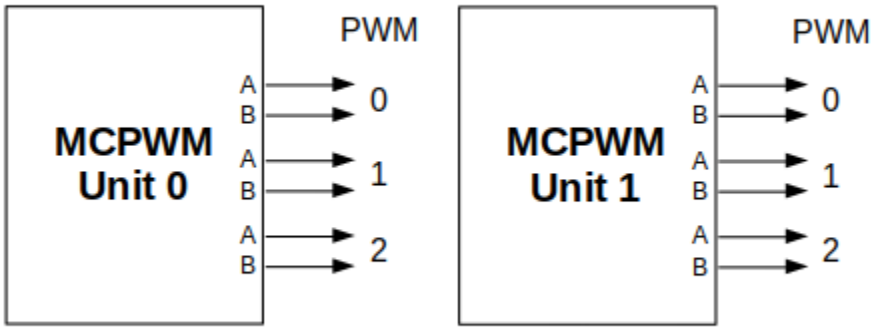


图 15: MCPWM Overview

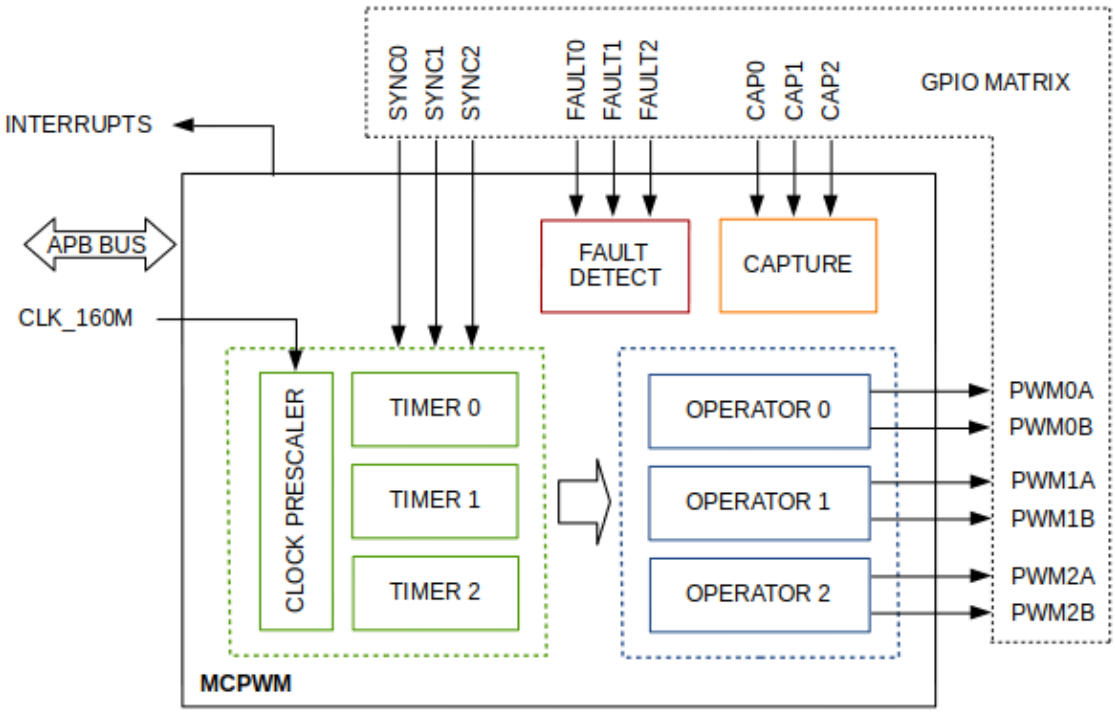


图 16: MCPWM Block Diagram

Description of this API starts with configuration of MCPWM' s **Timer** and **Generator** submodules to provide the basic motor control functionality. Then it discusses more advanced submodules and functionalities of a **Fault Handler**, signal **Capture**, **Carrier** and **Interrupts**.

Contents

- *Configure* a basic functionality of the outputs
- *Operate* the outputs to drive a motor
- *Adjust* how the motor is driven
- *Capture* external signals to provide additional control over the outputs
- Use *Fault Handler* to detect and manage faults
- Add a higher frequency *Carrier*, if output signals are passed through an isolation transformer
- Configuration and handling of *Interrupts*.

Configure

The scope of configuration depends on the motor type, in particular how many outputs and inputs are required, and what will be the sequence of signals to drive the motor.

In this case we will describe a simple configuration to control a brushed DC motor that is using only some of the available MCPWM' s resources. An example circuit is shown below. It includes a H-Bridge to switch polarization of a voltage applied to the motor (M) and to provide sufficient current to drive it.

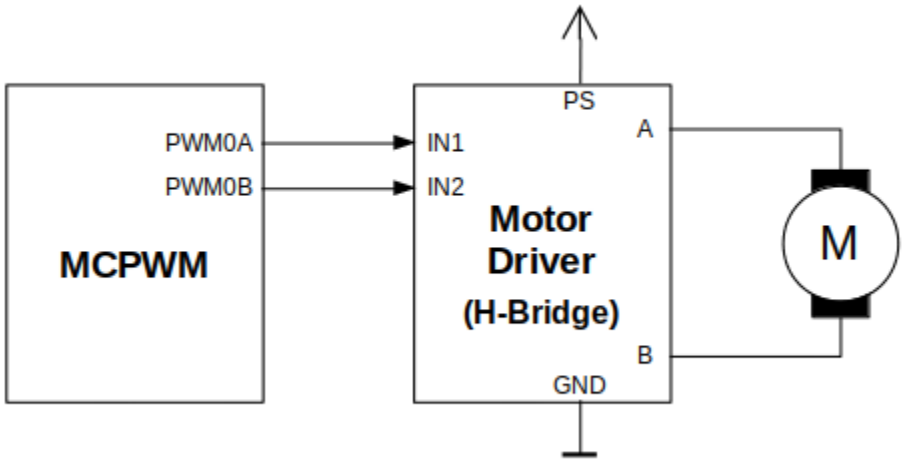


图 17: Example of Brushed DC Motor Control with MCPWM

Configuration covers the following steps:

1. Selection of a MPWn unit that will be used to drive the motor. There are two units available on-board of ESP32 and enumerated in *mcpwm_unit_t*.
2. Initialization of two GPIOs as output signals within selected unit by calling *mcpwm_gpio_init()*. The two output signals are typically used to command the motor to rotate right or left. All available signal options are listed in *mcpwm_io_signals_t*. To set more than a single pin at a time, use function *mcpwm_set_pin()* together with *mcpwm_pin_config_t*.
3. Selection of a timer. There are three timers available within the unit. The timers are listed in *mcpwm_timer_t*.
4. Setting of the timer frequency and initial duty within *mcpwm_config_t* structure.
5. Calling of *mcpwm_init()* with the above parameters to make the configuration effective.

Operate

To operate a motor connected to the MCPWM unit, e.g. turn it left or right, or vary the speed, we should apply some control signals to the unit' s outputs. The outputs are organized into three pairs. Within a pair they are labeled "A" and "B" and each driven by a submodule called an "Generator" . To provide a PWM signal, the Operator itself, which contains two Generator, should be clocked by one of three available Timers. To make the API simpler, each Timer is automatically associated by the API to drive an Operator of the same index, e.g. Timer 0 is associated with Operator 0.

There are the following basic ways to control the outputs:

- We can drive particular signal steady high or steady low with function *mcpwm_set_signal_high()* or *mcpwm_set_signal_low()*. This will make the motor to turn with a maximum speed or stop. Depending on selected output A or B the motor will rotate either right or left.
- Another option is to drive the outputs with the PWM signal by calling *mcpwm_start()* or *mcpwm_stop()*. The motor speed will be proportional to the PWM duty.
- To vary PWM' s duty call *mcpwm_set_duty()* and provide the duty value in %. Optionally, you may call *mcpwm_set_duty_in_us()*, if you prefer to set the duty in microseconds. Checking of currently set value is possible by calling *mcpwm_get_duty()*. Phase of the PWM signal may be altered by calling *mcpwm_set_duty_type()*. The duty is set individually for each A and B output using *mcpwm_generator_t* in specific function calls. The duty value refers either to high or low output signal duration. This is configured when calling *mcpwm_init()*, as discussed in section *Configure*, and selecting one of options from *mcpwm_duty_type_t*.

注解: Call function *mcpwm_set_duty_type()* every time after *mcpwm_set_signal_high()* or *mcpwm_set_signal_low()* to resume with previously set duty cycle.

Adjust

There are couple of ways to adjust a signal on the outputs and changing how the motor operates.

- Set specific PWM frequency by calling *mcpwm_set_frequency()*. This may be required to adjust to electrical or mechanical characteristics of particular motor and driver. To check what frequency is set, use function *mcpwm_get_frequency()*.
- Introduce a dead time between outputs A and B when they are changing the state to reverse direction of the motor rotation. This is to make up for on/off switching delay of the motor driver FETs. The dead time options are defined in *mcpwm_deadtime_type_t* and enabled by calling *mcpwm_deadtime_enable()*. To disable this functionality call *mcpwm_deadtime_disable()*.
- Synchronize outputs of operator submodules, e.g. to get raising edge of PWM0A/B and PWM1A/B to start exactly at the same time, or shift them between each other by a given phase. Synchronization is triggered by `SYNC SIGNALS` shown on the *block diagram* of the MCPWM above, and defined in *mcpwm_sync_signal_t*. To attach the signal to a GPIO call *mcpwm_gpio_init()*. You can then enable synchronization with function *mcpwm_sync_enable()*. As input parameters provide MCPWM unit, timer to synchronize, the synchronization signal and a phase to delay the timer.

注解: Synchronization signals are referred to using two different enumerations. First one *mcpwm_io_signals_t* is used together with function *mcpwm_gpio_init()* when selecting a GPIO as the signal input source. The second one *mcpwm_sync_signal_t* is used when enabling or disabling synchronization with *mcpwm_sync_enable()* or *mcpwm_sync_disable()*.

- Vary the pattern of the A/B output signals by getting MCPWM counters to count up, down and up/down (automatically changing the count direction). Respective configuration is done when calling *mcpwm_init()*, as discussed in section *Configure*, and selecting one of counter types from *mcpwm_counter_type_t*. For explanation of how A/B PWM output signals are generated please refer to ESP32 Technical Reference Manual.

Capture

One of requirements of BLDC (Brushless DC, see figure below) motor control is sensing of the rotor position. To facilitate this task each MCPWM unit provides three sensing inputs together with dedicated hardware. The hardware is able to detect the input signal' s edge and measure time between signals. As result the control software is simpler and the CPU power may be used for other tasks.

The capture functionality may be used for other types of motors or tasks. The functionality is enabled in two steps:

1. Configuration of GPIOs to act as the capture signal inputs by calling functions *mcpwm_gpio_init()* or *mcpwm_set_pin()*, that were described in section *Configure*.

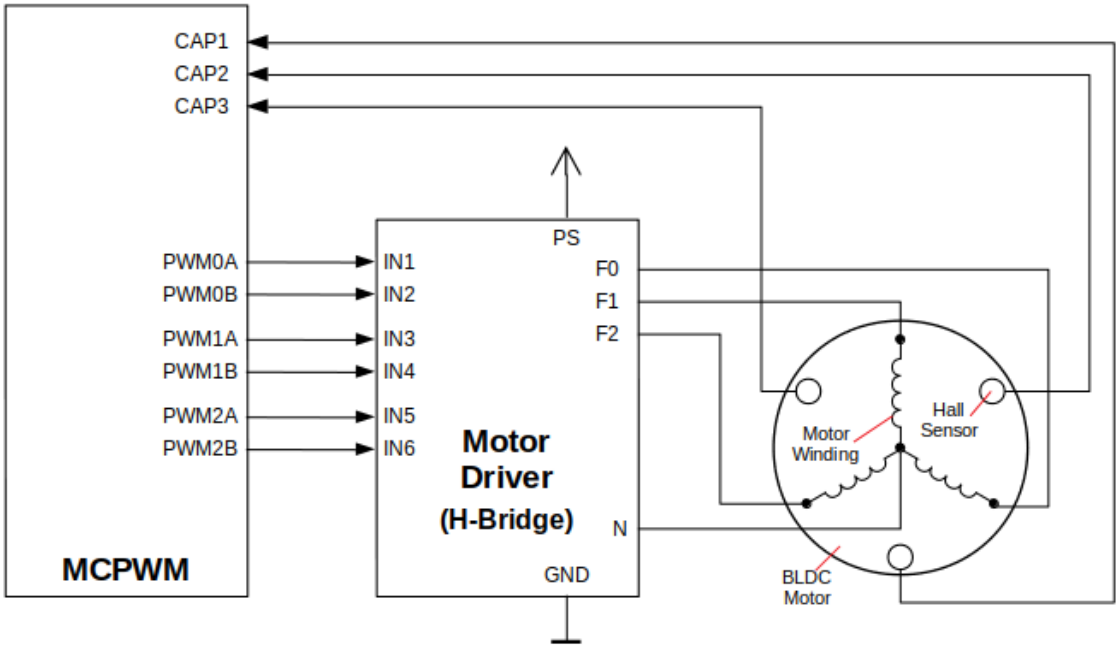


图 18: Example of Brushless DC Motor Control with MCPWM

2. Enabling of the functionality itself by invoking *mcpwm_capture_enable()*, selecting desired signal input from *mcpwm_capture_signal_t*, setting the signal edge with *mcpwm_capture_on_edge_t* and the signal count prescaler.

Within the second step above a 32-bit capture timer is enabled. The timer runs continuously driven by the APB clock. The clock frequency is typically 80 MHz. On each capture event the capture timer' s value is stored in time-stamp register that may be then checked by calling *mcpwm_capture_signal_get_value()*. The edge of the last signal may be checked with *mcpwm_capture_signal_get_edge()*.

If not required anymore, the capture functionality may be disabled with *mcpwm_capture_disable()*.

Fault Handler

Each unit of the MCPWM is able to sense external signals with information about failure of the motor, the motor driver or any other device connected to the MCPWM. There are three fault inputs per unit that may be routed to user selectable GPIOs. The MCPWM may be configured to perform one of four predefined actions on A/B outputs when a fault signal is received:

- lock current state of the output
- set the output low
- set the output high

- toggle the output

The user should determine possible failure modes of the motor and what action should be performed on detection of particular fault, e.g. drive all outputs low for a brushed motor, or lock current state for a stepper motor, etc. As result of this action the motor should be put into a safe state to reduce likelihood of a damage caused by the fault.

The fault handler functionality is enabled in two steps:

1. Configuration of GPIOs to act as fault signal inputs. This is done in analogous way as described for capture signals in section above. It includes setting the signal level to trigger the fault as defined in *mcpwm_fault_input_level_t*.
2. Initialization of the fault handler by calling either *mcpwm_fault_set_oneshot_mode()* or *mcpwm_fault_set_cyc_mode()*. These functions set the mode that MCPWM should operate once fault signal becomes inactive. There are two modes possible:
   - State of MCPWM unit will be locked until reset - *mcpwm_fault_set_oneshot_mode()*.
   - The MCPWM will resume operation once fault signal becoming inactive - *mcpwm_fault_set_cyc_mode()*.

   The function call parameters include selection of one of three fault inputs defined in *mcpwm_fault_signal_t* and specific action on outputs A and B defined in *mcpwm_action_on_pwmxa_t* and *mcpwm_action_on_pwmxb_t*.

Particular fault signal may be disabled at the runtime by calling *mcpwm_fault_deinit()*.

Carrier

The MCPWM has a carrier submodule used if galvanic isolation from the motor driver is required by passing the A/B output signals through transformers. Any of A and B output signals may be at 100% duty and not changing whenever motor is required to run steady at the full load. Coupling of non alternating signals with a transformer is problematic, so the signals are modulated by the carrier submodule to create an AC waveform, to make the coupling possible.

To use the carrier submodule, it should be first initialized by calling *mcpwm_carrier_init()*. The carrier parameters are defined in *mcpwm_carrier_config_t* structure invoked within the function call. Then the carrier functionality may be enabled by calling *mcpwm_carrier_enable()*.

The carrier parameters may be then alerted at a runtime by calling dedicated functions to change individual fields of the *mcpwm_carrier_config_t* structure, like *mcpwm_carrier_set_period()*, *mcpwm_carrier_set_duty_cycle()*, *mcpwm_carrier_output_invert()*, etc.

This includes enabling and setting duration of the first pulse of the career with *mcpwm_carrier_oneshot_mode_enable()*. For more details please refer to "PWM Carrier Submodule" section of the ESP32 Technical Reference Manual.

To disable carrier functionality call *mcpwm_carrier_disable()*.

Interrupts

Registering of the MCPWM interrupt handler is possible by calling *mcpwm_isr_register()*.

Application Example

Examples of using MCPWM for motor control: peripherals/mcpwm:

- Demonstration how to use each submodule of the MCPWM - peripherals/mcpwm/mcpwm_basic_config
- Control of BLDC (brushless DC) motor with hall sensor feedback - peripherals/mcpwm/mcpwm_bldc_control
- Brushed DC motor control - peripherals/mcpwm/mcpwm_brushed_dc_control
- Servo motor control - peripherals/mcpwm/mcpwm_servo_control

API Reference

Header File

- soc/include/hal/mcpwm_types.h

Enumerations

enum mcpwm_intr_t

Interrupts for MCPWM.

Values:

MCPWM_LL_INTR_CAP0 = BIT(27)

Capture 0 happened.

MCPWM_LL_INTR_CAP1 = BIT(28)

Capture 1 happened.

MCPWM_LL_INTR_CAP2 = BIT(29)

Capture 2 happened.

enum mcpwm_counter_type_t

Select type of MCPWM counter.

Values:

MCPWM_UP_COUNTER = 1

For asymmetric MCPWM

`MCPWM_DOWN_COUNTER`
For asymmetric MCPWM

`MCPWM_UP_DOWN_COUNTER`
For symmetric MCPWM, frequency is half of MCPWM frequency set

`MCPWM_COUNTER_MAX`
Maximum counter mode

enum mcpwm_duty_type_t
Select type of MCPWM duty cycle mode.

Values:

`MCPWM_DUTY_MODE_0` = 0
Active high duty, i.e. duty cycle proportional to high time for asymmetric MCPWM

`MCPWM_DUTY_MODE_1`
Active low duty, i.e. duty cycle proportional to low time for asymmetric MCPWM, out of phase(inverted) MCPWM

`MCPWM_HAL_GENERATOR_MODE_FORCE_LOW`

`MCPWM_HAL_GENERATOR_MODE_FORCE_HIGH`

`MCPWM_DUTY_MODE_MAX`
Num of duty cycle modes

enum mcpwm_output_action_t
MCPWM select action to be taken on the output when event happens.

Values:

`MCPWM_ACTION_NO_CHANGE` = 0
No change in the output

`MCPWM_ACTION_FORCE_LOW`
Make output low

`MCPWM_ACTION_FORCE_HIGH`
Make output high

`MCPWM_ACTION_TOGGLE`
Make output toggle

enum mcpwm_deadtime_type_t
MCPWM deadtime types, used to generate deadtime, RED refers to rising edge delay and FED refers to falling edge delay.

Values:

`MCPWM_DEADTIME_BYPASS` = 0
Bypass the deadtime

`MCPWM_BYPASS_RED`
MCPWMXA = no change, MCPWMXB = falling edge delay

`MCPWM_BYPASS_FED`
MCPWMXA = rising edge delay, MCPWMXB = no change

`MCPWM_ACTIVE_HIGH_MODE`
MCPWMXA = rising edge delay, MCPWMXB = falling edge delay

`MCPWM_ACTIVE_LOW_MODE`
MCPWMXA = compliment of rising edge delay, MCPWMXB = compliment of falling edge delay

`MCPWM_ACTIVE_HIGH_COMPLIMENT_MODE`
MCPWMXA = rising edge delay, MCPWMXB = compliment of falling edge delay

`MCPWM_ACTIVE_LOW_COMPLIMENT_MODE`
MCPWMXA = compliment of rising edge delay, MCPWMXB = falling edge delay

`MCPWM_ACTIVE_RED_FED_FROM_PWMXA`
MCPWMXA = MCPWMXB = rising edge delay as well as falling edge delay, generated from MCPWMXA

`MCPWM_ACTIVE_RED_FED_FROM_PWMXB`
MCPWMXA = MCPWMXB = rising edge delay as well as falling edge delay, generated from MCPWMXB

`MCPWM_DEADTIME_TYPE_MAX`

enum mcpwm_sync_signal_t
MCPWM select sync signal input.

Values:

`MCPWM_SELECT_SYNC0` = 4
Select SYNC0 as input

`MCPWM_SELECT_SYNC1`
Select SYNC1 as input

`MCPWM_SELECT_SYNC2`
Select SYNC2 as input

enum mcpwm_capture_on_edge_t
MCPWM select capture starts from which edge.

Values:

`MCPWM_NEG_EDGE` = BIT(0)
Capture the negative edge

`MCPWM_POS_EDGE` = BIT(1)
Capture the positive edge

Header File

- driver/include/driver/mcpwm.h

Functions

警告: doxygenfunction: Unable to resolve multiple matches for function "_Static_assert" with arguments () in doxygen xml output for project "esp32-idf" from directory: xml_in/. Potential matches:

```
- _Static_assert(MCPWM_UNIT_MAX, "MCPWM unit number not equal to chip capabilities")
- _Static_assert(sizeof( ulp_insn_t ), "ULP coprocessor instruction size should be 4␣
↪bytes")
```

esp_err_t **mcpwm_gpio_init**(*mcpwm_unit_t mcpwm_num*, *mcpwm_io_signals_t io_signal*, int *gpio_num*)

This function initializes each gpio signal for MCPWM.

Note This function initializes one gpio at a time.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `mcpwm_num`: set MCPWM unit(0-1)
- `io_signal`: set MCPWM signals, each MCPWM unit has 6 output(MCPWMXA, MCPWMXB) and 9 input(SYNC_X, FAULT_X, CAP_X) 'X' is timer_num(0-2)
- `gpio_num`: set this to configure gpio for MCPWM, if you want to use gpio16, gpio_num = 16

esp_err_t **mcpwm_set_pin**(*mcpwm_unit_t mcpwm_num*, **const** *mcpwm_pin_config_t* **mcpwm_pin*)

Initialize MCPWM gpio structure.

Note This function can be used to initialize more then one gpio at a time.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `mcpwm_num`: set MCPWM unit(0-1)
- `mcpwm_pin`: MCPWM pin structure

esp_err_t **mcpwm_init**(*mcpwm_unit_t mcpwm_num*, *mcpwm_timer_t timer_num*, **const** *mcpwm_config_t* ***mcpwm_conf**)

Initialize MCPWM parameters.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `mcpwm_num`: set MCPWM unit(0-1)
- `timer_num`: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers.
- `mcpwm_conf`: configure structure *mcpwm_config_t*

esp_err_t **mcpwm_set_frequency**(*mcpwm_unit_t mcpwm_num*, *mcpwm_timer_t timer_num*, uint32_t *frequency*)

Set frequency(in Hz) of MCPWM timer.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `mcpwm_num`: set MCPWM unit(0-1)
- `timer_num`: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers
- `frequency`: set the frequency in Hz of each timer

esp_err_t **mcpwm_set_duty**(*mcpwm_unit_t mcpwm_num*, *mcpwm_timer_t timer_num*, *mcpwm_generator_t gen*, float *duty*)

Set duty cycle of each operator(MCPWMXA/MCPWMXB)

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `mcpwm_num`: set MCPWM unit(0-1)
- `timer_num`: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers

- `gen`: set the generator(MCPWMXA/MCPWMXB), 'X' is operator number selected
- `duty`: set duty cycle in %(i.e for 62.3% duty cycle, duty = 62.3) of each operator

esp_err_t **mcpwm_set_duty_in_us**(*mcpwm_unit_t mcpwm_num*, *mcpwm_timer_t timer_num*, *mcpwm_generator_t gen*, uint32_t *duty_in_us*)

Set duty cycle of each operator(MCPWMXA/MCPWMXB) in us.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `mcpwm_num`: set MCPWM unit(0-1)
- `timer_num`: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers
- `gen`: set the generator(MCPWMXA/MCPWMXB), 'x' is operator number selected
- `duty_in_us`: set duty value in microseconds of each operator

esp_err_t **mcpwm_set_duty_type**(*mcpwm_unit_t mcpwm_num*, *mcpwm_timer_t timer_num*, *mcpwm_generator_t gen*, *mcpwm_duty_type_t duty_type*)

Set duty either active high or active low(out of phase/inverted)

Note Call this function every time after mcpwm_set_signal_high or mcpwm_set_signal_low to resume with previously set duty cycle

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `mcpwm_num`: set MCPWM unit(0-1)
- `timer_num`: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers
- `gen`: set the generator(MCPWMXA/MCPWMXB), 'x' is operator number selected
- `duty_type`: set active low or active high duty type

uint32_t **mcpwm_get_frequency**(*mcpwm_unit_t mcpwm_num*, *mcpwm_timer_t timer_num*)

Get frequency of timer.

Return

- frequency of timer

Parameters

- `mcpwm_num`: set MCPWM unit(0-1)
- `timer_num`: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers

float **mcpwm_get_duty**(*mcpwm_unit_t mcpwm_num*, *mcpwm_timer_t timer_num*, *mcpwm_operator_t gen*)

Get duty cycle of each operator.

Return

- duty cycle in % of each operator(56.7 means duty is 56.7%)

Parameters

- `mcpwm_num`: set MCPWM unit(0-1)
- `timer_num`: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers
- `gen`: set the generator(MCPWMXA/MCPWMXB), 'x' is operator number selected

esp_err_t **mcpwm_set_signal_high**(*mcpwm_unit_t mcpwm_num*, *mcpwm_timer_t timer_num*, *mcpwm_generator_t gen*)

Use this function to set MCPWM signal high.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `mcpwm_num`: set MCPWM unit(0-1)
- `timer_num`: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers
- `gen`: set the operator(MCPWMXA/MCPWMXB), 'x' is timer number selected

esp_err_t **mcpwm_set_signal_low**(*mcpwm_unit_t mcpwm_num*, *mcpwm_timer_t timer_num*, *mcpwm_generator_t gen*)

Use this function to set MCPWM signal low.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `mcpwm_num`: set MCPWM unit(0-1)
- `timer_num`: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers
- `gen`: set the operator(MCPWMXA/MCPWMXB), 'x' is timer number selected

esp_err_t **mcpwm_start**(*mcpwm_unit_t mcpwm_num*, *mcpwm_timer_t timer_num*)

Start MCPWM signal on timer 'x'.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `mcpwm_num`: set MCPWM unit(0-1)
- `timer_num`: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers

esp_err_t **mcpwm_stop**(*mcpwm_unit_t mcpwm_num*, *mcpwm_timer_t timer_num*)

Start MCPWM signal on timer 'x'.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `mcpwm_num`: set MCPWM unit(0-1)
- `timer_num`: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers

esp_err_t **mcpwm_carrier_init**(*mcpwm_unit_t mcpwm_num*, *mcpwm_timer_t timer_num*, **const** *mcpwm_carrier_config_t *carrier_conf*)

Initialize carrier configuration.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `mcpwm_num`: set MCPWM unit(0-1)
- `timer_num`: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers
- `carrier_conf`: configure structure *mcpwm_carrier_config_t*

esp_err_t **mcpwm_carrier_enable**(*mcpwm_unit_t mcpwm_num*, *mcpwm_timer_t timer_num*)

Enable MCPWM carrier submodule, for respective timer.

Return

- ESP_OK Success

- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `mcpwm_num`: set MCPWM unit(0-1)
- `timer_num`: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers

esp_err_t **mcpwm_carrier_disable**(*mcpwm_unit_t mcpwm_num*, *mcpwm_timer_t timer_num*)

Disable MCPWM carrier submodule, for respective timer.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `mcpwm_num`: set MCPWM unit(0-1)
- `timer_num`: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers

esp_err_t **mcpwm_carrier_set_period**(*mcpwm_unit_t mcpwm_num*, *mcpwm_timer_t timer_num*, uint8_t *carrier_period*)

Set period of carrier.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `mcpwm_num`: set MCPWM unit(0-1)
- `timer_num`: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers
- `carrier_period`: set the carrier period of each timer, carrier period = (carrier_period + 1)*800ns (carrier_period <= 15)

esp_err_t **mcpwm_carrier_set_duty_cycle**(*mcpwm_unit_t mcpwm_num*, *mcpwm_timer_t timer_num*, uint8_t *carrier_duty*)

Set duty_cycle of carrier.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `mcpwm_num`: set MCPWM unit(0-1)

- `timer_num`: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers
- `carrier_duty`: set duty_cycle of carrier , carrier duty cycle = carrier_duty*12.5% (chop_duty <= 7)

esp_err_t **mcpwm_carrier_oneshot_mode_enable**(*mcpwm_unit_t mcpwm_num*, *mcpwm_timer_t timer_num*, uint8_t *pulse_width*)

Enable and set width of first pulse in carrier oneshot mode.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `mcpwm_num`: set MCPWM unit(0-1)
- `timer_num`: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers
- `pulse_width`: set pulse width of first pulse in oneshot mode, width = (carrier period)*(pulse_width +1) (pulse_width <= 15)

esp_err_t **mcpwm_carrier_oneshot_mode_disable**(*mcpwm_unit_t mcpwm_num*, *mcpwm_timer_t timer_num*)

Disable oneshot mode, width of first pulse = carrier period.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `mcpwm_num`: set MCPWM unit(0-1)
- `timer_num`: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers

esp_err_t **mcpwm_carrier_output_invert**(*mcpwm_unit_t mcpwm_num*, *mcpwm_timer_t timer_num*, *mcpwm_carrier_out_ivt_t carrier_ivt_mode*)

Enable or disable carrier output inversion.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `mcpwm_num`: set MCPWM unit(0-1)

- `timer_num`: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers
- `carrier_ivt_mode`: enable or disable carrier output inversion

esp_err_t **mcpwm_deadtime_enable**(*mcpwm_unit_t mcpwm_num*, *mcpwm_timer_t timer_num*, *mcpwm_deadtime_type_t dt_mode*, uint32_t *red*, uint32_t *fed*)

Enable and initialize deadtime for each MCPWM timer.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `mcpwm_num`: set MCPWM unit(0-1)
- `timer_num`: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers
- `dt_mode`: set deadtime mode
- `red`: set rising edge delay = red*100ns
- `fed`: set rising edge delay = fed*100ns

esp_err_t **mcpwm_deadtime_disable**(*mcpwm_unit_t mcpwm_num*, *mcpwm_timer_t timer_num*)

Disable deadtime on MCPWM timer.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `mcpwm_num`: set MCPWM unit(0-1)
- `timer_num`: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers

esp_err_t **mcpwm_fault_init**(*mcpwm_unit_t mcpwm_num*, *mcpwm_fault_input_level_t input_level*, *mcpwm_fault_signal_t fault_sig*)

Initialize fault submodule, currently low level triggering is not supported.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `mcpwm_num`: set MCPWM unit(0-1)

- `intput_level`: set fault signal level, which will cause fault to occur
- `fault_sig`: set the fault pin, which needs to be enabled

esp_err_t **mcpwm_fault_set_oneshot_mode**(*mcpwm_unit_t mcpwm_num*, *mcpwm_timer_t timer_num*, *mcpwm_fault_signal_t fault_sig*, *mcpwm_output_action_t action_on_pwmxa*, *mcpwm_output_action_t action_on_pwmxb*)

Set oneshot mode on fault detection, once fault occur in oneshot mode reset is required to resume MCPWM signals.

Note currently low level triggering is not supported

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `mcpwm_num`: set MCPWM unit(0-1)
- `timer_num`: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers
- `fault_sig`: set the fault pin, which needs to be enabled for oneshot mode
- `action_on_pwmxa`: action to be taken on MCPWMXA when fault occurs, either no change or high or low or toggle
- `action_on_pwmxb`: action to be taken on MCPWMXB when fault occurs, either no change or high or low or toggle

esp_err_t **mcpwm_fault_set_cyc_mode**(*mcpwm_unit_t mcpwm_num*, *mcpwm_timer_t timer_num*, *mcpwm_fault_signal_t fault_sig*, *mcpwm_output_action_t action_on_pwmxa*, *mcpwm_output_action_t action_on_pwmxb*)

Set cycle-by-cycle mode on fault detection, once fault occur in cyc mode MCPWM signal resumes as soon as fault signal becomes inactive.

Note currently low level triggering is not supported

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `mcpwm_num`: set MCPWM unit(0-1)
- `timer_num`: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers

- `fault_sig`: set the fault pin, which needs to be enabled for cyc mode
- `action_on_pwmxa`: action to be taken on MCPWMXA when fault occurs, either no change or high or low or toggle
- `action_on_pwmxb`: action to be taken on MCPWMXB when fault occurs, either no change or high or low or toggle

esp_err_t **mcpwm_fault_deinit**(*mcpwm_unit_t mcpwm_num*, *mcpwm_fault_signal_t fault_sig*)

Disable fault signal.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `mcpwm_num`: set MCPWM unit(0-1)
- `fault_sig`: fault pin, which needs to be disabled

esp_err_t **mcpwm_capture_enable**(*mcpwm_unit_t mcpwm_num*, *mcpwm_capture_signal_t cap_sig*, *mcpwm_capture_on_edge_t cap_edge*, uint32_t *num_of_pulse*)

Initialize capture submodule.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `mcpwm_num`: set MCPWM unit(0-1)
- `cap_edge`: set capture edge, BIT(0) - negative edge, BIT(1) - positive edge
- `cap_sig`: capture pin, which needs to be enabled
- `num_of_pulse`: count time between rising/falling edge between 2 *(pulses mentioned), counter uses APB_CLK

esp_err_t **mcpwm_capture_disable**(*mcpwm_unit_t mcpwm_num*, *mcpwm_capture_signal_t cap_sig*)

Disable capture signal.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `mcpwm_num`: set MCPWM unit(0-1)
- `cap_sig`: capture pin, which needs to be disabled

uint32_t **mcpwm_capture_signal_get_value**(*mcpwm_unit_t mcpwm_num*, *mcpwm_capture_signal_t cap_sig*)

Get capture value.

Return Captured value

Parameters

- `mcpwm_num`: set MCPWM unit(0-1)
- `cap_sig`: capture pin on which value is to be measured

uint32_t **mcpwm_capture_signal_get_edge**(*mcpwm_unit_t mcpwm_num*, *mcpwm_capture_signal_t cap_sig*)

Get edge of capture signal.

Return Capture signal edge: 1 - positive edge, 2 - negtive edge

Parameters

- `mcpwm_num`: set MCPWM unit(0-1)
- `cap_sig`: capture pin of whose edge is to be determined

esp_err_t **mcpwm_sync_enable**(*mcpwm_unit_t mcpwm_num*, *mcpwm_timer_t timer_num*, *mcpwm_sync_signal_t sync_sig*, uint32_t *phase_val*)

Initialize sync submodule.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `mcpwm_num`: set MCPWM unit(0-1)
- `timer_num`: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers
- `sync_sig`: set the synchronization pin, which needs to be enabled
- `phase_val`: phase value in 1/1000 (for 86.7%, phase_val = 867) which timer moves to on sync signal

esp_err_t **mcpwm_sync_disable**(*mcpwm_unit_t mcpwm_num*, *mcpwm_timer_t timer_num*)

Disable sync submodule on given timer.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `mcpwm_num`: set MCPWM unit(0-1)
- `timer_num`: set timer number(0-2) of MCPWM, each MCPWM unit has 3 timers

esp_err_t **mcpwm_isr_register**(*mcpwm_unit_t mcpwm_num*, void (**fn*))void *
, void **arg*, int *intr_alloc_flags*, *intr_handle_t* **handle*Register MCPWM interrupt handler, the handler is an ISR. the handler will be attached to the same CPU core that this function is running on.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Function pointer error.

Parameters

- `mcpwm_num`: set MCPWM unit(0-1)
- `fn`: interrupt handler function.
- `arg`: user-supplied argument passed to the handler function.
- `intr_alloc_flags`: flags used to allocate the interrupt. One or multiple (ORred) ESP_INTR_FLAG_* values. see esp_intr_alloc.h for more info.
- `handle`: pointer to return handle. If non-NULL, a handle for the interrupt will be returned here.

Structures

struct mcpwm_pin_config_t

MCPWM pin number for.

Public Members

int **mcpwm0a_out_num**

MCPWM0A out pin

int **mcpwm0b_out_num**

MCPWM0A out pin

int **mcpwm1a_out_num**

MCPWM0A out pin

int **mcpwm1b_out_num**
MCPWM0A out pin

int **mcpwm2a_out_num**
MCPWM0A out pin

int **mcpwm2b_out_num**
MCPWM0A out pin

int **mcpwm_sync0_in_num**
SYNC0 in pin

int **mcpwm_sync1_in_num**
SYNC1 in pin

int **mcpwm_sync2_in_num**
SYNC2 in pin

int **mcpwm_fault0_in_num**
FAULT0 in pin

int **mcpwm_fault1_in_num**
FAULT1 in pin

int **mcpwm_fault2_in_num**
FAULT2 in pin

int **mcpwm_cap0_in_num**
CAP0 in pin

int **mcpwm_cap1_in_num**
CAP1 in pin

int **mcpwm_cap2_in_num**
CAP2 in pin

struct mcpwm_config_t
MCPWM config structure.

Public Members

uint32_t **frequency**
Set frequency of MCPWM in Hz

float **cmpr_a**
Set % duty cycle for operator a(MCPWMXA), i.e for 62.3% duty cycle, duty_a = 62.3

float **cmpr_b**
Set % duty cycle for operator b(MCPWMXB), i.e for 48% duty cycle, duty_b = 48.0

mcpwm_duty_type_t **duty_mode**

Set type of duty cycle

mcpwm_counter_type_t **counter_mode**

Set type of MCPWM counter

struct mcpwm_carrier_config_t

MCPWM config carrier structure.

Public Members

uint8_t **carrier_period**

Set carrier period = (carrier_period + 1)*800ns, carrier_period should be < 16

uint8_t **carrier_duty**

Set carrier duty cycle, carrier_duty should be less than 8 (increment every 12.5%)

uint8_t **pulse_width_in_os**

Set pulse width of first pulse in one shot mode = (carrier period)*(pulse_width_in_os + 1), should be less then 16

mcpwm_carrier_os_t **carrier_os_mode**

Enable or disable carrier oneshot mode

mcpwm_carrier_out_ivt_t **carrier_ivt_mode**

Invert output of carrier

Macros

MCPWM_OPR_A

MCPWM_OPR_B

MCPWM_OPR_MAX

MCPWM_NO_CHANGE_IN_MCPWMXA

MCPWM_FORCE_MCPWMXA_LOW

MCPWM_FORCE_MCPWMXA_HIGH

MCPWM_TOG_MCPWMXA

MCPWM_NO_CHANGE_IN_MCPWMXB

MCPWM_FORCE_MCPWMXB_LOW

MCPWM_FORCE_MCPWMXB_HIGH

MCPWM_TOG_MCPWMXB

Type Definitions

typedef *mcpwm_generator_t* mcpwm_operator_t

typedef *mcpwm_output_action_t* mcpwm_action_on_pwmxa_t

typedef *mcpwm_output_action_t* mcpwm_action_on_pwmxb_t

Enumerations

enum mcpwm_io_signals_t

IO signals for the MCPWM.

- 6 MCPWM output pins that generate PWM signals
- 3 MCPWM fault input pins to detect faults like overcurrent, overvoltage, etc.
- 3 MCPWM sync input pins to synchronize MCPWM outputs signals
- 3 MCPWM capture input pins to gather feedback from controlled motors, using e.g. hall sensors

Values:

MCPWM0A = 0

PWM0A output pin

MCPWM0B

PWM0B output pin

MCPWM1A

PWM1A output pin

MCPWM1B

PWM1B output pin

MCPWM2A

PWM2A output pin

MCPWM2B

PWM2B output pin

MCPWM_SYNC_0

SYNC0 input pin

MCPWM_SYNC_1

SYNC1 input pin

MCPWM_SYNC_2

SYNC2 input pin

`MCPWM_FAULT_0`
FAULT0 input pin

`MCPWM_FAULT_1`
FAULT1 input pin

`MCPWM_FAULT_2`
FAULT2 input pin

`MCPWM_CAP_0` = 84
CAP0 input pin

`MCPWM_CAP_1`
CAP1 input pin

`MCPWM_CAP_2`
CAP2 input pin

enum mcpwm_unit_t
Select MCPWM unit.

Values:

`MCPWM_UNIT_0` = 0
MCPWM unit0 selected

`MCPWM_UNIT_1`
MCPWM unit1 selected

`MCPWM_UNIT_MAX`
Num of MCPWM units on ESP32

enum mcpwm_timer_t
Select MCPWM timer.

Values:

`MCPWM_TIMER_0` = 0
Select MCPWM timer0

`MCPWM_TIMER_1`
Select MCPWM timer1

`MCPWM_TIMER_2`
Select MCPWM timer2

`MCPWM_TIMER_MAX`
Num of MCPWM timers on ESP32

enum mcpwm_generator_t
Select MCPWM operator.

Values:

`MCPWM_GEN_A` = 0

Select MCPWMXA, where 'X' is operator number

`MCPWM_GEN_B`

Select MCPWMXB, where 'X' is operator number

`MCPWM_GEN_MAX`

Num of generators to each operator of MCPWM

enum mcpwm_carrier_os_t

MCPWM carrier oneshot mode, in this mode the width of the first pulse of carrier can be programmed.

Values:

`MCPWM_ONESHOT_MODE_DIS` = 0

Enable oneshot mode

`MCPWM_ONESHOT_MODE_EN`

Disable oneshot mode

enum mcpwm_carrier_out_ivt_t

MCPWM carrier output inversion, high frequency carrier signal active with MCPWM signal is high.

Values:

`MCPWM_CARRIER_OUT_IVT_DIS` = 0

Enable carrier output inversion

`MCPWM_CARRIER_OUT_IVT_EN`

Disable carrier output inversion

enum mcpwm_fault_signal_t

MCPWM select fault signal input.

Values:

`MCPWM_SELECT_F0` = 0

Select F0 as input

`MCPWM_SELECT_F1`

Select F1 as input

`MCPWM_SELECT_F2`

Select F2 as input

enum mcpwm_fault_input_level_t

MCPWM select triggering level of fault signal.

Values:

`MCPWM_LOW_LEVEL_TGR` = 0

Fault condition occurs when fault input signal goes from high to low, currently not supported

`MCPWM_HIGH_LEVEL_TGR`
Fault condition occurs when fault input signal goes low to high

`enum mcpwm_capture_signal_t`
MCPWM select capture signal input.

Values:

`MCPWM_SELECT_CAP0` = 0
Select CAP0 as input

`MCPWM_SELECT_CAP1`
Select CAP1 as input

`MCPWM_SELECT_CAP2`
Select CAP2 as input

2.3.9 Pulse Counter

Introduction

The PCNT (Pulse Counter) module is designed to count the number of rising and/or falling edges of an input signal. Each pulse counter unit has a 16-bit signed counter register and two channels that can be configured to either increment or decrement the counter. Each channel has a signal input that accepts signal edges to be detected, as well as a control input that can be used to enable or disable the signal input. The inputs have optional filters that can be used to discard unwanted glitches in the signal.

Functionality Overview

Description of functionality of this API has been broken down into four sections:

- *Configuration* - describes counter's configuration parameters and how to setup the counter.
- *Operating the Counter* - provides information on control functions to pause, measure and clear the counter.
- *Filtering Pulses* - describes options to filtering pulses and the counter control signals.
- *Using Interrupts* - presents how to trigger interrupts on specific states of the counter.

Configuration

The PCNT module has eight independent counting "units" numbered from 0 to 7. In the API they are referred to using *pcnt_unit_t*. Each unit has two independent channels numbered as 0 and 1 and specified with *pcnt_channel_t*.

The configuration is provided separately per unit's channel using *pcnt_config_t* and covers:

- The unit and the channel number this configuration refers to.
- GPIO numbers of the pulse input and the pulse gate input.
- Two pairs of parameters: *pcnt_ctrl_mode_t* and *pcnt_count_mode_t* to define how the counter reacts depending on the the status of control signal and how counting is done positive / negative edge of the pulses.
- Two limit values (minimum / maximum) that are used to establish watchpoints and trigger interrupts when the pulse count is meeting particular limit.

Setting up of particular channel is then done by calling a function *pcnt_unit_config()* with above *pcnt_config_t* as the input parameter.

To disable the pulse or the control input pin in configuration, provide `PCNT_PIN_NOT_USED` instead of the GPIO number.

Operating the Counter

After doing setup with *pcnt_unit_config()*, the counter immediately starts to operate. The accumulated pulse count can be checked by calling *pcnt_get_counter_value()*.

There are couple of functions that allow to control the counter' s operation: *pcnt_counter_pause()*, *pcnt_counter_resume()* and *pcnt_counter_clear()*

It is also possible to dynamically change the previously set up counter modes with *pcnt_unit_config()* by calling *pcnt_set_mode()*.

If desired, the pulse input pin and the control input pin may be changed "on the fly" using *pcnt_set_pin()*. To disable particular input provide as a function parameter `PCNT_PIN_NOT_USED` instead of the GPIO number.

注解: For the counter not to miss any pulses, the pulse duration should be longer than one APB_CLK cycle (12.5 ns). The pulses are sampled on the edges of the APB_CLK clock and may be missed, if fall between the edges. This applies to counter operation with or without a *filer*.

Filtering Pulses

The PCNT unit features filters on each of the pulse and control inputs, adding the option to ignore short glitches in the signals.

The length of ignored pulses is provided in APB_CLK clock cycles by calling *pcnt_set_filter_value()*. The current filter setting may be checked with *pcnt_get_filter_value()*. The APB_CLK clock is running at 80 MHz.

The filter is put into operation / suspended by calling *pcnt_filter_enable()* / *pcnt_filter_disable()*.

Using Interrupts

There are five counter state watch events, defined in *pcnt_evt_type_t*, that are able to trigger an interrupt. The event happens on the pulse counter reaching specific values:

- Minimum or maximum count values: `counter_l_lim` or `counter_h_lim` provided in *pcnt_config_t* as discussed in *Configuration*
- Threshold 0 or Threshold 1 values set using function *pcnt_set_event_value()*.
- Pulse count = 0

To register, enable or disable an interrupt to service the above events, call *pcnt_isr_register()*, *pcnt_intr_enable()*. and *pcnt_intr_disable()*. To enable or disable events on reaching threshold values, you will also need to call functions *pcnt_event_enable()* and *pcnt_event_disable()*.

In order to check what are the threshold values currently set, use function *pcnt_get_event_value()*.

Application Example

Pulse counter with control signal and event interrupt example: peripherals/pcnt.

API Reference

Header File

- driver/include/driver/pcnt.h

Functions

esp_err_t **pcnt_unit_config**(**const** *pcnt_config_t* **pcnt_config*)

Configure Pulse Counter unit.

Note This function will disable three events: PCNT_EVT_L_LIM, PCNT_EVT_H_LIM, PCNT_EVT_ZERO.

Return

- ESP_OK Success
- ESP_ERR_INVALID_STATE pcnt driver already initialized
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `pcnt_config`: Pointer of Pulse Counter unit configure parameter

esp_err_t **pcnt_get_counter_value**(*pcnt_unit_t pcnt_unit*, int16_t **count*)

Get pulse counter value.

Return

- ESP_OK Success
- ESP_ERR_INVALID_STATE pcnt driver has not been initialized
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `pcnt_unit`: Pulse Counter unit number
- `count`: Pointer to accept counter value

esp_err_t **pcnt_counter_pause**(*pcnt_unit_t pcnt_unit*)

Pause PCNT counter of PCNT unit.

Return

- ESP_OK Success
- ESP_ERR_INVALID_STATE pcnt driver has not been initialized
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `pcnt_unit`: PCNT unit number

esp_err_t **pcnt_counter_resume**(*pcnt_unit_t pcnt_unit*)

Resume counting for PCNT counter.

Return

- ESP_OK Success
- ESP_ERR_INVALID_STATE pcnt driver has not been initialized
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `pcnt_unit`: PCNT unit number, select from pcnt_unit_t

esp_err_t **pcnt_counter_clear**(*pcnt_unit_t pcnt_unit*)

Clear and reset PCNT counter value to zero.

Return

- ESP_OK Success

- ESP_ERR_INVALID_STATE pcnt driver has not been initialized
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `pcnt_unit`: PCNT unit number, select from pcnt_unit_t

esp_err_t **pcnt_intr_enable**(*pcnt_unit_t pcnt_unit*)

Enable PCNT interrupt for PCNT unit.

Note Each Pulse counter unit has five watch point events that share the same interrupt. Configure events with pcnt_event_enable() and pcnt_event_disable()

Return

- ESP_OK Success
- ESP_ERR_INVALID_STATE pcnt driver has not been initialized
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `pcnt_unit`: PCNT unit number

esp_err_t **pcnt_intr_disable**(*pcnt_unit_t pcnt_unit*)

Disable PCNT interrupt for PCNT unit.

Return

- ESP_OK Success
- ESP_ERR_INVALID_STATE pcnt driver has not been initialized
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `pcnt_unit`: PCNT unit number

esp_err_t **pcnt_event_enable**(*pcnt_unit_t unit*, *pcnt_evt_type_t evt_type*)

Enable PCNT event of PCNT unit.

Return

- ESP_OK Success
- ESP_ERR_INVALID_STATE pcnt driver has not been initialized
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `unit`: PCNT unit number

- `evt_type`: Watch point event type. All enabled events share the same interrupt (one interrupt per pulse counter unit).

esp_err_t **pcnt_event_disable**(*pcnt_unit_t unit*, *pcnt_evt_type_t evt_type*)

Disable PCNT event of PCNT unit.

Return

- ESP_OK Success
- ESP_ERR_INVALID_STATE pcnt driver has not been initialized
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `unit`: PCNT unit number
- `evt_type`: Watch point event type. All enabled events share the same interrupt (one interrupt per pulse counter unit).

esp_err_t **pcnt_set_event_value**(*pcnt_unit_t unit*, *pcnt_evt_type_t evt_type*, int16_t *value*)

Set PCNT event value of PCNT unit.

Return

- ESP_OK Success
- ESP_ERR_INVALID_STATE pcnt driver has not been initialized
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `unit`: PCNT unit number
- `evt_type`: Watch point event type. All enabled events share the same interrupt (one interrupt per pulse counter unit).
- `value`: Counter value for PCNT event

esp_err_t **pcnt_get_event_value**(*pcnt_unit_t unit*, *pcnt_evt_type_t evt_type*, int16_t **value*)

Get PCNT event value of PCNT unit.

Return

- ESP_OK Success
- ESP_ERR_INVALID_STATE pcnt driver has not been initialized
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `unit`: PCNT unit number
- `evt_type`: Watch point event type. All enabled events share the same interrupt (one interrupt per pulse counter unit).
- `value`: Pointer to accept counter value for PCNT event

esp_err_t **pcnt_isr_register**(void (*fn))void *
, void *arg, int *intr_alloc_flags*, *pcnt_isr_handle_t* *handleRegister PCNT interrupt handler, the handler is an ISR. The handler will be attached to the same CPU core that this function is running on. Please do not use pcnt_isr_service_install if this function was called.

Return

- ESP_OK Success
- ESP_ERR_NOT_FOUND Can not find the interrupt that matches the flags.
- ESP_ERR_INVALID_ARG Function pointer error.

Parameters

- `fn`: Interrupt handler function.
- `arg`: Parameter for handler function
- `intr_alloc_flags`: Flags used to allocate the interrupt. One or multiple (ORred) ESP_INTR_FLAG_* values. See esp_intr_alloc.h for more info.
- `handle`: Pointer to return handle. If non-NULL, a handle for the interrupt will be returned here. Calling esp_intr_free to unregister this ISR service if needed, but only if the handle is not NULL.

esp_err_t **pcnt_set_pin**(*pcnt_unit_t unit*, *pcnt_channel_t channel*, int *pulse_io*, int *ctrl_io*)
Configure PCNT pulse signal input pin and control input pin.

Note Set the signal input to PCNT_PIN_NOT_USED if unused.

Return

- ESP_OK Success
- ESP_ERR_INVALID_STATE pcnt driver has not been initialized
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `unit`: PCNT unit number
- `channel`: PCNT channel number
- `pulse_io`: Pulse signal input GPIO

- `ctrl_io`: Control signal input GPIO

esp_err_t **pcnt_filter_enable**(*pcnt_unit_t unit*)

Enable PCNT input filter.

Return

- ESP_OK Success
- ESP_ERR_INVALID_STATE pcnt driver has not been initialized
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `unit`: PCNT unit number

esp_err_t **pcnt_filter_disable**(*pcnt_unit_t unit*)

Disable PCNT input filter.

Return

- ESP_OK Success
- ESP_ERR_INVALID_STATE pcnt driver has not been initialized
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `unit`: PCNT unit number

esp_err_t **pcnt_set_filter_value**(*pcnt_unit_t unit*, uint16_t *filter_val*)

Set PCNT filter value.

Note filter_val is a 10-bit value, so the maximum filter_val should be limited to 1023.

Return

- ESP_OK Success
- ESP_ERR_INVALID_STATE pcnt driver has not been initialized
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `unit`: PCNT unit number
- `filter_val`: PCNT signal filter value, counter in APB_CLK cycles. Any pulses lasting shorter than this will be ignored when the filter is enabled.

esp_err_t **pcnt_get_filter_value**(*pcnt_unit_t unit*, uint16_t **filter_val*)

Get PCNT filter value.

Return

- ESP_OK Success
- ESP_ERR_INVALID_STATE pcnt driver has not been initialized
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `unit`: PCNT unit number
- `filter_val`: Pointer to accept PCNT filter value.

esp_err_t **pcnt_set_mode**(*pcnt_unit_t unit*, *pcnt_channel_t channel*, *pcnt_count_mode_t pos_mode*, *pcnt_count_mode_t neg_mode*, *pcnt_ctrl_mode_t hctrl_mode*, *pcnt_ctrl_mode_t lctrl_mode*)

Set PCNT counter mode.

Return

- ESP_OK Success
- ESP_ERR_INVALID_STATE pcnt driver has not been initialized
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `unit`: PCNT unit number
- `channel`: PCNT channel number
- `pos_mode`: Counter mode when detecting positive edge
- `neg_mode`: Counter mode when detecting negative edge
- `hctrl_mode`: Counter mode when control signal is high level
- `lctrl_mode`: Counter mode when control signal is low level

esp_err_t **pcnt_isr_handler_add**(*pcnt_unit_t unit*, void (**isr_handler*))void *

, void **args*Add ISR handler for specified unit.

Call this function after using pcnt_isr_service_install() to install the PCNT driver' s ISR handler service.

The ISR handlers do not need to be declared with IRAM_ATTR, unless you pass the ESP_INTR_FLAG_IRAM flag when allocating the ISR in pcnt_isr_service_install().

This ISR handler will be called from an ISR. So there is a stack size limit (configurable as "ISR stack size" in menuconfig). This limit is smaller compared to a global PCNT interrupt handler due to the additional level of indirection.

Return

- ESP_OK Success
- ESP_ERR_INVALID_STATE pcnt driver has not been initialized
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `unit`: PCNT unit number
- `isr_handler`: Interrupt handler function.
- `args`: Parameter for handler function

esp_err_t **pcnt_isr_service_install**(int *intr_alloc_flags*)

Install PCNT ISR service.

Note We can manage different interrupt service for each unit. This function will use the default ISR handle service, Calling pcnt_isr_service_uninstall to uninstall the default service if needed. Please do not use pcnt_isr_register if this function was called.

Return

- ESP_OK Success
- ESP_ERR_INVALID_STATE pcnt driver has not been initialized
- ESP_ERR_NO_MEM No memory to install this service
- ESP_ERR_INVALID_STATE ISR service already installed

Parameters

- `intr_alloc_flags`: Flags used to allocate the interrupt. One or multiple (ORred) ESP_INTR_FLAG_* values. See esp_intr_alloc.h for more info.

void **pcnt_isr_service_uninstall**(void)

Uninstall PCNT ISR service, freeing related resources.

esp_err_t **pcnt_isr_handler_remove**(*pcnt_unit_t unit*)

Delete ISR handler for specified unit.

Return

- ESP_OK Success
- ESP_ERR_INVALID_STATE pcnt driver has not been initialized
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `unit`: PCNT unit number

Type Definitions

typedef *intr_handle_t* **pcnt_isr_handle_t**

Header File

- soc/include/hal/pcnt_types.h

Structures

struct pcnt_config_t

Pulse Counter configuration for a single channel.

Public Members

int **pulse_gpio_num**

Pulse input GPIO number, if you want to use GPIO16, enter pulse_gpio_num = 16, a negative value will be ignored

int **ctrl_gpio_num**

Control signal input GPIO number, a negative value will be ignored

pcnt_ctrl_mode_t **lctrl_mode**

PCNT low control mode

pcnt_ctrl_mode_t **hctrl_mode**

PCNT high control mode

pcnt_count_mode_t **pos_mode**

PCNT positive edge count mode

pcnt_count_mode_t **neg_mode**

PCNT negative edge count mode

int16_t **counter_h_lim**

Maximum counter value

int16_t **counter_l_lim**

Minimum counter value

pcnt_unit_t **unit**

PCNT unit number

pcnt_channel_t **channel**

the PCNT channel

Type Definitions

typedef int pcnt_port_t

PCNT port number, the max port number is (PCNT_PORT_MAX - 1).

typedef int pcnt_unit_t

Selection of all available PCNT units.

Enumerations

enum pcnt_ctrl_mode_t

Selection of available modes that determine the counter's action depending on the state of the control signal's input GPIO.

Note Configuration covers two actions, one for high, and one for low level on the control input

Values:

PCNT_MODE_KEEP = 0

Control mode: won't change counter mode

PCNT_MODE_REVERSE = 1

Control mode: invert counter mode(increase -> decrease, decrease -> increase)

PCNT_MODE_DISABLE = 2

Control mode: Inhibit counter(counter value will not change in this condition)

PCNT_MODE_MAX

enum pcnt_count_mode_t

Selection of available modes that determine the counter's action on the edge of the pulse signal's input GPIO.

Note Configuration covers two actions, one for positive, and one for negative edge on the pulse input

Values:

PCNT_COUNT_DIS = 0

Counter mode: Inhibit counter(counter value will not change in this condition)

PCNT_COUNT_INC = 1

Counter mode: Increase counter value

PCNT_COUNT_DEC = 2

Counter mode: Decrease counter value

PCNT_COUNT_MAX

enum pcnt_channel_t

Selection of channels available for a single PCNT unit.

Values:

PCNT_CHANNEL_0 = 0x00

PCNT channel 0

PCNT_CHANNEL_1 = 0x01

PCNT channel 1

PCNT_CHANNEL_MAX

enum pcnt_evt_type_t

Selection of counter' s events the may trigger an interrupt.

Values:

PCNT_EVT_THRES_1 = BIT(2)

PCNT watch point event: threshold1 value event

PCNT_EVT_THRES_0 = BIT(3)

PCNT watch point event: threshold0 value event

PCNT_EVT_L_LIM = BIT(4)

PCNT watch point event: Minimum counter value

PCNT_EVT_H_LIM = BIT(5)

PCNT watch point event: Maximum counter value

PCNT_EVT_ZERO = BIT(6)

PCNT watch point event: counter value zero event

PCNT_EVT_MAX

2.3.10 RMT

The RMT (Remote Control) module driver can be used to send and receive infrared remote control signals. Due to flexibility of RMT module, the driver can also be used to generate or receive many other types of signals.

The signal, which consists of a series of pulses, is generated by RMT' s transmitter based on a list of values. The values define the pulse duration and a binary level, see below. The transmitter can also provide a carrier and modulate it with provided pulses.

The reverse operation is performed by the receiver, where a series of pulses is decoded into a list of values containing the pulse duration and binary level. A filter may be applied to remove high frequency noise from the input signal.

There couple of typical steps to setup and operate the RMT and they are discussed in the following sections:

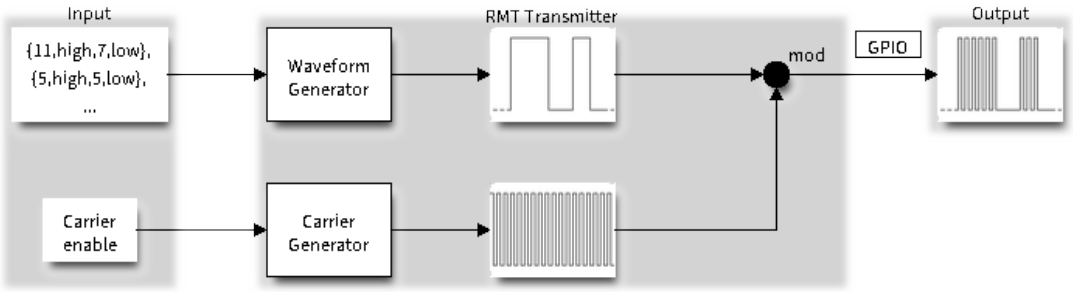


图 19: RMT Transmitter Overview

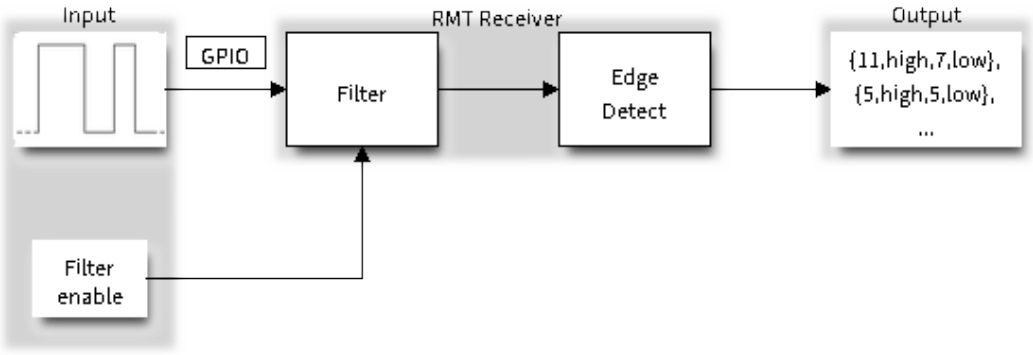


图 20: RMT Receiver Overview

1. *Configure Driver*
2. *Transmit Data* or *Receive Data*
3. *Change Operation Parameters*
4. *Use Interrupts*

The RMT has eight channels numbered from zero to seven. Each channel is able to independently transmit or receive data. They are referred to using indexes defined in structure *rmt_channel_t*.

Configure Driver

There are several parameters that define how particular channel operates. Most of these parameters are configured by setting specific members of *rmt_config_t* structure. Some of the parameters are common to both transmit or receive mode, and some are mode specific. They are all discussed below.

Common Parameters

- The **channel** to be configured, select one from the *rmt_channel_t* enumerator.
- The RMT **operation mode** - whether this channel is used to transmit or receive data, selected by setting a **rmt_mode** members to one of the values from *rmt_mode_t*.
- What is the **pin number** to transmit or receive RMT signals, selected by setting **gpio_num**.
- How many **memory blocks** will be used by the channel, set with **mem_block_num**.
- A **clock divider**, that will determine the range of pulse length generated by the RMT transmitter or discriminated by the receiver. Selected by setting **clk_div** to a value within [1 .. 255] range. The RMT source clock is typically APB CLK, 80Mhz by default.

注解: The period of a square wave after the clock divider is called a 'tick'. The length of the pulses generated by the RMT transmitter or discriminated by the receiver is configured in number of 'ticks'.

There are also couple of specific parameters that should be set up depending if selected channel is configured in *Transmit Mode* or *Receive Mode*:

Transmit Mode

When configuring channel in transmit mode, set **tx_config** and the following members of *rmt_tx_config_t*:

- Transmit the currently configured data items in a loop - **loop_en**
- Enable the RMT carrier signal - **carrier_en**
- Frequency of the carrier in Hz - **carrier_freq_hz**

- Duty cycle of the carrier signal in percent (%) - **carrier_duty_percent**
- Level of the RMT output, when the carrier is applied - **carrier_level**
- Enable the RMT output if idle - **idle_output_en**
- Set the signal level on the RMT output if idle - **idle_level**

Receive Mode

In receive mode, set **rx_config** and the following members of *rmt_rx_config_t*:

- Enable a filter on the input of the RMT receiver - **filter_en**
- A threshold of the filter, set in the number of ticks - **filter_ticks_thresh**. Pulses shorter than this setting will be filtered out. Note, that the range of entered tick values is [0..255].
- A pulse length threshold that will turn the RMT receiver idle, set in number of ticks - **idle_threshold**. The receiver will ignore pulses longer than this setting.

Finalize Configuration

Once the *rmt_config_t* structure is populated with parameters, it should be then invoked with *rmt_config()* to make the configuration effective.

The last configuration step is installation of the driver in memory by calling *rmt_driver_install()*. If `rx_buf_size` parameter of this function is > 0, then a ring buffer for incoming data will be allocated. A default ISR handler will be installed, see a note in *Use Interrupts*.

Now, depending on how the channel is configured, we are ready to either *Transmit Data* or *Receive Data*. This is described in next two sections.

Transmit Data

Before being able to transmit some RMT pulses, we need to define the pulse pattern. The minimum pattern recognized by the RMT controller, later called an ‘item’, is provided in a structure `rmt_item32_t`, see soc/esp32/include/soc/rmt_caps.h. Each item consists of two pairs of two values. The first value in a pair describes the signal duration in ticks and is 15 bits long, the second provides the signal level (high or low) and is contained in a single bit. A block of couple of items and the structure of an item is presented below.

For a simple example how to define a block of items see peripherals/rmt/morse_code.

The items are provided to the RMT controller by calling function *rmt_write_items()*. This function also automatically triggers start of transmission. It may be called to wait for transmission completion or exit just after transmission start. In such case you can wait for the transmission end by calling *rmt_wait_tx_done()*. This function does not limit the number of data items to transmit. It is using an interrupt to successively copy the new data chunks to RMT’s internal memory as previously provided data are sent out.

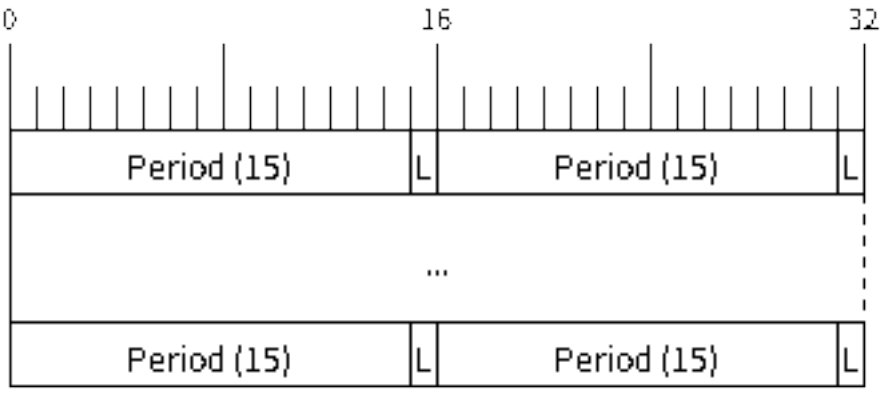


图 21: Structure of RMT items (L - signal level)

Another way to provide data for transmission is by calling *rmt_fill_tx_items()*. In this case transmission is not started automatically. To control the transmission process use *rmt_tx_start()* and *rmt_tx_stop()*. The number of items to sent is restricted by the size of memory blocks allocated in the RMT controller's internal memory, see *rmt_set_mem_block_num()*.

Receive Data

Before starting the receiver we need some storage for incoming items. The RMT controller has 512 x 32-bits of internal RAM shared between all eight channels. In typical scenarios it is not enough as an ultimate storage for all incoming (and outgoing) items. Therefore this API supports retrieval of incoming items on the fly to save them in a ring buffer of a size defined by the user. The size is provided when calling *rmt_driver_install()* discussed above. To get a handle to this buffer call *rmt_get_ringbuf_handle()*.

With the above steps complete we can start the receiver by calling *rmt_rx_start()* and then move to checking what's inside the buffer. To do so, you can use common FreeRTOS functions that interact with the ring buffer. Please see an example how to do it in peripherals/rmt/ir_protocols.

To stop the receiver, call *rmt_rx_stop()*.

Change Operation Parameters

Previously described function *rmt_config()* provides a convenient way to set several configuration parameters in one shot. This is usually done on application start. Then, when the application is running, the API provides an alternate way to update individual parameters by calling dedicated functions. Each function refers to the specific RMT channel provided as the first input parameter. Most of the functions have *_get_* counterpart to read back the currently configured value.

Parameters Common to Transmit and Receive Mode

- Selection of a GPIO pin number on the input or output of the RMT - *rmt_set_pin()*
- Number of memory blocks allocated for the incoming or outgoing data - *rmt_set_mem_pd()*
- Setting of the clock divider - *rmt_set_clk_div()*
- Selection of the clock source, note that currently one clock source is supported, the APB clock which is 80Mhz - *rmt_set_source_clk()*

Transmit Mode Parameters

- Enable or disable the loop back mode for the transmitter - *rmt_set_tx_loop_mode()*
- Binary level on the output to apply the carrier - *rmt_set_tx_carrier()*, selected from *rmt_carrier_level_t*
- Determines the binary level on the output when transmitter is idle - *rmt_set_idle_level()*, selected from *rmt_idle_level_t*

Receive Mode Parameters

- The filter setting - *rmt_set_rx_filter()*
- The receiver threshold setting - *rmt_set_rx_idle_thresh()*
- Whether the transmitter or receiver is entitled to access RMT' s memory - *rmt_set_memory_owner()*, selection is from *rmt_mem_owner_t*.

Use Interrupts

Registering of an interrupt handler for the RMT controller is done be calling *rmt_isr_register()*.

注解: When calling *rmt_driver_install()* to use the system RMT driver, a default ISR is being installed. In such a case you cannot register a generic ISR handler with *rmt_isr_register()*.

The RMT controller triggers interrupts on four specific events describes below. To enable interrupts on these events, the following functions are provided:

- The RMT receiver has finished receiving a signal - *rmt_set_rx_intr_en()*
- The RMT transmitter has finished transmitting the signal - *rmt_set_tx_intr_en()*
- The number of events the transmitter has sent matches a threshold value *rmt_set_tx_thr_intr_en()*
- Ownership to the RMT memory block has been violated - *rmt_set_err_intr_en()*

Setting or clearing an interrupt enable mask for specific channels and events may be also done by calling *rmt_set_intr_enable_mask()* or *rmt_clr_intr_enable_mask()*.

When servicing an interrupt within an ISR, the interrupt need to explicitly cleared. To do so, set specific bits described as `RMT.int_clr.val.chN_event_name` and defined as a `volatile struct` in soc/esp32/include/soc/rmt_struct.h, where N is the RMT channel number [0, 7] and the `event_name` is one of four events described above.

If you do not need an ISR anymore, you can deregister it by calling a function *rmt_isr_deregister()*.

Uninstall Driver

If the RMT driver has been installed with *rmt_driver_install()* for some specific period of time and then not required, the driver may be removed to free allocated resources by calling *rmt_driver_uninstall()*.

Application Examples

- A simple RMT TX example: peripherals/rmt/morse_code.
- Another RMT TX example, specific to drive a common RGB LED strip: peripherals/rmt/led_strip.
- NEC remote control TX and RX example: peripherals/rmt/ir_protocols.

API Reference

Header File

- driver/include/driver/rmt.h

Functions

esp_err_t **rmt_set_clk_div**(*rmt_channel_t channel*, uint8_t *div_cnt*)

Set RMT clock divider, channel clock is divided from source clock.

Return

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success

Parameters

- `channel`: RMT channel
- `div_cnt`: RMT counter clock divider

esp_err_t **rmt_get_clk_div**(*rmt_channel_t channel*, uint8_t **div_cnt*)

Get RMT clock divider, channel clock is divided from source clock.

Return

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success

Parameters

- `channel`: RMT channel
- `div_cnt`: pointer to accept RMT counter divider

esp_err_t **rmt_set_rx_idle_thresh**(*rmt_channel_t channel*, uint16_t *thresh*)

Set RMT RX idle threshold value.

In receive mode, when no edge is detected on the input signal for longer than idle_thres channel clock cycles, the receive process is finished.

Return

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success

Parameters

- `channel`: RMT channel
- `thresh`: RMT RX idle threshold

esp_err_t **rmt_get_rx_idle_thresh**(*rmt_channel_t channel*, uint16_t **thresh*)

Get RMT idle threshold value.

In receive mode, when no edge is detected on the input signal for longer than idle_thres channel clock cycles, the receive process is finished.

Return

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success

Parameters

- `channel`: RMT channel
- `thresh`: pointer to accept RMT RX idle threshold value

esp_err_t **rmt_set_mem_block_num**(*rmt_channel_t channel*, uint8_t *rmt_mem_num*)

Set RMT memory block number for RMT channel.

This function is used to configure the amount of memory blocks allocated to channel n The 8 channels share a 512x32-bit RAM block which can be read and written by the processor cores over the APB bus, as well as read by the transmitters and written by the receivers.

The RAM address range for channel n is start_addr_CHn to end_addr_CHn, which are defined by: Memory block start address is RMT_CHANNEL_MEM(n) (in soc/rmt_reg.h), that is, start_addr_chn = RMT base address + 0x800 + 64 * 4 * n, and end_addr_chn = RMT base address + 0x800 + 64 * 4 * n + 64 * 4 * RMT_MEM_SIZE_CHn mod 512 * 4

Note If memory block number of one channel is set to a value greater than 1, this channel will occupy the memory block of the next channel. Channel 0 can use at most 8 blocks of memory, accordingly channel 7 can only use one memory block.

Return

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success

Parameters

- `channel`: RMT channel
- `rmt_mem_num`: RMT RX memory block number, one block has 64 * 32 bits.

esp_err_t **rmt_get_mem_block_num**(*rmt_channel_t channel*, uint8_t **rmt_mem_num*)

Get RMT memory block number.

Return

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success

Parameters

- `channel`: RMT channel
- `rmt_mem_num`: Pointer to accept RMT RX memory block number

esp_err_t **rmt_set_tx_carrier**(*rmt_channel_t channel*, bool *carrier_en*, uint16_t *high_level*, uint16_t *low_level*, *rmt_carrier_level_t carrier_level*)

Configure RMT carrier for TX signal.

Set different values for carrier_high and carrier_low to set different frequency of carrier. The unit of carrier_high/low is the source clock tick, not the divided channel counter clock.

Return

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success

Parameters

- `channel`: RMT channel
- `carrier_en`: Whether to enable output carrier.
- `high_level`: High level duration of carrier
- `low_level`: Low level duration of carrier.
- `carrier_level`: Configure the way carrier wave is modulated for channel.
  - 1’ b1:transmit on low output level
  - 1’ b0:transmit on high output level

esp_err_t **rmt_set_mem_pd**(*rmt_channel_t channel*, bool *pd_en*)

Set RMT memory in low power mode.

Reduce power consumed by memory. 1:memory is in low power state.

Return

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success

Parameters

- `channel`: RMT channel
- `pd_en`: RMT memory low power enable.

esp_err_t **rmt_get_mem_pd**(*rmt_channel_t channel*, bool **pd_en*)

Get RMT memory low power mode.

Return

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success

Parameters

- `channel`: RMT channel
- `pd_en`: Pointer to accept RMT memory low power mode.

esp_err_t **rmt_tx_start**(*rmt_channel_t channel*, bool *tx_idx_rst*)

Set RMT start sending data from memory.

Return

- ESP_ERR_INVALID_ARG Parameter error

- ESP_OK Success

Parameters

- `channel`: RMT channel
- `tx_idx_rst`: Set true to reset memory index for TX. Otherwise, transmitter will continue sending from the last index in memory.

esp_err_t **rmt_tx_stop**(*rmt_channel_t channel*)

Set RMT stop sending.

Return

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success

Parameters

- `channel`: RMT channel

esp_err_t **rmt_rx_start**(*rmt_channel_t channel*, bool *rx_idx_rst*)

Set RMT start receiving data.

Return

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success

Parameters

- `channel`: RMT channel
- `rx_idx_rst`: Set true to reset memory index for receiver. Otherwise, receiver will continue receiving data to the last index in memory.

esp_err_t **rmt_rx_stop**(*rmt_channel_t channel*)

Set RMT stop receiving data.

Return

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success

Parameters

- `channel`: RMT channel

esp_err_t **rmt_memory_rw_rst**(*rmt_channel_t channel*)

Reset RMT TX/RX memory index.

Return

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success

Parameters

- `channel`: RMT channel

esp_err_t **rmt_set_memory_owner**(*rmt_channel_t channel*, *rmt_mem_owner_t owner*)

Set RMT memory owner.

Return

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success

Parameters

- `channel`: RMT channel
- `owner`: To set when the transmitter or receiver can process the memory of channel.

esp_err_t **rmt_get_memory_owner**(*rmt_channel_t channel*, *rmt_mem_owner_t* **owner*)

Get RMT memory owner.

Return

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success

Parameters

- `channel`: RMT channel
- `owner`: Pointer to get memory owner.

esp_err_t **rmt_set_tx_loop_mode**(*rmt_channel_t channel*, bool *loop_en*)

Set RMT tx loop mode.

Return

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success

Parameters

- `channel`: RMT channel
- `loop_en`: Enable RMT transmitter loop sending mode. If set true, transmitter will continue sending from the first data to the last data in channel over and over again in a loop.

esp_err_t **rmt_get_tx_loop_mode**(*rmt_channel_t channel*, bool **loop_en*)

Get RMT tx loop mode.

Return

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success

Parameters

- `channel`: RMT channel
- `loop_en`: Pointer to accept RMT transmitter loop sending mode.

esp_err_t **rmt_set_rx_filter**(*rmt_channel_t channel*, bool *rx_filter_en*, uint8_t *thresh*)

Set RMT RX filter.

In receive mode, channel will ignore input pulse when the pulse width is smaller than threshold. Counted in source clock, not divided counter clock.

Return

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success

Parameters

- `channel`: RMT channel
- `rx_filter_en`: To enable RMT receiver filter.
- `thresh`: Threshold of pulse width for receiver.

esp_err_t **rmt_set_source_clk**(*rmt_channel_t channel*, *rmt_source_clk_t base_clk*)

Set RMT source clock.

RMT module has two clock sources:

1. APB clock which is 80Mhz
2. REF tick clock, which would be 1Mhz (not supported in this version).

Return

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success

Parameters

- `channel`: RMT channel
- `base_clk`: To choose source clock for RMT module.

esp_err_t **rmt_get_source_clk**(*rmt_channel_t channel*, *rmt_source_clk_t* **src_clk*)

Get RMT source clock.

RMT module has two clock sources:

1. APB clock which is 80Mhz
2. REF tick clock, which would be 1Mhz (not supported in this version).

Return

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success

Parameters

- `channel`: RMT channel
- `src_clk`: Pointer to accept source clock for RMT module.

esp_err_t **rmt_set_idle_level**(*rmt_channel_t channel*, bool *idle_out_en*, *rmt_idle_level_t level*)

Set RMT idle output level for transmitter.

Return

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success

Parameters

- `channel`: RMT channel
- `idle_out_en`: To enable idle level output.
- `level`: To set the output signal's level for channel in idle state.

esp_err_t **rmt_get_idle_level**(*rmt_channel_t channel*, bool **idle_out_en*, *rmt_idle_level_t* **level*)

Get RMT idle output level for transmitter.

Return

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success

Parameters

- `channel`: RMT channel
- `idle_out_en`: Pointer to accept value of enable idle.
- `level`: Pointer to accept value of output signal's level in idle state for specified channel.

esp_err_t **rmt_get_status**(*rmt_channel_t channel*, uint32_t **status*)

Get RMT status.

Return

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success

Parameters

- `channel`: RMT channel
- `status`: Pointer to accept channel status. Please refer to RMT_CHnSTATUS_REG(n=0~7) in `rmt_reg.h` for more details of each field.

void **rmt_set_intr_enable_mask**(uint32_t *mask*)

Set mask value to RMT interrupt enable register.

Parameters

- `mask`: Bit mask to set to the register

void **rmt_clr_intr_enable_mask**(uint32_t *mask*)

Clear mask value to RMT interrupt enable register.

Parameters

- `mask`: Bit mask to clear the register

esp_err_t **rmt_set_rx_intr_en**(*rmt_channel_t channel*, bool *en*)

Set RMT RX interrupt enable.

Return

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success

Parameters

- `channel`: RMT channel
- `en`: enable or disable RX interrupt.

esp_err_t **rmt_set_err_intr_en**(*rmt_channel_t channel*, bool *en*)

Set RMT RX error interrupt enable.

Return

- ESP_ERR_INVALID_ARG Parameter error

- ESP_OK Success

Parameters

- `channel`: RMT channel
- `en`: enable or disable RX err interrupt.

esp_err_t **rmt_set_tx_intr_en**(*rmt_channel_t channel*, bool *en*)

Set RMT TX interrupt enable.

Return

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success

Parameters

- `channel`: RMT channel
- `en`: enable or disable TX interrupt.

esp_err_t **rmt_set_tx_thr_intr_en**(*rmt_channel_t channel*, bool *en*, uint16_t *evt_thresh*)

Set RMT TX threshold event interrupt enable.

An interrupt will be triggered when the number of transmitted items reaches the threshold value

Return

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success

Parameters

- `channel`: RMT channel
- `en`: enable or disable TX event interrupt.
- `evt_thresh`: RMT event interrupt threshold value

esp_err_t **rmt_set_pin**(*rmt_channel_t channel*, *rmt_mode_t mode*, *gpio_num_t gpio_num*)

Set RMT pin.

Return

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success

Parameters

- `channel`: RMT channel

- `mode`: TX or RX mode for RMT
- `gpio_num`: GPIO number to transmit or receive the signal.

esp_err_t **rmt_config**(**const** *rmt_config_t* **rmt_param*)

Configure RMT parameters.

Return

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success

Parameters

- `rmt_param`: RMT parameter struct

esp_err_t **rmt_isr_register**(void (**fn*))void *

, void **arg*, int *intr_alloc_flags*, *rmt_isr_handle_t* **handle*Register RMT interrupt handler, the handler is an ISR.

The handler will be attached to the same CPU core that this function is running on.

Note If you already called rmt_driver_install to use system RMT driver, please do not register ISR handler again.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Function pointer error.
- ESP_FAIL System driver installed, can not register ISR handler for RMT

Parameters

- `fn`: Interrupt handler function.
- `arg`: Parameter for the handler function
- `intr_alloc_flags`: Flags used to allocate the interrupt. One or multiple (ORred) ESP_INTR_FLAG_* values. See esp_intr_alloc.h for more info.
- `handle`: If non-zero, a handle to later clean up the ISR gets stored here.

esp_err_t **rmt_isr_deregister**(*rmt_isr_handle_t handle*)

Deregister previously registered RMT interrupt handler.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Handle invalid

Parameters

- `handle`: Handle obtained from rmt_isr_register

esp_err_t **rmt_fill_tx_items**(*rmt_channel_t channel*, const rmt_item32_t **item*, uint16_t *item_num*, uint16_t *mem_offset*)

Fill memory data of channel with given RMT items.

Return

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success

Parameters

- `channel`: RMT channel
- `item`: Pointer of items.
- `item_num`: RMT sending items number.
- `mem_offset`: Index offset of memory.

esp_err_t **rmt_driver_install**(*rmt_channel_t channel*, size_t *rx_buf_size*, int *intr_alloc_flags*)

Initialize RMT driver.

Return

- ESP_ERR_INVALID_STATE Driver is already installed, call rmt_driver_uninstall first.
- ESP_ERR_NO_MEM Memory allocation failure
- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success

Parameters

- `channel`: RMT channel
- `rx_buf_size`: Size of RMT RX ringbuffer. Can be 0 if the RX ringbuffer is not used.
- `intr_alloc_flags`: Flags for the RMT driver interrupt handler. Pass 0 for default flags. See esp_intr_alloc.h for details. If ESP_INTR_FLAG_IRAM is used, please do not use the memory allocated from psram when calling rmt_write_items.

esp_err_t **rmt_driver_uninstall**(*rmt_channel_t channel*)

Uninstall RMT driver.

Return

- ESP_ERR_INVALID_ARG Parameter error

- ESP_OK Success

Parameters

- `channel`: RMT channel

esp_err_t **rmt_get_channel_status**(*rmt_channel_status_result_t* **channel_status*)

Get the current status of eight channels.

Note Do not call this function if it is possible that `rmt_driver_uninstall` will be called at the same time.

Return

- ESP_ERR_INVALID_ARG Parameter is NULL
- ESP_OK Success

Parameters

- `channel_status`: store the current status of each channel

esp_err_t **rmt_get_counter_clock**(*rmt_channel_t channel*, uint32_t **clock_hz*)

Get speed of channel' s internal counter clock.

Return

- ESP_ERR_INVALID_ARG Parameter is NULL
- ESP_OK Success

Parameters

- `channel`: RMT channel
- `clock_hz`: counter clock speed, in hz

esp_err_t **rmt_write_items**(*rmt_channel_t channel*, **const** rmt_item32_t **rmt_item*, int *item_num*, bool *wait_tx_done*)

RMT send waveform from rmt_item array.

This API allows user to send waveform with any length.

Note This function will not copy data, instead, it will point to the original items, and send the waveform items. If wait_tx_done is set to true, this function will block and will not return until all items have been sent out. If wait_tx_done is set to false, this function will return immediately, and the driver interrupt will continue sending the items. We must make sure the item data will not be damaged when the driver is still sending items in driver interrupt.

Return

- ESP_ERR_INVALID_ARG Parameter error

- ESP_OK Success

Parameters

- `channel`: RMT channel
- `rmt_item`: head point of RMT items array. If ESP_INTR_FLAG_IRAM is used, please do not use the memory allocated from psram when calling rmt_write_items.
- `item_num`: RMT data item number.
- `wait_tx_done`:
  - If set 1, it will block the task and wait for sending done.
  - If set 0, it will not wait and return immediately.

esp_err_t **rmt_wait_tx_done**(*rmt_channel_t channel*, TickType_t *wait_time*)

Wait RMT TX finished.

Return

- ESP_OK RMT Tx done successfully
- ESP_ERR_TIMEOUT Exceeded the ‘wait_time’ given
- ESP_ERR_INVALID_ARG Parameter error
- ESP_FAIL Driver not installed

Parameters

- `channel`: RMT channel
- `wait_time`: Maximum time in ticks to wait for transmission to be complete. If set 0, return immediately with ESP_ERR_TIMEOUT if TX is busy (polling).

esp_err_t **rmt_get_ringbuf_handle**(*rmt_channel_t channel*, *RingbufHandle_t* **buf_handle*)

Get ringbuffer from RMT.

Users can get the RMT RX ringbuffer handle, and process the RX data.

Return

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success

Parameters

- `channel`: RMT channel
- `buf_handle`: Pointer to buffer handle to accept RX ringbuffer handle.

esp_err_t **rmt_translator_init**(*rmt_channel_t channel*, *sample_to_rmt_t fn*)

Init rmt translator and register user callback. The callback will convert the raw data that needs to be sent to rmt format. If a channel is initialized more than once, tha user callback will be replaced by the later.

Return

- ESP_FAIL Init fail.
- ESP_OK Init success.

Parameters

- `channel`: RMT channel .
- `fn`: Point to the data conversion function.

esp_err_t **rmt_write_sample**(*rmt_channel_t channel*, **const** uint8_t **src*, size_t *src_size*, bool *wait_tx_done*)

Translate uint8_t type of data into rmt format and send it out. Requires rmt_translator_init to init the translator first.

Return

- ESP_FAIL Send fail
- ESP_OK Send success

Parameters

- `channel`: RMT channel .
- `src`: Pointer to the raw data.
- `src_size`: The size of the raw data.
- `wait_tx_done`: Set true to wait all data send done.

rmt_tx_end_callback_t **rmt_register_tx_end_callback**(*rmt_tx_end_fn_t function*, void **arg*)

Registers a callback that will be called when transmission ends.

Called by rmt_driver_isr_default in interrupt context.

Note Requires rmt_driver_install to install the default ISR handler.

Return the previous callback settings (members will be set to NULL if there was none)

Parameters

- `function`: Function to be called from the default interrupt handler or NULL.
- `arg`: Argument which will be provided to the callback when it is called.

Structures

struct rmt_tx_config_t

Data struct of RMT TX configure parameters.

Public Members

uint32_t **carrier_freq_hz**

RMT carrier frequency

rmt_carrier_level_t **carrier_level**

Level of the RMT output, when the carrier is applied

rmt_idle_level_t **idle_level**

RMT idle level

uint8_t **carrier_duty_percent**

RMT carrier duty (%)

bool **carrier_en**

RMT carrier enable

bool **loop_en**

Enable sending RMT items in a loop

bool **idle_output_en**

RMT idle level output enable

struct rmt_rx_config_t

Data struct of RMT RX configure parameters.

Public Members

uint16_t **idle_threshold**

RMT RX idle threshold

uint8_t **filter_ticks_thresh**

RMT filter tick number

bool **filter_en**

RMT receiver filter enable

struct rmt_config_t

Data struct of RMT configure parameters.

Public Members

rmt_mode_t **rmt_mode**
RMT mode: transmitter or receiver

rmt_channel_t **channel**
RMT channel

gpio_num_t **gpio_num**
RMT GPIO number

uint8_t **clk_div**
RMT channel counter divider

uint8_t **mem_block_num**
RMT memory block number

rmt_tx_config_t **tx_config**
RMT TX parameter

rmt_rx_config_t **rx_config**
RMT RX parameter

struct rmt_tx_end_callback_t
Structure encapsulating a RMT TX end callback.

Public Members

rmt_tx_end_fn_t **function**
Function which is called on RMT TX end

void ***arg**
Optional argument passed to function

Macros

RMT_MEM_ITEM_NUM
Define memory space of each RMT channel (in words = 4 bytes)

RMT_DEFAULT_CONFIG_TX(gpio, channel_id)
Default configuration for Tx channel.

RMT_DEFAULT_CONFIG_RX(gpio, channel_id)
Default configuration for RX channel.

Type Definitions

typedef *intr_handle_t* **rmt_isr_handle_t**

RMT interrupt handle.

typedef void (***rmt_tx_end_fn_t**)(*rmt_channel_t* channel, void *arg)

Type of RMT Tx End callback function.

typedef void (***sample_to_rmt_t**)(**const** void *src, rmt_item32_t *dest, size_t src_size, size_t wanted_num, size_t *translated_size, size_t *item_num)

User callback function to convert uint8_t type data to rmt format(rmt_item32_t).

This function may be called from an ISR, so, the code should be short and efficient.

Note In fact, item_num should be a multiple of translated_size, e.g. : When we convert each byte of uint8_t type data to rmt format data, the relation between item_num and translated_size should be `item_num = translated_size*8`.

Parameters

- `src`: Pointer to the buffer storing the raw data that needs to be converted to rmt format.
- `dest`: Pointer to the buffer storing the rmt format data.
- `src_size`: The raw data size.
- `wanted_num`: The number of rmt format data that wanted to get.
- `translated_size`: The size of the raw data that has been converted to rmt format, it should return 0 if no data is converted in user callback.
- `item_num`: The number of the rmt format data that actually converted to, it can be less than wanted_num if there is not enough raw data, but cannot exceed wanted_num. it should return 0 if no data was converted.

Header File

- soc/include/hal/rmt_types.h

Structures

struct **rmt_channel_status_result_t**

Data struct of RMT channel status.

Public Members

rmt_channel_status_t **status**[**RMT_CHANNEL_MAX**]

Store the current status of each channel

Type Definitions

typedef rmt_channel_id_t **rmt_channel_t**

RMT Channel Type.

Enumerations

enum rmt_mem_owner_t

RMT Internal Memory Owner.

Values:

RMT_MEM_OWNER_TX

RMT RX mode, RMT transmitter owns the memory block

RMT_MEM_OWNER_RX

RMT RX mode, RMT receiver owns the memory block

RMT_MEM_OWNER_MAX

enum rmt_source_clk_t

Clock Source of RMT Channel.

Values:

RMT_BASECLK_REF

RMT source clock system reference tick, 1MHz by default (not supported in this version)

RMT_BASECLK_APB

RMT source clock is APB CLK, 80Mhz by default

RMT_BASECLK_MAX

enum rmt_data_mode_t

RMT Data Mode.

Note We highly recommended to use MEM mode not FIFO mode since there will be some gotcha in FIFO mode.

Values:

RMT_DATA_MODE_FIFO

RMT_DATA_MODE_MEM

RMT_DATA_MODE_MAX

enum rmt_mode_t

RMT Channel Working Mode (TX or RX)

Values:

RMT_MODE_TX
RMT TX mode

RMT_MODE_RX
RMT RX mode

RMT_MODE_MAX

enum rmt_idle_level_t
RMT Idle Level.

Values:

RMT_IDLE_LEVEL_LOW
RMT TX idle level: low Level

RMT_IDLE_LEVEL_HIGH
RMT TX idle level: high Level

RMT_IDLE_LEVEL_MAX

enum rmt_carrier_level_t
RMT Carrier Level.

Values:

RMT_CARRIER_LEVEL_LOW
RMT carrier wave is modulated for low Level output

RMT_CARRIER_LEVEL_HIGH
RMT carrier wave is modulated for high Level output

RMT_CARRIER_LEVEL_MAX

enum rmt_channel_status_t
RMT Channel Status.

Values:

RMT_CHANNEL_UNINIT
RMT channel uninitialized

RMT_CHANNEL_IDLE
RMT channel status idle

RMT_CHANNEL_BUSY
RMT channel status busy

2.3.11 SDMMC Host Driver

Overview

ESP32’ s SDMMC host peripheral has two slots:

- Slot 0 (*SDMMC_HOST_SLOT_0*) is an 8-bit slot. It uses `HS1_*` signals in the PIN MUX.
- Slot 1 (*SDMMC_HOST_SLOT_1*) is a 4-bit slot. It uses `HS2_*` signals in the PIN MUX.

Pin mappings of these slots are given in the table below.

Signal	Slot 0	Slot 1
CMD	GPIO11	GPIO15
CLK	GPIO6	GPIO14
D0	GPIO7	GPIO2
D1	GPIO8	GPIO4
D2	GPIO9	GPIO12
D3	GPIO10	GPIO13
D4	GPIO16	
D5	GPIO17	
D6	GPIO5	
D7	GPIO18	
CD	any input via GPIO matrix	
WP	any input via GPIO matrix	

The Card Detect and Write Protect signals can be routed to arbitrary pins using the GPIO matrix. To reserve the pins, set the `gpio_cd` and `gpio_wp` members of the *sdmmc_slot_config_t* structure before calling *sdmmc_host_init_slot()*. Please note that it is not advised to specify a Card Detect pin when working with SDIO cards, because the card detect signal in ESP32 can also trigger SDIO slave interrupt.

警告: Pins used by Slot 0 (`HS1_*`) are also used to connect the SPI flash chip in ESP32-WROOM and ESP32-WROVER modules. These pins cannot be shared between an SD card and SPI flash. If you need to use Slot 0, connect SPI flash to different pins and set eFuses accordingly.

Supported Speed Modes

SDMMC Host driver supports the following speed modes:

- Default Speed (20 MHz), 4-line/1-line (with SD cards), and 8-line (with 3.3 V eMMC)
- High Speed (40 MHz), 4-line/1-line (with SD cards), and 8-line (with 3.3 V eMMC)
- High Speed DDR (40 MHz), 4-line (with 3.3 V eMMC)

Speed modes not supported at present:

- High Speed DDR mode, 8-line eMMC
- UHS-I 1.8 V modes, 4-line SD cards

Using the SDMMC Host Driver

Of all the functions listed below, only the following ones will be used directly by most applications:

- *sdmmc_host_init()*
- *sdmmc_host_init_slot()*
- *sdmmc_host_deinit()*

Other functions, such as the ones given below, will be called by the SD/MMC protocol layer via function pointers in the *sdmmc_host_t* structure:

- *sdmmc_host_set_bus_width()*
- *sdmmc_host_set_card_clk()*
- *sdmmc_host_do_transaction()*

Configuring Bus Width and Frequency

With the default initializers for *sdmmc_host_t* and *sdmmc_slot_config_t* (*SDMMC_HOST_DEFAULT* and *SDMMC_SLOT_CONFIG_DEFAULT*), SDMMC Host driver will attempt to use the widest bus supported by the card (4 lines for SD, 8 lines for eMMC) and the frequency of 20 MHz.

In the designs where communication at 40 MHz frequency can be achieved, it is possible to increase the bus frequency by changing the `max_freq_khz` field of *sdmmc_host_t*:

```
sdmmc_host_t host = SDMMC_HOST_DEFAULT();
host.max_freq_khz = SDMMC_FREQ_HIGHSPEED;
```

To configure the bus width, set the `width` field of *sdmmc_slot_config_t*. For example, to set 1-line mode:

```
sdmmc_slot_config_t slot = SDMMC_SLOT_CONFIG_DEFAULT();
slot.width = 1;
```

See also

See *SD/SDIO/MMC Driver* for the higher level driver which implements the protocol layer.

See *SD SPI Host Driver* for a similar driver which uses the SPI controller and is limited to SD protocol' s SPI mode.

See *SD Pull-up Requirements* for pullup support and compatibilities of modules and development kits.

API Reference

Header File

- driver/include/driver/sdmmc_host.h

Functions

esp_err_t **sdmmc_host_init**(void)

Initialize SDMMC host peripheral.

Note This function is not thread safe

Return

- ESP_OK on success
- ESP_ERR_INVALID_STATE if sdmmc_host_init was already called
- ESP_ERR_NO_MEM if memory can not be allocated

esp_err_t **sdmmc_host_init_slot**(int *slot*, **const** *sdmmc_slot_config_t* **slot_config*)

Initialize given slot of SDMMC peripheral.

On the ESP32, SDMMC peripheral has two slots:

- Slot 0: 8-bit wide, maps to HS1_* signals in PIN MUX
- Slot 1: 4-bit wide, maps to HS2_* signals in PIN MUX

Card detect and write protect signals can be routed to arbitrary GPIOs using GPIO matrix.

Note This function is not thread safe

Return

- ESP_OK on success
- ESP_ERR_INVALID_STATE if host has not been initialized using sdmmc_host_init

Parameters

- **slot**: slot number (SDMMC_HOST_SLOT_0 or SDMMC_HOST_SLOT_1)
- **slot_config**: additional configuration for the slot

esp_err_t **sdmmc_host_set_bus_width**(int *slot*, size_t *width*)

Select bus width to be used for data transfer.

SD/MMC card must be initialized prior to this command, and a command to set bus width has to be sent to the card (e.g. SD_APP_SET_BUS_WIDTH)

Note This function is not thread safe

Return

- ESP_OK on success
- ESP_ERR_INVALID_ARG if slot number or width is not valid

Parameters

- `slot`: slot number (SDMMC_HOST_SLOT_0 or SDMMC_HOST_SLOT_1)
- `width`: bus width (1, 4, or 8 for slot 0; 1 or 4 for slot 1)

size_t **sdmmc_host_get_slot_width**(int *slot*)

Get bus width configured in `sdmmc_host_init_slot` to be used for data transfer.

Return configured bus width of the specified slot.

Parameters

- `slot`: slot number (SDMMC_HOST_SLOT_0 or SDMMC_HOST_SLOT_1)

esp_err_t **sdmmc_host_set_card_clk**(int *slot*, uint32_t *freq_khz*)

Set card clock frequency.

Currently only integer fractions of 40MHz clock can be used. For High Speed cards, 40MHz can be used. For Default Speed cards, 20MHz can be used.

Note This function is not thread safe

Return

- ESP_OK on success
- other error codes may be returned in the future

Parameters

- `slot`: slot number (SDMMC_HOST_SLOT_0 or SDMMC_HOST_SLOT_1)
- `freq_khz`: card clock frequency, in kHz

esp_err_t **sdmmc_host_set_bus_ddr_mode**(int *slot*, bool *ddr_enabled*)

Enable or disable DDR mode of SD interface.

Return

- ESP_OK on success
- ESP_ERR_NOT_SUPPORTED if DDR mode is not supported on this slot

Parameters

- `slot`: slot number (SDMMC_HOST_SLOT_0 or SDMMC_HOST_SLOT_1)

- `ddr_enabled`: enable or disable DDR mode

esp_err_t **sdmmc_host_do_transaction**(int *slot*, *sdmmc_command_t* **cmdinfo*)

Send command to the card and get response.

This function returns when command is sent and response is received, or data is transferred, or timeout occurs.

Note This function is not thread safe w.r.t. init/deinit functions, and bus width/clock speed configuration functions. Multiple tasks can call sdmmc_host_do_transaction as long as other sdmmc_host_* functions are not called.

Attention Data buffer passed in cmdinfo->data must be in DMA capable memory

Return

- ESP_OK on success
- ESP_ERR_TIMEOUT if response or data transfer has timed out
- ESP_ERR_INVALID_CRC if response or data transfer CRC check has failed
- ESP_ERR_INVALID_RESPONSE if the card has sent an invalid response
- ESP_ERR_INVALID_SIZE if the size of data transfer is not valid in SD protocol
- ESP_ERR_INVALID_ARG if the data buffer is not in DMA capable memory

Parameters

- `slot`: slot number (SDMMC_HOST_SLOT_0 or SDMMC_HOST_SLOT_1)
- `cmdinfo`: pointer to structure describing command and data to transfer

esp_err_t **sdmmc_host_io_int_enable**(int *slot*)

Enable IO interrupts.

This function configures the host to accept SDIO interrupts.

Return returns ESP_OK, other errors possible in the future

Parameters

- `slot`: slot number (SDMMC_HOST_SLOT_0 or SDMMC_HOST_SLOT_1)

esp_err_t **sdmmc_host_io_int_wait**(int *slot*, TickType_t *timeout_ticks*)

Block until an SDIO interrupt is received, or timeout occurs.

Return

- ESP_OK on success (interrupt received)
- ESP_ERR_TIMEOUT if the interrupt did not occur within timeout_ticks

Parameters

- `slot`: slot number (SDMMC_HOST_SLOT_0 or SDMMC_HOST_SLOT_1)
- `timeout_ticks`: number of RTOS ticks to wait for the interrupt

esp_err_t **sdmmc_host_deinit**(void)

Disable SDMMC host and release allocated resources.

Note This function is not thread safe

Return

- ESP_OK on success
- ESP_ERR_INVALID_STATE if sdmmc_host_init function has not been called

esp_err_t **sdmmc_host_pullup_en**(int *slot*, int *width*)

Enable the pull-ups of sd pins.

Note You should always place actual pullups on the lines instead of using this function. Internal pullup resistance are high and not sufficient, may cause instability in products. This is for debug or examples only.

Return

- ESP_OK: if success
- ESP_ERR_INVALID_ARG: if configured width larger than maximum the slot can support

Parameters

- `slot`: Slot to use, normally set it to 1.
- `width`: Bit width of your configuration, 1 or 4.

Structures

struct sdmmc_slot_config_t

Extra configuration for SDMMC peripheral slot

Public Members

gpio_num_t **gpio_cd**

GPIO number of card detect signal.

gpio_num_t **gpio_wp**

GPIO number of write protect signal.

uint8_t **width**

Bus width used by the slot (might be less than the max width supported)

uint32_t **flags**

Features used by this slot.

Macros

`SDMMC_HOST_SLOT_0`

SDMMC slot 0.

`SDMMC_HOST_SLOT_1`

SDMMC slot 1.

`SDMMC_HOST_DEFAULT()`

Default *sdmmc_host_t* structure initializer for SDMMC peripheral.

Uses SDMMC peripheral, with 4-bit mode enabled, and max frequency set to 20MHz

`SDMMC_SLOT_FLAG_INTERNAL_PULLUP`

Enable internal pullups on enabled pins. The internal pullups are insufficient however, please make sure external pullups are connected on the bus. This is for debug / example purpose only.

`SDMMC_SLOT_NO_CD`

indicates that card detect line is not used

`SDMMC_SLOT_NO_WP`

indicates that write protect line is not used

`SDMMC_SLOT_WIDTH_DEFAULT`

use the default width for the slot (8 for slot 0, 4 for slot 1)

`SDMMC_SLOT_CONFIG_DEFAULT()`

Macro defining default configuration of SDMMC host slot

2.3.12 SD SPI Host Driver

Overview

The SD SPI host driver allows using the SPI2 (HSPI) or SPI3 (VSPI) controller for communication with SD cards. This driver's naming pattern was adopted from the *SDMMC Host* driver due to their similarity. Likewise, the APIs of both drivers are also very similar.

The SD SPI host driver has the following modes:

- **SPI mode**: offers lower throughput but makes pin selection more flexible. With the help of the GPIO matrix, an SPI peripheral's signals can be routed to any ESP32 pin.

- **1-bit SD mode**: offers higher throughput but requires routing the signals through their dedicated IO_MUX pins only.

The SD SPI driver uses software-controlled CS signal.

Currently, the SD SPI driver cannot handle multi-threaded environments as does not support time-division multiplexing on the same SPI bus. It means that if your application needs to communicate with an SD card and other devices on the same SPI bus, the application itself must ensure that its different tasks do not try to access the SPI slaves at the same time.

How to Use

The state and configurations of the SD SPI host driver are stored in a *sdmmc_host_t* structure. This structure can be initialized using the *SDSPI_HOST_DEFAULT* macro.

The state and configurations of the SD slot are stored in a *sdmmc_slot_config_t* structure. Use the macro *SDSPI_SLOT_CONFIG_DEFAULT* to initialize the structure and to fill in the default pin mappings (SD mode pin mappings).

Only the following driver's API functions are normally used by most applications:

- *sdspi_host_init()*
- *sdspi_host_init_slot()*
- *sdspi_host_deinit()*

Other functions are mostly used by the protocol level SD/SDIO/MMC driver via function pointers in the *sdmmc_host_t* structure. For more details, see *the SD/SDIO/MMC Driver*.

注解: SD over SPI does not support speeds above *SDMMC_FREQ_DEFAULT* due to the limitations of the SPI driver.

API Reference

Header File

- driver/include/driver/sdspi_host.h

Functions

esp_err_t **sdspi_host_init**(void)

Initialize SD SPI driver.

Note This function is not thread safe

Return

- ESP_OK on success
- other error codes may be returned in future versions

esp_err_t **sdspi_host_init_slot**(int *slot*, **const** *sdspi_slot_config_t* **slot_config*)

Initialize SD SPI driver for the specific SPI controller.

Note This function is not thread safe

Note The SDIO over sdspi needs an extra interrupt line. Call `gpio_install_isr_service()` before this function.

Return

- ESP_OK on success
- ESP_ERR_INVALID_ARG if sdspi_init_slot has invalid arguments
- ESP_ERR_NO_MEM if memory can not be allocated
- other errors from the underlying spi_master and gpio drivers

Parameters

- `slot`: SPI controller to use (HSPI_HOST or VSPI_HOST)
- `slot_config`: pointer to slot configuration structure

esp_err_t **sdspi_host_do_transaction**(int *slot*, *sdmmc_command_t* **cmdinfo*)

Send command to the card and get response.

This function returns when command is sent and response is received, or data is transferred, or timeout occurs.

Note This function is not thread safe w.r.t. init/deinit functions, and bus width/clock speed configuration functions. Multiple tasks can call sdspi_host_do_transaction as long as other sdspi_host_* functions are not called.

Return

- ESP_OK on success
- ESP_ERR_TIMEOUT if response or data transfer has timed out
- ESP_ERR_INVALID_CRC if response or data transfer CRC check has failed
- ESP_ERR_INVALID_RESPONSE if the card has sent an invalid response

Parameters

- `slot`: SPI controller (HSPI_HOST or VSPI_HOST)
- `cmdinfo`: pointer to structure describing command and data to transfer

esp_err_t **sdspi_host_set_card_clk**(int *slot*, uint32_t *freq_khz*)

Set card clock frequency.

Currently only integer fractions of 40MHz clock can be used. For High Speed cards, 40MHz can be used. For Default Speed cards, 20MHz can be used.

Note This function is not thread safe

Return

- ESP_OK on success
- other error codes may be returned in the future

Parameters

- `slot`: SPI controller (HSPI_HOST or VSPI_HOST)
- `freq_khz`: card clock frequency, in kHz

esp_err_t **sdspi_host_deinit**(void)

Release resources allocated using sdspi_host_init.

Note This function is not thread safe

Return

- ESP_OK on success
- ESP_ERR_INVALID_STATE if sdspi_host_init function has not been called

esp_err_t **sdspi_host_io_int_enable**(int *slot*)

Enable SDIO interrupt.

Return

- ESP_OK on success

Parameters

- `slot`: SPI controller to use (HSPI_HOST or VSPI_HOST)

esp_err_t **sdspi_host_io_int_wait**(int *slot*, TickType_t *timeout_ticks*)

Wait for SDIO interrupt until timeout.

Return

- ESP_OK on success

Parameters

- `slot`: SPI controller to use (HSPI_HOST or VSPI_HOST)
- `timeout_ticks`: Ticks to wait before timeout.

Structures

struct sdspi_slot_config_t

Extra configuration for SPI host

Public Members

gpio_num_t **gpio_miso**

GPIO number of MISO signal.

gpio_num_t **gpio_mosi**

GPIO number of MOSI signal.

gpio_num_t **gpio_sck**

GPIO number of SCK signal.

gpio_num_t **gpio_cs**

GPIO number of CS signal.

gpio_num_t **gpio_cd**

GPIO number of card detect signal.

gpio_num_t **gpio_wp**

GPIO number of write protect signal.

gpio_num_t **gpio_int**

GPIO number of interrupt line (input) for SDIO card.

int **dma_channel**

DMA channel to be used by SPI driver (1 or 2)

Macros

SDSPI_HOST_DEFAULT()

Default *sdmmc_host_t* structure initializer for SD over SPI driver.

Uses SPI mode and max frequency set to 20MHz

'slot' can be set to one of HSPI_HOST, VSPI_HOST.

SDSPI_SLOT_NO_CD

indicates that card detect line is not used

SDSPI_SLOT_NO_WP

indicates that write protect line is not used

SDSPI_SLOT_NO_INT

indicates that interrupt line is not used

`SDSPI_SLOT_CONFIG_DEFAULT()`

Macro defining default configuration of SPI host

2.3.13 SDIO Card Slave Driver

Overview

The ESP32 SDIO Card peripherals (Host, Slave) shares two sets of pins as below table. The first set is usually occupied by SPI0 bus which is responsible for the SPI flash holding the code to run. This means SDIO slave driver can only runs on the second set of pins while SDIO host is not using it.

The SDIO slave can run under 3 modes: SPI, 1-bit SD and 4-bit SD modes, which is detected automatically by the hardware. According to the SDIO specification, CMD and DAT0-3 lines should be pulled up no matter in 1-bit, 4-bit or SPI mode.

Connections

Pin Name	Corresponding pins in SPI mode	Slot1	Slot2
		GPIO Number	
CLK	SCLK	6	14
CMD	MOSI	11	15
DAT0	MISO	7	2
DAT1	Interrupt	8	4
DAT2	N.C. (pullup)	9	12
DAT3	#CS	10	13

- 1-bit SD mode: Connect CLK, CMD, DAT0, DAT1 pins and the ground.
- 4-bit SD mode: Connect all pins and the ground.
- SPI mode: Connect SCLK, MOSI, MISO, Interrupt, #CS pins and the ground.

注解: Please check if CMD and DATA lines D0-D3 of the card are properly pulled up by 10 KOhm resistors. This should be ensured even in 1-bit mode or SPI mode. Most official modules don't offer these pullups internally. If you are using official development boards, check *Overview of Compatibility* to see whether your development boards have such pullups.

注解: Most official modules have conflicts on strapping pins with the SDIO slave function. If you are using a ESP32 module with 3.3 V flash inside, you have to burn the EFUSE when you are developing on the module for the first time. See *Overview of Compatibility* to see how to make your modules compatible with the SDIO.

Here is a list for modules/kits with 3.3 V flash:

- Modules: ESP32-PICO-D4, ESP32-WROOM-32 series (including ESP32-SOLO-1), ESP32-WROVER-B and ESP32-WROVER-IB
- Kits: ESP32-PICO-KIT, ESP32-DevKitC (till v4), ESP32-WROVER-KIT (v4.1 (also known as ESP32-WROVER-KIT-VB), v2, v1 (also known as DevKitJ v1))

You can tell the version of your ESP23-WROVER-KIT version from the module on it: v4.1 are with ESP32-WROVER-B modules, v3 are with ESP32-WROVER modules, while v2 and v1 are with ESP32-WROOM-32 modules.

Refer to *SD Pull-up Requirements* for more technical details of the pullups.

SD Pull-up Requirements

Espressif hardware products are designed for multiple use cases which may require different pull states on pins. For this reason, the pull state of particular pins on certain products will need to be adjusted to provide the pull-ups required in the SD bus.

SD pull-up requirements apply to cases where ESP32 uses the SPI controller to communicate with SD cards. When an SD card is operating in SPI mode or 1-bit SD mode, the CMD and DATA (DAT0 - DAT3) lines of the SD bus must be pulled up by 10 kOhm resistors. Slaves should also have pull-ups on all above-mentioned lines (regardless of whether these lines are connected to the host) in order to prevent SD cards from entering a wrong state.

By default, the MTDI bootstrapping pin is incompatible with the DAT2 line pull-up if the flash voltage is 3.3 V. For more information, see *MTDI Strapping Pin* below.

This document has the following structure:

- *Overview of compatibility* between the default pull states on pins of Espressif' s products and the states required by the SD bus
- *Solutions* - ideas on how to resolve compatibility issues
- *Related information* - other relevant information

Overview of Compatibility

This section provides an overview of compatibility issues that might occur when using SDIO (secure digital input output). Since the SD bus needs to be connected to pull-ups, these issues should be resolved regardless of whether they are related to master (host) or slave (device). Each issue has links to its respective solution. A solution for a host and device may differ.

Systems on a Chip (SoCs)

- ESP32 (except for D2WD versions, see ESP32 datasheet):
  - *No Pull-ups*
  - *Conflicts Between Bootstrap and SDIO on DAT2* for models with 3.3 V flash chip
- ESP32-D2WD:
  - *No Pull-ups*
  - *No Pull-up on GPIO12*

Systems in Packages (SIP)

- ESP32-PICO-D4:
  - *No Pull-ups*
  - *Conflicts Between Bootstrap and SDIO on DAT2*

Modules

- ESP32-WROOM-32 Series, including ESP32-WROOM-32, ESP32-WROOM-32D, ESP32-WROOM-32U, and ESP32-SOLO-1
  - *No Pull-ups*
  - *Conflicts Between Bootstrap and SDIO on DAT2*
- ESP32-WROVER Series, including ESP32-WROVER and ESP32-WROVER-I
  - *No Pull-ups*
- ESP32-WROVER-B Series, including ESP32-WROVER-B and ESP32-WROVER-IB
  - *No Pull-ups*
  - *Conflicts Between Bootstrap and SDIO on DAT2*

Development Boards

- ESP32-PICO-KIT, including PICO-KIT v4.1, v4.0, and v3
  - *No Pull-ups*
  - *Conflicts Between Bootstrap and SDIO on DAT2*
  - *Download Mode Not Working (minor issue)*
- ESP32-DevKitC, including ESP32-DevKitC v4 and v2

  - *No Pull-ups*
  - *Conflicts Between Bootstrap and SDIO on DAT2*
  - *Download Mode Not Working (minor issue)*

- ESP-WROVER-KIT
  - Required pull-ups are provided
  - *Pull-up Conflicts on GPIO13* (v4.1, v3, v2, and v1)
  - *Conflicts Between Bootstrap and SDIO on DAT2* (v4.1, v2, and v1)
  - *Download Mode Not Working (minor issue)* (v2, v1)

  You can determine the version of your ESP23-WROVER-KIT by checking which module is mounted on it:
  - ESP32-WROVER-B on v4.1
  - ESP32-WROVER on v3
  - ESP32-WROOM-32 on v1 and v2
- ESP32-LyraTD-MSC
  - Required pull-ups are provided
  - *Conflicts Between Bootstrap and SDIO on DAT2*
- ESP32-LyraT
  - Required pull-ups are provided
  - *Pull-up Conflicts on GPIO13*

Non-Espressif Hosts

Please make sure that your SDIO host provides necessary pull-ups for all SD bus signals.

Solutions

No Pull-ups

If you use a development board without pull-ups, you can do the following:

- If your host and slave device are on seperate boards, replace one of them with a board that has pull-ups. For the list of Espressif' s development boards with pull-ups, go to *Development Boards*.
- Attach external pull-ups by connecting each pin which requires a pull-up to VDD via a 10 kOhm resistor.

Pull-up Conflicts on GPIO13

If DAT3 of your device is not properly pulled up, you have the following options:

- Use 1-bit SD mode and tie the device' s DAT3 to VDD
- Use SPI mode
- **Perform one of the following actions on the GPIO13 pin:**
  - Remove the pull-down resistors
  - Attach a pull-up resistor of less than 5 kOhm (2 kOhm suggested)
  - Pull it up or drive it high either by using the host or with 3.3 V on VDD in 1-bit SD mode

Conflicts Between Bootstrap and SDIO on DAT2

There is a conflict between the boot strapping requirements of the ESP32 and the SDIO protocol. For details, see *MTDI Strapping Pin*.

To resolve the conflict, you have the following options:

1. (Recommended) Burn the flash voltage selection eFuses. This will permanently configure the internal regulator' s output voltage to 3.3 V, and GPIO12 will not be used as a bootstrapping pin. After that, connect a pull-up resistor to GPIO12.

   > **警告:** Burning eFuses is irreversible! The issue list above might be out of date, so please make sure that the module you are burning has a 3.3 V flash chip by checking the information on http://www.espressif.com/. If you burn the 3.3 V eFuses on a module with a 1.8 V flash chip, the module will stop functioning.

   If you are sure that you need to irreversibly burn eFuses, go to your ESP-IDF directory and run the following command:

   ```
   components/esptool_py/esptool/espefuse.py set_flash_voltage 3.3V
   ```

   This command will burn the *XPD_SDIO_TIEH*, *XPD_SDIO_FORCE*, and *XPD_SDIO_REG* eFuses. After all the three eFuses are burned to value 1, the internal VDD_SDIO flash voltage regulator will be permanently set to 3.3 V. You will see the following log if the burning succeeds:

   ```
   espefuse.py v2.6
   Connecting....

   Enable internal flash voltage regulator (VDD_SDIO) to 3.3 V.
   ```

(下页继续)

(续上页)

```
The following eFuses are burned: XPD_SDIO_FORCE, XPD_SDIO_REG, XPD_SDIO_TIEH.
This is an irreversible operation.
Type 'BURN' (all capitals) to continue.
BURN
VDD_SDIO setting complete.
```

To check the status of the eFuses, run:

```
``components/esptool_py/esptool/espefuse.py summary``
```

If running from an automated flashing script, `espefuse.py` has an option `--do-not-confirm`.

For more details, see the ESP32 Technical Reference Manual (PDF).

2. **If using 1-bit SD mode or SPI mode**, disconnect the DAT2 pin and make sure it is pulled high. For this, do one the following:
   - Leave the host' s DAT2 floating and directly connect the slave' s DAT2 to VDD.
   - For a slave device, build a firmware with the option `SDIO_SLAVE_FLAG_DAT2_DISABLED` and reflash your device. This option will help avoid slave detecting on the DAT2 line. Note that 4-bit SD mode will no longer be supported by the standard Card Common Control Register (CCCR); however, the host will not be aware of that. The use of 4-bit SD mode will have to be disabled on the host' s side.

No Pull-up on GPIO12

Your module is compatible with the SDIO protocol. Just connect GPIO12 to VDD via a 10 kOhm resistor.

Download Mode Not Working (minor issue)

When the GPIO2 pin is pulled high in accordance with the SD pull-up requirements, you cannot enter Download mode because GPIO2 is a bootstrapping pin which in this case must be pulled low.

There are the following solutions:

- For boards that require shorting the GPIO0 and GPIO2 pins with a jumper, put the jumper in place, and the auto-reset circuit will pull GPIO2 low along with GPIO0 before entering Download mode.
- For boards with components attached to their GPIO2 pin (such as pull-down resistors and/or LEDs), check the schematic of your development board for anything connected to GPIO2.
  - **LEDs** would not affect operation in most cases.
  - **Pull-down resistors** can interfere with DAT0 signals and must be removed.

If the above solutions do not work for you, please determine if it is the host or slave device that has pull-ups affecting their GPIO2, then locate these pull-ups and remove them.

Related Information

MTDI Strapping Pin

MTDI (GPIO12) is used as a bootstrapping pin to select the output voltage of an internal regulator (VDD_SDIO) which powers the flash chip. This pin has an internal pull-down, so, if left unconnected, it will read low at startup, which will lead to selecting the default 3.3 V operation.

All ESP32-WROVER modules, excluding ESP32-WROVER-B, use 1.8 V flash and have internal pull-ups on GPIO12. Other modules that use 3.3 V flash have no pull-ups on the GPIO12 pin, and this pin is slightly pulled down internally.

When adding a pull-up to this pin for SD card operation, consider the following:

- For boards that do not use the internal regulator (VDD_SDIO) to power flash, GPIO12 can be pulled high.
- For boards using 1.8 V flash chips, GPIO12 needs to be pulled high at reset. This is fully compatible with the SD card operation.
- On boards using the internal regulator and a 3.3 V flash chip, GPIO12 must be pulled low at reset. This is incompatible with the SD card operation. For reference information on compatibility of Espressif' s boards with the SD card operation, see *Overview of Compatibility*.

Internal Pull-ups and Strapping Requirements

Using external resistors is always preferable. However, Espressif' s products have internal weak pull-up and pull-down resistors which can be enabled and used instead of external ones. Please keep in mind that this solution cannot guarantee reliable SDIO communication.

With that said, the information about these internal pull-ups and strapping requirements can still be useful. Espressif hardware products have different weak internal pull-ups / pull-downs connected to CMD and DATA pins. The table below shows the default pull-up and pull-down states of the CMD and DATA pins.

The following abbreviations are used in the table:

- **WPU**: Weak pull-up inside the SoC
- **WPD**: Weak pull-down inside the SoC
- **PU**: Pull-up inside Espressif modules but outside the SoC

表 2: Default pull-up and pull-down states of the CMD and DATA pins

GPIO number	Pin Name	Startup State	Strapping Requirement
15	CMD	WPU	
2	DAT0	WPD	Low for Download mode
4	DAT1	WPD	
12	DAT2	PU for 1.8 V flash; WPD for 3.3 V flash	High for 1.8 V flash; Low for 3.3 V flash
13	DAT3	WPU	

The host initialize the slave into SD mode by first sending CMD0 with DAT3 pin high, or in SPI mode by sending CMD0 with CS pin (the same pin as DAT3) low.

After the initialization, the host can enable the 4-bit SD mode by writing CCCR register 0x07 by CMD52. All the bus detection process are handled by the slave peripheral.

The host has to communicate with the slave by an ESP-slave-specific protocol. The slave driver offers 3 services over Function 1 access by CMD52 and CMD53: (1) a sending FIFO and a receiving FIFO, (2) 52 8-bit R/W registers shared by host and slave, (3) 16 interrupt sources (8 from host to slave, and 8 from slave to host).

Terminology

The SDIO slave driver uses the following terms:

- Transfer: a transfer is always started by a command token from the host, and may contain a reply and several data blocks. ESP32 slave software is based on transfers.
- Sending: slave to host transfers.
- Receiving: host to slave transfers.

注解: Register names in ESP Rechnical Reference Manual are oriented from the point of view of the host, i.e. 'rx' registers refer to sending, while 'tx' registers refer to receiving. We' re not using *tx* or *rx* in the driver to avoid ambiguities.

- FIFO: specific address in Function 1 that can be access by CMD53 to read/write large amount of data. The address is related to the length requested to read from/write to the slave in a single transfer: *requested length* = 0x1F800-address.
- Ownership: When the driver takes ownership of a buffer, it means the driver can randomly read/write the buffer (usually via DMA). The application should not read/write the buffer until the ownership is returned to the application. If the application reads from a buffer owned by a receiving driver, the

data read can be random; if the application writes to a buffer owned by a sending driver, the data sent may be corrupted.

- Requested length: The length requested in one transfer determined by the FIFO address.
- Transfer length: The length requested in one transfer determined by the CMD53 byte/block count field.

注解: Requested length is different from the transfer length. ESP32 slave DMA base on the *requested length* rather than the *transfer length*. The *transfer length* should be no shorter than the *requested length*, and the rest part will be filled with 0 (sending) or discard (receiving).

- Receiving buffer size: The buffer size is pre-defined between the host and the slave before communication starts. Slave application has to set the buffer size during initialization by the `recv_buffer_size` member of `sdio_slave_config_t`.
- Interrupts: the esp32 slave support interrupts in two directions: from host to slave (called slave interrupts below) and from slave to host (called host interrupts below). See more in *Interrupts*.
- Registers: specific address in Function 1 access by CMD52 or CMD53.

Communication with ESP SDIO Slave

The host should initialize the ESP32 SDIO slave according to the standard SDIO initialization process (Sector 3.1.2 of SDIO Simplified Specification), which is described briefly in *ESP SDIO slave initialization*.

Furthermore, there' s an ESP32-specific upper-level communication protocol upon the CMD52/CMD53 to Func 1. Please refer to *ESP SDIO slave protocol*. There is also a component *ESP Serial Slave Link* for ESP32 master to communicate with ESP32 SDIO slave, see example peripherals/sdio when programming your host.

Communication with ESP SDIO Slave

ESP SDIO slave initialization

The host should initialize the ESP32 SDIO slave according to the standard SDIO initialization process (Sector 3.1.2 of SDIO Simplified Specification). In this specification and below, the SDIO slave is also called an (SD)IO card. All the initialization CMD52 and CMD53 are sent to Func 0 (CIA region). Here is an brief example on how to do this:

1. **SDIO reset** CMD52 (Write 0x6=0x8)
2. **SD reset** CMD0
3. **Check whether IO card (optional)** CMD8

4. **Send SDIO op cond and wait for card ready** CMD5 arg = 0x00000000

   CMD5 arg = 0x00ff8000 (according to the response above, poll until ready)

   Example: Arg of R4 after first CMD5 (arg=0x00000000) is 0xXXFFFF00.

   Keep sending CMD5 with arg=0x00FFFF00 until the R4 shows card ready (arg bit 31=1).

5. **Set address** CMD3
6. **Select card** CMD7 (arg address according to CMD3 response)

   Example: Arg of R6 after CMD3 is 0x0001xxxx.

   Arg of CMD7 should be 0x00010000.

7. **Select 4-bit mode (optional)** CMD52 (Write 0x07=0x02)
8. **Enable func1** CMD52 (Write 0x02=0x02)
9. **Enable SDIO interrupt (required if interrupt line (DAT1) is used)** CMD52 (Write 0x04=0x03)
10. **Set Func0 blocksize (optional, default value is 512 (0x200))** CMD52/53 (Read 0x10~0x11)

    CMD52/53 (Write 0x10=0x00)

    CMD52/53 (Write 0x11=0x02)

    CMD52/53 (Read 0x10~0x11, read to check the final value)

11. **Set Func1 blocksize (optional, default value is 512 (0x200))** CMD52/53 (Read 0x110~0x111)

    CMD52/53 (Write 0x110=0x00)

    CMD52/53 (Write 0x111=0x02)

    CMD52/53 (Read 0x110~0x111, read to check the final value)

ESP SDIO slave protocol

The protocol is based on Function 1 access by CMD52 and CMD53, offering 3 services: (1) sending and receiving FIFO, (2) 52 8-bit R/W register shared by host and slave, (3) 8 general purpose interrupt sources from host to slave and 8 in the opposite direction.

There is a component *esp_serial_slave_link* implementing the logic of this protocol for ESP32 master to communicate with the ESP32 slave. See *ESP Serial Slave Link*.

The host should access the registers below as described to communicate with slave.

Slave register table

32-bit

- 0x044 (TOKEN_RDATA): in which bit 27-16 holds the receiving buffer number.
- 0x058 (INT_ST): holds the interrupt source bits from slave to host.
- 0x060 (PKT_LEN): holds the accumulated length (by byte) to be sent from slave to host.
- 0x0D4 (INT_CLR): write 1 to clear interrupt bits corresponding to INT_ST.
- 0x0DC (INT_ENA): mask bits for interrupts from slave to host.

8-bit

Shared general purpose registers:

- 0x06C-0x077: R/W registers 0-11 shared by slave and host.
- 0x07A-0x07B: R/W registers 14-15 shared by slave and host.
- 0x07E-0x07F: R/W registers 18-19 shared by slave and host.
- 0x088-0x08B: R/W registers 24-27 shared by slave and host.
- 0x09C-0x0BB: R/W registers 32-63 shared by slave and host.

Interrupt Registers: - 0x08D (SLAVE_INT): bits for host to interrupt slave. auto clear.

FIFO (sending and receiving)

0x090 - 0x1F7FF are reserved for FIFOs.

The address of CMD53 is related to the length requested to read from/write to the slave in a single transfer:

requested length = 0x1F800-address

The slave will respond with the length according to the length field in CMD53, with the data longer than *requested length* filled with 0 (sending) or discard (receiving).

注解: This includes both the block and the byte mode of CMD53.

The function number should be set to 1, OP Code should be set to 1 (for CMD53).

It is allowed to use CMD53 mode combination of block+byte to get higher effeciency when accessing the FIFO by arbitrary length. E.g. The block size is set to 512 by default, you can write/get 1031 bytes of data to/from the FIFO by:

1. Send CMD53 in block mode, block count=2 (1024 bytes) to address 0x1F3F9=0x1F800-**1031**.
2. Then send CMD53 in byte mode, byte count=8 (or 7 if your controller supports that) to address 0x1F7F9=0x1F800-**7**.

Interrupts

For the host interrupts, the slave raise the interrupt by pulling DAT1 line down at a proper time (level sensitive). The host detect this and read the INT_ST register to see the source. Then the host can clear it by writing the INT_CLR register and do something with the interrupt. The host can also mask unneeded sources by clearing the bits in INT_ENA register corresponding to the sources. If all the sources are cleared (or masked), the DAT1 line goes inactive.

`sdio_slave_hostint_t` (*SDIO Card Slave Driver*) shows the bit definition corresponding to host interrupt sources.

For the slave interrupts, the host send transfers to write the SLAVE_INT register. Once a bit is written from 0 to 1, the slave hardware and driver will detect it and inform the app.

Receiving FIFO

To write the receiving FIFO in the slave, host should work in the following steps:

1. Read the TOKEN1 field (bits 27-16) of TOKEN_RDATA (0x044) register. The buffer number remaining is TOKEN1 minus the number of buffers used by host.
2. Make sure the buffer number is sufficient (*recv_buffer_size* * *buffer_num* is greater than data to write, *recv_buffer_size* is pre-defined between the host and the slave before the communication starts). Or go back to step 1 until the buffer is enough.
3. Write to the FIFO address with CMD53. Note that the *requested length* should not be larger than calculated in step 2, and the FIFO address is related to *rquested length*.
4. Calculate used buffers, note that non-full buffer at the tail should be seen as one that is used.

Sending FIFO

To read the sending FIFO in the slave, host should work in the following steps:

1. Wait for the interrupt line to be active (optional, low by default).
2. Read (poll) the interrupt bits in INT_ST register to see whether new packets exists.
3. If new packets are ready, reads the PKT_LEN reg. The data length to read from slave is PKT_LEN minuses the length that has been read from the host. If the PKT_LEN is not larger than used, wait and poll until the slave is ready and update the PKT_LEN.
4. Read from the FIFO with CMD53. Note that the *requested length* should not be larger than calculated in step3, and the FIFO address is related to *requested length*.

5. Recored read length.

Interrupts

There are interrupts from host to slave, and from slave to host to help communicating conveniently.

Slave Interrupts

The host can interrupt the slave by writing any one bit in the register 0x08D. Once any bit of the register is set, an interrupt is raised and the SDIO slave driver calls the callback function defined in the `slave_intr_cb` member in the `sdio_slave_config_t` structure.

注解: The callback function is called in the ISR, do not use any delay, loop or spinlock in the callback.

There' s another set of functions can be used. You can call `sdio_slave_wait_int` to wait for an interrupt within a certain time, or call `sdio_slave_clear_int` to clear interrupts from host. The callback function can work with the wait functions perfectly.

Host Interrupts

The slave can interrupt the host by an interrupt line (at certain time) which is level sensitive. When the host see the interrupt line pulled down, it may read the slave interrupt status register, to see the interrupt source. Host can clear interrupt bits, or choose to disable a interrupt source. The interrupt line will hold active until all the sources are cleared or disabled.

There are several dedicated interrupt sources as well as general purpose sources. see `sdio_slave_hostint_t` for more information.

Shared Registers

There are 52 8-bit R/W shared registers to share information between host and slave. The slave can write or read the registers at any time by `sdio_slave_read_reg` and `sdio_slave_write_reg`. The host can access (R/W) the register by CMD52 or CMD53.

Receiving FIFO

When the host is going to send the slave some packets, it has to check whether the slave is ready to receive by reading the buffer number of slave.

To allow the host sending data to the slave, the application has to load buffers to the slave driver by the following steps:

1. Register the buffer by calling `sdio_slave_recv_register_buf`, and get the handle of the registered buffer. The driver will allocate memory for the linked-list descriptor needed to link the buffer onto the hardware.
2. Load buffers onto the driver by passing the buffer handle to `sdio_slave_recv_load_buf`.
3. Call `sdio_slave_recv` to get the received data. If non-blocking call is needed, set `wait=0`.
4. Pass the handle of processed buffer back to the driver by `sdio_recv_load_buf` again.

注解: To avoid overhead from copying data, the driver itself doesn't have any buffer inside, the application is responsible to offer new buffers in time. The DMA will automatically store received data to the buffer.

Sending FIFO

Each time the slave has data to send, it raises an interrupt and the host will request for the packet length. There are two sending modes:

- Stream Mode: when a buffer is loaded to the driver, the buffer length will be counted into the packet length requested by host in the incoming communications. Regardless previous packets are sent or not. This means the host can get data of several buffers in one transfer.
- Packet Mode: the packet length is updated packet by packet, and only when previous packet is sent. This means that the host can only get data of one buffer in one transfer.

注解: To avoid overhead from copying data, the driver itself doesn't have any buffer inside. Namely, the DMA takes data directly from the buffer provided by the application. The application should not touch the buffer until the sending is finished.

The sending mode can be set in the `sending_mode` member of `sdio_slave_config_t`, and the buffer numbers can be set in the `send_queue_size`. All the buffers are restricted to be no larger than 4092 bytes. Though in the stream mode several buffers can be sent in one transfer, each buffer is still counted as one in the queue.

The application can call `sdio_slave_transmit` to send packets. In this case the function returns when the transfer is successfully done, so the queue is not fully used. When higher effeciency is required, the application can use the following functions instead:

1. Pass buffer information (address, length, as well as an `arg` indicating the buffer) to `sdio_slave_send_queue`. If non-blocking call is needed, set `wait=0`. If the `wait` is not `portMAX_DELAY` (wait until success), application has to check the result to know whether the data is put in to the queue or discard.
2. Call `sdio_slave_send_get_finished` to get and deal with a finished transfer. A buffer should be keep unmodified until returned from `sdio_slave_send_get_finished`. This means the buffer is actually

sent to the host, rather than just staying in the queue.

There are several ways to use the `arg` in the queue parameter:

1. Directly point `arg` to a dynamic-allocated buffer, and use the `arg` to free it when transfer finished.
2. Wrap transfer informations in a transfer structure, and point `arg` to the structure. You can use the structure to do more things like:

```
typedef struct {
    uint8_t* buffer;
    size_t   size;
    int      id;
}sdio_transfer_t;

//and send as:
sdio_transfer_t trans = {
    .buffer = ADDRESS_TO_SEND,
    .size = 8,
    .id = 3,  //the 3rd transfer so far
};
sdio_slave_send_queue(trans.buffer, trans.size, &trans, portMAX_DELAY);

//... maybe more transfers are sent here

//and deal with finished transfer as:
sdio_transfer_t* arg = NULL;
sdio_slave_send_get_finished((void**)&arg, portMAX_DELAY);
ESP_LOGI("tag", "(%d) successfully send %d bytes of %p", arg->id, arg->size, arg->
↪buffer);
some_post_callback(arg); //do more things
```

3. Working with the receiving part of this driver, point `arg` to the receive buffer handle of this buffer. So that we can directly use the buffer to receive data when it' s sent:

```
uint8_t buffer[256]={1,2,3,4,5,6,7,8};
sdio_slave_buf_handle_t handle = sdio_slave_recv_register_buf(buffer);
sdio_slave_send_queue(buffer, 8, handle, portMAX_DELAY);

//... maybe more transfers are sent here

//and load finished buffer to receive as
sdio_slave_buf_handle_t handle = NULL;
```

(下页继续)

(续上页)

```
sdio_slave_send_get_finished((void**)&handle, portMAX_DELAY);
sdio_slave_recv_load_buf(handle);
```

More about this, see peripherals/sdio.

Application Example

Slave/master communication: peripherals/sdio.

API Reference

Header File

- soc/include/hal/sdio_slave_types.h

Enumerations

enum sdio_slave_hostint_t

Mask of interrupts sending to the host.

Values:

SDIO_SLAVE_HOSTINT_BIT0 = BIT(0)

General purpose interrupt bit 0.

SDIO_SLAVE_HOSTINT_BIT1 = BIT(1)

SDIO_SLAVE_HOSTINT_BIT2 = BIT(2)

SDIO_SLAVE_HOSTINT_BIT3 = BIT(3)

SDIO_SLAVE_HOSTINT_BIT4 = BIT(4)

SDIO_SLAVE_HOSTINT_BIT5 = BIT(5)

SDIO_SLAVE_HOSTINT_BIT6 = BIT(6)

SDIO_SLAVE_HOSTINT_BIT7 = BIT(7)

SDIO_SLAVE_HOSTINT_SEND_NEW_PACKET = BIT(23)

New packet available.

enum sdio_slave_timing_t

Timing of SDIO slave.

Values:

`SDIO_SLAVE_TIMING_PSEND_PSAMPLE` = 0
: Send at posedge, and sample at posedge. Default value for HS mode. Normally there' s no problem using this to work in DS mode.

`SDIO_SLAVE_TIMING_NSEND_PSAMPLE`
: Send at negedge, and sample at posedge. Default value for DS mode and below.

`SDIO_SLAVE_TIMING_PSEND_NSAMPLE`
: Send at posedge, and sample at negedge.

`SDIO_SLAVE_TIMING_NSEND_NSAMPLE`
: Send at negedge, and sample at negedge.

enum sdio_slave_sending_mode_t
: Configuration of SDIO slave mode.

Values:

`SDIO_SLAVE_SEND_STREAM` = 0
: Stream mode, all packets to send will be combined as one if possible.

`SDIO_SLAVE_SEND_PACKET` = 1
: Packet mode, one packets will be sent one after another (only increase packet_len if last packet sent).

Header File

- driver/include/driver/sdio_slave.h

Functions

esp_err_t **sdio_slave_initialize**(*sdio_slave_config_t* **config*)
: Initialize the sdio slave driver

Return

- ESP_ERR_NOT_FOUND if no free interrupt found.
- ESP_ERR_INVALID_STATE if already initialized.
- ESP_ERR_NO_MEM if fail due to memory allocation failed.
- ESP_OK if success

Parameters

- `config`: Configuration of the sdio slave driver.

void **sdio_slave_deinit**(void)

De-initialize the sdio slave driver to release the resources.

esp_err_t **sdio_slave_start**(void)

Start hardware for sending and receiving, as well as set the IOREADY1 to 1.

Note The driver will continue sending from previous data and PKT_LEN counting, keep data received as well as start receiving from current TOKEN1 counting. See `sdio_slave_reset`.

Return

- ESP_ERR_INVALID_STATE if already started.
- ESP_OK otherwise.

void **sdio_slave_stop**(void)

Stop hardware from sending and receiving, also set IOREADY1 to 0.

Note this will not clear the data already in the driver, and also not reset the PKT_LEN and TOKEN1 counting. Call `sdio_slave_reset` to do that.

esp_err_t **sdio_slave_reset**(void)

Clear the data still in the driver, as well as reset the PKT_LEN and TOKEN1 counting.

Return always return ESP_OK.

sdio_slave_buf_handle_t **sdio_slave_recv_register_buf**(uint8_t **start*)

Register buffer used for receiving. All buffers should be registered before used, and then can be used (again) in the driver by the handle returned.

Note The driver will use and only use the amount of space specified in the `recv_buffer_size` member set in the *sdio_slave_config_t*. All buffers should be larger than that. The buffer is used by the DMA, so it should be DMA capable and 32-bit aligned.

Return The buffer handle if success, otherwise NULL.

Parameters

- `start`: The start address of the buffer.

esp_err_t **sdio_slave_recv_unregister_buf**(*sdio_slave_buf_handle_t handle*)

Unregister buffer from driver, and free the space used by the descriptor pointing to the buffer.

Return ESP_OK if success, ESP_ERR_INVALID_ARG if the handle is NULL or the buffer is being used.

Parameters

- `handle`: Handle to the buffer to release.

esp_err_t **sdio_slave_recv_load_buf**(*sdio_slave_buf_handle_t handle*)

Load buffer to the queue waiting to receive data. The driver takes ownership of the buffer until the buffer is returned by `sdio_slave_send_get_finished` after the transaction is finished.

Return

- ESP_ERR_INVALID_ARG if invalid handle or the buffer is already in the queue. Only after the buffer is returened by `sdio_slave_recv` can you load it again.
- ESP_OK if success

Parameters

- `handle`: Handle to the buffer ready to receive data.

esp_err_t **sdio_slave_recv**(*sdio_slave_buf_handle_t* **handle_ret*, uint8_t ****out_addr*, size_t **out_len*, TickType_t *wait*)

Get received data if exist. The driver returns the ownership of the buffer to the app.

Note Call `sdio_slave_load_buf` with the handle to re-load the buffer onto the link list, and receive with the same buffer again. The address and length of the buffer got here is the same as got from `sdio_slave_get_buffer`.

Return

- ESP_ERR_INVALID_ARG if handle_ret is NULL
- ESP_ERR_TIMEOUT if timeout before receiving new data
- ESP_OK if success

Parameters

- `handle_ret`: Handle to the buffer holding received data. Use this handle in `sdio_slave_recv_load_buf` to receive in the same buffer again.
- `out_addr`: Output of the start address, set to NULL if not needed.
- `out_len`: Actual length of the data in the buffer, set to NULL if not needed.
- `wait`: Time to wait before data received.

uint8_t ***sdio_slave_recv_get_buf**(*sdio_slave_buf_handle_t handle*, size_t **len_o*)

Retrieve the buffer corresponding to a handle.

Return buffer address if success, otherwise NULL.

Parameters

- `handle`: Handle to get the buffer.
- `len_o`: Output of buffer length

esp_err_t **sdio_slave_send_queue**(uint8_t **addr*, size_t *len*, void **arg*, TickType_t *wait*)

Put a new sending transfer into the send queue. The driver takes ownership of the buffer until the buffer is returned by `sdio_slave_send_get_finished` after the transaction is finished.

Return

- ESP_ERR_INVALID_ARG if the length is not greater than 0.
- ESP_ERR_TIMEOUT if the queue is still full until timeout.
- ESP_OK if success.

Parameters

- `addr`: Address for data to be sent. The buffer should be DMA capable and 32-bit aligned.
- `len`: Length of the data, should not be longer than 4092 bytes (may support longer in the future).
- `arg`: Argument to returned in `sdio_slave_send_get_finished`. The argument can be used to indicate which transaction is done, or as a parameter for a callback. Set to NULL if not needed.
- `wait`: Time to wait if the buffer is full.

esp_err_t **sdio_slave_send_get_finished**(void ***out_arg*, TickType_t *wait*)

Return the ownership of a finished transaction.

Return ESP_ERR_TIMEOUT if no transaction finished, or ESP_OK if succeed.

Parameters

- `out_arg`: Argument of the finished transaction. Set to NULL if unused.
- `wait`: Time to wait if there' s no finished sending transaction.

esp_err_t **sdio_slave_transmit**(uint8_t **addr*, size_t *len*)

Start a new sending transfer, and wait for it (blocked) to be finished.

Return

- ESP_ERR_INVALID_ARG if the length of descriptor is not greater than 0.
- ESP_ERR_TIMEOUT if the queue is full or host do not start a transfer before timeout.
- ESP_OK if success.

Parameters

- `addr`: Start address of the buffer to send
- `len`: Length of buffer to send.

uint8_t **sdio_slave_read_reg**(int *pos*)

Read the spi slave register shared with host.

Note register 28 to 31 are reserved for interrupt vector.

Return value of the register.

Parameters

- `pos`: register address, 0-27 or 32-63.

esp_err_t **sdio_slave_write_reg**(int *pos*, uint8_t *reg*)

Write the spi slave register shared with host.

Note register 29 and 31 are used for interrupt vector.

Return ESP_ERR_INVALID_ARG if address wrong, otherwise ESP_OK.

Parameters

- `pos`: register address, 0-11, 14-15, 18-19, 24-27 and 32-63, other address are reserved.
- `reg`: the value to write.

sdio_slave_hostint_t **sdio_slave_get_host_intena**(void)

Get the interrupt enable for host.

Return the interrupt mask.

void **sdio_slave_set_host_intena**(*sdio_slave_hostint_t* *mask*)

Set the interrupt enable for host.

Parameters

- `mask`: Enable mask for host interrupt.

esp_err_t **sdio_slave_send_host_int**(uint8_t *pos*)

Interrupt the host by general purpose interrupt.

Return

- ESP_ERR_INVALID_ARG if interrupt num error
- ESP_OK otherwise

Parameters

- `pos`: Interrupt num, 0-7.

void **sdio_slave_clear_host_int**(*sdio_slave_hostint_t* *mask*)

Clear general purpose interrupt to host.

Parameters

- `mask`: Interrupt bits to clear, by bit mask.

esp_err_t **sdio_slave_wait_int**(int *pos*, TickType_t *wait*)

Wait for general purpose interrupt from host.

Note this clears the interrupt at the same time.

Return ESP_OK if success, ESP_ERR_TIMEOUT if timeout.

Parameters

- `pos`: Interrupt source number to wait for. is set.
- `wait`: Time to wait before interrupt triggered.

Structures

struct sdio_slave_config_t

Configuration of SDIO slave.

Public Members

sdio_slave_timing_t **timing**

timing of sdio_slave. see `sdio_slave_timing_t`.

sdio_slave_sending_mode_t **sending_mode**

mode of sdio_slave. `SDIO_SLAVE_MODE_STREAM` if the data needs to be sent as much as possible; `SDIO_SLAVE_MODE_PACKET` if the data should be sent in packets.

int **send_queue_size**

max buffers that can be queued before sending.

size_t **recv_buffer_size**

If buffer_size is too small, it costs more CPU time to handle larger number of buffers. If buffer_size is too large, the space larger than the transaction length is left blank but still counts a buffer, and the buffers are easily run out. Should be set according to length of data really transferred. All data that do not fully fill a buffer is still counted as one buffer. E.g. 10 bytes data costs 2 buffers if the size is 8 bytes per buffer. Buffer size of the slave pre-defined between host and slave before communication. All receive buffer given to the driver should be larger than this.

sdio_event_cb_t **event_cb**

when the host interrupts slave, this callback will be called with interrupt number (0-7).

uint32_t **flags**

Features to be enabled for the slave, combinations of `SDIO_SLAVE_FLAG_*`.

Macros

`SDIO_SLAVE_RECV_MAX_BUFFER`

`SDIO_SLAVE_FLAG_DAT2_DISABLED`

It is required by the SD specification that all 4 data lines should be used and pulled up even in 1-bit mode or SPI mode. However, as a feature, the user can specify this flag to make use of DAT2 pin in 1-bit mode. Note that the host cannot read CCCR registers to know we don't support 4-bit mode anymore, please do this at your own risk.

`SDIO_SLAVE_FLAG_HOST_INTR_DISABLED`

The DAT1 line is used as the interrupt line in SDIO protocol. However, as a feature, the user can specify this flag to make use of DAT1 pin of the slave in 1-bit mode. Note that the host has to do polling to the interrupt registers to know whether there are interrupts from the slave. And it cannot read CCCR registers to know we don't support 4-bit mode anymore, please do this at your own risk.

`SDIO_SLAVE_FLAG_INTERNAL_PULLUP`

Enable internal pullups for enabled pins. It is required by the SD specification that all the 4 data lines should be pulled up even in 1-bit mode or SPI mode. Note that the internal pull-ups are not sufficient for stable communication, please do connect external pull-ups on the bus. This is only for example and debug use.

Type Definitions

`typedef` void (*`sdio_event_cb_t`)(uint8_t event)

`typedef` void *`sdio_slave_buf_handle_t`

Handle of a receive buffer, register a handle by calling `sdio_slave_recv_register_buf`. Use the handle to load the buffer to the driver, or call `sdio_slave_recv_unregister_buf` if it is no longer used.

2.3.14 Sigma-delta Modulation

Introduction

ESP32 has a second-order sigma-delta modulation module. This driver configures the channels of the sigma-delta module.

Functionality Overview

There are eight independent sigma-delta modulation channels identified with *`sigmadelta_channel_t`*. Each channel is capable to output the binary, hardware generated signal with the sigma-delta modulation.

Selected channel should be set up by providing configuration parameters in *sigmadelta_config_t* and then applying this configuration with *sigmadelta_config()*.

Another option is to call individual functions, that will configure all required parameters one by one:

- **Prescaler** of the sigma-delta generator - *sigmadelta_set_prescale()*
- **Duty** of the output signal - *sigmadelta_set_duty()*
- **GPIO pin** to output modulated signal - *sigmadelta_set_pin()*

The range of the 'duty' input parameter of *sigmadelta_set_duty()* is from -128 to 127 (eight bit signed integer). If zero value is set, then the output signal' s duty will be about 50%, see description of *sigmadelta_set_duty()*.

Application Example

Sigma-delta Modulation example: peripherals/sigmadelta.

API Reference

Header File

- driver/include/driver/sigmadelta.h

Functions

esp_err_t **sigmadelta_config**(**const** *sigmadelta_config_t* **config*)

Configure Sigma-delta channel.

Return

- ESP_OK Success
- ESP_ERR_INVALID_STATE sigmadelta driver already initialized
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `config`: Pointer of Sigma-delta channel configuration struct

esp_err_t **sigmadelta_set_duty**(*sigmadelta_channel_t* *channel*, int8_t *duty*)

Set Sigma-delta channel duty.

This function is used to set Sigma-delta channel duty, If you add a capacitor between the output pin and ground, the average output voltage will be Vdc = VDDIO / 256 * duty + VDDIO/2, where VDDIO is the power supply voltage.

Return

- ESP_OK Success
- ESP_ERR_INVALID_STATE sigmadelta driver has not been initialized
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `channel`: Sigma-delta channel number
- `duty`: Sigma-delta duty of one channel, the value ranges from -128 to 127, recommended range is -90 ~ 90. The waveform is more like a random one in this range.

esp_err_t **sigmadelta_set_prescale**(*sigmadelta_channel_t channel*, uint8_t *prescale*)

Set Sigma-delta channel' s clock pre-scale value. The source clock is APP_CLK, 80MHz. The clock frequency of the sigma-delta channel is APP_CLK / pre_scale.

Return

- ESP_OK Success
- ESP_ERR_INVALID_STATE sigmadelta driver has not been initialized
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `channel`: Sigma-delta channel number
- `prescale`: The divider of source clock, ranges from 0 to 255

esp_err_t **sigmadelta_set_pin**(*sigmadelta_channel_t channel*, *gpio_num_t gpio_num*)

Set Sigma-delta signal output pin.

Return

- ESP_OK Success
- ESP_ERR_INVALID_STATE sigmadelta driver has not been initialized
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `channel`: Sigma-delta channel number
- `gpio_num`: GPIO number of output pin.

Header File

- soc/include/hal/sigmadelta_types.h

Structures

struct sigmadelta_config_t

Sigma-delta configure struct.

Public Members

sigmadelta_channel_t **channel**

Sigma-delta channel number

int8_t **sigmadelta_duty**

Sigma-delta duty, duty ranges from -128 to 127.

uint8_t **sigmadelta_prescale**

Sigma-delta prescale, prescale ranges from 0 to 255.

uint8_t **sigmadelta_gpio**

Sigma-delta output io number, refer to gpio.h for more details.

Type Definitions

typedef int **sigmadelta_port_t**

SIGMADELTA port number, the max port number is (SIGMADELTA_NUM_MAX -1).

typedef int **sigmadelta_channel_t**

Sigma-delta channel list.

2.3.15 SPI Master Driver

SPI Master driver is a program that controls ESP32' s SPI peripherals while they function as masters.

Overview of ESP32' s SPI peripherals

ESP32 integrates four SPI peripherals.

- SPI0 and SPI1 are used internally to access the ESP32' s attached flash memory and thus are currently not open to users. They share one signal bus via an arbiter.
- SPI2 and SPI3 are general purpose SPI controllers, sometimes referred to as HSPI and VSPI, respectively. They are open to users. SPI2 and SPI3 have independent signal buses with the same respective names. Each bus has three CS lines to drive up to three SPI slaves.

Terminology

The terms used in relation to the SPI master driver are given in the table below.

Term	Definition
Host	The SPI controller peripheral inside ESP32 that initiates SPI transmissions over the bus, and acts as an SPI Master. This may be the SPI2 or SPI3 peripheral. (The driver will also support the SPI1 peripheral in the future.)
Device	SPI slave device. An SPI bus may be connected to one or more Devices. Each Device shares the MOSI, MISO and SCLK signals but is only active on the bus when the Host asserts the Device' s individual CS line.
Bus	A signal bus, common to all Devices connected to one Host. In general, a bus includes the following lines: MISO, MOSI, SCLK, one or more CS lines, and, optionally, QUADWP and QUADHD. So Devices are connected to the same lines, with the exception that each Device has its own CS line. Several Devices can also share one CS line if connected in the daisy-chain manner.
• **MISO**	Master In, Slave Out, a.k.a. Q. Data transmission from a Device to Host.
• **MOSI**	Master Out, Slave In, a.k.a. D. Data transmission from a Host to Device.
• **SCLK**	Serial Clock. Oscillating signal generated by a Host that keeps the transmission of data bits in sync.
• **CS**	Chip Select. Allows a Host to select individual Device(s) connected to the bus in order to send or receive data.
• **QUADWP**	Write Protect signal. Only used for 4-bit (qio/qout) transactions.
• **QUADHD**	Hold signal. Only used for 4-bit (qio/qout) transactions.
• **Assertion**	The action of activating a line. The opposite action of returning the line back to inactive (back to idle) is called *de-assertion*.
Transaction	One instance of a Host asserting a CS line, transferring data to and from a Device, and de-asserting the CS line. Transactions are atomic, which means they can never be interrupted by another transaction.
Launch edge	Edge of the clock at which the source register *launches* the signal onto the line.
Latch edge	Edge of the clock at which the destination register

Driver Features

The SPI master driver governs communications of Hosts with Devices. The driver supports the following features:

- Multi-threaded environments
- Transparent handling of DMA transfers while reading and writing data
- Automatic time-division multiplexing of data coming from different Devices on the same signal bus

警告: The SPI master driver has the concept of multiple Devices connected to a single bus (sharing a single ESP32 SPI peripheral). As long as each Device is accessed by only one task, the driver is thread safe. However, if multiple tasks try to access the same SPI Device, the driver is **not thread-safe**. In this case, it is recommended to either:

- Refactor your application so that each SPI peripheral is only accessed by a single task at a time.
- Add a mutex lock around the shared Device using *xSemaphoreCreateMutex*.

SPI Transactions

An SPI bus transaction consists of five phases which can be found in the table below. Any of these phases can be skipped.

Phase	Description
Command	In this phase, a command (0-16 bit) is written to the bus by the Host.
Address	In this phase, an address (0-64 bit) is transmitted over the bus by the Host.
Write	Host sends data to a Device. This data follows the optional command and address phases and is indistinguishable from them at the electrical level.
Dummy	This phase is configurable and is used to meet the timing requirements.
Read	Device sends data to its Host.

The attributes of a transaction are determined by the bus configuration structure *spi_bus_config_t*, device configuration structure *spi_device_interface_config_t*, and transaction configuration structure *spi_transaction_t*.

An SPI Host can send full-duplex transactions, during which the read and write phases occur simultaneously. The total transaction length is determined by the sum of the following members:

- *spi_device_interface_config_t::command_bits*
- *spi_device_interface_config_t::address_bits*

- *spi_transaction_t::length*

While the member *spi_transaction_t::rxlength* only determines the length of data received into the buffer.

In half-duplex transactions, the read and write phases are not simultaneous (one direction at a time). The lengths of the write and read phases are determined by `length` and `rxlength` members of the struct *spi_transaction_t* respectively.

The command and address phases are optional, as not every SPI device requires a command and/or address. This is reflected in the Device' s configuration: if `command_bits` and/or `address_bits` are set to zero, no command or address phase will occur.

The read and write phases can also be optional, as not every transaction requires both writing and reading data. If `rx_buffer` is NULL and `SPI_TRANS_USE_RXDATA` is not set, the read phase is skipped. If `tx_buffer` is NULL and `SPI_TRANS_USE_TXDATA` is not set, the write phase is skipped.

The driver supports two types of transactions: the interrupt transactions and polling transactions. The programmer can choose to use a different transaction type per Device. If your Device requires both transaction types, see *Notes on Sending Mixed Transactions to the Same Device*.

Interrupt Transactions

Interrupt transactions will block the transaction routine until the transaction completes, thus allowing the CPU to run other tasks.

An application task can queue multiple transactions, and the driver will automatically handle them one-by-one in the interrupt service routine (ISR). It allows the task to switch to other procedures until all the transactions complete.

Polling Transactions

Polling transactions do not use interrupts. The routine keeps polling the SPI Host' s status bit until the transaction is finished.

All the tasks that use interrupt transactions can be blocked by the queue. At this point, they will need to wait for the ISR to run twice before the transaction is finished. Polling transactions save time otherwise spent on queue handling and context switching, which results in smaller transaction intervals. The disadvantage is that the CPU is busy while these transactions are in progress.

The *spi_device_polling_end()* routine needs an overhead of at least 1 us to unblock other tasks when the transaction is finished. It is strongly recommended to wrap a series of polling transactions using the functions *spi_device_acquire_bus()* and *spi_device_release_bus()* to avoid the overhead. For more information, see *Bus Acquiring*.

Command and Address Phases

During the command and address phases, the members `cmd` and `addr` in the struct *spi_transaction_t* are sent to the bus, nothing is read at this time. The default lengths of the command and address phases are set in *spi_device_interface_config_t* by calling *spi_bus_add_device()*. If the flags `SPI_TRANS_VARIABLE_CMD` and `SPI_TRANS_VARIABLE_ADDR` in the member *spi_transaction_t::flags* are not set, the driver automatically sets the length of these phases to default values during Device initialization.

If the lengths of the command and address phases need to be variable, declare the struct *spi_transaction_ext_t*, set the flags `SPI_TRANS_VARIABLE_CMD` and/or `SPI_TRANS_VARIABLE_ADDR` in the member *spi_transaction_ext_t::base* and configure the rest of base as usual. Then the length of each phase will be equal to `command_bits` and `address_bits` set in the struct *spi_transaction_ext_t*.

Write and Read Phases

Normally, the data that needs to be transferred to or from a Device will be read from or written to a chunk of memory indicated by the members `rx_buffer` and `tx_buffer` of the structure *spi_transaction_t*. If DMA is enabled for transfers, the buffers are required to be:

1. Allocated in DMA-capable internal memory. If *external PSRAM is enabled*, this means using `pvPortMallocCaps(size, MALLOC_CAP_DMA)`.
2. 32-bit aligned (staring from a 32-bit boundary and having a length of multiples of 4 bytes).

If these requirements are not satisfied, the transaction efficiency will be affected due to the allocation and copying of temporary buffers.

注解: Half-duplex transactions with both read and write phases are not supported when using DMA. For details and workarounds, see *Known Issues*.

Bus Acquiring

Sometimes you might want to send SPI transactions exclusively and continuously so that it takes as little time as possible. For this, you can use bus acquiring, which helps to suspend transactions (both polling or interrupt) to other devices until the bus is released. To acquire and release a bus, use the functions *spi_device_acquire_bus()* and *spi_device_release_bus()*.

Driver Usage

- Initialize an SPI bus by calling the function *spi_bus_initialize()*. Make sure to set the correct I/O pins in the struct *spi_bus_config_t*. Set the signals that are not needed to `-1`.

- Register a Device connected to the bus with the driver by calling the function *spi_bus_add_device()*. Make sure to configure any timing requirements the device might need with the parameter `dev_config`. You should now have obtained the Device' s handle which will be used when sending a transaction to it.
- To interact with the Device, fill one or more *spi_transaction_t* structs with any transaction parameters required. Then send the structs either using a polling transaction or an interrupt transaction:
  - ***Interrupt*** Either queue all transactions by calling the function *spi_device_queue_trans()* and, at a later time, query the result using the function *spi_device_get_trans_result()*, or handle all requests synchronously by feeding them into *spi_device_transmit()*.
  - ***Polling*** Call the function *spi_device_polling_transmit()* to send polling transactions. Alternatively, if you want to insert something in between, send the transactions by using *spi_device_polling_start()* and *spi_device_polling_end()*.
- (Optional) To perform back-to-back transactions with a Device, call the function *spi_device_acquire_bus()* before sending transactions and *spi_device_release_bus()* after the transactions have been sent.
- (Optional) To unload the driver for a certain Device, call *spi_bus_remove_device()* with the Device handle as an argument.
- (Optional) To remove the driver for a bus, make sure no more drivers are attached and call *spi_bus_free()*.

The example code for the SPI master driver can be found in the peripherals/spi_master directory of ESP-IDF examples.

Transactions with Data Not Exceeding 32 Bits

When the transaction data size is equal to or less than 32 bits, it will be sub-optimal to allocate a buffer for the data. The data can be directly stored in the transaction struct instead. For transmitted data, it can be achieved by using the `tx_data` member and setting the `SPI_TRANS_USE_TXDATA` flag on the transmission. For received data, use `rx_data` and set `SPI_TRANS_USE_RXDATA`. In both cases, do not touch the `tx_buffer` or `rx_buffer` members, because they use the same memory locations as `tx_data` and `rx_data`.

Transactions with Integers Other Than `uint8_t`

An SPI Host reads and writes data into memory byte by byte. By default, data is sent with the most significant bit (MSB) first, as LSB first used in rare cases. If a value less than 8 bits needs to be sent, the bits should be written into memory in the MSB first manner.

For example, if `0b00010` needs to be sent, it should be written into a `uint8_t` variable, and the length for reading should be set to 5 bits. The Device will still receive 8 bits with 3 additional "random" bits, so the reading must be performed correctly.

On top of that, ESP32 is a little-endian chip, which means that the least significant byte of `uint16_t` and `uint32_t` variables is stored at the smallest address. Hence, if `uint16_t` is stored in memory, bits [7:0] are sent first, followed by bits [15:8].

For cases when the data to be transmitted has the size differing from `uint8_t` arrays, the following macros can be used to transform data to the format that can be sent by the SPI driver directly:

- *SPI_SWAP_DATA_TX* for data to be transmitted
- *SPI_SWAP_DATA_RX* for data received

Notes on Sending Mixed Transactions to the Same Device

To reduce coding complexity, send only one type of transactions (interrupt or polling) to one Device. However, you still can send both interrupt and polling transactions alternately. The notes below explain how to do this.

The polling transactions should be initiated only after all the polling and interrupt transactions are finished.

Since an unfinished polling transaction blocks other transactions, please do not forget to call the function *spi_device_polling_end()* after *spi_device_polling_start()* to allow other transactions or to allow other Devices to use the bus. Remember that if there is no need to switch to other tasks during your polling transaction, you can initiate a transaction with *spi_device_polling_transmit()* so that it will be ended automatically.

In-flight polling transactions are disturbed by the ISR operation to accommodate interrupt transactions. Always make sure that all the interrupt transactions sent to the ISR are finished before you call *spi_device_polling_start()*. To do that, you can keep calling *spi_device_get_trans_result()* until all the transactions are returned.

To have better control of the calling sequence of functions, send mixed transactions to the same Device only within a single task.

GPIO Matrix and IO_MUX

Most of ESP32' s peripheral signals have direct connection to their dedicated IO_MUX pins. However, the signals can also be routed to any other available pins using the less direct GPIO matrix. If at least one signal is routed through the GPIO matrix, then all signals will be routed through it.

The GPIO matrix introduces flexibility of routing but also brings the following disadvantages:

- Increases the input delay of the MISO signal, which makes MISO setup time violations more likely. If SPI needs to operate at high speeds, use dedicated IO_MUX pins.
- Allows signals with clock frequencies only up to 40 MHz, as opposed to 80 MHz if IO_MUX pins are used.

注解: For more details about the influence of the MISO input delay on the maximum clock frequency, see *Timing Considerations*.

The IO_MUX pins for SPI buses are given below.

Pin Name	SPI2	SPI3
	GPIO Number	
CS0*	15	5
SCLK	14	18
MISO	12	19
MOSI	13	23
QUADWP	2	22
QUADHD	4	21

- Only the first Device attached to the bus can use the CS0 pin.

Transfer Speed Considerations

There are three factors limiting the transfer speed:

- Transaction interval
- SPI clock frequency
- Cache miss of SPI functions, including callbacks

The main parameter that determines the transfer speed for large transactions is clock frequency. For multiple small transactions, the transfer speed is mostly determined by the length of transaction intervals.

Transaction Interval

Transaction interval is the time that software requires to set up SPI peripheral registers and to copy data to FIFOs, or to set up DMA links.

Interrupt transactions allow appending extra overhead to accommodate the cost of FreeRTOS queues and the time needed for switching between tasks and the ISR.

For **interrupt transactions**, the CPU can switch to other tasks when a transaction is in progress. This saves the CPU time but increases the interval. See *Interrupt Transactions*. For **polling transactions**, it does not block the task but allows to do polling when the transaction is in progress. For more information, see *Polling Transactions*.

If DMA is enabled, setting up the linked list requires about 2 us per transaction. When a master is transferring data, it automatically reads the data from the linked list. If DMA is not enabled, the CPU has

to write and read each byte from the FIFO by itself. Usually, this is faster than 2 us, but the transaction length is limited to 64 bytes for both write and read.

Typical transaction interval timings for one byte of data are given below.

	Typical Transaction Time (us)	
	Interrupt	Polling
DMA	24	8
No DMA	22	7

SPI Clock Frequency

Transferring each byte takes eight times the clock period *8/fspi*. If the clock frequency is too high, the use of some functions might be limited. See *Timing Considerations*.

Cache Miss

The default config puts only the ISR into the IRAM. Other SPI related functions, including the driver itself and the callback, might suffer from the cache miss and will need to wait until the code is read from the flash. Select *CONFIG_SPI_MASTER_IN_IRAM* to put the whole SPI driver into IRAM and put the entire callback(s) and its callee functions into IRAM to prevent cache miss.

For an interrupt transaction, the overall cost is *20+8n/Fspi[MHz]* [us] for n bytes transferred in one transaction. Hence, the transferring speed is: *n/(20+8n/Fspi)*. An example of transferring speed at 8 MHz clock speed is given in the following table.

Frequency (MHz)	Transaction Interval (us)	Transaction Length (bytes)	Total Time (us)	Total Speed (KBps)
8	25	1	26	38.5
8	25	8	33	242.4
8	25	16	41	490.2
8	25	64	89	719.1
8	25	128	153	836.6

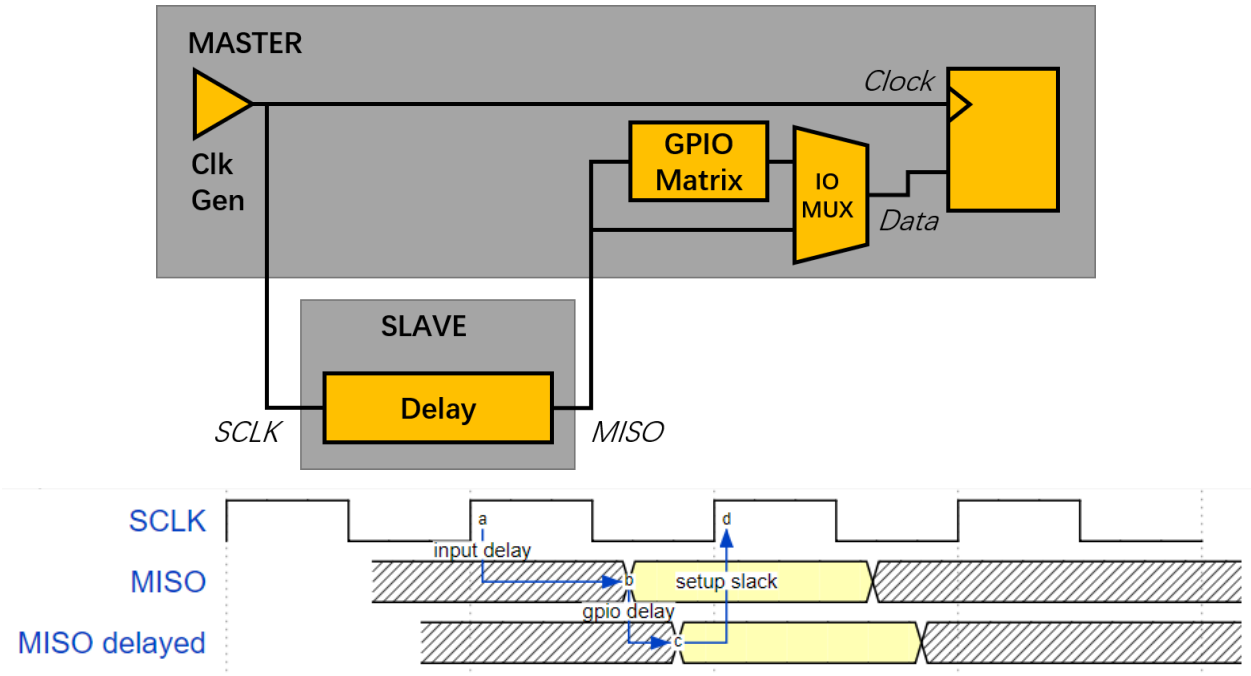
When a transaction length is short, the cost of transaction interval is high. If possible, try to squash several short transactions into one transaction to achieve a higher transfer speed.

Please note that the ISR is disabled during flash operation by default. To keep sending transactions during flash operations, enable *CONFIG_SPI_MASTER_ISR_IN_IRAM* and set `ESP_INTR_FLAG_IRAM` in the member *spi_bus_config_t::intr_flags*. In this case, all the transactions queued before starting flash operations will be handled by the ISR in parallel. Also note that the callback of each Device and their callee

functions should be in IRAM, or your callback will crash due to cache miss. For more details, see *IRAM 安全中断处理程序*.

Timing Considerations

As shown in the figure below, there is a delay on the MISO line after the SCLK launch edge and before the signal is latched by the internal register. As a result, the MISO pin setup time is the limiting factor for the SPI clock speed. When the delay is too long, the setup slack is < 0, and the setup timing requirement is violated, which results in the failure to perform the reading correctly.



The maximum allowed frequency is dependent on:

- `input_delay_ns` - maximum data valid time on the MISO bus after a clock cycle on SCLK starts
- If the IO_MUX pin or the GPIO Matrix is used

When the GPIO matrix is used, the maximum allowed frequency is reduced to about 33~77% in comparison to the existing *input delay*. To retain a higher frequency, you have to use the IO_MUX pins or the *dummy bit workaround*. You can obtain the maximum reading frequency of the master by using the function *`spi_get_freq_limit()`*.

Dummy bit workaround: Dummy clocks, during which the Host does not read data, can be inserted before the read phase begins. The Device still sees the dummy clocks and sends out data, but the Host does not read until the read phase comes. This compensates for the lack of the MISO setup time required by the Host and allows the Host to do reading at a higher frequency.

In the ideal case, if the Device is so fast that the input delay is shorter than an APB clock cycle - 12.5 ns - the maximum frequency at which the Host can read (or read and write) in different conditions is as follows:

Frequency Limit (MHz)		Dummy Bits Used By Driver	Comments
GPIO matrix	IO_MUX pins		
26.6	80	No	
40	–	Yes	Half-duplex, no DMA allowed

If the Host only writes data, the *dummy bit workaround* and the frequency check can be disabled by setting the bit *SPI_DEVICE_NO_DUMMY* in the member *spi_device_interface_config_t::flags*. When disabled, the output frequency can be 80MHz, even if the GPIO matrix is used.

spi_device_interface_config_t::flags

The SPI master driver can work even if the `input_delay_ns` in the structure *spi_device_interface_config_t* is set to 0. However, setting an accurate value helps to:

- Calculate the frequency limit for full-duplex transactions
- Compensate the timing correctly with dummy bits for half-duplex transactions

You can approximate the maximum data valid time after the launch edge of SPI clocks by checking the statistics in the AC characteristics chapter of your Device' s specification or measure the time on an oscilloscope or logic analyzer.

Please note that the actual PCB layout design and the excessive loads may increase the input delay. It means that non-optimal wiring and/or a load capacitor on the bus will most likely lead to the input delay values exceeding the values given in the Device specification or measured while the bus is floating.

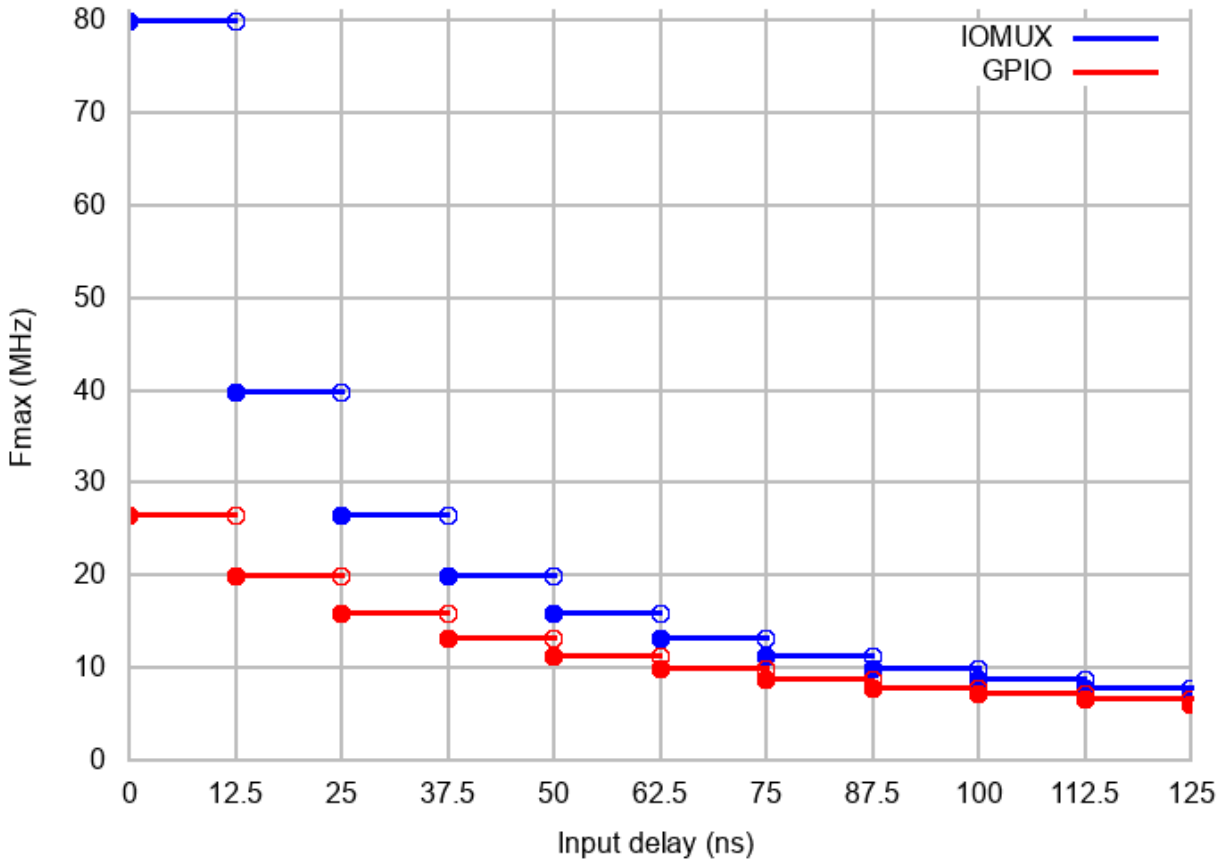
Some typical delay values are shown in the following table.

Device	Input delay (ns)
Ideal Device	0
ESP32 slave using IO_MUX*	50
ESP32 slave using GPIO_MUX*	75
ESP32' s slave device is on a different physical chip.	

The MISO path delay (valid time) consists of a slave' s *input delay* plus master' s *GPIO matrix delay*. This delay determines the frequency limit above which full-duplex transfers will not work as well as the dummy bits used in the half-duplex transactions. The frequency limit is:

Freq limit [MHz] = 80 / (floor(MISO delay[ns]/12.5) + 1)

The figure below shows the relationship between frequency limit and input delay. Two extra APB clock cycle periods should be added to the MISO delay if the master uses the GPIO matrix.



Corresponding frequency limits for different Devices with different *input delay* times are shown in the table below.

Master	Input delay (ns)	MISO path delay (ns)	Freq. limit (MHz)
IO_MUX (0ns)	0	0	80
	50	50	16
	75	75	11.43
GPIO (25ns)	0	25	26.67
	50	75	11.43
	75	100	8.89

Known Issues

1. Half-duplex transactions are not compatible with DMA when both writing and reading phases are used.

   If such transactions are required, you have to use one of the alternative solutions:

   1. Use full-duplex transactions instead.

2. Disable DMA by setting the bus initialization function' s last parameter to 0 as follows: `ret=spi_bus_initialize(VSPI_HOST, &buscfg, 0);`

   This can prohibit you from transmitting and receiving data longer than 64 bytes.

3. Try using the command and address fields to replace the write phase.

2. Full-duplex transactions are not compatible with the *dummy bit workaround*, hence the frequency is limited. See *dummy bit speed-up workaround*.
3. `cs_ena_pretrans` is not compatible with the command and address phases of full-duplex transactions.

Application Example

The code example for displaying graphics on an ESP32-WROVER-KIT' s 320x240 LCD screen can be found in the peripherals/spi_master directory of ESP-IDF examples.

API Reference - SPI Common

Header File

- soc/include/hal/spi_types.h

Enumerations

enum spi_host_device_t

Enum with the three SPI peripherals that are software-accessible in it.

Values:

SPI1_HOST =0

SPI1.

SPI2_HOST =1

SPI2.

SPI3_HOST =2

SPI3.

Header File

- driver/include/driver/spi_common.h

Functions

esp_err_t **spi_bus_initialize**(*spi_host_device_t host*, **const** *spi_bus_config_t* **bus_config*, int *dma_chan*)

Initialize a SPI bus.

Warning For now, only supports HSPI and VSPI.

Warning If a DMA channel is selected, any transmit and receive buffer used should be allocated in DMA-capable memory.

Warning The ISR of SPI is always executed on the core which calls this function. Never starve the ISR on this core or the SPI transactions will not be handled.

Return

- ESP_ERR_INVALID_ARG if configuration is invalid
- ESP_ERR_INVALID_STATE if host already is in use
- ESP_ERR_NO_MEM if out of memory
- ESP_OK on success

Parameters

- `host`: SPI peripheral that controls this bus
- `bus_config`: Pointer to a *spi_bus_config_t* struct specifying how the host should be initialized
- `dma_chan`: Either channel 1 or 2, or 0 in the case when no DMA is required. Selecting a DMA channel for a SPI bus allows transfers on the bus to have sizes only limited by the amount of internal memory. Selecting no DMA channel (by passing the value 0) limits the amount of bytes transfered to a maximum of 64. Set to 0 if only the SPI flash uses this bus.

esp_err_t **spi_bus_free**(*spi_host_device_t host*)

Free a SPI bus.

Warning In order for this to succeed, all devices have to be removed first.

Return

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_ERR_INVALID_STATE if not all devices on the bus are freed
- ESP_OK on success

Parameters

- `host`: SPI peripheral to free

Structures

struct spi_bus_config_t

This is a configuration structure for a SPI bus.

You can use this structure to specify the GPIO pins of the bus. Normally, the driver will use the GPIO matrix to route the signals. An exception is made when all signals either can be routed through the IO_MUX or are -1. In that case, the IO_MUX is used, allowing for >40MHz speeds.

Note Be advised that the slave driver does not use the quadwp/quadhd lines and fields in *spi_bus_config_t* refering to these lines will be ignored and can thus safely be left uninitialized.

Public Members

int **mosi_io_num**

GPIO pin for Master Out Slave In (=spi_d) signal, or -1 if not used.

int **miso_io_num**

GPIO pin for Master In Slave Out (=spi_q) signal, or -1 if not used.

int **sclk_io_num**

GPIO pin for Spi CLocK signal, or -1 if not used.

int **quadwp_io_num**

GPIO pin for WP (Write Protect) signal which is used as D2 in 4-bit communication modes, or -1 if not used.

int **quadhd_io_num**

GPIO pin for HD (HolD) signal which is used as D3 in 4-bit communication modes, or -1 if not used.

int **max_transfer_sz**

Maximum transfer size, in bytes. Defaults to 4094 if 0.

uint32_t **flags**

Abilities of bus to be checked by the driver. Or-ed value of `SPICOMMON_BUSFLAG_*` flags.

int **intr_flags**

Interrupt flag for the bus to set the priority, and IRAM attribute, see `esp_intr_alloc.h`. Note that the EDGE, INTRDISABLED attribute are ignored by the driver. Note that if ESP_INTR_FLAG_IRAM is set, ALL the callbacks of the driver, and their callee functions, should be put in the IRAM.

Macros

`SPI_MAX_DMA_LEN`

SPI_SWAP_DATA_TX(DATA, LEN)

Transform unsigned integer of length <= 32 bits to the format which can be sent by the SPI driver directly.

E.g. to send 9 bits of data, you can:

```
uint16_t data = SPI_SWAP_DATA_TX(0x145, 9);
```

Then points tx_buffer to `&data`.

Parameters

- `DATA`: Data to be sent, can be uint8_t, uint16_t or uint32_t.
- `LEN`: Length of data to be sent, since the SPI peripheral sends from the MSB, this helps to shift the data to the MSB.

SPI_SWAP_DATA_RX(DATA, LEN)

Transform received data of length <= 32 bits to the format of an unsigned integer.

E.g. to transform the data of 15 bits placed in a 4-byte array to integer:

```
uint16_t data = SPI_SWAP_DATA_RX(*(uint32_t*)t->rx_data, 15);
```

Parameters

- `DATA`: Data to be rearranged, can be uint8_t, uint16_t or uint32_t.
- `LEN`: Length of data received, since the SPI peripheral writes from the MSB, this helps to shift the data to the LSB.

SPICOMMON_BUSFLAG_SLAVE

Initialize I/O in slave mode.

SPICOMMON_BUSFLAG_MASTER

Initialize I/O in master mode.

SPICOMMON_BUSFLAG_IOMUX_PINS

Check using iomux pins. Or indicates the pins are configured through the IO mux rather than GPIO matrix.

SPICOMMON_BUSFLAG_SCLK

Check existing of SCLK pin. Or indicates CLK line initialized.

SPICOMMON_BUSFLAG_MISO

Check existing of MISO pin. Or indicates MISO line initialized.

SPICOMMON_BUSFLAG_MOSI

Check existing of MOSI pin. Or indicates CLK line initialized.

`SPICOMMON_BUSFLAG_DUAL`
: Check MOSI and MISO pins can output. Or indicates bus able to work under DIO mode.

`SPICOMMON_BUSFLAG_WPHD`
: Check existing of WP and HD pins. Or indicates WP & HD pins initialized.

`SPICOMMON_BUSFLAG_QUAD`
: Check existing of MOSI/MISO/WP/HD pins as output. Or indicates bus able to work under QIO mode.

`SPICOMMON_BUSFLAG_NATIVE_PINS`

API Reference - SPI Master

Header File

- driver/include/driver/spi_master.h

Functions

esp_err_t **spi_bus_add_device**(*spi_host_device_t host*, **const** *spi_device_interface_config_t *dev_config*, *spi_device_handle_t *handle*)

Allocate a device on a SPI bus.

This initializes the internal structures for a device, plus allocates a CS pin on the indicated SPI master peripheral and routes it to the indicated GPIO. All SPI master devices have three CS pins and can thus control up to three devices.

Note While in general, speeds up to 80MHz on the dedicated SPI pins and 40MHz on GPIO-matrix-routed pins are supported, full-duplex transfers routed over the GPIO matrix only support speeds up to 26MHz.

Return

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_ERR_NOT_FOUND if host doesn't have any free CS slots
- ESP_ERR_NO_MEM if out of memory
- ESP_OK on success

Parameters

- `host`: SPI peripheral to allocate device on
- `dev_config`: SPI interface protocol config for the device
- `handle`: Pointer to variable to hold the device handle

esp_err_t **spi_bus_remove_device**(*spi_device_handle_t handle*)

Remove a device from the SPI bus.

Return

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_ERR_INVALID_STATE if device already is freed
- ESP_OK on success

Parameters

- `handle`: Device handle to free

esp_err_t **spi_device_queue_trans**(*spi_device_handle_t handle*, *spi_transaction_t* **trans_desc*, TickType_t *ticks_to_wait*)

Queue a SPI transaction for interrupt transaction execution. Get the result by `spi_device_get_trans_result`.

Note Normally a device cannot start (queue) polling and interrupt transactions simultaneously.

Return

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_ERR_TIMEOUT if there was no room in the queue before ticks_to_wait expired
- ESP_ERR_NO_MEM if allocating DMA-capable temporary buffer failed
- ESP_ERR_INVALID_STATE if previous transactions are not finished
- ESP_OK on success

Parameters

- `handle`: Device handle obtained using spi_host_add_dev
- `trans_desc`: Description of transaction to execute
- `ticks_to_wait`: Ticks to wait until there' s room in the queue; use portMAX_DELAY to never time out.

esp_err_t **spi_device_get_trans_result**(*spi_device_handle_t handle*, *spi_transaction_t* ***trans_desc*, TickType_t *ticks_to_wait*)

Get the result of a SPI transaction queued earlier by `spi_device_queue_trans`.

This routine will wait until a transaction to the given device succesfully completed. It will then return the description of the completed transaction so software can inspect the result and e.g. free the memory or re-use the buffers.

Return

- ESP_ERR_INVALID_ARG if parameter is invalid

- ESP_ERR_TIMEOUT if there was no completed transaction before ticks_to_wait expired
- ESP_OK on success

Parameters

- `handle`: Device handle obtained using spi_host_add_dev
- `trans_desc`: Pointer to variable able to contain a pointer to the description of the transaction that is executed. The descriptor should not be modified until the descriptor is returned by spi_device_get_trans_result.
- `ticks_to_wait`: Ticks to wait until there' s a returned item; use portMAX_DELAY to never time out.

esp_err_t **spi_device_transmit**(*spi_device_handle_t handle*, *spi_transaction_t* **trans_desc*)

Send a SPI transaction, wait for it to complete, and return the result.

This function is the equivalent of calling spi_device_queue_trans() followed by spi_device_get_trans_result(). Do not use this when there is still a transaction separately queued (started) from spi_device_queue_trans() or polling_start/transmit that hasn' t been finalized.

Note This function is not thread safe when multiple tasks access the same SPI device. Normally a device cannot start (queue) polling and interrupt transactions simutanuously.

Return

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_OK on success

Parameters

- `handle`: Device handle obtained using spi_host_add_dev
- `trans_desc`: Description of transaction to execute

esp_err_t **spi_device_polling_start**(*spi_device_handle_t handle*, *spi_transaction_t* **trans_desc*, TickType_t *ticks_to_wait*)

Immediately start a polling transaction.

Note Normally a device cannot start (queue) polling and interrupt transactions simutanuously. Moreover, a device cannot start a new polling transaction if another polling transaction is not finished.

Return

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_ERR_TIMEOUT if the device cannot get control of the bus before `ticks_to_wait` expired
- ESP_ERR_NO_MEM if allocating DMA-capable temporary buffer failed

- ESP_ERR_INVALID_STATE if previous transactions are not finished
- ESP_OK on success

Parameters

- `handle`: Device handle obtained using spi_host_add_dev
- `trans_desc`: Description of transaction to execute
- `ticks_to_wait`: Ticks to wait until there' s room in the queue; currently only portMAX_DELAY is supported.

esp_err_t **spi_device_polling_end**(*spi_device_handle_t handle*, TickType_t *ticks_to_wait*)

Poll until the polling transaction ends.

This routine will not return until the transaction to the given device has succesfully completed. The task is not blocked, but actively busy-spins for the transaction to be completed.

Return

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_ERR_TIMEOUT if the transaction cannot finish before ticks_to_wait expired
- ESP_OK on success

Parameters

- `handle`: Device handle obtained using spi_host_add_dev
- `ticks_to_wait`: Ticks to wait until there' s a returned item; use portMAX_DELAY to never time out.

esp_err_t **spi_device_polling_transmit**(*spi_device_handle_t handle*, *spi_transaction_t* **trans_desc*)

Send a polling transaction, wait for it to complete, and return the result.

This function is the equivalent of calling spi_device_polling_start() followed by spi_device_polling_end(). Do not use this when there is still a transaction that hasn' t been finalized.

Note This function is not thread safe when multiple tasks access the same SPI device. Normally a device cannot start (queue) polling and interrupt transactions simutanuously.

Return

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_OK on success

Parameters

- `handle`: Device handle obtained using spi_host_add_dev

- `trans_desc`: Description of transaction to execute

esp_err_t **spi_device_acquire_bus**(*spi_device_handle_t device*, TickType_t *wait*)

Occupy the SPI bus for a device to do continuous transactions.

Transactions to all other devices will be put off until `spi_device_release_bus` is called.

Note The function will wait until all the existing transactions have been sent.

Return

- ESP_ERR_INVALID_ARG : `wait` is not set to portMAX_DELAY.
- ESP_OK : Success.

Parameters

- `device`: The device to occupy the bus.
- `wait`: Time to wait before the the bus is occupied by the device. Currently MUST set to portMAX_DELAY.

void **spi_device_release_bus**(*spi_device_handle_t dev*)

Release the SPI bus occupied by the device. All other devices can start sending transactions.

Parameters

- `dev`: The device to release the bus.

int **spi_cal_clock**(int *fapb*, int *hz*, int *duty_cycle*, uint32_t **reg_o*)

Calculate the working frequency that is most close to desired frequency, and also the register value.

Parameters

- `fapb`: The frequency of apb clock, should be `APB_CLK_FREQ`.
- `hz`: Desired working frequency
- `duty_cycle`: Duty cycle of the spi clock
- `reg_o`: Output of value to be set in clock register, or NULL if not needed.

Return Actual working frequency that most fit.

int **spi_get_actual_clock**(int *fapb*, int *hz*, int *duty_cycle*)

Calculate the working frequency that is most close to desired frequency.

Return Actual working frequency that most fit.

Parameters

- `fapb`: The frequency of apb clock, should be `APB_CLK_FREQ`.

- `hz`: Desired working frequency
- `duty_cycle`: Duty cycle of the spi clock

void **spi_get_timing**(bool *gpio_is_used*, int *input_delay_ns*, int *eff_clk*, int **dummy_o*, int **cycles_remain_o*)

Calculate the timing settings of specified frequency and settings.

Note If **dummy_o* is not zero, it means dummy bits should be applied in half duplex mode, and full duplex mode may not work.

Parameters

- `gpio_is_used`: True if using GPIO matrix, or False if iomux pins are used.
- `input_delay_ns`: Input delay from SCLK launch edge to MISO data valid.
- `eff_clk`: Effective clock frequency (in Hz) from spi_cal_clock.
- `dummy_o`: Address of dummy bits used output. Set to NULL if not needed.
- `cycles_remain_o`: Address of cycles remaining (after dummy bits are used) output.
  - -1 If too many cycles remaining, suggest to compensate half a clock.
  - 0 If no remaining cycles or dummy bits are not used.
  - positive value: cycles suggest to compensate.

int **spi_get_freq_limit**(bool *gpio_is_used*, int *input_delay_ns*)

Get the frequency limit of current configurations. SPI master working at this limit is OK, while above the limit, full duplex mode and DMA will not work, and dummy bits will be aplied in the half duplex mode.

Return Frequency limit of current configurations.

Parameters

- `gpio_is_used`: True if using GPIO matrix, or False if native pins are used.
- `input_delay_ns`: Input delay from SCLK launch edge to MISO data valid.

Structures

struct spi_device_interface_config_t

This is a configuration for a SPI slave device that is connected to one of the SPI buses.

Public Members

uint8_t **command_bits**

Default amount of bits in command phase (0-16), used when `SPI_TRANS_VARIABLE_CMD` is not used, otherwise ignored.

uint8_t **address_bits**

Default amount of bits in address phase (0-64), used when `SPI_TRANS_VARIABLE_ADDR` is not used, otherwise ignored.

uint8_t **dummy_bits**

Amount of dummy bits to insert between address and data phase.

uint8_t **mode**

SPI mode (0-3)

uint16_t **duty_cycle_pos**

Duty cycle of positive clock, in 1/256th increments (128 = 50%/50% duty). Setting this to 0 (=not setting it) is equivalent to setting this to 128.

uint16_t **cs_ena_pretrans**

Amount of SPI bit-cycles the cs should be activated before the transmission (0-16). This only works on half-duplex transactions.

uint8_t **cs_ena_posttrans**

Amount of SPI bit-cycles the cs should stay active after the transmission (0-16)

int **clock_speed_hz**

Clock speed, divisors of 80MHz, in Hz. See `SPI_MASTER_FREQ_*`.

int **input_delay_ns**

Maximum data valid time of slave. The time required between SCLK and MISO valid, including the possible clock delay from slave to master. The driver uses this value to give an extra delay before the MISO is ready on the line. Leave at 0 unless you know you need a delay. For better timing performance at high frequency (over 8MHz), it' s suggest to have the right value.

int **spics_io_num**

CS GPIO pin for this device, or -1 if not used.

uint32_t **flags**

Bitwise OR of SPI_DEVICE_* flags.

int **queue_size**

Transaction queue size. This sets how many transactions can be 'in the air' (queued using spi_device_queue_trans but not yet finished using spi_device_get_trans_result) at the same time.

transaction_cb_t **pre_cb**

Callback to be called before a transmission is started.

This callback is called within interrupt context should be in IRAM for best performance, see "Transferring Speed" section in the SPI Master documentation for full details. If not, the callback may crash during flash operation when the driver is initialized with ESP_INTR_FLAG_IRAM.

transaction_cb_t **post_cb**

Callback to be called after a transmission has completed.

This callback is called within interrupt context should be in IRAM for best performance, see "Transferring Speed" section in the SPI Master documentation for full details. If not, the callback may crash during flash operation when the driver is initialized with ESP_INTR_FLAG_IRAM.

struct spi_transaction_t

This structure describes one SPI transaction. The descriptor should not be modified until the transaction finishes.

Public Members

uint32_t **flags**

Bitwise OR of SPI_TRANS_* flags.

uint16_t **cmd**

Command data, of which the length is set in the `command_bits` of *spi_device_interface_config_t*.

NOTE: this field, used to be "command" in ESP-IDF 2.1 and before, is re-written to be used in a new way in ESP-IDF 3.0.

Example: write 0x0123 and command_bits=12 to send command 0x12, 0x3_ (in previous version, you may have to write 0x3_12).

uint64_t **addr**

Address data, of which the length is set in the `address_bits` of *spi_device_interface_config_t*.

NOTE: this field, used to be "address" in ESP-IDF 2.1 and before, is re-written to be used in a new way in ESP-IDF3.0.

Example: write 0x123400 and address_bits=24 to send address of 0x12, 0x34, 0x00 (in previous version, you may have to write 0x12340000).

size_t **length**

Total data length, in bits.

size_t **rxlength**

Total data length received, should be not greater than `length` in full-duplex mode (0 defaults this to the value of `length`).

void ***user**

User-defined variable. Can be used to store eg transaction ID.

const void ***tx_buffer**

Pointer to transmit buffer, or NULL for no MOSI phase.

uint8_t **tx_data**[4]

If SPI_TRANS_USE_TXDATA is set, data set here is sent directly from this variable.

void ***rx_buffer**

Pointer to receive buffer, or NULL for no MISO phase. Written by 4 bytes-unit if DMA is used.

uint8_t **rx_data**[4]

If SPI_TRANS_USE_RXDATA is set, data is received directly to this variable.

struct spi_transaction_ext_t

This struct is for SPI transactions which may change their address and command length. Please do set the flags in base to `SPI_TRANS_VARIABLE_CMD_ADR` to use the bit length here.

Public Members

struct *spi_transaction_t* **base**

Transaction data, so that pointer to *spi_transaction_t* can be converted into *spi_transaction_ext_t*.

uint8_t **command_bits**

The command length in this transaction, in bits.

uint8_t **address_bits**

The address length in this transaction, in bits.

uint8_t **dummy_bits**

The dummy length in this transaction, in bits.

Macros

SPI_DEVICE_TXBIT_LSBFIRST

Transmit command/address/data LSB first instead of the default MSB first.

SPI master clock is divided by 80MHz apb clock. Below defines are example frequencies, and are accurate. Be free to specify a random frequency, it will be rounded to closest frequency (to macros below if above 8MHz). 8MHz

SPI_DEVICE_RXBIT_LSBFIRST

Receive data LSB first instead of the default MSB first.

SPI_DEVICE_BIT_LSBFIRST

Transmit and receive LSB first.

SPI_DEVICE_3WIRE

Use MOSI (=spid) for both sending and receiving data.

SPI_DEVICE_POSITIVE_CS

Make CS positive during a transaction instead of negative.

`SPI_DEVICE_HALFDUPLEX`
: Transmit data before receiving it, instead of simultaneously.

`SPI_DEVICE_CLK_AS_CS`
: Output clock on CS line if CS is active.

`SPI_DEVICE_NO_DUMMY`
: There are timing issue when reading at high frequency (the frequency is related to whether iomux pins are used, valid time after slave sees the clock).

  - In half-duplex mode, the driver automatically inserts dummy bits before reading phase to fix the timing issue. Set this flag to disable this feature.
  - In full-duplex mode, however, the hardware cannot use dummy bits, so there is no way to prevent data being read from getting corrupted. Set this flag to confirm that you' re going to work with output only, or read without dummy bits at your own risk.

`SPI_DEVICE_DDRCLK`

`SPI_TRANS_MODE_DIO`
: Transmit/receive data in 2-bit mode.

`SPI_TRANS_MODE_QIO`
: Transmit/receive data in 4-bit mode.

`SPI_TRANS_USE_RXDATA`
: Receive into rx_data member of *spi_transaction_t* instead into memory at rx_buffer.

`SPI_TRANS_USE_TXDATA`
: Transmit tx_data member of *spi_transaction_t* instead of data at tx_buffer. Do not set tx_buffer when using this.

`SPI_TRANS_MODE_DIOQIO_ADDR`
: Also transmit address in mode selected by SPI_MODE_DIO/SPI_MODE_QIO.

`SPI_TRANS_VARIABLE_CMD`
: Use the `command_bits` in *spi_transaction_ext_t* rather than default value in *spi_device_interface_config_t*.

`SPI_TRANS_VARIABLE_ADDR`
: Use the `address_bits` in *spi_transaction_ext_t* rather than default value in *spi_device_interface_config_t*.

`SPI_TRANS_VARIABLE_DUMMY`
: Use the `dummy_bits` in *spi_transaction_ext_t* rather than default value in *spi_device_interface_config_t*.

`SPI_TRANS_SET_CD`
: Set the CD pin.

Type Definitions

typedef struct *spi_transaction_t* **spi_transaction_t**

typedef void (***transaction_cb_t**)(*spi_transaction_t* *trans)

typedef struct spi_device_t ***spi_device_handle_t**

Handle for a device on a SPI bus.

2.3.16 SPI Slave Driver

SPI Slave driver is a program that controls ESP32' s SPI peripherals while they function as slaves.

Overview of ESP32' s SPI peripherals

ESP32 integrates two general purpose SPI controllers which can be used as slave nodes driven by an off-chip SPI master

- SPI2, sometimes referred to as HSPI
- SPI3, sometimes referred to as VSPI

SPI2 and SPI3 have independent signal buses with the same respective names.

Terminology

The terms used in relation to the SPI slave driver are given in the table below.

Term	Definition
Host	The SPI controller peripheral external to ESP32 that initiates SPI transmissions over the bus, and acts as an SPI Master.
Device	SPI slave device, in this case the SPI2 and SPI3 controllers. Each Device shares the MOSI, MISO and SCLK signals but is only active on the bus when the Host asserts the Device' s individual CS line.
Bus	A signal bus, common to all Devices connected to one Host. In general, a bus includes the following lines: MISO, MOSI, SCLK, one or more CS lines, and, optionally, QUADWP and QUADHD. So Devices are connected to the same lines, with the exception that each Device has its own CS line. Several Devices can also share one CS line if connected in the daisy-chain manner.
• **MISO**	Master In, Slave Out, a.k.a. Q. Data transmission from a Device to Host.
• **MOSI**	Master In, Slave Out, a.k.a. D. Data transmission from a Host to Device.
• **SCLK**	Serial Clock. Oscillating signal generated by a Host that keeps the transmission of data bits in sync.
• **CS**	Chip Select. Allows a Host to select individual Device(s) connected to the bus in order to send or receive data.
• **QUADWP**	Write Protect signal. Only used for 4-bit (qio/qout) transactions.
• **QUADHD**	Hold signal. Only used for 4-bit (qio/qout) transactions.
• **Assertion**	The action of activating a line. The opposite action of returning the line back to inactive (back to idle) is called *de-assertion*.
Transaction	One instance of a Host asserting a CS line, transferring data to and from a Device, and de-asserting the CS line. Transactions are atomic, which means they can never be interrupted by another transaction.
Launch edge	Edge of the clock at which the source register *launches* the signal onto the line.
Latch edge	Edge of the clock at which the destination register *latches in* the signal.

Driver Features

The SPI slave driver allows using the SPI2 and/or SPI3 peripherals as full-duplex Devices. The driver can send/receive transactions up to 64 bytes in length, or utilize DMA to send/receive longer transactions. However, there are some *known issues* related to DMA.

SPI Transactions

A full-duplex SPI transaction begins when the Host asserts the CS line and starts sending out clock pulses on the SCLK line. Every clock pulse, a data bit is shifted from the Host to the Device on the MOSI line and back on the MISO line at the same time. At the end of the transaction, the Host de-asserts the CS line.

The attributes of a transaction are determined by the configuration structure for an SPI host acting as a slave device *spi_slave_interface_config_t*, and transaction configuration structure *spi_slave_transaction_t*.

As not every transaction requires both writing and reading data, you have a choice to configure the *spi_transaction_t* structure for TX only, RX only, or TX and RX transactions. If *spi_slave_transaction_t::rx_buffer* is set to NULL, the read phase will be skipped. If *spi_slave_transaction_t::tx_buffer* is set to NULL, the write phase will be skipped.

注解: A Host should not start a transaction before its Device is ready for receiving data. It is recommended to use another GPIO pin for a handshake signal to sync the Devices. For more details, see *Transaction Interval*.

Driver Usage

- Initialize an SPI peripheral as a Device by calling the function cpp:func:*spi_slave_initialize*. Make sure to set the correct I/O pins in the struct `bus_config`. Set the unused signals to `-1`. If transactions will be longer than 32 bytes, allow a DMA channel 1 or 2 by setting the parameter `dma_chan` to `1` or `2` respectively. Otherwise, set `dma_chan` to `0`.
- Before initiating transactions, fill one or more *spi_slave_transaction_t* structs with the transaction parameters required. Either queue all transactions by calling the function *spi_slave_queue_trans()* and, at a later time, query the result by using the function *spi_slave_get_trans_result()*, or handle all requests individually by feeding them into *spi_slave_transmit()*. The latter two functions will be blocked until the Host has initiated and finished a transaction, causing the queued data to be sent and received.
- (Optional) To unload the SPI slave driver, call *spi_slave_free()*.

Transaction Data and Master/Slave Length Mismatches

Normally, the data that needs to be transferred to or from a Device is read or written to a chunk of memory indicated by the `rx_buffer` and `tx_buffer` members of the *spi_transaction_t* structure. The SPI driver can be configured to use DMA for transfers, in which case these buffers must be allocated in DMA-capable memory using `pvPortMallocCaps(size, MALLOC_CAP_DMA)`.

The amount of data that the driver can read or write to the buffers is limited by the member *spi_transaction_t::length*. However, this member does not define the actual length of an SPI transaction. A transaction' s length is determined by a Host which drives the clock and CS lines. The actual length of the transmission can be read only after a transaction is finished from the member *spi_slave_transaction_t::trans_len*.

If the length of the transmission is greater than the buffer length, only the initial number of bits specified in the `length` member will be sent and received. In this case, `trans_len` is set to `length` instead of the actual transaction length. To meet the actual transaction length requirements, set `length` to a value greater than the maximum `trans_len` expected. If the transmission length is shorter than the buffer length, only the data equal to the length of the buffer will be transmitted.

警告: The ESP32 DMA hardware has a limit to the number of bytes sent by a Host and received by a Device. The transaction length must be longer than 8 bytes and a multiple of 4 bytes; otherwise, the SPI hardware might fail to receive the last 1 to 7 bytes.

GPIO Matrix and IO_MUX

Most of ESP32' s peripheral signals have direct connection to their dedicated IO_MUX pins. However, the signals can also be routed to any other available pins using the less direct GPIO matrix.

If at least one signal is routed through the GPIO matrix, then all signals will be routed through it. The GPIO matrix samples all signals at 80 MHz and transmits them between the GPIO and the peripheral.

If the driver is configured so that all SPI signals are either routed to their dedicated IO_MUX pins or are not connected at all, the GPIO matrix will be bypassed.

The GPIO matrix introduces flexibility of routing but also increases the input delay of the MISO signal, which makes MISO setup time violations more likely. If SPI needs to operate at high speeds, use dedicated IO_MUX pins.

注解: For more details about the influence of the MISO input delay on the maximum clock frequency, see *Timing Considerations*.

The IO_MUX pins for SPI buses are given below.

Pin Name	SPI2	SPI3
	GPIO Number	
CS0*	15	5
SCLK	14	18
MISO	12	19
MOSI	13	23
QUADWP	2	22
QUADHD	4	21

- Only the first Device attached to the bus can use the CS0 pin.

Speed and Timing Considerations

Transaction Interval

The ESP32 SPI slave peripherals are designed as general purpose Devices controlled by a CPU. As opposed to dedicated slaves, CPU-based SPI Devices have a limited number of pre-defined registers. All transactions must be handled by the CPU, which means that the transfers and responses are not real-time, and there might be noticeable latency.

As a solution, a Device' s response rate can be doubled by using the functions *spi_slave_queue_trans()* and then *spi_slave_get_trans_result()* instead of using *spi_slave_transmit()*.

You can also configure a GPIO pin through which the Device will signal to the Host when it is ready for a new transaction. A code example of this can be found in peripherals/spi_slave.

SCLK Frequency Requirements

The SPI slaves are designed to operate at up to 10 MHz. The data cannot be recognized or received correctly if the clock is too fast or does not have a 50% duty cycle.

On top of that, there are additional requirements for the data to meet the timing constraints:

- **Read (MOSI):** The Device can read data correctly only if the data is already set at the launch edge. Although it is usually the case for most masters.
- **Write (MISO):** The output delay of the MISO signal needs to be shorter than half of a clock cycle period so that the MISO line is stable before the next latch edge. Given that the clock is balanced, the output delay and frequency limitations in different cases are given below.

	Output delay of MISO (ns)	Freq. limit (MHz)
IO_MUX	43.75	<11.4
GPIO matrix	68.75	<7.2

Note:

1. If the frequency is equal to the limitation, it can lead to random errors.
2. The clock uncertainty between Host and Device (12.5ns) is included.
3. The output delay is measured under ideal circumstances (no load). If the MISO pin is heavily loaded, the output delay will be longer, and the maximum allowed frequency will be lower.

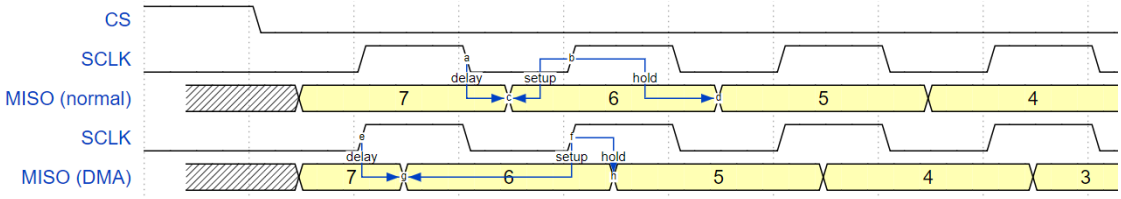
Exception: The frequency is allowed to be higher if the master has more tolerance for the MISO setup time, e.g., latch data at the next edge than expected, or configurable latching time.

Restrictions and Known Issues

1. If DMA is enabled, the rx buffer should be word-aligned (starting from a 32-bit boundary and having a length of multiples of 4 bytes). Otherwise, DMA may write incorrectly or not in a boundary aligned manner. The driver reports an error if this condition is not satisfied.

   Also, a Host should write lengths that are multiples of 4 bytes. The data with inappropriate lengths will be discarded.

2. Furthermore, DMA requires SPI modes 1 and 3. For SPI modes 0 and 2, the MISO signal has to be launched half a clock cycle earlier to meet the timing. The new timing is as follows:



If DMA is enabled, a Device' s launch edge is half of an SPI clock cycle ahead of the normal time, shifting to the Master' s actual latch edge. In this case, if the GPIO matrix is bypassed, the hold time for data sampling is 68.75 ns and no longer a half of an SPI clock cycle. If the GPIO matrix is used, the hold time will increase to 93.75 ns. The Host should sample the data immediately at the latch edge or communicate in SPI modes 1 or 3. If your Host cannot meet these timing requirements, initialize your Device without DMA.

Application Example

The code example for Device/Host communication can be found in the peripherals/spi_slave directory of ESP-IDF examples.

API Reference

Header File

- driver/include/driver/spi_slave.h

Functions

esp_err_t **spi_slave_initialize**(*spi_host_device_t host*, **const** *spi_bus_config_t* **bus_config*, **const** *spi_slave_interface_config_t* **slave_config*, int *dma_chan*)

Initialize a SPI bus as a slave interface.

Warning For now, only supports HSPI and VSPI.

Warning If a DMA channel is selected, any transmit and receive buffer used should be allocated in DMA-capable memory.

Warning The ISR of SPI is always executed on the core which calls this function. Never starve the ISR on this core or the SPI transactions will not be handled.

Return

- ESP_ERR_INVALID_ARG if configuration is invalid
- ESP_ERR_INVALID_STATE if host already is in use
- ESP_ERR_NO_MEM if out of memory
- ESP_OK on success

Parameters

- `host`: SPI peripheral to use as a SPI slave interface
- `bus_config`: Pointer to a *spi_bus_config_t* struct specifying how the host should be initialized
- `slave_config`: Pointer to a *spi_slave_interface_config_t* struct specifying the details for the slave interface
- `dma_chan`: Either 1 or 2. A SPI bus used by this driver must have a DMA channel associated with it. The SPI hardware has two DMA channels to share. This parameter indicates which one to use.

esp_err_t **spi_slave_free**(*spi_host_device_t host*)

Free a SPI bus claimed as a SPI slave interface.

Return

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_ERR_INVALID_STATE if not all devices on the bus are freed

- ESP_OK on success

Parameters

- `host`: SPI peripheral to free

esp_err_t **spi_slave_queue_trans**(*spi_host_device_t host*, **const** *spi_slave_transaction_t* **trans_desc*, TickType_t *ticks_to_wait*)

Queue a SPI transaction for execution.

Queues a SPI transaction to be executed by this slave device. (The transaction queue size was specified when the slave device was initialised via spi_slave_initialize.) This function may block if the queue is full (depending on the ticks_to_wait parameter). No SPI operation is directly initiated by this function, the next queued transaction will happen when the master initiates a SPI transaction by pulling down CS and sending out clock signals.

This function hands over ownership of the buffers in `trans_desc` to the SPI slave driver; the application is not to access this memory until `spi_slave_queue_trans` is called to hand ownership back to the application.

Return

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_OK on success

Parameters

- `host`: SPI peripheral that is acting as a slave
- `trans_desc`: Description of transaction to execute. Not const because we may want to write status back into the transaction description.
- `ticks_to_wait`: Ticks to wait until there' s room in the queue; use portMAX_DELAY to never time out.

esp_err_t **spi_slave_get_trans_result**(*spi_host_device_t host*, *spi_slave_transaction_t* ***trans_desc*, TickType_t *ticks_to_wait*)

Get the result of a SPI transaction queued earlier.

This routine will wait until a transaction to the given device (queued earlier with spi_slave_queue_trans) has succesfully completed. It will then return the description of the completed transaction so software can inspect the result and e.g. free the memory or re-use the buffers.

It is mandatory to eventually use this function for any transaction queued by `spi_slave_queue_trans`.

Return

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_OK on success

Parameters

- `host`: SPI peripheral to that is acting as a slave
- `trans_desc`: Pointer to variable able to contain a pointer to the description of the transaction that is executed
- `ticks_to_wait`: Ticks to wait until there' s a returned item; use portMAX_DELAY to never time out.

esp_err_t **spi_slave_transmit**(*spi_host_device_t host*, *spi_slave_transaction_t *trans_desc*, TickType_t *ticks_to_wait*)

Do a SPI transaction.

Essentially does the same as spi_slave_queue_trans followed by spi_slave_get_trans_result. Do not use this when there is still a transaction queued that hasn' t been finalized using spi_slave_get_trans_result.

Return

- ESP_ERR_INVALID_ARG if parameter is invalid
- ESP_OK on success

Parameters

- `host`: SPI peripheral to that is acting as a slave
- `trans_desc`: Pointer to variable able to contain a pointer to the description of the transaction that is executed. Not const because we may want to write status back into the transaction description.
- `ticks_to_wait`: Ticks to wait until there' s a returned item; use portMAX_DELAY to never time out.

Structures

struct spi_slave_interface_config_t

This is a configuration for a SPI host acting as a slave device.

Public Members

int **spics_io_num**

CS GPIO pin for this device.

uint32_t **flags**

Bitwise OR of SPI_SLAVE_* flags.

int **queue_size**

Transaction queue size. This sets how many transactions can be 'in the air' (queued using spi_slave_queue_trans but not yet finished using spi_slave_get_trans_result) at the same time.

uint8_t **mode**

SPI mode (0-3)

slave_transaction_cb_t **post_setup_cb**

Callback called after the SPI registers are loaded with new data.

This callback is called within interrupt context should be in IRAM for best performance, see "Transferring Speed" section in the SPI Master documentation for full details. If not, the callback may crash during flash operation when the driver is initialized with ESP_INTR_FLAG_IRAM.

slave_transaction_cb_t **post_trans_cb**

Callback called after a transaction is done.

This callback is called within interrupt context should be in IRAM for best performance, see "Transferring Speed" section in the SPI Master documentation for full details. If not, the callback may crash during flash operation when the driver is initialized with ESP_INTR_FLAG_IRAM.

struct spi_slave_transaction_t

This structure describes one SPI transaction

Public Members

size_t **length**

Total data length, in bits.

size_t **trans_len**

Transaction data length, in bits.

const void ***tx_buffer**

Pointer to transmit buffer, or NULL for no MOSI phase.

void ***rx_buffer**

Pointer to receive buffer, or NULL for no MISO phase. When the DMA is anabled, must start at WORD boundary (`rx_buffer%4==0`), and has length of a multiple of 4 bytes.

void ***user**

User-defined variable. Can be used to store eg transaction ID.

Macros

SPI_SLAVE_TXBIT_LSBFIRST

Transmit command/address/data LSB first instead of the default MSB first.

`SPI_SLAVE_RXBIT_LSBFIRST`
: Receive data LSB first instead of the default MSB first.

`SPI_SLAVE_BIT_LSBFIRST`
: Transmit and receive LSB first.

Type Definitions

`typedef struct` *spi_slave_transaction_t* `spi_slave_transaction_t`

`typedef` void (*`slave_transaction_cb_t`)(*spi_slave_transaction_t* *trans)

2.3.17 ESP32-S2 Temperature Sensor

Overview

The ESP32-S2 has a built-in temperature sensor. The temperature sensor module contains an 8-bit Sigma-Delta ADC and a temperature offset DAC. The conversion relationship is the first columns of the table below. Among them, offset = 0 is the main measurement option, and other values are extended measurement options.

offset	measure range(Celsius)	measure error(Celsius)
-2	50 ~ 125	< 3
-1	20 ~ 100	< 2
0	-10 ~ 80	< 1
1	-30 ~ 50	< 2
2	-40 ~ 20	< 3

Application Example

Temperature sensor reading example: peripherals/temp_sensor.

API Reference - Normal Temp Sensor

Header File

- driver/esp32s2beta/include/temp_sensor.h

Functions

esp_err_t `temp_sensor_set_config`(*temp_sensor_config_t tsens*)
: Set parameter of temperature sensor.

Return

- ESP_OK Success

Parameters

- `tsens`:

esp_err_t **temp_sensor_get_config**(*temp_sensor_config_t* **tsens*)

Get parameter of temperature sensor.

Return

- ESP_OK Success

Parameters

- `tsens`:

esp_err_t **temp_sensor_start**(void)

Start temperature sensor measure.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG

esp_err_t **temp_sensor_stop**(void)

Stop temperature sensor measure.

Return

- ESP_OK Success

esp_err_t **temp_sensor_read_raw**(uint32_t **tsens_out*)

Read temperature sensor raw data.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG `tsens_out` is NULL
- ESP_ERR_INVALID_STATE temperature sensor dont start

Parameters

- `tsens_out`: Pointer to raw data, Range: 0 ~ 255

esp_err_t **temp_sensor_read_celsius**(float **celsius*)

Read temperature sensor data that is converted to degrees Celsius.

Note Should not be called from interrupt.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG ARG is NULL.
- ESP_ERR_INVALID_STATE The ambient temperature is out of range.

Parameters

- `celsius`: The measure output value.

Structures

`struct temp_sensor_config_t`

Configuration for temperature sensor reading.

Public Members

temp_sensor_dac_offset_t **dac_offset**

The temperature measurement range is configured with a built-in temperature offset DAC.

uint8_t **clk_div**

Default: 6

Macros

`TSENS_CONFIG_DEFAULT()`

Enumerations

`enum temp_sensor_dac_offset_t`

Values:

`TSENS_DAC_L0` = 0

offset = -2, measure range: 50℃ ~ 125℃, error < 3℃.

`TSENS_DAC_L1`

offset = -1, measure range: 20℃ ~ 100℃, error < 2℃.

`TSENS_DAC_L2`

offset = 0, measure range:-10℃ ~ 80℃, error < 1℃.

`TSENS_DAC_L3`

offset = 1, measure range:-30℃ ~ 50℃, error < 2℃.

`TSENS_DAC_L4`
: offset = 2, measure range:-40℃ ~ 20℃, error < 3℃.

`TSENS_DAC_MAX`

`TSENS_DAC_DEFAULT` = TSENS_DAC_L2

2.3.18 定时器

[English]

简介

ESP32 芯片提供两组硬件定时器，每组包含两个通用硬件定时器。所有定时器均为 64 位通用定时器，包括 16 位预分频器和 64 位自动重载向上/向下计数器。

功能概述

下文介绍了配置和操作定时器的常规步骤：

- 定时器初始化 - 启动定时器前应设置的参数，以及每个设置提供的具体功能。
- 定时器控制 - 如何读取定时器的值，如何暂停/启动定时器以及如何改变定时器的操作方式。
- 警报 - 如何设置和使用警报。
- 中断- 如何使能和使用中断。

定时器初始化

两个 ESP32 定时器组中，每组都有两个定时器，两组共有四个定时器供使用。ESP32 定时器组应使用 *timer_group_t* 识别，而每组中的个体定时器则应使用 *timer_idx_t* 识别。

首先调用 *timer_init()* 函数，并将 *timer_config_t* 结构体传递给此函数，用于定义定时器的工作方式，实现定时器初始化。特别注意以下定时器参数可设置为：

- **分频器**：设置定时器中计数器计数的速度，`divider` 的设置将用作输入的 80 MHz APB_CLK 时钟的除数。
- **模式**：设置计数器是递增还是递减。可通过从 *timer_count_dir_t* 中选取一个值，后使用 `counter_dir` 来选择模式。
- **计数器使能**：如果计数器已使能，则在调用 *timer_init()* 后计数器将立即开始递增/递减。您可通过从 *timer_start_t* 中选取一个值，后使用 `counter_en` 改变此行为。
- **报警使能**：可使用 `alarm_en` 设置。

- **自动重载**: 设置计数器是否应该在定时器警报上使用 auto_reload 自动重载首个计数值，还是继续递增或递减。
- **中断类型**: 选择定时器警报上应触发的中断类型，请设置*timer_intr_mode_t* 中定义的值。

要获取定时器设置的当前值，请使用函数*timer_get_config()*。

定时器控制

定时器使能后便开始计数。要使能定时器，可首先设置 counter_en 为 true，然后调用函数*timer_init()*，或者直接调用函数*timer_start()*。您可通过调用函数*timer_set_counter_value()* 来指定定时器的首个计数值。要检查定时器的当前值，调用函数*timer_get_counter_value()* 或*timer_get_counter_time_sec()*。

可通过调用函数*timer_pause()* 随时暂停定时器。要再次启动它，调用函数*timer_start()*。

要重新配置定时器，可调用函数*timer_init()*，该函数详细介绍见定时器初始化。

除此之外，还可通过使用专有函数更改个别设置来重新配置定时器：

设置	专有函数	描述
分频器	*timer_set_divider()*	更改计数频率。为避免发生不可预测情况，更改分频器时应暂停定时器。如果定时器正在运行，则使用*timer_set_divider()* 将其暂停并更改设置，然后重启定时器。
模式	*timer_set_counter_mode()*	设置计数器应递增还是递减
自动重载	*timer_set_auto_reload()*	设置是否应在定时器警报上重载首个计数值

警报

要设置警报，先调用函数*timer_set_alarm_value()*，然后使用*timer_set_alarm()* 使能警报。当调用函数*timer_init()* 时，也可以在定时器初始化阶段使能警报。

警报已使能且定时器达到警报值后，根据配置，可能会出现以下两种行为：

- 如果先前已配置，此时将触发中断。有关如何配置中断，请参见中断。
- 如 auto_reload 已使能，定时器的计数器将重新加载，从先前配置好的值开始再次计数。应使用函数*timer_set_counter_value()* 预先设置该值。

注解:

- 如果已设置警报值且定时器已超过该值，则将立即触发警报。

- 一旦触发后，警报将自动关闭，需要重新使能以再次触发。

要检查某特定的警报值，调用函数*timer_get_alarm_value()*。

中断

可通过调用函数*timer_isr_register()* 为特定定时器组和定时器注册中断处理程序。

调用*timer_group_intr_enable()* 使能定时器组的中断程序，调用*timer_enable_intr()* 使能某定时器的中断程序。调用*timer_group_intr_disable()* 关闭定时器组的中断程序，调用*timer_disable_intr()* 关闭某定时器的中断程序。

在中断服务程序（ISR）中处理中断时，需要明确地清除中断状态位。为此，请设置定义在 soc/esp32/include/soc/timer_group_struct.h 中的 `TIMERGN.int_clr_timers.tM` 结构。该结构中 `N` 是定时器组别编号 [0, 1]，`M` 是定时器编号 [0, 1]。例如，要清除定时器组别 0 中定时器 1 的中断状态位，请调用以下命令:

```
TIMERG0.int_clr_timers.t1 = 1
```

有关如何使用中断，请参阅应用示例。

应用示例

64 位硬件定时器示例：peripherals/timer_group。

API 参考

Header File

- driver/include/driver/timer.h

Functions

esp_err_t **timer_get_counter_value**(*timer_group_t group_num*, *timer_idx_t timer_num*, uint64_t **timer_val*)

Read the counter value of hardware timer.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `group_num`: Timer group, 0 for TIMERG0 or 1 for TIMERG1
- `timer_num`: Timer index, 0 for hw_timer[0] & 1 for hw_timer[1]
- `timer_val`: Pointer to accept timer counter value.

esp_err_t **timer_get_counter_time_sec**(*timer_group_t group_num*, *timer_idx_t timer_num*, double **time*)

Read the counter value of hardware timer, in unit of a given scale.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `group_num`: Timer group, 0 for TIMERG0 or 1 for TIMERG1
- `timer_num`: Timer index, 0 for hw_timer[0] & 1 for hw_timer[1]
- `time`: Pointer, type of double*, to accept timer counter value, in seconds.

esp_err_t **timer_set_counter_value**(*timer_group_t group_num*, *timer_idx_t timer_num*, uint64_t *load_val*)

Set counter value to hardware timer.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `group_num`: Timer group, 0 for TIMERG0 or 1 for TIMERG1
- `timer_num`: Timer index, 0 for hw_timer[0] & 1 for hw_timer[1]
- `load_val`: Counter value to write to the hardware timer.

esp_err_t **timer_start**(*timer_group_t group_num*, *timer_idx_t timer_num*)

Start the counter of hardware timer.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `group_num`: Timer group number, 0 for TIMERG0 or 1 for TIMERG1

- `timer_num`: Timer index, 0 for hw_timer[0] & 1 for hw_timer[1]

esp_err_t **timer_pause**(*timer_group_t group_num*, *timer_idx_t timer_num*)

Pause the counter of hardware timer.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `group_num`: Timer group number, 0 for TIMERG0 or 1 for TIMERG1
- `timer_num`: Timer index, 0 for hw_timer[0] & 1 for hw_timer[1]

esp_err_t **timer_set_counter_mode**(*timer_group_t group_num*, *timer_idx_t timer_num*, *timer_count_dir_t counter_dir*)

Set counting mode for hardware timer.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `group_num`: Timer group number, 0 for TIMERG0 or 1 for TIMERG1
- `timer_num`: Timer index, 0 for hw_timer[0] & 1 for hw_timer[1]
- `counter_dir`: Counting direction of timer, count-up or count-down

esp_err_t **timer_set_auto_reload**(*timer_group_t group_num*, *timer_idx_t timer_num*, *timer_autoreload_t reload*)

Enable or disable counter reload function when alarm event occurs.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `group_num`: Timer group number, 0 for TIMERG0 or 1 for TIMERG1
- `timer_num`: Timer index, 0 for hw_timer[0] & 1 for hw_timer[1]
- `reload`: Counter reload mode.

esp_err_t **timer_set_divider**(*timer_group_t group_num*, *timer_idx_t timer_num*, uint32_t *divider*)

Set hardware timer source clock divider. Timer groups clock are divider from APB clock.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `group_num`: Timer group number, 0 for TIMERG0 or 1 for TIMERG1
- `timer_num`: Timer index, 0 for hw_timer[0] & 1 for hw_timer[1]
- `divider`: Timer clock divider value. The divider's range is from from 2 to 65536.

esp_err_t **timer_set_alarm_value**(*timer_group_t group_num*, *timer_idx_t timer_num*, uint64_t *alarm_value*)

Set timer alarm value.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `group_num`: Timer group, 0 for TIMERG0 or 1 for TIMERG1
- `timer_num`: Timer index, 0 for hw_timer[0] & 1 for hw_timer[1]
- `alarm_value`: A 64-bit value to set the alarm value.

esp_err_t **timer_get_alarm_value**(*timer_group_t group_num*, *timer_idx_t timer_num*, uint64_t **alarm_value*)

Get timer alarm value.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `group_num`: Timer group, 0 for TIMERG0 or 1 for TIMERG1
- `timer_num`: Timer index, 0 for hw_timer[0] & 1 for hw_timer[1]
- `alarm_value`: Pointer of A 64-bit value to accept the alarm value.

esp_err_t **timer_set_alarm**(*timer_group_t group_num*, *timer_idx_t timer_num*, *timer_alarm_t alarm_en*)

Enable or disable generation of timer alarm events.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `group_num`: Timer group, 0 for TIMERG0 or 1 for TIMERG1
- `timer_num`: Timer index, 0 for hw_timer[0] & 1 for hw_timer[1]
- `alarm_en`: To enable or disable timer alarm function.

esp_err_t **timer_isr_callback_add**(*timer_group_t group_num*, *timer_idx_t timer_num*, *timer_isr_t isr_handler*, void **arg*, int *intr_alloc_flags*)

Add ISR handle callback for the corresponding timer.

The callback should return a bool value to determine whether need to do YIELD at the end of the ISR.

Note This ISR handler will be called from an ISR. This ISR handler do not need to handle interrupt status, and should be kept short. If you want to realize some specific applications or write the whole ISR, you can call timer_isr_register(···) to register ISR.

Parameters

- `group_num`: Timer group number
- `timer_num`: Timer index of timer group
- `isr_handler`: Interrupt handler function, it is a callback function.
- `arg`: Parameter for handler function
- `intr_alloc_flags`: Flags used to allocate the interrupt. One or multiple (ORred) ESP_INTR_FLAG_* values. See esp_intr_alloc.h for more info.

If the intr_alloc_flags value ESP_INTR_FLAG_IRAM is set, the handler function must be declared with IRAM_ATTR attribute and can only call functions in IRAM or ROM. It cannot call other timer APIs.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **timer_isr_callback_remove**(*timer_group_t group_num*, *timer_idx_t timer_num*)

Remove ISR handle callback for the corresponding timer.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `group_num`: Timer group number
- `timer_num`: Timer index of timer group

esp_err_t **timer_isr_register**(*timer_group_t group_num*, *timer_idx_t timer_num*, void (**fn*)(void *), void **arg*, int *intr_alloc_flags*, *timer_isr_handle_t* **handle*)

Register Timer interrupt handler, the handler is an ISR. The handler will be attached to the same CPU core that this function is running on.

If the intr_alloc_flags value ESP_INTR_FLAG_IRAM is set, the handler function must be declared with IRAM_ATTR attribute and can only call functions in IRAM or ROM. It cannot call other timer APIs. Use direct register access to configure timers from inside the ISR in this case.

Note If use this function to reigster ISR, you need to write the whole ISR. In the interrupt handler, you need to call timer_spinlock_take(..) before your handling, and call timer_spinlock_give(···) after your handling.

Parameters

- `group_num`: Timer group number
- `timer_num`: Timer index of timer group
- `fn`: Interrupt handler function.
- `arg`: Parameter for handler function
- `intr_alloc_flags`: Flags used to allocate the interrupt. One or multiple (ORred) ESP_INTR_FLAG_* values. See esp_intr_alloc.h for more info.
- `handle`: Pointer to return handle. If non-NULL, a handle for the interrupt will be returned here.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **timer_init**(*timer_group_t group_num*, *timer_idx_t timer_num*, **const** *timer_config_t* **config*)

Initializes and configure the timer.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `group_num`: Timer group number, 0 for TIMERG0 or 1 for TIMERG1
- `timer_num`: Timer index, 0 for hw_timer[0] & 1 for hw_timer[1]
- `config`: Pointer to timer initialization parameters.

esp_err_t **timer_deinit**(*timer_group_t group_num*, *timer_idx_t timer_num*)

Deinitializes the timer.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `group_num`: Timer group number, 0 for TIMERG0 or 1 for TIMERG1
- `timer_num`: Timer index, 0 for hw_timer[0] & 1 for hw_timer[1]

esp_err_t **timer_get_config**(*timer_group_t group_num*, *timer_idx_t timer_num*, *timer_config_t *config*)

Get timer configure value.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `group_num`: Timer group number, 0 for TIMERG0 or 1 for TIMERG1
- `timer_num`: Timer index, 0 for hw_timer[0] & 1 for hw_timer[1]
- `config`: Pointer of struct to accept timer parameters.

esp_err_t **timer_group_intr_enable**(*timer_group_t group_num*, *timer_intr_t intr_mask*)

Enable timer group interrupt, by enable mask.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `group_num`: Timer group number, 0 for TIMERG0 or 1 for TIMERG1
- `intr_mask`: Timer interrupt enable mask.
  - TIMER_INTR_T0: t0 interrupt
  - TIMER_INTR_T1: t1 interrupt
  - TIMER_INTR_WDT: watchdog interrupt

esp_err_t **timer_group_intr_disable**(*timer_group_t group_num*, *timer_intr_t intr_mask*)

Disable timer group interrupt, by disable mask.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `group_num`: Timer group number, 0 for TIMERG0 or 1 for TIMERG1
- `intr_mask`: Timer interrupt disable mask.
  - TIMER_INTR_T0: t0 interrupt
  - TIMER_INTR_T1: t1 interrupt
  - TIMER_INTR_WDT: watchdog interrupt

esp_err_t **timer_enable_intr**(*timer_group_t group_num*, *timer_idx_t timer_num*)

Enable timer interrupt.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `group_num`: Timer group number, 0 for TIMERG0 or 1 for TIMERG1
- `timer_num`: Timer index.

esp_err_t **timer_disable_intr**(*timer_group_t group_num*, *timer_idx_t timer_num*)

Disable timer interrupt.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `group_num`: Timer group number, 0 for TIMERG0 or 1 for TIMERG1
- `timer_num`: Timer index.

void **timer_group_intr_clr_in_isr**(*timer_group_t group_num*, *timer_idx_t timer_num*)

Clear timer interrupt status, just used in ISR.

Return

- None

Parameters

- `group_num`: Timer group number, 0 for TIMERG0 or 1 for TIMERG1
- `timer_num`: Timer index.

void **timer_group_clr_intr_status_in_isr**(*timer_group_t group_num*, *timer_idx_t timer_num*)

Clear timer interrupt status, just used in ISR.

Return

- None

Parameters

- `group_num`: Timer group number, 0 for TIMERG0 or 1 for TIMERG1
- `timer_num`: Timer index.

void **timer_group_enable_alarm_in_isr**(*timer_group_t group_num*, *timer_idx_t timer_num*)

Enable alarm interrupt, just used in ISR.

Return

- None

Parameters

- `group_num`: Timer group number, 0 for TIMERG0 or 1 for TIMERG1
- `timer_num`: Timer index.

uint64_t **timer_group_get_counter_value_in_isr**(*timer_group_t group_num*, *timer_idx_t timer_num*)

Get the current counter value, just used in ISR.

Return

- Counter value

Parameters

- `group_num`: Timer group number, 0 for TIMERG0 or 1 for TIMERG1
- `timer_num`: Timer index.

void **timer_group_set_alarm_value_in_isr**(*timer_group_t group_num*, *timer_idx_t timer_num*, uint64_t *alarm_val*)

Set the alarm threshold for the timer, just used in ISR.

Return

- None

Parameters

- `group_num`: Timer group number, 0 for TIMERG0 or 1 for TIMERG1
- `timer_num`: Timer index.
- `alarm_val`: Alarm threshold.

void **timer_group_set_counter_enable_in_isr**(*timer_group_t group_num*, *timer_idx_t timer_num*, *timer_start_t counter_en*)

Enable/disable a counter, just used in ISR.

Return

- None

Parameters

- `group_num`: Timer group number, 0 for TIMERG0 or 1 for TIMERG1
- `timer_num`: Timer index.
- `counter_en`: Enable/disable.

timer_intr_t **timer_group_intr_get_in_isr**(*timer_group_t group_num*)

Get the masked interrupt status, just used in ISR.

Return

- Interrupt status

Parameters

- `group_num`: Timer group number, 0 for TIMERG0 or 1 for TIMERG1

uint32_t **timer_group_get_intr_status_in_isr**(*timer_group_t group_num*)

Get interrupt status, just used in ISR.

Return

- Interrupt status

Parameters

- `group_num`: Timer group number, 0 for TIMERG0 or 1 for TIMERG1

void **timer_group_clr_intr_sta_in_isr**(*timer_group_t group_num*, *timer_intr_t intr_mask*)

Clear the masked interrupt status, just used in ISR.

Return

- None

Parameters

- `group_num`: Timer group number, 0 for TIMERG0 or 1 for TIMERG1
- `intr_mask`: Masked interrupt.

bool **timer_group_get_auto_reload_in_isr**(*timer_group_t group_num*, *timer_idx_t timer_num*)

Get auto reload enable status, just used in ISR.

Return

- True Auto reload enabled
- False Auto reload disabled

Parameters

- `group_num`: Timer group number, 0 for TIMERG0 or 1 for TIMERG1
- `timer_num`: Timer index

esp_err_t **timer_spinlock_take**(*timer_group_t group_num*)

Take timer spinlock to enter critical protect.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `group_num`: Timer group number, 0 for TIMERG0 or 1 for TIMERG1

esp_err_t **timer_spinlock_give**(*timer_group_t group_num*)

Give timer spinlock to exit critical protect.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `group_num`: Timer group number, 0 for TIMERG0 or 1 for TIMERG1

Macros

`TIMER_BASE_CLK`

Frequency of the clock on the input of the timer groups

Type Definitions

`typedef` bool (*`timer_isr_t`)(void *)

Interrupt handle callback function. User need to retrun a bool value in callback.

Return

- True Do task yield at the end of ISR
- False Not do task yield at the end of ISR

Note If you called FreeRTOS functions in callback, you need to return true or false based on the retrun value of argument `pxHigherPriorityTaskWoken`. For example, `xQueueSendFromISR` is called in callback, if the return value `pxHigherPriorityTaskWoken` of any FreeRTOS calls is pdTRUE, return true; otherwise return false.

`typedef` *intr_handle_t* `timer_isr_handle_t`

Interrupt handle, used in order to free the isr after use. Aliases to an int handle for now.

Header File

- soc/include/hal/timer_types.h

Structures

`struct timer_config_t`

Data structure with timer’ s configuration settings.

Public Members

timer_alarm_t `alarm_en`

Timer alarm enable

timer_start_t `counter_en`

Counter enable

timer_intr_mode_t **intr_type**

Interrupt mode

timer_count_dir_t **counter_dir**

Counter direction

timer_autoreload_t **auto_reload**

Timer auto-reload

uint32_t **divider**

Counter clock divider. The divider’ s range is from from 2 to 65536.

Enumerations

enum timer_group_t

Selects a Timer-Group out of 2 available groups.

Values:

TIMER_GROUP_0 = 0

Hw timer group 0

TIMER_GROUP_1 = 1

Hw timer group 1

TIMER_GROUP_MAX

enum timer_idx_t

Select a hardware timer from timer groups.

Values:

TIMER_0 = 0

Select timer0 of GROUPx

TIMER_1 = 1

Select timer1 of GROUPx

TIMER_MAX

enum timer_count_dir_t

Decides the direction of counter.

Values:

TIMER_COUNT_DOWN = 0

Descending Count from cnt.high|cnt.low

TIMER_COUNT_UP = 1

Ascending Count from Zero

TIMER_COUNT_MAX

enum timer_start_t

Decides whether timer is on or paused.

Values:

TIMER_PAUSE = 0

Pause timer counter

TIMER_START = 1

Start timer counter

enum timer_intr_t

Interrupt types of the timer.

Values:

TIMER_INTR_T0 = BIT(0)

interrupt of timer 0

TIMER_INTR_T1 = BIT(1)

interrupt of timer 1

TIMER_INTR_WDT = BIT(2)

interrupt of watchdog

TIMER_INTR_NONE = 0

enum timer_wdt_behavior_t

Behavior of the watchdog if a stage times out.

Values:

TIMER_WDT_OFF = 0

The stage is turned off

TIMER_WDT_INT = 1

The stage will trigger an interrupt

TIMER_WDT_RESET_CPU = 2

The stage will reset the CPU

TIMER_WDT_RESET_SYSTEM = 3

The stage will reset the whole system

enum timer_alarm_t

Decides whether to enable alarm mode.

Values:

TIMER_ALARM_DIS = 0

Disable timer alarm

TIMER_ALARM_EN = 1
Enable timer alarm

TIMER_ALARM_MAX

enum timer_intr_mode_t
Select interrupt type if running in alarm mode.

Values:

TIMER_INTR_LEVEL = 0
Interrupt mode: level mode

TIMER_INTR_MAX

enum timer_autoreload_t
Select if Alarm needs to be loaded by software or automatically reload by hardware.

Values:

TIMER_AUTORELOAD_DIS = 0
Disable auto-reload: hardware will not load counter value after an alarm event

TIMER_AUTORELOAD_EN = 1
Enable auto-reload: hardware will load counter value after an alarm event

TIMER_AUTORELOAD_MAX

2.3.19 触摸传感器

[English]

概述

触摸传感器系统由保护覆盖层、触摸电极、绝缘基板和走线组成，保护覆盖层位于最上层，绝缘基板上设有电极及走线。用户触摸覆盖层将产生电容变化，根据电容变化判断此次触摸是否为有效触摸行为。

ESP32 可支持最多 10 个电容式触摸板/GPIO，触摸板可以以矩阵或滑条等方式组合使用，从而覆盖更大触感区域及更多触感点。触摸传感由有限状态机 (FSM) 硬件控制，由软件或专用硬件计时器发起。

如需了解触摸传感器设计、操作及其控制寄存器等相关信息，请参考《ESP32 技术参考手册》(PDF)，您也可以在《ESP32 技术参考手册》中查看这一子系统是如何运行的。

请参考 触摸传感器应用方案简介，查看触摸传感器设计详情和固件开发指南。如果不想亲自在多种配置环境下测试触摸传感器，请查看 ESP32 触摸功能开发套件。

功能介绍

下面将 API 分解成几个函数组进行介绍，帮助您快速了解以下功能：

- 初始化触摸板驱动程序
- 配置触摸板 GPIO 管脚
- 触摸状态测量
- 调整测量参数（优化测量）
- 过滤触摸测量
- 触摸监测方式
- 设置中断信号监测触碰动作
- 中断触发

请前往*API 参考* 章节，查看某一函数的具体描述。*应用示例* 章节则介绍了此 API 的具体实现。

初始化触摸板驱动程序

使用触摸板之前，需要先调用 `touch_pad_init()` 函数初始化触摸板驱动程序。此函数设置了*API 参考* 项下的 *Macros* 中列出的几项 `..._DEFAULT` 驱动程序参数，同时删除之前设置过的触摸板信息（如有），并禁用中断。

如果不再需要该驱动程序，可以调用 `touch_pad_deinit()` 释放已初始化的驱动程序。

配置触摸板 GPIO 管脚

调用 `touch_pad_config()` 使能某一 GPIO 的触感功能。

使用 `touch_pad_set_fsm_mode()` 选择触摸板测量（由 FSM 操作）是由硬件计时器自动启动，还是由软件自动启动。如果选择软件模式，请使用 `touch_pad_sw_start()` 启动 FSM。

触摸状态测量

借助以下两个函数从传感器读取原始数据或过滤后的数据：

- `touch_pad_read()`
- `touch_pad_read_filtered()`

这两个函数也可以用于检查触碰和释放触摸板时传感器读数变化范围，来评估触摸板设计，然后根据这些信息设定触摸阈值。

注解： 使用 `touch_pad_read_filtered()` 之前，需要先调用过滤触摸测量 中特定的滤波器函数初始化并配置该滤波器。

请参考应用示例 peripherals/touch_pad_read，查看如何使用这两个读值函数。

优化测量

触摸传感器设有数个可配置参数，以适应触摸板设计特点。例如，如果需要感知较细微的电容变化，则可以缩小触摸板充放电的参考电压范围。您可以使用 `touch_pad_set_voltage()` 函数设置电压参考低值和参考高值。

优化测量除了可以识别细微的电容变化之外，还可以降低应用程序功耗，但可能会增加测量噪声干扰。如果得到的动态读数范围结果比较理想，则可以调用 `touch_pad_set_meas_time()` 函数来减少测量时间，从而进一步降低功耗。

可用的测量参数及相应的‘set’函数总结如下：

- 触摸板充放电参数：
  - 电压门限：`touch_pad_set_voltage()`
  - 速率（斜率）`touch_pad_set_cnt_mode()`
- 测量时间：`touch_pad_set_meas_time()`

电压门限（参考低值/参考高值）、速率（斜率）与测量时间的关系如下图所示：

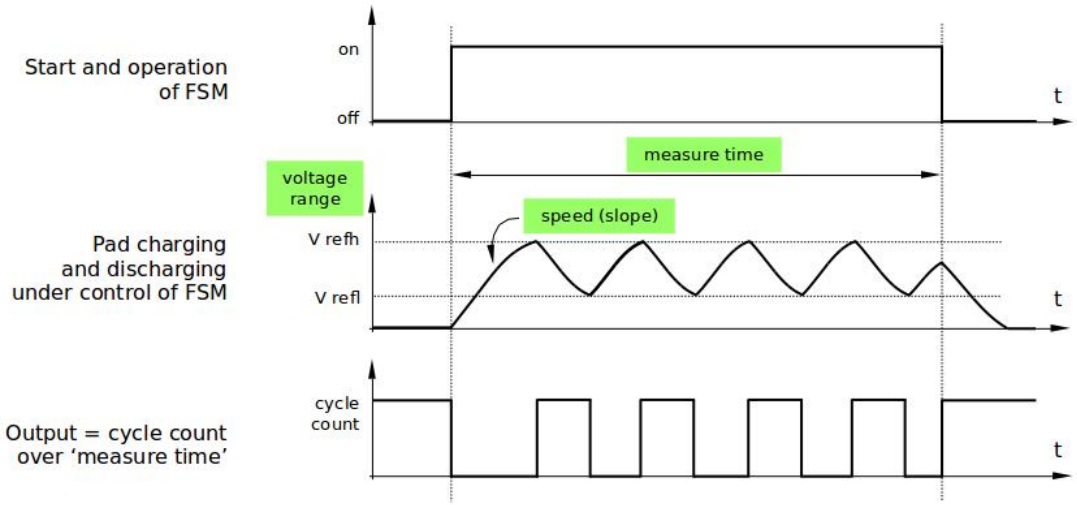


图 22: 触摸板 - 测量参数之间的关系

上图中的 *Output* 代表触摸传感器读值，即一个测量周期内测得的脉冲计数值。

所有函数均成对出现，用于设定某一特定参数，并获取当前参数值。例如：`touch_pad_set_voltage()` 和 `touch_pad_get_voltage()`。

过滤触摸测量

如果测量中存在噪声，可以使用提供的 API 函数对测量进行过滤。使用滤波器之前，请先调用 `touch_pad_filter_start()` 启动该滤波器。

滤波器类型为 IIR（无限脉冲响应滤波器），您可以调用 `touch_pad_set_filter_period()` 配置此类滤波器的采样周期。

如需停止滤波器，请调用 `touch_pad_filter_stop()` 函数。如果不再使用该滤波器，请调用 `touch_pad_filter_delete()` 删除此滤波器。

触摸监测

触摸监测基于用户配置的阈值和 FSM 执行的原始测量，并由 ESP32 硬件实现。你可以调用 `touch_pad_get_status()` 查看被触碰的触摸板，或调用 `touch_pad_clear_status()` 清除触摸状态信息。

您也可以将硬件触摸监测连接至中断，详细介绍见下一章节。

如果测量中存在噪声，且电容变化幅度较小，硬件触摸监测结果可能就不太理想。如需解决这一问题，不建议使用硬件监测或中断信号，建议您在自己的应用程序中采用测量过滤，并执行触摸监测。请参考 peripherals/touch_pad_interrupt，查看以上两种触摸监测的实现方式。

中断触发

在对触摸监测启用中断之前，请先设置一个触摸监测阈值。然后使用触摸状态测量 中所述的函数读取并显示触摸和释放触摸板时测得的结果。如果测量中存在噪声且相对电容变化较小，请使用滤波器。您也可以根据应用程序和环境条件，测试温度和电源电压变化对测量值的影响。

确定监测阈值后就可以在初始化时调用 `touch_pad_config()` 设置此阈值，或在运行时调用 `touch_pad_set_thresh()` 设置此阈值。

下一步就是设置如何触发中断。您可以设置在阈值以下或以上触发中断，具体触发模式由函数 `touch_pad_set_trigger_mode()` 设置。

最后您可以使用以下函数配置和管理中断调用：

- `touch_pad_isr_register()` / `touch_pad_isr_deregister()`
- `touch_pad_intr_enable()` / `touch_pad_intr_disable()`

中断配置完成后，您可以调用 `touch_pad_get_status()` 查看中断信号来自哪个触摸板，也可以调用 `touch_pad_clear_status()` 清除触摸板状态信息。

注解： 触摸监测中的中断信号基于原始/未经过滤的测量值（对比用户设置的阈值），并在硬件中实现。启用软件滤波 API 并不会影响这一过程，见过滤触摸测量。

从睡眠模式唤醒

如果使用触摸板中断将芯片从睡眠模式唤醒，您可以选择配置一些触摸板，例如 SET1 或 SET1 和 SET2，触摸这些触摸板将触发中断并唤醒芯片。请调用 `touch_pad_set_trigger_source()` 实现上述操作。

您可以使用以下函数管理‘SET’中触摸板所需的位模式配置：

- `touch_pad_set_group_mask()` / `touch_pad_get_group_mask()`
- `touch_pad_clear_group_mask()`

应用示例

- 触摸传感器读值示例：peripherals/touch_pad_read
- 触摸传感器中断示例：peripherals/touch_pad_interrupt

API 参考

Header File

- driver/include/driver/touch_pad.h

GPIO 宏查找表

您可以使用宏定义某一触摸板通道的 GPIO，或定义某一 GPIO 的通道。例如：

1. `TOUCH_PAD_NUM5_GPIO_NUM` 定义了通道 5 的 GPIO（即 GPIO 12）；
2. `TOUCH_PAD_GPIO4_CHANNEL` 定义了 GPIO 4 的通道（即通道 0）。

Header File

- soc/esp32/include/soc/touch_sensor_channel.h

Macros

`TOUCH_PAD_GPIO4_CHANNEL`

`TOUCH_PAD_NUM0_GPIO_NUM`

`TOUCH_PAD_GPIO0_CHANNEL`

`TOUCH_PAD_NUM1_GPIO_NUM`

`TOUCH_PAD_GPIO2_CHANNEL`

TOUCH_PAD_NUM2_GPIO_NUM

TOUCH_PAD_GPIO15_CHANNEL

TOUCH_PAD_NUM3_GPIO_NUM

TOUCH_PAD_GPIO13_CHANNEL

TOUCH_PAD_NUM4_GPIO_NUM

TOUCH_PAD_GPIO12_CHANNEL

TOUCH_PAD_NUM5_GPIO_NUM

TOUCH_PAD_GPIO14_CHANNEL

TOUCH_PAD_NUM6_GPIO_NUM

TOUCH_PAD_GPIO27_CHANNEL

TOUCH_PAD_NUM7_GPIO_NUM

TOUCH_PAD_GPIO33_CHANNEL

TOUCH_PAD_NUM8_GPIO_NUM

TOUCH_PAD_GPIO32_CHANNEL

TOUCH_PAD_NUM9_GPIO_NUM

2.3.20 UART

Overview

A Universal Asynchronous Receiver/Transmitter (UART) is a hardware feature that handles communication (i.e., timing requirements and data framing) using widely-adapted asynchronous serial communication interfaces, such as RS232, RS422, RS485. A UART provides a widely adopted and cheap method to realize full-duplex or half-duplex data exchange among different devices.

The ESP32 chip has three UART controllers (UART0, UART1, and UART2) that feature an identical set of registers for ease of programming and flexibility. Each UART controller is independently configurable with parameters such as baud rate, data bit length, bit ordering, number of stop bits, parity bit etc. All the controllers are compatible with UART-enabled devices from various manufacturers and can also support Infrared Data Association protocols (IrDA).

Functional Overview

The following overview describes how to establish communication between an ESP32 and other UART devices using the functions and data types of the UART driver. The overview reflects a typical programming workflow and is broken down into the sections provided below:

1. *Setting Communication Parameters* - Setting baud rate, data bits, stop bits, etc.
2. *Setting Communication Pins* - Assigning pins for connection to a device.
3. *Driver Installation* - Allocating ESP32's resources for the UART driver.
4. *Running UART Communication* - Sending / receiving data
5. *Using Interrupts* - Triggering interrupts on specific communication events
6. *Deleting a Driver* - Freeing allocated resources if a UART communication is no longer required

Steps 1 to 3 comprise the configuration stage. Step 4 is where the UART starts operating. Steps 5 and 6 are optional.

The UART driver's functions identify each of the three UART controllers using *uart_port_t*. This identification is needed for all the following function calls.

Setting Communication Parameters

UART communication parameters can be configured all in a single step or individually in multiple steps.

Single Step

Call the function *uart_param_config()* and pass to it a *uart_config_t* structure. The *uart_config_t* structure should contain all the required parameters. See the example below.

```
const int uart_num = UART_NUM_2;
uart_config_t uart_config = {
    .baud_rate = 115200,
    .data_bits = UART_DATA_8_BITS,
    .parity = UART_PARITY_DISABLE,
    .stop_bits = UART_STOP_BITS_1,
    .flow_ctrl = UART_HW_FLOWCTRL_CTS_RTS,
    .rx_flow_ctrl_thresh = 122,
};
// Configure UART parameters
ESP_ERROR_CHECK(uart_param_config(uart_num, &uart_config));
```

Multiple Steps

Configure specific parameters individually by calling a dedicated function from the table given below. These functions are also useful if re-configuring a single parameter.

表 3: Functions for Configuring specific parameters individually

Parameter to Configure	Function
Baud rate	*uart_set_baudrate()*
Number of transmitted bits	*uart_set_word_length()* selected out of *uart_word_length_t*
Parity control	*uart_set_parity()* selected out of *uart_parity_t*
Number of stop bits	*uart_set_stop_bits()* selected out of *uart_stop_bits_t*
Hardware flow control mode	*uart_set_hw_flow_ctrl()* selected out of *uart_hw_flowcontrol_t*
Communication mode	*uart_set_mode()* selected out of *uart_mode_t*

Each of the above functions has a `_get_` counterpart to check the currently set value. For example, to check the current baud rate value, call *uart_get_baudrate()*.

Setting Communication Pins

After setting communication parameters, configure the physical GPIO pins to which the other UART device will be connected. For this, call the function *uart_set_pin()* and specify the GPIO pin numbers to which the driver should route the Tx, Rx, RTS, and CTS signals. If you want to keep a currently allocated pin number for a specific signal, pass the macro *UART_PIN_NO_CHANGE*.

The same macro should be specified for pins that will not be used.

```
// Set UART pins(TX: IO16 (UART2 default), RX: IO17 (UART2 default), RTS: IO18, CTS:␣
↪IO19)
ESP_ERROR_CHECK(uart_set_pin(UART_NUM_2, UART_PIN_NO_CHANGE, UART_PIN_NO_CHANGE, 18,␣
↪19));
```

Driver Installation

Once the communication pins are set, install the driver by calling *uart_driver_install()* and specify the following parameters:

- Size of Tx ring buffer
- Size of Rx ring buffer
- Event queue handle and size
- Flags to allocate an interrupt

The function will allocate the required ESP32 resources for the UART driver.

```
// Setup UART buffered IO with event queue
const int uart_buffer_size = (1024 * 2);
```

(下页继续)

(续上页)

```
QueueHandle_t uart_queue;
// Install UART driver using an event queue here
ESP_ERROR_CHECK(uart_driver_install(UART_NUM_2, uart_buffer_size, \
                                        uart_buffer_size, 10, &uart_queue, 0));
```

Once this step is complete, you can connect the external UART device and check the communication.

Running UART Communication

Serial communication is controlled by each UART controller' s finite state machine (FSM).

The process of sending data involves the following steps:

1. Write data into Tx FIFO buffer
2. FSM serializes the data
3. FSM sends the data out

The process of receiving data is similar, but the steps are reversed:

1. FSM processes an incoming serial stream and parallelizes it
2. FSM writes the data into Rx FIFO buffer
3. Read the data from Rx FIFO buffer

Therefore, an application will be limited to writing and reading data from a respective buffer using *uart_write_bytes()* and *uart_read_bytes()* respectively, and the FSM will do the rest.

Transmitting

After preparing the data for transmission, call the function *uart_write_bytes()* and pass the data buffer' s address and data length to it. The function will copy the data to the Tx ring buffer (either immediately or after enough space is available), and then exit. When there is free space in the Tx FIFO buffer, an interrupt service routine (ISR) moves the data from the Tx ring buffer to the Tx FIFO buffer in the background. The code below demonstrates the use of this function.

```
// Write data to UART.
char* test_str = "This is a test string.\n";
uart_write_bytes(uart_num, (const char*)test_str, strlen(test_str));
```

The function *uart_write_bytes_with_break()* is similar to *uart_write_bytes()* but adds a serial break signal at the end of the transmission. A 'serial break signal' means holding the Tx line low for a period longer than one data frame.

```
// Write data to UART, end with a break signal.
uart_write_bytes_with_break(uart_num, "test break\n",strlen("test break\n"), 100);
```

Another function for writing data to the Tx FIFO buffer is *uart_tx_chars()*. Unlike *uart_write_bytes()*, this function will not block until space is available. Instead, it will write all data which can immediately fit into the hardware Tx FIFO, and then return the number of bytes that were written.

There is a 'companion' function *uart_wait_tx_done()* that monitors the status of the Tx FIFO buffer and returns once it is empty.

```
// Wait for packet to be sent
const int uart_num = UART_NUM_2;
ESP_ERROR_CHECK(uart_wait_tx_done(uart_num, 100)); // wait timeout is 100 RTOS ticks␣
↪(TickType_t)
```

Receiving

Once the data is received by the UART and saved in the Rx FIFO buffer, it needs to be retrieved using the function *uart_read_bytes()*. Before reading data, you can check the number of bytes available in the Rx FIFO buffer by calling *uart_get_buffered_data_len()*. An example of using these functions is given below.

```
// Read data from UART.
const int uart_num = UART_NUM_2;
uint8_t data[128];
int length = 0;
ESP_ERROR_CHECK(uart_get_buffered_data_len(uart_num, (size_t*)&length));
length = uart_read_bytes(uart_num, data, length, 100);
```

If the data in the Rx FIFO buffer is no longer needed, you can clear the buffer by calling *uart_flush()*.

Software Flow Control

If the hardware flow control is disabled, you can manually set the RTS and DTR signal levels by using the functions *uart_set_rts()* and *uart_set_dtr()* respectively.

Communication Mode Selection

The UART controller supports a number of communication modes. A mode can be selected using the function *uart_set_mode()*. Once a specific mode is selected, the UART driver will handle the behavior of

a connected UART device accordingly. As an example, it can control the RS485 driver chip using the RTS line to allow half-duplex RS485 communication.

```
// Setup UART in rs485 half duplex mode
ESP_ERROR_CHECK(uart_set_mode(uart_num, UART_MODE_RS485_HALF_DUPLEX));
```

Using Interrupts

There are many interrupts that can be generated following specific UART states or detected errors. The full list of available interrupts is provided in ESP32 Technical Reference Manual (PDF). You can enable or disable specific interrupts by calling *uart_enable_intr_mask()* or *uart_disable_intr_mask()* respectively. The mask of all interrupts is available as `UART_INTR_MASK`.

By default, the *uart_driver_install()* function installs the driver' s internal interrupt handler to manage the Tx and Rx ring buffers and provides high-level API functions like events (see below). It is also possible to register a lower level interrupt handler instead using *uart_isr_register()*, and to free it again using *uart_isr_free()*. Some UART driver functions which use the Tx and Rx ring buffers, events, etc. will not automatically work in this case - it is necessary to handle the interrupts directly in the ISR. Inside the custom handler implementation, clear the interrupt status bits using *uart_clear_intr_status()*.

The API provides a convenient way to handle specific interrupts discussed in this document by wrapping them into dedicated functions:

- **Event detection**: There are several events defined in *uart_event_type_t* that may be reported to a user application using the FreeRTOS queue functionality. You can enable this functionality when calling *uart_driver_install()* described in *Driver Installation*. An example of using Event detection can be found in peripherals/uart_events.
- **FIFO space threshold or transmission timeout reached**: The Tx and Rx FIFO buffers can trigger an interrupt when they are filled with a specific number of characters, or on a timeout of sending or receiving data. To use these interrupts, do the following:
  - Configure respective threshold values of the buffer length and timeout by entering them in the structure *uart_intr_config_t* and calling *uart_intr_config()*
  - Enable the interrupts using the functions *uart_enable_tx_intr()* and *uart_enable_rx_intr()*
  - Disable these interrupts using the corresponding functions *uart_disable_tx_intr()* or *uart_disable_rx_intr()*
- **Pattern detection**: An interrupt triggered on detecting a 'pattern' of the same character being received/sent repeatedly for a number of times. This functionality is demonstrated in the example peripherals/uart/uart_events. It can be used, e.g., to detect a command string followed by a specific number of identical characters (the 'pattern') added at the end of the command string. The following functions are available:
  - Configure and enable this interrupt using `uart_enable_pattern_det_intr()`

– Disable the interrupt using *uart_disable_pattern_det_intr()*

Macros

The API also defines several macros. For example, UART_FIFO_LEN defines the length of hardware FIFO buffers; UART_BITRATE_MAX gives the maximum baud rate supported by the UART controllers, etc.

Deleting a Driver

If the communication established with *uart_driver_install()* is no longer required, the driver can be removed to free allocated resources by calling *uart_driver_delete()*.

Overview of RS485 specific communication options

注解: The following section will use [UART_REGISTER_NAME].[UART_FIELD_BIT] to refer to UART register fields/bits. To find more information on a specific option bit, open Register Summary in the ESP32 Technical Reference Manual (PDF), use the register name to navigate to the register description and then find the field/bit.

- UART_RS485_CONF_REG.UART_RS485_EN: setting this bit enables RS485 communication mode support.
- UART_RS485_CONF_REG.UART_RS485TX_RX_EN: if this bit is set, the transmitter's output signal loops back to the receiver's input signal.
- UART_RS485_CONF_REG.UART_RS485RXBY_TX_EN: if this bit is set, the transmitter will still be sending data if the receiver is busy (remove collisions automatically by hardware).

The ESP32's RS485 UART hardware can detect signal collisions during transmission of a datagram and generate the interrupt UART_RS485_CLASH_INT if this interrupt is enabled. The term collision means that a transmitted datagram is not equal to the one received on the other end. Data collisions are usually associated with the presence of other active devices on the bus or might occur due to bus errors.

The collision detection feature allows handling collisions when their interrupts are activated and triggered. The interrupts UART_RS485_FRM_ERR_INT and UART_RS485_PARITY_ERR_INT can be used with the collision detection feature to control frame errors and parity bit errors accordingly in RS485 mode. This functionality is supported in the UART driver and can be used by selecting the *UART_MODE_RS485_APP_CTRL* mode (see the function *uart_set_mode()*).

The collision detection feature can work with circuit A and circuit C (see Section *Interface Connection Options*). In the case of using circuit A or B, the RTS pin connected to the DE pin of the bus driver should be controlled by the user application. Use the function *uart_get_collision_flag()* to check if the collision detection flag has been raised.

The ESP32' s UART controllers themselves do not support half-duplex communication as they cannot provide automatic control of the RTS pin connected to the ~RE/DE input of RS485 bus driver. However, half-duplex communication can be achieved via software control of the RTS pin by the UART driver. This can be enabled by selecting the *UART_MODE_RS485_HALF_DUPLEX* mode when calling *uart_set_mode()*.

Once the host starts writing data to the Tx FIFO buffer, the UART driver automatically asserts the RTS pin (logic 1); once the last bit of the data has been transmitted, the driver de-asserts the RTS pin (logic 0). To use this mode, the software would have to disable the hardware flow control function. This mode works with all the used circuits shown below.

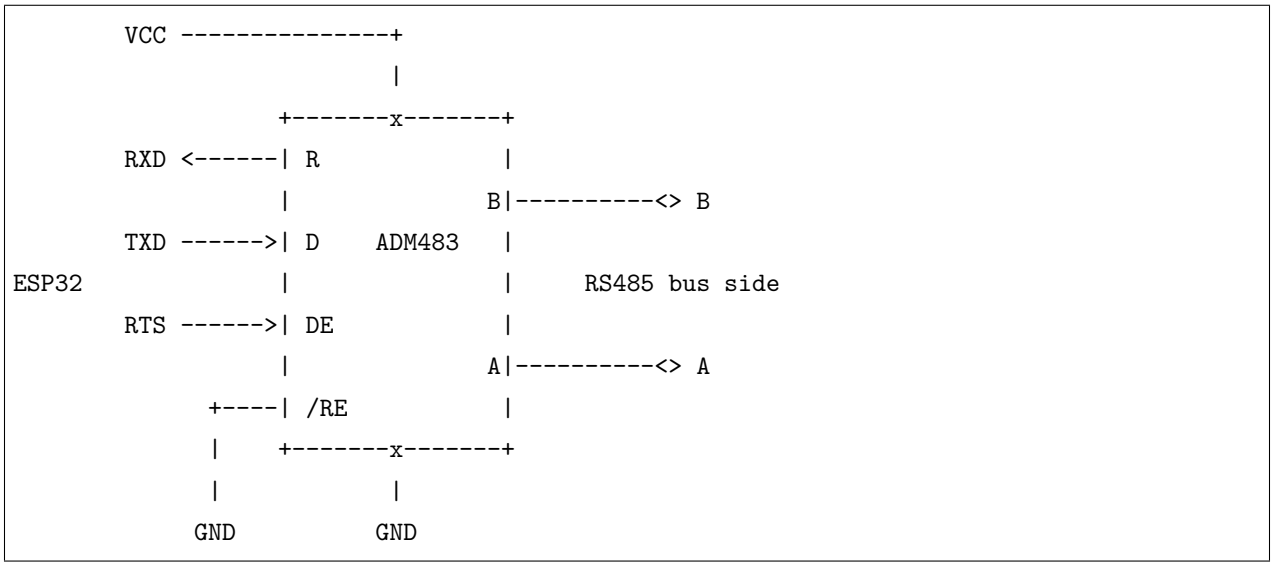
Interface Connection Options

This section provides example schematics to demonstrate the basic aspects of ESP32' s RS485 interface connection.

注解:

- The schematics below do **not** necessarily contain **all required elements**.
- The **analog devices** ADM483 & ADM2483 are examples of common RS485 transceivers and **can be replaced** with other similar transceivers.

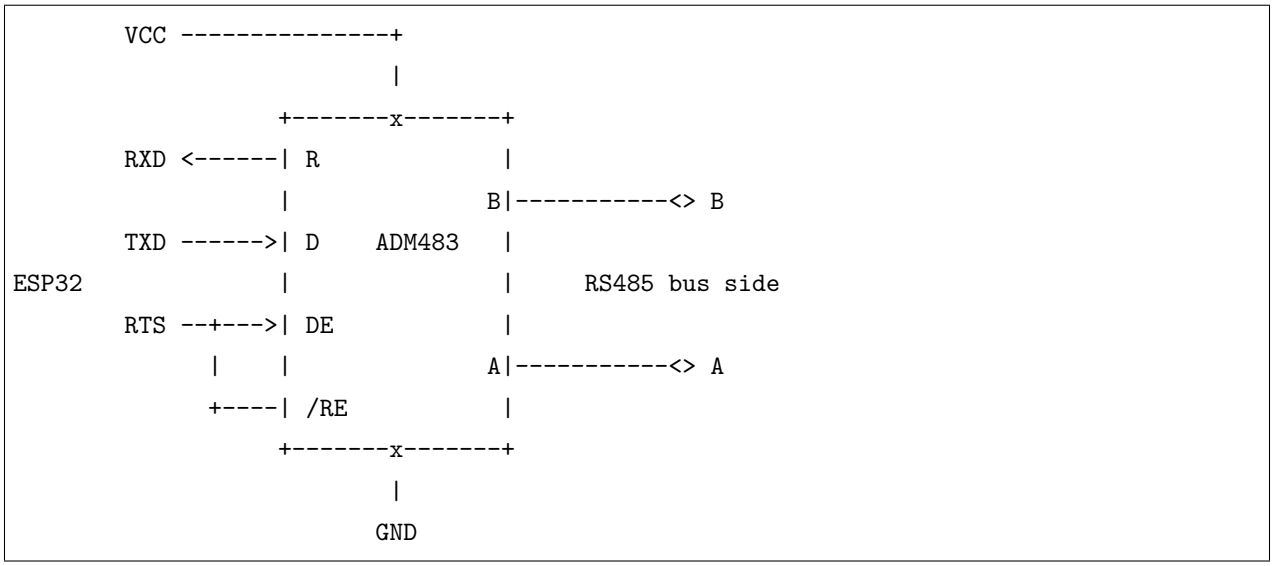
Circuit A: Collision Detection Circuit



This circuit is preferable because it allows for collision detection and is quite simple at the same time. The receiver in the line driver is constantly enabled, which allows the UART to monitor the RS485 bus. Echo

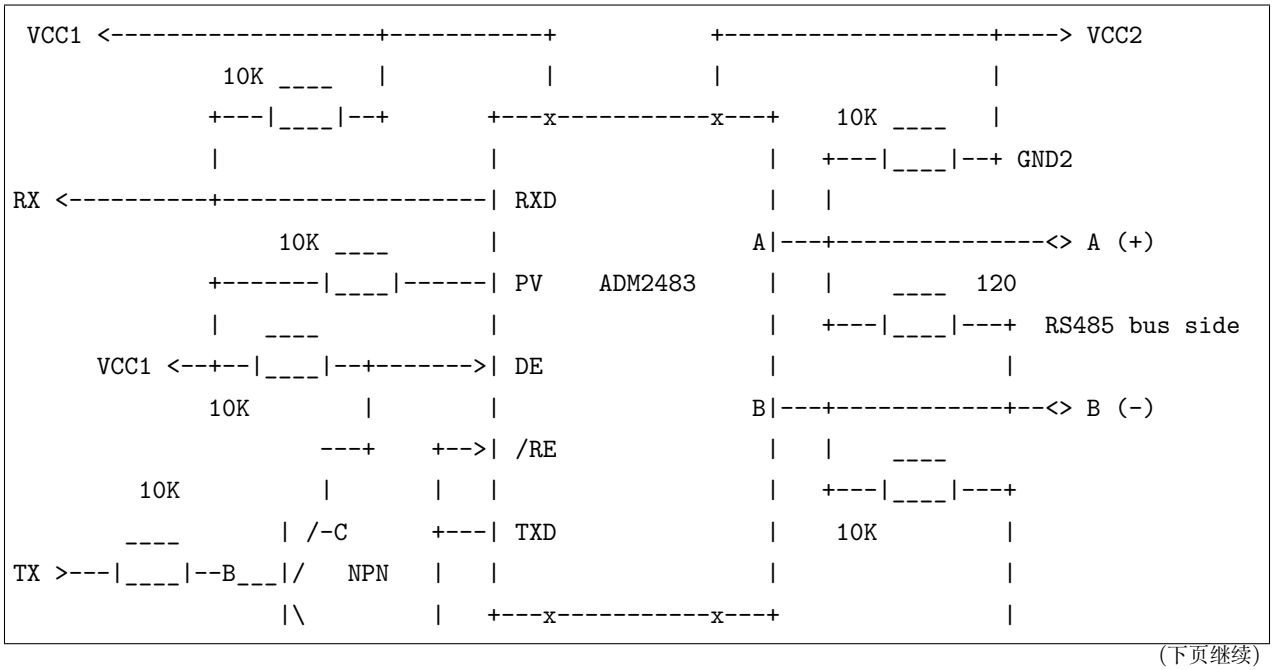
suppression is performed by the ESP32 hardware when the bit `UART_RS485_CONF_REG.UART_RS485TX_RX_EN` is enabled.

Circuit B: Manual Switching Transmitter/Receiver Without Collision Detection



This circuit does not allow for collision detection. It suppresses the null bytes that the hardware receives when the bit `UART_RS485_CONF_REG.UART_RS485TX_RX_EN` is set. The bit `UART_RS485_CONF_REG.UART_RS485RXBY_TX_EN` is not applicable in this case.

Circuit C: Auto Switching Transmitter/Receiver



(下页继续)

(续上页)

```
                    | \-E      |       |           |                    |
                      |       |       |           |                    |
                     GND1    GND1    GND1        GND2                 GND2
```

This galvanically isolated circuit does not require RTS pin control by a software application or driver because it controls the transceiver direction automatically. However, it requires suppressing null bytes during transmission by setting `UART_RS485_CONF_REG.UART_RS485RXBY_TX_EN` to 1 and `UART_RS485_CONF_REG.UART_RS485TX_RX_EN` to 0. This setup can work in any RS485 UART mode or even in *UART_MODE_UART*.

Application Examples

The table below describes the code examples available in the directory peripherals/uart/.

Code Example	Description
peripherals/uart/uart_echo	Configuring UART settings, installing the UART driver, and reading/writing over the UART1 interface.
peripherals/uart/uart_events	Reporting various communication events, using pattern detection interrupts.
peripherals/uart/uart_async_rxtxtasks	Transmitting and receiving data in two separate FreeRTOS tasks over the same UART.
peripherals/uart/uart_select	Using synchronous I/O multiplexing for UART file descriptors.
peripherals/uart/uart_echo_rs485	Setting up UART driver to communicate over RS485 interface in half-duplex mode. This example is similar to peripherals/uart/uart_echo but allows communication through an RS485 interface chip connected to ESP32 pins.
peripherals/uart/nmea0183_parser	Obtaining GPS information by parsing NMEA0183 statements received from GPS via the UART peripheral.

API Reference

Header File

- driver/include/driver/uart.h

Functions

esp_err_t **uart_driver_install**(*uart_port_t uart_num*, int *rx_buffer_size*, int *tx_buffer_size*, int *queue_size*, *QueueHandle_t* **uart_queue*, int *intr_alloc_flags*)

Install UART driver and set the UART to the default configuration.

UART ISR handler will be attached to the same CPU core that this function is running on.

Note Rx_buffer_size should be greater than UART_FIFO_LEN. Tx_buffer_size should be either zero or greater than UART_FIFO_LEN.

Return

- ESP_OK Success
- ESP_FAIL Parameter error

Parameters

- `uart_num`: UART port number, the max port number is (UART_NUM_MAX -1).
- `rx_buffer_size`: UART RX ring buffer size.
- `tx_buffer_size`: UART TX ring buffer size. If set to zero, driver will not use TX buffer, TX function will block task until all data have been sent out.
- `queue_size`: UART event queue size/depth.
- `uart_queue`: UART event queue handle (out param). On success, a new queue handle is written here to provide access to UART events. If set to NULL, driver will not use an event queue.
- `intr_alloc_flags`: Flags used to allocate the interrupt. One or multiple (ORred) ESP_INTR_FLAG_* values. See esp_intr_alloc.h for more info. Do not set ESP_INTR_FLAG_IRAM here (the driver's ISR handler is not located in IRAM)

esp_err_t **uart_driver_delete**(*uart_port_t uart_num*)

Uninstall UART driver.

Return

- ESP_OK Success
- ESP_FAIL Parameter error

Parameters

- `uart_num`: UART port number, the max port number is (UART_NUM_MAX -1).

bool **uart_is_driver_installed**(*uart_port_t uart_num*)

Checks whether the driver is installed or not.

Return

- true driver is installed
- false driver is not installed

Parameters

- `uart_num`: UART port number, the max port number is (UART_NUM_MAX -1).

esp_err_t **uart_set_word_length**(*uart_port_t uart_num*, *uart_word_length_t data_bit*)

Set UART data bits.

Return

- ESP_OK Success
- ESP_FAIL Parameter error

Parameters

- `uart_num`: UART port number, the max port number is (UART_NUM_MAX -1).
- `data_bit`: UART data bits

esp_err_t **uart_get_word_length**(*uart_port_t uart_num*, *uart_word_length_t *data_bit*)

Get the UART data bit configuration.

Return

- ESP_FAIL Parameter error
- ESP_OK Success, result will be put in (*data_bit)

Parameters

- `uart_num`: UART port number, the max port number is (UART_NUM_MAX -1).
- `data_bit`: Pointer to accept value of UART data bits.

esp_err_t **uart_set_stop_bits**(*uart_port_t uart_num*, *uart_stop_bits_t stop_bits*)

Set UART stop bits.

Return

- ESP_OK Success
- ESP_FAIL Fail

Parameters

- `uart_num`: UART port number, the max port number is (UART_NUM_MAX -1).
- `stop_bits`: UART stop bits

esp_err_t **uart_get_stop_bits**(*uart_port_t uart_num*, *uart_stop_bits_t *stop_bits*)

Get the UART stop bit configuration.

Return

- ESP_FAIL Parameter error
- ESP_OK Success, result will be put in (*stop_bit)

Parameters

- `uart_num`: UART port number, the max port number is (UART_NUM_MAX -1).
- `stop_bits`: Pointer to accept value of UART stop bits.

esp_err_t **uart_set_parity**(*uart_port_t uart_num*, *uart_parity_t parity_mode*)

Set UART parity mode.

Return

- ESP_FAIL Parameter error
- ESP_OK Success

Parameters

- `uart_num`: UART port number, the max port number is (UART_NUM_MAX -1).
- `parity_mode`: the enum of uart parity configuration

esp_err_t **uart_get_parity**(*uart_port_t uart_num*, *uart_parity_t *parity_mode*)

Get the UART parity mode configuration.

Return

- ESP_FAIL Parameter error
- ESP_OK Success, result will be put in (*parity_mode)

Parameters

- `uart_num`: UART port number, the max port number is (UART_NUM_MAX -1).
- `parity_mode`: Pointer to accept value of UART parity mode.

esp_err_t **uart_set_baudrate**(*uart_port_t uart_num*, uint32_t *baudrate*)

Set UART baud rate.

Return

- ESP_FAIL Parameter error
- ESP_OK Success

Parameters

- `uart_num`: UART port number, the max port number is (UART_NUM_MAX -1).
- `baudrate`: UART baud rate.

esp_err_t **uart_get_baudrate**(*uart_port_t uart_num*, uint32_t *baudrate*)

Get the UART baud rate configuration.

Return

- ESP_FAIL Parameter error
- ESP_OK Success, result will be put in (*baudrate)

Parameters

- `uart_num`: UART port number, the max port number is (UART_NUM_MAX -1).
- `baudrate`: Pointer to accept value of UART baud rate

esp_err_t **uart_set_line_inverse**(*uart_port_t uart_num*, uint32_t *inverse_mask*)

Set UART line inverse mode.

Return

- ESP_OK Success
- ESP_FAIL Parameter error

Parameters

- `uart_num`: UART port number, the max port number is (UART_NUM_MAX -1).
- `inverse_mask`: Choose the wires that need to be inverted. Using the ORred mask of `uart_signal_inv_t`

esp_err_t **uart_set_hw_flow_ctrl**(*uart_port_t uart_num*, *uart_hw_flowcontrol_t flow_ctrl*, uint8_t *rx_thresh*)

Set hardware flow control.

Return

- ESP_OK Success
- ESP_FAIL Parameter error

Parameters

- `uart_num`: UART port number, the max port number is (UART_NUM_MAX -1).
- `flow_ctrl`: Hardware flow control mode
- `rx_thresh`: Threshold of Hardware RX flow control (0 ~ UART_FIFO_LEN). Only when UART_HW_FLOWCTRL_RTS is set, will the rx_thresh value be set.

esp_err_t **uart_set_sw_flow_ctrl**(*uart_port_t uart_num*, bool *enable*, uint8_t *rx_thresh_xon*, uint8_t *rx_thresh_xoff*)

Set software flow control.

Return

- ESP_OK Success

- ESP_FAIL Parameter error

Parameters

- `uart_num`: UART_NUM_0, UART_NUM_1 or UART_NUM_2
- `enable`: switch on or off
- `rx_thresh_xon`: low water mark
- `rx_thresh_xoff`: high water mark

esp_err_t **uart_get_hw_flow_ctrl**(*uart_port_t uart_num*, *uart_hw_flowcontrol_t *flow_ctrl*)

Get the UART hardware flow control configuration.

Return

- ESP_FAIL Parameter error
- ESP_OK Success, result will be put in (*flow_ctrl)

Parameters

- `uart_num`: UART port number, the max port number is (UART_NUM_MAX -1).
- `flow_ctrl`: Option for different flow control mode.

esp_err_t **uart_clear_intr_status**(*uart_port_t uart_num*, uint32_t *clr_mask*)

Clear UART interrupt status.

Return

- ESP_OK Success
- ESP_FAIL Parameter error

Parameters

- `uart_num`: UART port number, the max port number is (UART_NUM_MAX -1).
- `clr_mask`: Bit mask of the interrupt status to be cleared.

esp_err_t **uart_enable_intr_mask**(*uart_port_t uart_num*, uint32_t *enable_mask*)

Set UART interrupt enable.

Return

- ESP_OK Success
- ESP_FAIL Parameter error

Parameters

- `uart_num`: UART port number, the max port number is (UART_NUM_MAX -1).

- `enable_mask`: Bit mask of the enable bits.

esp_err_t **uart_disable_intr_mask**(*uart_port_t uart_num*, uint32_t *disable_mask*)

Clear UART interrupt enable bits.

Return

- ESP_OK Success
- ESP_FAIL Parameter error

Parameters

- `uart_num`: UART port number, the max port number is (UART_NUM_MAX -1).
- `disable_mask`: Bit mask of the disable bits.

esp_err_t **uart_enable_rx_intr**(*uart_port_t uart_num*)

Enable UART RX interrupt (RX_FULL & RX_TIMEOUT INTERRUPT)

Return

- ESP_OK Success
- ESP_FAIL Parameter error

Parameters

- `uart_num`: UART port number, the max port number is (UART_NUM_MAX -1).

esp_err_t **uart_disable_rx_intr**(*uart_port_t uart_num*)

Disable UART RX interrupt (RX_FULL & RX_TIMEOUT INTERRUPT)

Return

- ESP_OK Success
- ESP_FAIL Parameter error

Parameters

- `uart_num`: UART port number, the max port number is (UART_NUM_MAX -1).

esp_err_t **uart_disable_tx_intr**(*uart_port_t uart_num*)

Disable UART TX interrupt (TX_FULL & TX_TIMEOUT INTERRUPT)

Return

- ESP_OK Success
- ESP_FAIL Parameter error

Parameters

- `uart_num`: UART port number

esp_err_t **uart_enable_tx_intr**(*uart_port_t uart_num*, int *enable*, int *thresh*)

Enable UART TX interrupt (TX_FULL & TX_TIMEOUT INTERRUPT)

Return

- ESP_OK Success
- ESP_FAIL Parameter error

Parameters

- `uart_num`: UART port number, the max port number is (UART_NUM_MAX -1).
- `enable`: 1: enable; 0: disable
- `thresh`: Threshold of TX interrupt, 0 ~ UART_FIFO_LEN

esp_err_t **uart_isr_register**(*uart_port_t uart_num*, void (**fn*))void *

, void **arg*, int *intr_alloc_flags*, *uart_isr_handle_t* **handle*Register UART interrupt handler (ISR).

Note UART ISR handler will be attached to the same CPU core that this function is running on.

Return

- ESP_OK Success
- ESP_FAIL Parameter error

Parameters

- `uart_num`: UART port number, the max port number is (UART_NUM_MAX -1).
- `fn`: Interrupt handler function.
- `arg`: parameter for handler function
- `intr_alloc_flags`: Flags used to allocate the interrupt. One or multiple (ORred) ESP_INTR_FLAG_* values. See esp_intr_alloc.h for more info.
- `handle`: Pointer to return handle. If non-NULL, a handle for the interrupt will be returned here.

esp_err_t **uart_isr_free**(*uart_port_t uart_num*)

Free UART interrupt handler registered by uart_isr_register. Must be called on the same core as uart_isr_register was called.

Return

- ESP_OK Success
- ESP_FAIL Parameter error

Parameters

- `uart_num`: UART port number, the max port number is (UART_NUM_MAX -1).

esp_err_t **uart_set_pin**(*uart_port_t uart_num*, int *tx_io_num*, int *rx_io_num*, int *rts_io_num*, int *cts_io_num*)

Set UART pin number.

Note Internal signal can be output to multiple GPIO pads. Only one GPIO pad can connect with input signal.

Note Instead of GPIO number a macro 'UART_PIN_NO_CHANGE' may be provided to keep the currently allocated pin.

Return

- ESP_OK Success
- ESP_FAIL Parameter error

Parameters

- `uart_num`: UART port number, the max port number is (UART_NUM_MAX -1).
- `tx_io_num`: UART TX pin GPIO number.
- `rx_io_num`: UART RX pin GPIO number.
- `rts_io_num`: UART RTS pin GPIO number.
- `cts_io_num`: UART CTS pin GPIO number.

esp_err_t **uart_set_rts**(*uart_port_t uart_num*, int *level*)

Manually set the UART RTS pin level.

Note UART must be configured with hardware flow control disabled.

Return

- ESP_OK Success
- ESP_FAIL Parameter error

Parameters

- `uart_num`: UART port number, the max port number is (UART_NUM_MAX -1).
- `level`: 1: RTS output low (active); 0: RTS output high (block)

esp_err_t **uart_set_dtr**(*uart_port_t uart_num*, int *level*)

Manually set the UART DTR pin level.

Return

- ESP_OK Success
- ESP_FAIL Parameter error

Parameters

- `uart_num`: UART port number, the max port number is (UART_NUM_MAX -1).
- `level`: 1: DTR output low; 0: DTR output high

esp_err_t **uart_set_tx_idle_num**(*uart_port_t uart_num*, uint16_t *idle_num*)

Set UART idle interval after tx FIFO is empty.

Return

- ESP_OK Success
- ESP_FAIL Parameter error

Parameters

- `uart_num`: UART port number, the max port number is (UART_NUM_MAX -1).
- `idle_num`: idle interval after tx FIFO is empty(unit: the time it takes to send one bit under current baudrate)

esp_err_t **uart_param_config**(*uart_port_t uart_num*, **const** *uart_config_t* **uart_config*)

Set UART configuration parameters.

Return

- ESP_OK Success
- ESP_FAIL Parameter error

Parameters

- `uart_num`: UART port number, the max port number is (UART_NUM_MAX -1).
- `uart_config`: UART parameter settings

esp_err_t **uart_intr_config**(*uart_port_t uart_num*, **const** *uart_intr_config_t* **intr_conf*)

Configure UART interrupts.

Return

- ESP_OK Success
- ESP_FAIL Parameter error

Parameters

- `uart_num`: UART port number, the max port number is (UART_NUM_MAX -1).
- `intr_conf`: UART interrupt settings

esp_err_t **uart_wait_tx_done**(*uart_port_t uart_num*, TickType_t *ticks_to_wait*)

Wait until UART TX FIFO is empty.

Return

- ESP_OK Success
- ESP_FAIL Parameter error
- ESP_ERR_TIMEOUT Timeout

Parameters

- `uart_num`: UART port number, the max port number is (UART_NUM_MAX -1).
- `ticks_to_wait`: Timeout, count in RTOS ticks

int **uart_tx_chars**(*uart_port_t uart_num*, **const** char ***buffer*, uint32_t *len*)

Send data to the UART port from a given buffer and length.

This function will not wait for enough space in TX FIFO. It will just fill the available TX FIFO and return when the FIFO is full.

Note This function should only be used when UART TX buffer is not enabled.

Return

- (-1) Parameter error
- OTHERS (>=0) The number of bytes pushed to the TX FIFO

Parameters

- `uart_num`: UART port number, the max port number is (UART_NUM_MAX -1).
- `buffer`: data buffer address
- `len`: data length to send

int **uart_write_bytes**(*uart_port_t uart_num*, **const** char ***src*, size_t *size*)

Send data to the UART port from a given buffer and length,.

If the UART driver' s parameter 'tx_buffer_size' is set to zero: This function will not return until all the data have been sent out, or at least pushed into TX FIFO.

Otherwise, if the 'tx_buffer_size' > 0, this function will return after copying all the data to tx ring buffer, UART ISR will then move data from the ring buffer to TX FIFO gradually.

Return

- (-1) Parameter error
- OTHERS (>=0) The number of bytes pushed to the TX FIFO

Parameters

- `uart_num`: UART port number, the max port number is (UART_NUM_MAX -1).
- `src`: data buffer address
- `size`: data length to send

int **uart_write_bytes_with_break**(*uart_port_t uart_num*, **const** char **src*, size_t *size*, int *brk_len*)

Send data to the UART port from a given buffer and length,.

If the UART driver' s parameter 'tx_buffer_size' is set to zero: This function will not return until all the data and the break signal have been sent out. After all data is sent out, send a break signal.

Otherwise, if the 'tx_buffer_size' > 0, this function will return after copying all the data to tx ring buffer, UART ISR will then move data from the ring buffer to TX FIFO gradually. After all data sent out, send a break signal.

Return

- (-1) Parameter error
- OTHERS (>=0) The number of bytes pushed to the TX FIFO

Parameters

- `uart_num`: UART port number, the max port number is (UART_NUM_MAX -1).
- `src`: data buffer address
- `size`: data length to send
- `brk_len`: break signal duration(unit: the time it takes to send one bit at current baudrate)

int **uart_read_bytes**(*uart_port_t uart_num*, uint8_t **buf*, uint32_t *length*, TickType_t *ticks_to_wait*)

UART read bytes from UART buffer.

Return

- (-1) Error
- OTHERS (>=0) The number of bytes read from UART FIFO

Parameters

- `uart_num`: UART port number, the max port number is (UART_NUM_MAX -1).
- `buf`: pointer to the buffer.
- `length`: data length
- `ticks_to_wait`: sTimeout, count in RTOS ticks

esp_err_t **uart_flush**(*uart_port_t uart_num*)

Alias of uart_flush_input. UART ring buffer flush. This will discard all data in the UART RX buffer.

Note Instead of waiting the data sent out, this function will clear UART rx buffer. In order to send all the data in tx FIFO, we can use uart_wait_tx_done function.

Return

- ESP_OK Success
- ESP_FAIL Parameter error

Parameters

- `uart_num`: UART port number, the max port number is (UART_NUM_MAX -1).

esp_err_t **uart_flush_input**(*uart_port_t uart_num*)

Clear input buffer, discard all the data is in the ring-buffer.

Note In order to send all the data in tx FIFO, we can use uart_wait_tx_done function.

Return

- ESP_OK Success
- ESP_FAIL Parameter error

Parameters

- `uart_num`: UART port number, the max port number is (UART_NUM_MAX -1).

esp_err_t **uart_get_buffered_data_len**(*uart_port_t uart_num*, size_t **size*)

UART get RX ring buffer cached data length.

Return

- ESP_OK Success
- ESP_FAIL Parameter error

Parameters

- `uart_num`: UART port number, the max port number is (UART_NUM_MAX -1).
- `size`: Pointer of size_t to accept cached data length

esp_err_t **uart_disable_pattern_det_intr**(*uart_port_t uart_num*)

UART disable pattern detect function. Designed for applications like ‘AT commands’. When the hardware detects a series of one same character, the interrupt will be triggered.

Return

- ESP_OK Success
- ESP_FAIL Parameter error

Parameters

- `uart_num`: UART port number, the max port number is (UART_NUM_MAX -1).

esp_err_t **uart_enable_pattern_det_baud_intr**(*uart_port_t uart_num*, char *pattern_chr*, uint8_t *chr_num*, int *chr_tout*, int *post_idle*, int *pre_idle*)

UART enable pattern detect function. Designed for applications like 'AT commands'. When the hardware detect a series of one same character, the interrupt will be triggered.

Return

- ESP_OK Success
- ESP_FAIL Parameter error

Parameters

- `uart_num`: UART port number.
- `pattern_chr`: character of the pattern.
- `chr_num`: number of the character, 8bit value.
- `chr_tout`: timeout of the interval between each pattern characters, 16bit value, unit is the baud-rate cycle you configured. When the duration is more than this value, it will not take this data as at_cmd char.
- `post_idle`: idle time after the last pattern character, 16bit value, unit is the baud-rate cycle you configured. When the duration is less than this value, it will not take the previous data as the last at_cmd char
- `pre_idle`: idle time before the first pattern character, 16bit value, unit is the baud-rate cycle you configured. When the duration is less than this value, it will not take this data as the first at_cmd char.

int **uart_pattern_pop_pos**(*uart_port_t uart_num*)

Return the nearest detected pattern position in buffer. The positions of the detected pattern are saved in a queue, this function will dequeue the first pattern position and move the pointer to next pattern position.

The following APIs will modify the pattern position info: uart_flush_input, uart_read_bytes, uart_driver_delete, uart_pop_pattern_pos It is the application's responsibility to ensure atomic access to the pattern queue and the rx data buffer when using pattern detect feature.

Note If the RX buffer is full and flow control is not enabled, the detected pattern may not be found in the rx buffer due to overflow.

Return

- (-1) No pattern found for current index or parameter error
- others the pattern position in rx buffer.

Parameters

- `uart_num`: UART port number, the max port number is (UART_NUM_MAX -1).

int **uart_pattern_get_pos**(*uart_port_t uart_num*)

Return the nearest detected pattern position in buffer. The positions of the detected pattern are saved in a queue, This function do nothing to the queue.

The following APIs will modify the pattern position info: uart_flush_input, uart_read_bytes, uart_driver_delete, uart_pop_pattern_pos It is the application' s responsibility to ensure atomic access to the pattern queue and the rx data buffer when using pattern detect feature.

Note If the RX buffer is full and flow control is not enabled, the detected pattern may not be found in the rx buffer due to overflow.

Return

- (-1) No pattern found for current index or parameter error
- others the pattern position in rx buffer.

Parameters

- `uart_num`: UART port number, the max port number is (UART_NUM_MAX -1).

esp_err_t **uart_pattern_queue_reset**(*uart_port_t uart_num*, int *queue_length*)

Allocate a new memory with the given length to save record the detected pattern position in rx buffer.

Return

- ESP_ERR_NO_MEM No enough memory
- ESP_ERR_INVALID_STATE Driver not installed
- ESP_FAIL Parameter error
- ESP_OK Success

Parameters

- `uart_num`: UART port number, the max port number is (UART_NUM_MAX -1).
- `queue_length`: Max queue length for the detected pattern. If the queue length is not large enough, some pattern positions might be lost. Set this value to the maximum number of patterns that could be saved in data buffer at the same time.

esp_err_t **uart_set_mode**(*uart_port_t uart_num*, *uart_mode_t mode*)

UART set communication mode.

Note This function must be executed after uart_driver_install(), when the driver object is initialized.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `uart_num`: Uart number to configure, the max port number is (UART_NUM_MAX -1).
- `mode`: UART UART mode to set

esp_err_t **uart_set_rx_full_threshold**(*uart_port_t uart_num*, int *threshold*)

Set uart threshold value for RX fifo full.

Note If application is using higher baudrate and it is observed that bytes in hardware RX fifo are overwritten then this threshold can be reduced

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_ERR_INVALID_STATE Driver is not installed

Parameters

- `uart_num`: UART_NUM_0, UART_NUM_1 or UART_NUM_2
- `threshold`: Threshold value above which RX fifo full interrupt is generated

esp_err_t **uart_set_tx_empty_threshold**(*uart_port_t uart_num*, int *threshold*)

Set uart threshold values for TX fifo empty.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_ERR_INVALID_STATE Driver is not installed

Parameters

- `uart_num`: UART_NUM_0, UART_NUM_1 or UART_NUM_2
- `threshold`: Threshold value below which TX fifo empty interrupt is generated

esp_err_t **uart_set_rx_timeout**(*uart_port_t uart_num*, **const** uint8_t *tout_thresh*)

UART set threshold timeout for TOUT feature.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

- ESP_ERR_INVALID_STATE Driver is not installed

Parameters

- `uart_num`: Uart number to configure, the max port number is (UART_NUM_MAX -1).
- `tout_thresh`: This parameter defines timeout threshold in uart symbol periods. The maximum value of threshold is 126. tout_thresh = 1, defines TOUT interrupt timeout equal to transmission time of one symbol (~11 bit) on current baudrate. If the time is expired the UART_RXFIFO_TOUT_INT interrupt is triggered. If tout_thresh == 0, the TOUT feature is disabled.

esp_err_t **uart_get_collision_flag**(*uart_port_t uart_num*, bool **collision_flag*)

Returns collision detection flag for RS485 mode Function returns the collision detection flag into variable pointed by collision_flag. *collision_flag = true, if collision detected else it is equal to false. This function should be executed when actual transmission is completed (after uart_write_bytes()).

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `uart_num`: Uart number to configure the max port number is (UART_NUM_MAX -1).
- `collision_flag`: Pointer to variable of type bool to return collision flag.

esp_err_t **uart_set_wakeup_threshold**(*uart_port_t uart_num*, int *wakeup_threshold*)

Set the number of RX pin signal edges for light sleep wakeup.

UART can be used to wake up the system from light sleep. This feature works by counting the number of positive edges on RX pin and comparing the count to the threshold. When the count exceeds the threshold, system is woken up from light sleep. This function allows setting the threshold value.

Stop bit and parity bits (if enabled) also contribute to the number of edges. For example, letter 'a' with ASCII code 97 is encoded as 0100001101 on the wire (with 8n1 configuration), start and stop bits included. This sequence has 3 positive edges (transitions from 0 to 1). Therefore, to wake up the system when 'a' is sent, set wakeup_threshold=3.

The character that triggers wakeup is not received by UART (i.e. it can not be obtained from UART FIFO). Depending on the baud rate, a few characters after that will also not be received. Note that when the chip enters and exits light sleep mode, APB frequency will be changing. To make sure that UART has correct baud rate all the time, select REF_TICK as UART clock source, by setting use_ref_tick field in *uart_config_t* to true.

Note in ESP32, the wakeup signal can only be input via IO_MUX (i.e. GPIO3 should be configured as function_1 to wake up UART0, GPIO9 should be configured as function_5 to wake up UART1),

UART2 does not support light sleep wakeup feature.

Return

- ESP_OK on success
- ESP_ERR_INVALID_ARG if uart_num is incorrect or wakeup_threshold is outside of [3, 0x3ff] range.

Parameters

- `uart_num`: UART number, the max port number is (UART_NUM_MAX -1).
- `wakeup_threshold`: number of RX edges for light sleep wakeup, value is 3 .. 0x3ff.

esp_err_t **uart_get_wakeup_threshold**(*uart_port_t uart_num*, int **out_wakeup_threshold*)

Get the number of RX pin signal edges for light sleep wakeup.

See description of uart_set_wakeup_threshold for the explanation of UART wakeup feature.

Return

- ESP_OK on success
- ESP_ERR_INVALID_ARG if out_wakeup_threshold is NULL

Parameters

- `uart_num`: UART number, the max port number is (UART_NUM_MAX -1).
- `out_wakeup_threshold`: output, set to the current value of wakeup threshold for the given UART.

esp_err_t **uart_wait_tx_idle_polling**(*uart_port_t uart_num*)

Wait until UART tx memory empty and the last char send ok (polling mode).

- **Return**
  - ESP_OK on success
  - ESP_ERR_INVALID_ARG Parameter error
  - ESP_FAIL Driver not installed

Parameters

- `uart_num`: UART number

esp_err_t **uart_set_loop_back**(*uart_port_t uart_num*, bool *loop_back_en*)

Configure TX signal loop back to RX module, just for the test usage.

- **Return**
  - ESP_OK on success

- ESP_ERR_INVALID_ARG Parameter error
- ESP_FAIL Driver not installed

Parameters

- `uart_num`: UART number
- `loop_back_en`: Set ture to enable the loop back function, else set it false.

Structures

struct uart_intr_config_t

UART interrupt configuration parameters for uart_intr_config function.

Public Members

uint32_t **intr_enable_mask**

UART interrupt enable mask, choose from UART_XXXX_INT_ENA_M under UART_INT_ENA_REG(i), connect with bit-or operator

uint8_t **rx_timeout_thresh**

UART timeout interrupt threshold (unit: time of sending one byte)

uint8_t **txfifo_empty_intr_thresh**

UART TX empty interrupt threshold.

uint8_t **rxfifo_full_thresh**

UART RX full interrupt threshold.

struct uart_event_t

Event structure used in UART event queue.

Public Members

uart_event_type_t **type**

UART event type

size_t **size**

UART data size for UART_DATA event

Macros

UART_NUM_0

UART port 0

`UART_NUM_1`
: UART port 1

`UART_NUM_MAX`
: UART port max

`UART_PIN_NO_CHANGE`
: Constant for uart_set_pin function which indicates that UART pin should not be changed

Type Definitions

`typedef` *intr_handle_t* `uart_isr_handle_t`

Enumerations

`enum uart_event_type_t`
: UART event types used in the ring buffer.

Values:

`UART_DATA`
: UART data event

`UART_BREAK`
: UART break event

`UART_BUFFER_FULL`
: UART RX buffer full event

`UART_FIFO_OVF`
: UART FIFO overflow event

`UART_FRAME_ERR`
: UART RX frame error event

`UART_PARITY_ERR`
: UART RX parity event

`UART_DATA_BREAK`
: UART TX data and break event

`UART_PATTERN_DET`
: UART pattern detected

`UART_EVENT_MAX`
: UART event max index

Header File

- soc/include/hal/uart_types.h

Structures

struct uart_at_cmd_t

UART AT cmd char configuration parameters Note that this function may different on different chip. Please refer to the TRM at confirguration.

Public Members

uint8_t **cmd_char**

UART AT cmd char

uint8_t **char_num**

AT cmd char repeat number

uint32_t **gap_tout**

gap time(in baud-rate) between AT cmd char

uint32_t **pre_idle**

the idle time(in baud-rate) between the non AT char and first AT char

uint32_t **post_idle**

the idle time(in baud-rate) between the last AT char and the none AT char

struct uart_sw_flowctrl_t

UART software flow control configuration parameters.

Public Members

uint8_t **xon_char**

Xon flow control char

uint8_t **xoff_char**

Xoff flow control char

uint8_t **xon_thrd**

If the software flow control is enabled and the data amount in rxfifo is less than xon_thrd, an xon_char will be sent

uint8_t **xoff_thrd**

If the software flow control is enabled and the data amount in rxfifo is more than xoff_thrd, an xoff_char will be sent

`struct uart_config_t`

UART configuration parameters for uart_param_config function.

Public Members

int **baud_rate**

UART baud rate

uart_word_length_t **data_bits**

UART byte size

uart_parity_t **parity**

UART parity mode

uart_stop_bits_t **stop_bits**

UART stop bits

uart_hw_flowcontrol_t **flow_ctrl**

UART HW flow control mode (cts/rts)

uint8_t **rx_flow_ctrl_thresh**

UART HW RTS threshold

uart_sclk_t **source_clk**

UART source clock selection

Type Definitions

`typedef` int **uart_port_t**

UART port number, can be UART_NUM_0 ~ (UART_NUM_MAX -1).

Enumerations

`enum uart_mode_t`

UART mode selection.

Values:

UART_MODE_UART = 0x00

mode: regular UART mode

UART_MODE_RS485_HALF_DUPLEX = 0x01

mode: half duplex RS485 UART mode control by RTS pin

UART_MODE_IRDA = 0x02

mode: IRDA UART mode

`UART_MODE_RS485_COLLISION_DETECT` = 0x03
mode: RS485 collision detection UART mode (used for test purposes)

`UART_MODE_RS485_APP_CTRL` = 0x04
mode: application control RS485 UART mode (used for test purposes)

enum uart_word_length_t
UART word length constants.

Values:

`UART_DATA_5_BITS` = 0x0
word length: 5bits

`UART_DATA_6_BITS` = 0x1
word length: 6bits

`UART_DATA_7_BITS` = 0x2
word length: 7bits

`UART_DATA_8_BITS` = 0x3
word length: 8bits

`UART_DATA_BITS_MAX` = 0x4

enum uart_stop_bits_t
UART stop bits number.

Values:

`UART_STOP_BITS_1` = 0x1
stop bit: 1bit

`UART_STOP_BITS_1_5` = 0x2
stop bit: 1.5bits

`UART_STOP_BITS_2` = 0x3
stop bit: 2bits

`UART_STOP_BITS_MAX` = 0x4

enum uart_parity_t
UART parity constants.

Values:

`UART_PARITY_DISABLE` = 0x0
Disable UART parity

`UART_PARITY_EVEN` = 0x2
Enable UART even parity

UART_PARITY_ODD = 0x3
Enable UART odd parity

enum uart_hw_flowcontrol_t
UART hardware flow control modes.

Values:

UART_HW_FLOWCTRL_DISABLE = 0x0
disable hardware flow control

UART_HW_FLOWCTRL_RTS = 0x1
enable RX hardware flow control (rts)

UART_HW_FLOWCTRL_CTS = 0x2
enable TX hardware flow control (cts)

UART_HW_FLOWCTRL_CTS_RTS = 0x3
enable hardware flow control

UART_HW_FLOWCTRL_MAX = 0x4

enum uart_signal_inv_t
UART signal bit map.

Values:

UART_SIGNAL_IRDA_TX_INV = (0x1 << 0)
inverse the UART irda_tx signal

UART_SIGNAL_IRDA_RX_INV = (0x1 << 1)
inverse the UART irda_rx signal

UART_SIGNAL_RXD_INV = (0x1 << 2)
inverse the UART rxd signal

UART_SIGNAL_CTS_INV = (0x1 << 3)
inverse the UART cts signal

UART_SIGNAL_DSR_INV = (0x1 << 4)
inverse the UART dsr signal

UART_SIGNAL_TXD_INV = (0x1 << 5)
inverse the UART txd signal

UART_SIGNAL_RTS_INV = (0x1 << 6)
inverse the UART rts signal

UART_SIGNAL_DTR_INV = (0x1 << 7)
inverse the UART dtr signal

enum uart_sclk_t
UART source clock.

Values:

`UART_SCLK_APB` = 0x0
UART source clock from APB

`UART_SCLK_REF_TICK` = 0x01
UART source clock from REF_TICK

GPIO Lookup Macros

The UART peripherals have dedicated IO_MUX pins to which they are connected directly. However, signals can also be routed to other pins using the less direct GPIO matrix. To use direct routes, you need to know which pin is a dedicated IO_MUX pin for a UART channel. GPIO Lookup Macros simplify the process of finding and assigning IO_MUX pins. You choose a macro based on either the IO_MUX pin number, or a required UART channel name, and the macro will return the matching counterpart for you. See some examples below.

注解: These macros are useful if you need very high UART baud rates (over 40 MHz), which means you will have to use IO_MUX pins only. In other cases, these macros can be ignored, and you can use the GPIO Matrix as it allows you to configure any GPIO pin for any UART function.

---

1. *UART_NUM_2_TXD_DIRECT_GPIO_NUM* returns the IO_MUX pin number of UART channel 2 TXD pin (pin 17)
2. *UART_GPIO19_DIRECT_CHANNEL* returns the UART number of GPIO 19 when connected to the UART peripheral via IO_MUX (this is UART_NUM_0)
3. *UART_CTS_GPIO19_DIRECT_CHANNEL* returns the UART number of GPIO 19 when used as the UART CTS pin via IO_MUX (this is UART_NUM_0). Similar to the above macro but specifies the pin function which is also part of the IO_MUX assignment.

Header File

- soc/esp32/include/soc/uart_channel.h

Macros

`UART_GPIO1_DIRECT_CHANNEL`

`UART_NUM_0_TXD_DIRECT_GPIO_NUM`

`UART_GPIO3_DIRECT_CHANNEL`

`UART_NUM_0_RXD_DIRECT_GPIO_NUM`

UART_GPIO19_DIRECT_CHANNEL

UART_NUM_0_CTS_DIRECT_GPIO_NUM

UART_GPIO22_DIRECT_CHANNEL

UART_NUM_0_RTS_DIRECT_GPIO_NUM

UART_TXD_GPIO1_DIRECT_CHANNEL

UART_RXD_GPIO3_DIRECT_CHANNEL

UART_CTS_GPIO19_DIRECT_CHANNEL

UART_RTS_GPIO22_DIRECT_CHANNEL

UART_GPIO10_DIRECT_CHANNEL

UART_NUM_1_TXD_DIRECT_GPIO_NUM

UART_GPIO9_DIRECT_CHANNEL

UART_NUM_1_RXD_DIRECT_GPIO_NUM

UART_GPIO6_DIRECT_CHANNEL

UART_NUM_1_CTS_DIRECT_GPIO_NUM

UART_GPIO11_DIRECT_CHANNEL

UART_NUM_1_RTS_DIRECT_GPIO_NUM

UART_TXD_GPIO10_DIRECT_CHANNEL

UART_RXD_GPIO9_DIRECT_CHANNEL

UART_CTS_GPIO6_DIRECT_CHANNEL

UART_RTS_GPIO11_DIRECT_CHANNEL

UART_GPIO17_DIRECT_CHANNEL

UART_NUM_2_TXD_DIRECT_GPIO_NUM

UART_GPIO16_DIRECT_CHANNEL

UART_NUM_2_RXD_DIRECT_GPIO_NUM

UART_GPIO8_DIRECT_CHANNEL

UART_NUM_2_CTS_DIRECT_GPIO_NUM

UART_GPIO7_DIRECT_CHANNEL

UART_NUM_2_RTS_DIRECT_GPIO_NUM

UART_TXD_GPIO17_DIRECT_CHANNEL

UART_RXD_GPIO16_DIRECT_CHANNEL

UART_CTS_GPIO8_DIRECT_CHANNEL

UART_RTS_GPIO7_DIRECT_CHANNEL

本部分的 API 示例代码存放在 ESP-IDF 示例项目的 peripherals 目录下。

2.4 应用层协议

[English]

2.4.1 mDNS 服务

[English]

概述

mDNS 是一种组播 UDP 服务，用来提供本地网络服务和主机发现。

绝大多数的操作系统默认都会安装 mDNS 服务，或者提供单独的安装包。Mac OS 默认会安装名为 Bonjour 的服务（该服务基于 mDNS），此外 Apple 还发布了适用于 Windows 系统的安装程序，可以在 官方支持 找到。在 Linux 上，mDNS 服务由 avahi 提供，通常也会被默认安装。

mDNS 属性

- hostname：设备会去响应的主机名，如果没有设置，会根据设备的网络接口名定义 hostname 。例如，my-esp32 会被解析为 my-esp32.local。
- default_instance：默认实例名（即易记的设备名），例如 Jhon's ESP32 Thing。如果没有设置，将会使用 hostname。

以下为 STA 接口启动 mDNS 服务并设置 hostname 和 default_instance 的示例方法：

```
void start_mdns_service()
{
    // 初始化 mDNS 服务
    esp_err_t err = mdns_init();
    if (err) {
        printf("MDNS Init failed: %d\n", err);
        return;
    }

    // 设置 hostname
    mdns_hostname_set("my-esp32");
```

(下页继续)

(续上页)

```
    // 设置默认实例
    mdns_instance_name_set("Jhon's ESP32 Thing");
}
```

mDNS 服务

mDNS 可以广播设备能够提供的网络服务的相关信息，每个服务会由以下属性构成。

- `instance_name`：实例名（即易记的服务名），例如 `Jhon's ESP32 Web Server`。如果没有定义，会使用 `default_instance`。
- `service_type`：（必需）服务类型，以下划线为前缀，这里 列出了常见的类型。
- `proto`：（必需）服务运行所依赖的协议，以下划线为前缀，例如 `_tcp` 或者 `_udp`。
- `port`：（必需）服务运行所用的端口号。
- `txt`：形如 `{var, val}` 的字符串数组，用于定义服务的属性。

添加一些服务和不同属性的示例方法:

```
void add_mdns_services()
{
    // 添加服务
    mdns_service_add(NULL, "_http", "_tcp", 80, NULL, 0);
    mdns_service_add(NULL, "_arduino", "_tcp", 3232, NULL, 0);
    mdns_service_add(NULL, "_myservice", "_udp", 1234, NULL, 0);

    // 注意：必须先添加服务，然后才能设置其属性
    // web 服务器使用自定义的实例名
    mdns_service_instance_name_set("_http", "_tcp", "Jhon's ESP32 Web Server");

    mdns_txt_item_t serviceTxtData[3] = {
        {"board","esp32"},
        {"u","user"},
        {"p","password"}
    };
    // 设置服务的文本数据（会释放并替换当前数据）
    mdns_service_txt_set("_http", "_tcp", serviceTxtData, 3);

    // 修改服务端口号
    mdns_service_port_set("_myservice", "_udp", 4321);
}
```

mDNS 查询

mDNS 提供查询服务和解析主机 IP/IPv6 地址的方法。

服务查询的结果会作为 `mdns_result_t` 类型对象的链表返回。

解析主机 IP 地址的示例方法:

```
void resolve_mdns_host(const char * host_name)
{
    printf("Query A: %s.local", host_name);

    struct ip4_addr addr;
    addr.addr = 0;

    esp_err_t err = mdns_query_a(host_name, 2000,  &addr);
    if(err){
        if(err == ESP_ERR_NOT_FOUND){
            printf("Host was not found!");
            return;
        }
        printf("Query Failed");
        return;
    }

    printf(IPSTR, IP2STR(&addr));
}
```

解析本地服务的示例方法:

```
static const char * if_str[] = {"STA", "AP", "ETH", "MAX"};
static const char * ip_protocol_str[] = {"V4", "V6", "MAX"};

void mdns_print_results(mdns_result_t * results){
    mdns_result_t * r = results;
    mdns_ip_addr_t * a = NULL;
    int i = 1, t;
    while(r){
        printf("%d: Interface: %s, Type: %s\n", i++, if_str[r->tcpip_if], ip_protocol_
↪str[r->ip_protocol]);
        if(r->instance_name){
            printf("  PTR : %s\n", r->instance_name);
        }
```

(下页继续)

(续上页)

```
        if(r->hostname){
            printf("  SRV : %s.local:%u\n", r->hostname, r->port);
        }
        if(r->txt_count){
            printf("  TXT : [%u] ", r->txt_count);
            for(t=0; t<r->txt_count; t++){
                printf("%s=%s; ", r->txt[t].key, r->txt[t].value);
            }
            printf("\n");
        }
        a = r->addr;
        while(a){
            if(a->addr.type == IPADDR_TYPE_V6){
                printf("  AAAA: " IPV6STR "\n", IPV62STR(a->addr.u_addr.ip6));
            } else {
                printf("  A   : " IPSTR "\n", IP2STR(&(a->addr.u_addr.ip4)));
            }
            a = a->next;
        }
        r = r->next;
    }

}

void find_mdns_service(const char * service_name, const char * proto)
{
    ESP_LOGI(TAG, "Query PTR: %s.%s.local", service_name, proto);

    mdns_result_t * results = NULL;
    esp_err_t err = mdns_query_ptr(service_name, proto, 3000, 20,  &results);
    if(err){
        ESP_LOGE(TAG, "Query Failed");
        return;
    }
    if(!results){
        ESP_LOGW(TAG, "No results found!");
        return;
    }

    mdns_print_results(results);
```

(下页继续)

(续上页)

```
    mdns_query_results_free(results);
}
```

使用上述方法的示例:

```
void my_app_some_method(){
    // 搜索 esp32-mdns.local
    resolve_mdns_host("esp32-mdns");

    // 搜索 HTTP 服务器
    find_mdns_service("_http", "_tcp");
    // 或者搜索文件服务器
    find_mdns_service("_smb", "_tcp"); // Windows 系统的共享服务
    find_mdns_service("_afpovertcp", "_tcp"); // Apple AFP 文件共享服务
    find_mdns_service("_nfs", "_tcp"); // NFS 服务器
    find_mdns_service("_ftp", "_tcp"); // FTP 服务器
    // 或者网络打印机
    find_mdns_service("_printer", "_tcp");
    find_mdns_service("_ipp", "_tcp");
}
```

应用示例

有关 mDNS 服务器和查询器的应用示例请参考 protocols/mdns。

API 参考

Header File

- mdns/include/mdns.h

Functions

esp_err_t **mdns_init**(void)

Initialize mDNS on given interface.

Return

- ESP_OK on success
- ESP_ERR_INVALID_STATE when failed to register event handler

- ESP_ERR_NO_MEM on memory error
- ESP_FAIL when failed to start mdns task

void **mdns_free**(void)

Stop and free mDNS server.

esp_err_t **mdns_hostname_set**(**const** char **hostname*)

Set the hostname for mDNS server required if you want to advertise services.

Return

- ESP_OK success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_ERR_NO_MEM memory error

Parameters

- `hostname`: Hostname to set

esp_err_t **mdns_instance_name_set**(**const** char **instance_name*)

Set the default instance name for mDNS server.

Return

- ESP_OK success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_ERR_NO_MEM memory error

Parameters

- `instance_name`: Instance name to set

esp_err_t **mdns_service_add**(**const** char **instance_name*, **const** char **service_type*, **const** char **proto*, uint16_t *port*, *mdns_txt_item_t txt*[], size_t *num_items*)

Add service to mDNS server.

Return

- ESP_OK success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_ERR_NO_MEM memory error
- ESP_FAIL failed to add service

Parameters

- `instance_name`: instance name to set. If NULL, global instance name or hostname will be used
- `service_type`: service type (_http, _ftp, etc)
- `proto`: service protocol (_tcp, _udp)
- `port`: service port
- `txt`: string array of TXT data (eg. {{ "var" ," val" },{ "other" ," 2" }})
- `num_items`: number of items in TXT data

esp_err_t **mdns_service_remove(const** char ***service_type**, **const** char ***proto**)

Remove service from mDNS server.

Return

- ESP_OK success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_ERR_NOT_FOUND Service not found
- ESP_ERR_NO_MEM memory error

Parameters

- `service_type`: service type (_http, _ftp, etc)
- `proto`: service protocol (_tcp, _udp)

esp_err_t **mdns_service_instance_name_set(const** char ***service_type**, **const** char ***proto**, **const** char ***instance_name**)

Set instance name for service.

Return

- ESP_OK success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_ERR_NOT_FOUND Service not found
- ESP_ERR_NO_MEM memory error

Parameters

- `service_type`: service type (_http, _ftp, etc)
- `proto`: service protocol (_tcp, _udp)
- `instance_name`: instance name to set

esp_err_t **mdns_service_port_set**(**const** char *_service_type_, **const** char *_proto_, uint16_t _port_)

Set service port.

Return

- ESP_OK success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_ERR_NOT_FOUND Service not found
- ESP_ERR_NO_MEM memory error

Parameters

- `service_type`: service type (_http, _ftp, etc)
- `proto`: service protocol (_tcp, _udp)
- `port`: service port

esp_err_t **mdns_service_txt_set**(**const** char *_service_type_, **const** char *_proto_, *mdns_txt_item_t* _txt_[], uint8_t _num_items_)

Replace all TXT items for service.

Return

- ESP_OK success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_ERR_NOT_FOUND Service not found
- ESP_ERR_NO_MEM memory error

Parameters

- `service_type`: service type (_http, _ftp, etc)
- `proto`: service protocol (_tcp, _udp)
- `txt`: array of TXT data (eg. {{ "var" ," val" },{ "other" ," 2" }})
- `num_items`: number of items in TXT data

esp_err_t **mdns_service_txt_item_set**(**const** char *_service_type_, **const** char *_proto_, **const** char *_key_, **const** char *_value_)

Set/Add TXT item for service TXT record.

Return

- ESP_OK success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_ERR_NOT_FOUND Service not found

- ESP_ERR_NO_MEM memory error

Parameters

- `service_type`: service type (_http, _ftp, etc)
- `proto`: service protocol (_tcp, _udp)
- `key`: the key that you want to add/update
- `value`: the new value of the key

esp_err_t **mdns_service_txt_item_remove**(**const** char **service_type*, **const** char **proto*, **const** char **key*)

Remove TXT item for service TXT record.

Return

- ESP_OK success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_ERR_NOT_FOUND Service not found
- ESP_ERR_NO_MEM memory error

Parameters

- `service_type`: service type (_http, _ftp, etc)
- `proto`: service protocol (_tcp, _udp)
- `key`: the key that you want to remove

esp_err_t **mdns_service_remove_all**(void)

Remove and free all services from mDNS server.

Return

- ESP_OK success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t **mdns_query**(**const** char **name*, **const** char **service_type*, **const** char **proto*, uint16_t *type*, uint32_t *timeout*, size_t *max_results*, *mdns_result_t* ***results*)

Query mDNS for host or service All following query methods are derived from this one.

Return

- ESP_OK success
- ESP_ERR_INVALID_STATE mDNS is not running
- ESP_ERR_NO_MEM memory error

- ESP_ERR_INVALID_ARG timeout was not given

Parameters

- `name`: service instance or host name (NULL for PTR queries)
- `service_type`: service type (_http, _arduino, _ftp etc.) (NULL for host queries)
- `proto`: service protocol (_tcp, _udp, etc.) (NULL for host queries)
- `type`: type of query (MDNS_TYPE_*)
- `timeout`: time in milliseconds to wait for answers.
- `max_results`: maximum results to be collected
- `results`: pointer to the results of the query results must be freed using mdns_query_results_free below

void **mdns_query_results_free**(*mdns_result_t* **results*)

Free query results.

Parameters

- `results`: linked list of results to be freed

esp_err_t **mdns_query_ptr**(**const** char **service_type*, **const** char **proto*, uint32_t *timeout*, size_t *max_results*, *mdns_result_t* ***results*)

Query mDNS for service.

Return

- ESP_OK success
- ESP_ERR_INVALID_STATE mDNS is not running
- ESP_ERR_NO_MEM memory error
- ESP_ERR_INVALID_ARG parameter error

Parameters

- `service_type`: service type (_http, _arduino, _ftp etc.)
- `proto`: service protocol (_tcp, _udp, etc.)
- `timeout`: time in milliseconds to wait for answer.
- `max_results`: maximum results to be collected
- `results`: pointer to the results of the query

esp_err_t **mdns_query_srv**(**const** char **instance_name*, **const** char **service_type*, **const** char **proto*, uint32_t *timeout*, *mdns_result_t* ***result*)

Query mDNS for SRV record.

Return

- ESP_OK success
- ESP_ERR_INVALID_STATE mDNS is not running
- ESP_ERR_NO_MEM memory error
- ESP_ERR_INVALID_ARG parameter error

Parameters

- `instance_name`: service instance name
- `service_type`: service type (_http, _arduino, _ftp etc.)
- `proto`: service protocol (_tcp, _udp, etc.)
- `timeout`: time in milliseconds to wait for answer.
- `result`: pointer to the result of the query

esp_err_t **mdns_query_txt(const** char ***instance_name*, **const** char ***service_type*, **const** char ***proto*, uint32_t *timeout*, *mdns_result_t* ***result*)

Query mDNS for TXT record.

Return

- ESP_OK success
- ESP_ERR_INVALID_STATE mDNS is not running
- ESP_ERR_NO_MEM memory error
- ESP_ERR_INVALID_ARG parameter error

Parameters

- `instance_name`: service instance name
- `service_type`: service type (_http, _arduino, _ftp etc.)
- `proto`: service protocol (_tcp, _udp, etc.)
- `timeout`: time in milliseconds to wait for answer.
- `result`: pointer to the result of the query

esp_err_t **mdns_query_a(const** char **host_name*, uint32_t *timeout*, esp_ip4_addr_t **addr*)

Query mDNS for A record.

Return

- ESP_OK success
- ESP_ERR_INVALID_STATE mDNS is not running

- ESP_ERR_NO_MEM memory error
- ESP_ERR_INVALID_ARG parameter error

Parameters

- `host_name`: host name to look for
- `timeout`: time in milliseconds to wait for answer.
- `addr`: pointer to the resulting IP4 address

esp_err_t **mdns_query_aaaa**(**const** char **host_name*, uint32_t *timeout*, esp_ip6_addr_t **addr*)

Query mDNS for A record.

Return

- ESP_OK success
- ESP_ERR_INVALID_STATE mDNS is not running
- ESP_ERR_NO_MEM memory error
- ESP_ERR_INVALID_ARG parameter error

Parameters

- `host_name`: host name to look for
- `timeout`: time in milliseconds to wait for answer. If 0, max_results needs to be defined
- `addr`: pointer to the resulting IP6 address

esp_err_t **mdns_handle_system_event**(void **ctx*, *system_event_t* **event*)

System event handler This method controls the service state on all active interfaces and applications are required to call it from the system event handler for normal operation of mDNS service.

Parameters

- `ctx`: The system event context
- `event`: The system event

Structures

struct mdns_txt_item_t

mDNS basic text item structure Used in mdns_service_add()

Public Members

const char ***key**

item key name

const char ***value**

item value string

struct mdns_ip_addr_s

mDNS query linked list IP item

Public Members

esp_ip_addr_t **addr**

IP address

struct *mdns_ip_addr_s* ***next**

next IP, or NULL for the last IP in the list

struct mdns_result_s

mDNS query result structure

Public Members

struct *mdns_result_s* ***next**

next result, or NULL for the last result in the list

mdns_if_t **tcpip_if**

interface index

mdns_ip_protocol_t **ip_protocol**

ip_protocol type of the interface (v4/v6)

char ***instance_name**

instance name

char ***hostname**

hostname

uint16_t **port**

service port

mdns_txt_item_t ***txt**

txt record

size_t **txt_count**

number of txt items

mdns_ip_addr_t ***addr**
linked list of IP addresses found

Macros

MDNS_TYPE_A

MDNS_TYPE_PTR

MDNS_TYPE_TXT

MDNS_TYPE_AAAA

MDNS_TYPE_SRV

MDNS_TYPE_OPT

MDNS_TYPE_NSEC

MDNS_TYPE_ANY

Type Definitions

typedef struct *mdns_ip_addr_s* **mdns_ip_addr_t**
mDNS query linked list IP item

typedef enum *mdns_if_internal* **mdns_if_t**

typedef struct *mdns_result_s* **mdns_result_t**
mDNS query result structure

Enumerations

enum mdns_ip_protocol_t
mDNS enum to specify the ip_protocol type

Values:

MDNS_IP_PROTOCOL_V4

MDNS_IP_PROTOCOL_V6

MDNS_IP_PROTOCOL_MAX

enum mdns_if_internal
Values:

MDNS_IF_STA = 0

MDNS_IF_AP = 1

`MDNS_IF_ETH` = 2

`MDNS_IF_MAX`

2.4.2 ESP-TLS

Overview

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

- *esp_tls_conn_new()*: for opening a new TLS connection.
- *esp_tls_conn_read()*: for reading from the connection.
- *esp_tls_conn_write()*: for writing into the connection.
- *esp_tls_conn_delete()*: for freeing up the connection.

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

Application Example

Simple HTTPS example that uses ESP-TLS to establish a secure socket connection: protocols/https_request.

Tree structure for ESP-TLS component

```
esp_tls.c
esp_tls.h
esp_tls_mbedtls.c
esp_tls_wolfssl.c
private_include
    esp_tls_mbedtls.h
    esp_tls_wolfssl.h
```

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

Underlying SSL/TLS Library Options

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

注解: *As the library options are internal to ESP-TLS, switching the libries will not change ESP-TLS specific code for a project.*

How to use wolfssl with ESP-IDF

There are two ways to use wolfssl in your project

1) Directly add wolfssl as a component in your project with following three commands.:

```
(First change directory (cd) to your project directory)
mkdir components
cd components
git clone https://github.com/espressif/esp-wolfssl.git
```

2) Add wolfssl as an extra component in your project.

- Download wolfssl with:

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

- Include esp-wolfssl in ESP-IDF with setting EXTRA_COMPONENT_DIRS in CMakeLists.txt/Makefile of your project as done in wolfssl/examples. For reference see Optional Project variables in *build-system.*

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

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

Comparison between mbedtls and wolfssl

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

Property	Wolfssl	Mbedtls
Total Heap Consumed	~19 Kb	~37 Kb
Task Stack Used	~2.2 Kb	~3.6 Kb
Bin size	~858 Kb	~736 Kb

注解: *These values are subject to change with change in configuration options and version of respective libraries.*

API Reference

Header File

- esp-tls/esp_tls.h

Functions

esp_tls_t ***esp_tls_init**(void)

Create TLS connection.

This function allocates and initializes esp-tls structure handle.

Return tls Pointer to esp-tls as esp-tls handle if successfully initialized, NULL if allocation error

esp_tls_t ***esp_tls_conn_new**(**const** char **hostname*, int *hostlen*, int *port*, **const** *esp_tls_cfg_t* **cfg*)

Create a new blocking TLS/SSL connection.

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

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

Return pointer to esp_tls_t, or NULL if connection couldn't be opened.

Parameters

- `hostname`: Hostname of the host.
- `hostlen`: Length of hostname.
- `port`: Port number of the host.
- `cfg`: TLS configuration as esp_tls_cfg_t. If you wish to open non-TLS connection, keep this NULL. For TLS connection, a pass pointer to esp_tls_cfg_t. At a minimum, this structure should be zero-initialized.

int **esp_tls_conn_new_sync**(**const** char *_hostname_, int _hostlen_, int _port_, **const** _esp_tls_cfg_t_ *_cfg_, _esp_tls_t_ *_tls_)

Create a new blocking TLS/SSL connection.

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

Return

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

Parameters

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

_esp_tls_t_ ***esp_tls_conn_http_new**(**const** char *_url_, **const** _esp_tls_cfg_t_ *_cfg_)

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

The behaviour is same as esp_tls_conn_new() API. However this API accepts host's url.

Return pointer to esp_tls_t, or NULL if connection couldn't be opened.

Parameters

- `url`: url of host.
- `cfg`: TLS configuration as esp_tls_cfg_t. If you wish to open non-TLS connection, keep this NULL. For TLS connection, a pass pointer to 'esp_tls_cfg_t'. At a minimum, this structure should be zero-initialized.

int **esp_tls_conn_new_async**(**const** char *_hostname_, int _hostlen_, int _port_, **const** _esp_tls_cfg_t_ *_cfg_, _esp_tls_t_ *_tls_)

Create a new non-blocking TLS/SSL connection.

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

Return

- -1 If connection establishment fails.

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

Parameters

- `hostname`: Hostname of the host.
- `hostlen`: Length of hostname.
- `port`: Port number of the host.
- `cfg`: TLS configuration as esp_tls_cfg_t. `non_block` member of this structure should be set to be true.
- `tls`: pointer to esp-tls as esp-tls handle.

int **esp_tls_conn_http_new_async**(**const** char *_url_, **const** _esp_tls_cfg_t_ *_cfg_, _esp_tls_t_ *_tls_)

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

The behaviour is same as esp_tls_conn_new() API. However this API accepts host's url.

Return

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

Parameters

- `url`: url of host.
- `cfg`: TLS configuration as esp_tls_cfg_t.
- `tls`: pointer to esp-tls as esp-tls handle.

static ssize_t **esp_tls_conn_write**(_esp_tls_t_ *_tls_, **const** void *_data_, size_t _datalen_)

Write from buffer 'data' into specified tls connection.

Return

- >0 if write operation was successful, the return value is the number of bytes actually written to the TLS/SSL connection.
- 0 if write operation was not successful. The underlying connection was closed.
- <0 if write operation was not successful, because either an error occured or an action must be taken by the calling process.

Parameters

- `tls`: pointer to esp-tls as esp-tls handle.

- `data`: Buffer from which data will be written.
- `datalen`: Length of data buffer.

`static` ssize_t **esp_tls_conn_read**(*esp_tls_t* *tls, void *data, size_t datalen)

Read from specified tls connection into the buffer 'data'.

Return

- >0 if read operation was successful, the return value is the number of bytes actually read from the TLS/SSL connection.
- 0 if read operation was not successful. The underlying connection was closed.
- <0 if read operation was not successful, because either an error occured or an action must be taken by the calling process.

Parameters

- `tls`: pointer to esp-tls as esp-tls handle.
- `data`: Buffer to hold read data.
- `datalen`: Length of data buffer.

void **esp_tls_conn_delete**(*esp_tls_t* *tls)

Close the TLS/SSL connection and free any allocated resources.

This function should be called to close each tls connection opened with esp_tls_conn_new() or esp_tls_conn_http_new() APIs.

Parameters

- `tls`: pointer to esp-tls as esp-tls handle.

ssize_t **esp_tls_get_bytes_avail**(*esp_tls_t* *tls)

Return the number of application data bytes remaining to be read from the current record.

This API is a wrapper over mbedtls's mbedtls_ssl_get_bytes_avail() API.

Return

- -1 in case of invalid arg
- bytes available in the application data record read buffer

Parameters

- `tls`: pointer to esp-tls as esp-tls handle.

esp_err_t **esp_tls_init_global_ca_store**(void)

Create a global CA store, initially empty.

This function should be called if the application wants to use the same CA store for multiple connections. This function initialises the global CA store which can be then set by calling esp_tls_set_global_ca_store(). To be effective, this function must be called before any call to esp_tls_set_global_ca_store().

Return

- ESP_OK if creating global CA store was successful.
- ESP_ERR_NO_MEM if an error occured when allocating the mbedTLS resources.

esp_err_t **esp_tls_set_global_ca_store**(**const** unsigned char **cacert_pem_buf*, **const** unsigned int *cacert_pem_bytes*)

Set the global CA store with the buffer provided in pem format.

This function should be called if the application wants to set the global CA store for multiple connections i.e. to add the certificates in the provided buffer to the certificate chain. This function implicitly calls esp_tls_init_global_ca_store() if it has not already been called. The application must call this function before calling esp_tls_conn_new().

Return

- ESP_OK if adding certificates was successful.
- Other if an error occured or an action must be taken by the calling process.

Parameters

- `cacert_pem_buf`: Buffer which has certificates in pem format. This buffer is used for creating a global CA store, which can be used by other tls connections.
- `cacert_pem_bytes`: Length of the buffer.

void **esp_tls_free_global_ca_store**(void)

Free the global CA store currently being used.

The memory being used by the global CA store to store all the parsed certificates is freed up. The application can call this API if it no longer needs the global CA store.

esp_err_t **esp_tls_get_and_clear_last_error**(*esp_tls_error_handle_t h*, int **esp_tls_code*, int **esp_tls_flags*)

Returns last error in *esp_tls* with detailed mbedtls related error codes. The error information is cleared internally upon return.

Return

- ESP_ERR_INVALID_STATE if invalid parameters
- ESP_OK (0) if no error occurred
- specific error code (based on ESP_ERR_ESP_TLS_BASE) otherwise

Parameters

- `h`: esp-tls error handle.
- `esp_tls_code`: last error code returned from mbedtls api (set to zero if none) This pointer could be NULL if caller does not care about esp_tls_code
- `esp_tls_flags`: last certification verification flags (set to zero if none) This pointer could be NULL if caller does not care about esp_tls_code

Structures

struct esp_tls_last_error

Error structure containing relevant errors in case tls error occurred.

Public Members

esp_err_t **last_error**

error code (based on ESP_ERR_ESP_TLS_BASE) of the last occurred error

int **esp_tls_error_code**

esp_tls error code from last *esp_tls* failed api

int **esp_tls_flags**

last certification verification flags

struct psk_key_hint

ESP-TLS preshared key and hint structure.

Public Members

const uint8_t ***key**

key in PSK authentication mode in binary format

const size_t **key_size**

length of the key

const char ***hint**

hint in PSK authentication mode in string format

struct esp_tls_cfg

ESP-TLS configuration parameters.

Note Note about format of certificates:

- This structure includes certificates of a Certificate Authority, of client or server as well as private keys, which may be of PEM or DER format. In case of PEM format, the buffer must be NULL terminated (with NULL character included in certificate size).

- Certificate Authority’ s certificate may be a chain of certificates in case of PEM format, but could be only one certificate in case of DER format
- Variables names of certificates and private key buffers and sizes are defined as unions providing backward compatibility for legacy *_pem_buf and *_pem_bytes names which suggested only PEM format was supported. It is encouraged to use generic names such as cacert_buf and cacert_bytes.

Public Members

const char ****alpn_protos**

Application protocols required for HTTP2. If HTTP2/ALPN support is required, a list of protocols that should be negotiated. The format is length followed by protocol name. For the most common cases the following is ok: const char **alpn_protos = { “h2” , NULL };

- where ‘h2’ is the protocol name

const unsigned char ***cacert_buf**

Certificate Authority’ s certificate in a buffer. Format may be PEM or DER, depending on mbedtls-support This buffer should be NULL terminated in case of PEM

const unsigned char ***cacert_pem_buf**

CA certificate buffer legacy name

unsigned int **cacert_bytes**

Size of Certificate Authority certificate pointed to by cacert_buf (including NULL-terminator in case of PEM format)

unsigned int **cacert_pem_bytes**

Size of Certificate Authority certificate legacy name

const unsigned char ***clientcert_buf**

Client certificate in a buffer Format may be PEM or DER, depending on mbedtls-support This buffer should be NULL terminated in case of PEM

const unsigned char ***clientcert_pem_buf**

Client certificate legacy name

unsigned int **clientcert_bytes**

Size of client certificate pointed to by clientcert_pem_buf (including NULL-terminator in case of PEM format)

unsigned int **clientcert_pem_bytes**

Size of client certificate legacy name

const unsigned char ***clientkey_buf**

Client key in a buffer Format may be PEM or DER, depending on mbedtls-support This buffer should be NULL terminated in case of PEM

const unsigned char ***clientkey_pem_buf**

Client key legacy name

unsigned int **clientkey_bytes**

Size of client key pointed to by clientkey_pem_buf (including NULL-terminator in case of PEM format)

unsigned int **clientkey_pem_bytes**

Size of client key legacy name

const unsigned char ***clientkey_password**

Client key decryption password string

unsigned int **clientkey_password_len**

String length of the password pointed to by clientkey_password

bool **non_block**

Configure non-blocking mode. If set to true the underneath socket will be configured in non blocking mode after tls session is established

int **timeout_ms**

Network timeout in milliseconds

bool **use_global_ca_store**

Use a global ca_store for all the connections in which this bool is set.

const char ***common_name**

If non-NULL, server certificate CN must match this name. If NULL, server certificate CN must match hostname.

bool **skip_common_name**

Skip any validation of server certificate CN field

const *psk_hint_key_t* ***psk_hint_key**

Pointer to PSK hint and key. if not NULL (and certificates are NULL) then PSK authentication is enabled with configured setup. Important note: the pointer must be valid for connection

struct esp_tls

ESP-TLS Connection Handle.

Public Members

int **sockfd**

Underlying socket file descriptor.

ssize_t (***read**)(**struct** *esp_tls* *tls, char *data, size_t datalen)

Callback function for reading data from TLS/SSL connection.

ssize_t (***write**)(**struct** *esp_tls* *tls, **const** char *data, size_t datalen)

Callback function for writing data to TLS/SSL connection.

esp_tls_conn_state_t **conn_state**
ESP-TLS Connection state

fd_set **rset**
read file descriptors

fd_set **wset**
write file descriptors

bool **is_tls**
indicates connection type (TLS or NON-TLS)

esp_tls_role_t **role**
esp-tls role

- ESP_TLS_CLIENT
- ESP_TLS_SERVER

esp_tls_error_handle_t **error_handle**
handle to error descriptor

Macros

ESP_ERR_ESP_TLS_BASE
Starting number of ESP-TLS error codes

ESP_ERR_ESP_TLS_CANNOT_RESOLVE_HOSTNAME
Error if hostname couldn't be resolved upon tls connection

ESP_ERR_ESP_TLS_CANNOT_CREATE_SOCKET
Failed to create socket

ESP_ERR_ESP_TLS_UNSUPPORTED_PROTOCOL_FAMILY
Unsupported protocol family

ESP_ERR_ESP_TLS_FAILED_CONNECT_TO_HOST
Failed to connect to host

ESP_ERR_ESP_TLS_SOCKET_SETOPT_FAILED
failed to set socket option

ESP_ERR_MBEDTLS_CERT_PARTLY_OK
mbedtls parse certificates was partly successful

ESP_ERR_MBEDTLS_CTR_DRBG_SEED_FAILED
mbedtls api returned error

ESP_ERR_MBEDTLS_SSL_SET_HOSTNAME_FAILED
mbedtls api returned error

ESP_ERR_MBEDTLS_SSL_CONFIG_DEFAULTS_FAILED
: mbedtls api returned error

ESP_ERR_MBEDTLS_SSL_CONF_ALPN_PROTOCOLS_FAILED
: mbedtls api returned error

ESP_ERR_MBEDTLS_X509_CRT_PARSE_FAILED
: mbedtls api returned error

ESP_ERR_MBEDTLS_SSL_CONF_OWN_CERT_FAILED
: mbedtls api returned error

ESP_ERR_MBEDTLS_SSL_SETUP_FAILED
: mbedtls api returned error

ESP_ERR_MBEDTLS_SSL_WRITE_FAILED
: mbedtls api returned error

ESP_ERR_MBEDTLS_PK_PARSE_KEY_FAILED
: mbedtls api returned failed

ESP_ERR_MBEDTLS_SSL_HANDSHAKE_FAILED
: mbedtls api returned failed

ESP_ERR_MBEDTLS_SSL_CONF_PSK_FAILED
: mbedtls api returned failed

ESP_ERR_ESP_TLS_CONNECTION_TIMEOUT
: new connection in esp_tls_low_level_conn connection timeouted

Type Definitions

typedef struct *esp_tls_last_error* ***esp_tls_error_handle_t**

typedef struct *esp_tls_last_error* **esp_tls_last_error_t**
: Error structure containing relevant errors in case tls error occurred.

typedef enum *esp_tls_conn_state* **esp_tls_conn_state_t**
: ESP-TLS Connection State.

typedef enum *esp_tls_role* **esp_tls_role_t**

typedef struct *psk_key_hint* **psk_hint_key_t**
: ESP-TLS preshared key and hint structure.

typedef struct *esp_tls_cfg* **esp_tls_cfg_t**
: ESP-TLS configuration parameters.

  Note Note about format of certificates:

- This structure includes certificates of a Certificate Authority, of client or server as well as private keys, which may be of PEM or DER format. In case of PEM format, the buffer must be NULL terminated (with NULL character included in certificate size).
- Certificate Authority' s certificate may be a chain of certificates in case of PEM format, but could be only one certificate in case of DER format
- Variables names of certificates and private key buffers and sizes are defined as unions providing backward compatibility for legacy *__pem__buf and *__pem__bytes names which suggested only PEM format was supported. It is encouraged to use generic names such as cacert__buf and cacert__bytes.

typedef struct *esp_tls* **esp_tls_t**

ESP-TLS Connection Handle.

Enumerations

enum esp_tls_conn_state

ESP-TLS Connection State.

Values:

ESP_TLS_INIT = 0

ESP_TLS_CONNECTING

ESP_TLS_HANDSHAKE

ESP_TLS_FAIL

ESP_TLS_DONE

enum esp_tls_role

Values:

ESP_TLS_CLIENT = 0

ESP_TLS_SERVER

2.4.3 ESP HTTP Client

Overview

`esp_http_client` provides an API for making HTTP/S requests from ESP-IDF programs. The steps to use this API for an HTTP request are:

- *`esp_http_client_init()`*: To use the HTTP client, the first thing we must do is create an `esp_http_client` by pass into this function with the *`esp_http_client_config_t`* configurations. Which configuration values we do not define, the library will use default.

- *esp_http_client_perform()*: The esp_http_client argument created from the init function is needed. This function performs all operations of the esp_http_client, from opening the connection, sending data, downloading data and closing the connection if necessary. All related events will be invoked in the event_handle (defined by *esp_http_client_config_t*). This function performs its job and blocks the current task until it' s done
- *esp_http_client_cleanup()*: After completing our **esp_http_client's** task, this is the last function to be called. It will close the connection (if any) and free up all the memory allocated to the HTTP client

Application Example

```
esp_err_t _http_event_handle(esp_http_client_event_t *evt)
{
    switch(evt->event_id) {
        case HTTP_EVENT_ERROR:
            ESP_LOGI(TAG, "HTTP_EVENT_ERROR");
            break;
        case HTTP_EVENT_ON_CONNECTED:
            ESP_LOGI(TAG, "HTTP_EVENT_ON_CONNECTED");
            break;
        case HTTP_EVENT_HEADER_SENT:
            ESP_LOGI(TAG, "HTTP_EVENT_HEADER_SENT");
            break;
        case HTTP_EVENT_ON_HEADER:
            ESP_LOGI(TAG, "HTTP_EVENT_ON_HEADER");
            printf("%.*s", evt->data_len, (char*)evt->data);
            break;
        case HTTP_EVENT_ON_DATA:
            ESP_LOGI(TAG, "HTTP_EVENT_ON_DATA, len=%d", evt->data_len);
            if (!esp_http_client_is_chunked_response(evt->client)) {
                printf("%.*s", evt->data_len, (char*)evt->data);
            }

            break;
        case HTTP_EVENT_ON_FINISH:
            ESP_LOGI(TAG, "HTTP_EVENT_ON_FINISH");
            break;
        case HTTP_EVENT_DISCONNECTED:
            ESP_LOGI(TAG, "HTTP_EVENT_DISCONNECTED");
            break;
```

(下页继续)

(续上页)

```
    }
    return ESP_OK;
}

esp_http_client_config_t config = {
   .url = "http://httpbin.org/redirect/2",
   .event_handler = _http_event_handle,
};
esp_http_client_handle_t client = esp_http_client_init(&config);
esp_err_t err = esp_http_client_perform(client);

if (err == ESP_OK) {
   ESP_LOGI(TAG, "Status = %d, content_length = %d",
            esp_http_client_get_status_code(client),
            esp_http_client_get_content_length(client));
}
esp_http_client_cleanup(client);
```

Persistent Connections

Persistent connections means that the HTTP client can re-use the same connection for several transfers. If the server does not request to close the connection with the `Connection: close` header, the new transfer with sample ip address, port, and protocol.

To allow the HTTP client to take full advantage of persistent connections, you should do as many of your file transfers as possible using the same handle.

Persistent Connections example

```
esp_err_t err;
esp_http_client_config_t config = {
    .url = "http://httpbin.org/get",
};
esp_http_client_handle_t client = esp_http_client_init(&config);
// first request
err = esp_http_client_perform(client);

// second request
esp_http_client_set_url(client, "http://httpbin.org/anything")
```

(下页继续)

(续上页)

```
esp_http_client_set_method(client, HTTP_METHOD_DELETE);
esp_http_client_set_header(client, "HeaderKey", "HeaderValue");
err = esp_http_client_perform(client);

esp_http_client_cleanup(client);
```

HTTPS

The HTTP client supports SSL connections using **mbedtls**, with the **url** configuration starting with `https` scheme (or `transport_type = HTTP_TRANSPORT_OVER_SSL`). HTTPS support can be configured via :ref:CONFIG_ENABLE_HTTPS (enabled by default)..

注解: By providing information using HTTPS, the library will use the SSL transport type to connect to the server. If you want to verify server, then need to provide additional certificate in PEM format, and provide to `cert_pem` in `esp_http_client_config_t`

HTTPS example

```
static void https()
{
    esp_http_client_config_t config = {
        .url = "https://www.howsmyssl.com",
        .cert_pem = howsmyssl_com_root_cert_pem_start,
    };
    esp_http_client_handle_t client = esp_http_client_init(&config);
    esp_err_t err = esp_http_client_perform(client);

    if (err == ESP_OK) {
        ESP_LOGI(TAG, "Status = %d, content_length = %d",
                esp_http_client_get_status_code(client),
                esp_http_client_get_content_length(client));
    }
    esp_http_client_cleanup(client);
}
```

HTTP Stream

Some applications need to open the connection and control the reading of the data in an active manner. the HTTP client supports some functions to make this easier, of course, once you use these functions you should not use the *esp_http_client_perform()* function with that handle, and *esp_http_client_init()* alway to called first to get the handle. Perform that functions in the order below:

- *esp_http_client_init()*: to create and handle
- `esp_http_client_set_*` or `esp_http_client_delete_*`: to modify the http connection information (optional)
- *esp_http_client_open()*: Open the http connection with `write_len` parameter, `write_len=0` if we only need read
- *esp_http_client_write()*: Upload data, max length equal to `write_len` of *esp_http_client_open()* function. We may not need to call it if `write_len=0`
- *esp_http_client_fetch_headers()*: After sending the headers and write data (if any) to the server, this function will read the HTTP Server response headers. Calling this function will return the `content-length` from the Server, and we can call *esp_http_client_get_status_code()* for the HTTP status of the connection.
- *esp_http_client_read()*: Now, we can read the HTTP stream by this function.
- *esp_http_client_close()*: We should the connection after finish
- *esp_http_client_cleanup()*: And release the resources

Perform HTTP request as Stream reader

Check the example function `http_perform_as_stream_reader` at protocols/esp_http_client.

HTTP Authentication

The HTTP client supports both **Basic** and **Digest** Authentication. By providing usernames and passwords in `url` or in the `username`, `password` of config entry. And with `auth_type = HTTP_AUTH_TYPE_BASIC`, the HTTP client takes only 1 perform to pass the authentication process. If `auth_type = HTTP_AUTH_TYPE_NONE`, but there are `username` and `password` in the configuration, the HTTP client takes 2 performs. The first time it connects to the server and receives the UNAUTHORIZED header. Based on this information, it will know which authentication method to choose, and perform it on the second.

Config authentication example with URI

```
esp_http_client_config_t config = {
    .url = "http://user:passwd@httpbin.org/basic-auth/user/passwd",
    .auth_type = HTTP_AUTH_TYPE_BASIC,
};
```

Config authentication example with username, password entry

```
esp_http_client_config_t config = {
    .url = "http://httpbin.org/basic-auth/user/passwd",
    .username = "user",
    .password = "passwd",
    .auth_type = HTTP_AUTH_TYPE_BASIC,
};
```

HTTP Client example: protocols/esp_http_client.

API Reference

Header File

- esp_http_client/include/esp_http_client.h

Functions

esp_http_client_handle_t **esp_http_client_init**(**const** *esp_http_client_config_t* **config*)

Start a HTTP session This function must be the first function to call, and it returns a esp_http_client_handle_t that you must use as input to other functions in the interface. This call MUST have a corresponding call to esp_http_client_cleanup when the operation is complete.

Return

- `esp_http_client_handle_t`
- NULL if any errors

Parameters

- `config`: The configurations, see `http_client_config_t`

esp_err_t **esp_http_client_perform**(*esp_http_client_handle_t client*)

Invoke this function after `esp_http_client_init` and all the options calls are made, and will perform the transfer as described in the options. It must be called with the same esp_http_client_handle_t as input as the esp_http_client_init call returned. esp_http_client_perform performs the entire

request in either blocking or non-blocking manner. By default, the API performs request in a blocking manner and returns when done, or if it failed, and in non-blocking manner, it returns if EAGAIN/EWOULDBLOCK or EINPROGRESS is encountered, or if it failed. And in case of non-blocking request, the user may call this API multiple times unless request & response is complete or there is a failure. To enable non-blocking esp_http_client_perform(), `is_async` member of *esp_http_client_config_t* must be set while making a call to esp_http_client_init() API. You can do any amount of calls to esp_http_client_perform while using the same esp_http_client_handle_t. The underlying connection may be kept open if the server allows it. If you intend to transfer more than one file, you are even encouraged to do so. esp_http_client will then attempt to re-use the same connection for the following transfers, thus making the operations faster, less CPU intense and using less network resources. Just note that you will have to use `esp_http_client_set_**` between the invokes to set options for the following esp_http_client_perform.

Note You must never call this function simultaneously from two places using the same client handle. Let the function return first before invoking it another time. If you want parallel transfers, you must use several esp_http_client_handle_t. This function include `esp_http_client_open` -> `esp_http_client_write` -> `esp_http_client_fetch_headers` -> `esp_http_client_read` (and option) `esp_http_client_close`.

Return

- ESP_OK on successful
- ESP_FAIL on error

Parameters

- `client`: The esp_http_client handle

esp_err_t **esp_http_client_set_url**(*esp_http_client_handle_t* *client*, **const** char **url*)

Set URL for client, when performing this behavior, the options in the URL will replace the old ones.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `client`: The esp_http_client handle
- `url`: The url

esp_err_t **esp_http_client_set_post_field**(*esp_http_client_handle_t* *client*, **const** char **data*, int *len*)

Set post data, this function must be called before `esp_http_client_perform`. Note: The data parameter passed to this function is a pointer and this function will not copy the data.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `client`: The esp_http_client handle
- `data`: post data pointer
- `len`: post length

int **esp_http_client_get_post_field**(*esp_http_client_handle_t* *client*, char **data*)

Get current post field information.

Return Size of post data

Parameters

- `client`: The client
- `data`: Point to post data pointer

esp_err_t **esp_http_client_set_header**(*esp_http_client_handle_t* *client*, **const** char *key, **const** char *value)

Set http request header, this function must be called after esp_http_client_init and before any perform function.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `client`: The esp_http_client handle
- `key`: The header key
- `value`: The header value

esp_err_t **esp_http_client_get_header**(*esp_http_client_handle_t* *client*, **const** char *key, char **value)

Get http request header. The value parameter will be set to NULL if there is no header which is same as the key specified, otherwise the address of header value will be assigned to value parameter. This function must be called after `esp_http_client_init`.

Return

- ESP_OK

- ESP_FAIL

Parameters

- `client`: The esp_http_client handle
- `key`: The header key
- `value`: The header value

esp_err_t **esp_http_client_get_username**(*esp_http_client_handle_t client*, char **value*)

Get http request username. The address of username buffer will be assigned to value parameter. This function must be called after `esp_http_client_init`.

Return

- ESP_OK
- ESP_ERR_INVALID_ARG

Parameters

- `client`: The esp_http_client handle
- `value`: The username value

esp_err_t **esp_http_client_set_username**(*esp_http_client_handle_t client*, **const** char **username*)

Set http request username. The value of username parameter will be assigned to username buffer. If the username parameter is NULL then username buffer will be freed.

Return

- ESP_OK
- ESP_ERR_INVALID_ARG

Parameters

- `client`: The esp_http_client handle
- `username`: The username value

esp_err_t **esp_http_client_get_password**(*esp_http_client_handle_t client*, char **value*)

Get http request password. The address of password buffer will be assigned to value parameter. This function must be called after `esp_http_client_init`.

Return

- ESP_OK
- ESP_ERR_INVALID_ARG

Parameters

- `client`: The esp_http_client handle
- `value`: The password value

esp_err_t **esp_http_client_set_password**(*esp_http_client_handle_t client*, char **password*)

Set http request password. The value of password parameter will be assigned to password buffer. If the password parameter is NULL then password buffer will be freed.

Return

- ESP_OK
- ESP_ERR_INVALID_ARG

Parameters

- `client`: The esp_http_client handle
- `password`: The password value

esp_err_t **esp_http_client_set_authtype**(*esp_http_client_handle_t client*, *esp_http_client_auth_type_t auth_type*)

Set http request auth_type.

Return

- ESP_OK
- ESP_ERR_INVALID_ARG

Parameters

- `client`: The esp_http_client handle
- `auth_type`: The esp_http_client auth type

esp_err_t **esp_http_client_set_method**(*esp_http_client_handle_t client*, *esp_http_client_method_t method*)

Set http request method.

Return

- ESP_OK
- ESP_ERR_INVALID_ARG

Parameters

- `client`: The esp_http_client handle
- `method`: The method

esp_err_t **esp_http_client_delete_header**(*esp_http_client_handle_t client*, **const** char **key*)

Delete http request header.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `client`: The esp_http_client handle
- `key`: The key

esp_err_t **esp_http_client_open**(*esp_http_client_handle_t client*, int *write_len*)

This function will be open the connection, write all header strings and return.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `client`: The esp_http_client handle
- `write_len`: HTTP Content length need to write to the server

int **esp_http_client_write**(*esp_http_client_handle_t client*, **const** char ***buffer*, int *len*)

This function will write data to the HTTP connection previously opened by esp_http_client_open()

Return

- (-1) if any errors
- Length of data written

Parameters

- `client`: The esp_http_client handle
- `buffer`: The buffer
- `len`: This value must not be larger than the write_len parameter provided to esp_http_client_open()

int **esp_http_client_fetch_headers**(*esp_http_client_handle_t client*)

This function need to call after esp_http_client_open, it will read from http stream, process all receive headers.

Return

- (0) if stream doesn' t contain content-length header, or chunked encoding (checked by `esp_http_client_is_chunked` response)

- (-1: ESP_FAIL) if any errors
- Download data length defined by content-length header

Parameters

- `client`: The esp_http_client handle

bool **esp_http_client_is_chunked_response**(*esp_http_client_handle_t client*)

Check response data is chunked.

Return true or false

Parameters

- `client`: The esp_http_client handle

int **esp_http_client_read**(*esp_http_client_handle_t client*, char **buffer*, int *len*)

Read data from http stream.

Return

- (-1) if any errors
- Length of data was read

Parameters

- `client`: The esp_http_client handle
- `buffer`: The buffer
- `len`: The length

int **esp_http_client_get_status_code**(*esp_http_client_handle_t client*)

Get http response status code, the valid value if this function invoke after `esp_http_client_perform`

Return Status code

Parameters

- `client`: The esp_http_client handle

int **esp_http_client_get_content_length**(*esp_http_client_handle_t client*)

Get http response content length (from header Content-Length) the valid value if this function invoke after `esp_http_client_perform`

Return

- (-1) Chunked transfer
- Content-Length value as bytes

Parameters

- `client`: The esp_http_client handle

esp_err_t **esp_http_client_close**(*esp_http_client_handle_t client*)

Close http connection, still kept all http request resources.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `client`: The esp_http_client handle

esp_err_t **esp_http_client_cleanup**(*esp_http_client_handle_t client*)

This function must be the last function to call for an session. It is the opposite of the esp_http_client_init function and must be called with the same handle as input that a esp_http_client_init call returned. This might close all connections this handle has used and possibly has kept open until now. Don't call this function if you intend to transfer more files, re-using handles is a key to good performance with esp_http_client.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `client`: The esp_http_client handle

esp_http_client_transport_t **esp_http_client_get_transport_type**(*esp_http_client_handle_t client*)

Get transport type.

Return

- HTTP_TRANSPORT_UNKNOWN
- HTTP_TRANSPORT_OVER_TCP
- HTTP_TRANSPORT_OVER_SSL

Parameters

- `client`: The esp_http_client handle

esp_err_t **esp_http_client_set_redirection**(*esp_http_client_handle_t client*)

Set redirection URL. When received the 30x code from the server, the client stores the redirect URL

provided by the server. This function will set the current URL to redirect to enable client to execute the redirection request.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `client`: The esp_http_client handle

void **esp_http_client_add_auth**(*esp_http_client_handle_t client*)

On receiving HTTP Status code 401, this API can be invoked to add authorization information.

Note There is a possibility of receiving body message with redirection status codes, thus make sure to flush off body data after calling this API.

Parameters

- `client`: The esp_http_client handle

bool **esp_http_client_is_complete_data_received**(*esp_http_client_handle_t client*)

Checks if entire data in the response has been read without any error.

Return

- true
- false

Parameters

- `client`: The esp_http_client handle

Structures

struct esp_http_client_event

HTTP Client events data.

Public Members

esp_http_client_event_id_t **event_id**

event_id, to know the cause of the event

esp_http_client_handle_t **client**

esp_http_client_handle_t context

void *data
: data of the event

int data_len
: data length of data

void *user_data
: user_data context, from *esp_http_client_config_t* user_data

char *header_key
: For HTTP_EVENT_ON_HEADER event_id, it's store current http header key

char *header_value
: For HTTP_EVENT_ON_HEADER event_id, it's store current http header value

struct esp_http_client_config_t
: HTTP configuration.

Public Members

const char *url
: HTTP URL, the information on the URL is most important, it overrides the other fields below, if any

const char *host
: Domain or IP as string

int port
: Port to connect, default depend on esp_http_client_transport_t (80 or 443)

const char *username
: Using for Http authentication

const char *password
: Using for Http authentication

esp_http_client_auth_type_t auth_type
: Http authentication type, see esp_http_client_auth_type_t

const char *path
: HTTP Path, if not set, default is /

const char *query
: HTTP query

const char *cert_pem
: SSL server certification, PEM format as string, if the client requires to verify server

const char *client_cert_pem
: SSL client certification, PEM format as string, if the server requires to verify client

const char ***client_key_pem**

SSL client key, PEM format as string, if the server requires to verify client

esp_http_client_method_t **method**

HTTP Method

int **timeout_ms**

Network timeout in milliseconds

bool **disable_auto_redirect**

Disable HTTP automatic redirects

int **max_redirection_count**

Max redirection number, using default value if zero

http_event_handle_cb **event_handler**

HTTP Event Handle

esp_http_client_transport_t **transport_type**

HTTP transport type, see **esp_http_client_transport_t**

int **buffer_size**

HTTP receive buffer size

int **buffer_size_tx**

HTTP transmit buffer size

void ***user_data**

HTTP user_data context

bool **is_async**

Set asynchronous mode, only supported with HTTPS for now

bool **use_global_ca_store**

Use a global ca_store for all the connections in which this bool is set.

bool **skip_cert_common_name_check**

Skip any validation of server certificate CN field

Macros

DEFAULT_HTTP_BUF_SIZE

ESP_ERR_HTTP_BASE

Starting number of HTTP error codes

ESP_ERR_HTTP_MAX_REDIRECT

The error exceeds the number of HTTP redirects

ESP_ERR_HTTP_CONNECT

Error open the HTTP connection

ESP_ERR_HTTP_WRITE_DATA
: Error write HTTP data

ESP_ERR_HTTP_FETCH_HEADER
: Error read HTTP header from server

ESP_ERR_HTTP_INVALID_TRANSPORT
: There are no transport support for the input scheme

ESP_ERR_HTTP_CONNECTING
: HTTP connection hasn’ t been established yet

ESP_ERR_HTTP_EAGAIN
: Mapping of errno EAGAIN to esp_err_t

Type Definitions

typedef struct esp_http_client *esp_http_client_handle_t

typedef struct *esp_http_client_event* *esp_http_client_event_handle_t

typedef struct *esp_http_client_event* esp_http_client_event_t
: HTTP Client events data.

typedef *esp_err_t* (*http_event_handle_cb)(*esp_http_client_event_t* *evt)

Enumerations

enum esp_http_client_event_id_t
: HTTP Client events id.

  Values:

  HTTP_EVENT_ERROR = 0
  : This event occurs when there are any errors during execution

  HTTP_EVENT_ON_CONNECTED
  : Once the HTTP has been connected to the server, no data exchange has been performed

  HTTP_EVENT_HEADERS_SENT
  : After sending all the headers to the server

  HTTP_EVENT_HEADER_SENT = HTTP_EVENT_HEADERS_SENT
  : This header has been kept for backward compatability and will be deprecated in future versions esp-idf

  HTTP_EVENT_ON_HEADER
  : Occurs when receiving each header sent from the server

HTTP_EVENT_ON_DATA
: Occurs when receiving data from the server, possibly multiple portions of the packet

HTTP_EVENT_ON_FINISH
: Occurs when finish a HTTP session

HTTP_EVENT_DISCONNECTED
: The connection has been disconnected

enum esp_http_client_transport_t
: HTTP Client transport.

Values:

HTTP_TRANSPORT_UNKNOWN = 0x0
: Unknown

HTTP_TRANSPORT_OVER_TCP
: Transport over tcp

HTTP_TRANSPORT_OVER_SSL
: Transport over ssl

enum esp_http_client_method_t
: HTTP method.

Values:

HTTP_METHOD_GET = 0
: HTTP GET Method

HTTP_METHOD_POST
: HTTP POST Method

HTTP_METHOD_PUT
: HTTP PUT Method

HTTP_METHOD_PATCH
: HTTP PATCH Method

HTTP_METHOD_DELETE
: HTTP DELETE Method

HTTP_METHOD_HEAD
: HTTP HEAD Method

HTTP_METHOD_NOTIFY
: HTTP NOTIFY Method

HTTP_METHOD_SUBSCRIBE
: HTTP SUBSCRIBE Method

`HTTP_METHOD_UNSUBSCRIBE`
HTTP UNSUBSCRIBE Method

`HTTP_METHOD_OPTIONS`
HTTP OPTIONS Method

`HTTP_METHOD_MAX`

`enum esp_http_client_auth_type_t`
HTTP Authentication type.

Values:

`HTTP_AUTH_TYPE_NONE` = 0
No authention

`HTTP_AUTH_TYPE_BASIC`
HTTP Basic authentication

`HTTP_AUTH_TYPE_DIGEST`
HTTP Disgest authentication

`enum HttpStatus_Code`
Enum for the HTTP status codes.

Values:

`HttpStatus_MovedPermanently` = 301

`HttpStatus_Found` = 302

`HttpStatus_TemporaryRedirect` = 307

`HttpStatus_Unauthorized` = 401

2.4.4 ESP WebSocket Client

Overview

The ESP WebSocket client is an implementation of WebSocket protocol client for ESP32

Features

- supports WebSocket over TCP, SSL with mbedtls
- Easy to setup with URI
- Multiple instances (Multiple clients in one application)

Configuration

URI

- Supports `ws`, `wss` schemes
- WebSocket samples:
  - `ws://websocket.org`: WebSocket over TCP, default port 80
  - `wss://websocket.org`: WebSocket over SSL, default port 443
- Minimal configurations:

```
const esp_websocket_client_config_t ws_cfg = {
    .uri = "ws://websocket.org",
};
```

- If there are any options related to the URI in `esp_websocket_client_config_t`, the option defined by the URI will be overridden. Sample:

```
const esp_websocket_client_config_t ws_cfg = {
    .uri = "ws://websocket.org:123",
    .port = 4567,
};
//WebSocket client will connect to websocket.org using port 4567
```

SSL

- Get certificate from server, example: `websocket.org openssl s_client -showcerts -connect websocket.org:443 </dev/null 2>/dev/null|openssl x509 -outform PEM >websocket_org.pem`
- Configuration:

```
const esp_websocket_client_config_t ws_cfg = {
    .uri = "wss://websocket.org",
    .cert_pem = (const char *)websocket_org_pem_start,
};
```

For more options on `esp_websocket_client_config_t`, please refer to API reference below

Application Example

Simple WebSocket example that uses esp_websocket_client to establish a websocket connection and send/receive data with the websocket.org Server: protocols/websocket.

API Reference

Header File

- esp_websocket_client/include/esp_websocket_client.h

Functions

esp_websocket_client_handle_t **esp_websocket_client_init**(**const** *esp_websocket_client_config_t* **config*)

Start a Websocket session This function must be the first function to call, and it returns a esp_websocket_client_handle_t that you must use as input to other functions in the interface. This call MUST have a corresponding call to esp_websocket_client_destroy when the operation is complete.

Return

- `esp_websocket_client_handle_t`
- NULL if any errors

Parameters

- `config`: The configuration

esp_err_t **esp_websocket_client_set_uri**(*esp_websocket_client_handle_t* *client*, **const** char **uri*)

Set URL for client, when performing this behavior, the options in the URL will replace the old ones Must stop the WebSocket client before set URI if the client has been connected.

Return esp_err_t

Parameters

- `client`: The client
- `uri`: The uri

esp_err_t **esp_websocket_client_start**(*esp_websocket_client_handle_t* *client*)

Open the WebSocket connection.

Return esp_err_t

Parameters

- `client`: The client

esp_err_t **esp_websocket_client_stop**(*esp_websocket_client_handle_t client*)

Close the WebSocket connection.

Return esp_err_t

Parameters

- `client`: The client

esp_err_t **esp_websocket_client_destroy**(*esp_websocket_client_handle_t client*)

Destroy the WebSocket connection and free all resources. This function must be the last function to call for an session. It is the opposite of the esp_websocket_client_init function and must be called with the same handle as input that a esp_websocket_client_init call returned. This might close all connections this handle has used.

Return esp_err_t

Parameters

- `client`: The client

int **esp_websocket_client_send**(*esp_websocket_client_handle_t client*, **const** char **data*, int *len*, TickType_t *timeout*)

Generic write data to the WebSocket connection; defaults to binary send.

Return

- Number of data was sent
- (-1) if any errors

Parameters

- `client`: The client
- `data`: The data
- `len`: The length
- `timeout`: Write data timeout in RTOS ticks

int **esp_websocket_client_send_bin**(*esp_websocket_client_handle_t client*, **const** char **data*, int *len*, TickType_t *timeout*)

Write binary data to the WebSocket connection (data send with WS OPCODE=02, i.e. binary)

Return

- Number of data was sent
- (-1) if any errors

Parameters

- `client`: The client
- `data`: The data
- `len`: The length
- `timeout`: Write data timeout in RTOS ticks

int **esp_websocket_client_send_text**(*esp_websocket_client_handle_t client*, **const** char **data*, int *len*, TickType_t *timeout*)

Write textual data to the WebSocket connection (data send with WS OPCODE=01, i.e. text)

Return

- Number of data was sent
- (-1) if any errors

Parameters

- `client`: The client
- `data`: The data
- `len`: The length
- `timeout`: Write data timeout in RTOS ticks

bool **esp_websocket_client_is_connected**(*esp_websocket_client_handle_t client*)

Check the WebSocket connection status.

Return

- true
- false

Parameters

- `client`: The client handle

esp_err_t **esp_websocket_register_events**(*esp_websocket_client_handle_t client*, *esp_websocket_event_id_t event*, *esp_event_handler_t event_handler*, void **event_handler_arg*)

Register the Websocket Events.

Return esp_err_t

Parameters

- `client`: The client handle

- `event`: The event id
- `event_handler`: The callback function
- `event_handler_arg`: User context

Structures

struct esp_websocket_event_data_t

Websocket event data.

Public Members

const char ***data_ptr**

Data pointer

int **data_len**

Data length

uint8_t **op_code**

Received opcode

esp_websocket_client_handle_t **client**

esp_websocket_client_handle_t context

void ***user_context**

user_data context, from *esp_websocket_client_config_t* user_data

int **payload_len**

Total payload length, payloads exceeding buffer will be posted through multiple events

int **payload_offset**

Actual offset for the data associated with this event

struct esp_websocket_client_config_t

Websocket client setup configuration.

Public Members

const char ***uri**

Websocket URI, the information on the URI can be overrides the other fields below, if any

const char ***host**

Domain or IP as string

int **port**

Port to connect, default depend on esp_websocket_transport_t (80 or 443)

const char ***username**

Using for Http authentication - Not supported for now

const char ***password**

Using for Http authentication - Not supported for now

const char ***path**

HTTP Path, if not set, default is `/`

bool **disable_auto_reconnect**

Disable the automatic reconnect function when disconnected

void ***user_context**

HTTP user data context

int **task_prio**

Websocket task priority

int **task_stack**

Websocket task stack

int **buffer_size**

Websocket buffer size

const char ***cert_pem**

SSL Certification, PEM format as string, if the client requires to verify server

esp_websocket_transport_t **transport**

Websocket transport type, see 'esp_websocket_transport_t

char ***subprotocol**

Websocket subprotocol

char ***user_agent**

Websocket user-agent

char ***headers**

Websocket additional headers

Type Definitions

typedef struct esp_websocket_client ***esp_websocket_client_handle_t**

Enumerations

enum esp_websocket_event_id_t

Websocket Client events id.

Values:

WEBSOCKET_EVENT_ANY = -1

WEBSOCKET_EVENT_ERROR = 0

This event occurs when there are any errors during execution

WEBSOCKET_EVENT_CONNECTED

Once the Websocket has been connected to the server, no data exchange has been performed

WEBSOCKET_EVENT_DISCONNECTED

The connection has been disconnected

WEBSOCKET_EVENT_DATA

When receiving data from the server, possibly multiple portions of the packet

WEBSOCKET_EVENT_MAX

enum esp_websocket_transport_t

Websocket Client transport.

Values:

WEBSOCKET_TRANSPORT_UNKNOWN = 0x0

Transport unknown

WEBSOCKET_TRANSPORT_OVER_TCP

Transport over tcp

WEBSOCKET_TRANSPORT_OVER_SSL

Transport over ssl

2.4.5 HTTP 服务器

[English]

概述

HTTP Server 组件提供了在 ESP32 上运行轻量级 Web 服务器的功能，下面介绍使用 HTTP Server 组件 API 的详细步骤：

- *httpd_start()*：创建 HTTP 服务器的实例，根据具体的配置为其分配内存和资源，并返回该服务器实例的句柄。服务器使用了两个套接字，一个用来监听 HTTP 流量（TCP 类型），另一个用来处理控制信号（UDP 类型），它们在服务器的任务循环中轮流使用。通过向 httpd_start() 传递 httpd_config_t 结构体，可以在创建服务器实例时配置任务的优先级和堆栈的大小。TCP 流量被解析为 HTTP 请求，根据请求的 URI 来调用用户注册的处理程序，在处理程序中需要发送回 HTTP 响应数据包。
- *httpd_stop()*：根据传入的句柄停止服务器，并释放相关联的内存和资源。这是一个阻塞函数，首先给服务器任务发送停止信号，然后等待其终止。期间服务器任务会关闭所有已打开的连接，删除已注册的 URI 处理程序，并将所有会话的上下文数据重置为空。

- *httpd_register_uri_handler()*：通过传入 httpd_uri_t 结构体类型的对象来注册 URI 处理程序。该结构体包含如下成员：uri 名字，method 类型（比如 HTTPD_GET/HTTPD_POST/HTTPD_PUT 等等），esp_err_t *handler (httpd_req_t *req) 类型的函数指针，指向用户上下文数据的 user_ctx 指针。

应用示例

```
/* URI 处理函数，在客户端发起 GET /uri 请求时被调用 */
esp_err_t get_handler(httpd_req_t *req)
{
    /* 发送回简单的响应数据包 */
    const char[] resp = "URI GET Response";
    httpd_resp_send(req, resp, strlen(resp));
    return ESP_OK;
}

/* URI 处理函数，在客户端发起 POST /uri 请求时被调用 */
esp_err_t post_handler(httpd_req_t *req)
{
    /* 定义 HTTP POST 请求数据的目标缓存区
     * httpd_req_recv() 只接收 char* 数据，但也可以是任意二进制数据（需要类型转换）
     * 对于字符串数据，null 终止符会被省略，content_len 会给出字符串的长度 */
    char[100] content;

    /* 如果内容长度大于缓冲区则截断 */
    size_t recv_size = MIN(req->content_len, sizeof(content));

    int ret = httpd_req_recv(req, content, recv_size);
    if (ret <= 0) {  /* 返回 0 表示连接已关闭 */
        /* 检查是否超时 */
        if (ret == HTTPD_SOCK_ERR_TIMEOUT) {
            /* 如果是超时，可以调用 httpd_req_recv() 重试
             * 简单起见，这里我们直接响应 HTTP 408（请求超时）错误给客户端 */
            httpd_resp_send_408(req);
        }
        /* 如果发生了错误，返回 ESP_FAIL 可以确保底层套接字被关闭 */
        return ESP_FAIL;
    }

    /* 发送简单的响应数据包 */
```

（下页继续）

(续上页)

```
    const char[] resp = "URI POST Response";
    httpd_resp_send(req, resp, strlen(resp));
    return ESP_OK;
}

/* GET /uri 的 URI 处理结构 */
httpd_uri_t uri_get = {
    .uri      = "/uri",
    .method   = HTTP_GET,
    .handler  = get_handler,
    .user_ctx = NULL
};

/* POST /uri 的 URI 处理结构 */
httpd_uri_t uri_post = {
    .uri      = "/uri",
    .method   = HTTP_POST,
    .handler  = post_handler,
    .user_ctx = NULL
};

/* 启动 Web 服务器的函数 */
httpd_handle_t start_webserver(void)
{
    /* 生成默认的配置参数 */
    httpd_config_t config = HTTPD_DEFAULT_CONFIG();

    /* 置空 esp_http_server 的实例句柄 */
    httpd_handle_t server = NULL;

    /* 启动 httpd server */
    if (httpd_start(&server, &config) == ESP_OK) {
        /* 注册 URI 处理程序 */
        httpd_register_uri_handler(server, &uri_get);
        httpd_register_uri_handler(server, &uri_post);
    }
    /* 如果服务器启动失败，返回的句柄是 NULL */
    return server;
}
```

(下页继续)

(续上页)

```
/* 停止 Web 服务器的函数 */
void stop_webserver(httpd_handle_t server)
{
    if (server) {
        /* 停止 httpd server */
        httpd_stop(server);
    }
}
```

简单 HTTP 服务器示例

请查看位于 protocols/http_server/simple 的 HTTP 服务器示例，该示例演示了如何处理任意内容长度的数据，读取请求头和 URL 查询参数，设置响应头。

HTTP 长连接

HTTP 服务器具有长连接的功能，允许重复使用同一个连接（会话）进行多次传输，同时保持会话的上下文数据。上下文数据可由处理程序动态分配，在这种情况下需要提前指定好自定义的回调函数，以便在连接/会话被关闭时释放这部分内存资源。

长连接示例

```
/* 自定义函数，用来释放上下文数据 */
void free_ctx_func(void *ctx)
{
    /* 也可以是 free 以外的代码逻辑 */
    free(ctx);
}

esp_err_t adder_post_handler(httpd_req_t *req)
{
    /* 如果会话上下文还不存在则新建一个 */
    if (! req->sess_ctx) {
        req->sess_ctx = malloc(sizeof(ANY_DATA_TYPE));  /*!< 指向上下文数据 */
        req->free_ctx = free_ctx_func;                  /*!< 释放上下文数据的函数 */
    }

    /* 访问上下文数据 */
```

(下页继续)

(续上页)

```
    ANY_DATA_TYPE *ctx_data = (ANY_DATA_TYPE *)req->sess_ctx;

    /* 响应 */
    ................
    ................
    ................

    return ESP_OK;
}
```

详情请参考位于 protocols/http_server/persistent_sockets 的示例代码。

API 参考

Header File

- esp_http_server/include/esp_http_server.h

Functions

esp_err_t **httpd_register_uri_handler**(*httpd_handle_t handle*, **const** *httpd_uri_t* **uri_handler*)

Registers a URI handler.

Example usage:

```
esp_err_t my_uri_handler(httpd_req_t* req)
{
    // Recv , Process and Send
    ....
    ....
    ....

    // Fail condition
    if (....) {
        // Return fail to close session //
        return ESP_FAIL;
    }

    // On success
    return ESP_OK;
```

(下页继续)

(续上页)

```
}

// URI handler structure
httpd_uri_t my_uri {
    .uri      = "/my_uri/path/xyz",
    .method   = HTTPD_GET,
    .handler  = my_uri_handler,
    .user_ctx = NULL
};

// Register handler
if (httpd_register_uri_handler(server_handle, &my_uri) != ESP_OK) {
   // If failed to register handler
   ....
}
```

Note URI handlers can be registered in real time as long as the server handle is valid.

Return

- ESP_OK : On successfully registering the handler
- ESP_ERR_INVALID_ARG : Null arguments
- ESP_ERR_HTTPD_HANDLERS_FULL : If no slots left for new handler
- ESP_ERR_HTTPD_HANDLER_EXISTS : If handler with same URI and method is already registered

Parameters

- `handle`: handle to HTTPD server instance
- `uri_handler`: pointer to handler that needs to be registered

esp_err_t **httpd_unregister_uri_handler**(*httpd_handle_t handle*, **const** char **uri*, *httpd_method_t method*)

Unregister a URI handler.

Return

- ESP_OK : On successfully deregistering the handler
- ESP_ERR_INVALID_ARG : Null arguments
- ESP_ERR_NOT_FOUND : Handler with specified URI and method not found

Parameters

- `handle`: handle to HTTPD server instance
- `uri`: URI string
- `method`: HTTP method

esp_err_t **httpd_unregister_uri**(*httpd_handle_t handle*, **const** char **uri*)

Unregister all URI handlers with the specified uri string.

Return

- ESP_OK : On successfully deregistering all such handlers
- ESP_ERR_INVALID_ARG : Null arguments
- ESP_ERR_NOT_FOUND : No handler registered with specified uri string

Parameters

- `handle`: handle to HTTPD server instance
- `uri`: uri string specifying all handlers that need to be deregisterd

esp_err_t **httpd_sess_set_recv_override**(*httpd_handle_t hd*, int *sockfd*, *httpd_recv_func_t recv_func*)

Override web server's receive function (by session FD)

This function overrides the web server's receive function. This same function is used to read HTTP request packets.

Note This API is supposed to be called either from the context of

- an http session APIs where sockfd is a valid parameter
- a URI handler where sockfd is obtained using httpd_req_to_sockfd()

Return

- ESP_OK : On successfully registering override
- ESP_ERR_INVALID_ARG : Null arguments

Parameters

- `hd`: HTTPD instance handle
- `sockfd`: Session socket FD
- `recv_func`: The receive function to be set for this session

esp_err_t **httpd_sess_set_send_override**(*httpd_handle_t hd*, int *sockfd*, *httpd_send_func_t send_func*)

Override web server's send function (by session FD)

This function overrides the web server’ s send function. This same function is used to send out any response to any HTTP request.

Note This API is supposed to be called either from the context of

- an http session APIs where sockfd is a valid parameter
- a URI handler where sockfd is obtained using httpd_req_to_sockfd()

Return

- ESP_OK : On successfully registering override
- ESP_ERR_INVALID_ARG : Null arguments

Parameters

- `hd`: HTTPD instance handle
- `sockfd`: Session socket FD
- `send_func`: The send function to be set for this session

esp_err_t **httpd_sess_set_pending_override**(*httpd_handle_t hd*, int *sockfd*, *httpd_pending_func_t pending_func*)

Override web server’ s pending function (by session FD)

This function overrides the web server’ s pending function. This function is used to test for pending bytes in a socket.

Note This API is supposed to be called either from the context of

- an http session APIs where sockfd is a valid parameter
- a URI handler where sockfd is obtained using httpd_req_to_sockfd()

Return

- ESP_OK : On successfully registering override
- ESP_ERR_INVALID_ARG : Null arguments

Parameters

- `hd`: HTTPD instance handle
- `sockfd`: Session socket FD
- `pending_func`: The receive function to be set for this session

int **httpd_req_to_sockfd**(*httpd_req_t* *r)

Get the Socket Descriptor from the HTTP request.

This API will return the socket descriptor of the session for which URI handler was executed on reception of HTTP request. This is useful when user wants to call functions that require session

socket fd, from within a URI handler, ie. : httpd_sess_get_ctx(), httpd_sess_trigger_close(), httpd_sess_update_lru_counter().

Note This API is supposed to be called only from the context of a URI handler where httpd_req_t* request pointer is valid.

Return

- Socket descriptor : The socket descriptor for this request
- -1 : Invalid/NULL request pointer

Parameters

- `r`: The request whose socket descriptor should be found

int **httpd_req_recv**(*httpd_req_t* *r, char *buf, size_t buf_len)

API to read content data from the HTTP request.

This API will read HTTP content data from the HTTP request into provided buffer. Use content_len provided in httpd_req_t structure to know the length of data to be fetched. If content_len is too large for the buffer then user may have to make multiple calls to this function, each time fetching 'buf_len' number of bytes, while the pointer to content data is incremented internally by the same number.

Note

- This API is supposed to be called only from the context of a URI handler where httpd_req_t* request pointer is valid.
- If an error is returned, the URI handler must further return an error. This will ensure that the erroneous socket is closed and cleaned up by the web server.
- Presently Chunked Encoding is not supported

Return

- Bytes : Number of bytes read into the buffer successfully
- 0 : Buffer length parameter is zero / connection closed by peer
- HTTPD_SOCK_ERR_INVALID : Invalid arguments
- HTTPD_SOCK_ERR_TIMEOUT : Timeout/interrupted while calling socket recv()
- HTTPD_SOCK_ERR_FAIL : Unrecoverable error while calling socket recv()

Parameters

- `r`: The request being responded to
- `buf`: Pointer to a buffer that the data will be read into
- `buf_len`: Length of the buffer

size_t **httpd_req_get_hdr_value_len**(*httpd_req_t* *r, **const** char *field)

Search for a field in request headers and return the string length of it' s value.

Note

- This API is supposed to be called only from the context of a URI handler where httpd_req_t* request pointer is valid.
- Once httpd_resp_send() API is called all request headers are purged, so request headers need be copied into separate buffers if they are required later.

Return

- Length : If field is found in the request URL
- Zero : Field not found / Invalid request / Null arguments

Parameters

- `r`: The request being responded to
- `field`: The header field to be searched in the request

esp_err_t **httpd_req_get_hdr_value_str**(*httpd_req_t* *r, **const** char *field, char *val, size_t *val_size*)

Get the value string of a field from the request headers.

Note

- This API is supposed to be called only from the context of a URI handler where httpd_req_t* request pointer is valid.
- Once httpd_resp_send() API is called all request headers are purged, so request headers need be copied into separate buffers if they are required later.
- If output size is greater than input, then the value is truncated, accompanied by truncation error as return value.
- Use httpd_req_get_hdr_value_len() to know the right buffer length

Return

- ESP_OK : Field found in the request header and value string copied
- ESP_ERR_NOT_FOUND : Key not found
- ESP_ERR_INVALID_ARG : Null arguments
- ESP_ERR_HTTPD_INVALID_REQ : Invalid HTTP request pointer
- ESP_ERR_HTTPD_RESULT_TRUNC : Value string truncated

Parameters

- `r`: The request being responded to

- `field`: The field to be searched in the header
- `val`: Pointer to the buffer into which the value will be copied if the field is found
- `val_size`: Size of the user buffer "val"

size_t **`httpd_req_get_url_query_len`**(*httpd_req_t* *r)

Get Query string length from the request URL.

Note This API is supposed to be called only from the context of a URI handler where httpd_req_t* request pointer is valid

Return

- Length : Query is found in the request URL
- Zero : Query not found / Null arguments / Invalid request

Parameters

- `r`: The request being responded to

esp_err_t **`httpd_req_get_url_query_str`**(*httpd_req_t* *r, char *buf, size_t *buf_len*)

Get Query string from the request URL.

Note

- Presently, the user can fetch the full URL query string, but decoding will have to be performed by the user. Request headers can be read using httpd_req_get_hdr_value_str() to know the 'Content-Type' (eg. Content-Type: application/x-www-form-urlencoded) and then the appropriate decoding algorithm needs to be applied.
- This API is supposed to be called only from the context of a URI handler where httpd_req_t* request pointer is valid
- If output size is greater than input, then the value is truncated, accompanied by truncation error as return value
- Prior to calling this function, one can use httpd_req_get_url_query_len() to know the query string length beforehand and hence allocate the buffer of right size (usually query string length + 1 for null termination) for storing the query string

Return

- ESP_OK : Query is found in the request URL and copied to buffer
- ESP_ERR_NOT_FOUND : Query not found
- ESP_ERR_INVALID_ARG : Null arguments
- ESP_ERR_HTTPD_INVALID_REQ : Invalid HTTP request pointer
- ESP_ERR_HTTPD_RESULT_TRUNC : Query string truncated

Parameters

- `r`: The request being responded to
- `buf`: Pointer to the buffer into which the query string will be copied (if found)
- `buf_len`: Length of output buffer

esp_err_t **httpd_query_key_value(const** char *_qry_, **const** char *_key_, char *_val_, size_t _val_size_)

Helper function to get a URL query tag from a query string of the type param1=val1&param2=val2.

Note

- The components of URL query string (keys and values) are not URLdecoded. The user must check for 'Content-Type' field in the request headers and then depending upon the specified encoding (URLencoded or otherwise) apply the appropriate decoding algorithm.
- If actual value size is greater than val_size, then the value is truncated, accompanied by truncation error as return value.

Return

- ESP_OK : Key is found in the URL query string and copied to buffer
- ESP_ERR_NOT_FOUND : Key not found
- ESP_ERR_INVALID_ARG : Null arguments
- ESP_ERR_HTTPD_RESULT_TRUNC : Value string truncated

Parameters

- `qry`: Pointer to query string
- `key`: The key to be searched in the query string
- `val`: Pointer to the buffer into which the value will be copied if the key is found
- `val_size`: Size of the user buffer "val"

bool **httpd_uri_match_wildcard(const** char *_uri_template_, **const** char *_uri_to_match_, size_t _match_upto_)

Test if a URI matches the given wildcard template.

Template may end with "?" to make the previous character optional (typically a slash), "*" for a wildcard match, and "?*" to make the previous character optional, and if present, allow anything to follow.

Example:

- * matches everything
- /foo/? matches /foo and /foo/
- /foo/* (sans the backslash) matches /foo/ and /foo/bar, but not /foo or /fo

- /foo/?* or /foo/*? (sans the backslash) matches /foo/, /foo/bar, and also /foo, but not /foox or /fo

The special characters "?" and "*" anywhere else in the template will be taken literally.

Return true if a match was found

Parameters

- `uri_template`: URI template (pattern)
- `uri_to_match`: URI to be matched
- `match_upto`: how many characters of the URI buffer to test (there may be trailing query string etc.)

esp_err_t **httpd_resp_send**(*httpd_req_t* *r, **const** char *buf, ssize_t buf_len)

API to send a complete HTTP response.

This API will send the data as an HTTP response to the request. This assumes that you have the entire response ready in a single buffer. If you wish to send response in incremental chunks use httpd_resp_send_chunk() instead.

If no status code and content-type were set, by default this will send 200 OK status code and content type as text/html. You may call the following functions before this API to configure the response headers : httpd_resp_set_status() - for setting the HTTP status string, httpd_resp_set_type() - for setting the Content Type, httpd_resp_set_hdr() - for appending any additional field value entries in the response header

Note

- This API is supposed to be called only from the context of a URI handler where httpd_req_t* request pointer is valid.
- Once this API is called, the request has been responded to.
- No additional data can then be sent for the request.
- Once this API is called, all request headers are purged, so request headers need be copied into separate buffers if they are required later.

Return

- ESP_OK : On successfully sending the response packet
- ESP_ERR_INVALID_ARG : Null request pointer
- ESP_ERR_HTTPD_RESP_HDR : Essential headers are too large for internal buffer
- ESP_ERR_HTTPD_RESP_SEND : Error in raw send
- ESP_ERR_HTTPD_INVALID_REQ : Invalid request

Parameters

- `r`: The request being responded to
- `buf`: Buffer from where the content is to be fetched
- `buf_len`: Length of the buffer, HTTPD_RESP_USE_STRLEN to use strlen()

esp_err_t **httpd_resp_send_chunk**(*httpd_req_t* *r, **const** char *buf, ssize_t buf_len)

API to send one HTTP chunk.

This API will send the data as an HTTP response to the request. This API will use chunked-encoding and send the response in the form of chunks. If you have the entire response contained in a single buffer, please use httpd_resp_send() instead.

If no status code and content-type were set, by default this will send 200 OK status code and content type as text/html. You may call the following functions before this API to configure the response headers httpd_resp_set_status() - for setting the HTTP status string, httpd_resp_set_type() - for setting the Content Type, httpd_resp_set_hdr() - for appending any additional field value entries in the response header

Note

- This API is supposed to be called only from the context of a URI handler where httpd_req_t* request pointer is valid.
- When you are finished sending all your chunks, you must call this function with buf_len as 0.
- Once this API is called, all request headers are purged, so request headers need be copied into separate buffers if they are required later.

Return

- ESP_OK : On successfully sending the response packet chunk
- ESP_ERR_INVALID_ARG : Null request pointer
- ESP_ERR_HTTPD_RESP_HDR : Essential headers are too large for internal buffer
- ESP_ERR_HTTPD_RESP_SEND : Error in raw send
- ESP_ERR_HTTPD_INVALID_REQ : Invalid request pointer

Parameters

- `r`: The request being responded to
- `buf`: Pointer to a buffer that stores the data
- `buf_len`: Length of the buffer, HTTPD_RESP_USE_STRLEN to use strlen()

`static` *esp_err_t* `httpd_resp_sendstr(`*httpd_req_t* *r, `const` char *str`)`

API to send a complete string as HTTP response.

This API simply calls http_resp_send with buffer length set to string length assuming the buffer contains a null terminated string

Return

- ESP_OK : On successfully sending the response packet
- ESP_ERR_INVALID_ARG : Null request pointer
- ESP_ERR_HTTPD_RESP_HDR : Essential headers are too large for internal buffer
- ESP_ERR_HTTPD_RESP_SEND : Error in raw send
- ESP_ERR_HTTPD_INVALID_REQ : Invalid request

Parameters

- `r`: The request being responded to
- `str`: String to be sent as response body

`static` *esp_err_t* `httpd_resp_sendstr_chunk(`*httpd_req_t* *r, `const` char *str`)`

API to send a string as an HTTP response chunk.

This API simply calls http_resp_send_chunk with buffer length set to string length assuming the buffer contains a null terminated string

Return

- ESP_OK : On successfully sending the response packet
- ESP_ERR_INVALID_ARG : Null request pointer
- ESP_ERR_HTTPD_RESP_HDR : Essential headers are too large for internal buffer
- ESP_ERR_HTTPD_RESP_SEND : Error in raw send
- ESP_ERR_HTTPD_INVALID_REQ : Invalid request

Parameters

- `r`: The request being responded to
- `str`: String to be sent as response body (NULL to finish response packet)

esp_err_t `httpd_resp_set_status(`*httpd_req_t* *r, `const` char *status`)`

API to set the HTTP status code.

This API sets the status of the HTTP response to the value specified. By default, the '200 OK' response is sent as the response.

Note

- This API is supposed to be called only from the context of a URI handler where httpd_req_t* request pointer is valid.
- This API only sets the status to this value. The status isn't sent out until any of the send APIs is executed.
- Make sure that the lifetime of the status string is valid till send function is called.

Return

- ESP_OK : On success
- ESP_ERR_INVALID_ARG : Null arguments
- ESP_ERR_HTTPD_INVALID_REQ : Invalid request pointer

Parameters

- `r`: The request being responded to
- `status`: The HTTP status code of this response

esp_err_t **httpd_resp_set_type**(*httpd_req_t* *r, **const** char *type)

API to set the HTTP content type.

This API sets the 'Content Type' field of the response. The default content type is 'text/html'.

Note

- This API is supposed to be called only from the context of a URI handler where httpd_req_t* request pointer is valid.
- This API only sets the content type to this value. The type isn't sent out until any of the send APIs is executed.
- Make sure that the lifetime of the type string is valid till send function is called.

Return

- ESP_OK : On success
- ESP_ERR_INVALID_ARG : Null arguments
- ESP_ERR_HTTPD_INVALID_REQ : Invalid request pointer

Parameters

- `r`: The request being responded to
- `type`: The Content Type of the response

esp_err_t **httpd_resp_set_hdr**(*httpd_req_t* *r, **const** char *field, **const** char *value)

API to append any additional headers.

This API sets any additional header fields that need to be sent in the response.

Note

- This API is supposed to be called only from the context of a URI handler where httpd_req_t* request pointer is valid.
- The header isn't sent out until any of the send APIs is executed.
- The maximum allowed number of additional headers is limited to value of max_resp_headers in config structure.
- Make sure that the lifetime of the field value strings are valid till send function is called.

Return

- ESP_OK : On successfully appending new header
- ESP_ERR_INVALID_ARG : Null arguments
- ESP_ERR_HTTPD_RESP_HDR : Total additional headers exceed max allowed
- ESP_ERR_HTTPD_INVALID_REQ : Invalid request pointer

Parameters

- `r`: The request being responded to
- `field`: The field name of the HTTP header
- `value`: The value of this HTTP header

esp_err_t **httpd_resp_send_err**(*httpd_req_t* **req*, *httpd_err_code_t error*, **const** char **msg*)

For sending out error code in response to HTTP request.

Note

- This API is supposed to be called only from the context of a URI handler where httpd_req_t* request pointer is valid.
- Once this API is called, all request headers are purged, so request headers need be copied into separate buffers if they are required later.
- If you wish to send additional data in the body of the response, please use the lower-level functions directly.

Return

- ESP_OK : On successfully sending the response packet
- ESP_ERR_INVALID_ARG : Null arguments
- ESP_ERR_HTTPD_RESP_SEND : Error in raw send
- ESP_ERR_HTTPD_INVALID_REQ : Invalid request pointer

Parameters

- `req`: Pointer to the HTTP request for which the response needs to be sent
- `error`: Error type to send
- `msg`: Error message string (pass NULL for default message)

`static` *esp_err_t* `httpd_resp_send_404(`*httpd_req_t* *`r)`

Helper function for HTTP 404.

Send HTTP 404 message. If you wish to send additional data in the body of the response, please use the lower-level functions directly.

Note

- This API is supposed to be called only from the context of a URI handler where httpd_req_t* request pointer is valid.
- Once this API is called, all request headers are purged, so request headers need be copied into separate buffers if they are required later.

Return

- ESP_OK : On successfully sending the response packet
- ESP_ERR_INVALID_ARG : Null arguments
- ESP_ERR_HTTPD_RESP_SEND : Error in raw send
- ESP_ERR_HTTPD_INVALID_REQ : Invalid request pointer

Parameters

- `r`: The request being responded to

`static` *esp_err_t* `httpd_resp_send_408(`*httpd_req_t* *`r)`

Helper function for HTTP 408.

Send HTTP 408 message. If you wish to send additional data in the body of the response, please use the lower-level functions directly.

Note

- This API is supposed to be called only from the context of a URI handler where httpd_req_t* request pointer is valid.
- Once this API is called, all request headers are purged, so request headers need be copied into separate buffers if they are required later.

Return

- ESP_OK : On successfully sending the response packet

- ESP_ERR_INVALID_ARG : Null arguments
- ESP_ERR_HTTPD_RESP_SEND : Error in raw send
- ESP_ERR_HTTPD_INVALID_REQ : Invalid request pointer

Parameters

- `r`: The request being responded to

`static` *esp_err_t* `httpd_resp_send_500(`*httpd_req_t* *r`)`

Helper function for HTTP 500.

Send HTTP 500 message. If you wish to send additional data in the body of the response, please use the lower-level functions directly.

Note

- This API is supposed to be called only from the context of a URI handler where httpd_req_t* request pointer is valid.
- Once this API is called, all request headers are purged, so request headers need be copied into separate buffers if they are required later.

Return

- ESP_OK : On successfully sending the response packet
- ESP_ERR_INVALID_ARG : Null arguments
- ESP_ERR_HTTPD_RESP_SEND : Error in raw send
- ESP_ERR_HTTPD_INVALID_REQ : Invalid request pointer

Parameters

- `r`: The request being responded to

int `httpd_send(`*httpd_req_t* *r, `const` char *buf, size_t *buf_len*`)`

Raw HTTP send.

Call this API if you wish to construct your custom response packet. When using this, all essential header, eg. HTTP version, Status Code, Content Type and Length, Encoding, etc. will have to be constructed manually, and HTTP delimeters (CRLF) will need to be placed correctly for separating sub-sections of the HTTP response packet.

If the send override function is set, this API will end up calling that function eventually to send data out.

Note

- This API is supposed to be called only from the context of a URI handler where httpd_req_t* request pointer is valid.

- Unless the response has the correct HTTP structure (which the user must now ensure) it is not guaranteed that it will be recognized by the client. For most cases, you wouldn' t have to call this API, but you would rather use either of : httpd_resp_send(), httpd_resp_send_chunk()

Return

- Bytes : Number of bytes that were sent successfully
- HTTPD_SOCK_ERR_INVALID : Invalid arguments
- HTTPD_SOCK_ERR_TIMEOUT : Timeout/interrupted while calling socket send()
- HTTPD_SOCK_ERR_FAIL : Unrecoverable error while calling socket send()

Parameters

- `r`: The request being responded to
- `buf`: Buffer from where the fully constructed packet is to be read
- `buf_len`: Length of the buffer

esp_err_t **httpd_register_err_handler**(*httpd_handle_t handle*, *httpd_err_code_t error*, *httpd_err_handler_func_t handler_fn*)

Function for registering HTTP error handlers.

This function maps a handler function to any supported error code given by `httpd_err_code_t`. See prototype `httpd_err_handler_func_t` above for details.

Return

- ESP_OK : handler registered successfully
- ESP_ERR_INVALID_ARG : invalid error code or server handle

Parameters

- `handle`: HTTP server handle
- `error`: Error type
- `handler_fn`: User implemented handler function (Pass NULL to unset any previously set handler)

esp_err_t **httpd_start**(*httpd_handle_t* **handle*, **const** *httpd_config_t* **config*)

Starts the web server.

Create an instance of HTTP server and allocate memory/resources for it depending upon the specified configuration.

Example usage:

```
//Function for starting the webserver
httpd_handle_t start_webserver(void)
{
    // Generate default configuration
    httpd_config_t config = HTTPD_DEFAULT_CONFIG();

    // Empty handle to http_server
    httpd_handle_t server = NULL;

    // Start the httpd server
    if (httpd_start(&server, &config) == ESP_OK) {
        // Register URI handlers
        httpd_register_uri_handler(server, &uri_get);
        httpd_register_uri_handler(server, &uri_post);
    }
    // If server failed to start, handle will be NULL
    return server;
}
```

Return

- ESP_OK : Instance created successfully
- ESP_ERR_INVALID_ARG : Null argument(s)
- ESP_ERR_HTTPD_ALLOC_MEM : Failed to allocate memory for instance
- ESP_ERR_HTTPD_TASK : Failed to launch server task

Parameters

- `config`: Configuration for new instance of the server
- `handle`: Handle to newly created instance of the server. NULL on error

esp_err_t **httpd_stop**(*httpd_handle_t handle*)

Stops the web server.

Deallocates memory/resources used by an HTTP server instance and deletes it. Once deleted the handle can no longer be used for accessing the instance.

Example usage:

```
// Function for stopping the webserver
void stop_webserver(httpd_handle_t server)
{
```

(下页继续)

(续上页)

```
    // Ensure handle is non NULL
    if (server != NULL) {
        // Stop the httpd server
        httpd_stop(server);
    }
}
```

Return

- ESP_OK : Server stopped successfully
- ESP_ERR_INVALID_ARG : Handle argument is Null

Parameters

- `handle`: Handle to server returned by httpd_start

esp_err_t **httpd_queue_work**(*httpd_handle_t handle*, *httpd_work_fn_t work*, void **arg*)

Queue execution of a function in HTTPD' s context.

This API queues a work function for asynchronous execution

Note Some protocols require that the web server generate some asynchronous data and send it to the persistently opened connection. This facility is for use by such protocols.

Return

- ESP_OK : On successfully queueing the work
- ESP_FAIL : Failure in ctrl socket
- ESP_ERR_INVALID_ARG : Null arguments

Parameters

- `handle`: Handle to server returned by httpd_start
- `work`: Pointer to the function to be executed in the HTTPD' s context
- `arg`: Pointer to the arguments that should be passed to this function

void **httpd_sess_get_ctx*(*httpd_handle_t handle*, int *sockfd*)

Get session context from socket descriptor.

Typically if a session context is created, it is available to URI handlers through the httpd_req_t structure. But, there are cases where the web server' s send/receive functions may require the context (for example, for accessing keying information etc). Since the send/receive function only have the socket descriptor at their disposal, this API provides them with a way to retrieve the session context.

Return

- void* : Pointer to the context associated with this session
- NULL : Empty context / Invalid handle / Invalid socket fd

Parameters

- `handle`: Handle to server returned by httpd_start
- `sockfd`: The socket descriptor for which the context should be extracted.

void **httpd_sess_set_ctx**(*httpd_handle_t handle*, int *sockfd*, void **ctx*, *httpd_free_ctx_fn_t free_fn*)

Set session context by socket descriptor.

Parameters

- `handle`: Handle to server returned by httpd_start
- `sockfd`: The socket descriptor for which the context should be extracted.
- `ctx`: Context object to assign to the session
- `free_fn`: Function that should be called to free the context

void ***httpd_sess_get_transport_ctx**(*httpd_handle_t handle*, int *sockfd*)

Get session 'transport' context by socket descriptor.

This context is used by the send/receive functions, for example to manage SSL context.

See httpd_sess_get_ctx()

Return

- void* : Pointer to the transport context associated with this session
- NULL : Empty context / Invalid handle / Invalid socket fd

Parameters

- `handle`: Handle to server returned by httpd_start
- `sockfd`: The socket descriptor for which the context should be extracted.

void **httpd_sess_set_transport_ctx**(*httpd_handle_t handle*, int *sockfd*, void **ctx*, *httpd_free_ctx_fn_t free_fn*)

Set session 'transport' context by socket descriptor.

See httpd_sess_set_ctx()

Parameters

- `handle`: Handle to server returned by httpd_start
- `sockfd`: The socket descriptor for which the context should be extracted.

- `ctx`: Transport context object to assign to the session
- `free_fn`: Function that should be called to free the transport context

void *`httpd_get_global_user_ctx`(*httpd_handle_t* *handle*)

Get HTTPD global user context (it was set in the server config struct)

Return global user context

Parameters

- `handle`: Handle to server returned by httpd_start

void *`httpd_get_global_transport_ctx`(*httpd_handle_t* *handle*)

Get HTTPD global transport context (it was set in the server config struct)

Return global transport context

Parameters

- `handle`: Handle to server returned by httpd_start

esp_err_t `httpd_sess_trigger_close`(*httpd_handle_t* *handle*, int *sockfd*)

Trigger an httpd session close externally.

Note Calling this API is only required in special circumstances wherein some application requires to close an httpd client session asynchronously.

Return

- ESP_OK : On successfully initiating closure
- ESP_FAIL : Failure to queue work
- ESP_ERR_NOT_FOUND : Socket fd not found
- ESP_ERR_INVALID_ARG : Null arguments

Parameters

- `handle`: Handle to server returned by httpd_start
- `sockfd`: The socket descriptor of the session to be closed

esp_err_t `httpd_sess_update_lru_counter`(*httpd_handle_t* *handle*, int *sockfd*)

Update LRU counter for a given socket.

LRU Counters are internally associated with each session to monitor how recently a session exchanged traffic. When LRU purge is enabled, if a client is requesting for connection but maximum number of sockets/sessions is reached, then the session having the earliest LRU counter is closed automatically.

Updating the LRU counter manually prevents the socket from being purged due to the Least Recently Used (LRU) logic, even though it might not have received traffic for some time. This is useful when all open sockets/session are frequently exchanging traffic but the user specifically wants one of the sessions to be kept open, irrespective of when it last exchanged a packet.

Note Calling this API is only necessary if the LRU Purge Enable option is enabled.

Return

- ESP_OK : Socket found and LRU counter updated
- ESP_ERR_NOT_FOUND : Socket not found
- ESP_ERR_INVALID_ARG : Null arguments

Parameters

- `handle`: Handle to server returned by httpd_start
- `sockfd`: The socket descriptor of the session for which LRU counter is to be updated

Structures

`struct httpd_config`

HTTP Server Configuration Structure.

Note Use HTTPD_DEFAULT_CONFIG() to initialize the configuration to a default value and then modify only those fields that are specifically determined by the use case.

Public Members

unsigned `task_priority`

Priority of FreeRTOS task which runs the server

size_t `stack_size`

The maximum stack size allowed for the server task

BaseType_t `core_id`

The core the HTTP server task will run on

uint16_t `server_port`

TCP Port number for receiving and transmitting HTTP traffic

uint16_t `ctrl_port`

UDP Port number for asynchronously exchanging control signals between various components of the server

uint16_t `max_open_sockets`

Max number of sockets/clients connected at any time

uint16_t **max_uri_handlers**

Maximum allowed uri handlers

uint16_t **max_resp_headers**

Maximum allowed additional headers in HTTP response

uint16_t **backlog_conn**

Number of backlog connections

bool **lru_purge_enable**

Purge "Least Recently Used" connection

uint16_t **recv_wait_timeout**

Timeout for recv function (in seconds)

uint16_t **send_wait_timeout**

Timeout for send function (in seconds)

void ***global_user_ctx**

Global user context.

This field can be used to store arbitrary user data within the server context. The value can be retrieved using the server handle, available e.g. in the httpd_req_t struct.

When shutting down, the server frees up the user context by calling free() on the global_user_ctx field. If you wish to use a custom function for freeing the global user context, please specify that here.

httpd_free_ctx_fn_t **global_user_ctx_free_fn**

Free function for global user context

void ***global_transport_ctx**

Global transport context.

Similar to global_user_ctx, but used for session encoding or encryption (e.g. to hold the SSL context). It will be freed using free(), unless global_transport_ctx_free_fn is specified.

httpd_free_ctx_fn_t **global_transport_ctx_free_fn**

Free function for global transport context

httpd_open_func_t **open_fn**

Custom session opening callback.

Called on a new session socket just after accept(), but before reading any data.

This is an opportunity to set up e.g. SSL encryption using global_transport_ctx and the send/recv/pending session overrides.

If a context needs to be maintained between these functions, store it in the session using httpd_sess_set_transport_ctx() and retrieve it later with httpd_sess_get_transport_ctx()

Returning a value other than ESP_OK will immediately close the new socket.

httpd_close_func_t **close_fn**

Custom session closing callback.

Called when a session is deleted, before freeing user and transport contexts and before closing the socket. This is a place for custom de-init code common to all sockets.

Set the user or transport context to NULL if it was freed here, so the server does not try to free it again.

This function is run for all terminated sessions, including sessions where the socket was closed by the network stack - that is, the file descriptor may not be valid anymore.

httpd_uri_match_func_t **uri_match_fn**

URI matcher function.

Called when searching for a matching URI: 1) whose request handler is to be executed right after an HTTP request is successfully parsed 2) in order to prevent duplication while registering a new URI handler using `httpd_register_uri_handler()`

Available options are: 1) NULL : Internally do basic matching using `strncmp()` 2) `httpd_uri_match_wildcard()` : URI wildcard matcher

Users can implement their own matching functions (See description of the `httpd_uri_match_func_t` function prototype)

struct httpd_req

HTTP Request Data Structure.

Public Members

httpd_handle_t **handle**

Handle to server instance

int **method**

The type of HTTP request, -1 if unsupported method

const char **uri**[**HTTPD_MAX_URI_LEN** + 1]

The URI of this request (1 byte extra for null termination)

size_t **content_len**

Length of the request body

void ***aux**

Internally used members

void ***user_ctx**

User context pointer passed during URI registration.

void ***sess_ctx**

Session Context Pointer

A session context. Contexts are maintained across 'sessions' for a given open TCP connection. One session could have multiple request responses. The web server will ensure that the context persists across all these request and responses.

By default, this is NULL. URI Handlers can set this to any meaningful value.

If the underlying socket gets closed, and this pointer is non-NULL, the web server will free up the context by calling free(), unless free_ctx function is set.

httpd_free_ctx_fn_t **free_ctx**

Pointer to free context hook

Function to free session context

If the web server' s socket closes, it frees up the session context by calling free() on the sess_ctx member. If you wish to use a custom function for freeing the session context, please specify that here.

bool **ignore_sess_ctx_changes**

Flag indicating if Session Context changes should be ignored

By default, if you change the sess_ctx in some URI handler, the http server will internally free the earlier context (if non NULL), after the URI handler returns. If you want to manage the allocation/reallocation/freeing of sess_ctx yourself, set this flag to true, so that the server will not perform any checks on it. The context will be cleared by the server (by calling free_ctx or free()) only if the socket gets closed.

struct httpd_uri

Structure for URI handler.

Public Members

const char ***uri**

The URI to handle

httpd_method_t **method**

Method supported by the URI

esp_err_t (***handler**)(*httpd_req_t* *r)

Handler to call for supported request method. This must return ESP_OK, or else the underlying socket will be closed.

void ***user_ctx**

Pointer to user context data which will be available to handler

Macros

HTTPD_MAX_REQ_HDR_LEN

HTTPD_MAX_URI_LEN

HTTPD_SOCK_ERR_FAIL

HTTPD_SOCK_ERR_INVALID

HTTPD_SOCK_ERR_TIMEOUT

HTTPD_200
: HTTP Response 200

HTTPD_204
: HTTP Response 204

HTTPD_207
: HTTP Response 207

HTTPD_400
: HTTP Response 400

HTTPD_404
: HTTP Response 404

HTTPD_408
: HTTP Response 408

HTTPD_500
: HTTP Response 500

HTTPD_TYPE_JSON
: HTTP Content type JSON

HTTPD_TYPE_TEXT
: HTTP Content type text/HTML

HTTPD_TYPE_OCTET
: HTTP Content type octext-stream

HTTPD_DEFAULT_CONFIG()

ESP_ERR_HTTPD_BASE
: Starting number of HTTPD error codes

ESP_ERR_HTTPD_HANDLERS_FULL
: All slots for registering URI handlers have been consumed

ESP_ERR_HTTPD_HANDLER_EXISTS
: URI handler with same method and target URI already registered

ESP_ERR_HTTPD_INVALID_REQ
: Invalid request pointer

`ESP_ERR_HTTPD_RESULT_TRUNC`
: Result string truncated

`ESP_ERR_HTTPD_RESP_HDR`
: Response header field larger than supported

`ESP_ERR_HTTPD_RESP_SEND`
: Error occured while sending response packet

`ESP_ERR_HTTPD_ALLOC_MEM`
: Failed to dynamically allocate memory for resource

`ESP_ERR_HTTPD_TASK`
: Failed to launch server task/thread

`HTTPD_RESP_USE_STRLEN`

Type Definitions

`typedef struct` *httpd_req* **`httpd_req_t`**
: HTTP Request Data Structure.

`typedef struct` *httpd_uri* **`httpd_uri_t`**
: Structure for URI handler.

`typedef` int (*`httpd_send_func_t`)(*httpd_handle_t* hd, int sockfd, `const` char *buf, size_t buf_len, int flags)
: Prototype for HTTPDs low-level send function.

  Note User specified send function must handle errors internally, depending upon the set value of errno, and return specific HTTPD_SOCK_ERR_ codes, which will eventually be conveyed as return value of httpd_send() function

  Return

  - Bytes : The number of bytes sent successfully
  - HTTPD_SOCK_ERR_INVALID : Invalid arguments
  - HTTPD_SOCK_ERR_TIMEOUT : Timeout/interrupted while calling socket send()
  - HTTPD_SOCK_ERR_FAIL : Unrecoverable error while calling socket send()

  Parameters

  - `hd`: server instance
  - `sockfd`: session socket file descriptor
  - `buf`: buffer with bytes to send
  - `buf_len`: data size

- `flags`: flags for the send() function

`typedef` int (*`httpd_recv_func_t`)(*httpd_handle_t* hd, int sockfd, char *buf, size_t buf_len, int flags)

Prototype for HTTPDs low-level recv function.

Note User specified recv function must handle errors internally, depending upon the set value of errno, and return specific HTTPD_SOCK_ERR_ codes, which will eventually be conveyed as return value of httpd_req_recv() function

Return

- Bytes : The number of bytes received successfully
- 0 : Buffer length parameter is zero / connection closed by peer
- HTTPD_SOCK_ERR_INVALID : Invalid arguments
- HTTPD_SOCK_ERR_TIMEOUT : Timeout/interrupted while calling socket recv()
- HTTPD_SOCK_ERR_FAIL : Unrecoverable error while calling socket recv()

Parameters

- `hd`: server instance
- `sockfd`: session socket file descriptor
- `buf`: buffer with bytes to send
- `buf_len`: data size
- `flags`: flags for the send() function

`typedef` int (*`httpd_pending_func_t`)(*httpd_handle_t* hd, int sockfd)

Prototype for HTTPDs low-level "get pending bytes" function.

Note User specified pending function must handle errors internally, depending upon the set value of errno, and return specific HTTPD_SOCK_ERR_ codes, which will be handled accordingly in the server task.

Return

- Bytes : The number of bytes waiting to be received
- HTTPD_SOCK_ERR_INVALID : Invalid arguments
- HTTPD_SOCK_ERR_TIMEOUT : Timeout/interrupted while calling socket pending()
- HTTPD_SOCK_ERR_FAIL : Unrecoverable error while calling socket pending()

Parameters

- `hd`: server instance

- `sockfd`: session socket file descriptor

typedef *esp_err_t* (***httpd_err_handler_func_t**)(*httpd_req_t* *req, *httpd_err_code_t* error)

Function prototype for HTTP error handling.

This function is executed upon HTTP errors generated during internal processing of an HTTP request. This is used to override the default behavior on error, which is to send HTTP error response and close the underlying socket.

Note

- If implemented, the server will not automatically send out HTTP error response codes, therefore, httpd_resp_send_err() must be invoked inside this function if user wishes to generate HTTP error responses.
- When invoked, the validity of `uri`, `method`, `content_len` and `user_ctx` fields of the httpd_req_t parameter is not guaranteed as the HTTP request may be partially received/parsed.
- The function must return ESP_OK if underlying socket needs to be kept open. Any other value will ensure that the socket is closed. The return value is ignored when error is of type `HTTPD_500_INTERNAL_SERVER_ERROR` and the socket closed anyway.

Return

- ESP_OK : error handled successful
- ESP_FAIL : failure indicates that the underlying socket needs to be closed

Parameters

- `req`: HTTP request for which the error needs to be handled
- `error`: Error type

typedef void ***httpd_handle_t**

HTTP Server Instance Handle.

Every instance of the server will have a unique handle.

typedef enum http_method **httpd_method_t**

HTTP Method Type wrapper over "enum http_method" available in "http_parser" library.

typedef void (***httpd_free_ctx_fn_t**)(void *ctx)

Prototype for freeing context data (if any)

Parameters

- `ctx`: object to free

typedef *esp_err_t* (*httpd_open_func_t)(*httpd_handle_t* hd, int sockfd)

Function prototype for opening a session.

Called immediately after the socket was opened to set up the send/recv functions and other parameters of the socket.

Return

- ESP_OK : On success
- Any value other than ESP_OK will signal the server to close the socket immediately

Parameters

- `hd`: server instance
- `sockfd`: session socket file descriptor

typedef void (*httpd_close_func_t)(*httpd_handle_t* hd, int sockfd)

Function prototype for closing a session.

Note It' s possible that the socket descriptor is invalid at this point, the function is called for all terminated sessions. Ensure proper handling of return codes.

Parameters

- `hd`: server instance
- `sockfd`: session socket file descriptor

typedef bool (*httpd_uri_match_func_t)(const char *reference_uri, const char *uri_to_match, size_t match_upto)

Function prototype for URI matching.

Return true on match

Parameters

- `reference_uri`: URI/template with respect to which the other URI is matched
- `uri_to_match`: URI/template being matched to the reference URI/template
- `match_upto`: For specifying the actual length of `uri_to_match` up to which the matching algorithm is to be applied (The maximum value is `strlen(uri_to_match)`, independent of the length of `reference_uri`)

typedef struct *httpd_config* httpd_config_t

HTTP Server Configuration Structure.

Note Use HTTPD_DEFAULT_CONFIG() to initialize the configuration to a default value and then modify only those fields that are specifically determined by the use case.

typedef void (*httpd_work_fn_t)(void *arg)

Prototype of the HTTPD work function Please refer to httpd_queue_work() for more details.

Parameters

- arg: The arguments for this work function

Enumerations

enum httpd_err_code_t

Error codes sent as HTTP response in case of errors encountered during processing of an HTTP request.

Values:

HTTPD_500_INTERNAL_SERVER_ERROR = 0

HTTPD_501_METHOD_NOT_IMPLEMENTED

HTTPD_505_VERSION_NOT_SUPPORTED

HTTPD_400_BAD_REQUEST

HTTPD_404_NOT_FOUND

HTTPD_405_METHOD_NOT_ALLOWED

HTTPD_408_REQ_TIMEOUT

HTTPD_411_LENGTH_REQUIRED

HTTPD_414_URI_TOO_LONG

HTTPD_431_REQ_HDR_FIELDS_TOO_LARGE

HTTPD_ERR_CODE_MAX

2.4.6 HTTPS server

Overview

This component is built on top of *esp_http_server*. The HTTPS server takes advantage of hooks and function overrides in the regular HTTP server to provide encryption using OpenSSL.

All documentation for *esp_http_server* applies also to a server you create this way.

Used APIs

The following API of *esp_http_server* should not be used with *esp_https_server*, as they are used internally to handle secure sessions and to maintain internal state:

- "send", "receive" and "pending" function overrides - secure socket handling

  - *httpd_sess_set_send_override()*
  - *httpd_sess_set_recv_override()*
  - *httpd_sess_set_pending_override()*

- "transport context" - both global and session
  - *httpd_sess_get_transport_ctx()* - returns SSL used for the session
  - *httpd_sess_set_transport_ctx()*
  - *httpd_get_global_transport_ctx()* - returns the shared SSL context
  - `httpd_config_t.global_transport_ctx`
  - `httpd_config_t.global_transport_ctx_free_fn`
  - `httpd_config_t.open_fn` - used to set up secure sockets

Everything else can be used without limitations.

Usage

Please see the example protocols/https_server to learn how to set up a secure server.

Basically all you need is to generate a certificate, embed it in the firmware, and provide its pointers and lengths to the start function via the init struct.

The server can be started with or without SSL by changing a flag in the init struct - `httpd_ssl_config.transport_mode`. This could be used e.g. for testing or in trusted environments where you prefer speed over security.

Performance

The initial session setup can take about two seconds, or more with slower clock speeds or more verbose logging. Subsequent requests through the open secure socket are much faster (down to under 100 ms).

API Reference

Header File

- esp_https_server/include/esp_https_server.h

Functions

esp_err_t **httpd_ssl_start**(*httpd_handle_t* **handle*, *httpd_ssl_config_t* **config*)

Create a SSL capable HTTP server (secure mode may be disabled in config)

Return success

Parameters

- `config`: - server config, must not be const. Does not have to stay valid after calling this function.
- `handle`: - storage for the server handle, must be a valid pointer

void **httpd_ssl_stop**(*httpd_handle_t handle*)

Stop the server. Blocks until the server is shut down.

Parameters

- `handle`:

Structures

struct httpd_ssl_config

HTTPS server config struct

Please use HTTPD_SSL_CONFIG_DEFAULT() to initialize it.

Public Members

httpd_config_t **httpd**

Underlying HTTPD server config

Parameters like task stack size and priority can be adjusted here.

const uint8_t ***cacert_pem**

CA certificate

size_t **cacert_len**

CA certificate byte length

const uint8_t ***prvtkey_pem**

Private key

size_t **prvtkey_len**

Private key byte length

httpd_ssl_transport_mode_t **transport_mode**

Transport Mode (default secure)

uint16_t **port_secure**

Port used when transport mode is secure (default 443)

uint16_t **port_insecure**

Port used when transport mode is insecure (default 80)

Macros

`HTTPD_SSL_CONFIG_DEFAULT()`

Default config struct init

(http_server default config had to be copied for customization)

Notes:

- port is set when starting the server, according to 'transport_mode'
- one socket uses ~ 40kB RAM with SSL, we reduce the default socket count to 4
- SSL sockets are usually long-lived, closing LRU prevents pool exhaustion DOS
- Stack size may need adjustments depending on the user application

Type Definitions

`typedef struct` *httpd_ssl_config* `httpd_ssl_config_t`

Enumerations

`enum httpd_ssl_transport_mode_t`

Values:

`HTTPD_SSL_TRANSPORT_SECURE`

`HTTPD_SSL_TRANSPORT_INSECURE`

2.4.7 ICMP Echo

Overview

ICMP (Internet Control Message Protocol) is used for diagnostic or control purposes or generated in response to errors in IP operations. The common network util `ping` is implemented based on the ICMP packets with the type field value of 0, also called `Echo Reply`.

During a ping session, the source host firstly sends out an ICMP echo request packet and wait for an ICMP echo reply with specific times. In this way, it also measures the round-trip time for the messages. After receiving a valid ICMP echo reply, the source host will generate statistics about the IP link layer (e.g. packet loss, elapsed time, etc).

It is common that IoT device needs to check whether a remote server is alive or not. The device should show the warnings to users when it got offline. It can be achieved by creating a ping session and sending/parsing ICMP echo packets periodically.

To make this internal procedure much easier for users, ESP-IDF provides some out-of-box APIs.

Create a new ping session

To create a ping session, you need to fill in the `esp_ping_config_t` configuration structure firstly, specifying target IP address, interval times, and etc. Optionally, you can also register some callback functions with the *esp_ping_callbacks_t*‘ structure.

Example method to create a new ping session and register callbacks:

```
static void test_on_ping_success(esp_ping_handle_t hdl, void *args)
{
    // optionally, get callback arguments
    // const char* str = (const char*) args;
    // printf("%s\r\n", str); // "foo"
    uint8_t ttl;
    uint16_t seqno;
    uint32_t elapsed_time, recv_len;
    ip_addr_t target_addr;
    esp_ping_get_profile(hdl, ESP_PING_PROF_SEQNO, &seqno, sizeof(seqno));
    esp_ping_get_profile(hdl, ESP_PING_PROF_TTL, &ttl, sizeof(ttl));
    esp_ping_get_profile(hdl, ESP_PING_PROF_IPADDR, &target_addr, sizeof(target_addr));
    esp_ping_get_profile(hdl, ESP_PING_PROF_SIZE, &recv_len, sizeof(recv_len));
    esp_ping_get_profile(hdl, ESP_PING_PROF_TIMEGAP, &elapsed_time, sizeof(elapsed_
↪time));
    printf("%d bytes from %s icmp_seq=%d ttl=%d time=%d ms\n",
           recv_len, inet_ntoa(target_addr.u_addr.ip4), seqno, ttl, elapsed_time);
}

static void test_on_ping_timeout(esp_ping_handle_t hdl, void *args)
{
    uint16_t seqno;
    ip_addr_t target_addr;
    esp_ping_get_profile(hdl, ESP_PING_PROF_SEQNO, &seqno, sizeof(seqno));
    esp_ping_get_profile(hdl, ESP_PING_PROF_IPADDR, &target_addr, sizeof(target_addr));
    printf("From %s icmp_seq=%d timeout\n", inet_ntoa(target_addr.u_addr.ip4), seqno);
}

static void test_on_ping_end(esp_ping_handle_t hdl, void *args)
{
    uint32_t transmitted;
    uint32_t received;
    uint32_t total_time_ms;
```

(下页继续)

(续上页)

```
    esp_ping_get_profile(hdl, ESP_PING_PROF_REQUEST, &transmitted, sizeof(transmitted));
    esp_ping_get_profile(hdl, ESP_PING_PROF_REPLY, &received, sizeof(received));
    esp_ping_get_profile(hdl, ESP_PING_PROF_DURATION, &total_time_ms, sizeof(total_time_
↪ms));
    printf("%d packets transmitted, %d received, time %dms\n", transmitted, received,␣
↪total_time_ms);
}

void initialize_ping()
{
    /* convert URL to IP address */
    getaddrinfo("www.espressif.com", NULL, &hint, &res) == 0);
    struct in_addr addr4 = ((struct sockaddr_in *) (res->ai_addr))->sin_addr;
    inet_addr_to_ip4addr(ip_2_ip4(&target_addr), &addr4);
    freeaddrinfo(res);

    esp_ping_config_t ping_config = ESP_PING_DEFAULT_CONFIG();
    ping_config.target_addr = target_addr;          // target IP address
    ping_config.count = ESP_PING_COUNT_INFINITE;    // ping in infinite mode, esp_ping_
↪stop can stop it

    /* set callback functions */
    esp_ping_callbacks_t cbs;
    cbs.on_ping_success = test_on_ping_success;
    cbs.on_ping_timeout = test_on_ping_timeout;
    cbs.on_ping_end = test_on_ping_end;
    cbs.cb_args = "foo";  // arguments that will feed to all callback functions, can be␣
↪NULL
    cbs.cb_args = eth_event_group;

    esp_ping_handle_t ping;
    esp_ping_new_session(&ping_config, &cbs, &ping);
}
```

Start and Stop ping session

You can start and stop ping session with the handle returned by `esp_ping_new_session`. Note that, the ping session won't start automatically after creation. If the ping session is stopped, and restart again, the sequence number in ICMP packets will recount from zero again.

Delete a ping session

If a ping session won't be used any more, you can delete it with `esp_ping_delete_session`. Please make sure the ping session is in stop state (i.e. you have called `esp_ping_stop` before or the ping session has finished all the procedures) when you call this function.

Get runtime statistics

As the example code above, you can call `esp_ping_get_profile` to get different runtime statistics of ping session in the callback function.

Application Example

ICMP echo example: protocols/icmp_echo

API Reference

Header File

- lwip/include/apps/ping/ping_sock.h

Functions

esp_err_t **esp_ping_new_session**(**const** *esp_ping_config_t* **config*, **const** *esp_ping_callbacks_t* **cbs*, *esp_ping_handle_t* **hdl_out*)

Create a ping session.

Return

- ESP_ERR_INVALID_ARG: invalid parameters (e.g. configuration is null, etc)
- ESP_ERR_NO_MEM: out of memory
- ESP_FAIL: other internal error (e.g. socket error)
- ESP_OK: create ping session successfully, user can take the ping handle to do follow-on jobs

Parameters

- `config`: ping configuration
- `cbs`: a bunch of callback functions invoked by internal ping task
- `hdl_out`: handle of ping session

esp_err_t **esp_ping_delete_session**(*esp_ping_handle_t hdl*)

Delete a ping session.

Return

- ESP_ERR_INVALID_ARG: invalid parameters (e.g. ping handle is null, etc)
- ESP_OK: delete ping session successfully

Parameters

- `hdl`: handle of ping session

esp_err_t **esp_ping_start**(*esp_ping_handle_t hdl*)

Start the ping session.

Return

- ESP_ERR_INVALID_ARG: invalid parameters (e.g. ping handle is null, etc)
- ESP_OK: start ping session successfully

Parameters

- `hdl`: handle of ping session

esp_err_t **esp_ping_stop**(*esp_ping_handle_t hdl*)

Stop the ping session.

Return

- ESP_ERR_INVALID_ARG: invalid parameters (e.g. ping handle is null, etc)
- ESP_OK: stop ping session successfully

Parameters

- `hdl`: handle of ping session

esp_err_t **esp_ping_get_profile**(*esp_ping_handle_t hdl*, *esp_ping_profile_t profile*, void **data*, uint32_t *size*)

Get runtime profile of ping session.

Return

- ESP_ERR_INVALID_ARG: invalid parameters (e.g. ping handle is null, etc)
- ESP_ERR_INVALID_SIZE: the actual profile data size doesn't match the "size" parameter
- ESP_OK: get profile successfully

Parameters

- `hdl`: handle of ping session
- `profile`: type of profile
- `data`: profile data

- `size`: profile data size

Structures

struct esp_ping_callbacks_t

Type of "ping" callback functions.

Public Members

void ***cb_args**

arguments for callback functions

void (***on_ping_success**)(*esp_ping_handle_t* hdl, void *args)

Invoked by internal ping thread when received ICMP echo reply packet.

void (***on_ping_timeout**)(*esp_ping_handle_t* hdl, void *args)

Invoked by internal ping thread when receive ICMP echo reply packet timeout.

void (***on_ping_end**)(*esp_ping_handle_t* hdl, void *args)

Invoked by internal ping thread when a ping session is finished.

struct esp_ping_config_t

Type of "ping" configuration.

Public Members

uint32_t **count**

A "ping" session contains count procedures

uint32_t **interval_ms**

Milliseconds between each ping procedure

uint32_t **timeout_ms**

Timeout value (in milliseconds) of each ping procedure

uint32_t **data_size**

Size of the data next to ICMP packet header

uint8_t **tos**

Type of Service, a field specified in the IP header

ip_addr_t **target_addr**

Target IP address, either IPv4 or IPv6

uint32_t **task_stack_size**

Stack size of internal ping task

uint32_t **task_prio**

Priority of internal ping task

Macros

ESP_PING_DEFAULT_CONFIG()

Default ping configuration.

ESP_PING_COUNT_INFINITE

Set ping count to zero will ping target infinitely

Type Definitions

typedef void ***esp_ping_handle_t**

Type of "ping" session handle.

Enumerations

enum esp_ping_profile_t

Profile of ping session.

Values:

ESP_PING_PROF_SEQNO

Sequence number of a ping procedure

ESP_PING_PROF_TTL

Time to live of a ping procedure

ESP_PING_PROF_REQUEST

Number of request packets sent out

ESP_PING_PROF_REPLY

Number of reply packets received

ESP_PING_PROF_IPADDR

IP address of replied target

ESP_PING_PROF_SIZE

Size of received packet

ESP_PING_PROF_TIMEGAP

Elapsed time between request and reply packet

ESP_PING_PROF_DURATION

Elapsed time of the whole ping session

2.4.8 ASIO port

Overview

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

ASIO documentation

Please refer to the original asio documentation at https://think-async.com/Asio/Documentation. Asio also comes with a number of examples which could be find under Documentation/Examples on that web site.

Supported features

ESP platform port currently supports only network asynchronous socket operations; does not support serial port and ssl. Internal asio settings for ESP include

- EXCEPTIONS are enabled in ASIO if enabled in menuconfig
- TYPEID is enabled in ASIO if enabled in menuconfig

Application Example

ESP examples are based on standard asio examples/protocols/asio:

- examples/protocols/asio/udp_echo_server
- examples/protocols/asio/tcp_echo_server
- examples/protocols/asio/chat_client
- examples/protocols/asio/chat_server

Please refer to the specific example README.md for details

2.4.9 ESP-MQTT

Overview

ESP-MQTT is an implementation of MQTT protocol client (MQTT is a lightweight publish/subscribe messaging protocol).

Features

- Supports MQTT over TCP, SSL with mbedtls, MQTT over Websocket, MQTT over Websocket Secure.
- Easy to setup with URI
- Multiple instances (Multiple clients in one application)
- Support subscribing, publishing, authentication, last will messages, keep alive pings and all 3 QoS levels (it should be a fully functional client).

Application Example

- protocols/mqtt/tcp: MQTT over tcp, default port 1883
- protocols/mqtt/ssl: MQTT over tcp, default port 8883
- protocols/mqtt/ssl_psk: MQTT over tcp using pre-shared keys for authentication, default port 8883
- protocols/mqtt/ws: MQTT over Websocket, default port 80
- protocols/mqtt/wss: MQTT over Websocket Secure, default port 443

Configuration

URI

- Curently support `mqtt`, `mqtts`, `ws`, `wss` schemes
- MQTT over TCP samples:
  - `mqtt://mqtt.eclipse.org`: MQTT over TCP, default port 1883:
  - `mqtt://mqtt.eclipse.org:1884` MQTT over TCP, port 1884:
  - `mqtt://username:password@mqtt.eclipse.org:1884` MQTT over TCP, port 1884, with username and password
- MQTT over SSL samples:
  - `mqtts://mqtt.eclipse.org`: MQTT over SSL, port 8883
  - `mqtts://mqtt.eclipse.org:8884`: MQTT over SSL, port 8884
- MQTT over Websocket samples:
  - `ws://mqtt.eclipse.org:80/mqtt`
- MQTT over Websocket Secure samples:
  - `wss://mqtt.eclipse.org:443/mqtt`
- Minimal configurations:

```
const esp_mqtt_client_config_t mqtt_cfg = {
    .uri = "mqtt://mqtt.eclipse.org",
    // .user_context = (void *)your_context
};
esp_mqtt_client_handle_t client = esp_mqtt_client_init(&mqtt_cfg);
esp_mqtt_client_register_event(client, ESP_EVENT_ANY_ID, mqtt_event_handler, client);
esp_mqtt_client_start(client);
```

- Note: By default mqtt client uses event loop library to post related mqtt events (connected, subscribed, published, etc.)

SSL

- Get certificate from server, example: `mqtt.eclipse.org openssl s_client -showcerts -connect mqtt.eclipse.org:8883 </dev/null 2>/dev/null|openssl x509 -outform PEM >mqtt_eclipse_org.pem`
- Check the sample application: `examples/mqtt_ssl`
- Configuration:

```
const esp_mqtt_client_config_t mqtt_cfg = {
    .uri = "mqtts://mqtt.eclipse.org:8883",
    .event_handle = mqtt_event_handler,
    .cert_pem = (const char *)mqtt_eclipse_org_pem_start,
};
```

If the certificate is not null-terminated then `cert_len` should also be set. Other SSL related configuration parameters are:

- `use_global_ca_store`: use the global certificate store to verify server certificate, see `esp-tls.h` for more information
- `client_cert_pem`: pointer to certificate data in PEM or DER format for SSL mutual authentication, default is NULL, not required if mutual authentication is not needed.
- `client_cert_len`: length of the buffer pointed to by client_cert_pem. May be 0 for null-terminated pem.
- `client_key_pem`: pointer to private key data in PEM or DER format for SSL mutual authentication, default is NULL, not required if mutual authentication is not needed.
- `client_key_len`: length of the buffer pointed to by client_key_pem. May be 0 for null-terminated pem.

- `psk_hint_key`: pointer to PSK struct defined in esp_tls.h to enable PSK authentication (as alternative to certificate verification). If not NULL and server/client certificates are NULL, PSK is enabled
- `alpn_protos`: NULL-terminated list of protocols to be used for ALPN.

Last Will and Testament

MQTT allows for a last will and testament (LWT) message to notify other clients when a client ungracefully disconnects. This is configured by the following fields in the `esp_mqtt_client_config_t`-struct.

- `lwt_topic`: pointer to the LWT message topic
- `lwt_msg`: pointer to the LWT message
- `lwt_msg_len`: length of the LWT message, required if `lwt_msg` is not null-terminated
- `lwt_qos`: quality of service for the LWT message
- `lwt_retain`: specifies the retain flag of the LWT message

Other Configuration Parameters

- `disable_clean_session`: determines the clean session flag for the connect message, defaults to a clean session
- `keepalive`: determines how many seconds the client will wait for a ping response before disconnecting, default is 120 seconds.
- `disable_auto_reconnect`: enable to stop the client from reconnecting to server after errors or disconnects
- `user_context`: custom context that will be passed to the event handler
- `task_prio`: MQTT task priority, defaults to 5
- `task_stack`: MQTT task stack size, defaults to 6144 bytes, setting this will override setting from menuconfig
- `buffer_size`: size of MQTT send/receive buffer, default is 1024 bytes
- `username`: pointer to the username used for connecting to the broker
- `password`: pointer to the password used for connecting to the broker
- `client_id`: pointer to the client id, defaults to `ESP32_%CHIPID%` where %CHIPID% are the last 3 bytes of MAC address in hex format
- `host`: MQTT broker domain (ipv4 as string), setting the uri will override this
- `port`: MQTT broker port, specifying the port in the uri will override this
- `transport`: sets the transport protocol, setting the uri will override this

- `refresh_connection_after_ms`: refresh connection after this value (in milliseconds)
- `event_handle`: handle for MQTT events as a callback in legacy mode
- `event_loop_handle`: handle for MQTT event loop library

For more options on `esp_mqtt_client_config_t`, please refer to API reference below

Change settings in Project Configuration Menu

The settings for MQTT can be found using `idf.py menuconfig`, under Component config -> ESP-MQTT Configuration

The following settings are available:

- *CONFIG_MQTT_PROTOCOL_311*: Enables 3.1.1 version of MQTT protocol
- *CONFIG_MQTT_TRANSPORT_SSL*, *CONFIG_MQTT_TRANSPORT_WEBSOCKET*: Enables specific MQTT transport layer, such as SSL, WEBSOCKET, WEBSOCKET_SECURE
- *CONFIG_MQTT_CUSTOM_OUTBOX*: Disables default implementation of mqtt_outbox, so a specific implementaion can be supplied

Events

The following events may be posted by the MQTT client:

- `MQTT_EVENT_BEFORE_CONNECT`: The client is initialized and about to start connecting to the broker.
- `MQTT_EVENT_CONNECTED`: The client has successfully established a connection to the broker. The client is now ready to send and receive data.
- `MQTT_EVENT_DISCONNECTED`: The client has aborted the connection due to being unable to read or write data, e.g. because the server is unavailable.
- `MQTT_EVENT_SUBSCRIBED`: The broker has acknowledged the client' s subscribe request. The event data will contain the message ID of the subscribe message.
- `MQTT_EVENT_UNSUBSCRIBED`: The broker has acknowledged the client' s unsubscribe request. The event data will contain the message ID of the unsubscribe message.
- `MQTT_EVENT_PUBLISHED`: The broker has acknowledged the client' s publish message. This will only be posted for Quality of Service level 1 and 2, as level 0 does not use acknowledgements. The event data will contain the message ID of the publish message.
- `MQTT_EVENT_DATA`: The client has received a publish message. The event data contains: message ID, name of the topic it was published to, received data and its length. For data that exceeds the internal buffer multiple *MQTT_EVENT_DATA* will be posted and *current_data_offset* and *total_data_len* from event data updated to keep track of the fragmented message.

- `MQTT_EVENT_ERROR`: The client has encountered an error. *esp_mqtt_error_type_t* from *error_handle* in the event data can be used to further determine the type of the error. The type of error will determine which parts of the *error_handle* struct is filled.

API Reference

Header File

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

Functions

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

Creates mqtt client handle based on the configuration.

Return mqtt_client_handle if successfully created, NULL on error

Parameters

- `config`: mqtt configuration structure

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

Sets mqtt connection URI. This API is usually used to overrides the URI configured in esp_mqtt_client_init.

Return ESP_FAIL if URI parse error, ESP_OK on success

Parameters

- `client`: mqtt client handle
- `uri`:

esp_err_t **esp_mqtt_client_start**(*esp_mqtt_client_handle_t* *client*)

Starts mqtt client with already created client handle.

Return ESP_OK on success ESP_ERR_INVALID_ARG on wrong initialization ESP_FAIL on other error

Parameters

- `client`: mqtt client handle

esp_err_t **esp_mqtt_client_reconnect**(*esp_mqtt_client_handle_t* *client*)

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

Return ESP_OK on success ESP_FAIL if client is in invalid state

Parameters

- `client`: mqtt client handle

esp_err_t **esp_mqtt_client_disconnect**(*esp_mqtt_client_handle_t client*)

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

Return ESP_OK on success

Parameters

- `client`: mqtt client handle

esp_err_t **esp_mqtt_client_stop**(*esp_mqtt_client_handle_t client*)

Stops mqtt client tasks.

Return ESP_OK on success ESP_FAIL if client is in invalid state

Parameters

- `client`: mqtt client handle

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

Subscribe the client to defined topic with defined qos.

Notes:

- Client must be connected to send subscribe message
- This API is could be executed from a user task or from a mqtt event callback i.e. internal mqtt task (API is protected by internal mutex, so it might block if a longer data receive operation is in progress.

Return message_id of the subscribe message on success -1 on failure

Parameters

- `client`: mqtt client handle
- `topic`:
- `qos`:

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

Unsubscribe the client from defined topic.

Notes:

- Client must be connected to send unsubscribe message
- It is thread safe, please refer to `esp_mqtt_client_subscribe` for details

Return message_id of the subscribe message on success -1 on failure

Parameters

- `client`: mqtt client handle
- `topic`:

int **esp_mqtt_client_publish**(*esp_mqtt_client_handle_t client*, **const** char **topic*, **const** char **data*, int *len*, int *qos*, int *retain*)

Client to send a publish message to the broker.

Notes:

- This API might block for several seconds, either due to network timeout (10s) or if publishing payloads longer than internal buffer (due to message fragmentation)
- Client doesn' t have to be connected to send publish message (although it would drop all qos=0 messages, qos>1 messages would be enqueued)
- It is thread safe, please refer to `esp_mqtt_client_subscribe` for details

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

Parameters

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

esp_err_t **esp_mqtt_client_destroy**(*esp_mqtt_client_handle_t client*)

Destroys the client handle.

Return ESP_OK

Parameters

- `client`: mqtt client handle

esp_err_t **esp_mqtt_set_config**(*esp_mqtt_client_handle_t client*, **const** *esp_mqtt_client_config_t *config*)

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

Return ESP_ERR_NO_MEM if failed to allocate ESP_OK on success

Parameters

- `client`: mqtt client handle
- `config`: mqtt configuration structure

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

Registers mqtt event.

Return ESP_ERR_NO_MEM if failed to allocate ESP_OK on success

Parameters

- `client`: mqtt client handle
- `event`: event type
- `event_handler`: handler callback
- `event_handler_arg`: handlers context

Structures

struct esp_mqtt_error_codes

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

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

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

| error_type | related member variables | note | | MQTT_ERROR_TYPE_ESP_TLS | esp_tls_last_esp_err, esp_tls_stack_err, esp_tls_cert_verify_flags | Error reported from esp-tls | | MQTT_ERROR_TYPE_CONNECTION_REFUSED | connect_return_code | Internal error reported from MQTT broker on connection |

Public Members

esp_err_t **esp_tls_last_esp_err**

last esp_err code reported from esp-tls component

int **esp_tls_stack_err**

tls specific error code reported from underlying tls stack

int **esp_tls_cert_verify_flags**

tls flags reported from underlying tls stack during certificate verification

esp_mqtt_error_type_t **error_type**

error type referring to the source of the error

esp_mqtt_connect_return_code_t **connect_return_code**

connection refused error code reported from MQTT broker on connection

struct esp_mqtt_event_t

MQTT event configuration structure

Public Members

esp_mqtt_event_id_t **event_id**

MQTT event type

esp_mqtt_client_handle_t **client**

MQTT client handle for this event

void ***user_context**

User context passed from MQTT client config

char ***data**

Data associated with this event

int **data_len**

Length of the data for this event

int **total_data_len**

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

int **current_data_offset**

Actual offset for the data associated with this event

char ***topic**

Topic associated with this event

int **topic_len**

Length of the topic for this event associated with this event

int **msg_id**

MQTT messaged id of message

int **session_present**

MQTT session_present flag for connection event

esp_mqtt_error_codes_t ***error_handle**

esp-mqtt error handle including esp-tls errors as well as internal mqtt errors

struct esp_mqtt_client_config_t

MQTT client configuration structure

Public Members

mqtt_event_callback_t **event_handle**
handle for MQTT events as a callback in legacy mode

esp_event_loop_handle_t **event_loop_handle**
handle for MQTT event loop library

const char ***host**
MQTT server domain (ipv4 as string)

const char ***uri**
Complete MQTT broker URI

uint32_t **port**
MQTT server port

const char ***client_id**
default client id is `ESP32_CHIPID%` where CHIPID% are last 3 bytes of MAC address in hex format

const char ***username**
MQTT username

const char ***password**
MQTT password

const char ***lwt_topic**
LWT (Last Will and Testament) message topic (NULL by default)

const char ***lwt_msg**
LWT message (NULL by default)

int **lwt_qos**
LWT message qos

int **lwt_retain**
LWT retained message flag

int **lwt_msg_len**
LWT message length

int **disable_clean_session**
mqtt clean session, default clean_session is true

int **keepalive**
mqtt keepalive, default is 120 seconds

bool **disable_auto_reconnect**
this mqtt client will reconnect to server (when errors/disconnect). Set disable_auto_reconnect=true to disable

void *user_context
: pass user context to this option, then can receive that context in event->user_context

int task_prio
: MQTT task priority, default is 5, can be changed in make menuconfig

int task_stack
: MQTT task stack size, default is 6144 bytes, can be changed in make menuconfig

int buffer_size
: size of MQTT send/receive buffer, default is 1024 (only receive buffer size if out_buffer_size defined)

const char *cert_pem
: Pointer to certificate data in PEM or DER format for server verify (with SSL), default is NULL, not required to verify the server. PEM-format must have a terminating NULL-character. DER-format requires the length to be passed in cert_len.

size_t cert_len
: Length of the buffer pointed to by cert_pem. May be 0 for null-terminated pem

const char *client_cert_pem
: Pointer to certificate data in PEM or DER format for SSL mutual authentication, default is NULL, not required if mutual authentication is not needed. If it is not NULL, also client_key_pem has to be provided. PEM-format must have a terminating NULL-character. DER-format requires the length to be passed in client_cert_len.

size_t client_cert_len
: Length of the buffer pointed to by client_cert_pem. May be 0 for null-terminated pem

const char *client_key_pem
: Pointer to private key data in PEM or DER format for SSL mutual authentication, default is NULL, not required if mutual authentication is not needed. If it is not NULL, also client_cert_pem has to be provided. PEM-format must have a terminating NULL-character. DER-format requires the length to be passed in client_key_len

size_t client_key_len
: Length of the buffer pointed to by client_key_pem. May be 0 for null-terminated pem

esp_mqtt_transport_t transport
: overrides URI transport

int refresh_connection_after_ms
: Refresh connection after this value (in milliseconds)

const struct *psk_key_hint* *psk_hint_key
: Pointer to PSK struct defined in esp_tls.h to enable PSK authentication (as alternative to certificate verification). If not NULL and server/client certificates are NULL, PSK is enabled

bool **use_global_ca_store**

Use a global ca_store for all the connections in which this bool is set.

int **reconnect_timeout_ms**

Reconnect to the broker after this value in miliseconds if auto reconnect is not disabled

const char ****alpn_protos**

NULL-terminated list of supported application protocols to be used for ALPN

const char ***clientkey_password**

Client key decryption password string

int **clientkey_password_len**

String length of the password pointed to by clientkey_password

esp_mqtt_protocol_ver_t **protocol_ver**

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

int **out_buffer_size**

size of MQTT output buffer. If not defined, both output and input buffers have the same size defined as **buffer_size**

Type Definitions

typedef struct esp_mqtt_client ***esp_mqtt_client_handle_t**

typedef struct *esp_mqtt_error_codes* **esp_mqtt_error_codes_t**

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

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

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

| error_type | related member variables | note | | MQTT_ERROR_TYPE_ESP_TLS | esp_tls_last_esp_err, esp_tls_stack_err, esp_tls_cert_verify_flags | Error reported from esp-tls | | MQTT_ERROR_TYPE_CONNECTION_REFUSED | connect_return_code | Internal error reported from MQTT broker on connection |

typedef *esp_mqtt_event_t* ***esp_mqtt_event_handle_t**

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

Enumerations

enum esp_mqtt_event_id_t

MQTT event types.

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

- `user_context` - user data from *esp_mqtt_client_config_t*
- `client` - mqtt client handle
- various other data depending on event type

Values:

`MQTT_EVENT_ANY` = -1

`MQTT_EVENT_ERROR` = 0

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

`MQTT_EVENT_CONNECTED`

connected event, additional context: session_present flag

`MQTT_EVENT_DISCONNECTED`

disconnected event

`MQTT_EVENT_SUBSCRIBED`

subscribed event, additional context: msg_id

`MQTT_EVENT_UNSUBSCRIBED`

unsubscribed event

`MQTT_EVENT_PUBLISHED`

published event, additional context: msg_id

`MQTT_EVENT_DATA`

data event, additional context:

- msg_id message id
- topic pointer to the received topic
- topic_len length of the topic
- data pointer to the received data
- data_len length of the data for this event
- current_data_offset offset of the current data for this event
- total_data_len total length of the data received Note: Multiple MQTT_EVENT_DATA could be fired for one message, if it is longer than internal buffer. In that case only first

event contains topic pointer and length, other contain data only with current data length and current data offset updating.

`MQTT_EVENT_BEFORE_CONNECT`
The event occurs before connecting

enum esp_mqtt_connect_return_code_t
MQTT connection error codes propagated via ERROR event

Values:

`MQTT_CONNECTION_ACCEPTED` = 0
Connection accepted

`MQTT_CONNECTION_REFUSE_PROTOCOL`
MQTT connection refused reason: Wrong protocol

`MQTT_CONNECTION_REFUSE_ID_REJECTED`
MQTT connection refused reason: ID rejected

`MQTT_CONNECTION_REFUSE_SERVER_UNAVAILABLE`
MQTT connection refused reason: Server unavailable

`MQTT_CONNECTION_REFUSE_BAD_USERNAME`
MQTT connection refused reason: Wrong user

`MQTT_CONNECTION_REFUSE_NOT_AUTHORIZED`
MQTT connection refused reason: Wrong username or password

enum esp_mqtt_error_type_t
MQTT connection error codes propagated via ERROR event

Values:

`MQTT_ERROR_TYPE_NONE` = 0

`MQTT_ERROR_TYPE_ESP_TLS`

`MQTT_ERROR_TYPE_CONNECTION_REFUSED`

enum esp_mqtt_transport_t
Values:

`MQTT_TRANSPORT_UNKNOWN` = 0x0

`MQTT_TRANSPORT_OVER_TCP`
MQTT over TCP, using scheme: `mqtt`

`MQTT_TRANSPORT_OVER_SSL`
MQTT over SSL, using scheme: `mqtts`

`MQTT_TRANSPORT_OVER_WS`
MQTT over Websocket, using scheme:: `ws`

`MQTT_TRANSPORT_OVER_WSS`
MQTT over Websocket Secure, using scheme: `wss`

`enum esp_mqtt_protocol_ver_t`
MQTT protocol version used for connection

Values:

`MQTT_PROTOCOL_UNDEFINED` = 0

`MQTT_PROTOCOL_V_3_1`

`MQTT_PROTOCOL_V_3_1_1`

2.4.10 ESP-Modbus

Overview

The Modbus serial communication protocol is de facto standard protocol widely used to connect industrial electronic devices. Modbus allows communication among many devices connected to the same network, for example, a system that measures temperature and humidity and communicates the results to a computer. The Modbus protocol uses several types of data: Holding Registers, Input Registers, Coils (single bit output), Discrete Inputs. Versions of the Modbus protocol exist for serial port and for Ethernet and other protocols that support the Internet protocol suite. There are many variants of Modbus protocols, some of them are:

- `Modbus RTU` —This is used in serial communication and makes use of a compact, binary representation of the data for protocol communication. The RTU format follows the commands/data with a cyclic redundancy check checksum as an error check mechanism to ensure the reliability of data. Modbus RTU is the most common implementation available for Modbus. A Modbus RTU message must be transmitted continuously without inter-character hesitations. Modbus messages are framed (separated) by idle (silent) periods. The RS-485 interface communication is usually used for this type.
- `Modbus ASCII` —This is used in serial communication and makes use of ASCII characters for protocol communication. The ASCII format uses a longitudinal redundancy check checksum. Modbus ASCII messages are framed by leading colon (":") and trailing newline (CR/LF).
- `Modbus TCP/IP or Modbus TCP` —This is a Modbus variant used for communications over TCP/IP networks, connecting over port 502. It does not require a checksum calculation, as lower layers already provide checksum protection.

Modbus common interface API overview

The API functions below provide common functionality to setup Modbus stack for slave and master implementation accordingly. ISP-IDF supports Modbus serial slave and master protocol stacks and provides modbus_controller interface API to interact with user application.

esp_err_t **mbc_slave_init**(mb_port_type_t *port_type*, void *******handler*)

Initialize Modbus controller and stack.

Return

- ESP_OK Success
- ESP_ERR_NO_MEM Parameter error

Parameters

- `handler`: handler(pointer) to master data structure
- `port_type`: type of stack

esp_err_t **mbc_master_init**(mb_port_type_t *port_type*, void *******handler*)

Initialize Modbus controller and stack.

Return

- ESP_OK Success
- ESP_ERR_NO_MEM Parameter error

Parameters

- `handler`: handler(pointer) to master data structure
- `port_type`: the type of port

The function initializes the Modbus controller interface and its active context (tasks, RTOS objects and other resources).

esp_err_t **mbc_slave_setup**(void ******comm_info*)

Set Modbus communication parameters for the controller.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Incorrect parameter data

Parameters

- `comm_info`: Communication parameters structure.

esp_err_t **mbc_master_setup**(void ******comm_info*)

Set Modbus communication parameters for the controller.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Incorrect parameter data

Parameters

- `comm_info`: Communication parameters structure.

The function is used to setup communication parameters of the Modbus stack. See the Modbus controller API documentation for more information.

mbc_slave_set_descriptor(): Initialization of slave descriptor.

mbc_master_set_descriptor(): Initialization of master descriptor.

The Modbus stack uses parameter description tables (descriptors) for communication. These are different for master and slave implementation of stack and should be assigned by the API call before start of communication.

esp_err_t **mbc_slave_start**(void)

Start Modbus communication stack.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Modbus stack start error

esp_err_t **mbc_master_start**(void)

Start Modbus communication stack.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Modbus stack start error

Modbus controller start function. Starts stack and interface and allows communication.

esp_err_t **mbc_slave_destroy**(void)

Destroy Modbus controller and stack.

Return

- ESP_OK Success
- ESP_ERR_INVALID_STATE Parameter error

esp_err_t **mbc_master_destroy**(void)

Destroy Modbus controller and stack.

Return

- ESP_OK Success
- ESP_ERR_INVALID_STATE Parameter error

This function stops Modbus communication stack and destroys controller interface.

There are some configurable parameters of modbus_controller interface and Modbus stack that can be configured using KConfig values in "Modbus configuration" menu. The most important option in KConfig menu is "Selection of Modbus stack support mode" that allows to select master or slave stack for implementation. See the examples for more information about how to use these API functions.

Modbus serial slave interface API overview

The slave stack requires the user defined structures which represent Modbus parameters accessed by stack. These structures should be prepared by user and be assigned to the modbus_controller interface using *mbc_slave_set_descriptor()* API call before start of communication. The interface API functions below are used for Modbus slave application:

esp_err_t **mbc_slave_set_descriptor**(mb_register_area_descriptor_t *descr_data*)

Set Modbus area descriptor.

Return

- ESP_OK: The appropriate descriptor is set
- ESP_ERR_INVALID_ARG: The argument is incorrect

Parameters

- `descr_data`: Modbus registers area descriptor structure

The function initializes Modbus communication descriptors for each type of Modbus register area (Holding Registers, Input Registers, Coils (single bit output), Discrete Inputs). Once areas are initialized and the *mbc_slave_start()* API is called the Modbus stack can access the data in user data structures by request from master. See the `mb_register_area_descriptor_t` and example for more information.

mb_event_group_t **mbc_slave_check_event**(mb_event_group_t *group*)

Wait for specific event on parameter change.

Return

- mb_event_group_t event bits triggered

Parameters

- `group`: Group event bit mask to wait for change

The blocking call to function waits for event specified in the input parameter as event mask. Once master access the parameter and event mask matches the parameter the application task will be unblocked and function will return ESP_OK. See the `mb_event_group_t` for more information about Modbus event masks.

esp_err_t **mbc_slave_get_param_info**(mb_param_info_t *_reg_info_, uint32_t _timeout_)

Get parameter information.

Return

- ESP_OK Success
- ESP_ERR_TIMEOUT Can not get data from parameter queue or queue overflow

Parameters

- `reg_info`: parameter info structure
- `timeout`: Timeout in milliseconds to read information from parameter queue

The function gets information about accessed parameters from modbus controller event queue. The KConfig 'CONFIG_FMB_CONTROLLER_NOTIFY_QUEUE_SIZE' key can be used to configure the notification queue size. The timeout parameter allows to specify timeout for waiting notification. The `mb_param_info_t` structure contain information about accessed parameter.

Modbus serial master interface API overview

The Modbus master implementation requires parameter description table be defined before start of stack. This table describes characteristics (physical parameters like temperature, humidity, etc.) and links them to Modbus registers in specific slave device in the Modbus segment. The table has to be assigned to the modbus_controller interface using *mbc_master_set_descriptor()* API call before start of communication.

Below are the interface API functions that are used to setup and use Modbus master stack from user application and can be executed in next order:

esp_err_t **mbc_master_set_descriptor**(**const** mb_parameter_descriptor_t *_descriptor_, **const** uint16_t _num_elements_)

Assign parameter description table for Modbus controller interface.

Return

- esp_err_t ESP_OK - set descriptor successfully
- esp_err_t ESP_ERR_INVALID_ARG - invalid argument in function call

Parameters

- `descriptor`: pointer to parameter description table
- `num_elements`: number of elements in the table

Assigns parameter description table for Modbus controller interface. The table has to be prepared by user according to particular

esp_err_t **mbc_master_send_request**(mb_param_request_t *request, void *data_ptr)

Send data request as defined in parameter request, waits response from slave and returns status of command execution. This function provides standard way for read/write access to Modbus devices in the network.

Return

- esp_err_t ESP_OK - request was successful
- esp_err_t ESP_ERR_INVALID_ARG - invalid argument of function
- esp_err_t ESP_ERR_INVALID_RESPONSE - an invalid response from slave
- esp_err_t ESP_ERR_TIMEOUT - operation timeout or no response from slave
- esp_err_t ESP_ERR_NOT_SUPPORTED - the request command is not supported by slave
- esp_err_t ESP_FAIL - slave returned an exception or other failure

Parameters

- `request`: pointer to request structure of type mb_param_request_t
- `data_ptr`: pointer to data buffer to send or received data (dependent of command field in request)

This function sends data request as defined in parameter request, waits response from corresponded slave and returns status of command execution. This function provides a standard way for read/write access to Modbus devices in the network.

esp_err_t **mbc_master_get_cid_info**(uint16_t *cid*, **const** mb_parameter_descriptor_t **param_info)

Get information about supported characteristic defined as cid. Uses parameter description table to get this information. The function will check if characteristic defined as a cid parameter is supported and returns its description in param_info. Returns ESP_ERR_NOT_FOUND if characteristic is not supported.

Return

- esp_err_t ESP_OK - request was successful and buffer contains the supported characteristic name
- esp_err_t ESP_ERR_INVALID_ARG - invalid argument of function
- esp_err_t ESP_ERR_NOT_FOUND - the characteristic (cid) not found
- esp_err_t ESP_FAIL - unknown error during lookup table processing

Parameters

- `cid`: characteristic id
- `param_info`: pointer to pointer of characteristic data.

The function gets information about supported characteristic defined as cid. It will check if characteristic is supported and returns its description.

esp_err_t **mbc_master_get_parameter**(uint16_t *cid*, char **name*, uint8_t **value*, uint8_t **type*)

Read parameter from modbus slave device whose name is defined by name and has cid. The additional data for request is taken from parameter description (lookup) table.

Return

- esp_err_t ESP_OK - request was successful and value buffer contains representation of actual parameter data from slave
- esp_err_t ESP_ERR_INVALID_ARG - invalid argument of function
- esp_err_t ESP_ERR_INVALID_RESPONSE - an invalid response from slave
- esp_err_t ESP_ERR_INVALID_STATE - invalid state during data processing or allocation failure
- esp_err_t ESP_ERR_TIMEOUT - operation timed out and no response from slave
- esp_err_t ESP_ERR_NOT_SUPPORTED - the request command is not supported by slave
- esp_err_t ESP_ERR_NOT_FOUND - the parameter is not found in the parameter description table
- esp_err_t ESP_FAIL - slave returned an exception or other failure

Parameters

- `cid`: id of the characteristic for parameter
- `name`: pointer into string name (key) of parameter (null terminated)
- `value`: pointer to data buffer of parameter
- `type`: parameter type associated with the name returned from parameter description table.

The function reads data of characteristic defined in parameters from Modbus slave device and returns its data. The additional data for request is taken from parameter description table.

esp_err_t **mbc_master_set_parameter**(uint16_t *cid*, char **name*, uint8_t **value*, uint8_t **type*)

Set characteristic' s value defined as a name and cid parameter. The additional data for cid parameter request is taken from master parameter lookup table.

Return

- esp_err_t ESP_OK - request was successful and value was saved in the slave device registers
- esp_err_t ESP_ERR_INVALID_ARG - invalid argument of function
- esp_err_t ESP_ERR_INVALID_RESPONSE - an invalid response from slave during processing of parameter

- esp_err_t ESP_ERR_INVALID_STATE - invalid state during data processing or allocation failure
- esp_err_t ESP_ERR_TIMEOUT - operation timed out and no response from slave
- esp_err_t ESP_ERR_NOT_SUPPORTED - the request command is not supported by slave
- esp_err_t ESP_FAIL - slave returned an exception or other failure

Parameters

- `cid`: id of the characteristic for parameter
- `name`: pointer into string name (key) of parameter (null terminated)
- `value`: pointer to data buffer of parameter (actual representation of json value field in binary form)
- `type`: pointer to parameter type associated with the name returned from parameter lookup table.

The function writes characteristic' s value defined as a name and cid parameter in corresponded slave device. The additional data for parameter request is taken from master parameter description table.

Application Example

The examples below use the FreeModbus library port for serial slave and master implementation accordingly. The selection of stack is performed through KConfig menu "Selection of Modbus stack support mode" and related configuration keys.

protocols/modbus/serial/mb_slave

protocols/modbus/serial/mb_master

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

2.4.11 ESP Local Control

Overview

ESP Local Control (**esp_local_ctrl**) component in ESP-IDF provides capability to control an ESP device over Wi-Fi + HTTPS or BLE. It provides access to application defined **properties** that are available for reading / writing via a set of configurable handlers.

Initialization of the **esp_local_ctrl** service over BLE transport is performed as follows:

```
esp_local_ctrl_config_t config = {
    .transport = ESP_LOCAL_CTRL_TRANSPORT_BLE,
    .transport_config = {
```

(下页继续)

(续上页)

```
        .ble = & (protocomm_ble_config_t) {
            .device_name  = SERVICE_NAME,
            .service_uuid = {
                /* LSB <---------------------------------------
                * ---------------------------------------> MSB */
                0x21, 0xd5, 0x3b, 0x8d, 0xbd, 0x75, 0x68, 0x8a,
                0xb4, 0x42, 0xeb, 0x31, 0x4a, 0x1e, 0x98, 0x3d
            }
        }
    },
    .handlers = {
        /* User defined handler functions */
        .get_prop_values = get_property_values,
        .set_prop_values = set_property_values,
        .usr_ctx         = NULL,
        .usr_ctx_free_fn = NULL
    },
    /* Maximum number of properties that may be set */
    .max_properties = 10
};

/* Start esp_local_ctrl service */
ESP_ERROR_CHECK(esp_local_ctrl_start(&config));
```

Similarly for HTTPS transport:

```
/* Set the configuration */
httpd_ssl_config_t https_conf = HTTPD_SSL_CONFIG_DEFAULT();

/* Load server certificate */
extern const unsigned char cacert_pem_start[] asm("_binary_cacert_pem_start");
extern const unsigned char cacert_pem_end[]   asm("_binary_cacert_pem_end");
https_conf.cacert_pem = cacert_pem_start;
https_conf.cacert_len = cacert_pem_end - cacert_pem_start;

/* Load server private key */
extern const unsigned char prvtkey_pem_start[] asm("_binary_prvtkey_pem_start
↪");
extern const unsigned char prvtkey_pem_end[]   asm("_binary_prvtkey_pem_end");
https_conf.prvtkey_pem = prvtkey_pem_start;
```

(下页继续)

(续上页)

```
https_conf.prvtkey_len = prvtkey_pem_end - prvtkey_pem_start;

esp_local_ctrl_config_t config = {
    .transport = ESP_LOCAL_CTRL_TRANSPORT_HTTPD,
    .transport_config = {
        .httpd = &https_conf
    },
    .handlers = {
        /* User defined handler functions */
        .get_prop_values = get_property_values,
        .set_prop_values = set_property_values,
        .usr_ctx         = NULL,
        .usr_ctx_free_fn = NULL
    },
    /* Maximum number of properties that may be set */
    .max_properties = 10
};

/* Start esp_local_ctrl service */
ESP_ERROR_CHECK(esp_local_ctrl_start(&config));
```

Creating a property

Now that we know how to start the **esp_local_ctrl** service, let' s add a property to it. Each property must have a unique *name* (string), a *type* (e.g. enum), *flags* (bit fields) and *size*.

The *size* is to be kept 0, if we want our property value to be of variable length (e.g. if its a string or bytestream). For fixed length property value data-types, like int, float, etc., setting the *size* field to the right value, helps **esp_local_ctrl** to perform internal checks on arguments received with write requests.

The interpretation of *type* and *flags* fields is totally upto the application, hence they may be used as enumerations, bitfields, or even simple integers. One way is to use *type* values to classify properties, while *flags* to specify characteristics of a property.

Here is an example property which is to function as a timestamp. It is assumed that the application defines *TYPE_TIMESTAMP* and *READONLY*, which are used for setting the *type* and *flags* fields here.

```
/* Create a timestamp property */
esp_local_ctrl_prop_t timestamp = {
    .name        = "timestamp",
    .type        = TYPE_TIMESTAMP,
```

(下页继续)

(续上页)

```
    .size         = sizeof(int32_t),
    .flags        = READONLY,
    .ctx          = func_get_time,
    .ctx_free_fn = NULL
};

/* Now register the property */
esp_local_ctrl_add_property(&timestamp);
```

Also notice that there is a ctx field, which is set to point to some custom *func_get_time()*. This can be used inside the property get / set handlers to retrieve timestamp.

Here is an example of *get_prop_values()* handler, which is used for retrieving the timestamp.

```
static esp_err_t get_property_values(size_t props_count,
                                     const esp_local_ctrl_prop_t *props,
                                     esp_local_ctrl_prop_val_t *prop_values,
                                     void *usr_ctx)
{
    for (uint32_t i = 0; i < props_count; i++) {
        ESP_LOGI(TAG, "Reading %s", props[i].name);
        if (props[i].type == TYPE_TIMESTAMP) {
            /* Obtain the timer function from ctx */
            int32_t (*func_get_time)(void) = props[i].ctx;

            /* Use static variable for saving the value.
             * This is essential because the value has to be
             * valid even after this function returns.
             * Alternative is to use dynamic allocation
             * and set the free_fn field */
            static int32_t ts = func_get_time();
            prop_values[i].data = &ts;
        }
    }
    return ESP_OK;
}
```

Here is an example of *set_prop_values()* handler. Notice how we restrict from writing to read-only properties.

```
static esp_err_t set_property_values(size_t props_count,
                                     const esp_local_ctrl_prop_t *props,
```

(下页继续)

(续上页)

```
                                    const esp_local_ctrl_prop_val_t *prop_
↪values,
                                    void *usr_ctx)
{
    for (uint32_t i = 0; i < props_count; i++) {
        if (props[i].flags & READONLY) {
            ESP_LOGE(TAG, "Cannot write to read-only property %s", props[i].
↪name);
            return ESP_ERR_INVALID_ARG;
        } else {
            ESP_LOGI(TAG, "Setting %s", props[i].name);

            /* For keeping it simple, lets only log the incoming data */
            ESP_LOG_BUFFER_HEX_LEVEL(TAG, prop_values[i].data,
                                    prop_values[i].size, ESP_LOG_INFO);
        }
    }
    return ESP_OK;
}
```

For complete example see protocols/esp_local_ctrl

Client Side Implementation

The client side implementation will have establish a protocomm session with the device first, over the supported mode of transport, and then send and receive protobuf messages understood by the **esp_local_ctrl** service. The service will translate these messages into requests and then call the appropriate handlers (set / get). Then, the generated response for each handler is again packed into a protobuf message and transmitted back to the client.

See below the various protobuf messages understood by the **esp_local_ctrl** service:

1. *get_prop_count* : This should simply return the total number of properties supported by the service
2. *get_prop_values* : This accepts an array of indices and should return the information (name, type, flags) and values of the properties corresponding to those indices
3. *set_prop_values* : This accepts an array of indices and an array of new values, which are used for setting the values of the properties corresponding to the indices

Note that indices may or may not be the same for a property, across multiple sessions. Therefore, the client must only use the names of the properties to uniquely identify them. So, every time a new session is established, the client should first call *get_prop_count* and then *get_prop_values*, hence form an index to name mapping for all properties. Now when calling *set_prop_values* for a set of properties, it must first

convert the names to indexes, using the created mapping. As emphasized earlier, the client must refresh the index to name mapping every time a new session is established with the same device.

The various protocomm endpoints provided by **esp_local_ctrl** are listed below:

表 4: Endpoints provided by ESP Local Control

Endpoint Name (BLE + GATT Server)	URI (HTTPS Server + mDNS)	Description
esp_local_ctrl/version	https://<mdns-hostname>.local/esp_local_ctrl/version	Endpoint used for retrieving version string
esp_local_ctrl/control	https://<mdns-hostname>.local/esp_local_ctrl/control	Endpoint used for sending / receiving control messages

API Reference

Header File

- esp_local_ctrl/include/esp_local_ctrl.h

Functions

`const` *esp_local_ctrl_transport_t* *`esp_local_ctrl_get_transport_ble`(void)

Function for obtaining BLE transport mode.

`const` *esp_local_ctrl_transport_t* *`esp_local_ctrl_get_transport_httpd`(void)

Function for obtaining HTTPD transport mode.

esp_err_t `esp_local_ctrl_start`(`const` *esp_local_ctrl_config_t* **config*)

Start local control service.

Return

- ESP_OK : Success
- ESP_FAIL : Failure

Parameters

- `config`: Pointer to configuration structure

esp_err_t `esp_local_ctrl_stop`(void)

Stop local control service.

esp_err_t **esp_local_ctrl_add_property**(**const** *esp_local_ctrl_prop_t* **prop*)

Add a new property.

This adds a new property and allocates internal resources for it. The total number of properties that could be added is limited by configuration option `max_properties`

Return

- ESP_OK : Success
- ESP_FAIL : Failure

Parameters

- `prop`: Property description structure

esp_err_t **esp_local_ctrl_remove_property**(**const** char **name*)

Remove a property.

This finds a property by name, and releases the internal resources which are associated with it.

Return

- ESP_OK : Success
- ESP_ERR_NOT_FOUND : Failure

Parameters

- `name`: Name of the property to remove

const *esp_local_ctrl_prop_t* ***esp_local_ctrl_get_property**(**const** char **name*)

Get property description structure by name.

This API may be used to get a property' s context structure `esp_local_ctrl_prop_t` when its name is known

Return

- Pointer to property
- NULL if not found

Parameters

- `name`: Name of the property to find

esp_err_t **esp_local_ctrl_set_handler**(**const** char **ep_name*, *protocomm_req_handler_t* *handler*, void **user_ctx*)

Register protocomm handler for a custom endpoint.

This API can be called by the application to register a protocomm handler for an endpoint after the local control service has started.

Note In case of BLE transport the names and uuids of all custom endpoints must be provided beforehand as a part of the `protocomm_ble_config_t` structure set in `esp_local_ctrl_config_t`, and passed to `esp_local_ctrl_start()`.

Return

- ESP_OK : Success
- ESP_FAIL : Failure

Parameters

- `ep_name`: Name of the endpoint
- `handler`: Endpoint handler function
- `user_ctx`: User data

Unions

`union esp_local_ctrl_transport_config_t`

#include <esp_local_ctrl.h> Transport mode (BLE / HTTPD) configuration.

Public Members

esp_local_ctrl_transport_config_ble_t ***ble**

This is same as `protocomm_ble_config_t`. See `protocomm_ble.h` for available configuration parameters.

esp_local_ctrl_transport_config_httpd_t ***httpd**

This is same as `httpd_ssl_config_t`. See `esp_https_server.h` for available configuration parameters.

Structures

`struct esp_local_ctrl_prop`

Property description data structure, which is to be populated and passed to the `esp_local_ctrl_add_property()` function.

Once a property is added, its structure is available for read-only access inside `get_prop_values()` and `set_prop_values()` handlers.

Public Members

char ***name**

Unique name of property

uint32_t **type**

Type of property. This may be set to application defined enums

size_t **size**

Size of the property value, which:

- if zero, the property can have values of variable size
- if non-zero, the property can have values of fixed size only, therefore, checks are performed internally by esp_local_ctrl when setting the value of such a property

uint32_t **flags**

Flags set for this property. This could be a bit field. A flag may indicate property behavior, e.g. read-only / constant

void ***ctx**

Pointer to some context data relevant for this property. This will be available for use inside the `get_prop_values` and `set_prop_values` handlers as a part of this property structure. When set, this is valid throughout the lifetime of a property, till either the property is removed or the esp_local_ctrl service is stopped.

void (***ctx_free_fn**)(void *ctx)

Function used by esp_local_ctrl to internally free the property context when `esp_local_ctrl_remove_property()` or `esp_local_ctrl_stop()` is called.

struct esp_local_ctrl_prop_val

Property value data structure. This gets passed to the `get_prop_values()` and `set_prop_values()` handlers for the purpose of retrieving or setting the present value of a property.

Public Members

void ***data**

Pointer to memory holding property value

size_t **size**

Size of property value

void (***free_fn**)(void *data)

This may be set by the application in `get_prop_values()` handler to tell `esp_local_ctrl` to call this function on the data pointer above, for freeing its resources after sending the `get_prop_values` response.

struct esp_local_ctrl_handlers

Handlers for receiving and responding to local control commands for getting and setting properties.

Public Members

esp_err_t (*`get_prop_values`)(size_t props_count, `const` *esp_local_ctrl_prop_t* props[], *esp_local_ctrl_prop_val_t* prop_values[], void *usr_ctx)

Handler function to be implemented for retrieving current values of properties.

Note If any of the properties have fixed sizes, the size field of corresponding element in `prop_values` need to be set

Return Returning different error codes will convey the corresponding protocol level errors to the client :

- ESP_OK : Success
- ESP_ERR_INVALID_ARG : InvalidArgument
- ESP_ERR_INVALID_STATE : InvalidProto
- All other error codes : InternalError

Parameters

- `props_count`: Total elements in the props array
- `props`: Array of properties, the current values for which have been requested by the client
- `prop_values`: Array of empty property values, the elements of which need to be populated with the current values of those properties specified by props argument
- `usr_ctx`: This provides value of the `usr_ctx` field of `esp_local_ctrl_handlers_t` structure

esp_err_t (*`set_prop_values`)(size_t props_count, `const` *esp_local_ctrl_prop_t* props[], `const` *esp_local_ctrl_prop_val_t* prop_values[], void *usr_ctx)

Handler function to be implemented for changing values of properties.

Note If any of the properties have variable sizes, the size field of the corresponding element in `prop_values` must be checked explicitly before making any assumptions on the size.

Return Returning different error codes will convey the corresponding protocol level errors to the client :

- ESP_OK : Success
- ESP_ERR_INVALID_ARG : InvalidArgument
- ESP_ERR_INVALID_STATE : InvalidProto
- All other error codes : InternalError

Parameters

- `props_count`: Total elements in the props array

- `props`: Array of properties, the values for which the client requests to change
- `prop_values`: Array of property values, the elements of which need to be used for updating those properties specified by props argument
- `usr_ctx`: This provides value of the `usr_ctx` field of `esp_local_ctrl_handlers_t` structure

void ***usr_ctx**

Context pointer to be passed to above handler functions upon invocation. This is different from the property level context, as this is valid throughout the lifetime of the `esp_local_ctrl` service, and freed only when the service is stopped.

void (***usr_ctx_free_fn**)(void *usr_ctx)

Pointer to function which will be internally invoked on `usr_ctx` for freeing the context resources when `esp_local_ctrl_stop()` is called.

struct esp_local_ctrl_config

Configuration structure to pass to `esp_local_ctrl_start()`

Public Members

const *esp_local_ctrl_transport_t* ***transport**

Transport layer over which service will be provided

esp_local_ctrl_transport_config_t **transport_config**

Transport layer over which service will be provided

esp_local_ctrl_handlers_t **handlers**

Register handlers for responding to get/set requests on properties

size_t **max_properties**

This limits the number of properties that are available at a time

Macros

ESP_LOCAL_CTRL_TRANSPORT_BLE

ESP_LOCAL_CTRL_TRANSPORT_HTTPD

Type Definitions

typedef struct *esp_local_ctrl_prop* **esp_local_ctrl_prop_t**

Property description data structure, which is to be populated and passed to the `esp_local_ctrl_add_property()` function.

Once a property is added, its structure is available for read-only access inside `get_prop_values()` and `set_prop_values()` handlers.

typedef struct *esp_local_ctrl_prop_val* **esp_local_ctrl_prop_val_t**

Property value data structure. This gets passed to the `get_prop_values()` and `set_prop_values()` handlers for the purpose of retrieving or setting the present value of a property.

typedef struct *esp_local_ctrl_handlers* **esp_local_ctrl_handlers_t**

Handlers for receiving and responding to local control commands for getting and setting properties.

typedef struct esp_local_ctrl_transport **esp_local_ctrl_transport_t**

Transport mode (BLE / HTTPD) over which the service will be provided.

This is forward declaration of a private structure, implemented internally by `esp_local_ctrl`.

typedef struct *protocomm_ble_config* **esp_local_ctrl_transport_config_ble_t**

Configuration for transport mode BLE.

This is a forward declaration for `protocomm_ble_config_t`. To use this, application must set CONFIG_BT_BLUEDROID_ENABLED and include `protocomm_ble.h`.

typedef struct *httpd_ssl_config* **esp_local_ctrl_transport_config_httpd_t**

Configuration for transport mode HTTPD.

This is a forward declaration for `httpd_ssl_config_t`. To use this, application must set CONFIG_ESP_HTTPS_SERVER_ENABLE and include `esp_https_server.h`

typedef struct *esp_local_ctrl_config* **esp_local_ctrl_config_t**

Configuration structure to pass to `esp_local_ctrl_start()`

2.4.12 ESP Serial Slave Link

Overview

Espressif provides several chips that can work as slaves. These slave devices rely on some common buses, and have their own communication protocols over those buses. The *esp_serial_slave_link* component is designed for the master to communicate with ESP slave devices through those protocols over the bus drivers.

After an *esp_serial_slave_link* device is initialized properly, the application can use it to communicate with the ESP slave devices conveniently.

For more details about ESP32 SDIO slave protocol, see document *Communication with ESP SDIO Slave*.

Terminology

- ESSL: Abbreviation for ESP Serial Slave Link, the component described by this document.
- Master: The device running the *esp_serial_slave_link* component.

- ESSL device: a virtual device on the master associated with an ESP slave device. The device context has the knowledge of the slave protocol above the bus, relying on some bus drivers to communicate with the slave.
- ESSL device handle: a handle to ESSL device context containing the configuration, status and data required by the ESSL component. The context stores the driver configurations, communication state, data shared by master and slave, etc.

  The context should be initialized before it is used, and get deinitialized if not used any more. The master application operates on the ESSL device through this handle.
- ESP slave: the slave device connected to the bus, which ESSL component is designed to communicate with.
- Bus: The bus over which the master and the slave communicate with each other.
- Slave protocol: The special communication protocol specified by Espressif HW/SW over the bus.
- TX buffer num: a counter, which is on the slave and can be read by the master, indicates the accumulated buffer numbers that the slave has loaded to the hardware to receive data from the master.
- RX data size: a counter, which is on the slave and can be read by the master, indicates the accumulated data size that the slave has loaded to the hardware to send to the master.

Services provided by ESP slave

There are some common services provided by the Espressif slaves:

1. Tohost Interrupts: The slave can inform the master about certain events by the interrupt line.
2. Frhost Interrupts: The master can inform the slave about certain events.
3. Tx FIFO (master to slave): the slave can send data in stream to the master. The SDIO slave can also indicate it has new data to send to master by the interrupt line.

   The slave updates the TX buffer num to inform the master how much data it can receive, and the master then read the TX buffer num, and take off the used buffer number to know how many buffers are remaining.
4. Rx FIFO (slave to master): the slave can receive data from the master in units of receiving buffers.

   The slave updates the RX data size to inform the master how much data it has prepared to send, and then the master read the data size, and take off the data length it has already received to know how many data is remaining.
5. Shared registers: the master can read some part of the registers on the slave, and also write these registers to let the slave read.

Initialization of ESP SDIO Slave Link

The ESP SDIO slave link (ESSL SDIO) devices relies on the sdmmc component. The ESSL device should be initialized as below:

1. Initialize a sdmmc card (see :doc:‘ Document of SDMMC driver </api-reference/storage/sdmmc>‘) structure.
2. Call *sdmmc_card_init()* to initialize the card.
3. Initialize the ESSL device with *essl_sdio_config_t*. The *card* member should be the *sdmmc_card_t* got in step 2, and the *recv_buffer_size* member should be filled correctly according to pre-negotiated value.
4. Call *essl_init()* to do initialization of the SDIO part.
5. Call *essl_wait_for_ready()* to wait for the slave to be ready.

APIs

After the initialization process above is performed, you can call the APIs below to make use of the services provided by the slave:

Interrupts

1. Call *essl_get_intr_ena()* to know which events will trigger the interrupts to the master.
2. Call *essl_set_intr_ena()* to set the events that will trigger interrupts to the master.
3. Call *essl_wait_int()* to wait until interrupt from the slave, or timeout.
4. When interrupt is triggered, call *essl_get_intr()* to know which events are active, and call *essl_clear_intr()* to clear them.
5. Call *essl_send_slave_intr()* to trigger general purpose interrupt of the slave.

TX FIFO

1. Call *essl_get_tx_buffer_num()* to know how many buffers the slave has prepared to receive data from the master. This is optional. The master will poll *tx_buffer_num* when it try to send packets to the slave, until the slave has enough buffer or timeout.
2. Call `essl_send_paket()` to send data to the slave.

RX FIFO

1. Call *essl_get_rx_data_size()* to know how many data the slave has prepared to send to the master. This is optional. When the master tries to receive data from the slave, it will update the *rx_data_size* for once, if the current *rx_data_size* is shorter than the buffer size the master prepared to receive. And it may poll the *rx_data_size* if the *rx_dat_size* keeps 0, until timeout.
2. Call *essl_get_packet()* to receive data from the slave.

Reset counters (Optional)

Call *essl_reset_cnt()* to reset the internal counter if you find the slave has reset its counter.

Application Example

The example below shows how ESP32 SDIO host and slave communicate with each other. The host use the ESSL SDIO.

peripherals/sdio.

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

API Reference

Header File

- esp_serial_slave_link/include/esp_serial_slave_link/essl.h

Functions

esp_err_t **essl_init**(*essl_handle_t handle*, uint32_t *wait_ms*)

Initialize the slave.

Return ESP_OK if success, or other value returned from lower layer `init`.

Parameters

- `handle`: Handle of a `essl` device.
- `wait_ms`: Millisecond to wait before timeout, will not wait at all if set to 0-9.

esp_err_t **essl_wait_for_ready**(*essl_handle_t handle*, uint32_t *wait_ms*)

Wait for interrupt of a ESP32 slave device.

Return

- ESP_OK if success
- One of the error codes from SDMMC host controller

Parameters

- `handle`: Handle of a `essl` device.
- `wait_ms`: Millisecond to wait before timeout, will not wait at all if set to 0-9.

esp_err_t **essl_get_tx_buffer_num**(*essl_handle_t handle*, uint32_t **out_tx_num*, uint32_t *wait_ms*)

Get buffer num for the host to send data to the slave. The buffers are size of `buffer_size`.

Return

- ESP_OK Success
- One of the error codes from SDMMC host controller

Parameters

- `handle`: Handle of a `essl` device.
- `out_tx_num`: Output of buffer num that host can send data to ESP32 slave.
- `wait_ms`: Millisecond to wait before timeout, will not wait at all if set to 0-9.

esp_err_t **essl_get_rx_data_size**(*essl_handle_t handle*, uint32_t **out_rx_size*, uint32_t *wait_ms*)

Get amount of data the ESP32 slave preparing to send to host.

Return

- ESP_OK Success
- One of the error codes from SDMMC host controller

Parameters

- `handle`: Handle of a `essl` device.
- `out_rx_size`: Output of data size to read from slave.
- `wait_ms`: Millisecond to wait before timeout, will not wait at all if set to 0-9.

esp_err_t **essl_reset_cnt**(*essl_handle_t handle*)

Reset the counters of this component. Usually you don't need to do this unless you know the slave is reset.

Parameters

- `handle`: Handle of a `essl` device.

esp_err_t **essl_send_packet**(*essl_handle_t handle*, **const** void **start*, size_t *length*, uint32_t *wait_ms*)

Send a packet to the ESP32 slave. The slave receive the packet into buffers whose size is `buffer_size` (configured during initialization).

Return

- ESP_OK Success
- ESP_ERR_TIMEOUT No buffer to use, or error ftrom SDMMC host controller
- One of the error codes from SDMMC host controller

Parameters

- `handle`: Handle of a `essl` device.
- `start`: Start address of the packet to send
- `length`: Length of data to send, if the packet is over-size, the it will be divided into blocks and hold into different buffers automatically.
- `wait_ms`: Millisecond to wait before timeout, will not wait at all if set to 0-9.

esp_err_t **essl_get_packet**(*essl_handle_t handle*, void **out_data*, size_t *size*, size_t **out_length*, uint32_t *wait_ms*)

Get a packet from ESP32 slave.

Return

- ESP_OK Success, all the data are read from the slave.
- ESP_ERR_NOT_FINISHED Read success, while there' re data remaining.
- One of the error codes from SDMMC host controller

Parameters

- `handle`: Handle of a `essl` device.
- `out_data`: Data output address
- `size`: The size of the output buffer, if the buffer is smaller than the size of data to receive from slave, the driver returns `ESP_ERR_NOT_FINISHED`
- `out_length`: Output of length the data actually received from slave.
- `wait_ms`: Millisecond to wait before timeout, will not wait at all if set to 0-9.

esp_err_t **essl_write_reg**(*essl_handle_t handle*, uint8_t *addr*, uint8_t *value*, uint8_t **value_o*, uint32_t *wait_ms*)

Write general purpose R/W registers (8-bit) of ESP32 slave.

Note sdio 28-31 are reserved, the lower API helps to skip.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Address not valid.
- One of the error codes from SDMMC host controller

Parameters

- `handle`: Handle of a `essl` device.
- `addr`: Address of register to write. Valid address: 0-59.
- `value`: Value to write to the register.
- `value_o`: Output of the returned written value.
- `wait_ms`: Millisecond to wait before timeout, will not wait at all if set to 0-9.

esp_err_t **essl_read_reg**(*essl_handle_t handle*, uint8_t *add*, uint8_t **value_o*, uint32_t *wait_ms*)

Read general purpose R/W registers (8-bit) of ESP32 slave.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Address not valid.
- One of the error codes from SDMMC host controller

Parameters

- `handle`: Handle of a `essl` device.
- `add`: Address of register to read. Valid address: 0-27, 32-63 (28-31 reserved, return interrupt bits on read).
- `value_o`: Output value read from the register.
- `wait_ms`: Millisecond to wait before timeout, will not wait at all if set to 0-9.

esp_err_t **essl_wait_int**(*essl_handle_t handle*, uint32_t *wait_ms*)

wait for an interrupt of the slave

Return

- ESP_ERR_NOT_SUPPORTED Currently our driver doesnot support SDIO with SPI interface.
- ESP_OK If interrupt triggered.
- ESP_ERR_TIMEOUT No interrupts before timeout.

Parameters

- `handle`: Handle of a `essl` device.
- `wait_ms`: Millisecond to wait before timeout, will not wait at all if set to 0-9.

esp_err_t **essl_clear_intr**(*essl_handle_t handle*, uint32_t *intr_mask*, uint32_t *wait_ms*)

Clear interrupt bits of ESP32 slave. All the bits set in the mask will be cleared, while other bits will stay the same.

Return

- ESP_OK Success
- One of the error codes from SDMMC host controller

Parameters

- `handle`: Handle of a `essl` device.
- `intr_mask`: Mask of interrupt bits to clear.
- `wait_ms`: Millisecond to wait before timeout, will not wait at all if set to 0-9.

esp_err_t **essl_get_intr**(*essl_handle_t handle*, uint32_t **intr_raw*, uint32_t **intr_st*, uint32_t *wait_ms*)

Get interrupt bits of ESP32 slave.

Return

- ESP_OK Success
- ESP_INVALID_ARG if both `intr_raw` and `intr_st` are NULL.
- One of the error codes from SDMMC host controller

Parameters

- `handle`: Handle of a `essl` device.
- `intr_raw`: Output of the raw interrupt bits. Set to NULL if only masked bits are read.
- `intr_st`: Output of the masked interrupt bits. set to NULL if only raw bits are read.
- `wait_ms`: Millisecond to wait before timeout, will not wait at all if set to 0-9.

esp_err_t **essl_set_intr_ena**(*essl_handle_t handle*, uint32_t *ena_mask*, uint32_t *wait_ms*)

Set interrupt enable bits of ESP32 slave. The slave only sends interrupt on the line when there is a bit both the raw status and the enable are set.

Return

- ESP_OK Success
- One of the error codes from SDMMC host controller

Parameters

- `handle`: Handle of a `essl` device.
- `ena_mask`: Mask of the interrupt bits to enable.
- `wait_ms`: Millisecond to wait before timeout, will not wait at all if set to 0-9.

esp_err_t **essl_get_intr_ena**(*essl_handle_t handle*, uint32_t **ena_mask_o*, uint32_t *wait_ms*)

Get interrupt enable bits of ESP32 slave.

Return

- ESP_OK Success
- One of the error codes from SDMMC host controller

Parameters

- `handle`: Handle of a `essl` device.
- `ena_mask_o`: Output of interrupt bit enable mask.
- `wait_ms`: Millisecond to wait before timeout, will not wait at all if set to 0-9.

esp_err_t **essl_send_slave_intr**(*essl_handle_t handle*, uint32_t *intr_mask*, uint32_t *wait_ms*)

Send interrupts to slave. Each bit of the interrupt will be triggered.

Return

- ESP_OK Success
- One of the error codes from SDMMC host controller

Parameters

- `handle`: Handle of a `essl` device.
- `intr_mask`: Mask of interrupt bits to send to slave.
- `wait_ms`: Millisecond to wait before timeout, will not wait at all if set to 0-9.

Macros

`ESP_ERR_NOT_FINISHED`

There is still remaining data.

Type Definitions

typedef struct essl_dev_t ***essl_handle_t**

Handle of an ESSL device.

Header File

- esp_serial_slave_link/include/esp_serial_slave_link/essl_sdio.h

Functions

esp_err_t **essl_sdio_init_dev**(*essl_handle_t* **out_handle*, **const** *essl_sdio_config_t* **config*)

Initialize the ESSL SDIO device and get its handle.

Return

- ESP_OK: on success
- ESP_ERR_NO_MEM: memory exhausted.

Parameters

- `out_handle`: Output of the handle.
- `config`: Configuration for the ESSL SDIO device.

esp_err_t **essl_sdio_deinit_dev**(*essl_handle_t handle*)

Deinitialize and free the space used by the ESSL SDIO device.

Return

- ESP_OK: on success
- ESP_ERR_INVALID_ARG: wrong handle passed

Parameters

- `handle`: Handle of the ESSL SDIO device to deinit.

Structures

struct essl_sdio_config_t

Configuration for the essl SDIO device.

Public Members

sdmmc_card_t ***card**

The initialized sdmmc card pointer of the slave.

int **recv_buffer_size**

The pre-negotiated recv buffer size used by both the host and the slave.

此 API 部分的示例代码在 ESP-IDF 示例工程的 protocols 目录下提供。

2.4.13 IP 网络层协议

IP 网络层协议（应用层协议之下）的文档位于连网 *API* 。

2.5 配网 API

[English]

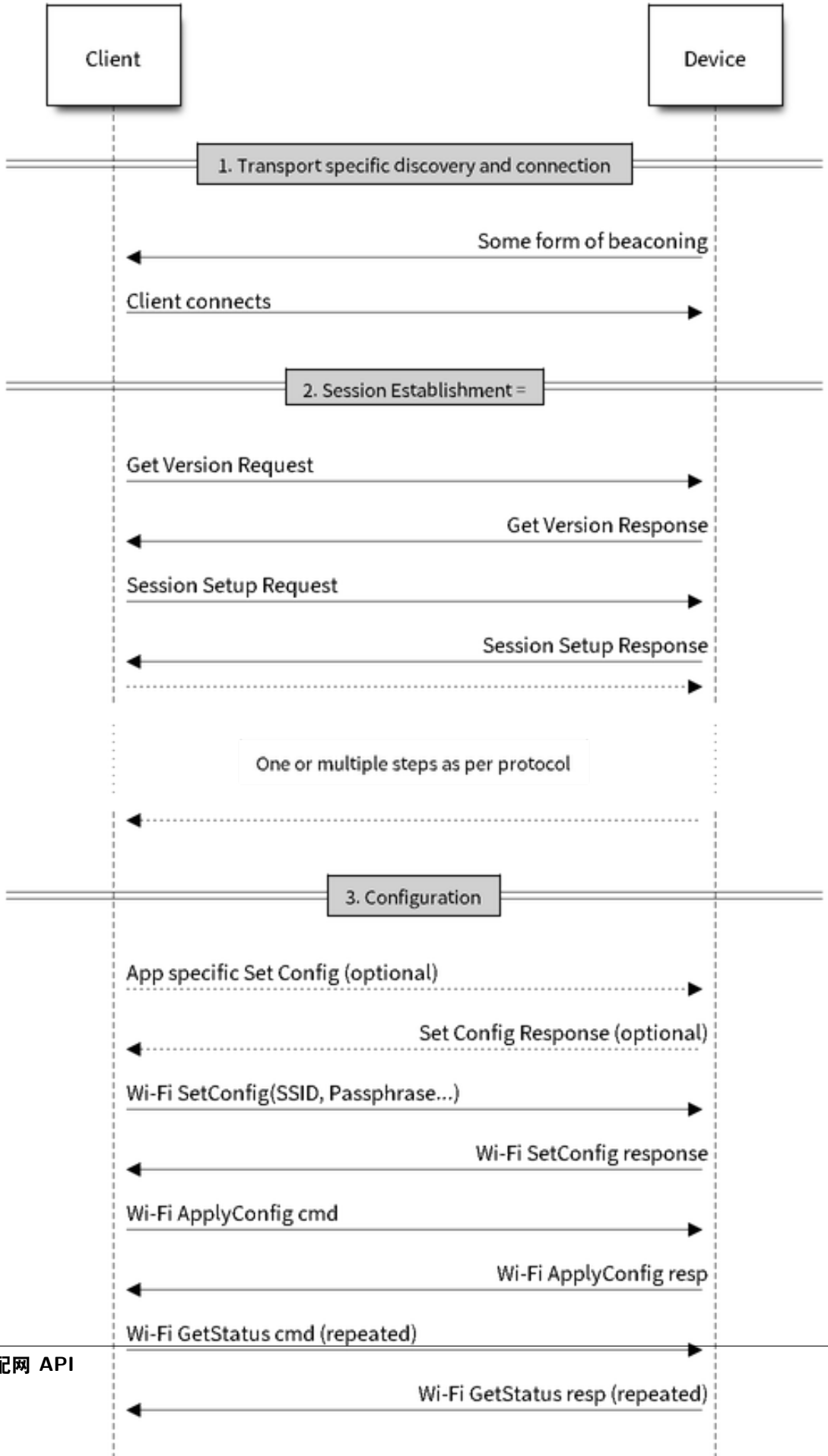
2.5.1 Unified Provisioning

Overview

Unified provisioning support in the ESP-IDF provides an extensible mechanism to the developers to configure the device with the Wi-Fi credentials and/or other custom configuration using various transports and different security schemes. Depending on the use-case it provides a complete and ready solution for Wi-Fi network provisioning along with example iOS and Android applications. Or developers can extend the device-side and phone-app side implementations to accommodate their requirements for sending additional configuration data. Following are the important features of this implementation.

1. *Extensible Protocol:* The protocol is completely flexible and it offers the ability for the developers to send custom configuration in the provisioning process. The data representation too is left to the application to decide.
2. *Transport Flexibility:* The protocol can work on Wi-Fi (SoftAP + HTTP server) or on BLE as a transport protocol. The framework provides an ability to add support for any other transport easily as long as command-response behaviour can be supported on the transport.
3. *Security Scheme Flexibility:* It's understood that each use-case may require different security scheme to secure the data that is exchanged in the provisioning process. Some applications may work with SoftAP that's WPA2 protected or BLE with "just-works" security. Or the applications may consider the transport to be insecure and may want application level security. The unified provisioning framework allows application to choose the security as deemed suitable.
4. *Compact Data Representation:* The protocol uses Google Protobufs as a data representation for session setup and Wi-Fi provisioning. They provide a compact data representation and ability to parse the data in multiple programming languages in native format. Please note that this data representation is not forced on application specific data and the developers may choose the representation of their choice.

Typical Provisioning Process


Client
Device
1. Transport specific discovery and connection
Some form of beaconing
Client connects
2. Session Establishment =
Get Version Request
Get Version Response
Session Setup Request
Session Setup Response
One or multiple steps as per protocol
3. Configuration
App specific Set Config (optional)
Set Config Response (optional)
Wi-Fi SetConfig(SSID, Passphrase...)
Wi-Fi SetConfig response
Wi-Fi ApplyConfig cmd
Wi-Fi ApplyConfig resp
Wi-Fi GetStatus cmd (repeated)
Wi-Fi GetStatus resp (repeated)

Deciding on Transport

Unified provisioning subsystem supports Wi-Fi (SoftAP+HTTP server) and BLE (GATT based) transport schemes. Following points need to be considered while selecting the best possible transport for provisioning.

1. BLE based transport has an advantage that in the provisioning process, the BLE communication channel stays intact between the device and the client. That provides reliable provisioning feedback.
2. BLE based provisioning implementation makes the user-experience better from the phone apps as on Android and iOS both, the phone app can discover and connect to the device without requiring user to go out of the phone app
3. BLE transport however consumes ~110KB memory at runtime. If the product does not use the BLE or BT functionality after provisioning is done, almost all the memory can be reclaimed back and can be added into the heap.
4. SoftAP based transport is highly interoperable; however as the same radio is shared between SoftAP and Station interface, the transport is not reliable in the phase when the Wi-Fi connection to external AP is attempted. Also, the client may roam back to different network when the SoftAP changes the channel at the time of Station connection.
5. SoftAP transport does not require much additional memory for the Wi-Fi use-cases
6. SoftAP based provisioning requires the phone app user to go to "System Settings" to connect to Wi-Fi network hosted by the device in case of iOS. The discovery (scanning) as well as connection API is not available for the iOS applications.

Deciding on Security

Depending on the transport and other constraints the security scheme needs to be selected by the application developers. Following considerations need to be given from the provisioning security perspective: 1. The configuration data sent from the client to the device and the response has to be secured. 2. The client should authenticate the device it is connected to. 3. The device manufacturer may choose proof-of-possession - a unique per device secret to be entered on the provisioning client as a security measure to make sure that the user can provisions the device in the possession.

There are two levels of security schemes. The developer may select one or combination depending on requirements.

1. *Transport Security:* SoftAP provisioning may choose WPA2 protected security with unique per-device passphrase. Per-device unique passphrase can also act as a proof-of-possession. For BLE, "just-works" security can be used as a transport level security after understanding the level of security it provides.
2. *Application Security:* The unified provisioning subsystem provides application level security (*security1*) that provides data protection and authentication (through proof-of-possession) if the application does not use the transport level security or if the transport level security is not sufficient for the use-case.

Device Discovery

The advertisement and device discovery is left to the application and depending on the protocol chosen, the phone apps and device firmware application can choose appropriate method to advertise and discovery.

For the SoftAP+HTTP transport, typically the SSID (network name) of the AP hosted by the device can be used for discovery.

For the BLE transport device name or primary service included in the advertisement or combination of both can be used for discovery.

Architecture

The below diagram shows architecture of unified provisioning.

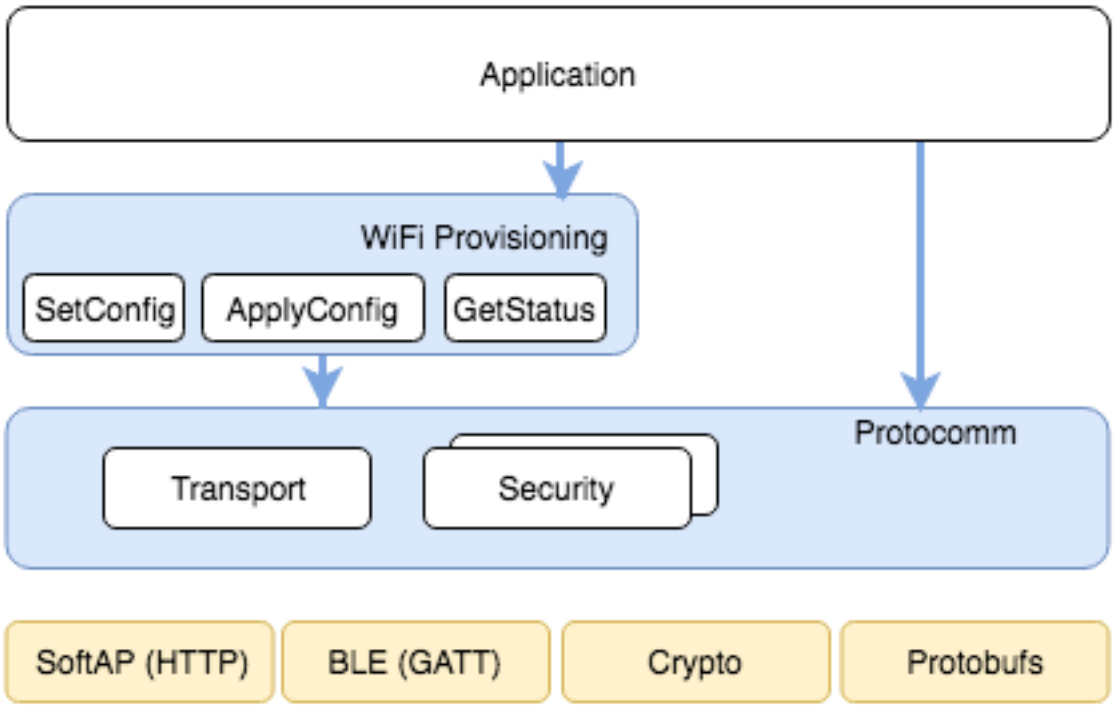


图 24: Unified Provisioning Architecture

It relies on the base layer called *Protocol Communication* (Protocol Communication) which provides a framework for security schemes and transport mechanisms. Wi-Fi Provisioning layer uses Protocomm to provide simple callbacks to the application for setting the configuration and getting the Wi-Fi status. The application has control over implementation of these callbacks. In addition application can directly use protocomm to register custom handlers.

Application creates a protocomm instance which is mapped to a specific transport and specific security scheme. Each transport in the protocomm has a concept of an "end-point" which corresponds to logical

channel for communication for specific type of information. For example security handshake happens on a different endpoint than the Wi-Fi configuration endpoint. Each end-point is identified using a string and depending on the transport internal representation of the end-point changes. In case of SoftAP+HTTP transport the end-point corresponds to URI whereas in case of BLE the end-point corresponds to GATT characteristic with specific UUID. Developers can create custom end-points and implement handler for the data that is received or sent over the same end-point.

Security Schemes

At present unified provisioning supports two security schemes: 1. Security0 - No security (No encryption) 2. Security1 - Curve25519 based key exchange, shared key derivation and AES256-CTR mode encryption of the data. It supports two modes :

a. Authorized - Proof of Possession (PoP) string used to authorize session and derive shared key

b. No Auth (Null PoP) - Shared key derived through key exchange only

Security1 scheme details are shown in the below sequence diagram

Sample Code

Please refer to *Protocol Communication* and *Wi-Fi Provisioning* for API guides and code snippets on example usage.

Various use case implementations can be found as examples under provisioning.

Provisioning Tools

Provisioning applications are available for various platforms, along with source code:

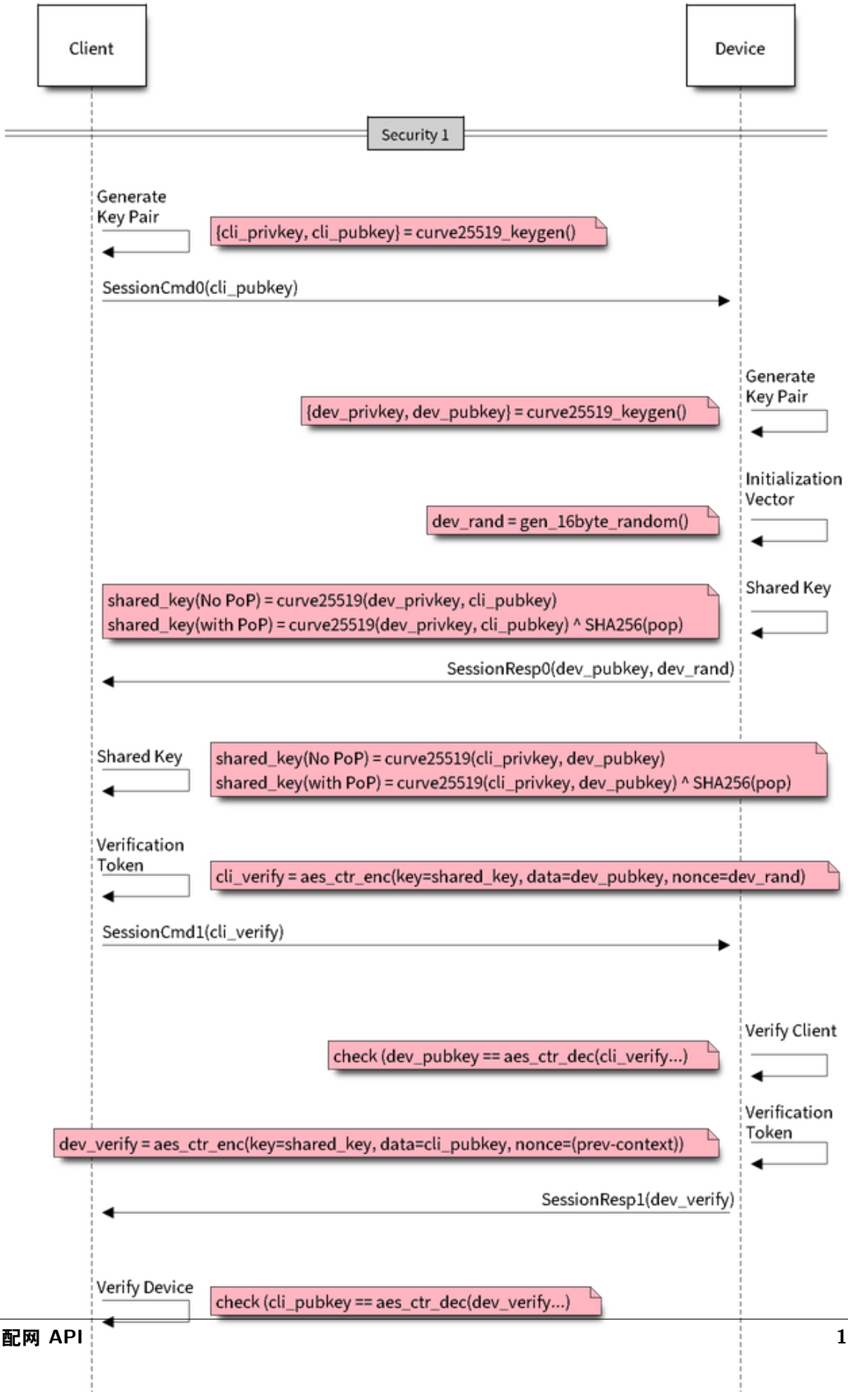
- Android : esp-idf-provisioning-android
- iOS : esp-idf-provisioning-ios
- Linux/MacOS/Windows : tools/esp_prov (a python based command line tool for provisioning)

The phone applications offer simple UI and thus more user centric, while the command line application is useful as a debugging tool for developers.

2.5.2 Protocol Communication

Overview

Protocol Communication (protocomm) component manages secure sessions and provides framework for multiple transports. The application can also use protocomm layer directly to have application specific extensions for the provisioning (or non-provisioning) use cases.


Client
Device
Security 1
Generate Key Pair
{cli_privkey, cli_pubkey} = curve25519_keygen()
SessionCmd0(cli_pubkey)
Generate Key Pair
{dev_privkey, dev_pubkey} = curve25519_keygen()
Initialization Vector
dev_rand = gen_16byte_random()
Shared Key
shared_key(No PoP) = curve25519(dev_privkey, cli_pubkey)
shared_key(with PoP) = curve25519(dev_privkey, cli_pubkey) ^ SHA256(pop)
SessionResp0(dev_pubkey, dev_rand)
Shared Key
shared_key(No PoP) = curve25519(cli_privkey, dev_pubkey)
shared_key(with PoP) = curve25519(cli_privkey, dev_pubkey) ^ SHA256(pop)
Verification Token
cli_verify = aes_ctr_enc(key=shared_key, data=dev_pubkey, nonce=dev_rand)
SessionCmd1(cli_verify)
Verify Client
check (dev_pubkey == aes_ctr_dec(cli_verify...)
Verification Token
dev_verify = aes_ctr_enc(key=shared_key, data=cli_pubkey, nonce=(prev-context))
SessionResp1(dev_verify)
Verify Device
check (cli_pubkey == aes_ctr_dec(dev_verify...)

Following features are available for provisioning :

- **Communication security at application level -**
  - protocomm_security0 (no security)
  - protocomm_security1 (curve25519 key exchange + AES-CTR encryption)
- Proof-of-possession (support with protocomm_security1 only)

Protocomm internally uses protobuf (protocol buffers) for secure session establishment. Though users can implement their own security (even without using protobuf). One can even use protocomm without any security layer.

Protocomm provides framework for various transports - WiFi (SoftAP+HTTPD), BLE, console - in which case the handler invocation is automatically taken care of on the device side (see Transport Examples below for code snippets).

Note that the client still needs to establish session (only for protocomm_security1) by performing the two way handshake. See *Unified Provisioning* for more details about the secure handshake logic.

Transport Example (SoftAP + HTTP) with Security 1

For complete example see provisioning/softap_prov

```
/* Endpoint handler to be registered with protocomm.
 * This simply echoes back the received data. */
esp_err_t echo_req_handler (uint32_t session_id,
                            const uint8_t *inbuf, ssize_t inlen,
                            uint8_t **outbuf, ssize_t *outlen,
                            void *priv_data)
{
    /* Session ID may be used for persistence */
    printf("Session ID : %d", session_id);

    /* Echo back the received data */
    *outlen = inlen;            /* Output data length updated */
    *outbuf = malloc(inlen);    /* This will be deallocated outside */
    memcpy(*outbuf, inbuf, inlen);

    /* Private data that was passed at the time of endpoint creation */
    uint32_t *priv = (uint32_t *) priv_data;
    if (priv) {
        printf("Private data : %d", *priv);
    }
```

(下页继续)

(续上页)

```
    return ESP_OK;
}

/* Example function for launching a protocomm instance over HTTP */
protocomm_t *start_pc(const char *pop_string)
{
    protocomm_t *pc = protocomm_new();


    /* Config for protocomm_httpd_start() */
    protocomm_httpd_config_t pc_config = {
        .data = {
        .config = PROTOCOMM_HTTPD_DEFAULT_CONFIG()
        }
    };

    /* Start protocomm server on top of HTTP */
    protocomm_httpd_start(pc, &pc_config);

    /* Create Proof of Possession object from pop_string. It must be valid
     * throughout the scope of protocomm endpoint. This need not be static,
     * ie. could be dynamically allocated and freed at the time of endpoint
     * removal */
    const static protocomm_security_pop_t pop_obj = {
        .data = (const uint8_t *) strdup(pop_string),
        .len = strlen(pop_string)
    };

    /* Set security for communication at application level. Just like for
     * request handlers, setting security creates an endpoint and registers
     * the handler provided by protocomm_security1. One can similarly use
     * protocomm_security0. Only one type of security can be set for a
     * protocomm instance at a time. */
    protocomm_set_security(pc, "security_endpoint", &protocomm_security1, &pop_
↪obj);

    /* Private data passed to the endpoint must be valid throughout the scope
     * of protocomm endpoint. This need not be static, ie. could be dynamically
     * allocated and freed at the time of endpoint removal */
```

(下页继续)

(续上页)

```
    static uint32_t priv_data = 1234;

    /* Add a new endpoint for the protocomm instance, identified by a unique␣
↪name
     * and register a handler function along with private data to be passed at␣
↪the
     * time of handler execution. Multiple endpoints can be added as long as␣
↪they
     * are identified by unique names */
    protocomm_add_endpoint(pc, "echo_req_endpoint",
                           echo_req_handler, (void *) &priv_data);
    return pc;
}

/* Example function for stopping a protocomm instance */
void stop_pc(protocomm_t *pc)
{
    /* Remove endpoint identified by it's unique name */
    protocomm_remove_endpoint(pc, "echo_req_endpoint");

    /* Remove security endpoint identified by it's name */
    protocomm_unset_security(pc, "security_endpoint");

    /* Stop HTTP server */
    protocomm_httpd_stop(pc);

    /* Delete (deallocate) the protocomm instance */
    protocomm_delete(pc);
}
```

Transport Example (BLE) with Security 0

For complete example see provisioning/ble_prov

```
/* Example function for launching a secure protocomm instance over BLE */
protocomm_t *start_pc()
{
    protocomm_t *pc = protocomm_new();
```

(下页继续)

(续上页)

```
    /* Endpoint UUIDs */
    protocomm_ble_name_uuid_t nu_lookup_table[] = {
        {"security_endpoint", 0xFF51},
        {"echo_req_endpoint", 0xFF52}
    };

    /* Config for protocomm_ble_start() */
    protocomm_ble_config_t config = {
        .service_uuid = {
            /* LSB <---------------------------------------
            * ---------------------------------------> MSB */
            0xfb, 0x34, 0x9b, 0x5f, 0x80, 0x00, 0x00, 0x80,
            0x00, 0x10, 0x00, 0x00, 0xFF, 0xFF, 0x00, 0x00,
        },
        .nu_lookup_count = sizeof(nu_lookup_table)/sizeof(nu_lookup_table[0]),
        .nu_lookup = nu_lookup_table
    };

    /* Start protocomm layer on top of BLE */
    protocomm_ble_start(pc, &config);

    /* For protocomm_security0, Proof of Possession is not used, and can be␣
↪kept NULL */
    protocomm_set_security(pc, "security_endpoint", &protocomm_security0,␣
↪NULL);
    protocomm_add_endpoint(pc, "echo_req_endpoint", echo_req_handler, NULL);
    return pc;
}

/* Example function for stopping a protocomm instance */
void stop_pc(protocomm_t *pc)
{
    protocomm_remove_endpoint(pc, "echo_req_endpoint");
    protocomm_unset_security(pc, "security_endpoint");

    /* Stop BLE protocomm service */
    protocomm_ble_stop(pc);

    protocomm_delete(pc);
}
```

API Reference

Header File

- protocomm/include/common/protocomm.h

Functions

protocomm_t ***protocomm_new**(void)

Create a new protocomm instance.

This API will return a new dynamically allocated protocomm instance with all elements of the protocomm_t structure initialized to NULL.

Return

- protocomm_t* : On success
- NULL : No memory for allocating new instance

void **protocomm_delete**(*protocomm_t* **pc*)

Delete a protocomm instance.

This API will deallocate a protocomm instance that was created using `protocomm_new()`.

Parameters

- `pc`: Pointer to the protocomm instance to be deleted

esp_err_t **protocomm_add_endpoint**(*protocomm_t* **pc*, **const** char **ep_name*, *protocomm_req_handler_t h*, void **priv_data*)

Add endpoint request handler for a protocomm instance.

This API will bind an endpoint handler function to the specified endpoint name, along with any private data that needs to be pass to the handler at the time of call.

Note

- An endpoint must be bound to a valid protocomm instance, created using `protocomm_new()`.
- This function internally calls the registered `add_endpoint()` function of the selected transport which is a member of the protocomm_t instance structure.

Return

- ESP_OK : Success
- ESP_FAIL : Error adding endpoint / Endpoint with this name already exists
- ESP_ERR_NO_MEM : Error allocating endpoint resource

- ESP_ERR_INVALID_ARG : Null instance/name/handler arguments

Parameters

- `pc`: Pointer to the protocomm instance
- `ep_name`: Endpoint identifier(name) string
- `h`: Endpoint handler function
- `priv_data`: Pointer to private data to be passed as a parameter to the handler function on call. Pass NULL if not needed.

esp_err_t **protocomm_remove_endpoint**(*protocomm_t* *pc, **const** char *ep_name)

Remove endpoint request handler for a protocomm instance.

This API will remove a registered endpoint handler identified by an endpoint name.

Note

- This function internally calls the registered `remove_endpoint()` function which is a member of the protocomm_t instance structure.

Return

- ESP_OK : Success
- ESP_ERR_NOT_FOUND : Endpoint with specified name doesn't exist
- ESP_ERR_INVALID_ARG : Null instance/name arguments

Parameters

- `pc`: Pointer to the protocomm instance
- `ep_name`: Endpoint identifier(name) string

esp_err_t **protocomm_open_session**(*protocomm_t* *pc, uint32_t *session_id*)

Allocates internal resources for new transport session.

Note

- An endpoint must be bound to a valid protocomm instance, created using `protocomm_new()`.

Return

- ESP_OK : Request handled successfully
- ESP_ERR_NO_MEM : Error allocating internal resource
- ESP_ERR_INVALID_ARG : Null instance/name arguments

Parameters

- `pc`: Pointer to the protocomm instance

- `session_id`: Unique ID for a communication session

esp_err_t **protocomm_close_session**(*protocomm_t* **pc*, uint32_t *session_id*)

Frees internal resources used by a transport session.

Note

- An endpoint must be bound to a valid protocomm instance, created using `protocomm_new()`.

Return

- ESP_OK : Request handled successfully
- ESP_ERR_INVALID_ARG : Null instance/name arguments

Parameters

- `pc`: Pointer to the protocomm instance
- `session_id`: Unique ID for a communication session

esp_err_t **protocomm_req_handle**(*protocomm_t* **pc*, **const** char **ep_name*, uint32_t *session_id*, **const** uint8_t **inbuf*, ssize_t *inlen*, uint8_t ***outbuf*, ssize_t **outlen*)

Calls the registered handler of an endpoint session for processing incoming data and generating the response.

Note

- An endpoint must be bound to a valid protocomm instance, created using `protocomm_new()`.
- Resulting output buffer must be deallocated by the caller.

Return

- ESP_OK : Request handled successfully
- ESP_FAIL : Internal error in execution of registered handler
- ESP_ERR_NO_MEM : Error allocating internal resource
- ESP_ERR_NOT_FOUND : Endpoint with specified name doesn’ t exist
- ESP_ERR_INVALID_ARG : Null instance/name arguments

Parameters

- `pc`: Pointer to the protocomm instance
- `ep_name`: Endpoint identifier(name) string
- `session_id`: Unique ID for a communication session
- `inbuf`: Input buffer contains input request data which is to be processed by the registered handler

- `inlen`: Length of the input buffer
- `outbuf`: Pointer to internally allocated output buffer, where the resulting response data output from the registered handler is to be stored
- `outlen`: Buffer length of the allocated output buffer

esp_err_t **protocomm_set_security**(*protocomm_t* *pc, **const** char *ep_name, **const** *protocomm_security_t* *sec, **const** *protocomm_security_pop_t* *pop)

Add endpoint security for a protocomm instance.

This API will bind a security session establisher to the specified endpoint name, along with any proof of possession that may be required for authenticating a session client.

Note

- An endpoint must be bound to a valid protocomm instance, created using `protocomm_new()`.
- The choice of security can be any `protocomm_security_t` instance. Choices `protocomm_security0` and `protocomm_security1` are readily available.

Return

- ESP_OK : Success
- ESP_FAIL : Error adding endpoint / Endpoint with this name already exists
- ESP_ERR_INVALID_STATE : Security endpoint already set
- ESP_ERR_NO_MEM : Error allocating endpoint resource
- ESP_ERR_INVALID_ARG : Null instance/name/handler arguments

Parameters

- `pc`: Pointer to the protocomm instance
- `ep_name`: Endpoint identifier(name) string
- `sec`: Pointer to endpoint security instance
- `pop`: Pointer to proof of possession for authenticating a client

esp_err_t **protocomm_unset_security**(*protocomm_t* *pc, **const** char *ep_name)

Remove endpoint security for a protocomm instance.

This API will remove a registered security endpoint identified by an endpoint name.

Return

- ESP_OK : Success
- ESP_ERR_NOT_FOUND : Endpoint with specified name doesn't exist

- ESP_ERR_INVALID_ARG : Null instance/name arguments

Parameters

- `pc`: Pointer to the protocomm instance
- `ep_name`: Endpoint identifier(name) string

esp_err_t **protocomm_set_version**(*protocomm_t* **pc*, **const** char **ep_name*, **const** char **version*)

Set endpoint for version verification.

This API can be used for setting an application specific protocol version which can be verified by clients through the endpoint.

Note

- An endpoint must be bound to a valid protocomm instance, created using `protocomm_new()`.

Return

- ESP_OK : Success
- ESP_FAIL : Error adding endpoint / Endpoint with this name already exists
- ESP_ERR_INVALID_STATE : Version endpoint already set
- ESP_ERR_NO_MEM : Error allocating endpoint resource
- ESP_ERR_INVALID_ARG : Null instance/name/handler arguments

Parameters

- `pc`: Pointer to the protocomm instance
- `ep_name`: Endpoint identifier(name) string
- `version`: Version identifier(name) string

esp_err_t **protocomm_unset_version**(*protocomm_t* **pc*, **const** char **ep_name*)

Remove version verification endpoint from a protocomm instance.

This API will remove a registered version endpoint identified by an endpoint name.

Return

- ESP_OK : Success
- ESP_ERR_NOT_FOUND : Endpoint with specified name doesn't exist
- ESP_ERR_INVALID_ARG : Null instance/name arguments

Parameters

- `pc`: Pointer to the protocomm instance
- `ep_name`: Endpoint identifier(name) string

Type Definitions

typedef *esp_err_t* (***protocomm_req_handler_t**)(uint32_t session_id, **const** uint8_t *inbuf, ssize_t inlen, uint8_t **outbuf, ssize_t *outlen, void *priv_data)

Function prototype for protocomm endpoint handler.

typedef struct protocomm **protocomm_t**

This structure corresponds to a unique instance of protocomm returned when the API `protocomm_new()` is called. The remaining Protocomm APIs require this object as the first parameter.

Note Structure of the protocomm object is kept private

Header File

- protocomm/include/security/protocomm_security.h

Structures

struct protocomm_security_pop

Proof Of Possession for authenticating a secure session.

Public Members

const uint8_t ***data**

Pointer to buffer containing the proof of possession data

uint16_t **len**

Length (in bytes) of the proof of possession data

struct protocomm_security

Protocomm security object structure.

The member functions are used for implementing secure protocomm sessions.

Note This structure should not have any dynamic members to allow re-entrancy

Public Members

int **ver**

Unique version number of security implementation

esp_err_t (***init**)(*protocomm_security_handle_t* *handle)

Function for initializing/allocating security infrastructure

esp_err_t (***cleanup**)(*protocomm_security_handle_t* handle)
Function for deallocating security infrastructure

esp_err_t (***new_transport_session**)(*protocomm_security_handle_t* handle, uint32_t session_id)
Starts new secure transport session with specified ID

esp_err_t (***close_transport_session**)(*protocomm_security_handle_t* handle, uint32_t session_id)
Closes a secure transport session with specified ID

esp_err_t (***security_req_handler**)(*protocomm_security_handle_t* handle, **const** *protocomm_security_pop_t* *pop, uint32_t session_id, **const** uint8_t *inbuf, ssize_t inlen, uint8_t **outbuf, ssize_t *outlen, void *priv_data)
Handler function for authenticating connection request and establishing secure session

esp_err_t (***encrypt**)(*protocomm_security_handle_t* handle, uint32_t session_id, **const** uint8_t *inbuf, ssize_t inlen, uint8_t *outbuf, ssize_t *outlen)
Function which implements the encryption algorithm

esp_err_t (***decrypt**)(*protocomm_security_handle_t* handle, uint32_t session_id, **const** uint8_t *inbuf, ssize_t inlen, uint8_t *outbuf, ssize_t *outlen)
Function which implements the decryption algorithm

Type Definitions

typedef struct *protocomm_security_pop* **protocomm_security_pop_t**
Proof Of Possession for authenticating a secure session.

typedef void ***protocomm_security_handle_t**

typedef struct *protocomm_security* **protocomm_security_t**
Protocomm security object structure.

The member functions are used for implementing secure protocomm sessions.

Note This structure should not have any dynamic members to allow re-entrancy

Header File

- protocomm/include/security/protocomm_security0.h

Header File

- protocomm/include/security/protocomm_security1.h

Header File

- protocomm/include/transports/protocomm_httpd.h

Functions

esp_err_t **protocomm_httpd_start**(*protocomm_t* *pc, **const** *protocomm_httpd_config_t* *config)

Start HTTPD protocomm transport.

This API internally creates a framework to allow endpoint registration and security configuration for the protocomm.

Note This is a singleton. ie. Protocomm can have multiple instances, but only one instance can be bound to an HTTP transport layer.

Return

- ESP_OK : Success
- ESP_ERR_INVALID_ARG : Null arguments
- ESP_ERR_NOT_SUPPORTED : Transport layer bound to another protocomm instance
- ESP_ERR_INVALID_STATE : Transport layer already bound to this protocomm instance
- ESP_ERR_NO_MEM : Memory allocation for server resource failed
- ESP_ERR_HTTPD_* : HTTP server error on start

Parameters

- `pc`: Protocomm instance pointer obtained from protocomm_new()
- `config`: Pointer to config structure for initializing HTTP server

esp_err_t **protocomm_httpd_stop**(*protocomm_t* *pc)

Stop HTTPD protocomm transport.

This API cleans up the HTTPD transport protocomm and frees all the handlers registered with the protocomm.

Return

- ESP_OK : Success
- ESP_ERR_INVALID_ARG : Null / incorrect protocomm instance pointer

Parameters

- `pc`: Same protocomm instance that was passed to protocomm_httpd_start()

Unions

union protocomm_httpd_config_data_t

#include <protocomm_httpd.h> Protocomm HTTPD Configuration Data

Public Members

void *handle

HTTP Server Handle, if ext_handle_provided is set to true

protocomm_http_server_config_t config

HTTP Server Configuration, if a server is not already active

Structures

struct protocomm_http_server_config_t

Config parameters for protocomm HTTP server.

Public Members

uint16_t port

Port on which the HTTP server will listen

size_t stack_size

Stack size of server task, adjusted depending upon stack usage of endpoint handler

unsigned task_priority

Priority of server task

struct protocomm_httpd_config_t

Config parameters for protocomm HTTP server.

Public Members

bool ext_handle_provided

Flag to indicate of an external HTTP Server Handle has been provided. In such as case, protocomm will use the same HTTP Server and not start a new one internally.

protocomm_httpd_config_data_t data

Protocomm HTTPD Configuration Data

Macros

PROTOCOMM_HTTPD_DEFAULT_CONFIG()

Header File

- protocomm/include/transports/protocomm_ble.h

Functions

esp_err_t **protocomm_ble_start**(*protocomm_t* *pc, **const** *protocomm_ble_config_t* *config)

Start Bluetooth Low Energy based transport layer for provisioning.

Initialize and start required BLE service for provisioning. This includes the initialization for characteristics/service for BLE.

Return

- ESP_OK : Success
- ESP_FAIL : Simple BLE start error
- ESP_ERR_NO_MEM : Error allocating memory for internal resources
- ESP_ERR_INVALID_STATE : Error in ble config
- ESP_ERR_INVALID_ARG : Null arguments

Parameters

- `pc`: Protocomm instance pointer obtained from protocomm_new()
- `config`: Pointer to config structure for initializing BLE

esp_err_t **protocomm_ble_stop**(*protocomm_t* *pc)

Stop Bluetooth Low Energy based transport layer for provisioning.

Stops service/task responsible for BLE based interactions for provisioning

Note You might want to optionally reclaim memory from Bluetooth. Refer to the documentation of `esp_bt_mem_release` in that case.

Return

- ESP_OK : Success
- ESP_FAIL : Simple BLE stop error
- ESP_ERR_INVALID_ARG : Null / incorrect protocomm instance

Parameters

- `pc`: Same protocomm instance that was passed to protocomm_ble_start()

Structures

struct name_uuid

This structure maps handler required by protocomm layer to UUIDs which are used to uniquely identify BLE characteristics from a smartphone or a similar client device.

Public Members

const char ***name**

Name of the handler, which is passed to protocomm layer

uint16_t **uuid**

UUID to be assigned to the BLE characteristic which is mapped to the handler

struct protocomm_ble_config

Config parameters for protocomm BLE service.

Public Members

char **device_name**[**MAX_BLE_DEVNAME_LEN**]

BLE device name being broadcast at the time of provisioning

uint8_t **service_uuid**[**BLE_UUID128_VAL_LENGTH**]

128 bit UUID of the provisioning service

ssize_t **nu_lookup_count**

Number of entries in the Name-UUID lookup table

protocomm_ble_name_uuid_t ***nu_lookup**

Pointer to the Name-UUID lookup table

Macros

MAX_BLE_DEVNAME_LEN

BLE device name cannot be larger than this value 31 bytes (max scan response size) - 1 byte (length) - 1 byte (type) = 29 bytes

BLE_UUID128_VAL_LENGTH

Type Definitions

typedef struct *name_uuid* **protocomm_ble_name_uuid_t**

This structure maps handler required by protocomm layer to UUIDs which are used to uniquely identify BLE characteristics from a smartphone or a similar client device.

`typedef struct` *protocomm_ble_config* `protocomm_ble_config_t`

Config parameters for protocomm BLE service.

2.5.3 Wi-Fi Provisioning

Overview

This component provides APIs that control Wi-Fi provisioning service for receiving and configuring Wi-Fi credentials over SoftAP or BLE transport via secure *Protocol Communication (protocomm)* sessions. The set of `wifi_prov_mgr_` APIs help in quickly implementing a provisioning service having necessary features with minimal amount of code and sufficient flexibility.

Initialization

wifi_prov_mgr_init() is called to configure and initialize the provisioning manager and thus this must be called prior to invoking any other `wifi_prov_mgr_` APIs. Note that the manager relies on other components of IDF, namely NVS, TCP/IP, Event Loop and Wi-Fi (and optionally mDNS), hence these must be initialized beforehand. The manager can be de-initialized at any moment by making a call to *wifi_prov_mgr_deinit()*.

```
wifi_prov_mgr_config_t config = {
    .scheme = wifi_prov_scheme_ble,
    .scheme_event_handler = WIFI_PROV_SCHEME_BLE_EVENT_HANDLER_FREE_BTDM
};

ESP_ERR_CHECK( wifi_prov_mgr_init(config) );
```

The configuration structure `wifi_prov_mgr_config_t` has a few fields to specify the behavior desired of the manager :

- *scheme* : This is used to specify the provisioning scheme. Each scheme corresponds to one of the modes of transport supported by protocomm. Hence, we have three options :
  - `wifi_prov_scheme_ble` : BLE transport and GATT Server for handling provisioning commands
  - `wifi_prov_scheme_softap` : Wi-Fi SoftAP transport and HTTP Server for handling provisioning commands
  - `wifi_prov_scheme_console` : Serial transport and console for handling provisioning commands
- *scheme_event_handler* : An event handler defined along with scheme. Choosing appropriate scheme specific event handler allows the manager to take care of certain matters automatically. Presently this is not used for either SoftAP or Console based provisioning, but is very convenient for BLE. To understand how, we must recall that Bluetooth requires quite

some amount of memory to function and once provisioning is finished, the main application may want to reclaim back this memory (or part of it, if it needs to use either BLE or classic BT). Also, upon every future reboot of a provisioned device, this reclamation of memory needs to be performed again. To reduce this complication in using `wifi_prov_scheme_ble`, the scheme specific handlers have been defined, and depending upon the chosen handler, the BLE / classic BT / BTDM memory will be freed automatically when the provisioning manager is de-initialized. The available options are:

  - `WIFI_PROV_SCHEME_BLE_EVENT_HANDLER_FREE_BTDM` - Free both classic BT and BLE (BTDM) memory. Used when main application doesn' t require Bluetooth at all.
  - `WIFI_PROV_SCHEME_BLE_EVENT_HANDLER_FREE_BLE` - Free only BLE memory. Used when main application requires classic BT.
  - `WIFI_PROV_SCHEME_BLE_EVENT_HANDLER_FREE_BT` - Free only classic BT. Used when main application requires BLE. In this case freeing happens right when the manager is initialized.
  - `WIFI_PROV_EVENT_HANDLER_NONE` - Don' t use any scheme specific handler. Used when provisioning scheme is not BLE (i.e. SoftAP or Console), or when main application wants to handle the memory reclaiming on its own, or needs both BLE and classic BT to function.

- *app_event_handler* (Deprecated) : It is now recommended to catch `WIFI_PROV_EVENT``s that are emitted to the default event loop handler. See definition of ``wifi_prov_cb_event_t` for the list of events that are generated by the provisioning service. Here is an excerpt showing some of the provisioning events:

```
static void event_handler(void* arg, esp_event_base_t event_base,
                          int event_id, void* event_data)
{
    if (event_base == WIFI_PROV_EVENT) {
        switch (event_id) {
            case WIFI_PROV_START:
                ESP_LOGI(TAG, "Provisioning started");
                break;
            case WIFI_PROV_CRED_RECV: {
                wifi_sta_config_t *wifi_sta_cfg = (wifi_sta_config_t *)event_
↪data;
                ESP_LOGI(TAG, "Received Wi-Fi credentials"
                         "\n\tSSID     : %s\n\tPassword : %s",
                         (const char *) wifi_sta_cfg->ssid,
                         (const char *) wifi_sta_cfg->password);
                break;
```

(下页继续)

(续上页)

```
            }
            case WIFI_PROV_CRED_FAIL: {
                wifi_prov_sta_fail_reason_t *reason = (wifi_prov_sta_fail_
↪reason_t *)event_data;
                ESP_LOGE(TAG, "Provisioning failed!\n\tReason : %s"
                         "\n\tPlease reset to factory and retry provisioning",
                         (*reason == WIFI_PROV_STA_AUTH_ERROR) ?
                         "Wi-Fi station authentication failed" : "Wi-Fi access-
↪point not found");
                break;
            }
            case WIFI_PROV_CRED_SUCCESS:
                ESP_LOGI(TAG, "Provisioning successful");
                break;
            case WIFI_PROV_END:
                /* De-initialize manager once provisioning is finished */
                wifi_prov_mgr_deinit();
                break;
            default:
                break;
        }
    }
}
```

The manager can be de-initialized at any moment by making a call to *wifi_prov_mgr_deinit()*.

Check Provisioning State

Whether device is provisioned or not can be checked at runtime by calling *wifi_prov_mgr_is_provisioned()*. This internally checks if the Wi-Fi credentials are stored in NVS.

Note that presently manager does not have its own NVS namespace for storage of Wi-Fi credentials, instead it relies on the `esp_wifi_` APIs to set and get the credentials stored in NVS from the default location.

If provisioning state needs to be reset, any of the following approaches may be taken :

- the associated part of NVS partition has to be erased manually
- main application must implement some logic to call `esp_wifi_` APIs for erasing the credentials at runtime
- main application must implement some logic to force start the provisioning irrespective of

the provisioning state

```
bool provisioned = false;
ESP_ERR_CHECK( wifi_prov_mgr_is_provisioned(&provisioned) );
```

Start Provisioning Service

At the time of starting provisioning we need to specify a service name and the corresponding key. These translate to :

- Wi-Fi SoftAP SSID and passphrase, respectively, when scheme is `wifi_prov_scheme_softap`
- BLE Device name (service key is ignored) when scheme is `wifi_prov_scheme_ble`

Also, since internally the manager uses *protocomm*, we have the option of choosing one of the security features provided by it :

- Security 1 is secure communication which consists of a prior handshake involving X25519 key exchange along with authentication using a proof of possession (*pop*), followed by AES-CTR for encryption/decryption of subsequent messages
- Security 0 is simply plain text communication. In this case the *pop* is simply ignored

See *Provisioning* for details about the security features.

```
const char *service_name = "my_device";
const char *service_key  = "password";

wifi_prov_security_t security = WIFI_PROV_SECURITY_1;
const char *pop = "abcd1234";

ESP_ERR_CHECK( wifi_prov_mgr_start_provisioning(security, pop, service_name,␣
↪service_key) );
```

The provisioning service will automatically finish only if it receives valid Wi-Fi AP credentials followed by successfully connection of device to the AP (IP obtained). Regardless of that, the provisioning service can be stopped at any moment by making a call to *`wifi_prov_mgr_stop_provisioning()`*.

注解: If the device fails to connect with the provided credentials, it won't accept new credentials anymore, but the provisioning service will keep on running (only to convey failure to the client), until the device is restarted. Upon restart the provisioning state will turn out to be true this time (as credentials will be found in NVS), but device will again fail to connect with those same credentials (unless an AP with the matching credentials somehow does become available). This situation can be fixed by resetting the credentials in NVS or force starting the provisioning service. This has been explained above in *Check Provisioning State*.

Waiting For Completion

Typically, the main application will wait for the provisioning to finish, then de-initialize the manager to free up resources and finally start executing its own logic.

There are two ways for making this possible. The simpler way is to use a blocking call to *wifi_prov_mgr_wait()*.

```
// Start provisioning service
ESP_ERR_CHECK( wifi_prov_mgr_start_provisioning(security, pop, service_name,␣
↪service_key) );

// Wait for service to complete
wifi_prov_mgr_wait();

// Finally de-initialize the manager
wifi_prov_mgr_deinit();
```

The other way is to use the default event loop handler to catch `WIFI_PROV_EVENT``s and call :cpp:func:`wifi_prov_mgr_deinit()` when event ID is ``WIFI_PROV_END`:

```
static void event_handler(void* arg, esp_event_base_t event_base,
                          int event_id, void* event_data)
{
    if (event_base == WIFI_PROV_EVENT && event_id == WIFI_PROV_END) {
        /* De-initialize manager once provisioning is finished */
        wifi_prov_mgr_deinit();
    }
}
```

User Side Implementation

When the service is started, the device to be provisioned is identified by the advertised service name which, depending upon the selected transport, is either the BLE device name or the SoftAP SSID.

When using SoftAP transport, for allowing service discovery, mDNS must be initialized before starting provisioning. In this case the hostname set by the main application is used, and the service type is internally set to *_esp_wifi_prov*.

When using BLE transport, a custom 128 bit UUID should be set using *wifi_prov_scheme_ble_set_service_uuid()*. This UUID will be included in the BLE advertisement and will correspond to the primary GATT service that provides provisioning endpoints as GATT characteristics. Each GATT characteristic will be formed using the primary service UUID as base, with

different auto assigned 12th and 13th bytes (assume counting starts from 0th byte). Since, an endpoint characteristic UUID is auto assigned, it shouldn' t be used to identify the endpoint. Instead, client side applications should identify the endpoints by reading the User Characteristic Description (0x2901) descriptor for each characteristic, which contains the endpoint name of the characteristic. For example, if the service UUID is set to *55cc035e-fb27-4f80-be02-3c60828b7451*, each endpoint characteristic will be assigned a UUID like *55cc_____-fb27-4f80-be02-3c60828b7451*, with unique values at the 12th and 13th bytes.

Once connected to the device, the provisioning related protocomm endpoints can be identified as follows :

表 5: Endpoints provided by Provisioning Service

Endpoint Name (BLE + GATT Server)	URI (SoftAP + HTTP Server + mDNS)	Description
prov-session	http://<mdns-hostname>.local/prov-session	Security endpoint used for session establishment
prov-scan	http://wifi-prov.local/prov-scan	Endpoint used for starting Wi-Fi scan and receiving scan results
prov-config	http://<mdns-hostname>.local/prov-config	Endpoint used for configuring Wi-Fi credentials on device
proto-ver	http://<mdns-hostname>.local/proto-ver	Endpoint for retrieving version info

Immediately after connecting, the client application may fetch the version / capabilities information from the *proto-ver* endpoint. All communications to this endpoint are un-encrypted, hence necessary information (that may be relevant for deciding compatibility) can be retrieved before establishing a secure session. The response is in JSON format and looks like : `prov: { ver: v1.1, cap: [no_pop] }, my_app: { ver: 1.345, cap: [cloud, local_ctrl] },.....` Here label *prov* provides provisioning service version (*ver*) and capabilities (*cap*). For now, only *no_pop* capability is supported, which indicates that the service doesn' t require proof of possession for authentication. Any application related version / capabilities will be given by other labels (like *my_app* in this example). These additional fields are set using *wifi_prov_mgr_set_app_info()*.

User side applications need to implement the signature handshaking required for establishing and authenticating secure protocomm sessions as per the security scheme configured for use (this is not needed when manager is configured to use protocomm security 0).

See Unified Provisioning for more details about the secure handshake and encryption used. Applications must use the *.proto* files found under protocomm/proto, which define the Protobuf message structures supported

by *prov-session* endpoint.

Once a session is established, Wi-Fi credentials are configured using the following set of *wifi_config* commands, serialized as Protobuf messages (the corresponding *.proto* files can be found under wifi_provisioning/proto) :

- *get_status* - For querying the Wi-Fi connection status. The device will respond with a status which will be one of connecting / connected / disconnected. If status is disconnected, a disconnection reason will also be included in the status response.
- *set_config* - For setting the Wi-Fi connection credentials
- *apply_config* - For applying the credentials saved during *set_config* and start the Wi-Fi station

After session establishment, client can also request Wi-Fi scan results from the device. The results returned is a list of AP SSIDs, sorted in descending order of signal strength. This allows client applications to display APs nearby to the device at the time of provisioning, and users can select one of the SSIDs and provide the password which is then sent using the *wifi_config* commands described above. The *wifi_scan* endpoint supports the following protobuf commands :

- *scan_start* - For starting Wi-Fi scan with various options :
  - *blocking* (input) - If true, the command returns only when the scanning is finished
  - *passive* (input) - If true scan is started in passive mode (this may be slower) instead of active mode
  - *group_channels* (input) - This specifies whether to scan all channels in one go (when zero) or perform scanning of channels in groups, with 120ms delay between scanning of consecutive groups, and the value of this parameter sets the number of channels in each group. This is useful when transport mode is SoftAP, where scanning all channels in one go may not give the Wi-Fi driver enough time to send out beacons, and hence may cause disconnection with any connected stations. When scanning in groups, the manager will wait for atleast 120ms after completing scan on a group of channels, and thus allow the driver to send out the beacons. For example, given that the total number of Wi-Fi channels is 14, then setting group_channels to 4, will create 5 groups, with each group having 3 channels, except the last one which will have $14 \% 3 = 2$ channels. So, when scan is started, the first 3 channels will be scanned, followed by a 120ms delay, and then the next 3 channels, and so on, until all the 14 channels have been scanned. One may need to adjust this parameter as having only few channels in a group may slow down the overall scan time, while having too many may again cause disconnection. Usually a value of 4 should work for most cases. Note that for any other mode of transport, e.g. BLE, this can be safely set to 0, and hence achieve the fastest overall scanning time.
  - *period_ms* (input) - Scan parameter specifying how long to wait on each channel
- *scan_status* - Gives the status of scanning process :
  - *scan_finished* (output) - When scan has finished this returns true

  - *result_count* (output) - This gives the total number of results obtained till now. If scan is yet happening this number will keep on updating

- *scan_result* - For fetching scan results. This can be called even if scan is still on going
  - *start_index* (input) - Starting index from where to fetch the entries from the results list
  - *count* (input) - Number of entries to fetch from the starting index
  - *entries* (output) - List of entries returned. Each entry consists of *ssid*, *channel* and *rssi* information

Additional Endpoints

In case users want to have some additional protocomm endpoints customized to their requirements, this is done in two steps. First is creation of an endpoint with a specific name, and the second step is the registration of a handler for this endpoint. See *protocomm* for the function signature of an endpoint handler. A custom endpoint must be created after initialization and before starting the provisioning service. Whereas, the protocomm handler is registered for this endpoint only after starting the provisioning service.

```
wifi_prov_mgr_init(config);
wifi_prov_mgr_endpoint_create("custom-endpoint");
wifi_prov_mgr_start_provisioning(security, pop, service_name, service_key);
wifi_prov_mgr_endpoint_register("custom-endpoint", custom_ep_handler, custom_
↪ep_data);
```

When the provisioning service stops, the endpoint is unregistered automatically.

One can also choose to call *wifi_prov_mgr_endpoint_unregister()* to manually deactivate an endpoint at runtime. This can also be used to deactivate the internal endpoints used by the provisioning service.

When / How To Stop Provisioning Service?

The default behavior is that once the device successfully connects using the Wi-Fi credentials set by the *apply_config* command, the provisioning service will be stopped (and BLE / SoftAP turned off) automatically after responding to the next *get_status* command. If *get_status* command is not received by the device, the service will be stopped after a 30s timeout.

On the other hand, if device was not able to connect using the provided Wi-Fi credentials, due to incorrect SSID / passphrase, the service will keep running, and *get_status* will keep responding with disconnected status and reason for disconnection. Any further attempts to provide another set of Wi-Fi credentials, will be rejected. These credentials will be preserved, unless the provisioning service is force started, or NVS erased.

If this default behavior is not desired, it can be disabled by calling *wifi_prov_mgr_disable_auto_stop()*. Now the provisioning service will only be stopped after an explicit call to

wifi_prov_mgr_stop_provisioning(), which returns immediately after scheduling a task for stopping the service. The service stops after a certain delay and WIFI_PROV_END event gets emitted. This delay is specified by the argument to *wifi_prov_mgr_disable_auto_stop()*.

The customized behavior is useful for applications which want the provisioning service to be stopped some time after the Wi-Fi connection is successfully established. For example, if the application requires the device to connect to some cloud service and obtain another set of credentials, and exchange this credentials over a custom protocomm endpoint, then after successfully doing so stop the provisioning service by calling *wifi_prov_mgr_stop_provisioning()* inside the protocomm handler itself. The right amount of delay ensures that the transport resources are freed only after the response from the protocomm handler reaches the client side application.

Application Examples

For complete example implementation see provisioning/manager

API Reference

Header File

- wifi_provisioning/include/wifi_provisioning/manager.h

Functions

esp_err_t **wifi_prov_mgr_init**(*wifi_prov_mgr_config_t config*)

Initialize provisioning manager instance.

Configures the manager and allocates internal resources

Configuration specifies the provisioning scheme (transport) and event handlers

Event WIFI_PROV_INIT is emitted right after initialization is complete

Return

- ESP_OK : Success
- ESP_FAIL : Fail

Parameters

- `config`: Configuration structure

void **wifi_prov_mgr_deinit**(void)

Stop provisioning (if running) and release resource used by the manager.

Event WIFI_PROV_DEINIT is emitted right after de-initialization is finished

If provisioning service is still active when this API is called, it first stops the service, hence emitting WIFI_PROV_END, and then performs the de-initialization

esp_err_t **wifi_prov_mgr_is_provisioned**(bool **provisioned*)

Checks if device is provisioned.

This checks if Wi-Fi credentials are present on the NVS

The Wi-Fi credentials are assumed to be kept in the same NVS namespace as used by esp_wifi component

If one were to call esp_wifi_set_config() directly instead of going through the provisioning process, this function will still yield true (i.e. device will be found to be provisioned)

Note Calling wifi_prov_mgr_start_provisioning() automatically resets the provision state, irrespective of what the state was prior to making the call.

Return

- ESP_OK : Retrieved provision state successfully
- ESP_FAIL : Wi-Fi not initialized
- ESP_ERR_INVALID_ARG : Null argument supplied
- ESP_ERR_INVALID_STATE : Manager not initialized

Parameters

- **provisioned**: True if provisioned, else false

esp_err_t **wifi_prov_mgr_start_provisioning**(*wifi_prov_security_t* *security*, **const** char **pop*, **const** char **service_name*, **const** char **service_key*)

Start provisioning service.

This starts the provisioning service according to the scheme configured at the time of initialization. For scheme :

- wifi_prov_scheme_ble : This starts protocomm_ble, which internally initializes BLE transport and starts GATT server for handling provisioning requests
- wifi_prov_scheme_softap : This activates SoftAP mode of Wi-Fi and starts protocomm_httpd, which internally starts an HTTP server for handling provisioning requests (If mDNS is active it also starts advertising service with type _esp_wifi_prov._tcp)

Event WIFI_PROV_START is emitted right after provisioning starts without failure

Note This API will start provisioning service even if device is found to be already provisioned, i.e. wifi_prov_mgr_is_provisioned() yields true

Return

- ESP_OK : Provisioning started successfully
- ESP_FAIL : Failed to start provisioning service
- ESP_ERR_INVALID_STATE : Provisioning manager not initialized or already started

Parameters

- `security`: Specify which protocomm security scheme to use :
  - WIFI_PROV_SECURITY_0 : For no security
  - WIFI_PROV_SECURITY_1 : x25519 secure handshake for session establishment followed by AES-CTR encryption of provisioning messages
- `pop`: Pointer to proof of possession string (NULL if not needed). This is relevant only for protocomm security 1, in which case it is used for authenticating secure session
- `service_name`: Unique name of the service. This translates to:
  - Wi-Fi SSID when provisioning mode is softAP
  - Device name when provisioning mode is BLE
- `service_key`: Key required by client to access the service (NULL if not needed). This translates to:
  - Wi-Fi password when provisioning mode is softAP
  - ignored when provisioning mode is BLE

void **wifi_prov_mgr_stop_provisioning**(void)

Stop provisioning service.

If provisioning service is active, this API will initiate a process to stop the service and return. Once the service actually stops, the event WIFI_PROV_END will be emitted.

If wifi_prov_mgr_deinit() is called without calling this API first, it will automatically stop the provisioning service and emit the WIFI_PROV_END, followed by WIFI_PROV_DEINIT, before returning.

This API will generally be used along with wifi_prov_mgr_disable_auto_stop() in the scenario when the main application has registered its own endpoints, and wishes that the provisioning service is stopped only when some protocomm command from the client side application is received.

Calling this API inside an endpoint handler, with sufficient cleanup_delay, will allow the response / acknowledgment to be sent successfully before the underlying protocomm service is stopped.

Cleaup_delay is set when calling wifi_prov_mgr_disable_auto_stop(). If not specified, it defaults to 1000ms.

For straightforward cases, using this API is usually not necessary as provisioning is stopped automatically once WIFI_PROV_CRED_SUCCESS is emitted. Stopping is delayed (maximum 30 seconds)

thus allowing the client side application to query for Wi-Fi state, i.e. after receiving the first query and sending `Wi-Fi state connected` response the service is stopped immediately.

void **wifi_prov_mgr_wait**(void)

Wait for provisioning service to finish.

Calling this API will block until provisioning service is stopped i.e. till event WIFI_PROV_END is emitted.

This will not block if provisioning is not started or not initialized.

esp_err_t **wifi_prov_mgr_disable_auto_stop**(uint32_t *cleanup_delay*)

Disable auto stopping of provisioning service upon completion.

By default, once provisioning is complete, the provisioning service is automatically stopped, and all endpoints (along with those registered by main application) are deactivated.

This API is useful in the case when main application wishes to close provisioning service only after it receives some protocomm command from the client side app. For example, after connecting to Wi-Fi, the device may want to connect to the cloud, and only once that is successfully, the device is said to be fully configured. But, then it is upto the main application to explicitly call wifi_prov_mgr_stop_provisioning() later when the device is fully configured and the provisioning service is no longer required.

Note This must be called before executing wifi_prov_mgr_start_provisioning()

Return

- ESP_OK : Success
- ESP_ERR_INVALID_STATE : Manager not initialized or provisioning service already started

Parameters

- `cleanup_delay`: Sets the delay after which the actual cleanup of transport related resources is done after a call to wifi_prov_mgr_stop_provisioning() returns. Minimum allowed value is 100ms. If not specified, this will default to 1000ms.

esp_err_t **wifi_prov_mgr_set_app_info**(**const** char ***label*, **const** char ***version*, **const** char ***capabilities*, size_t *total_capabilities*)

Set application version and capabilities in the JSON data returned by proto-ver endpoint.

This function can be called multiple times, to specify information about the various application specific services running on the device, identified by unique labels.

The provisioning service itself registers an entry in the JSON data, by the label "prov", containing only provisioning service version and capabilities. Application services should use a label other than "prov" so as not to overwrite this.

Note This must be called before executing wifi_prov_mgr_start_provisioning()

Return

- ESP_OK : Success
- ESP_ERR_INVALID_STATE : Manager not initialized or provisioning service already started
- ESP_ERR_NO_MEM : Failed to allocate memory for version string
- ESP_ERR_INVALID_ARG : Null argument

Parameters

- `label`: String indicating the application name.
- `version`: String indicating the application version. There is no constraint on format.
- `capabilities`: Array of strings with capabilities. These could be used by the client side app to know the application registered endpoint capabilities
- `total_capabilities`: Size of capabilities array

esp_err_t **wifi_prov_mgr_endpoint_create**(**const** char **ep_name*)

Create an additional endpoint and allocate internal resources for it.

This API is to be called by the application if it wants to create an additional endpoint. All additional endpoints will be assigned UUIDs starting from 0xFF54 and so on in the order of execution.

protocomm handler for the created endpoint is to be registered later using wifi_prov_mgr_endpoint_register() after provisioning has started.

Note This API can only be called BEFORE provisioning is started

Note Additional endpoints can be used for configuring client provided parameters other than Wi-Fi credentials, that are necessary for the main application and hence must be set prior to starting the application

Note After session establishment, the additional endpoints must be targeted first by the client side application before sending Wi-Fi configuration, because once Wi-Fi configuration finishes the provisioning service is stopped and hence all endpoints are unregistered

Return

- ESP_OK : Success
- ESP_FAIL : Failure

Parameters

- `ep_name`: unique name of the endpoint

esp_err_t **wifi_prov_mgr_endpoint_register**(**const** char **ep_name*, *protocomm_req_handler_t handler*, void **user_ctx*)

Register a handler for the previously created endpoint.

This API can be called by the application to register a protocomm handler to any endpoint that was created using wifi_prov_mgr_endpoint_create().

Note This API can only be called AFTER provisioning has started

Note Additional endpoints can be used for configuring client provided parameters other than Wi-Fi credentials, that are necessary for the main application and hence must be set prior to starting the application

Note After session establishment, the additional endpoints must be targeted first by the client side application before sending Wi-Fi configuration, because once Wi-Fi configuration finishes the provisioning service is stopped and hence all endpoints are unregistered

Return

- ESP_OK : Success
- ESP_FAIL : Failure

Parameters

- `ep_name`: Name of the endpoint
- `handler`: Endpoint handler function
- `user_ctx`: User data

void **wifi_prov_mgr_endpoint_unregister**(**const** char *_ep_name_)

Unregister the handler for an endpoint.

This API can be called if the application wants to selectively unregister the handler of an endpoint while the provisioning is still in progress.

All the endpoint handlers are unregistered automatically when the provisioning stops.

Parameters

- `ep_name`: Name of the endpoint

esp_err_t **wifi_prov_mgr_event_handler**(void *_ctx_, *system_event_t* *_event_)

Event handler for provisioning manager.

This is called from the main event handler and controls the provisioning manager' s internal state machine depending on incoming Wi-Fi events

Note : This function is DEPRECATED, because events are now handled internally using the event loop library, esp_event. Calling this will do nothing and simply return ESP_OK.

Return

- ESP_OK : Event handled successfully

Parameters

- `ctx`: Event context data
- `event`: Event info

esp_err_t **wifi_prov_mgr_get_wifi_state**(*wifi_prov_sta_state_t *state*)

Get state of Wi-Fi Station during provisioning.

Return

- ESP_OK : Successfully retrieved Wi-Fi state
- ESP_FAIL : Provisioning app not running

Parameters

- `state`: Pointer to wifi_prov_sta_state_t variable to be filled

esp_err_t **wifi_prov_mgr_get_wifi_disconnect_reason**(*wifi_prov_sta_fail_reason_t *reason*)

Get reason code in case of Wi-Fi station disconnection during provisioning.

Return

- ESP_OK : Successfully retrieved Wi-Fi disconnect reason
- ESP_FAIL : Provisioning app not running

Parameters

- `reason`: Pointer to wifi_prov_sta_fail_reason_t variable to be filled

esp_err_t **wifi_prov_mgr_configure_sta**(*wifi_config_t *wifi_cfg*)

Runs Wi-Fi as Station with the supplied configuration.

Configures the Wi-Fi station mode to connect to the AP with SSID and password specified in config structure and sets Wi-Fi to run as station.

This is automatically called by provisioning service upon receiving new credentials.

If credentials are to be supplied to the manager via a different mode other than through protocomm, then this API needs to be called.

Event WIFI_PROV_CRED_RECV is emitted after credentials have been applied and Wi-Fi station started

Return

- ESP_OK : Wi-Fi configured and started successfully
- ESP_FAIL : Failed to set configuration

Parameters

- `wifi_cfg`: Pointer to Wi-Fi configuration structure

Structures

struct wifi_prov_event_handler_t

Event handler that is used by the manager while provisioning service is active.

Public Members

wifi_prov_cb_func_t **event_cb**

Callback function to be executed on provisioning events

void ***user_data**

User context data to pass as parameter to callback function

struct wifi_prov_scheme

Structure for specifying the provisioning scheme to be followed by the manager.

Note Ready to use schemes are available:

- wifi_prov_scheme_ble : for provisioning over BLE transport + GATT server
- wifi_prov_scheme_softap : for provisioning over SoftAP transport + HTTP server
- wifi_prov_scheme_console : for provisioning over Serial UART transport + Console (for debugging)

Public Members

esp_err_t (***prov_start**)(*protocomm_t* *pc, void *config)

Function which is to be called by the manager when it is to start the provisioning service associated with a protocomm instance and a scheme specific configuration

esp_err_t (***prov_stop**)(*protocomm_t* *pc)

Function which is to be called by the manager to stop the provisioning service previously associated with a protocomm instance

void *(***new_config**)(void)

Function which is to be called by the manager to generate a new configuration for the provisioning service, that is to be passed to *prov_start()*

void (***delete_config**)(void *config)

Function which is to be called by the manager to delete a configuration generated using *new_config()*

esp_err_t (***set_config_service**)(void *config, **const** char *service_name, **const** char *service_key)

Function which is to be called by the manager to set the service name and key values in the configuration structure

esp_err_t (***set_config_endpoint**)(void *config, **const** char *endpoint_name, uint16_t uuid)

Function which is to be called by the manager to set a protocomm endpoint with an identifying name and UUID in the configuration structure

wifi_mode_t **wifi_mode**

Sets mode of operation of Wi-Fi during provisioning This is set to :

- WIFI_MODE_APSTA for SoftAP transport
- WIFI_MODE_STA for BLE transport

struct wifi_prov_mgr_config_t

Structure for specifying the manager configuration.

Public Members

wifi_prov_scheme_t **scheme**

Provisioning scheme to use. Following schemes are already available:

- wifi_prov_scheme_ble : for provisioning over BLE transport + GATT server
- wifi_prov_scheme_softap : for provisioning over SoftAP transport + HTTP server + mDNS (optional)
- wifi_prov_scheme_console : for provisioning over Serial UART transport + Console (for debugging)

wifi_prov_event_handler_t **scheme_event_handler**

Event handler required by the scheme for incorporating scheme specific behavior while provisioning manager is running. Various options may be provided by the scheme for setting this field. Use WIFI_PROV_EVENT_HANDLER_NONE when not used. When using scheme wifi_prov_scheme_ble, the following options are available:

- WIFI_PROV_SCHEME_BLE_EVENT_HANDLER_FREE_BTDM
- WIFI_PROV_SCHEME_BLE_EVENT_HANDLER_FREE_BLE
- WIFI_PROV_SCHEME_BLE_EVENT_HANDLER_FREE_BT

wifi_prov_event_handler_t **app_event_handler**

Event handler that can be set for the purpose of incorporating application specific behavior. Use WIFI_PROV_EVENT_HANDLER_NONE when not used.

Macros

WIFI_PROV_EVENT_HANDLER_NONE

Event handler can be set to none if not used.

Type Definitions

typedef void (*wifi_prov_cb_func_t)(void *user_data, *wifi_prov_cb_event_t* event, void *event_data)

typedef struct *wifi_prov_scheme* wifi_prov_scheme_t

Structure for specifying the provisioning scheme to be followed by the manager.

Note Ready to use schemes are available:

- wifi_prov_scheme_ble : for provisioning over BLE transport + GATT server
- wifi_prov_scheme_softap : for provisioning over SoftAP transport + HTTP server
- wifi_prov_scheme_console : for provisioning over Serial UART transport + Console (for debugging)

typedef enum *wifi_prov_security* wifi_prov_security_t

Security modes supported by the Provisioning Manager.

These are same as the security modes provided by protocomm

Enumerations

enum wifi_prov_cb_event_t

Events generated by manager.

These events are generated in order of declaration and, for the stretch of time between initialization and de-initialization of the manager, each event is signaled only once

Values:

WIFI_PROV_INIT

Emitted when the manager is initialized

WIFI_PROV_START

Indicates that provisioning has started

WIFI_PROV_CRED_RECV

Emitted when Wi-Fi AP credentials are received via `protocomm` endpoint `wifi_config`. The event data in this case is a pointer to the corresponding *wifi_sta_config_t* structure

WIFI_PROV_CRED_FAIL

Emitted when device fails to connect to the AP of which the credentials were received earlier on event `WIFI_PROV_CRED_RECV`. The event data in this case is a pointer to the disconnection reason code with type `wifi_prov_sta_fail_reason_t`

`WIFI_PROV_CRED_SUCCESS`
Emitted when device successfully connects to the AP of which the credentials were received earlier on event `WIFI_PROV_CRED_RECV`

`WIFI_PROV_END`
Signals that provisioning service has stopped

`WIFI_PROV_DEINIT`
Signals that manager has been de-initialized

enum wifi_prov_security
Security modes supported by the Provisioning Manager.

These are same as the security modes provided by protocomm

Values:

`WIFI_PROV_SECURITY_0` = 0
No security (plain-text communication)

`WIFI_PROV_SECURITY_1`
This secure communication mode consists of X25519 key exchange

- proof of possession (pop) based authentication
- AES-CTR encryption

Header File

- wifi_provisioning/include/wifi_provisioning/scheme_ble.h

Functions

void **wifi_prov_scheme_ble_event_cb_free_btdm**(void *user_data, *wifi_prov_cb_event_t* event, void *event_data)

void **wifi_prov_scheme_ble_event_cb_free_ble**(void *user_data, *wifi_prov_cb_event_t* event, void *event_data)

void **wifi_prov_scheme_ble_event_cb_free_bt**(void *user_data, *wifi_prov_cb_event_t* event, void *event_data)

esp_err_t **wifi_prov_scheme_ble_set_service_uuid**(uint8_t *uuid128)
Set the 128 bit GATT service UUID used for provisioning.

This API is used to override the default 128 bit provisioning service UUID, which is 0000ffff-0000-1000-8000-00805f9b34fb.

This must be called before starting provisioning, i.e. before making a call to wifi_prov_mgr_start_provisioning(), otherwise the default UUID will be used.

Note The data being pointed to by the argument must be valid atleast till provisioning is started. Upon start, the manager will store an internal copy of this UUID, and this data can be freed or invalidated afterwords.

Return

- ESP_OK : Success
- ESP_ERR_INVALID_ARG : Null argument

Parameters

- `uuid128`: A custom 128 bit UUID

Macros

`WIFI_PROV_SCHEME_BLE_EVENT_HANDLER_FREE_BTDM`

`WIFI_PROV_SCHEME_BLE_EVENT_HANDLER_FREE_BLE`

`WIFI_PROV_SCHEME_BLE_EVENT_HANDLER_FREE_BT`

Header File

- wifi_provisioning/include/wifi_provisioning/scheme_softap.h

Functions

void **`wifi_prov_scheme_softap_set_httpd_handle`**(void *_handle_)

Provide HTTPD Server handle externally.

Useful in cases wherein applications need the webserver for some different operations, and do not want the wifi provisioning component to start/stop a new instance.

Note This API should be called before wifi_prov_mgr_start_provisioning()

Parameters

- `handle`: Handle to HTTPD server instance

Header File

- wifi_provisioning/include/wifi_provisioning/scheme_console.h

Header File

- wifi_provisioning/include/wifi_provisioning/wifi_config.h

Functions

esp_err_t **wifi_prov_config_data_handler**(uint32_t *session_id*, **const** uint8_t **inbuf*, ssize_t *inlen*, uint8_t ***outbuf*, ssize_t **outlen*, void **priv_data*)

Handler for receiving and responding to requests from master.

This is to be registered as the `wifi_config` endpoint handler (protocomm `protocomm_req_handler_t`) using `protocomm_add_endpoint()`

Structures

struct wifi_prov_sta_conn_info_t

WiFi STA connected status information.

Public Members

char **ip_addr**[IP4ADDR_STRLEN_MAX]

IP Address received by station

char **bssid**[6]

BSSID of the AP to which connection was estalished

char **ssid**[33]

SSID of the to which connection was estalished

uint8_t **channel**

Channel of the AP

uint8_t **auth_mode**

Authorization mode of the AP

struct wifi_prov_config_get_data_t

WiFi status data to be sent in response to `get_status` request from master.

Public Members

wifi_prov_sta_state_t **wifi_state**

WiFi state of the station

wifi_prov_sta_fail_reason_t **fail_reason**

Reason for disconnection (valid only when `wifi_state` is `WIFI_STATION_DISCONNECTED`)

wifi_prov_sta_conn_info_t **conn_info**

Connection information (valid only when `wifi_state` is `WIFI_STATION_CONNECTED`)

struct wifi_prov_config_set_data_t

WiFi config data received by slave during `set_config` request from master.

Public Members

char **ssid**[33]

SSID of the AP to which the slave is to be connected

char **password**[64]

Password of the AP

char **bssid**[6]

BSSID of the AP

uint8_t **channel**

Channel of the AP

struct wifi_prov_config_handlers

Internal handlers for receiving and responding to protocomm requests from master.

This is to be passed as priv_data for protocomm request handler (refer to `wifi_prov_config_data_handler()`) when calling `protocomm_add_endpoint()`.

Public Members

esp_err_t (***get_status_handler**)(*wifi_prov_config_get_data_t* *resp_data, *wifi_prov_ctx_t* **ctx)

Handler function called when connection status of the slave (in WiFi station mode) is requested

esp_err_t (***set_config_handler**)(**const** *wifi_prov_config_set_data_t* *req_data, *wifi_prov_ctx_t* **ctx)

Handler function called when WiFi connection configuration (eg. AP SSID, password, etc.) of the slave (in WiFi station mode) is to be set to user provided values

esp_err_t (***apply_config_handler**)(*wifi_prov_ctx_t* **ctx)

Handler function for applying the configuration that was set in `set_config_handler`. After applying the station may get connected to the AP or may fail to connect. The slave must be ready to convey the updated connection status information when `get_status_handler` is invoked again by the master.

wifi_prov_ctx_t ***ctx**

Context pointer to be passed to above handler functions upon invocation

Type Definitions

typedef struct wifi_prov_ctx **wifi_prov_ctx_t**

Type of context data passed to each get/set/apply handler function set in

wifi_prov_config_handlers structure.

This is passed as an opaque pointer, thereby allowing it be defined later in application code as per requirements.

typedef struct *wifi_prov_config_handlers* **wifi_prov_config_handlers_t**

Internal handlers for receiving and responding to protocomm requests from master.

This is to be passed as priv_data for protocomm request handler (refer to `wifi_prov_config_data_handler()`) when calling `protocomm_add_endpoint()`.

Enumerations

enum wifi_prov_sta_state_t

WiFi STA status for conveying back to the provisioning master.

Values:

WIFI_PROV_STA_CONNECTING

WIFI_PROV_STA_CONNECTED

WIFI_PROV_STA_DISCONNECTED

enum wifi_prov_sta_fail_reason_t

WiFi STA connection fail reason.

Values:

WIFI_PROV_STA_AUTH_ERROR

WIFI_PROV_STA_AP_NOT_FOUND

本部分的 API 示例代码存放在 ESP-IDF 示例项目的 provisioning 目录下。

2.6 存储 API

[English]

2.6.1 SPI Flash API

[English]

概述

SPI Flash 组件提供外部 flash 数据读取、写入、擦除和内存映射相关的 API 函数，同时也提供了更高层级的，面向分区的 API 函数（定义在分区表 中）。

与 ESP-IDF V4.0 之前的 API 不同，这一版 API 功能并不局限于主 SPI Flash 芯片（即运行程序的 SPI Flash 芯片）。使用不同的芯片指针，您可以通过 SPI0/1 或 HSPI/VSPI 总线访问外部 flash。

注解： ESP-IDF V4.0 之后的 flash API 不再是原子的。因此，如果 flash 操作地址有重叠，且写操作与读操作同时执行，读操作可能会返回一部分写入之前的数据，返回一部分写入之后的数据。

Kconfig 选项*CONFIG_SPI_FLASH_USE_LEGACY_IMPL* 可将 `spi_flash_*` 函数切换至 ESP-IDF V4.0 之前的实现。但是，如果同时使用新旧 API，代码量可能会增多。

即便未启用*CONFIG_SPI_FLASH_USE_LEGACY_IMPL*，加密读取和加密写入操作也均使用旧实现。因此，仅有主 flash 芯片支持加密操作，其他不同片选（经 SPI1 访问的 flash 芯片）则不支持加密操作。

初始化 Flash 设备

在使用 `esp_flash_*` API 之前，您需要在 SPI 总线上初始化芯片。

1. 调用*spi_bus_initialize()* 初始化 SPI 总线，此函数将初始化总线上设备间共享的资源，如 I/O、DMA 及中断等。
2. 调用*spi_bus_add_flash_device()* 将 flash 设备连接到总线上。然后分配内存，填充 `esp_flash_t` 结构体，同时初始化 CS I/O。
3. 调用*esp_flash_init()* 与芯片进行通信。后续操作会依据芯片类型不同而有差异。

注解： 目前，多个 flash 芯片可连接到同一总线。但尚不支持在同一个 SPI 总线上使用 `esp_flash_*` 和 `spi_device_*` 设备。

SPI Flash 访问 API

如下所示为处理 flash 中数据的函数集：

- *esp_flash_read()*：将数据从 flash 读取到 RAM；
- *esp_flash_write()*：将数据从 RAM 写入到 flash；
- *esp_flash_erase_region()*：擦除 flash 中指定区域的数据；
- *esp_flash_erase_chip()*：擦除整个 flash；
- `esp_flash_get_chip_size()`：返回 menuconfig 中设置的 flash 芯片容量（以字节为单位）。

一般来说，请尽量避免对主 SPI flash 芯片直接使用原始 SPI flash 函数，如需对主 SPI flash 芯片进行操作，请使用分区专用函数。

SPI Flash 容量

SPI flash 容量存储于引导程序映像头部（烧录偏移量为 0x1000）的一个字段。

默认情况下，引导程序写入 flash 时，esptool.py 将引导程序写入 flash 时，会自动检测 SPI flash 容量，同时使用正确容量更新引导程序的头部。您也可以在工程配置中设置 `CONFIG_ESPTOOLPY_FLASHSIZE`，生成固定的 flash 容量。

如需在运行时覆盖已配置的 flash 容量，请配置 `g_rom_flashchip` 结构中的 `chip_size`。`esp_flash_*` 函数使用此容量（于软件和 ROM 中）进行边界检查。

SPI1 Flash 并发约束

由于 SPI1 flash 也被用于执行固件（通过指令 cache 或数据 cache ），因此在执行读取、写入及擦除操作时，必须禁用这些 cache。这意味着在执行 flash 写操作时，两个 CPU 必须从 IRAM 运行代码，且只能从 DRAM 中读取数据。

如果您使用本文档中 API 函数，上述限制将自动生效且透明（无需您额外关注），但这些限制可能会影响系统中的其他任务的性能。

除 SPI0/1 以外的 SPI 总线上的其它 flash 芯片则不受这种限制。

请参阅*应用程序内存分布*，查看 IRAM、DRAM 和 flash cache 的区别。

为避免意外读取 flash cache，一个 CPU 在启动 flash 写入或擦除操作时，另一个 CPU 将阻塞，并且在 flash 操作完成前，两个 CPU 上的所有的非 IRAM 安全的中断都会被禁用。

IRAM 安全中断处理程序

如果您需要在 flash 操作期间运行中断处理程序（比如低延迟操作），请在*注册中断处理程序* 时设置 `ESP_INTR_FLAG_IRAM`。

请确保中断处理程序访问的所有数据和函数（包括其调用的数据和函数）都存储在 IRAM 或 DRAM 中。

为函数添加 `IRAM_ATTR` 属性:

```
#include "esp_attr.h"

void IRAM_ATTR gpio_isr_handler(void* arg)
{
    // ...
}
```

为常量添加 `DRAM_ATTR` 和 `DRAM_STR` 属性:

```
void IRAM_ATTR gpio_isr_handler(void* arg)
{
   const static DRAM_ATTR uint8_t INDEX_DATA[] = { 45, 33, 12, 0 };
   const static char *MSG = DRAM_STR("I am a string stored in RAM");
}
```

辨别哪些数据应标记为 DRAM_ATTR 可能会比较困难，除非明确标记为 DRAM_ATTR，否则编译器依然可能将某些变量或表达式当做常量（即便没有 const 标记），并将其放入 flash。

如果函数或符号未被正确放入 IRAM/DRAM 中，当中断处理程序在 flash 操作期间从 flash cache 中读取数据，则会产生非法指令异常（这是因为代码未被正确放入 IRAM）或读取垃圾数据（这是因为常数未被正确放入 DRAM），而导致崩溃。

分区表 API

ESP-IDF 工程使用分区表保存 SPI flash 各区信息，包括引导程序、各种应用程序二进制文件、数据及文件系统等。请参考分区表，查看详细信息。

该组件在 esp_partition.h 中声明了一些 API 函数，用以枚举在分区表中找到的分区，并对这些分区执行操作：

- *esp_partition_find()*：在分区表中查找特定类型的条目，返回一个不透明迭代器；
- *esp_partition_get()*：返回一个结构，描述给定迭代器的分区；
- *esp_partition_next()*：将迭代器移至下一个找到的分区；
- *esp_partition_iterator_release()*：释放 esp_partition_find 中返回的迭代器；
- *esp_partition_find_first()*：返回一个结构，描述 esp_partition_find 中找到的第一个分区；
- *esp_partition_read()*、*esp_partition_write()* 和*esp_partition_erase_range()* 在分区边界内执行，等同于 spi_flash_read()、spi_flash_write() 和 spi_flash_erase_range()。

注解： 请在应用程序代码中使用上述 esp_partition_* API 函数，而非低层级的 spi_flash_* API 函数。分区表 API 函数根据存储在分区表中的数据，进行边界检查并计算在 flash 中的正确偏移量。

SPI Flash 加密

您可以对 SPI flash 内容进行加密，并在硬件层对其进行透明解密。

请参阅*Flash 加密*，查看详细信息。

内存映射 API

ESP32 内存硬件可以将 flash 部分区域映射到指令地址空间和数据地址空间，此映射仅用于读操作。不能通过写入 flash 映射的存储区域来改变 flash 中内容。

Flash 以 64 KB 页为单位进行地址映射。内存映射硬件最多可将 4 MB flash 映射到数据地址空间，将 16 MB flash 映射到指令地址空间。请参考《ESP32 技术参考手册》查看内存映射硬件的详细信息。

请注意，有些 64 KB 页还用于将应用程序映射到内存中，因此实际可用的 64 KB 页会更少一些。

Flash 加密 启用时，使用内存映射区域从 flash 读取数据是解密 flash 的唯一方法，解密需在硬件层进行。

内存映射 API 在 `esp_spi_flash.h` 和 `esp_partition.h` 中声明：

- `spi_flash_mmap()`：将 flash 物理地址区域映射到 CPU 指令空间或数据空间；
- `spi_flash_munmap()`：取消上述区域的映射；
- *`esp_partition_mmap()`*：将分区的一部分映射至 CPU 指令空间或数据空间；

  `spi_flash_mmap()` 和*`esp_partition_mmap()`* 的区别如下：

- `spi_flash_mmap()`：需要给定一个 64 KB 对齐的物理地址；
- *`esp_partition_mmap()`*：给定分区内任意偏移量即可，此函数根据需要将返回的指针调整至指向映射内存。

内存映射在 64 KB 块中进行，如果分区已传递给 `esp_partition_mmap`，则可读取分区外数据。

实现

`esp_flash_t` 结构包含芯片数据和该 API 的三个重要部分：

1. 主机驱动，为访问芯片提供硬件支持；
2. 芯片驱动，为不同芯片提供兼容性服务；
3. OS 函数，在不同阶段（一级或二级 Boot 或者应用程序阶段）为部分 OS 函数提供支持（如一些锁、延迟）。

主机驱动

主机驱动依赖 `soc/include/hal` 文件夹下 `spi_flash_host_drv.h` 定义的 `spi_flash_host_driver_t` 接口。该接口提供了一些与芯片通信常用的函数。

在 SPI HAL 文件中，有些函数是基于现有的 ESP32 memory-spi 来实现的。但是，由于 ESP32 速度限制，HAL 层无法提供某些读命令的高速实现（所以这些命令根本没有在 HAL 的文件中被实现）。`memspi_host_driver.h` 和 `.c` 文件使用 HAL 提供的 `common_command` 函数实现上述读命令的高速版本，并将所有它实现的及 HAL 函数封装为 `spi_flash_host_driver_t` 供更上层调用。

您也可以实现自己的主机驱动，甚至只通过简单的 GPIO。只要实现了 `spi_flash_host_driver_t` 中所有函数，不管底层硬件是什么，esp_flash API 都可以访问 flash。

芯片驱动

芯片驱动在 `spi_flash_chip_driver.h` 中进行定义，并将主机驱动提供的基本函数进行封装以供 API 层使用。

有些操作需在执行前先发送命令，或在执行后读取状态，因此有些芯片需要不同的命令或值以及通信方式。

`generic chip` 芯片代表了常见的 flash 芯片，其他芯片驱动可以在通用芯片的基础上进行开发。

芯片驱动依赖主机驱动。

OS 函数

OS 函数层提供访问锁和延迟的方法。

该锁定用于解决 SPI Flash 芯片访问和其他函数之间的冲突。例如，经 SPI0/1 访问 flash 芯片时，应当禁用 cache（平时用于取代码和 PSRAM 数据）。另一种情况是，一些没有 CS 线或者 CS 线受软件控制的设备（如通过 SPI 接口的 SD 卡控制）需要在一段时间内独占总线。

延时则用于某些长时操作，需要主机处于等待状态或执行轮询。

顶层 API 将芯片驱动和 OS 函数封装成一个完整的组件，并提供参数检查。

另请参考

- 分区表
- *OTA API* 提供了高层 API 用于更新存储在 flash 中的 app 固件。
- *NVS API* 提供了结构化 API 用于存储 SPI flash 中的碎片数据。

实现细节

必须确保操作期间，两个 CPU 均未从 flash 运行代码，实现细节如下：

- 单核模式下，SDK 在执行 flash 操作前将禁用中断或调度算法。
- 双核模式下，实现细节更为复杂，SDK 需确保两个 CPU 均未运行 flash 代码。

如果有 SPI flash API 在 CPU A（PRO 或 APP）上调用，它使用 `esp_ipc_call` API 在 CPU B 上运行 `spi_flash_op_block_func` 函数。`esp_ipc_call` API 在 CPU B 上唤醒一个高优先级任务，即运行 `spi_flash_op_block_func` 函数。运行该函数将禁用 CPU B 上的 cache，并使用 `s_flash_op_can_start` 旗帜来标志 cache 已禁用。然后，CPU A 上的任务也会禁用 cache 并继续执行 flash 操作。

执行 flash 操作时，CPU A 和 CPU B 仍然可以执行中断操作。默认中断代码均存储于 RAM 中，如果新添加了中断分配 API，则应添加一个标志位以请求在 flash 操作期间禁用该新分配的中断。

Flash 操作完成后，CPU A 上的函数将设置另一标志位，即 `s_flash_op_complete`，用以通知 CPU B 上的任务可以重新启用 cache 并释放 CPU。接着，CPU A 上的函数也重新启用 cache，并将控制权返还给调用者。

另外，所有 API 函数均受互斥量 `s_flash_op_mutex` 保护。

在单核环境中（启用*CONFIG_FREERTOS_UNICORE*），您需要禁用上述两个 cache 以防发生 CPU 间通信。

SPI Flash API 参考

Header File

- spi_flash/include/esp_flash_spi_init.h

Functions

esp_err_t **spi_bus_add_flash_device**(*esp_flash_t* ***out_chip*, **const** *esp_flash_spi_device_config_t* **config*)

Add a SPI Flash device onto the SPI bus.

The bus should be already initialized by `spi_bus_initialization`.

Return

- ESP_ERR_INVALID_ARG: out_chip is NULL, or some field in the config is invalid.
- ESP_ERR_NO_MEM: failed to allocate memory for the chip structures.
- ESP_OK: success.

Parameters

- `out_chip`: Pointer to hold the initialized chip.
- `config`: Configuration of the chips to initialize.

esp_err_t **spi_bus_remove_flash_device**(*esp_flash_t* **chip*)

Remove a SPI Flash device from the SPI bus.

Return

- ESP_ERR_INVALID_ARG: The chip is invalid.
- ESP_OK: success.

Parameters

- `chip`: The flash device to remove.

Structures

`struct esp_flash_spi_device_config_t`

Configurations for the SPI Flash to init.

Public Members

spi_host_device_t **host_id**

Bus to use.

int **cs_id**

CS pin (signal) to use.

int **cs_io_num**

GPIO pin to output the CS signal.

esp_flash_io_mode_t **io_mode**

IO mode to read from the Flash.

esp_flash_speed_t **speed**

Speed of the Flash clock.

int **input_delay_ns**

Input delay of the data pins, in ns. Set to 0 if unknown.

Header File

- spi_flash/include/esp_flash.h

Functions

esp_err_t **esp_flash_init**(*esp_flash_t* **chip*)

Initialise SPI flash chip interface.

This function must be called before any other API functions are called for this chip.

Note Only the `host` and `read_mode` fields of the chip structure must be initialised before this function is called. Other fields may be auto-detected if left set to zero or NULL.

Note If the chip->drv pointer is NULL, chip chip_drv will be auto-detected based on its manufacturer & product IDs. See `esp_flash_registered_flash_drivers` pointer for details of this process.

Return ESP_OK on success, or a flash error code if initialisation fails.

Parameters

- `chip`: Pointer to SPI flash chip to use. If NULL, esp_flash_default_chip is substituted.

bool **esp_flash_chip_driver_initialized**(const *esp_flash_t* *chip)

Check if appropriate chip driver is set.

Return true if set, otherwise false.

Parameters

- `chip`: Pointer to SPI flash chip to use. If NULL, esp_flash_default_chip is substituted.

esp_err_t **esp_flash_read_id**(*esp_flash_t* *chip, uint32_t *out_id)

Read flash ID via the common "RDID" SPI flash command.

ID is a 24-bit value. Lower 16 bits of 'id' are the chip ID, upper 8 bits are the manufacturer ID.

Parameters

- `chip`: Pointer to identify flash chip. Must have been successfully initialised via esp_flash_init()
- `out_id`: Pointer to receive ID value.

Return ESP_OK on success, or a flash error code if operation failed.

esp_err_t **esp_flash_get_size**(*esp_flash_t* *chip, uint32_t *out_size)

Detect flash size based on flash ID.

Note Most flash chips use a common format for flash ID, where the lower 4 bits specify the size as a power of 2. If the manufacturer doesn't follow this convention, the size may be incorrectly detected.

Return ESP_OK on success, or a flash error code if operation failed.

Parameters

- `chip`: Pointer to identify flash chip. Must have been successfully initialised via esp_flash_init()
- `out_size`: Detected size in bytes.

esp_err_t **esp_flash_erase_chip**(*esp_flash_t* *chip)

Erase flash chip contents.

Return ESP_OK on success, or a flash error code if operation failed.

Parameters

- `chip`: Pointer to identify flash chip. Must have been successfully initialised via esp_flash_init()

esp_err_t **esp_flash_erase_region**(*esp_flash_t* **chip*, uint32_t *start*, uint32_t *len*)

Erase a region of the flash chip.

Sector size is specifyed in chip->drv->sector_size field (typically 4096 bytes.) ESP_ERR_INVALID_ARG will be returned if the start & length are not a multiple of this size.

Parameters

- `chip`: Pointer to identify flash chip. Must have been successfully initialised via esp_flash_init()
- `start`: Address to start erasing flash. Must be sector aligned.
- `len`: Length of region to erase. Must also be sector aligned.

Erase is performed using block (multi-sector) erases where possible (block size is specified in chip->drv->block_erase_size field, typically 65536 bytes). Remaining sectors are erased using individual sector erase commands.

Return ESP_OK on success, or a flash error code if operation failed.

esp_err_t **esp_flash_get_chip_write_protect**(*esp_flash_t* **chip*, bool **write_protected*)

Read if the entire chip is write protected.

Note A correct result for this flag depends on the SPI flash chip model and chip_drv in use (via the 'chip->drv' field).

Return ESP_OK on success, or a flash error code if operation failed.

Parameters

- `chip`: Pointer to identify flash chip. Must have been successfully initialised via esp_flash_init()
- `write_protected`: Pointer to boolean, set to the value of the write protect flag.

esp_err_t **esp_flash_set_chip_write_protect**(*esp_flash_t* **chip*, bool *write_protect*)

Set write protection for the SPI flash chip.

Some SPI flash chips may require a power cycle before write protect status can be cleared. Otherwise, write protection can be removed via a follow-up call to this function.

Note Correct behaviour of this function depends on the SPI flash chip model and chip_drv in use (via the 'chip->drv' field).

Parameters

- `chip`: Pointer to identify flash chip. Must have been successfully initialised via esp_flash_init()
- `write_protect`: Boolean value for the write protect flag

Return ESP_OK on success, or a flash error code if operation failed.

esp_err_t **esp_flash_get_protectable_regions**(**const** *esp_flash_t* **chip*, **const** *esp_flash_region_t* ***out_regions*, uint32_t **out_num_regions*)

Read the list of individually protectable regions of this SPI flash chip.

Note Correct behaviour of this function depends on the SPI flash chip model and chip_drv in use (via the 'chip->drv' field).

Return ESP_OK on success, or a flash error code if operation failed.

Parameters

- `chip`: Pointer to identify flash chip. Must have been successfully initialised via esp_flash_init()
- `out_regions`: Pointer to receive a pointer to the array of protectable regions of the chip.
- `out_num_regions`: Pointer to an integer receiving the count of protectable regions in the array returned in 'regions'.

esp_err_t **esp_flash_get_protected_region**(*esp_flash_t* **chip*, **const** *esp_flash_region_t* **region*, bool **out_protected*)

Detect if a region of the SPI flash chip is protected.

Note It is possible for this result to be false and write operations to still fail, if protection is enabled for the entire chip.

Note Correct behaviour of this function depends on the SPI flash chip model and chip_drv in use (via the 'chip->drv' field).

Return ESP_OK on success, or a flash error code if operation failed.

Parameters

- `chip`: Pointer to identify flash chip. Must have been successfully initialised via esp_flash_init()
- `region`: Pointer to a struct describing a protected region. This must match one of the regions returned from esp_flash_get_protectable_regions(…).
- `out_protected`: Pointer to a flag which is set based on the protected status for this region.

esp_err_t **esp_flash_set_protected_region**(*esp_flash_t* *chip, const *esp_flash_region_t* *region, bool *protect*)

Update the protected status for a region of the SPI flash chip.

Note It is possible for the region protection flag to be cleared and write operations to still fail, if protection is enabled for the entire chip.

Note Correct behaviour of this function depends on the SPI flash chip model and chip_drv in use (via the 'chip->drv' field).

Return ESP_OK on success, or a flash error code if operation failed.

Parameters

- `chip`: Pointer to identify flash chip. Must have been successfully initialised via esp_flash_init()
- `region`: Pointer to a struct describing a protected region. This must match one of the regions returned from esp_flash_get_protectable_regions(···).
- `protect`: Write protection flag to set.

esp_err_t **esp_flash_read**(*esp_flash_t* *chip, void *buffer, uint32_t *address*, uint32_t *length*)

Read data from the SPI flash chip.

There are no alignment constraints on buffer, address or length.

Parameters

- `chip`: Pointer to identify flash chip. Must have been successfully initialised via esp_flash_init()
- `buffer`: Pointer to a buffer where the data will be read. To get better performance, this should be in the DRAM and word aligned.
- `address`: Address on flash to read from. Must be less than chip->size field.
- `length`: Length (in bytes) of data to read.

Note If on-chip flash encryption is used, this function returns raw (ie encrypted) data. Use the flash cache to transparently decrypt data.

Return

- ESP_OK: success
- ESP_ERR_NO_MEM: Buffer is in external PSRAM which cannot be concurrently accessed, and a temporary internal buffer could not be allocated.
- or a flash error code if operation failed.

esp_err_t **esp_flash_write**(*esp_flash_t* *chip, **const** void *buffer, uint32_t address, uint32_t length)

Write data to the SPI flash chip.

There are no alignment constraints on buffer, address or length.

Parameters

- `chip`: Pointer to identify flash chip. Must have been successfully initialised via esp_flash_init()
- `address`: Address on flash to write to. Must be previously erased (SPI NOR flash can only write bits 1->0).
- `buffer`: Pointer to a buffer with the data to write. To get better performance, this should be in the DRAM and word aligned.
- `length`: Length (in bytes) of data to write.

Return ESP_OK on success, or a flash error code if operation failed.

esp_err_t **esp_flash_write_encrypted**(*esp_flash_t* *chip, uint32_t address, **const** void *buffer, uint32_t length)

Encrypted and write data to the SPI flash chip using on-chip hardware flash encryption.

Note Both address & length must be 16 byte aligned, as this is the encryption block size

Return

- ESP_OK: on success
- ESP_ERR_NOT_SUPPORTED: encrypted write not supported for this chip.
- ESP_ERR_INVALID_ARG: Either the address, buffer or length is invalid.
- or other flash error code from spi_flash_write_encrypted().

Parameters

- `chip`: Pointer to identify flash chip. Must be NULL (the main flash chip). For other chips, encrypted write is not supported.
- `address`: Address on flash to write to. 16 byte aligned. Must be previously erased (SPI NOR flash can only write bits 1->0).
- `buffer`: Pointer to a buffer with the data to write.
- `length`: Length (in bytes) of data to write. 16 byte aligned.

esp_err_t **esp_flash_read_encrypted**(*esp_flash_t* *chip, uint32_t address, void *out_buffer, uint32_t length)

Read and decrypt data from the SPI flash chip using on-chip hardware flash encryption.

Return

- ESP_OK: on success
- ESP_ERR_NOT_SUPPORTED: encrypted read not supported for this chip.
- or other flash error code from spi_flash_read_encrypted().

Parameters

- `chip`: Pointer to identify flash chip. Must be NULL (the main flash chip). For other chips, encrypted read is not supported.
- `address`: Address on flash to read from.
- `out_buffer`: Pointer to a buffer for the data to read to.
- `length`: Length (in bytes) of data to read.

`static` bool **esp_flash_is_quad_mode**(`const` *esp_flash_t* **chip*)

Returns true if chip is configured for Quad I/O or Quad Fast Read.

Return true if flash works in quad mode, otherwise false

Parameters

- `chip`: Pointer to SPI flash chip to use. If NULL, esp_flash_default_chip is substituted.

Structures

`struct esp_flash_region_t`

Structure for describing a region of flash.

Public Members

uint32_t **offset**

Start address of this region.

uint32_t **size**

Size of the region.

`struct esp_flash_os_functions_t`

OS-level integration hooks for accessing flash chips inside a running OS

Public Members

esp_err_t (***start**)(void *arg)

Called before commencing any flash operation. Does not need to be recursive (ie is called at most once for each call to 'end').

esp_err_t (***end**)(void *arg)

Called after completing any flash operation.

esp_err_t (***region_protected**)(void *arg, size_t start_addr, size_t size)

Called before any erase/write operations to check whether the region is limited by the OS

esp_err_t (***delay_ms**)(void *arg, unsigned ms)

Delay for at least 'ms' milliseconds. Called in between 'start' and 'end'.

struct esp_flash_t

Structure to describe a SPI flash chip connected to the system.

Structure must be initialized before use (passed to esp_flash_init()).

Public Members

spi_flash_host_driver_t ***host**

Pointer to hardware-specific "host_driver" structure. Must be initialized before used.

const *spi_flash_chip_t* ***chip_drv**

Pointer to chip-model-specific "adapter"structure. If NULL, will be detected during initialisation.

const *esp_flash_os_functions_t* ***os_func**

Pointer to os-specific hook structure. Call `esp_flash_init_os_functions()` to setup this field, after the host is properly initialized.

void ***os_func_data**

Pointer to argument for os-specific hooks. Left NULL and will be initialized with `os_func`.

esp_flash_io_mode_t **read_mode**

Configured SPI flash read mode. Set before `esp_flash_init` is called.

uint32_t **size**

Size of SPI flash in bytes. If 0, size will be detected during initialisation.

uint32_t **chip_id**

Detected chip id.

Type Definitions

typedef struct *spi_flash_chip_t* **spi_flash_chip_t**

typedef struct *esp_flash_t* **esp_flash_t**

Header File

- soc/include/hal/spi_flash_types.h

Structures

struct spi_flash_trans_t

Definition of a common transaction. Also holds the return value.

Public Members

uint8_t **command**

Command to send, always 8bits.

uint8_t **mosi_len**

Output data length, in bytes.

uint8_t **miso_len**

Input data length, in bytes.

uint8_t **address_bitlen**

Length of address in bits, set to 0 if command does not need an address.

uint32_t **address**

Address to perform operation on.

const uint8_t ***mosi_data**

Output data to salve.

uint8_t ***miso_data**

[out] Input data from slave, little endian

struct spi_flash_host_driver_t

Host driver configuration and context structure.

Public Members

void ***driver_data**

Configuration and static data used by the specific host driver. The type is determined by the host driver.

esp_err_t (***dev_config**)(*spi_flash_host_driver_t* *driver)

Configure the device-related register before transactions. This saves some time to re-configure those registers when we send continuously

esp_err_t (***common_command**)(*spi_flash_host_driver_t* *driver, *spi_flash_trans_t* *t)

Send an user-defined spi transaction to the device.

esp_err_t (***read_id**)(*spi_flash_host_driver_t* *driver, uint32_t *id)

Read flash ID.

void (*erase_chip)(*spi_flash_host_driver_t* *driver)
Erase whole flash chip.

void (*erase_sector)(*spi_flash_host_driver_t* *driver, uint32_t start_address)
Erase a specific sector by its start address.

void (*erase_block)(*spi_flash_host_driver_t* *driver, uint32_t start_address)
Erase a specific block by its start address.

esp_err_t (*read_status)(*spi_flash_host_driver_t* *driver, uint8_t *out_sr)
Read the status of the flash chip.

esp_err_t (*set_write_protect)(*spi_flash_host_driver_t* *driver, bool wp)
Disable write protection.

void (*program_page)(*spi_flash_host_driver_t* *driver, const void *buffer, uint32_t address, uint32_t length)
Program a page of the flash. Check max_write_bytes for the maximum allowed writing length.

bool (*supports_direct_write)(*spi_flash_host_driver_t* *driver, const void *p)
Check whether need to allocate new buffer to write

bool (*supports_direct_read)(*spi_flash_host_driver_t* *driver, const void *p)
Check whether need to allocate new buffer to read

int max_write_bytes
maximum length of program_page

esp_err_t (*read)(*spi_flash_host_driver_t* *driver, void *buffer, uint32_t address, uint32_t read_len)
Read data from the flash. Check max_read_bytes for the maximum allowed reading length.

int max_read_bytes
maximum length of read

bool (*host_idle)(*spi_flash_host_driver_t* *driver)
Check whether the host is idle to perform new operations.

esp_err_t (*configure_host_io_mode)(*spi_flash_host_driver_t* *driver, uint32_t command, uint32_t addr_bitlen, int dummy_bitlen_base, *esp_flash_io_mode_t* io_mode)
Configure the host to work at different read mode. Responsible to compensate the timing and set IO mode.

void (*poll_cmd_done)(*spi_flash_host_driver_t* *driver)
Internal use, poll the HW until the last operation is done.

esp_err_t (*flush_cache)(*spi_flash_host_driver_t* *driver, uint32_t addr, uint32_t size)
For some host (SPI1), they are shared with a cache. When the data is modified, the cache needs to be flushed. Left NULL if not supported.

Macros

`ESP_FLASH_SPEED_MIN`

Lowest speed supported by the driver, currently 5 MHz.

`SPI_FLASH_READ_MODE_MIN`

Slowest io mode supported by ESP32, currently SlowRd.

Type Definitions

typedef struct *spi_flash_host_driver_t* **spi_flash_host_driver_t**

Enumerations

enum esp_flash_speed_t

SPI flash clock speed values, always refer to them by the enum rather than the actual value (more speed may be appended into the list).

A strategy to select the maximum allowed speed is to enumerate from the `ESP_FLSH_SPEED_MAX-1` or highest frequency supported by your flash, and decrease the speed until the probing success.

Values:

`ESP_FLASH_5MHZ` = 0

The flash runs under 5MHz.

`ESP_FLASH_10MHZ`

The flash runs under 10MHz.

`ESP_FLASH_20MHZ`

The flash runs under 20MHz.

`ESP_FLASH_26MHZ`

The flash runs under 26MHz.

`ESP_FLASH_40MHZ`

The flash runs under 40MHz.

`ESP_FLASH_80MHZ`

The flash runs under 80MHz.

`ESP_FLASH_SPEED_MAX`

The maximum frequency supported by the host is `ESP_FLASH_SPEED_MAX-1`.

enum esp_flash_io_mode_t

Mode used for reading from SPI flash.

Values:

`SPI_FLASH_SLOWRD` = 0

Data read using single I/O, some limits on speed.

`SPI_FLASH_FASTRD`

Data read using single I/O, no limit on speed.

`SPI_FLASH_DOUT`

Data read using dual I/O.

`SPI_FLASH_DIO`

Both address & data transferred using dual I/O.

`SPI_FLASH_QOUT`

Data read using quad I/O.

`SPI_FLASH_QIO`

Both address & data transferred using quad I/O.

`SPI_FLASH_READ_MODE_MAX`

The fastest io mode supported by the host is `ESP_FLASH_READ_MODE_MAX-1`.

分区表 API 参考

Header File

- spi_flash/include/esp_partition.h

Functions

esp_partition_iterator_t **esp_partition_find**(*esp_partition_type_t type*, *esp_partition_subtype_t subtype*, **const** char ***label*)

Find partition based on one or more parameters.

Return iterator which can be used to enumerate all the partitions found, or NULL if no partitions were found. Iterator obtained through this function has to be released using esp_partition_iterator_release when not used any more.

Parameters

- `type`: Partition type, one of esp_partition_type_t values
- `subtype`: Partition subtype, one of esp_partition_subtype_t values. To find all partitions of given type, use ESP_PARTITION_SUBTYPE_ANY.
- `label`: (optional) Partition label. Set this value if looking for partition with a specific name. Pass NULL otherwise.

const *esp_partition_t* ***esp_partition_find_first**(*esp_partition_type_t* *type*, *esp_partition_subtype_t* *subtype*, const char **label*)

Find first partition based on one or more parameters.

Return pointer to *esp_partition_t* structure, or NULL if no partition is found. This pointer is valid for the lifetime of the application.

Parameters

- `type`: Partition type, one of esp_partition_type_t values
- `subtype`: Partition subtype, one of esp_partition_subtype_t values. To find all partitions of given type, use ESP_PARTITION_SUBTYPE_ANY.
- `label`: (optional) Partition label. Set this value if looking for partition with a specific name. Pass NULL otherwise.

const *esp_partition_t* ***esp_partition_get**(*esp_partition_iterator_t* *iterator*)

Get *esp_partition_t* structure for given partition.

Return pointer to *esp_partition_t* structure. This pointer is valid for the lifetime of the application.

Parameters

- `iterator`: Iterator obtained using esp_partition_find. Must be non-NULL.

esp_partition_iterator_t **esp_partition_next**(*esp_partition_iterator_t* *iterator*)

Move partition iterator to the next partition found.

Any copies of the iterator will be invalid after this call.

Return NULL if no partition was found, valid esp_partition_iterator_t otherwise.

Parameters

- `iterator`: Iterator obtained using esp_partition_find. Must be non-NULL.

void **esp_partition_iterator_release**(*esp_partition_iterator_t* *iterator*)

Release partition iterator.

Parameters

- `iterator`: Iterator obtained using esp_partition_find. Must be non-NULL.

const *esp_partition_t* ***esp_partition_verify**(const *esp_partition_t* **partition*)

Verify partition data.

Given a pointer to partition data, verify this partition exists in the partition table (all fields match.)

This function is also useful to take partition data which may be in a RAM buffer and convert it to a pointer to the permanent partition data stored in flash.

Pointers returned from this function can be compared directly to the address of any pointer returned from *esp_partition_get()*, as a test for equality.

Return

- If partition not found, returns NULL.
- If found, returns a pointer to the *esp_partition_t* structure in flash. This pointer is always valid for the lifetime of the application.

Parameters

- `partition`: Pointer to partition data to verify. Must be non-NULL. All fields of this structure must match the partition table entry in flash for this function to return a successful match.

esp_err_t **esp_partition_read**(**const** *esp_partition_t* **partition*, size_t *src_offset*, void **dst*, size_t *size*)

Read data from the partition.

Return ESP_OK, if data was read successfully; ESP_ERR_INVALID_ARG, if src_offset exceeds partition size; ESP_ERR_INVALID_SIZE, if read would go out of bounds of the partition; or one of error codes from lower-level flash driver.

Parameters

- `partition`: Pointer to partition structure obtained using esp_partition_find_first or esp_partition_get. Must be non-NULL.
- `dst`: Pointer to the buffer where data should be stored. Pointer must be non-NULL and buffer must be at least 'size' bytes long.
- `src_offset`: Address of the data to be read, relative to the beginning of the partition.
- `size`: Size of data to be read, in bytes.

esp_err_t **esp_partition_write**(**const** *esp_partition_t* **partition*, size_t *dst_offset*, **const** void **src*, size_t *size*)

Write data to the partition.

Before writing data to flash, corresponding region of flash needs to be erased. This can be done using esp_partition_erase_range function.

Partitions marked with an encryption flag will automatically be written via the spi_flash_write_encrypted() function. If writing to an encrypted partition, all write offsets and lengths must be multiples of 16 bytes. See the spi_flash_write_encrypted() function for more details. Unencrypted partitions do not have this restriction.

Note Prior to writing to flash memory, make sure it has been erased with esp_partition_erase_range call.

Return ESP_OK, if data was written successfully; ESP_ERR_INVALID_ARG, if dst_offset exceeds partition size; ESP_ERR_INVALID_SIZE, if write would go out of bounds of the partition; or one of error codes from lower-level flash driver.

Parameters

- `partition`: Pointer to partition structure obtained using esp_partition_find_first or esp_partition_get. Must be non-NULL.
- `dst_offset`: Address where the data should be written, relative to the beginning of the partition.
- `src`: Pointer to the source buffer. Pointer must be non-NULL and buffer must be at least 'size' bytes long.
- `size`: Size of data to be written, in bytes.

esp_err_t **esp_partition_erase_range(const** *esp_partition_t* ******partition*, size_t *offset*, size_t *size*)

Erase part of the partition.

Return ESP_OK, if the range was erased successfully; ESP_ERR_INVALID_ARG, if iterator or dst are NULL; ESP_ERR_INVALID_SIZE, if erase would go out of bounds of the partition; or one of error codes from lower-level flash driver.

Parameters

- `partition`: Pointer to partition structure obtained using esp_partition_find_first or esp_partition_get. Must be non-NULL.
- `offset`: Offset from the beginning of partition where erase operation should start. Must be aligned to 4 kilobytes.
- `size`: Size of the range which should be erased, in bytes. Must be divisible by 4 kilobytes.

esp_err_t **esp_partition_mmap(const** *esp_partition_t* ******partition*, size_t *offset*, size_t *size*, spi_flash_mmap_memory_t *memory*, **const** void *******out_ptr*, spi_flash_mmap_handle_t ******out_handle*)

Configure MMU to map partition into data memory.

Unlike spi_flash_mmap function, which requires a 64kB aligned base address, this function doesn' t impose such a requirement. If offset results in a flash address which is not aligned to 64kB boundary, address will be rounded to the lower 64kB boundary, so that mapped region includes requested range. Pointer returned via out_ptr argument will be adjusted to point to the requested offset (not necessarily to the beginning of mmap-ed region).

To release mapped memory, pass handle returned via out_handle argument to spi_flash_munmap function.

Return ESP_OK, if successful

Parameters

- `partition`: Pointer to partition structure obtained using esp_partition_find_first or esp_partition_get. Must be non-NULL.
- `offset`: Offset from the beginning of partition where mapping should start.
- `size`: Size of the area to be mapped.
- `memory`: Memory space where the region should be mapped
- `out_ptr`: Output, pointer to the mapped memory region
- `out_handle`: Output, handle which should be used for spi_flash_munmap call

esp_err_t **esp_partition_get_sha256(const** *esp_partition_t* ***partition**, uint8_t ***sha_256**)

Get SHA-256 digest for required partition.

For apps with SHA-256 appended to the app image, the result is the appended SHA-256 value for the app image content. The hash is verified before returning, if app content is invalid then the function returns ESP_ERR_IMAGE_INVALID. For apps without SHA-256 appended to the image, the result is the SHA-256 of all bytes in the app image. For other partition types, the result is the SHA-256 of the entire partition.

Return

- ESP_OK: In case of successful operation.
- ESP_ERR_INVALID_ARG: The size was 0 or the sha_256 was NULL.
- ESP_ERR_NO_MEM: Cannot allocate memory for sha256 operation.
- ESP_ERR_IMAGE_INVALID: App partition doesn't contain a valid app image.
- ESP_FAIL: An allocation error occurred.

Parameters

- `partition`: Pointer to info for partition containing app or data. (fields: address, size and type, are required to be filled).
- `sha_256`: Returned SHA-256 digest for a given partition.

bool **esp_partition_check_identity(const** *esp_partition_t* ***partition_1**, **const** *esp_partition_t* ***partition_2**)

Check for the identity of two partitions by SHA-256 digest.

Return

- True: In case of the two firmware is equal.
- False: Otherwise

Parameters

- `partition_1`: Pointer to info for partition 1 containing app or data. (fields: address, size and type, are required to be filled).
- `partition_2`: Pointer to info for partition 2 containing app or data. (fields: address, size and type, are required to be filled).

esp_err_t **esp_partition_register_external**(*esp_flash_t* **flash_chip*, size_t *offset*, size_t *size*, **const** char **label*, *esp_partition_type_t* *type*, *esp_partition_subtype_t* *subtype*, **const** *esp_partition_t* ***out_partition*)

Register a partition on an external flash chip.

This API allows designating certain areas of external flash chips (identified by the *esp_flash_t* structure) as partitions. This allows using them with components which access SPI flash through the esp_partition API.

Return

- ESP_OK on success
- ESP_ERR_NOT_SUPPORTED if CONFIG_CONFIG_SPI_FLASH_USE_LEGACY_IMPL is enabled
- ESP_ERR_NO_MEM if memory allocation has failed
- ESP_ERR_INVALID_ARG if the new partition overlaps another partition on the same flash chip
- ESP_ERR_INVALID_SIZE if the partition doesn't fit into the flash chip size

Parameters

- `flash_chip`: Pointer to the structure identifying the flash chip
- `offset`: Address in bytes, where the partition starts
- `size`: Size of the partition in bytes
- `label`: Partition name
- `type`: One of the partition types (ESP_PARTITION_TYPE_*). Note that applications can not be booted from external flash chips, so using ESP_PARTITION_TYPE_APP is not supported.
- `subtype`: One of the partition subtypes (ESP_PARTITION_SUBTYPE_*)
- `out_partition`: Output, if non-NULL, receives the pointer to the resulting *esp_partition_t* structure

esp_err_t **esp_partition_deregister_external**(**const** *esp_partition_t* **partition*)

Deregister the partition previously registered using esp_partition_register_external.

Return

- ESP_OK on success
- ESP_ERR_NOT_FOUND if the partition pointer is not found
- ESP_ERR_INVALID_ARG if the partition comes from the partition table
- ESP_ERR_INVALID_ARG if the partition was not registered using esp_partition_register_external function.

Parameters

- **partition**: pointer to the partition structure obtained from esp_partition_register_external,

Structures

struct esp_partition_t

partition information structure

This is not the format in flash, that format is esp_partition_info_t.

However, this is the format used by this API.

Public Members

esp_flash_t ***flash_chip**

SPI flash chip on which the partition resides

esp_partition_type_t **type**

partition type (app/data)

esp_partition_subtype_t **subtype**

partition subtype

uint32_t **address**

starting address of the partition in flash

uint32_t **size**

size of the partition, in bytes

char **label**[17]

partition label, zero-terminated ASCII string

bool **encrypted**

flag is set to true if partition is encrypted

Macros

`ESP_PARTITION_SUBTYPE_OTA(i)`
Convenience macro to get esp_partition_subtype_t value for the i-th OTA partition.

Type Definitions

typedef struct esp_partition_iterator_opaque_ ***esp_partition_iterator_t**
Opaque partition iterator type.

Enumerations

enum esp_partition_type_t
Partition type.

Note Keep this enum in sync with PartitionDefinition class gen_esp32part.py

Values:

`ESP_PARTITION_TYPE_APP` = 0x00
Application partition type.

`ESP_PARTITION_TYPE_DATA` = 0x01
Data partition type.

enum esp_partition_subtype_t
Partition subtype.

Note Keep this enum in sync with PartitionDefinition class gen_esp32part.py

Values:

`ESP_PARTITION_SUBTYPE_APP_FACTORY` = 0x00
Factory application partition.

`ESP_PARTITION_SUBTYPE_APP_OTA_MIN` = 0x10
Base for OTA partition subtypes.

`ESP_PARTITION_SUBTYPE_APP_OTA_0` = ESP_PARTITION_SUBTYPE_APP_OTA_MIN + 0
OTA partition 0.

`ESP_PARTITION_SUBTYPE_APP_OTA_1` = ESP_PARTITION_SUBTYPE_APP_OTA_MIN + 1
OTA partition 1.

`ESP_PARTITION_SUBTYPE_APP_OTA_2` = ESP_PARTITION_SUBTYPE_APP_OTA_MIN + 2
OTA partition 2.

ESP_PARTITION_SUBTYPE_APP_OTA_3 = ESP_PARTITION_SUBTYPE_APP_OTA_MIN + 3
OTA partition 3.

ESP_PARTITION_SUBTYPE_APP_OTA_4 = ESP_PARTITION_SUBTYPE_APP_OTA_MIN + 4
OTA partition 4.

ESP_PARTITION_SUBTYPE_APP_OTA_5 = ESP_PARTITION_SUBTYPE_APP_OTA_MIN + 5
OTA partition 5.

ESP_PARTITION_SUBTYPE_APP_OTA_6 = ESP_PARTITION_SUBTYPE_APP_OTA_MIN + 6
OTA partition 6.

ESP_PARTITION_SUBTYPE_APP_OTA_7 = ESP_PARTITION_SUBTYPE_APP_OTA_MIN + 7
OTA partition 7.

ESP_PARTITION_SUBTYPE_APP_OTA_8 = ESP_PARTITION_SUBTYPE_APP_OTA_MIN + 8
OTA partition 8.

ESP_PARTITION_SUBTYPE_APP_OTA_9 = ESP_PARTITION_SUBTYPE_APP_OTA_MIN + 9
OTA partition 9.

ESP_PARTITION_SUBTYPE_APP_OTA_10 = ESP_PARTITION_SUBTYPE_APP_OTA_MIN + 10
OTA partition 10.

ESP_PARTITION_SUBTYPE_APP_OTA_11 = ESP_PARTITION_SUBTYPE_APP_OTA_MIN + 11
OTA partition 11.

ESP_PARTITION_SUBTYPE_APP_OTA_12 = ESP_PARTITION_SUBTYPE_APP_OTA_MIN + 12
OTA partition 12.

ESP_PARTITION_SUBTYPE_APP_OTA_13 = ESP_PARTITION_SUBTYPE_APP_OTA_MIN + 13
OTA partition 13.

ESP_PARTITION_SUBTYPE_APP_OTA_14 = ESP_PARTITION_SUBTYPE_APP_OTA_MIN + 14
OTA partition 14.

ESP_PARTITION_SUBTYPE_APP_OTA_15 = ESP_PARTITION_SUBTYPE_APP_OTA_MIN + 15
OTA partition 15.

ESP_PARTITION_SUBTYPE_APP_OTA_MAX = ESP_PARTITION_SUBTYPE_APP_OTA_MIN + 16
Max subtype of OTA partition.

ESP_PARTITION_SUBTYPE_APP_TEST = 0x20
Test application partition.

ESP_PARTITION_SUBTYPE_DATA_OTA = 0x00
OTA selection partition.

ESP_PARTITION_SUBTYPE_DATA_PHY = 0x01
PHY init data partition.

`ESP_PARTITION_SUBTYPE_DATA_NVS` = 0x02
NVS partition.

`ESP_PARTITION_SUBTYPE_DATA_COREDUMP` = 0x03
COREDUMP partition.

`ESP_PARTITION_SUBTYPE_DATA_NVS_KEYS` = 0x04
Partition for NVS keys.

`ESP_PARTITION_SUBTYPE_DATA_EFUSE_EM` = 0x05
Partition for emulate eFuse bits.

`ESP_PARTITION_SUBTYPE_DATA_ESPHTTPD` = 0x80
ESPHTTPD partition.

`ESP_PARTITION_SUBTYPE_DATA_FAT` = 0x81
FAT partition.

`ESP_PARTITION_SUBTYPE_DATA_SPIFFS` = 0x82
SPIFFS partition.

`ESP_PARTITION_SUBTYPE_ANY` = 0xff
Used to search for partitions with any subtype.

Flash 加密 API 参考

Header File

- bootloader_support/include/esp_flash_encrypt.h

Functions

`static` bool **esp_flash_encryption_enabled**(void)
Is flash encryption currently enabled in hardware?

Flash encryption is enabled if the FLASH_CRYPT_CNT efuse has an odd number of bits set.

Return true if flash encryption is enabled.

esp_err_t **esp_flash_encrypt_check_and_update**(void)

esp_err_t **esp_flash_encrypt_region**(uint32_t *src_addr*, size_t *data_length*)
Encrypt-in-place a block of flash sectors.

Note This function resets RTC_WDT between operations with sectors.

Return ESP_OK if all operations succeeded, ESP_ERR_FLASH_OP_FAIL if SPI flash fails, ESP_ERR_FLASH_OP_TIMEOUT if flash times out.

Parameters

- `src_addr`: Source offset in flash. Should be multiple of 4096 bytes.
- `data_length`: Length of data to encrypt in bytes. Will be rounded up to next multiple of 4096 bytes.

void **esp_flash_write_protect_crypt_cnt**(void)

Write protect FLASH_CRYPT_CNT.

Intended to be called as a part of boot process if flash encryption is enabled but secure boot is not used. This should protect against serial re-flashing of an unauthorised code in absence of secure boot.

esp_flash_enc_mode_t **esp_get_flash_encryption_mode**(void)

Return the flash encryption mode.

The API is called during boot process but can also be called by application to check the current flash encryption mode of ESP32

Return

void **esp_flash_encryption_init_checks**(void)

Check the flash encryption mode during startup.

Verifies the flash encryption config during startup:

Note This function is called automatically during app startup, it doesn't need to be called from the app.

- Correct any insecure flash encryption settings if hardware Secure Boot is enabled.
- Log warnings if the efuse config doesn't match the project config in any way

Enumerations

enum esp_flash_enc_mode_t

Values:

ESP_FLASH_ENC_MODE_DISABLED

ESP_FLASH_ENC_MODE_DEVELOPMENT

ESP_FLASH_ENC_MODE_RELEASE

2.6.2 SD/SDIO/MMC 驱动程序

[English]

概述

SD/SDIO/MMC 驱动是一种基于 SDMMC 和 SD SPI 主机驱动的协议级驱动程序，目前已支持 SD 存储器、SDIO 卡和 eMMC 芯片。

SDMMC 主机驱动和 SD SPI 主机驱动（driver/include/driver/sdmmc_host.h）为以下功能提供 API：

- 发送命令至从设备
- 接收和发送数据
- 处理总线错误

初始化函数及配置函数：

- 如需初始化和配置 SDMMC 主机，请参阅*SDMMC 主机 API*
- 如需初始化和配置 SD SPI 主机，请参阅*SD SPI 主机 API*

本文档中所述的 SDMMC 协议层仅处理 SD 协议相关事项，例如卡初始化和数据传输命令。

协议层通过*sdmmc_host_t* 结构体和主机协同工作，该结构体包含指向主机各类函数的指针。

应用示例

ESP-IDF storage/sd_card 目录下提供了 SDMMC 驱动与 FatFs 库组合使用的示例，演示了先初始化卡，然后使用 POSIX 和 C 库 API 向卡读写数据。请参考示例目录下 README.md 文件，查看更多详细信息。

协议层 API

协议层具备*sdmmc_host_t* 结构体，此结构体描述了 SD/MMC 主机驱动，列出了其功能，并提供指向驱动程序函数的指针。协议层将卡信息储存于*sdmmc_card_t* 结构体中。向 SD/MMC 主机发送命令时，协议层调用时需要一个*sdmmc_command_t* 结构体来描述命令、参数、预期返回值和需传输的数据（如有）。

用于 SD 存储卡的 API

1. 初始化主机，请调用主机驱动函数，例如*sdmmc_host_init()* 和*sdmmc_host_init_slot()*；
2. 初始化卡，请调用*sdmmc_card_init()*，并将参数 `host` （即主机驱动信息）和参数 `card` （指向*sdmmc_card_t* 结构体的指针）传递给此函数。函数运行结束后，将会向*sdmmc_card_t* 结构体填充该卡的信息；
3. 读取或写入卡的扇区，请分别调用*sdmmc_read_sectors()* 和*sdmmc_write_sectors()*，并将参数 `card`（指向卡信息结构的指针）传递给函数；
4. 如果不再使用该卡，请调用主机驱动函数，例如*sdmmc_host_deinit()*，以禁用主机外设，并释放驱动程序分配的资源。

用于 eMMC 芯片的 API

从协议层的角度而言，eMMC 存储芯片与 SD 存储卡相同。尽管 eMMC 是芯片，不具备卡的外形，但由于协议相似 (*sdmmc_card_t*, *sdmmc_card_init*)，用于 SD 卡的一些概念同样适用于 eMMC 芯片。注意，eMMC 芯片不可通过 SPI 使用，因此它与 SD API 主机驱动不兼容。

如需初始化 eMMC 内存并执行读/写操作，请参照上一章节 SD 卡操作步骤。

用于 SDIO 卡的 API

SDIO 卡初始化和检测过程与 SD 存储卡相同，唯一的区别是 SDIO 模式下数据传输命令不同。

在卡初始化和卡检测（通过运行*sdmmc_card_init()*）期间，驱动仅配置 SDIO 卡如下寄存器：

1. I/O 中止 (0x06) 寄存器：在该寄存器中设置 RES 位可重置卡的 I/O 部分；
2. 总线接口控制 (0x07) 寄存器：如果主机和插槽配置中启用 4 线模式，则驱动程序会尝试在该寄存器中设置总线宽度字段。如果字段设置成功，则从机支持 4 线模式，主机也切换至 4 线模式；
3. 高速（0x13）寄存器：如果主机配置中启用高速模式，则会在该寄存器中设置 SHS 位。

注意，驱动程序不会在 (1) I/O 使能寄存器和 Int 使能寄存器，及 (2) I/O 块大小中，设置任何位。应用程序可通过调用*sdmmc_io_write_byte()* 来设置相关位。

如需设置卡配置或传输数据，请根据您的具体情况选择下表中的函数：

操作	读函数	写函数
使用 IO_RW_DIRECT (CMD52) 读写单个字节。	*sdmmc_io_read_byte()*	*sdmmc_io_write_byte()*
使用 IO_RW_EXTENDED (CMD53) 的字节模式读写多个字节。	*sdmmc_io_read_bytes()*	*sdmmc_io_write_bytes()*
块模式下，使用 IO_RW_EXTENDED (CMD53) 读写数据块。	*sdmmc_io_read_blocks()*	*sdmmc_io_write_blocks()*

使用*sdmmc_io_enable_int()* 函数，应用程序可启用 SDIO 中断。

在单线模式下使用 SDIO 时，还需要连接 D1 线来启用 SDIO 中断。

如果您需要应用程序保持等待直至发生 SDIO 中断，请使用*sdmmc_io_wait_int()* 函数。

复合卡（存储 + SDIO）

该驱动程序不支持 SDIO 复合卡，复合卡会被视为 SDIO 卡。

线程安全

多数应用程序仅需在一个任务中使用协议层。因此，协议层在*sdmmc_card_t* 结构体或在访问 SDMMC 或 SD SPI 主机驱动程序时不使用任何类型的锁。这种锁通常在较高层级实现，例如文件系统驱动程序。

API 参考

Header File

- sdmmc/include/sdmmc_cmd.h

Functions

esp_err_t **sdmmc_card_init**(**const** *sdmmc_host_t* **host*, *sdmmc_card_t* **out_card*)

Probe and initialize SD/MMC card using given host

Note Only SD cards (SDSC and SDHC/SDXC) are supported now. Support for MMC/eMMC cards will be added later.

Return

- ESP_OK on success
- One of the error codes from SDMMC host controller

Parameters

- `host`: pointer to structure defining host controller
- `out_card`: pointer to structure which will receive information about the card when the function completes

void **sdmmc_card_print_info**(FILE **stream*, **const** *sdmmc_card_t* **card*)

Print information about the card to a stream.

Parameters

- `stream`: stream obtained using fopen or fdopen
- `card`: card information structure initialized using sdmmc_card_init

esp_err_t **sdmmc_write_sectors**(*sdmmc_card_t* **card*, **const** void **src*, size_t *start_sector*, size_t *sector_count*)

Write given number of sectors to SD/MMC card

Return

- ESP_OK on success

- One of the error codes from SDMMC host controller

Parameters

- `card`: pointer to card information structure previously initialized using sdmmc_card_init
- `src`: pointer to data buffer to read data from; data size must be equal to sector_count * card->csd.sector_size
- `start_sector`: sector where to start writing
- `sector_count`: number of sectors to write

esp_err_t **sdmmc_read_sectors**(*sdmmc_card_t* **card*, void **dst*, size_t *start_sector*, size_t *sector_count*)

Write given number of sectors to SD/MMC card

Return

- ESP_OK on success
- One of the error codes from SDMMC host controller

Parameters

- `card`: pointer to card information structure previously initialized using sdmmc_card_init
- `dst`: pointer to data buffer to write into; buffer size must be at least sector_count * card->csd.sector_size
- `start_sector`: sector where to start reading
- `sector_count`: number of sectors to read

esp_err_t **sdmmc_io_read_byte**(*sdmmc_card_t* **card*, uint32_t *function*, uint32_t *reg*, uint8_t **out_byte*)

Read one byte from an SDIO card using IO_RW_DIRECT (CMD52)

Return

- ESP_OK on success
- One of the error codes from SDMMC host controller

Parameters

- `card`: pointer to card information structure previously initialized using sdmmc_card_init
- `function`: IO function number
- `reg`: byte address within IO function
- `out_byte`: output, receives the value read from the card

esp_err_t **sdmmc_io_write_byte**(*sdmmc_card_t* **card*, uint32_t *function*, uint32_t *reg*, uint8_t *in_byte*, uint8_t **out_byte*)

Write one byte to an SDIO card using IO_RW_DIRECT (CMD52)

Return

- ESP_OK on success
- One of the error codes from SDMMC host controller

Parameters

- `card`: pointer to card information structure previously initialized using sdmmc_card_init
- `function`: IO function number
- `reg`: byte address within IO function
- `in_byte`: value to be written
- `out_byte`: if not NULL, receives new byte value read from the card (read-after-write).

esp_err_t **sdmmc_io_read_bytes**(*sdmmc_card_t* **card*, uint32_t *function*, uint32_t *addr*, void **dst*, size_t *size*)

Read multiple bytes from an SDIO card using IO_RW_EXTENDED (CMD53)

This function performs read operation using CMD53 in byte mode. For block mode, see sdmmc_io_read_blocks.

Return

- ESP_OK on success
- ESP_ERR_INVALID_SIZE if size exceeds 512 bytes
- One of the error codes from SDMMC host controller

Parameters

- `card`: pointer to card information structure previously initialized using sdmmc_card_init
- `function`: IO function number
- `addr`: byte address within IO function where reading starts
- `dst`: buffer which receives the data read from card
- `size`: number of bytes to read

esp_err_t **sdmmc_io_write_bytes**(*sdmmc_card_t* **card*, uint32_t *function*, uint32_t *addr*, **const** void **src*, size_t *size*)

Write multiple bytes to an SDIO card using IO_RW_EXTENDED (CMD53)

This function performs write operation using CMD53 in byte mode. For block mode, see sdmmc_io_write_blocks.

Return

- ESP_OK on success
- ESP_ERR_INVALID_SIZE if size exceeds 512 bytes
- One of the error codes from SDMMC host controller

Parameters

- `card`: pointer to card information structure previously initialized using sdmmc_card_init
- `function`: IO function number
- `addr`: byte address within IO function where writing starts
- `src`: data to be written
- `size`: number of bytes to write

esp_err_t **sdmmc_io_read_blocks**(*sdmmc_card_t* **card*, uint32_t *function*, uint32_t *addr*, void **dst*, size_t *size*)

Read blocks of data from an SDIO card using IO_RW_EXTENDED (CMD53)

This function performs read operation using CMD53 in block mode. For byte mode, see sdmmc_io_read_bytes.

Return

- ESP_OK on success
- ESP_ERR_INVALID_SIZE if size is not divisible by 512 bytes
- One of the error codes from SDMMC host controller

Parameters

- `card`: pointer to card information structure previously initialized using sdmmc_card_init
- `function`: IO function number
- `addr`: byte address within IO function where writing starts
- `dst`: buffer which receives the data read from card
- `size`: number of bytes to read, must be divisible by the card block size.

esp_err_t **sdmmc_io_write_blocks**(*sdmmc_card_t* **card*, uint32_t *function*, uint32_t *addr*, **const** void **src*, size_t *size*)

Write blocks of data to an SDIO card using IO_RW_EXTENDED (CMD53)

This function performs write operation using CMD53 in block mode. For byte mode, see sdmmc_io_write_bytes.

Return

- ESP_OK on success
- ESP_ERR_INVALID_SIZE if size is not divisible by 512 bytes
- One of the error codes from SDMMC host controller

Parameters

- `card`: pointer to card information structure previously initialized using sdmmc_card_init
- `function`: IO function number
- `addr`: byte address within IO function where writing starts
- `src`: data to be written
- `size`: number of bytes to read, must be divisible by the card block size.

esp_err_t **sdmmc_io_enable_int**(*sdmmc_card_t* **card*)

Enable SDIO interrupt in the SDMMC host

Return

- ESP_OK on success
- ESP_ERR_NOT_SUPPORTED if the host controller does not support IO interrupts

Parameters

- `card`: pointer to card information structure previously initialized using sdmmc_card_init

esp_err_t **sdmmc_io_wait_int**(*sdmmc_card_t* **card*, TickType_t *timeout_ticks*)

Block until an SDIO interrupt is received

Slave uses D1 line to signal interrupt condition to the host. This function can be used to wait for the interrupt.

Return

- ESP_OK if the interrupt is received
- ESP_ERR_NOT_SUPPORTED if the host controller does not support IO interrupts
- ESP_ERR_TIMEOUT if the interrupt does not happen in timeout_ticks

Parameters

- `card`: pointer to card information structure previously initialized using sdmmc_card_init
- `timeout_ticks`: time to wait for the interrupt, in RTOS ticks

esp_err_t **sdmmc_io_get_cis_data**(*sdmmc_card_t* **card*, uint8_t **out_buffer*, size_t *buffer_size*, size_t **inout_cis_size*)

Get the data of CIS region of a SDIO card.

You may provide a buffer not sufficient to store all the CIS data. In this case, this functions store as much data into your buffer as possible. Also, this function will try to get and return the size required for you.

Return

- ESP_OK: on success
- ESP_ERR_INVALID_RESPONSE: if the card does not (correctly) support CIS.
- ESP_ERR_INVALID_SIZE: CIS_CODE_END found, but buffer_size is less than required size, which is stored in the inout_cis_size then.
- ESP_ERR_NOT_FOUND: if the CIS_CODE_END not found. Increase input value of inout_cis_size or set it to 0, if you still want to search for the end; output value of inout_cis_size is invalid in this case.
- and other error code return from sdmmc_io_read_bytes

Parameters

- `card`: pointer to card information structure previously initialized using sdmmc_card_init
- `out_buffer`: Output buffer of the CIS data
- `buffer_size`: Size of the buffer.
- `inout_cis_size`: Mandatory, pointer to a size, input and output.
  - input: Limitation of maximum searching range, should be 0 or larger than buffer_size. The function searches for CIS_CODE_END until this range. Set to 0 to search infinitely.
  - output: The size required to store all the CIS data, if CIS_CODE_END is found.

esp_err_t **sdmmc_io_print_cis_info**(uint8_t **buffer*, size_t *buffer_size*, FILE **fp*)

Parse and print the CIS information of a SDIO card.

Note Not all the CIS codes and all kinds of tuples are supported. If you see some unresolved code, you can add the parsing of these code in sdmmc_io.c and contribute to the IDF through the Github repository.

```
using sdmmc_card_init
```

Return

- ESP_OK: on success
- ESP_ERR_NOT_SUPPORTED: if the value from the card is not supported to be parsed.
- ESP_ERR_INVALID_SIZE: if the CIS size fields are not correct.

Parameters

- `buffer`: Buffer to parse
- `buffer_size`: Size of the buffer.
- `fp`: File pointer to print to, set to NULL to print to stdout.

Header File

- driver/include/driver/sdmmc_types.h

Structures

struct sdmmc_csd_t

Decoded values from SD card Card Specific Data register

Public Members

int **csd_ver**

CSD structure format

int **mmc_ver**

MMC version (for CID format)

int **capacity**

total number of sectors

int **sector_size**

sector size in bytes

int **read_block_len**

block length for reads

int **card_command_class**

Card Command Class for SD

int **tr_speed**

Max transfer speed

struct sdmmc_cid_t

Decoded values from SD card Card IDentification register

Public Members

int **mfg_id**

manufacturer identification number

int **oem_id**

OEM/product identification number

char **name**[8]

product name (MMC v1 has the longest)

int **revision**

product revision

int **serial**

product serial number

int **date**

manufacturing date

struct sdmmc_scr_t

Decoded values from SD Configuration Register

Public Members

int **sd_spec**

SD Physical layer specification version, reported by card

int **bus_width**

bus widths supported by card: BIT(0) —1-bit bus, BIT(2) —4-bit bus

struct sdmmc_ext_csd_t

Decoded values of Extended Card Specific Data

Public Members

uint8_t **power_class**

Power class used by the card

struct sdmmc_switch_func_rsp_t

SD SWITCH_FUNC response buffer

Public Members

uint32_t **data**[512 / 8 / sizeof(uint32_t)]

response data

struct sdmmc_command_t

SD/MMC command information

Public Members

uint32_t **opcode**
SD or MMC command index

uint32_t **arg**
SD/MMC command argument

sdmmc_response_t **response**
response buffer

void ***data**
buffer to send or read into

size_t **datalen**
length of data buffer

size_t **blklen**
block length

int **flags**
see below

esp_err_t **error**
error returned from transfer

int **timeout_ms**
response timeout, in milliseconds

struct sdmmc_host_t
SD/MMC Host description

This structure defines properties of SD/MMC host and functions of SD/MMC host which can be used by upper layers.

Public Members

uint32_t **flags**
flags defining host properties

int **slot**
slot number, to be passed to host functions

int **max_freq_khz**
max frequency supported by the host

float **io_voltage**
I/O voltage used by the controller (voltage switching is not supported)

esp_err_t (***init**)(void)
Host function to initialize the driver

esp_err_t (***set_bus_width**)(int slot, size_t width)
host function to set bus width

size_t (***get_bus_width**)(int slot)
host function to get bus width

esp_err_t (***set_bus_ddr_mode**)(int slot, bool ddr_enable)
host function to set DDR mode

esp_err_t (***set_card_clk**)(int slot, uint32_t freq_khz)
host function to set card clock frequency

esp_err_t (***do_transaction**)(int slot, *sdmmc_command_t* *cmdinfo)
host function to do a transaction

esp_err_t (***deinit**)(void)
host function to deinitialize the driver

esp_err_t (***io_int_enable**)(int slot)
Host function to enable SDIO interrupt line

esp_err_t (***io_int_wait**)(int slot, TickType_t timeout_ticks)
Host function to wait for SDIO interrupt line to be active

int **command_timeout_ms**
timeout, in milliseconds, of a single command. Set to 0 to use the default value.

struct sdmmc_card_t
SD/MMC card information structure

Public Members

sdmmc_host_t **host**
Host with which the card is associated

uint32_t **ocr**
OCR (Operation Conditions Register) value

sdmmc_cid_t **cid**
decoded CID (Card IDentification) register value

sdmmc_response_t **raw_cid**
raw CID of MMC card to be decoded after the CSD is fetched in the data transfer mode

sdmmc_csd_t **csd**
decoded CSD (Card-Specific Data) register value

sdmmc_scr_t **scr**
: decoded SCR (SD card Configuration Register) value

sdmmc_ext_csd_t **ext_csd**
: decoded EXT_CSD (Extended Card Specific Data) register value

uint16_t **rca**
: RCA (Relative Card Address)

uint16_t **max_freq_khz**
: Maximum frequency, in kHz, supported by the card

uint32_t **is_mem**
: Bit indicates if the card is a memory card

uint32_t **is_sdio**
: Bit indicates if the card is an IO card

uint32_t **is_mmc**
: Bit indicates if the card is MMC

uint32_t **num_io_functions**
: If is_sdio is 1, contains the number of IO functions on the card

uint32_t **log_bus_width**
: log2(bus width supported by card)

uint32_t **is_ddr**
: Card supports DDR mode

uint32_t **reserved**
: Reserved for future expansion

Macros

SDMMC_HOST_FLAG_1BIT
: host supports 1-line SD and MMC protocol

SDMMC_HOST_FLAG_4BIT
: host supports 4-line SD and MMC protocol

SDMMC_HOST_FLAG_8BIT
: host supports 8-line MMC protocol

SDMMC_HOST_FLAG_SPI
: host supports SPI protocol

SDMMC_HOST_FLAG_DDR
: host supports DDR mode for SD/MMC

`SDMMC_FREQ_DEFAULT`
SD/MMC Default speed (limited by clock divider)

`SDMMC_FREQ_HIGHSPEED`
SD High speed (limited by clock divider)

`SDMMC_FREQ_PROBING`
SD/MMC probing speed

`SDMMC_FREQ_52M`
MMC 52MHz speed

`SDMMC_FREQ_26M`
MMC 26MHz speed

Type Definitions

`typedef` uint32_t `sdmmc_response_t`[4]
SD/MMC command response buffer

2.6.3 非易失性存储库

[English]

简介

非易失性存储 (NVS) 库主要用于在 flash 中存储键值格式的数据。本文档将详细介绍 NVS 常用的一些概念。

底层存储

NVS 通过调用 `spi_flash_{read|write|erase}` API 对主 flash 的部分空间进行读、写、擦除操作，包括 `data` 类型和 `nvs` 子类型的所有分区。应用程序可调用 `nvs_open` API 选择使用带有 `nvs` 标签的分区，也可以通过调用 `nvs_open_from_part` API 选择使用指定名称的任意分区。

NVS 库后续版本可能会增加其他存储器后端，实现将数据保存至其他 flash 芯片（SPI 或 I2C 接口）、RTC 或 FRAM 中。

注解： 如果 NVS 分区被截断（例如，更改分区表布局时），则应擦除分区内容。可以使用 ESP-IDF 构建系统中的 `idf.py erase_flash` 命令擦除 flash 上的所有内容。

注解： NVS 最适合存储一些较小的数据，而非字符串或二进制大对象 (BLOB) 等较大的数据。如需存储较大的 BLOB 或者字符串，请考虑使用基于磨损均衡库的 FAT 文件系统。

键值对

NVS 的操作对象为键值对，其中键是 ASCII 字符串，当前支持最大键长为 15 个字符，值可以为以下几种类型：

- 整数型：`uint8_t`、`int8_t`、`uint16_t`、`int16_t`、`uint32_t`、`int32_t`、`uint64_t` 和 `int64_t`；
- 以 `\0` 结尾的字符串；
- 可变长度的二进制数据 (BLOB)

注解： 字符串值当前上限为 4000 字节，其中包括空终止符。BLOB 值上限为 508,000 字节或分区大小减去 4000 字节的 97.6%，以较低值为准。

后续可能会增加对 `float` 和 `double` 等其他类型数据的支持。

键必须唯一。为现有的键写入新的值可能产生如下结果：

- 如果新旧值数据类型相同，则更新值；
- 如果新旧值数据类型不同，则返回错误。

读取值时也会执行数据类型检查。如果读取操作的数据类型与该值的数据类型不匹配，则返回错误。

命名空间

为了减少不同组件之间键名的潜在冲突，NVS 将每个键值对分配给一个命名空间。命名空间的命名规则遵循键名的命名规则，即最多可占 15 个字符。命名空间的名称在调用 `nvs_open` 或 `nvs_open_from_part` 中指定，调用后将返回一个不透明句柄，用于后续调用 `nvs_get_*`、`nvs_set_*` 和 `nvs_commit` 函数。这样，一个句柄关联一个命名空间，键名便不会与其他命名空间中相同键名冲突。请注意，不同 NVS 分区中具有相同名称的命名空间将被视为不同的命名空间。

安全性、篡改性及鲁棒性

NVS 与 ESP32 flash 加密系统不直接兼容。但如果 NVS 加密与 ESP32 flash 加密一起使用时，数据仍可以加密形式存储。更多详情请参阅*NVS 加密*。

如果未启用 NVS 加密，任何对 flash 芯片有物理访问权限的人都可以修改、擦除或添加键值对。NVS 加密启用后，如果不知道相应的 NVS 加密密钥，则无法修改或添加键值对并将其识别为有效键值。但是，针对擦除操作没有相应的防篡改功能。

当 flash 处于不一致状态时，NVS 库会尝试恢复。在任何时间点关闭设备电源，然后重新打开电源，不会导致数据丢失；但如果关闭设备电源时正在写入新的键值对，这一键值对可能会丢失。该库还应当能对 flash 中的任意数据进行正确初始化。

内部实现

键值对日志

NVS 按顺序存储键值对，新的键值对添加在最后。因此，如需更新某一键值对，实际是在日志最后增加一对新的键值对，同时将旧的键值对标记为已擦除。

页面和条目

NVS 库在其操作中主要使用两个实体：页面和条目。页面是一个逻辑结构，用于存储部分的整体日志。逻辑页面对应 flash 的一个物理扇区，正在使用中的页面具有与之相关联的序列号。序列号赋予了页面顺序，较高的序列号对应较晚创建的页面。页面有以下几种状态：

空或未初始化 页面对应的 flash 扇区为空白状态（所有字节均为 `0xff`）。此时，页面未存储任何数据且没有关联的序列号。

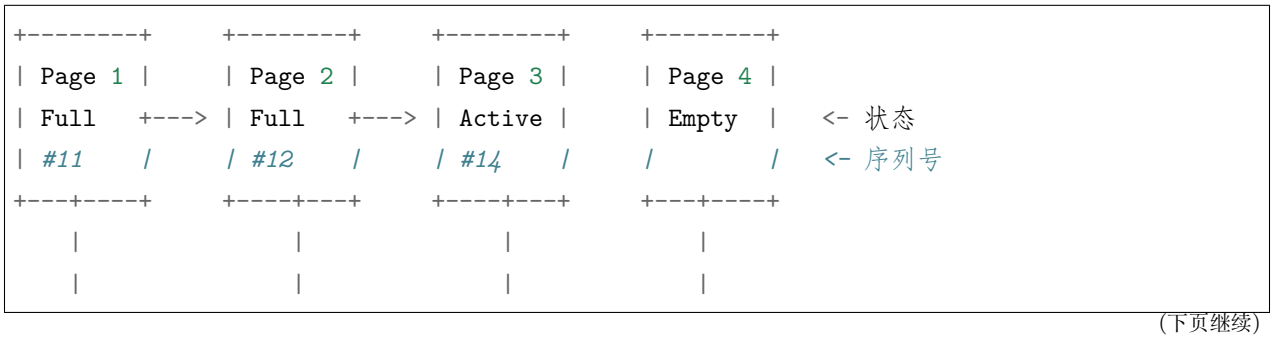
活跃状态 此时 flash 已完成初始化，页头部写入 flash，页面已具备有效序列号。页面中存在一些空条目，可写入数据。任意时刻，至多有一个页面处于活跃状态。

写满状态 Flash 已写满键值对，状态不再改变。用户无法向写满状态下的页面写入新键值对，但仍可将一些键值对标记为已擦除。

擦除状态 未擦除的键值对将移至其他页面，以便擦除当前页面。这一状态仅为暂时性状态，即 API 调用返回时，页面应脱离这一状态。如果设备突然断电，下次开机时，设备将继续把未擦除的键值对移至其他页面，并继续擦除当前页面。

损坏状态 页头部包含无效数据，无法进一步解析该页面中的数据，因此之前写入该页面的所有条目均无法访问。相应的 flash 扇区并不会被立即擦除，而是与其他处于未初始化状态的扇区一起等待后续使用。这一状态可能对调试有用。

Flash 扇区映射至逻辑页面并没有特定的顺序，NVS 库会检查存储在 flash 扇区的页面序列号，并根据序列号组织页面。



(下页继续)

(续上页)

```
    |               |              |              |
+---v------+  +-----v----+  +------v---+  +------v---+
| Sector 3 |  | Sector 0 |  | Sector 2 |  | Sector 1 |    <- 物理扇区
+----------+  +----------+  +----------+  +----------+
```

页面结构

当前，我们假设 flash 扇区大小为 4096 字节，并且 ESP32 flash 加密硬件在 32 字节块上运行。未来有可能引入一些编译时可配置项（可通过 menuconfig 进行配置），以适配具有不同扇区大小的 flash 芯片。但目前尚不清楚 SPI flash 驱动和 SPI flash cache 之类的系统组件是否支持其他扇区大小。

页面由头部、条目状态位图和条目三部分组成。为了实现与 ESP32 flash 加密功能兼容，条目大小设置为 32 字节。如果键值为整数型，条目则保存一个键值对；如果键值为字符串或 BLOB 类型，则条目仅保存一个键值对的部分内容（更多信息详见条目结构描述）。

页面结构如下图所示，括号内数字表示该部分的大小（以字节为单位）：

```
+----------+-------------+-------------+-------------------------+
| State (4) | Seq. no. (4) | version (1) | Unused (19) | CRC32 (4) |   页头部 (32)
+----------+-------------+-------------+-------------------------+
|                Entry state bitmap (32)                          |
+-----------------------------------------------------------------+
|                        Entry 0 (32)                             |
+-----------------------------------------------------------------+
|                        Entry 1 (32)                             |
+-----------------------------------------------------------------+
/                                                                 /
/                                                                 /
+-----------------------------------------------------------------+
|                        Entry 125 (32)                           |
+-----------------------------------------------------------------+
```

头部和条目状态位图写入 flash 时不加密。如果启用了 ESP32 flash 加密功能，则条目写入 flash 时将会加密。

通过将 0 写入某些位可以定义页面状态值，表示状态改变。因此，如果需要变更页面状态，并不一定要擦除页面，除非要将其变更为擦除状态。

头部中的 `version` 字段反映了所用的 NVS 格式版本。为实现向后兼容，版本升级从 0xff 开始依次递减（例如，version-1 为 0xff，version-2 为 0xfe 等）。

头部中 CRC32 值是由不包含状态值的条目计算所得（4 到 28 字节）。当前未使用的条目用 `0xff` 字节填充。

条目结构和条目状态位图详细信息见下文描述。

条目和条目状态位图

每个条目可处于以下三种状态之一，每个状态在条目状态位图中用两位表示。位图中的最后四位 (256 - 2 * 126) 未使用。

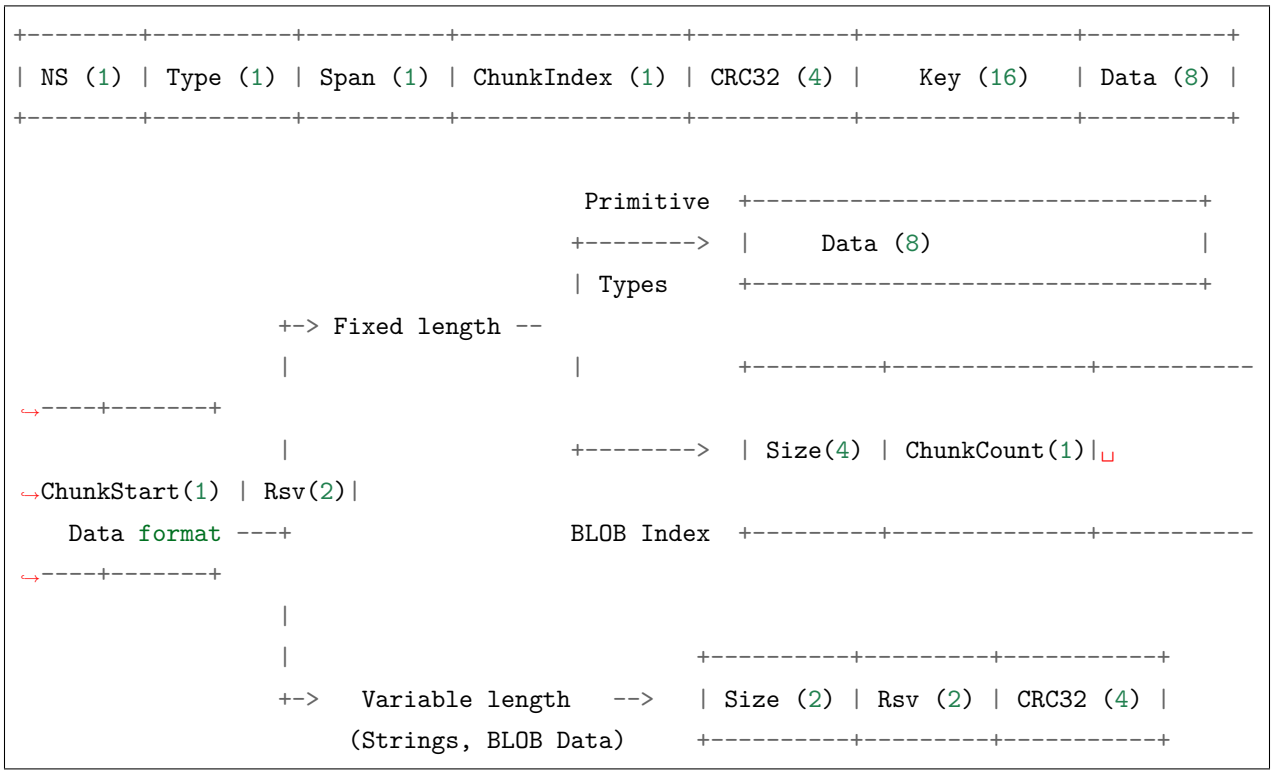
空 (2' b11) 条目还未写入任何内容，处于未初始化状态（全部字节为 0xff）。

写入（2' b10） 一个键值对（或跨多个条目的键值对的部分内容）已写入条目中。

擦除（2' b00） 条目中的键值对已丢弃，条目内容不再解析。

条目结构

如果键值类型为基础类型，即 1 - 8 个字节长度的整数型，条目将保存一个键值对；如果键值类型为字符串或 BLOB 类型，条目将保存整个键值对的部分内容。另外，如果键值为字符串类型且跨多个条目，则键值所跨的所有条目均保存在同一页面。BLOB 则可以切分为多个块，实现跨多个页面。BLOB 索引是一个附加的固定长度元数据条目，用于追踪 BLOB 块。目前条目仍支持早期 BLOB 格式（可读取可修改），但这些 BLOB 一经修改，即以新格式储存至条目。



条目结构中各个字段含义如下：

命名空间 (NS, NameSpace) 该条目的命名空间索引，详细信息见命名空间实现章节。

类型 (Type) 一个字节表示的值的数据类型，可能的类型见 nvs_types.h 中 ItemType 枚举。

跨度 (Span) 该键值对所用的条目数量。如果键值为整数型，条目数量即为 1。如果键值为字符串或 BLOB，则条目数量取决于值的长度。

块索引 (ChunkIndex) 用于存储 BLOB 类型数据块的索引。如果键值为其他数据类型，则此处索引应写入 `0xff`。

CRC32 对条目下所有字节进行校验，所得的校验和（CRC32 字段不计算在内）。

键 (Key) 即以零结尾的 ASCII 字符串，字符串最长为 15 字节，不包含最后一个字节的 NULL (`\0`) 终止符。

数据 (Data) 如果键值类型为整数型，则数据字段仅包含键值。如果键值小于八个字节，使用 `0xff` 填充未使用的部分（右侧）。

如果键值类型为 BLOB 索引条目，则该字段的八个字节将保存以下数据块信息：

- **块大小** 整个 BLOB 数据的大小（以字节为单位）。该字段仅用于 BLOB 索引类型条目。
- **ChunkCount** 存储过程中 BLOB 分成的数据块数量。该字段仅用于 BLOB 索引类型条目。
- **ChunkStart** BLOB 第一个数据块的块索引，后续数据块索引依次递增，步长为 1。该字段仅用于 BLOB 索引类型条目。

如果键值类型为字符串或 BLOB 数据块，数据字段的这八个字节将保存该键值的一些附加信息，如下所示：

- **数据大小** 实际数据的大小（以字节为单位）。如果键值类型为字符串，此字段也应将零终止符包含在内。此字段仅用于字符串和 BLOB 类型条目。
- **CRC32** 数据所有字节的校验和，该字段仅用于字符串和 BLOB 类型条目。

可变长度值（字符串和 BLOB）写入后续条目，每个条目 32 字节。第一个条目的 span 字段将指明使用了多少条目。

命名空间

如上所述，每个键值对属于一个命名空间。命名空间标识符（字符串）也作为键值对的键，存储在索引为 0 的命名空间中。与这些键对应的值就是这些命名空间的索引。

```
+-----------------------------------------+
| NS=0 Type=uint8_t Key="wifi" Value=1    |   Entry describing namespace "wifi"
+-----------------------------------------+
| NS=1 Type=uint32_t Key="channel" Value=6  |   Key "channel" in namespace "wifi"
+-----------------------------------------+
| NS=0 Type=uint8_t Key="pwm" Value=2     |   Entry describing namespace "pwm"
+-----------------------------------------+
| NS=2 Type=uint16_t Key="channel" Value=20 |   Key "channel" in namespace "pwm"
+-----------------------------------------+
```

条目哈希列表

为了减少对 flash 执行的读操作次数，Page 类对象均设有一个列表，包含一对数据：条目索引和条目哈希值。该列表可大大提高检索速度，而无需迭代所有条目并逐个从 flash 中读取。`Page::findItem` 首先从哈希列表中检索条目哈希值，如果条目存在，则在页面内给出条目索引。由于哈希冲突，在哈希列表中检索条目哈希值可能会得到不同的条目，对 flash 中条目再次迭代可解决这一冲突。

哈希列表中每个节点均包含一个 24 位哈希值和 8 位条目索引。哈希值根据条目命名空间、键名和块索引由 CRC32 计算所得，计算结果保留 24 位。为减少将 32 位条目存储在链表中的开销，链表采用了数组的双向链表。每个数组占用 128 个字节，包含 29 个条目、两个链表指针和一个 32 位计数字段。因此，每页额外需要的 RAM 最少为 128 字节，最多为 640 字节。

NVS 加密

NVS 分区内存储的数据可使用 AES-XTS 进行加密，类似于 IEEE P1619 磁盘加密标准中提到的加密方式。为了实现加密，每个条目被均视为一个扇区，并将条目相对地址（相对于分区开头）传递给加密算法，用作扇区号。NVS 加密所需的密钥存储于其他分区，并进行了*flash 加密*。因此，在使用 NVS 加密前应先启用*flash 加密*。

NVS 密钥分区

应用程序如果想使用 NVS 加密，则需要编译进一个类型为 `data`，子类型为 `key` 的密钥分区。该分区应标记为已加密，且最小为 4096 字节，具体结构见下表。如需了解更多详细信息，请参考分区表。

```
+-----------+--------------+-------------+----+
|               XTS encryption key(32)         |
+----------------------------------------------+
|               XTS tweak key (32)             |
+----------------------------------------------+
|                   CRC32(4)                   |
+----------------------------------------------+
```

使用 NVS 分区生成程序生成上述分区表，并烧录至设备。由于分区已标记为已加密，而且启用了*flash 加密*，引导程序在首次启动时将使用 flash 加密对密钥分区进行加密。您也可以在设备启动后调用 `nvs_flash.h` 提供的 `nvs_flash_generate_keys` API 生成加密密钥，然后再将密钥以加密形式写入密钥分区。

应用程序可以使用不同的密钥对不同的 NVS 分区进行加密，这样就会需要多个加密密钥分区。应用程序应为加解密操作提供正确的密钥或密钥分区。

加密读取/写入

`nvs_get_*` 和 `nvs_set_*` 等 NVS API 函数同样可以对 NVS 加密分区执行读写操作。但用于初始化 NVS 非加密分区和加密分区的 API 则有所不同：初始化 NVS 非加密分区

可以使用 nvs_flash_init 和 nvs_flash_init_partition，但初始化 NVS 加密分区则需调用 nvs_flash_secure_init 和 nvs_flash_secure_init_partition。上述 API 函数所需的 nvs_sec_cfg_t 结构可使用 nvs_flash_generate_keys 或者 nvs_flash_read_security_cfg 进行填充。

应用程序如需在加密状态下执行 NVS 读写操作，应遵循以下步骤：

1. 使用 esp_partition_find* API 查找密钥分区和 NVS 数据分区；
2. 使用 nvs_flash_read_security_cfg 或 nvs_flash_generate_keys API 填充 nvs_sec_cfg_t 结构；
3. 使用 nvs_flash_secure_init 或 nvs_flash_secure_init_partition API 初始化 NVS flash 分区；
4. 使用 nvs_open 或 nvs_open_from_part API 打开命名空间；
5. 使用 nvs_get_* 或 nvs_set_* API 执行 NVS 读取/写入操作；
6. 使用 nvs_flash_deinit API 释放已初始化的 NVS 分区。

NVS 迭代器

迭代器允许根据指定的分区名称、命名空间和数据类型轮询 NVS 中存储的键值对。

您可以使用以下函数，执行相关操作：

- nvs_entry_find：返回一个不透明句柄，用于后续调用 nvs_entry_next 和 nvs_entry_info 函数；
- nvs_entry_next：返回指向下一个键值对的迭代器；
- nvs_entry_info：返回每个键值对的信息。

如果未找到符合标准的键值对，nvs_entry_find 和 nvs_entry_next 将返回 NULL，此时不必释放迭代器。若不再需要迭代器，可使用 nvs_release_iterator 释放迭代器。

NVS 分区生成程序

NVS 分区生成程序帮助生成 NVS 分区二进制文件，可使用烧录程序将二进制文件单独烧录至特定分区。烧录至分区上的键值对由 CSV 文件提供，详情请参考*NVS 分区生成程序*。

应用示例

ESP-IDF storage 目录下提供了两个代码示例：

storage/nvs_rw_value

> 演示如何读取及写入 NVS 单个整数值。
>
> 此示例中的值表示 ESP32 模组重启次数。NVS 中数据不会因为模组重启而丢失，因此只有将这一值存储于 NVS 中，才能起到重启次数计数器的作用。
>
> 该示例也演示了如何检测读取/写入操作是否成功，以及某个特定值是否在 NVS 中尚未初始化。诊断程序以纯文本形式提供，帮助您追踪程序流程，及时发现问题。

storage/nvs_rw_blob

演示如何读取及写入 NVS 单个整数值和 Blob（二进制大对象），并在 NVS 中存储这一数值，即便 ESP32 模组重启也不会消失。

- value - 记录 ESP32 模组软重启次数和硬重启次数。
- blob - 内含记录模组运行次数的表格。此表格将被从 NVS 读取至动态分配的 RAM 上。每次手动软重启后，表格内运行次数即增加一次，新加的运行次数被写入 NVS。下拉 GPIO0 即可手动软重启。

该示例也演示了如何执行诊断程序以检测读取/写入操作是否成功。

API 参考

Header File

- nvs_flash/include/nvs_flash.h

Functions

esp_err_t **nvs_flash_init**(void)

Initialize the default NVS partition.

This API initialises the default NVS partition. The default NVS partition is the one that is labeled "nvs" in the partition table.

Return

- ESP_OK if storage was successfully initialized.
- ESP_ERR_NVS_NO_FREE_PAGES if the NVS storage contains no empty pages (which may happen if NVS partition was truncated)
- ESP_ERR_NOT_FOUND if no partition with label "nvs" is found in the partition table
- one of the error codes from the underlying flash storage driver

esp_err_t **nvs_flash_init_partition**(**const** char *_partition_label_)

Initialize NVS flash storage for the specified partition.

Return

- ESP_OK if storage was successfully initialized.
- ESP_ERR_NVS_NO_FREE_PAGES if the NVS storage contains no empty pages (which may happen if NVS partition was truncated)
- ESP_ERR_NOT_FOUND if specified partition is not found in the partition table

- one of the error codes from the underlying flash storage driver

Parameters

- `partition_label`: Label of the partition. Must be no longer than 16 characters.

esp_err_t **nvs_flash_deinit**(void)

Deinitialize NVS storage for the default NVS partition.

Default NVS partition is the partition with "nvs" label in the partition table.

Return

- ESP_OK on success (storage was deinitialized)
- ESP_ERR_NVS_NOT_INITIALIZED if the storage was not initialized prior to this call

esp_err_t **nvs_flash_deinit_partition**(**const** char *_partition_label_)

Deinitialize NVS storage for the given NVS partition.

Return

- ESP_OK on success
- ESP_ERR_NVS_NOT_INITIALIZED if the storage for given partition was not initialized prior to this call

Parameters

- `partition_label`: Label of the partition

esp_err_t **nvs_flash_erase**(void)

Erase the default NVS partition.

This function erases all contents of the default NVS partition (one with label "nvs")

Return

- ESP_OK on success
- ESP_ERR_NOT_FOUND if there is no NVS partition labeled "nvs" in the partition table

esp_err_t **nvs_flash_erase_partition**(**const** char *_part_name_)

Erase specified NVS partition.

This function erases all contents of specified NVS partition

Return

- ESP_OK on success
- ESP_ERR_NOT_FOUND if there is no NVS partition with the specified name in the partition table

Parameters

- `part_name`: Name (label) of the partition to be erased

esp_err_t **nvs_flash_secure_init**(*nvs_sec_cfg_t* **cfg*)

Initialize the default NVS partition.

This API initialises the default NVS partition. The default NVS partition is the one that is labeled "nvs" in the partition table.

Return

- ESP_OK if storage was successfully initialized.
- ESP_ERR_NVS_NO_FREE_PAGES if the NVS storage contains no empty pages (which may happen if NVS partition was truncated)
- ESP_ERR_NOT_FOUND if no partition with label "nvs" is found in the partition table
- one of the error codes from the underlying flash storage driver

Parameters

- `cfg`: Security configuration (keys) to be used for NVS encryption/decryption. If cfg is NULL, no encryption is used.

esp_err_t **nvs_flash_secure_init_partition**(const char **partition_label*, *nvs_sec_cfg_t* **cfg*)

Initialize NVS flash storage for the specified partition.

Return

- ESP_OK if storage was successfully initialized.
- ESP_ERR_NVS_NO_FREE_PAGES if the NVS storage contains no empty pages (which may happen if NVS partition was truncated)
- ESP_ERR_NOT_FOUND if specified partition is not found in the partition table
- one of the error codes from the underlying flash storage driver

Parameters

- `partition_label`: Label of the partition. Note that internally a reference to passed value is kept and it should be accessible for future operations
- `cfg`: Security configuration (keys) to be used for NVS encryption/decryption. If cfg is null, no encryption/decryption is used.

esp_err_t **nvs_flash_generate_keys**(const *esp_partition_t* **partition*, *nvs_sec_cfg_t* **cfg*)

Generate and store NVS keys in the provided esp partition.

Return -ESP_OK, if cfg was read successfully; -or error codes from esp_partition_write/erase APIs.

Parameters

- `partition`: Pointer to partition structure obtained using esp_partition_find_first or esp_partition_get. Must be non-NULL.
- `cfg`: Pointer to nvs security configuration structure. Pointer must be non-NULL. Generated keys will be populated in this structure.

esp_err_t **nvs_flash_read_security_cfg**(**const** *esp_partition_t* **partition*, *nvs_sec_cfg_t* **cfg*)

Read NVS security configuration from a partition.

Note Provided parition is assumed to be marked ‘encrypted’.

Return -ESP_OK, if cfg was read successfully; -ESP_ERR_NVS_KEYS_NOT_INITIALIZED, if the partition is not yet written with keys. -ESP_ERR_NVS_CORRUPT_KEY_PART, if the partition containing keys is found to be corrupt -or error codes from esp_partition_read API.

Parameters

- `partition`: Pointer to partition structure obtained using esp_partition_find_first or esp_partition_get. Must be non-NULL.
- `cfg`: Pointer to nvs security configuration structure. Pointer must be non-NULL.

Structures

struct nvs_sec_cfg_t

Key for encryption and decryption.

Public Members

uint8_t **eky**[**NVS_KEY_SIZE**]

XTS encryption and decryption key

uint8_t **tky**[**NVS_KEY_SIZE**]

XTS tweak key

Macros

NVS_KEY_SIZE

Header File

- nvs_flash/include/nvs.h

Functions

esp_err_t **nvs_set_i8**(*nvs_handle_t handle*, **const** char **key*, int8_t *value*)

set value for given key

This family of functions set value for the key, given its name. Note that actual storage will not be updated until nvs_commit function is called.

Return

- ESP_OK if value was set successfully
- ESP_ERR_NVS_INVALID_HANDLE if handle has been closed or is NULL
- ESP_ERR_NVS_READ_ONLY if storage handle was opened as read only
- ESP_ERR_NVS_INVALID_NAME if key name doesn't satisfy constraints
- ESP_ERR_NVS_NOT_ENOUGH_SPACE if there is not enough space in the underlying storage to save the value
- ESP_ERR_NVS_REMOVE_FAILED if the value wasn't updated because flash write operation has failed. The value was written however, and update will be finished after re-initialization of nvs, provided that flash operation doesn't fail again.
- ESP_ERR_NVS_VALUE_TOO_LONG if the string value is too long

Parameters

- `handle`: Handle obtained from nvs_open function. Handles that were opened read only cannot be used.
- `key`: Key name. Maximal length is determined by the underlying implementation, but is guaranteed to be at least 15 characters. Shouldn't be empty.
- `value`: The value to set. For strings, the maximum length (including null character) is 4000 bytes.

esp_err_t **nvs_set_u8**(*nvs_handle_t handle*, **const** char **key*, uint8_t *value*)

esp_err_t **nvs_set_i16**(*nvs_handle_t handle*, **const** char **key*, int16_t *value*)

esp_err_t **nvs_set_u16**(*nvs_handle_t handle*, **const** char **key*, uint16_t *value*)

esp_err_t **nvs_set_i32**(*nvs_handle_t handle*, **const** char **key*, int32_t *value*)

esp_err_t **nvs_set_u32**(*nvs_handle_t handle*, **const** char **key*, uint32_t *value*)

esp_err_t **nvs_set_i64**(*nvs_handle_t handle*, **const** char **key*, int64_t *value*)

esp_err_t **nvs_set_u64**(*nvs_handle_t handle*, **const** char **key*, uint64_t *value*)

esp_err_t **nvs_set_str**(*nvs_handle_t handle*, **const** char **key*, **const** char **value*)

esp_err_t **nvs_get_i8**(*nvs_handle_t handle*, **const** char ***key*, int8_t ***out_value*)

get value for given key

These functions retrieve value for the key, given its name. If key does not exist, or the requested variable type doesn't match the type which was used when setting a value, an error is returned.

In case of any error, out_value is not modified.

All functions expect out_value to be a pointer to an already allocated variable of the given type.

```
// Example of using nvs_get_i32:
int32_t max_buffer_size = 4096; // default value
esp_err_t err = nvs_get_i32(my_handle, "max_buffer_size", &max_buffer_size);
assert(err == ESP_OK || err == ESP_ERR_NVS_NOT_FOUND);
// if ESP_ERR_NVS_NOT_FOUND was returned, max_buffer_size will still
// have its default value.
```

Return

- ESP_OK if the value was retrieved successfully
- ESP_ERR_NVS_NOT_FOUND if the requested key doesn't exist
- ESP_ERR_NVS_INVALID_HANDLE if handle has been closed or is NULL
- ESP_ERR_NVS_INVALID_NAME if key name doesn't satisfy constraints
- ESP_ERR_NVS_INVALID_LENGTH if length is not sufficient to store data

Parameters

- `handle`: Handle obtained from nvs_open function.
- `key`: Key name. Maximal length is determined by the underlying implementation, but is guaranteed to be at least 15 characters. Shouldn't be empty.
- `out_value`: Pointer to the output value. May be NULL for nvs_get_str and nvs_get_blob, in this case required length will be returned in length argument.

esp_err_t **nvs_get_u8**(*nvs_handle_t handle*, **const** char ***key*, uint8_t ***out_value*)

esp_err_t **nvs_get_i16**(*nvs_handle_t handle*, **const** char ***key*, int16_t ***out_value*)

esp_err_t **nvs_get_u16**(*nvs_handle_t handle*, **const** char ***key*, uint16_t ***out_value*)

esp_err_t **nvs_get_i32**(*nvs_handle_t handle*, **const** char ***key*, int32_t ***out_value*)

esp_err_t **nvs_get_u32**(*nvs_handle_t handle*, **const** char ***key*, uint32_t ***out_value*)

esp_err_t **nvs_get_i64**(*nvs_handle_t handle*, **const** char ***key*, int64_t ***out_value*)

esp_err_t **nvs_get_u64**(*nvs_handle_t handle*, **const** char ***key*, uint64_t ***out_value*)

esp_err_t **nvs_get_str**(*nvs_handle_t handle*, **const** char ***key*, char ***out_value*, size_t ***length*)

get value for given key

These functions retrieve the data of an entry, given its key. If key does not exist, or the requested variable type doesn't match the type which was used when setting a value, an error is returned.

In case of any error, out_value is not modified.

All functions expect out_value to be a pointer to an already allocated variable of the given type.

nvs_get_str and nvs_get_blob functions support WinAPI-style length queries. To get the size necessary to store the value, call nvs_get_str or nvs_get_blob with zero out_value and non-zero pointer to length. Variable pointed to by length argument will be set to the required length. For nvs_get_str, this length includes the zero terminator. When calling nvs_get_str and nvs_get_blob with non-zero out_value, length has to be non-zero and has to point to the length available in out_value. It is suggested that nvs_get/set_str is used for zero-terminated C strings, and nvs_get/set_blob used for arbitrary data structures.

```
// Example (without error checking) of using nvs_get_str to get a string into␣
↪dynamic array:
size_t required_size;
nvs_get_str(my_handle, "server_name", NULL, &required_size);
char* server_name = malloc(required_size);
nvs_get_str(my_handle, "server_name", server_name, &required_size);

// Example (without error checking) of using nvs_get_blob to get a binary data
into a static array:
uint8_t mac_addr[6];
size_t size = sizeof(mac_addr);
nvs_get_blob(my_handle, "dst_mac_addr", mac_addr, &size);
```

Return

- ESP_OK if the value was retrieved successfully
- ESP_ERR_NVS_NOT_FOUND if the requested key doesn't exist
- ESP_ERR_NVS_INVALID_HANDLE if handle has been closed or is NULL
- ESP_ERR_NVS_INVALID_NAME if key name doesn't satisfy constraints
- ESP_ERR_NVS_INVALID_LENGTH if length is not sufficient to store data

Parameters

- `handle`: Handle obtained from nvs_open function.
- `key`: Key name. Maximal length is determined by the underlying implementation, but is guaranteed to be at least 15 characters. Shouldn't be empty.

- `out_value`: Pointer to the output value. May be NULL for nvs_get_str and nvs_get_blob, in this case required length will be returned in length argument.
- `length`: A non-zero pointer to the variable holding the length of out_value. In case out_value a zero, will be set to the length required to hold the value. In case out_value is not zero, will be set to the actual length of the value written. For nvs_get_str this includes zero terminator.

esp_err_t **nvs_get_blob**(*nvs_handle_t handle*, **const** char ***key*, void ***out_value*, size_t ***length*)

esp_err_t **nvs_open**(**const** char ***name*, *nvs_open_mode_t open_mode*, *nvs_handle_t* ***out_handle*)

Open non-volatile storage with a given namespace from the default NVS partition.

Multiple internal ESP-IDF and third party application modules can store their key-value pairs in the NVS module. In order to reduce possible conflicts on key names, each module can use its own namespace. The default NVS partition is the one that is labelled "nvs" in the partition table.

Return

- ESP_OK if storage handle was opened successfully
- ESP_ERR_NVS_NOT_INITIALIZED if the storage driver is not initialized
- ESP_ERR_NVS_PART_NOT_FOUND if the partition with label "nvs" is not found
- ESP_ERR_NVS_NOT_FOUND id namespace doesn't exist yet and mode is NVS_READONLY
- ESP_ERR_NVS_INVALID_NAME if namespace name doesn't satisfy constraints
- other error codes from the underlying storage driver

Parameters

- `name`: Namespace name. Maximal length is determined by the underlying implementation, but is guaranteed to be at least 15 characters. Shouldn't be empty.
- `open_mode`: NVS_READWRITE or NVS_READONLY. If NVS_READONLY, will open a handle for reading only. All write requests will be rejected for this handle.
- `out_handle`: If successful (return code is zero), handle will be returned in this argument.

esp_err_t **nvs_open_from_partition**(**const** char ***part_name*, **const** char ***name*, *nvs_open_mode_t open_mode*, *nvs_handle_t* ***out_handle*)

Open non-volatile storage with a given namespace from specified partition.

The behaviour is same as nvs_open() API. However this API can operate on a specified NVS partition instead of default NVS partition. Note that the specified partition must be registered with NVS using nvs_flash_init_partition() API.

Return

- ESP_OK if storage handle was opened successfully
- ESP_ERR_NVS_NOT_INITIALIZED if the storage driver is not initialized
- ESP_ERR_NVS_PART_NOT_FOUND if the partition with specified name is not found
- ESP_ERR_NVS_NOT_FOUND id namespace doesn't exist yet and mode is NVS_READONLY
- ESP_ERR_NVS_INVALID_NAME if namespace name doesn't satisfy constraints
- other error codes from the underlying storage driver

Parameters

- `part_name`: Label (name) of the partition of interest for object read/write/erase
- `name`: Namespace name. Maximal length is determined by the underlying implementation, but is guaranteed to be at least 15 characters. Shouldn't be empty.
- `open_mode`: NVS_READWRITE or NVS_READONLY. If NVS_READONLY, will open a handle for reading only. All write requests will be rejected for this handle.
- `out_handle`: If successful (return code is zero), handle will be returned in this argument.

esp_err_t **nvs_set_blob**(*nvs_handle_t handle*, **const** char ***key*, **const** void ***value*, size_t *length*)

set variable length binary value for given key

This family of functions set value for the key, given its name. Note that actual storage will not be updated until nvs_commit function is called.

Return

- ESP_OK if value was set successfully
- ESP_ERR_NVS_INVALID_HANDLE if handle has been closed or is NULL
- ESP_ERR_NVS_READ_ONLY if storage handle was opened as read only
- ESP_ERR_NVS_INVALID_NAME if key name doesn't satisfy constraints
- ESP_ERR_NVS_NOT_ENOUGH_SPACE if there is not enough space in the underlying storage to save the value
- ESP_ERR_NVS_REMOVE_FAILED if the value wasn't updated because flash write operation has failed. The value was written however, and update will be finished after re-initialization of nvs, provided that flash operation doesn't fail again.
- ESP_ERR_NVS_VALUE_TOO_LONG if the value is too long

Parameters

- `handle`: Handle obtained from nvs_open function. Handles that were opened read only cannot be used.

- `key`: Key name. Maximal length is 15 characters. Shouldn't be empty.
- `value`: The value to set.
- `length`: length of binary value to set, in bytes; Maximum length is 508000 bytes or (97.6% of the partition size - 4000) bytes whichever is lower.

esp_err_t **nvs_erase_key**(*nvs_handle_t handle*, **const** char ***key*)

Erase key-value pair with given key name.

Note that actual storage may not be updated until nvs_commit function is called.

Return

- ESP_OK if erase operation was successful
- ESP_ERR_NVS_INVALID_HANDLE if handle has been closed or is NULL
- ESP_ERR_NVS_READ_ONLY if handle was opened as read only
- ESP_ERR_NVS_NOT_FOUND if the requested key doesn't exist
- other error codes from the underlying storage driver

Parameters

- `handle`: Storage handle obtained with nvs_open. Handles that were opened read only cannot be used.
- `key`: Key name. Maximal length is determined by the underlying implementation, but is guaranteed to be at least 15 characters. Shouldn't be empty.

esp_err_t **nvs_erase_all**(*nvs_handle_t handle*)

Erase all key-value pairs in a namespace.

Note that actual storage may not be updated until nvs_commit function is called.

Return

- ESP_OK if erase operation was successful
- ESP_ERR_NVS_INVALID_HANDLE if handle has been closed or is NULL
- ESP_ERR_NVS_READ_ONLY if handle was opened as read only
- other error codes from the underlying storage driver

Parameters

- `handle`: Storage handle obtained with nvs_open. Handles that were opened read only cannot be used.

esp_err_t **nvs_commit**(*nvs_handle_t handle*)

Write any pending changes to non-volatile storage.

After setting any values, nvs_commit() must be called to ensure changes are written to non-volatile storage. Individual implementations may write to storage at other times, but this is not guaranteed.

Return

- ESP_OK if the changes have been written successfully
- ESP_ERR_NVS_INVALID_HANDLE if handle has been closed or is NULL
- other error codes from the underlying storage driver

Parameters

- **handle**: Storage handle obtained with nvs_open. Handles that were opened read only cannot be used.

void **nvs_close**(*nvs_handle_t handle*)

Close the storage handle and free any allocated resources.

This function should be called for each handle opened with nvs_open once the handle is not in use any more. Closing the handle may not automatically write the changes to nonvolatile storage. This has to be done explicitly using nvs_commit function. Once this function is called on a handle, the handle should no longer be used.

Parameters

- **handle**: Storage handle to close

esp_err_t **nvs_get_stats**(**const** char **part_name*, *nvs_stats_t* **nvs_stats*)

Fill structure *nvs_stats_t*. It provides info about used memory the partition.

This function calculates to runtime the number of used entries, free entries, total entries, and amount namespace in partition.

```
// Example of nvs_get_stats() to get the number of used entries and free entries:
nvs_stats_t nvs_stats;
nvs_get_stats(NULL, &nvs_stats);
printf("Count: UsedEntries = (%d), FreeEntries = (%d), AllEntries = (%d)\n",
       nvs_stats.used_entries, nvs_stats.free_entries, nvs_stats.total_entries);
```

Return

- ESP_OK if the changes have been written successfully. Return param nvs_stats will be filled.

- ESP_ERR_NVS_PART_NOT_FOUND if the partition with label "name" is not found. Return param nvs_stats will be filled 0.
- ESP_ERR_NVS_NOT_INITIALIZED if the storage driver is not initialized. Return param nvs_stats will be filled 0.
- ESP_ERR_INVALID_ARG if nvs_stats equal to NULL.
- ESP_ERR_INVALID_STATE if there is page with the status of INVALID. Return param nvs_stats will be filled not with correct values because not all pages will be counted. Counting will be interrupted at the first INVALID page.

Parameters

- `part_name`: Partition name NVS in the partition table. If pass a NULL than will use NVS_DEFAULT_PART_NAME ("nvs").
- `nvs_stats`: Returns filled structure nvs_states_t. It provides info about used memory the partition.

esp_err_t **nvs_get_used_entry_count**(*nvs_handle_t handle*, size_t **used_entries*)

Calculate all entries in a namespace.

Note that to find out the total number of records occupied by the namespace, add one to the returned value used_entries (if err is equal to ESP_OK). Because the name space entry takes one entry.

```
// Example of nvs_get_used_entry_count() to get amount of all key-value pairs in␣
↪one namespace:
nvs_handle_t handle;
nvs_open("namespace1", NVS_READWRITE, &handle);
...
size_t used_entries;
size_t total_entries_namespace;
if(nvs_get_used_entry_count(handle, &used_entries) == ESP_OK){
    // the total number of records occupied by the namespace
    total_entries_namespace = used_entries + 1;
}
```

Return

- ESP_OK if the changes have been written successfully. Return param used_entries will be filled valid value.
- ESP_ERR_NVS_NOT_INITIALIZED if the storage driver is not initialized. Return param used_entries will be filled 0.
- ESP_ERR_NVS_INVALID_HANDLE if handle has been closed or is NULL. Return param used_entries will be filled 0.

- ESP_ERR_INVALID_ARG if used_entries equal to NULL.
- Other error codes from the underlying storage driver. Return param used_entries will be filled 0.

Parameters

- `handle`: Handle obtained from nvs_open function.
- `used_entries`: Returns amount of used entries from a namespace.

nvs_iterator_t **nvs_entry_find**(**const** char *_part_name_, **const** char *_namespace_name_, *nvs_type_t type*)

Create an iterator to enumerate NVS entries based on one or more parameters.

```
// Example of listing all the key-value pairs of any type under specified partition␣
↪and namespace
nvs_iterator_t it = nvs_entry_find(partition, namespace, NVS_TYPE_ANY);
while (it != NULL) {
        nvs_entry_info_t info;
        nvs_entry_info(it, &info);
        it = nvs_entry_next(it);
        printf("key '%s', type '%d' \n", info.key, info.type);
};
// Note: no need to release iterator obtained from nvs_entry_find function when
//       nvs_entry_find or nvs_entry_next function return NULL, indicating no other
//       element for specified criteria was found.
}
```

Return Iterator used to enumerate all the entries found, or NULL if no entry satisfying criteria was found. Iterator obtained through this function has to be released using nvs_release_iterator when not used any more.

Parameters

- `part_name`: Partition name
- `namespace_name`: Set this value if looking for entries with a specific namespace. Pass NULL otherwise.
- `type`: One of nvs_type_t values.

nvs_iterator_t **nvs_entry_next**(*nvs_iterator_t iterator*)

Returns next item matching the iterator criteria, NULL if no such item exists.

Note that any copies of the iterator will be invalid after this call.

Return NULL if no entry was found, valid nvs_iterator_t otherwise.

Parameters

- `iterator`: Iterator obtained from nvs_entry_find function. Must be non-NULL.

void **nvs_entry_info**(*nvs_iterator_t iterator*, *nvs_entry_info_t *out_info*)

Fills *nvs_entry_info_t* structure with information about entry pointed to by the iterator.

Parameters

- `iterator`: Iterator obtained from nvs_entry_find or nvs_entry_next function. Must be non-NULL.
- `out_info`: Structure to which entry information is copied.

void **nvs_release_iterator**(*nvs_iterator_t iterator*)

Release iterator.

Parameters

- `iterator`: Release iterator obtained from nvs_entry_find function. NULL argument is allowed.

Structures

struct nvs_entry_info_t

information about entry obtained from nvs_entry_info function

Public Members

char **namespace_name**[16]

Namespace to which key-value belong

char **key**[16]

Key of stored key-value pair

nvs_type_t **type**

Type of stored key-value pair

struct nvs_stats_t

Note Info about storage space NVS.

Public Members

size_t **used_entries**

Amount of used entries.

size_t **free_entries**
: Amount of free entries.

size_t **total_entries**
: Amount all available entries.

size_t **namespace_count**
: Amount name space.

Macros

ESP_ERR_NVS_BASE
: Starting number of error codes

ESP_ERR_NVS_NOT_INITIALIZED
: The storage driver is not initialized

ESP_ERR_NVS_NOT_FOUND
: Id namespace doesn’t exist yet and mode is NVS_READONLY

ESP_ERR_NVS_TYPE_MISMATCH
: The type of set or get operation doesn’t match the type of value stored in NVS

ESP_ERR_NVS_READ_ONLY
: Storage handle was opened as read only

ESP_ERR_NVS_NOT_ENOUGH_SPACE
: There is not enough space in the underlying storage to save the value

ESP_ERR_NVS_INVALID_NAME
: Namespace name doesn’t satisfy constraints

ESP_ERR_NVS_INVALID_HANDLE
: Handle has been closed or is NULL

ESP_ERR_NVS_REMOVE_FAILED
: The value wasn’t updated because flash write operation has failed. The value was written however, and update will be finished after re-initialization of nvs, provided that flash operation doesn’t fail again.

ESP_ERR_NVS_KEY_TOO_LONG
: Key name is too long

ESP_ERR_NVS_PAGE_FULL
: Internal error; never returned by nvs API functions

ESP_ERR_NVS_INVALID_STATE
: NVS is in an inconsistent state due to a previous error. Call nvs_flash_init and nvs_open again, then retry.

ESP_ERR_NVS_INVALID_LENGTH
: String or blob length is not sufficient to store data

ESP_ERR_NVS_NO_FREE_PAGES
: NVS partition doesn’ t contain any empty pages. This may happen if NVS partition was truncated. Erase the whole partition and call nvs_flash_init again.

ESP_ERR_NVS_VALUE_TOO_LONG
: String or blob length is longer than supported by the implementation

ESP_ERR_NVS_PART_NOT_FOUND
: Partition with specified name is not found in the partition table

ESP_ERR_NVS_NEW_VERSION_FOUND
: NVS partition contains data in new format and cannot be recognized by this version of code

ESP_ERR_NVS_XTS_ENCR_FAILED
: XTS encryption failed while writing NVS entry

ESP_ERR_NVS_XTS_DECR_FAILED
: XTS decryption failed while reading NVS entry

ESP_ERR_NVS_XTS_CFG_FAILED
: XTS configuration setting failed

ESP_ERR_NVS_XTS_CFG_NOT_FOUND
: XTS configuration not found

ESP_ERR_NVS_ENCR_NOT_SUPPORTED
: NVS encryption is not supported in this version

ESP_ERR_NVS_KEYS_NOT_INITIALIZED
: NVS key partition is uninitialized

ESP_ERR_NVS_CORRUPT_KEY_PART
: NVS key partition is corrupt

ESP_ERR_NVS_CONTENT_DIFFERS
: Internal error; never returned by nvs API functions. NVS key is different in comparison

NVS_DEFAULT_PART_NAME
: Default partition name of the NVS partition in the partition table

NVS_PART_NAME_MAX_SIZE
: maximum length of partition name (excluding null terminator)

Type Definitions

typedef uint32_t **nvs_handle_t**
: Opaque pointer type representing non-volatile storage handle

typedef *nvs_handle_t* **nvs_handle**

typedef *nvs_open_mode_t* **nvs_open_mode**

typedef struct nvs_opaque_iterator_t ***nvs_iterator_t**

Opaque pointer type representing iterator to nvs entries

Enumerations

enum nvs_open_mode_t

Mode of opening the non-volatile storage.

Values:

NVS_READONLY

Read only

NVS_READWRITE

Read and write

enum nvs_type_t

Types of variables.

Values:

NVS_TYPE_U8 = 0x01

Type uint8_t

NVS_TYPE_I8 = 0x11

Type int8_t

NVS_TYPE_U16 = 0x02

Type uint16_t

NVS_TYPE_I16 = 0x12

Type int16_t

NVS_TYPE_U32 = 0x04

Type uint32_t

NVS_TYPE_I32 = 0x14

Type int32_t

NVS_TYPE_U64 = 0x08

Type uint64_t

NVS_TYPE_I64 = 0x18

Type int64_t

NVS_TYPE_STR = 0x21

Type string

`NVS_TYPE_BLOB` = 0x42
Type blob

`NVS_TYPE_ANY` = 0xff
Must be last

2.6.4 NVS 分区生成程序

[English]

介绍

NVS 分区生成程序 (nvs_flash/nvs_partition_generator/nvs_partition_gen.py) 根据 CSV 文件中的键值对生成二进制文件。该二进制文件与*非易失性存储器 (NVS)* 中定义的 NVS 结构兼容。NVS 分区生成程序适合用于生成二进制数据（Blob），其中包括设备生产时可从外部烧录的 ODM/OEM 数据。这也使得生产制造商在使用同一个固件的基础上，通过自定义参数，如序列号等，为每个设备生成不同配置。

准备工作

在加密模式下使用该程序，需安装下列软件包：

- cryptography package

根目录下的 *requirements.txt* 包含必需 python 包，请预先安装。

CSV 文件格式

.csv 文件每行需包含四个参数，以逗号隔开。具体参数描述见下表：

序号	参数	描述	说明
1	Key	主键，应用程序可通过查询此键来获取数据。	
2	Type	支持 `file`、`data` 和 `namespace`。	
3	Encoding	支持 `u8`、`i8`、`u16`、`u32`、`i32`、`string`、`hex2bin`、`base64` 和 `binary`。决定二进制 `bin` 文件中 value 被编码成的类型。`string` 和 `binary` 编码的区别在于，`string` 数据以 NULL 字符结尾，`binary` 数据则不是。	`file` 类型当前仅支持 `hex2bin`、`base64`、`string` 和 `binary` 编码。
4	Value	Data value	`namespace` 字段的 `encoding` 和 `value` 应为空。`namespace` 的 `encoding` 和 `value` 为固定值，不可设置。这些单元格中的所有值都会被忽视。

注解： CSV 文件的第一行应为列标题，不可设置。

此类 CSV 文件的 Dump 示例如下:

```
key,type,encoding,value      <-- 列标题
namespace_name,namespace,,   <-- 第一个条目为 "namespace"
key1,data,u8,1
key2,file,string,/path/to/file
```

注解:

请确保:

- 逗号‘,’前后无空格;
- CSV 文件每行末尾无空格。

NVS 条目和命名空间 (namespace)

如 CSV 文件中出现命名空间条目，后续条目均会被视为该命名空间的一部分，直至找到下一个命名空间条目。找到新命名空间条目后，后续所有条目都会被视为新命名空间的一部分。

注解： CSV 文件中第一个条目应始终为 `namespace`。

支持多页 Blob

默认情况下，二进制 Blob 可跨多页，格式参考条目结构 章节。如需使用旧版格式，可在程序中禁用该功能。

支持加密

NVS 分区生成程序还可使用 AES-XTS 加密生成二进制加密文件。更多信息详见*NVS 加密*。

支持解密

如果 NVS 二进制文件采用了 AES-XTS 加密，该程序还可对此类文件进行解密，更多信息详见*NVS 加密*。

运行程序

使用方法:

```
python nvs_partition_gen.py [-h] {generate,generate-key,encrypt,decrypt} ...
```

可选参数:

序号	参数	描述
1	-h, –help	显示帮助信息并退出

命令:

```
运行 nvs_partition_gen.py {command} -h 查看更多帮助信息
```

序号	参数	描述
1	generate	生成 NVS 分区
2	generate-key	生成加密密钥
3	encrypt	加密 NVS 分区
4	decrypt	解密 NVS 分区

生成 NVS 分区（默认模式）

使用方法:

```
python nvs_partition_gen.py generate [-h] [--version {1,2}] [--outdir OUTDIR]
                                     input output size
```

位置参数:

参数	描述
input	待解析的 CSV 文件路径
output	NVS 二进制文件的输出路径
size	NVS 分区大小（以字节为单位，且为 4096 的整数倍）

可选参数:

参数	描述
-h, –help	显示帮助信息并退出
–version {1,2}	• 设置多页 Blob 版本。 • 版本 1：禁用多页 Blob； • 版本 2：启用多页 Blob； • 默认版本：版本 2。
–outdir OUTDIR	输出目录，用于存储创建的文件。(默认当前目录)

运行如下命令创建 NVS 分区，该程序同时会提供 CSV 示例文件:

```
python nvs_partition_gen.py generate sample_singlepage_blob.csv sample.bin 0x3000
```

仅生成加密密钥

使用方法:

```
python nvs_partition_gen.py generate-key [-h] [--keyfile KEYFILE]
                                         [--outdir OUTDIR]
```

可选参数:

参数	描述
-h, –help	显示帮助信息并退出
–keyfile KEYFILE	加密密钥文件的输出路径
–outdir OUTDIR	输出目录，用于存储创建的文件。(默认当前目录)

运行以下命令仅生成加密密钥:

```
python nvs_partition_gen.py generate-key
```

生成 NVS 加密分区

使用方法:

```
python nvs_partition_gen.py encrypt [-h] [--version {1,2}] [--keygen]
                                    [--keyfile KEYFILE] [--inputkey INPUTKEY]
                                    [--outdir OUTDIR]
                                    input output size
```

位置参数:

参数	描述
input	待解析 CSV 文件的路径
output	NVS 二进制文件的输出路径
size	NVS 分区大小（以字节为单位，且为 4096 的整数倍）

可选参数:

参数	描述
-h, –help	显示帮助信息并退出
–version {1,2}	• 设置多页 Blob 版本。 • 版本 1：禁用多页 Blob； • 版本 2：启用多页 Blob； • 默认版本：版本 2。
–keygen	生成 NVS 分区加密密钥
–keyfile KEYFILE	密钥文件的输出路径
–inputkey INPUTKEY	内含 NVS 分区加密密钥的文件
–outdir OUTDIR	输出目录，用于存储创建的文件（默认当前目录）

运行以下命令加密 NVS 分区，该程序同时会提供一个 CSV 示例文件。

- 通过 NVS 分区生成程序生成加密密钥来加密:

```
python nvs_partition_gen.py encrypt sample_singlepage_blob.csv sample_encr.bin␣
↪0x3000 --keygen
```

注解: 创建的加密密钥格式为 `<outdir>/keys/keys-<timestamp>.bin`。

- 通过 NVS 分区生成程序生成加密密钥，并将密钥存储于自定义的文件中:

```
python nvs_partition_gen.py encrypt sample_singlepage_blob.csv sample_encr.bin␣
↪0x3000 --keygen --keyfile sample_keys.bin
```

注解: 创建的加密密钥格式为 `<outdir>/keys/keys-<timestamp>.bin`。

注解: 加密密钥存储于新建文件的 `keys/` 目录下，与 NVS 密钥分区结构兼容。更多信息请参考*NVS 密钥*

分区。

- 将加密密钥用作二进制输入文件来进行加密:

```
python nvs_partition_gen.py encrypt sample_singlepage_blob.csv sample_encr.bin␣
↪0x3000 --inputkey sample_keys.bin
```

解密 NVS 分区

使用方法:

```
python nvs_partition_gen.py decrypt [-h] [--outdir OUTDIR] input key output
```

位置参数:

参数	描述
input	待解析的 NVS 加密分区文件路径
key	含有解密密钥的文件路径
output	已解密的二进制文件输出路径

可选参数:

参数	描述
-h, –help	显示帮助信息并退出
–outdir OUTDIR	输出目录，用于存储创建的文件（默认当前目录）

运行以下命令解密已加密的 NVS 分区:

```
python nvs_partition_gen.py decrypt sample_encr.bin sample_keys.bin sample_decr.bin
```

您可以自定义格式版本号:

- 版本 1：禁用多页 Blob
- 版本 2：启用多页 Blob

版本 1：禁用多页 Blob

如需禁用多页 Blob，请按照如下命令将版本参数设置为 1，以此格式运行分区生成程序。该程序同时会提供一个 CSV 示例文件:

```
python nvs_partition_gen.py generate sample_singlepage_blob.csv sample.bin 0x3000 --
↪version 1
```

版本 2：启用多页 Blob

如需启用多页 Blob，请按照如下命令将版本参数设置为 2，以此格式运行分区生成程序。该程序同时会提供一个 CSV 示例文件:

```
python nvs_partition_gen.py generate sample_multipage_blob.csv sample.bin 0x4000 --
↪version 2
```

注解： NVS 分区最小为 0x3000 字节。

注解： 将二进制文件烧录至设备时，请确保与应用的 sdkconfig 设置一致。

说明

- 分区生成程序不会对重复键进行检查，而将数据同时写入这两个重复键中。请注意不要使用同名的键；
- 新页面创建后，前一页的空白处不会再写入数据。CSV 文件中的字段须按次序排列以优化内存；
- 暂不支持 64 位数据类型。

2.6.5 虚拟文件系统组件

[English]

概述

虚拟文件系统 (VFS) 组件可为一些驱动提供一个统一接口。有了该接口，用户可像操作普通文件一样操作虚拟文件。这类驱动程序可以是 FAT、SPIFFS 等真实文件系统，也可以是有文件类接口的设备驱动程序。

VFS 组件支持 C 库函数（如 fopen 和 fprintf 等）与文件系统 (FS) 驱动程序协同工作。在高层级，每个 FS 驱动程序均与某些路径前缀相关联。当一个 C 库函数需要打开文件时，VFS 组件将搜索与该文件所在文件路径相关联的 FS 驱动程序，并将调用传递给该驱动程序。针对该文件的读取、写入等其他操作的调用也将传递给这个驱动程序。

例如，您可以使用 `/fat` 前缀注册 FAT 文件系统驱动，之后即可调用 `fopen("/fat/file.txt", "w")`。之后，VFS 将调用 FAT 驱动的 `open` 函数，并将参数 `/file.txt` 和合适的打开模式传递给 `open` 函数；后续对返回的 `FILE*` 数据流调用 C 库函数也同样会传递给 FAT 驱动。

注册 FS 驱动程序

如需注册 FS 驱动程序，首先要定义一个*esp_vfs_t* 结构体实例，并用指向 FS API 的函数指针填充它。

```
esp_vfs_t myfs = {
    .flags = ESP_VFS_FLAG_DEFAULT,
    .write = &myfs_write,
    .open = &myfs_open,
    .fstat = &myfs_fstat,
    .close = &myfs_close,
    .read = &myfs_read,
};

ESP_ERROR_CHECK(esp_vfs_register("/data", &myfs, NULL));
```

在上述代码中需要用到 read、write 或 read_p、write_p，具体使用哪组函数由 FS 驱动程序 API 的声明方式决定。

示例 1：声明 API 函数时不带额外的上下文指针参数，即 FS 驱动程序为单例模式，此时使用 write

```
ssize_t myfs_write(int fd, const void * data, size_t size);

// In definition of esp_vfs_t:
    .flags = ESP_VFS_FLAG_DEFAULT,
    .write = &myfs_write,
// ... other members initialized

// When registering FS, context pointer (third argument) is NULL:
ESP_ERROR_CHECK(esp_vfs_register("/data", &myfs, NULL));
```

示例 2：声明 API 函数时需要一个额外的上下文指针作为参数，即可支持多个 FS 驱动程序实例，此时使用 write_p

```
ssize_t myfs_write(myfs_t* fs, int fd, const void * data, size_t size);

// In definition of esp_vfs_t:
    .flags = ESP_VFS_FLAG_CONTEXT_PTR,
    .write_p = &myfs_write,
// ... other members initialized

// When registering FS, pass the FS context pointer into the third argument
// (hypothetical myfs_mount function is used for illustrative purposes)
myfs_t* myfs_inst1 = myfs_mount(partition1->offset, partition1->size);
```

(下页继续)

(续上页)

```
ESP_ERROR_CHECK(esp_vfs_register("/data1", &myfs, myfs_inst1));

// Can register another instance:
myfs_t* myfs_inst2 = myfs_mount(partition2->offset, partition2->size);
ESP_ERROR_CHECK(esp_vfs_register("/data2", &myfs, myfs_inst2));
```

同步输入/输出多路复用

如需通过 `select()` 使用同步输入/输出多路复用，首先需要把 `start_select()` 和 `end_select()` 注册到 VFS，如下所示：

```
// In definition of esp_vfs_t:
    .start_select = &uart_start_select,
    .end_select = &uart_end_select,
// ... other members initialized
```

调用 `start_select()` 设置环境，用以检测某一 VFS 文件描述符的读取/写入/错误条件。调用 `end_select()` 终止、析构或释放 `start_select()` 设置的资源。请在 vfs/vfs_uart.c 中查看 UART 外设参考实现、*`esp_vfs_dev_uart_register()`*、`uart_start_select()` 和 `uart_end_select()` 函数。

请参考以下示例，查看如何使用 VFS 文件描述符调用 `select()`：

- peripherals/uart_select
- system/select

如果 `select()` 用于套接字文件描述符，您可以启用 `CONFIG_LWIP_USE_ONLY_LWIP_SELECT` 选项来减少代码量，提高性能。

路径

已注册的 FS 驱动程序均有一个路径前缀与之关联，此路径前缀即为分区的挂载点。

如果挂载点中嵌套了其他挂载点，则在打开文件时使用具有最长匹配路径前缀的挂载点。例如，假设以下文件系统已在 VFS 中注册：

- 在 /data 下注册 FS 驱动程序 1
- 在 /data/static 下注册 FS 驱动程序 2

那么：

- 打开 `/data/log.txt` 会调用驱动程序 FS 1；
- 打开 `/data/static/index.html` 需调用 FS 驱动程序 2；
- 即便 FS 驱动程序 2 中没有 `/index.html`，也不会在 FS 驱动程序 1 中查找 `/static/index.html`。

挂载点名称必须以路径分隔符 (/) 开头，且分隔符后至少包含一个字符。但在以下情况中，VFS 同样支持空的挂载点名称：1. 应用程序需要提供一个”最后方案“下使用的文件系统；2. 应用程序需要同时覆盖 VFS 功能。如果没有与路径匹配的前缀，就会使用到这种文件系统。

VFS 不会对路径中的点 (.) 进行特殊处理，也不会将 `..` 视为对父目录的引用。在上述示例中，使用 `/data/static/../log.txt` 路径不会调用 FS 驱动程序 1 打开 `/log.txt`。特定的 FS 驱动程序（如 FATFS）可能以不同的方式处理文件名中的点。

执行打开文件操作时，FS 驱动程序仅得到文件的相对路径（挂载点前缀已经被去除）：

1. 以 `/data` 为路径前缀注册 `myfs` 驱动；
2. 应用程序调用 `fopen("/data/config.json", ...);`
3. VFS 调用 `myfs_open("/config.json", ...);`
4. `myfs` 驱动打开 `/config.json` 文件。

VFS 对文件路径长度没有限制，但文件系统路径前缀受 `ESP_VFS_PATH_MAX` 限制，即路径前缀上限为 `ESP_VFS_PATH_MAX`。各个文件系统驱动则可能会对自己的文件名长度设置一些限制。

文件描述符

文件描述符是一组很小的正整数，从 `0` 到 `FD_SETSIZE - 1`，`FD_SETSIZE` 在 newlib `sys/types.h` 中定义。最大文件描述符由 `CONFIG_LWIP_MAX_SOCKETS` 定义，且为套接字保留。VFS 中包含一个名为 `s_fd_table` 的查找表，用于将全局文件描述符映射至 `s_vfs` 数组中注册的 VFS 驱动索引。

标准 IO 流 (stdin, stdout, stderr)

如果 menuconfig 中 `UART for console output` 选项没有设置为 `None`，则 `stdin`、`stdout` 和 `stderr` 将默认从 UART 读取或写入。UART0 或 UART1 可用作标准 IO。默认情况下，UART0 使用 115200 波特率，TX 管脚为 GPIO1，RX 管脚为 GPIO3。您可以在 menuconfig 中更改上述参数。

对 `stdout` 或 `stderr` 执行写入操作将会向 UART 发送 FIFO 发送字符，对 `stdin` 执行读取操作则会从 UART 接收 FIFO 中取出字符。

默认情况下，VFS 使用简单的函数对 UART 进行读写操作。在所有数据放进 UART FIFO 之前，写操作将处于 busy-wait 状态，读操处于非阻塞状态，仅返回 FIFO 中已有数据。由于读操作为非阻塞，高层级 C 库函数调用（如 `fscanf("%d\n", &var);`）可能获取不到所需结果。

如果应用程序使用 UART 驱动，则可以调用 `esp_vfs_dev_uart_use_driver` 函数来指导 VFS 使用驱动中断、读写阻塞功能等。您也可以调用 `esp_vfs_dev_uart_use_nonblocking` 来恢复非阻塞函数。

VFS 还为输入和输出提供换行符转换功能（可选）。多数应用程序在程序内部发送或接收以 LF（‘ n’‘）结尾的行，但不同的终端程序可能需要不同的换行符，比如 CR 或 CRLF。应用程序可以通过 menuconfig 或者调用 `esp_vfs_dev_uart_set_rx_line_endings` 和 `esp_vfs_dev_uart_set_tx_line_endings` 为输入输出配置换行符。

标准流和 FreeRTOS 任务

stdin、stdout 和 stderr 的 FILE 对象在所有 FreeRTOS 任务之间共享，指向这些对象的指针分别存储在每个任务的 struct _reent 中。

预处理器把如下代码：

```
fprintf(stderr, "42\n");
```

解释为：

```
fprintf(__getreent()->_stderr, "42\n");
```

其中 __getreent() 函数将为每个任务返回一个指向 struct _reent 的指针 (newlib/include/sys/reent.h#L370-L417)。每个任务的 TCB 均拥有一个 struct _reent 结构体，任务初始化后，struct _reent 结构体中的 _stdin、_stdout 和 _stderr 将会被赋予 _GLOBAL_REENT 中 _stdin、_stdout 和 _stderr 的值，_GLOBAL_REENT 即为 FreeRTOS 启动之前所用结构体。

这样设计带来的结果是：

- 允许重定向给定任务的 stdin、stdout 和 stderr，而不影响其他任务，例如通过 stdin = fopen("/dev/uart/1", "r");
- 但使用 fclose 关闭默认 stdin、stdout 或 stderr 将同时关闭相应的 FILE 流对象，因此会影响其他任务；
- 如需更改新任务的默认 stdin、stdout 和 stderr 流，请在创建新任务之前修改 _GLOBAL_REENT->_stdin (_stdout、_stderr)。

应用示例

指南 （未完成）

API 参考

Header File

- vfs/include/esp_vfs.h

Functions

ssize_t **esp_vfs_write**(**struct** _reent *r, int *fd*, **const** void *data, size_t *size*)

These functions are to be used in newlib syscall table. They will be called by newlib when it needs to use any of the syscalls.

off_t **esp_vfs_lseek**(**struct** _reent *r, int *fd*, off_t *size*, int *mode*)

ssize_t **esp_vfs_read**(**struct** _reent *r, int *fd*, void **dst*, size_t *size*)

int **esp_vfs_open**(**struct** _reent *r, **const** char **path*, int *flags*, int *mode*)

int **esp_vfs_close**(**struct** _reent *r, int *fd*)

int **esp_vfs_fstat**(**struct** _reent *r, int *fd*, **struct** stat **st*)

int **esp_vfs_stat**(**struct** _reent *r, **const** char **path*, **struct** stat **st*)

int **esp_vfs_link**(**struct** _reent *r, **const** char **n1*, **const** char **n2*)

int **esp_vfs_unlink**(**struct** _reent *r, **const** char **path*)

int **esp_vfs_rename**(**struct** _reent *r, **const** char **src*, **const** char **dst*)

int **esp_vfs_utime**(**const** char **path*, **const struct** utimbuf **times*)

esp_err_t **esp_vfs_register**(**const** char **base_path*, **const** *esp_vfs_t* **vfs*, void **ctx*)

Register a virtual filesystem for given path prefix.

Return ESP_OK if successful, ESP_ERR_NO_MEM if too many VFSes are registered.

Parameters

- `base_path`: file path prefix associated with the filesystem. Must be a zero-terminated C string, up to ESP_VFS_PATH_MAX characters long, and at least 2 characters long. Name must start with a "/" and must not end with "/" . For example, "/data" or "/dev/spi" are valid. These VFSes would then be called to handle file paths such as "/data/myfile.txt" or "/dev/spi/0" .
- `vfs`: Pointer to *esp_vfs_t*, a structure which maps syscalls to the filesystem driver functions. VFS component doesn' t assume ownership of this pointer.
- `ctx`: If vfs->flags has ESP_VFS_FLAG_CONTEXT_PTR set, a pointer which should be passed to VFS functions. Otherwise, NULL.

esp_err_t **esp_vfs_register_fd_range**(**const** *esp_vfs_t* **vfs*, void **ctx*, int *min_fd*, int *max_fd*)

Special case function for registering a VFS that uses a method other than open() to open new file descriptors from the interval <min_fd; max_fd).

This is a special-purpose function intended for registering LWIP sockets to VFS.

Return ESP_OK if successful, ESP_ERR_NO_MEM if too many VFSes are registered, ESP_ERR_INVALID_ARG if the file descriptor boundaries are incorrect.

Parameters

- `vfs`: Pointer to *esp_vfs_t*. Meaning is the same as for esp_vfs_register().
- `ctx`: Pointer to context structure. Meaning is the same as for esp_vfs_register().
- `min_fd`: The smallest file descriptor this VFS will use.

- `max_fd`: Upper boundary for file descriptors this VFS will use (the biggest file descriptor plus one).

esp_err_t **esp_vfs_register_with_id**(**const** *esp_vfs_t* **vfs*, void **ctx*, *esp_vfs_id_t* **vfs_id*)

Special case function for registering a VFS that uses a method other than open() to open new file descriptors. In comparison with esp_vfs_register_fd_range, this function doesn't pre-registers an interval of file descriptors. File descriptors can be registered later, by using esp_vfs_register_fd.

Return ESP_OK if successful, ESP_ERR_NO_MEM if too many VFSes are registered, ESP_ERR_INVALID_ARG if the file descriptor boundaries are incorrect.

Parameters

- `vfs`: Pointer to *esp_vfs_t*. Meaning is the same as for esp_vfs_register().
- `ctx`: Pointer to context structure. Meaning is the same as for esp_vfs_register().
- `vfs_id`: Here will be written the VFS ID which can be passed to esp_vfs_register_fd for registering file descriptors.

esp_err_t **esp_vfs_unregister**(**const** char **base_path*)

Unregister a virtual filesystem for given path prefix

Return ESP_OK if successful, ESP_ERR_INVALID_STATE if VFS for given prefix hasn't been registered

Parameters

- `base_path`: file prefix previously used in esp_vfs_register call

esp_err_t **esp_vfs_register_fd**(*esp_vfs_id_t* *vfs_id*, int **fd*)

Special function for registering another file descriptor for a VFS registered by esp_vfs_register_with_id.

Return ESP_OK if the registration is successful, ESP_ERR_NO_MEM if too many file descriptors are registered, ESP_ERR_INVALID_ARG if the arguments are incorrect.

Parameters

- `vfs_id`: VFS identificator returned by esp_vfs_register_with_id.
- `fd`: The registered file descriptor will be written to this address.

esp_err_t **esp_vfs_unregister_fd**(*esp_vfs_id_t* *vfs_id*, int *fd*)

Special function for unregistering a file descriptor belonging to a VFS registered by esp_vfs_register_with_id.

Return ESP_OK if the registration is successful, ESP_ERR_INVALID_ARG if the arguments are incorrect.

Parameters

- `vfs_id`: VFS identificator returned by esp_vfs_register_with_id.
- `fd`: File descriptor which should be unregistered.

int **esp_vfs_select**(int *nfds*, fd_set **readfds*, fd_set **writefds*, fd_set **errorfds*, **struct** timeval **timeout*)

Synchronous I/O multiplexing which implements the functionality of POSIX select() for VFS.

Return The number of descriptors set in the descriptor sets, or -1 when an error (specified by errno) have occurred.

Parameters

- `nfds`: Specifies the range of descriptors which should be checked. The first nfds descriptors will be checked in each set.
- `readfds`: If not NULL, then points to a descriptor set that on input specifies which descriptors should be checked for being ready to read, and on output indicates which descriptors are ready to read.
- `writefds`: If not NULL, then points to a descriptor set that on input specifies which descriptors should be checked for being ready to write, and on output indicates which descriptors are ready to write.
- `errorfds`: If not NULL, then points to a descriptor set that on input specifies which descriptors should be checked for error conditions, and on output indicates which descriptors have error conditions.
- `timeout`: If not NULL, then points to timeval structure which specifies the time period after which the functions should time-out and return. If it is NULL, then the function will not time-out.

void **esp_vfs_select_triggered**(*esp_vfs_select_sem_t sem*)

Notification from a VFS driver about a read/write/error condition.

This function is called when the VFS driver detects a read/write/error condition as it was requested by the previous call to start_select.

Parameters

- `sem`: semaphore structure which was passed to the driver by the start_select call

void **esp_vfs_select_triggered_isr**(*esp_vfs_select_sem_t sem*, BaseType_t **woken*)

Notification from a VFS driver about a read/write/error condition (ISR version)

This function is called when the VFS driver detects a read/write/error condition as it was requested by the previous call to start_select.

Parameters

- `sem`: semaphore structure which was passed to the driver by the start_select call
- `woken`: is set to pdTRUE if the function wakes up a task with higher priority

int **esp_vfs_poll**(**struct** pollfd *_fds_, nfds_t _nfds_, int _timeout_)

Implements the VFS layer for synchronous I/O multiplexing by poll()

The implementation is based on esp_vfs_select. The parameters and return values are compatible with POSIX poll().

Return A positive return value indicates the number of file descriptors that have been selected. The 0 return value indicates a timed-out poll. -1 is return on failure and errno is set accordingly.

Parameters

- `fds`: Pointer to the array containing file descriptors and events poll() should consider.
- `nfds`: Number of items in the array fds.
- `timeout`: Poll() should wait at least timeout milliseconds. If the value is 0 then it should return immediately. If the value is -1 then it should wait (block) until the event occurs.

ssize_t **esp_vfs_pread**(int _fd_, void *_dst_, size_t _size_, off_t _offset_)

Implements the VFS layer of POSIX pread()

Return A positive return value indicates the number of bytes read. -1 is return on failure and errno is set accordingly.

Parameters

- `fd`: File descriptor used for read
- `dst`: Pointer to the buffer where the output will be written
- `size`: Number of bytes to be read
- `offset`: Starting offset of the read

ssize_t **esp_vfs_pwrite**(int _fd_, **const** void *_src_, size_t _size_, off_t _offset_)

Implements the VFS layer of POSIX pwrite()

Return A positive return value indicates the number of bytes written. -1 is return on failure and errno is set accordingly.

Parameters

- `fd`: File descriptor used for write
- `src`: Pointer to the buffer from where the output will be read
- `size`: Number of bytes to write

- `offset`: Starting offset of the write

Structures

struct esp_vfs_select_sem_t

VFS semaphore type for select()

Public Members

bool **is_sem_local**

type of "sem" is SemaphoreHandle_t when true, defined by socket driver otherwise

void ***sem**

semaphore instance

struct esp_vfs_t

VFS definition structure.

This structure should be filled with pointers to corresponding FS driver functions.

VFS component will translate all FDs so that the filesystem implementation sees them starting at zero. The caller sees a global FD which is prefixed with an pre-filesystem-implementation.

Some FS implementations expect some state (e.g. pointer to some structure) to be passed in as a first argument. For these implementations, populate the members of this structure which have _p suffix, set flags member to ESP_VFS_FLAG_CONTEXT_PTR and provide the context pointer to esp_vfs_register function. If the implementation doesn't use this extra argument, populate the members without _p suffix and set flags member to ESP_VFS_FLAG_DEFAULT.

If the FS driver doesn't provide some of the functions, set corresponding members to NULL.

Public Members

int **flags**

ESP_VFS_FLAG_CONTEXT_PTR or ESP_VFS_FLAG_DEFAULT

esp_err_t (***start_select**)(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, *esp_vfs_select_sem_t* sem, void **end_select_args)

start_select is called for setting up synchronous I/O multiplexing of the desired file descriptors in the given VFS

int (***socket_select**)(int nfds, fd_set *readfds, fd_set *writefds, fd_set *errorfds, **struct** timeval *timeout)

socket select function for socket FDs with the functionality of POSIX select(); this should be set only for the socket VFS

void (*stop_socket_select)(void *sem)
: called by VFS to interrupt the socket_select call when select is activated from a non-socket VFS driver; set only for the socket driver

void (*stop_socket_select_isr)(void *sem, BaseType_t *woken)
: stop_socket_select which can be called from ISR; set only for the socket driver

void *(*get_socket_select_semaphore)(void)
: end_select is called to stop the I/O multiplexing and deinitialize the environment created by start_select for the given VFS

esp_err_t (*end_select)(void *end_select_args)
: get_socket_select_semaphore returns semaphore allocated in the socket driver; set only for the socket driver

Macros

MAX_FDS
: Maximum number of (global) file descriptors.

ESP_VFS_PATH_MAX
: Maximum length of path prefix (not including zero terminator)

ESP_VFS_FLAG_DEFAULT
: Default value of flags member in *esp_vfs_t* structure.

ESP_VFS_FLAG_CONTEXT_PTR
: Flag which indicates that FS needs extra context pointer in syscalls.

Type Definitions

typedef int esp_vfs_id_t

Header File

- vfs/include/esp_vfs_dev.h

Functions

void esp_vfs_dev_uart_register(void)
: add /dev/uart virtual filesystem driver

  This function is called from startup code to enable serial output

void **esp_vfs_dev_uart_set_rx_line_endings**(*esp_line_endings_t mode*)

Set the line endings expected to be received on UART.

This specifies the conversion between line endings received on UART and newlines (‘ , LF) passed into stdin:

- ESP_LINE_ENDINGS_CRLF: convert CRLF to LF
- ESP_LINE_ENDINGS_CR: convert CR to LF
- ESP_LINE_ENDINGS_LF: no modification

Note this function is not thread safe w.r.t. reading from UART

Parameters

- `mode`: line endings expected on UART

void **esp_vfs_dev_uart_set_tx_line_endings**(*esp_line_endings_t mode*)

Set the line endings to sent to UART.

This specifies the conversion between newlines (‘ , LF) on stdout and line endings sent over UART:

- ESP_LINE_ENDINGS_CRLF: convert LF to CRLF
- ESP_LINE_ENDINGS_CR: convert LF to CR
- ESP_LINE_ENDINGS_LF: no modification

Note this function is not thread safe w.r.t. writing to UART

Parameters

- `mode`: line endings to send to UART

void **esp_vfs_dev_uart_use_nonblocking**(int *uart_num*)

set VFS to use simple functions for reading and writing UART Read is non-blocking, write is busy waiting until TX FIFO has enough space. These functions are used by default.

Parameters

- `uart_num`: UART peripheral number

void **esp_vfs_dev_uart_use_driver**(int *uart_num*)

set VFS to use UART driver for reading and writing

Note application must configure UART driver before calling these functions With these functions, read and write are blocking and interrupt-driven.

Parameters

- `uart_num`: UART peripheral number

Enumerations

enum esp_line_endings_t

Line ending settings.

Values:

ESP_LINE_ENDINGS_CRLF

CR + LF.

ESP_LINE_ENDINGS_CR

CR.

ESP_LINE_ENDINGS_LF

LF.

2.6.6 FAT 文件系统

[English]

ESP-IDF 使用 FatFs 库来实现 FAT 文件系统。FatFs 库位于 `fatfs` 组件中，您可以直接使用，也可以借助 C 标准库和 POSIX API 通过 VFS（虚拟文件系统）使用 FatFs 库的大多数功能。

此外，我们对 FatFs 库进行了扩展，新增了支持可插拔磁盘 I/O 调度层，从而允许在运行时将 FatFs 驱动映射到物理磁盘。

FatFs 与 VFS 配合使用

fatfs/vfs/esp_vfs_fat.h 头文件定义了连接 FatFs 和 VFS 的函数。

函数*esp_vfs_fat_register()* 分配一个 `FATFS` 结构，并在 VFS 中注册特定路径前缀。如果文件路径以此前缀开头，则对此文件的后续操作将转至 FatFs API。函数*esp_vfs_fat_unregister_path()* 删除在 VFS 中的注册，并释放 `FATFS` 结构。

多数应用程序在使用 `esp_vfs_fat_` 函数时，采用如下步骤：

1. 调用*esp_vfs_fat_register()*，指定：
   - 挂载文件系统的路径前缀（例如，`"/sdcard"` 或 `"/spiflash"`）
   - FatFs 驱动编号
   - 一个用于接收指向 `FATFS` 结构指针的变量
2. 调用*ff_diskio_register()* 为上述步骤中的驱动编号注册磁盘 I/O 驱动；
3. 调用 FatFs 函数 `f_mount`，或 `f_fdisk`，`f_mkfs`，并使用与传递到*esp_vfs_fat_register()* 相同的驱动编号挂载文件系统。请参考 FatFs 文档，查看更多信息；

4. 调用 C 标准库和 POSIX API 对路径中带有步骤 1 中所述前缀的文件（例如，"/sdcard/hello.txt"）执行打开、读取、写入、擦除、复制等操作。
5. 您可以选择直接调用 FatFs 库函数，但需要使用没有 VFS 前缀的路径（例如，"/hello.txt"）；
6. 关闭所有打开的文件；
7. 调用 f_mount 并使用 NULL FATFS* 参数为与上述编号相同的驱动卸载文件系统；
8. 调用 FatFs 函数*ff_diskio_register()* 并使用 NULL ff_diskio_impl_t* 参数和相同的驱动编号来释放注册的磁盘 I/O 驱动。
9. 调用*esp_vfs_fat_unregister_path()* 并使用文件系统挂载的路径将 FatFs 从 NVS 中移除，并释放步骤 1 中分配的 FatFs 结构。

esp_vfs_fat_sdmmc_mount 和 esp_vfs_fat_sdmmc_unmount 这两个便捷函数对上述步骤进行了封装，并加入对 SD 卡初始化的处理，非常便捷。我们将在下一章节详细介绍这两个函数。

esp_err_t **esp_vfs_fat_register**(**const** char **base_path*, **const** char **fat_drive*, size_t *max_files*, FATFS ***out_fs*)

Register FATFS with VFS component.

This function registers given FAT drive in VFS, at the specified base path. If only one drive is used, fat_drive argument can be an empty string. Refer to FATFS library documentation on how to specify FAT drive. This function also allocates FATFS structure which should be used for f_mount call.

Note This function doesn't mount the drive into FATFS, it just connects POSIX and C standard library IO function with FATFS. You need to mount desired drive into FATFS separately.

Return

- ESP_OK on success
- ESP_ERR_INVALID_STATE if esp_vfs_fat_register was already called
- ESP_ERR_NO_MEM if not enough memory or too many VFSes already registered

Parameters

- **base_path**: path prefix where FATFS should be registered
- **fat_drive**: FATFS drive specification; if only one drive is used, can be an empty string
- **max_files**: maximum number of files which can be open at the same time
- **out_fs**: pointer to FATFS structure which can be used for FATFS f_mount call is returned via this argument.

esp_err_t **esp_vfs_fat_unregister_path**(**const** char **base_path*)

Un-register FATFS from VFS.

Note FATFS structure returned by esp_vfs_fat_register is destroyed after this call. Make sure to call f_mount function to unmount it before calling esp_vfs_fat_unregister_ctx. Difference between

this function and the one above is that this one will release the correct drive, while the one above will release the last registered one

Return

- ESP_OK on success
- ESP_ERR_INVALID_STATE if FATFS is not registered in VFS

Parameters

- `base_path`: path prefix where FATFS is registered. This is the same used when esp_vfs_fat_register was called

FatFs 与 VFS 和 SD 卡配合使用

fatfs/vfs/esp_vfs_fat.h 头文件定义了两个便捷函数*esp_vfs_fat_sdmmc_mount()*和*esp_vfs_fat_sdmmc_unmount()*。这两个函数分别执行上一章节的步骤 1-3 和步骤 7-9，并初始化 SD 卡，但仅提供有限的错误处理功能。我们鼓励开发人员查看源代码并将更多高级功能集成到产品应用中。

esp_vfs_fat_sdmmc_unmount() 函数用于卸载文件系统并释放从*esp_vfs_fat_sdmmc_mount()* 函数获取的资源。

esp_err_t **esp_vfs_fat_sdmmc_mount**(**const** char **base_path*, **const** *sdmmc_host_t* **host_config*, **const** void **slot_config*, **const** *esp_vfs_fat_mount_config_t* **mount_config*, *sdmmc_card_t* ***out_card*)

Convenience function to get FAT filesystem on SD card registered in VFS.

This is an all-in-one function which does the following:

- initializes SDMMC driver or SPI driver with configuration in host_config
- initializes SD card with configuration in slot_config
- mounts FAT partition on SD card using FATFS library, with configuration in mount_config
- registers FATFS library with VFS, with prefix given by base_prefix variable

This function is intended to make example code more compact. For real world applications, developers should implement the logic of probing SD card, locating and mounting partition, and registering FATFS in VFS, with proper error checking and handling of exceptional conditions.

Return

- ESP_OK on success
- ESP_ERR_INVALID_STATE if esp_vfs_fat_sdmmc_mount was already called
- ESP_ERR_NO_MEM if memory can not be allocated
- ESP_FAIL if partition can not be mounted
- other error codes from SDMMC or SPI drivers, SDMMC protocol, or FATFS drivers

Parameters

- `base_path`: path where partition should be registered (e.g. "/sdcard")
- `host_config`: Pointer to structure describing SDMMC host. When using SDMMC peripheral, this structure can be initialized using SDMMC_HOST_DEFAULT() macro. When using SPI peripheral, this structure can be initialized using SDSPI_HOST_DEFAULT() macro.
- `slot_config`: Pointer to structure with slot configuration. For SDMMC peripheral, pass a pointer to *sdmmc_slot_config_t* structure initialized using SDMMC_SLOT_CONFIG_DEFAULT. For SPI peripheral, pass a pointer to *sdspi_slot_config_t* structure initialized using SDSPI_SLOT_CONFIG_DEFAULT.
- `mount_config`: pointer to structure with extra parameters for mounting FATFS
- `out_card`: if not NULL, pointer to the card information structure will be returned via this argument

struct esp_vfs_fat_mount_config_t

Configuration arguments for esp_vfs_fat_sdmmc_mount and esp_vfs_fat_spiflash_mount functions.

Public Members

bool **format_if_mount_failed**

If FAT partition can not be mounted, and this parameter is true, create partition table and format the filesystem.

int **max_files**

Max number of open files.

size_t **allocation_unit_size**

If format_if_mount_failed is set, and mount fails, format the card with given allocation unit size. Must be a power of 2, between sector size and 128 * sector size. For SD cards, sector size is always 512 bytes. For wear_levelling, sector size is determined by CONFIG_WL_SECTOR_SIZE option.

Using larger allocation unit size will result in higher read/write performance and higher overhead when storing small files.

Setting this field to 0 will result in allocation unit set to the sector size.

esp_err_t **esp_vfs_fat_sdmmc_unmount**(void)

Unmount FAT filesystem and release resources acquired using esp_vfs_fat_sdmmc_mount.

Return

- ESP_OK on success

- ESP_ERR_INVALID_STATE if esp_vfs_fat_sdmmc_mount hasn't been called

FatFs 与 VFS 配合使用（只读模式下）

fatfs/vfs/esp_vfs_fat.h 头文件也定义了两个便捷函数*esp_vfs_fat_rawflash_mount()*和*esp_vfs_fat_rawflash_unmount()*。上述两个函数分别对 FAT 只读分区执行步骤 1-3 和步骤 7-9。有些数据分区仅在工厂时写入一次，之后在整个硬件生命周期内都不会再有任何改动。利用上述两个函数处理这种数据分区非常方便。

esp_err_t **esp_vfs_fat_rawflash_mount**(**const** char *_base_path_, **const** char *_partition_label_, **const** *esp_vfs_fat_mount_config_t* *_mount_config_)

Convenience function to initialize read-only FAT filesystem and register it in VFS.

This is an all-in-one function which does the following:

- finds the partition with defined partition_label. Partition label should be configured in the partition table.
- mounts FAT partition using FATFS library
- registers FATFS library with VFS, with prefix given by base_prefix variable

Note Wear levelling is not used when FAT is mounted in read-only mode using this function.

Return

- ESP_OK on success
- ESP_ERR_NOT_FOUND if the partition table does not contain FATFS partition with given label
- ESP_ERR_INVALID_STATE if esp_vfs_fat_rawflash_mount was already called for the same partition
- ESP_ERR_NO_MEM if memory can not be allocated
- ESP_FAIL if partition can not be mounted
- other error codes from SPI flash driver, or FATFS drivers

Parameters

- `base_path`: path where FATFS partition should be mounted (e.g. "/spiflash")
- `partition_label`: label of the partition which should be used
- `mount_config`: pointer to structure with extra parameters for mounting FATFS

esp_err_t **esp_vfs_fat_rawflash_unmount**(**const** char *_base_path_, **const** char *_partition_label_)

Unmount FAT filesystem and release resources acquired using esp_vfs_fat_rawflash_mount.

Return

- ESP_OK on success
- ESP_ERR_INVALID_STATE if esp_vfs_fat_spiflash_mount hasn't been called

Parameters

- `base_path`: path where partition should be registered (e.g. "/spiflash")
- `partition_label`: label of partition to be unmounted

FatFs 磁盘 I/O 层

我们对 FatFs API 函数进行了扩展，实现了运行期间注册磁盘 I/O 驱动。

上述 API 为 SD/MMC 卡提供了磁盘 I/O 函数实现方式，可使用*ff_diskio_register_sdmmc()* 注册指定的 FatFs 驱动编号。

void **ff_diskio_register**(BYTE *pdrv*, **const** *ff_diskio_impl_t* **discio_impl*)

Register or unregister diskio driver for given drive number.

When FATFS library calls one of disk_xxx functions for driver number pdrv, corresponding function in discio_impl for given pdrv will be called.

Parameters

- `pdrv`: drive number
- `discio_impl`: pointer to *ff_diskio_impl_t* structure with diskio functions or NULL to unregister and free previously registered drive

struct ff_diskio_impl_t

Structure of pointers to disk IO driver functions.

See FatFs documentation for details about these functions

Public Members

DSTATUS (***init**)(unsigned char pdrv)

disk initialization function

DSTATUS (***status**)(unsigned char pdrv)

disk status check function

DRESULT (***read**)(unsigned char pdrv, unsigned char *buff, uint32_t sector, unsigned count)

sector read function

DRESULT (***write**)(unsigned char pdrv, **const** unsigned char *buff, uint32_t sector, unsigned count)

sector write function

DRESULT (*ioctl)(unsigned char pdrv, unsigned char cmd, void *buff)
function to get info about disk and do some misc operations

void **ff_diskio_register_sdmmc**(unsigned char *pdrv*, *sdmmc_card_t* **card*)
Register SD/MMC diskio driver

Parameters

- **pdrv**: drive number
- **card**: pointer to *sdmmc_card_t* structure describing a card; card should be initialized before calling f_mount.

esp_err_t **ff_diskio_register_wl_partition**(unsigned char *pdrv*, *wl_handle_t* *flash_handle*)
Register spi flash partition

Parameters

- **pdrv**: drive number
- **flash_handle**: handle of the wear levelling partition.

esp_err_t **ff_diskio_register_raw_partition**(unsigned char *pdrv*, **const** *esp_partition_t* **part_handle*)
Register spi flash partition

Parameters

- **pdrv**: drive number
- **part_handle**: pointer to raw flash partition.

2.6.7 磨损均衡 API

[English]

概述

ESP32 所使用的 flash，特别是 SPI flash 多数具备扇区结构，且每个扇区仅允许有限次数的擦除/修改操作。为了避免过度使用某一扇区，乐鑫提供了磨损均衡组件，无需用户介入即可帮助用户均衡各个扇区之间的磨损。

磨损均衡组件包含了通过分区组件对外部 SPI flash 进行数据读取、写入、擦除和存储器映射相关的 API 函数。磨损均衡组件还具有软件上更高级别的 API 函数，与*FAT 文件系统* 协同工作。

磨损均衡组件与 FAT 文件系统组件共用 FAT 文件系统的扇区，扇区大小为 4096 字节，是标准 flash 扇区的大小。在这种模式下，磨损均衡组件性能达到最佳，但需要在 RAM 中占用更多内存。

为了节省内存，磨损均衡组件还提供了另外两种模式，均使用 512 字节大小的扇区：

- **性能模式**：先将数据保存在 RAM 中，擦除扇区，然后将数据存储回 flash。如果设备在扇区擦写过程中突然断电，则整个扇区（4096 字节）数据将全部丢失。
- **安全模式**：数据先保存在 flash 中空余扇区，擦除扇区后，数据即存储回去。如果设备断电，上电后可立即恢复数据。

设备默认设置如下：

- 定义扇区大小为 512 字节
- 默认使用性能模式

您可以使用配置菜单更改设置。

磨损均衡组件不会将数据缓存在 RAM 中。写入和擦除函数直接修改 flash，函数返回后，flash 即完成修改。

磨损均衡访问 API

处理 flash 数据常用的 API 如下所示：

- `wl_mount` - 为指定分区挂载并初始化磨损均衡模块
- `wl_unmount` - 卸载分区并释放磨损均衡模块
- `wl_erase_range` - 擦除 flash 中指定的地址范围
- `wl_write` - 将数据写入分区
- `wl_read` - 从分区读取数据
- `wl_size` - 返回可用内存的大小（以字节为单位）
- `wl_sector_size` - 返回一个扇区的大小

请尽量避免直接使用原始磨损均衡函数，建议您使用文件系统特定的函数。

内存大小

内存大小是根据分区参数在磨损均衡模块中计算所得，由于模块使用 flash 部分扇区存储内部数据，因此计算所得内存大小有少许偏差。

另请参阅

- *FAT 文件系统*
- 分区表

应用示例

storage/wear_levelling 中提供了一款磨损均衡驱动与 FatFs 库结合使用的示例。该示例初始化磨损均衡驱动，挂载 FAT 文件系统分区，并使用 POSIX（可移植操作系统接口）和 C 库 API 从中写入和读取数据。如需了解更多信息，请参考 storage/wear_levelling/README.md。

高级 API 参考

头文件

- fatfs/src/esp_vfs_fat.h

函数

esp_err_t **esp_vfs_fat_spiflash_mount**(**const** char ***base_path*, **const** char ***partition_label*, **const** *esp_vfs_fat_mount_config_t* **mount_config*, *wl_handle_t* **wl_handle*)

Convenience function to initialize FAT filesystem in SPI flash and register it in VFS.

This is an all-in-one function which does the following:

- finds the partition with defined partition_label. Partition label should be configured in the partition table.
- initializes flash wear levelling library on top of the given partition
- mounts FAT partition using FATFS library on top of flash wear levelling library
- registers FATFS library with VFS, with prefix given by base_prefix variable

This function is intended to make example code more compact.

Return

- ESP_OK on success
- ESP_ERR_NOT_FOUND if the partition table does not contain FATFS partition with given label
- ESP_ERR_INVALID_STATE if esp_vfs_fat_spiflash_mount was already called
- ESP_ERR_NO_MEM if memory can not be allocated
- ESP_FAIL if partition can not be mounted
- other error codes from wear levelling library, SPI flash driver, or FATFS drivers

Parameters

- `base_path`: path where FATFS partition should be mounted (e.g. "/spiflash")

- `partition_label`: label of the partition which should be used
- `mount_config`: pointer to structure with extra parameters for mounting FATFS
- `wl_handle`: wear levelling driver handle

struct esp_vfs_fat_mount_config_t

Configuration arguments for esp_vfs_fat_sdmmc_mount and esp_vfs_fat_spiflash_mount functions.

Public Members

bool **format_if_mount_failed**

If FAT partition can not be mounted, and this parameter is true, create partition table and format the filesystem.

int **max_files**

Max number of open files.

size_t **allocation_unit_size**

If format_if_mount_failed is set, and mount fails, format the card with given allocation unit size. Must be a power of 2, between sector size and 128 * sector size. For SD cards, sector size is always 512 bytes. For wear_levelling, sector size is determined by CONFIG_WL_SECTOR_SIZE option.

Using larger allocation unit size will result in higher read/write performance and higher overhead when storing small files.

Setting this field to 0 will result in allocation unit set to the sector size.

esp_err_t **esp_vfs_fat_spiflash_unmount(const** char **base_path*, *wl_handle_t wl_handle*)

Unmount FAT filesystem and release resources acquired using esp_vfs_fat_spiflash_mount.

Return

- ESP_OK on success
- ESP_ERR_INVALID_STATE if esp_vfs_fat_spiflash_mount hasn't been called

Parameters

- `base_path`: path where partition should be registered (e.g. "/spiflash")
- `wl_handle`: wear levelling driver handle returned by esp_vfs_fat_spiflash_mount

中层 API 参考

Header File

- wear_levelling/include/wear_levelling.h

Functions

esp_err_t **wl_mount**(**const** *esp_partition_t* **partition*, *wl_handle_t* **out_handle*)

Mount WL for defined partition.

Return

- ESP_OK, if the allocation was successfully;
- ESP_ERR_INVALID_ARG, if WL allocation was unsuccessful;
- ESP_ERR_NO_MEM, if there was no memory to allocate WL components;

Parameters

- `partition`: that will be used for access
- `out_handle`: handle of the WL instance

esp_err_t **wl_unmount**(*wl_handle_t* *handle*)

Unmount WL for defined partition.

Return

- ESP_OK, if the operation completed successfully;
- or one of error codes from lower-level flash driver.

Parameters

- `handle`: WL partition handle

esp_err_t **wl_erase_range**(*wl_handle_t* *handle*, size_t *start_addr*, size_t *size*)

Erase part of the WL storage.

Return

- ESP_OK, if the range was erased successfully;
- ESP_ERR_INVALID_ARG, if iterator or dst are NULL;
- ESP_ERR_INVALID_SIZE, if erase would go out of bounds of the partition;
- or one of error codes from lower-level flash driver.

Parameters

- `handle`: WL handle that are related to the partition

- `start_addr`: Address where erase operation should start. Must be aligned to the result of function wl_sector_size(···).
- `size`: Size of the range which should be erased, in bytes. Must be divisible by result of function wl_sector_size(···)..

esp_err_t **wl_write**(*wl_handle_t handle*, size_t *dest_addr*, **const** void **src*, size_t *size*)

Write data to the WL storage.

Before writing data to flash, corresponding region of flash needs to be erased. This can be done using wl_erase_range function.

Note Prior to writing to WL storage, make sure it has been erased with wl_erase_range call.

Return

- ESP_OK, if data was written successfully;
- ESP_ERR_INVALID_ARG, if dst_offset exceeds partition size;
- ESP_ERR_INVALID_SIZE, if write would go out of bounds of the partition;
- or one of error codes from lower-level flash driver.

Parameters

- `handle`: WL handle that are related to the partition
- `dest_addr`: Address where the data should be written, relative to the beginning of the partition.
- `src`: Pointer to the source buffer. Pointer must be non-NULL and buffer must be at least 'size' bytes long.
- `size`: Size of data to be written, in bytes.

esp_err_t **wl_read**(*wl_handle_t handle*, size_t *src_addr*, void **dest*, size_t *size*)

Read data from the WL storage.

Return

- ESP_OK, if data was read successfully;
- ESP_ERR_INVALID_ARG, if src_offset exceeds partition size;
- ESP_ERR_INVALID_SIZE, if read would go out of bounds of the partition;
- or one of error codes from lower-level flash driver.

Parameters

- `handle`: WL module instance that was initialized before

- `dest`: Pointer to the buffer where data should be stored. Pointer must be non-NULL and buffer must be at least 'size' bytes long.
- `src_addr`: Address of the data to be read, relative to the beginning of the partition.
- `size`: Size of data to be read, in bytes.

size_t **wl_size**(*wl_handle_t handle*)

Get size of the WL storage.

Return usable size, in bytes

Parameters

- `handle`: WL module handle that was initialized before

size_t **wl_sector_size**(*wl_handle_t handle*)

Get sector size of the WL instance.

Return sector size, in bytes

Parameters

- `handle`: WL module handle that was initialized before

Macros

`WL_INVALID_HANDLE`

Type Definitions

typedef int32_t **wl_handle_t**

wear levelling handle

2.6.8 SPIFFS 文件系统

[English]

概述

SPIFFS 是一个用于 SPI NOR flash 设备的嵌入式文件系统，支持磨损均衡、文件系统一致性检查等功能。

说明

- 目前，SPIFFS 尚不支持目录，但可以生成扁平结构。如果 SPIFFS 挂载在 `/spiffs` 下，在 `/spiffs/tmp/myfile.txt` 路径下创建一个文件则会在 SPIFFS 中生成一个名为 `/tmp/myfile.txt` 的文件，而不是在 `/spiffs/tmp` 下生成名为 `myfile.txt` 的文件；
- SPIFFS 并非实时栈，每次写操作耗时不等；
- 目前，SPIFFS 尚不支持检测或处理已损坏的块。

工具

spiffsgen.py

spiffsgen.py:

```
python spiffsgen.py <image_size> <base_dir> <output_file>
```

参数（必选）说明如下：

- **image_size**：分区大小，用于烧录生成的 SPIFFS 映像；
- **base_dir**：创建 SPIFFS 映像的目录；
- **output_file**：SPIFFS 映像输出文件。

其他参数（可选）也参与控制映像的生成，您可以运行以下帮助命令，查看这些参数的具体信息:

```
python spiffsgen.py --help
```

上述可选参数对应 SPIFFS 构建配置选项。若想顺利生成可用的映像，请确保使用的参数或配置与构建 SPIFFS 时所用的参数或配置相同。运行帮助命令将显示参数所对应的 SPIFFS 构建配置。如未指定参数，将使用帮助信息中的默认值。

映像生成后，您可以使用 `esptool.py` 或 `parttool.py` 烧录映像。

您可以在命令行或脚本中手动单独调用 `spiffsgen.py`，也可以直接从构建系统调用 `spiffs_create_partition_image` 来使用 `spiffsgen.py`。

在 Make 构建系统中运行:

```
SPIFFS_IMAGE_FLASH_IN_PROJECT := ...
SPIFFS_IMAGE_DEPENDS := ...
$(eval $(call spiffs_create_partition_image,<partition>,<base_dir>))
```

在 CMake 构建系统中运行:

```
spiffs_create_partition_image(<partition> <base_dir> [FLASH_IN_PROJECT] [DEPENDS dep dep␣
↪dep...])
```

在构建系统中使用 `spiffsgen.py` 更为方便，构建配置自动传递给 `spiffsgen.py` 工具，确保生成的映像可用于构建。比如，单独调用 `spiffsgen.py` 时需要用到 *image_size* 参数，但在构建系统中调用 `spiffs_create_partition_image` 时，仅需要 *partition* 参数，映像大小将直接从工程分区表中获取。

Make 构建系统和 CMake 构建系统结构有所不同，请注意以下几点：

- 在 Make 构建系统中使用 `spiffs_create_partition_image`，需从工程 Makefile 中调用；
- 在 CMake 构建系统中使用 `spiffs_create_partition_image`，需从组件 CMakeLists.txt 文件调用。

您也可以指定 `FLASH_IN_PROJECT`，然后使用 `idf.py flash` 或 `make flash` 将映像与应用程序二进制文件、分区表等一起自动烧录至设备，例如：

在 Make 构建系统中运行:

```
SPIFFS_IMAGE_FLASH_IN_PROJECT := 1
$(eval $(call spiffs_create_partition_image,<partition>,<base_dir>))
```

在 CMake 构建系统中运行:

```
spiffs_create_partition_image(my_spiffs_partition my_folder FLASH_IN_PROJECT)
```

不指定 FLASH_IN_PROJECT/SPIFFS_IMAGE_FLASH_IN_PROJECT 也可以生成映像，但须使用 `esptool.py`、`parttool.py` 或自定义构建系统目标手动烧录。

有时基本目录中的内容是在构建时生成的，您可以使用 DEPENDS/SPIFFS_IMAGE_DEPENDS 指定目标，因此可以在生成映像之前执行此目标。

在 Make 构建系统中运行:

```
dep:
    ...

SPIFFS_IMAGE_DEPENDS := dep
$(eval $(call spiffs_create_partition_image,<partition>,<base_dir>))
```

在 CMake 构建系统中运行:

```
add_custom_target(dep COMMAND ...)

spiffs_create_partition_image(my_spiffs_partition my_folder DEPENDS dep)
```

请参考 examples/storage/spiffsgen>，查看示例。

mkspiffs

您也可以使用 mkspiffs 工具创建 SPIFFS 分区映像。与 `spiffsgen.py` 相似，mkspiffs 也可以用于从指定文件夹中生成映像，然后使用 `esptool.py` 烧录映像。

该工具需要获取以下参数：

- **Block Size**：4096（SPI flash 标准）
- **Page Size**：256（SPI flash 标准）
- **Image Size**：分区大小（以字节为单位，可从分区表中获取）
- **Partition Offset**：分区起始地址（可从分区表内获取）

运行以下命令，将文件夹打包成 1 MB 大小的映像：

```
mkspiffs -c [src_folder] -b 4096 -p 256 -s 0x100000 spiffs.bin
```

运行以下命令，将映像烧录到 ESP32（偏移量：0x110000）：

```
python esptool.py --chip esp32 --port [port] --baud [baud] write_flash -z 0x110000␣
↪spiffs.bin
```

选择合适的 SPIFFS 工具

上面介绍的两款 SPIFFS 工具功能相似，需根据实际情况，选择合适的一款。

以下情况优先选用 `spiffsgen.py` 工具：

1. 仅需在构建时简单生成 SPIFFS 映像，请选择使用 `spiffsgen.py`，`spiffsgen.py` 可以直接在构建系统中使用函数或命令生成 SPIFFS 映像。
2. 主机没有可用的 C/C++ 编译器时，可以选择使用 `spiffsgen.py` 工具，`spiffsgen.py` 不需要编译。

以下情况优先选用 `mkspiffs` 工具：

1. 如果您除了需要生成映像外，还需要拆包 SPIFFS 映像，请选择使用 `mkspiffs` 工具。`spiffsgen.py` 目前尚不支持此功能。
2. 如果您当前环境中 Python 解释器不可用，但主机编译器可用，或者有预编译的 `mkspiffs` 二进制文件，此时请选择使用 `mkspiffs` 工具。但是，`mkspiffs` 没有集成到构建系统，用户必须自己完成以下工作：在构建期间编译 `mkspiffs` （如果未使用预编译的二进制文件），为输出文件创建构建规则或目标，将适当的参数传递给工具等。

另请参阅

- 分区表

应用示例

storage/spiffs 目录下提供了 SPIFFS 应用示例。该示例初始化并挂载了一个 SPIFFS 分区，然后使用 POSIX 和 C 库 API 写入和读取数据。请参考 `example` 目录下的 README.md 文件，查看详细信息。

高级 API 参考

Header File

- spiffs/include/esp_spiffs.h

Functions

esp_err_t **esp_vfs_spiffs_register**(**const** *esp_vfs_spiffs_conf_t* **conf*)

Register and mount SPIFFS to VFS with given path prefix.

Return

- ESP_OK if success
- ESP_ERR_NO_MEM if objects could not be allocated
- ESP_ERR_INVALID_STATE if already mounted or partition is encrypted
- ESP_ERR_NOT_FOUND if partition for SPIFFS was not found
- ESP_FAIL if mount or format fails

Parameters

- `conf`: Pointer to *esp_vfs_spiffs_conf_t* configuration structure

esp_err_t **esp_vfs_spiffs_unregister**(**const** char **partition_label*)

Unregister and unmount SPIFFS from VFS

Return

- ESP_OK if successful
- ESP_ERR_INVALID_STATE already unregistered

Parameters

- `partition_label`: Same label as passed to esp_vfs_spiffs_register.

bool **esp_spiffs_mounted**(**const** char **partition_label*)

Check if SPIFFS is mounted

Return

- true if mounted
- false if not mounted

Parameters

- `partition_label`: Optional, label of the partition to check. If not specified, first partition with subtype=spiffs is used.

esp_err_t **esp_spiffs_format(const** char **partition_label)**

Format the SPIFFS partition

Return

- ESP_OK if successful
- ESP_FAIL on error

Parameters

- `partition_label`: Same label as passed to esp_vfs_spiffs_register.

esp_err_t **esp_spiffs_info(const** char **partition_label*, size_t **total_bytes*, size_t **used_bytes*)

Get information for SPIFFS

Return

- ESP_OK if success
- ESP_ERR_INVALID_STATE if not mounted

Parameters

- `partition_label`: Same label as passed to esp_vfs_spiffs_register
- `total_bytes`: Size of the file system
- `used_bytes`: Current used bytes in the file system

Structures

struct esp_vfs_spiffs_conf_t

Configuration structure for esp_vfs_spiffs_register.

Public Members

const char ***base_path**

File path prefix associated with the filesystem.

const char *partition_label
: Optional, label of SPIFFS partition to use. If set to NULL, first partition with subtype=spiffs will be used.

size_t max_files
: Maximum files that could be open at the same time.

bool format_if_mount_failed
: If true, it will format the file system if it fails to mount.

2.6.9 量产程序

[English]

介绍

这一程序主要用于量产时为每一设备创建工厂 NVS（非易失性存储器）分区映像。NVS 分区映像由 CSV（逗号分隔值）文件生成，文件中包含了用户提供的配置项及配置值。

注意，该程序仅创建用于量产的二进制映像，您需要使用以下工具将映像烧录到设备上：

- esptool.py
- Flash 下载工具（仅适用于 Windows）
- 直接烧录程序

准备工作

该程序需要用到分区公用程序。

- **操作系统要求：**
  - Linux、MacOS 或 Windows（标准版）
- **安装依赖包：**
  - Python：https://www.python.org/downloads/。

注解：

使用该程序之前，请确保：

- Python 路径已添加到 PATH 环境变量中；
- 已经安装 *requirement.txt* 中的软件包，*requirement.txt* 在 esp-idf 根目录下。

具体流程

CSV 配置文件

CSV 配置文件中包含设备待烧录的配置信息，定义了待烧录的配置项。例如定义 `firmware_key` (`key`) 的 `type` 为 `data`，`encoding` 为 `hex2bin`。

配置文件中数据格式如下（*REPEAT* 标签可选）：

```
name1,namespace,    <-- 第一行为 "namespace" 条目
key1,type1,encoding1
key2,type2,encoding2,REPEAT
name2,namespace,
key3,type3,encoding3
key4,type4,encoding4
```

注解: 文件第一行应始终为 `namespace` 条目。

每行应包含三个参数：`key`、`type` 和 `encoding`，并以逗号分隔。如果有 `REPEAT` 标签，则主 CSV 文件中所有设备此键值均相同。

有关各个参数的详细说明，请参阅 NVS 分区生成程序的 README 文件。

CSV 配置文件示例如下:

```
app,namespace,
firmware_key,data,hex2bin
serial_no,data,string,REPEAT <-- "serial_no" 被标记为 "REPEAT"
device_no,data,i32
```

注解:

请确保:

- 逗号‘,’前后无空格；
- CSV 文件每行末尾无空格。

主 CSV 文件

主 CSV 文件中包含设备待烧录的详细信息，文件中每行均对应一个设备实体。主 CSV 文件中的 key 应首先在 CSV 配置文件中定义。

主 CSV 文件的数据格式如下:

```
key1,key2,key3,.....
value1,value2,value3,.... <-- 对应一个设备实体
value4,value5,value6,.... <-- 对应一个设备实体
value7,value8,value9,.... <-- 对应一个设备实体
```

注解: 文件中键 (key) 名应始终置于文件首行。从配置文件中获取的键，在此文件中的排列顺序应与其在配置文件中的排列顺序相同。主 CSV 文件同时可以包含其它列（键），这些列将被视为元数据，而不会编译进最终二进制文件。

每行应包含相应键的键值 (value)，并用逗号隔开。如果某键带有 REPEAT 标签，则仅需在第二行（即第一个条目）输入对应的值，后面其他行为空。

参数描述如下:

value Data value

value 是与键对应的键值。

主 CSV 文件示例如下:

```
id,firmware_key,serial_no,device_no
1,1a2b3c4d5e6faabb,A1,101 <-- 对应一个设备实体（在 CSV 配置文件中标记为 `REPEAT` 的键，除第一个条目外，其他均为空）
2,1a2b3c4d5e6fccdd,,102   <-- 对应一个设备实体
3,1a2b3c4d5e6feeff,,103   <-- 对应一个设备实体
```

注解: 如果出现 *REPEAT* 标签，则会在相同目录下生成一个新的主 CSV 文件用作主输入文件，并在每行为带有 *REPEAT* 标签的键插入键值。

量产程序还会创建中间 CSV 文件，NVS 分区程序将使用此 CSV 文件作为输入，然后生成二进制文件。

中间 CSV 文件的格式如下:

```
key,type,encoding,value
key,namespace, ,
key1,type1,encoding1,value1
key2,type2,encoding2,value2
```

此步骤将为每一设备生成一个中间 CSV 文件。

运行量产程序

使用方法:

```
python mfg_gen.py [-h] {generate,generate-key} ...
```

可选参数:

序号	参数	描述
1	-h, –help	显示帮助信息并退出

命令:

运行 mfg_gen.py {command} -h 查看更多帮助信息

序号	参数	描述
1	generate	生成 NVS 分区
2	generate-key	生成加密密钥

为每个设备生成工厂映像（默认）

使用方法:

```
python mfg_gen.py generate [-h] [--fileid FILEID] [--version {1,2}] [--keygen]
                               [--keyfile KEYFILE] [--inputkey INPUTKEY]
                               [--outdir OUTDIR]
                               conf values prefix size
```

位置参数:

参数	描述
conf	待解析的 CSV 配置文件路径
values	待解析的主 CSV 文件路径
prefix	每个输出文件名前缀的唯一名称
size	NVS 分区大小（以字节为单位，且为 4096 的整数倍）

可选参数:

参数	描述
-h, –help	显示帮助信息并退出
–fileid FILEID	每个文件名后缀的唯一文件标识符（主 CSV 文件中的任意键），默认为数值 1、2、3…
–version {1,2}	• 设置多页 Blob 版本。 • 版本 1 - 禁用多页 Blob； • 版本 2 - 启用多页 Blob； • 默认版本：版本 2
–keygen	生成 NVS 分区加密密钥
–inputkey INPUTKEY	内含 NVS 分区加密密钥的文件
–outdir OUTDIR	输出目录，用于存储创建的文件（默认当前目录）

请运行以下命令为每个设备生成工厂映像，量产程序同时提供了一个 CSV 示例文件:

```
python mfg_gen.py generate samples/sample_config.csmples/sample_values_singlepage_blob.
↪csv Sample 0x3000
```

主 CSV 文件应在 `file` 类型下设置一个相对路径，指向运行该程序的当前目录。

为每个设备生成工厂加密映像

运行以下命令为每一设备生成工厂加密映像，量产程序同时提供了一个 CSV 示例文件。

- 通过量产程序生成加密密钥来进行加密:

```
python mfg_gen.py generate samples/sample_config.csv samples/sample_values_
↪singlepage_blob.csv Sample 0x3000 --keygen
```

注解: 创建的加密密钥格式为 `<outdir>/keys/keys-<prefix>-<fileid>.bin`。

注解: 加密密钥存储于新建文件的 `keys/` 目录下，与 NVS 密钥分区结构兼容。更多信息请参考*NVS 密钥分区*。

- 提供加密密钥用作二进制输入文件来进行加密:

```
python mfg_gen.py generate samples/sample_config.csv samples/sample_values_
↪singlepage_blob.csv Sample 0x3000 --inputkey keys/sample_keys.bin
```

仅生成加密密钥

使用方法:

```
python mfg_gen.py generate-key [-h] [--keyfile KEYFILE] [--outdir OUTDIR]
```

可选参数：

参数	描述
-h, –help	显示帮助信息并退出
–keyfile KEYFILE	加密密钥文件的输出路径
–outdir OUTDIR	输出目录，用于存储创建的文件（默认当前目录）

运行以下命令仅生成加密密钥:

```
python mfg_gen.py generate-key
```

注解： 创建的加密密钥格式为 `<outdir>/keys/keys-<timestamp>.bin`。时间戳格式为：`%m-%d_%H-%M`。

注解： 如需自定义目标文件名，请使用 –keyfile 参数。

生成的加密密钥二进制文件还可以用于为每个设备的工厂映像加密。

`fileid` 参数的默认值为 1、2、3…，与主 CSV 文件中的行一一对应，内含设备配置值。

运行量产程序时，将在指定的 `outdir` 目录下创建以下文件夹：

- `bin/` 存储生成的二进制文件
- `csv/` 存储生成的中间 CSV 文件
- `keys/` 存储加密密钥（创建工厂加密映像时会用到）

此部分 API 代码示例详见 ESP-IDF 项下 storage 目录。

2.7 System API

2.7.1 App Image Format

An application image consists of the following structures:

1. The *esp_image_header_t* structure describes the mode of SPI flash and the count of memory segments.

2. The *esp_image_segment_header_t* structure describes each segment, its length, and its location in ESP32' s memory, followed by the data with a length of `data_len`. The data offset for each segment in the image is calculated in the following way:

- offset for 0 Segment = sizeof(*esp_image_header_t*) + sizeof(*esp_image_segment_header_t*).
- offset for 1 Segment = offset for 0 Segment + length of 0 Segment + sizeof(*esp_image_segment_header_t*).
- offset for 2 Segment = offset for 1 Segment + length of 1 Segment + sizeof(*esp_image_segment_header_t*).
- ...

The count of each segment is defined in the `segment_count` field that is stored in *esp_image_header_t*. The count cannot be more than `ESP_IMAGE_MAX_SEGMENTS`.

To get the list of your image segments, please run the following command:

```
esptool.py --chip esp32 image_info build/app.bin
```

```
esptool.py v2.3.1
Image version: 1
Entry point: 40080ea4
13 segments
Segment 1: len 0x13ce0 load 0x3f400020 file_offs 0x00000018 SOC_DROM
Segment 2: len 0x00000 load 0x3ff80000 file_offs 0x00013d00 SOC_RTC_DRAM
Segment 3: len 0x00000 load 0x3ff80000 file_offs 0x00013d08 SOC_RTC_DRAM
Segment 4: len 0x028e0 load 0x3ffb0000 file_offs 0x00013d10 DRAM
Segment 5: len 0x00000 load 0x3ffb28e0 file_offs 0x000165f8 DRAM
Segment 6: len 0x00400 load 0x40080000 file_offs 0x00016600 SOC_IRAM
Segment 7: len 0x09600 load 0x40080400 file_offs 0x00016a08 SOC_IRAM
Segment 8: len 0x62e4c load 0x400d0018 file_offs 0x00020010 SOC_IROM
Segment 9: len 0x06cec load 0x40089a00 file_offs 0x00082e64 SOC_IROM
Segment 10: len 0x00000 load 0x400c0000 file_offs 0x00089b58 SOC_RTC_IRAM
Segment 11: len 0x00004 load 0x50000000 file_offs 0x00089b60 SOC_RTC_DATA
Segment 12: len 0x00000 load 0x50000004 file_offs 0x00089b6c SOC_RTC_DATA
Segment 13: len 0x00000 load 0x50000004 file_offs 0x00089b74 SOC_RTC_DATA
Checksum: e8 (valid)Validation Hash:␣
↪407089ca0eae2bbf83b4120979d3354b1c938a49cb7a0c997f240474ef2ec76b (valid)
```

You can also see the information on segments in the IDF logs while your application is booting:

```
I (443) esp_image: segment 0: paddr=0x00020020 vaddr=0x3f400020 size=0x13ce0 ( 81120) map
I (489) esp_image: segment 1: paddr=0x00033d08 vaddr=0x3ff80000 size=0x00000 ( 0) load
```

(下页继续)

(续上页)

```
I (530) esp_image: segment 2: paddr=0x00033d10 vaddr=0x3ff80000 size=0x00000 ( 0) load
I (571) esp_image: segment 3: paddr=0x00033d18 vaddr=0x3ffb0000 size=0x028e0 ( 10464)␣
↪load
I (612) esp_image: segment 4: paddr=0x00036600 vaddr=0x3ffb28e0 size=0x00000 ( 0) load
I (654) esp_image: segment 5: paddr=0x00036608 vaddr=0x40080000 size=0x00400 ( 1024) load
I (695) esp_image: segment 6: paddr=0x00036a10 vaddr=0x40080400 size=0x09600 ( 38400)␣
↪load
I (737) esp_image: segment 7: paddr=0x00040018 vaddr=0x400d0018 size=0x62e4c (405068) map
I (847) esp_image: segment 8: paddr=0x000a2e6c vaddr=0x40089a00 size=0x06cec ( 27884)␣
↪load
I (888) esp_image: segment 9: paddr=0x000a9b60 vaddr=0x400c0000 size=0x00000 ( 0) load
I (929) esp_image: segment 10: paddr=0x000a9b68 vaddr=0x50000000 size=0x00004 ( 4) load
I (971) esp_image: segment 11: paddr=0x000a9b74 vaddr=0x50000004 size=0x00000 ( 0) load
I (1012) esp_image: segment 12: paddr=0x000a9b7c vaddr=0x50000004 size=0x00000 ( 0) load
```

For more details on the type of memory segments and their address ranges, see the ESP32 Technical Reference Manual, Section 1.3.2 *Embedded Memory*.

3. The image has a single checksum byte after the last segment. This byte is written on a sixteen byte padded boundary, so the application image might need padding.

4. If the `hash_appended` field from *esp_image_header_t* is set then a SHA256 checksum will be appended. The value of SHA256 is calculated on the range from first byte and up to this field. The length of this field is 32 bytes.

5. If the options *CONFIG_SECURE_SIGNED_APPS_SCHEME* is set to ECDSA then the application image will have additional 68 bytes for an ECDSA signature, which includes:

- version word (4 bytes),
- signature data (64 bytes).

Application Description

The `DROM` segment starts with the *esp_app_desc_t* structure which carries specific fields describing the application:

- `secure_version` - see *Anti-rollback*.
- `version` - see *App version*. *
- `project_name` is filled from `PROJECT_NAME`. *
- `time` and `date` - compile time and date.
- `idf_ver` - version of ESP-IDF. *

- `app_elf_sha256` - contains sha256 for the elf application file.

`*` - The maximum length is 32 characters, including null-termination character. For example, if the length of `PROJECT_NAME` exceeds 32 characters, the excess characters will be disregarded.

This structure is useful for identification of images uploaded OTA because it has a fixed offset = sizeof(*esp_image_header_t*) + sizeof(*esp_image_segment_header_t*). As soon as a device receives the first fragment containing this structure, it has all the information to determine whether the update should be continued or not.

Adding a Custom Structure to an Application

Customer also has the opportunity to have similar structure with a fixed offset relative to the beginning of the image. The following pattern can be used to add a custom structure to your image:

```
const __attribute__((section(".rodata_custom_desc"))) esp_custom_app_desc_t custom_app_
↪desc = { ... }
```

Offset for custom structure is sizeof(*esp_image_header_t*) + sizeof(*esp_image_segment_header_t*) + sizeof(*esp_app_desc_t*).

To guarantee that the custom structure is located in the image even if it is not used, you need to add:

- For Make: add `COMPONENT_ADD_LDFLAGS += -u custom_app_desc` into `component.mk`
- For Cmake: add `target_link_libraries(${COMPONENT_TARGET} "-u custom_app_desc")` into `CMakeLists.txt`

API Reference

Header File

- bootloader_support/include/esp_app_format.h

Structures

struct esp_image_header_t

Main header of binary image.

Public Members

uint8_t **magic**

Magic word ESP_IMAGE_HEADER_MAGIC

uint8_t **segment_count**

Count of memory segments

uint8_t **spi_mode**

flash read mode (esp_image_spi_mode_t as uint8_t)

uint8_t **spi_speed**

flash frequency (esp_image_spi_freq_t as uint8_t)

uint8_t **spi_size**

flash chip size (esp_image_flash_size_t as uint8_t)

uint32_t **entry_addr**

Entry address

uint8_t **wp_pin**

WP pin when SPI pins set via efuse (read by ROM bootloader, the IDF bootloader uses software to configure the WP pin and sets this field to 0xEE=disabled)

uint8_t **spi_pin_drv**[3]

Drive settings for the SPI flash pins (read by ROM bootloader)

esp_chip_id_t **chip_id**

Chip identification number

uint8_t **min_chip_rev**

Minimum chip revision supported by image

uint8_t **reserved**[8]

Reserved bytes in additional header space, currently unused

uint8_t **hash_appended**

If 1, a SHA256 digest "simple hash" (of the entire image) is appended after the checksum. Included in image length. This digest is separate to secure boot and only used for detecting corruption. For secure boot signed images, the signature is appended after this (and the simple hash is included in the signed data).

struct esp_image_segment_header_t

Header of binary image segment.

Public Members

uint32_t **load_addr**

Address of segment

uint32_t **data_len**

Length of data

struct esp_app_desc_t

Description about application.

Public Members

uint32_t **magic_word**
Magic word ESP_APP_DESC_MAGIC_WORD

uint32_t **secure_version**
Secure version

uint32_t **reserv1**[2]
reserv1

char **version**[32]
Application version

char **project_name**[32]
Project name

char **time**[16]
Compile time

char **date**[16]
Compile date

char **idf_ver**[32]
Version IDF

uint8_t **app_elf_sha256**[32]
sha256 of elf file

uint32_t **reserv2**[20]
reserv2

Macros

ESP_IMAGE_HEADER_MAGIC
The magic word for the *esp_image_header_t* structure.

ESP_IMAGE_MAX_SEGMENTS
Max count of segments in the image.

ESP_APP_DESC_MAGIC_WORD
The magic word for the esp_app_desc structure that is in DROM.

Enumerations

enum esp_chip_id_t
ESP chip ID.

Values:

`ESP_CHIP_ID_ESP32` = 0x0000
chip ID: ESP32

`ESP_CHIP_ID_INVALID` = 0xFFFF
Invalid chip ID (we defined it to make sure the esp_chip_id_t is 2 bytes size)

enum esp_image_spi_mode_t
SPI flash mode, used in *esp_image_header_t*.

Values:

`ESP_IMAGE_SPI_MODE_QIO`
SPI mode QIO

`ESP_IMAGE_SPI_MODE_QOUT`
SPI mode QOUT

`ESP_IMAGE_SPI_MODE_DIO`
SPI mode DIO

`ESP_IMAGE_SPI_MODE_DOUT`
SPI mode DOUT

`ESP_IMAGE_SPI_MODE_FAST_READ`
SPI mode FAST_READ

`ESP_IMAGE_SPI_MODE_SLOW_READ`
SPI mode SLOW_READ

enum esp_image_spi_freq_t
SPI flash clock frequency.

Values:

`ESP_IMAGE_SPI_SPEED_40M`
SPI clock frequency 40 MHz

`ESP_IMAGE_SPI_SPEED_26M`
SPI clock frequency 26 MHz

`ESP_IMAGE_SPI_SPEED_20M`
SPI clock frequency 20 MHz

`ESP_IMAGE_SPI_SPEED_80M` = 0xF
SPI clock frequency 80 MHz

enum esp_image_flash_size_t
Supported SPI flash sizes.

Values:

`ESP_IMAGE_FLASH_SIZE_1MB` = 0
SPI flash size 1 MB

`ESP_IMAGE_FLASH_SIZE_2MB`
SPI flash size 2 MB

`ESP_IMAGE_FLASH_SIZE_4MB`
SPI flash size 4 MB

`ESP_IMAGE_FLASH_SIZE_8MB`
SPI flash size 8 MB

`ESP_IMAGE_FLASH_SIZE_16MB`
SPI flash size 16 MB

`ESP_IMAGE_FLASH_SIZE_MAX`
SPI flash size MAX

2.7.2 Application Level Tracing

Overview

IDF provides useful feature for program behaviour analysis: application level tracing. It is implemented in the corresponding library and can be enabled via menuconfig. This feature allows to transfer arbitrary data between host and ESP32 via JTAG interface with small overhead on program execution. Developers can use this library to send application specific state of execution to the host and receive commands or other type of info in the opposite direction at runtime. The main use cases of this library are:

1. Collecting application specific data, see 特定应用程序的跟踪
2. Lightweight logging to the host, see 记录日志到主机
3. System behaviour analysis, see 基于 *SEGGER SystemView* 的系统行为分析

API Reference

Header File

- app_trace/include/esp_app_trace.h

Functions

esp_err_t **esp_apptrace_init**(void)
Initializes application tracing module.

Note Should be called before any esp_apptrace_xxx call.

Return ESP_OK on success, otherwise see esp_err_t

void **esp_apptrace_down_buffer_config**(uint8_t *_buf_, uint32_t _size_)

Configures down buffer.

Note Needs to be called before initiating any data transfer using esp_apptrace_buffer_get and esp_apptrace_write. This function does not protect internal data by lock.

Parameters

- `buf`: Address of buffer to use for down channel (host to target) data.
- `size`: Size of the buffer.

uint8_t ***esp_apptrace_buffer_get**(*esp_apptrace_dest_t* _dest_, uint32_t _size_, uint32_t _tmo_)

Allocates buffer for trace data. After data in buffer are ready to be sent off esp_apptrace_buffer_put must be called to indicate it.

Return non-NULL on success, otherwise NULL.

Parameters

- `dest`: Indicates HW interface to send data.
- `size`: Size of data to write to trace buffer.
- `tmo`: Timeout for operation (in us). Use ESP_APPTRACE_TMO_INFINITE to wait indefinitely.

esp_err_t **esp_apptrace_buffer_put**(*esp_apptrace_dest_t* _dest_, uint8_t *_ptr_, uint32_t _tmo_)

Indicates that the data in buffer are ready to be sent off. This function is a counterpart of and must be preceeded by esp_apptrace_buffer_get.

Return ESP_OK on success, otherwise see esp_err_t

Parameters

- `dest`: Indicates HW interface to send data. Should be identical to the same parameter in call to esp_apptrace_buffer_get.
- `ptr`: Address of trace buffer to release. Should be the value returned by call to esp_apptrace_buffer_get.
- `tmo`: Timeout for operation (in us). Use ESP_APPTRACE_TMO_INFINITE to wait indefinitely.

esp_err_t **esp_apptrace_write**(*esp_apptrace_dest_t* _dest_, **const** void *_data_, uint32_t _size_, uint32_t _tmo_)

Writes data to trace buffer.

Return ESP_OK on success, otherwise see esp_err_t

Parameters

- `dest`: Indicates HW interface to send data.
- `data`: Address of data to write to trace buffer.
- `size`: Size of data to write to trace buffer.
- `tmo`: Timeout for operation (in us). Use ESP_APPTRACE_TMO_INFINITE to wait indefinitely.

int **esp_apptrace_vprintf_to**(*esp_apptrace_dest_t* *dest*, uint32_t *tmo*, **const** char ***fmt*, va_list *ap*)

vprintf-like function to sent log messages to host via specified HW interface.

Return Number of bytes written.

Parameters

- `dest`: Indicates HW interface to send data.
- `tmo`: Timeout for operation (in us). Use ESP_APPTRACE_TMO_INFINITE to wait indefinitely.
- `fmt`: Address of format string.
- `ap`: List of arguments.

int **esp_apptrace_vprintf**(**const** char ***fmt*, va_list *ap*)

vprintf-like function to sent log messages to host.

Return Number of bytes written.

Parameters

- `fmt`: Address of format string.
- `ap`: List of arguments.

esp_err_t **esp_apptrace_flush**(*esp_apptrace_dest_t* *dest*, uint32_t *tmo*)

Flushes remaining data in trace buffer to host.

Return ESP_OK on success, otherwise see esp_err_t

Parameters

- `dest`: Indicates HW interface to flush data on.
- `tmo`: Timeout for operation (in us). Use ESP_APPTRACE_TMO_INFINITE to wait indefinitely.

esp_err_t **esp_apptrace_flush_nolock**(*esp_apptrace_dest_t* *dest*, uint32_t *min_sz*, uint32_t *tmo*)

Flushes remaining data in trace buffer to host without locking internal data. This is special version of esp_apptrace_flush which should be called from panic handler.

Return ESP_OK on success, otherwise see esp_err_t

Parameters

- `dest`: Indicates HW interface to flush data on.
- `min_sz`: Threshold for flushing data. If current filling level is above this value, data will be flushed. TRAX destinations only.
- `tmo`: Timeout for operation (in us). Use ESP_APPTRACE_TMO_INFINITE to wait indefinitely.

esp_err_t **esp_apptrace_read**(*esp_apptrace_dest_t dest*, void **data*, uint32_t **size*, uint32_t *tmo*)

Reads host data from trace buffer.

Return ESP_OK on success, otherwise see esp_err_t

Parameters

- `dest`: Indicates HW interface to read the data on.
- `data`: Address of buffer to put data from trace buffer.
- `size`: Pointer to store size of read data. Before call to this function pointed memory must hold requested size of data
- `tmo`: Timeout for operation (in us). Use ESP_APPTRACE_TMO_INFINITE to wait indefinitely.

uint8_t ***esp_apptrace_down_buffer_get**(*esp_apptrace_dest_t dest*, uint32_t **size*, uint32_t *tmo*)

Retrieves incoming data buffer if any. After data in buffer are processed esp_apptrace_down_buffer_put must be called to indicate it.

Return non-NULL on success, otherwise NULL.

Parameters

- `dest`: Indicates HW interface to receive data.
- `size`: Address to store size of available data in down buffer. Must be initialized with requested value.
- `tmo`: Timeout for operation (in us). Use ESP_APPTRACE_TMO_INFINITE to wait indefinitely.

esp_err_t **esp_apptrace_down_buffer_put**(*esp_apptrace_dest_t dest*, uint8_t **ptr*, uint32_t *tmo*)

Indicates that the data in down buffer are processed. This function is a counterpart of and must be preceeded by esp_apptrace_down_buffer_get.

Return ESP_OK on success, otherwise see esp_err_t

Parameters

- `dest`: Indicates HW interface to receive data. Should be identical to the same parameter in call to esp_apptrace_down_buffer_get.
- `ptr`: Address of trace buffer to release. Should be the value returned by call to esp_apptrace_down_buffer_get.
- `tmo`: Timeout for operation (in us). Use ESP_APPTRACE_TMO_INFINITE to wait indefinitely.

bool **esp_apptrace_host_is_connected**(*esp_apptrace_dest_t dest*)

Checks whether host is connected.

Return true if host is connected, otherwise false

Parameters

- `dest`: Indicates HW interface to use.

void ***esp_apptrace_fopen**(*esp_apptrace_dest_t dest*, **const** char **path*, **const** char **mode*)

Opens file on host. This function has the same semantic as 'fopen' except for the first argument.

Return non zero file handle on success, otherwise 0

Parameters

- `dest`: Indicates HW interface to use.
- `path`: Path to file.
- `mode`: Mode string. See fopen for details.

int **esp_apptrace_fclose**(*esp_apptrace_dest_t dest*, void **stream*)

Closes file on host. This function has the same semantic as 'fclose' except for the first argument.

Return Zero on success, otherwise non-zero. See fclose for details.

Parameters

- `dest`: Indicates HW interface to use.
- `stream`: File handle returned by esp_apptrace_fopen.

size_t **esp_apptrace_fwrite**(*esp_apptrace_dest_t dest*, **const** void **ptr*, size_t *size*, size_t *nmemb*, void **stream*)

Writes to file on host. This function has the same semantic as 'fwrite' except for the first argument.

Return Number of written items. See fwrite for details.

Parameters

- `dest`: Indicates HW interface to use.

- `ptr`: Address of data to write.
- `size`: Size of an item.
- `nmemb`: Number of items to write.
- `stream`: File handle returned by esp_apptrace_fopen.

size_t **esp_apptrace_fread**(*esp_apptrace_dest_t* *dest*, void **ptr*, size_t *size*, size_t *nmemb*, void **stream*)

Read file on host. This function has the same semantic as 'fread' except for the first argument.

Return Number of read items. See fread for details.

Parameters

- `dest`: Indicates HW interface to use.
- `ptr`: Address to store read data.
- `size`: Size of an item.
- `nmemb`: Number of items to read.
- `stream`: File handle returned by esp_apptrace_fopen.

int **esp_apptrace_fseek**(*esp_apptrace_dest_t* *dest*, void **stream*, long *offset*, int *whence*)

Set position indicator in file on host. This function has the same semantic as 'fseek' except for the first argument.

Return Zero on success, otherwise non-zero. See fseek for details.

Parameters

- `dest`: Indicates HW interface to use.
- `stream`: File handle returned by esp_apptrace_fopen.
- `offset`: Offset. See fseek for details.
- `whence`: Position in file. See fseek for details.

int **esp_apptrace_ftell**(*esp_apptrace_dest_t* *dest*, void **stream*)

Get current position indicator for file on host. This function has the same semantic as 'ftell' except for the first argument.

Return Current position in file. See ftell for details.

Parameters

- `dest`: Indicates HW interface to use.
- `stream`: File handle returned by esp_apptrace_fopen.

int **esp_apptrace_fstop**(*esp_apptrace_dest_t* *dest*)

Indicates to the host that all file operations are completed. This function should be called after all file operations are finished and indicate to the host that it can perform cleanup operations (close open files etc.).

Return ESP_OK on success, otherwise see esp_err_t

Parameters

- `dest`: Indicates HW interface to use.

void **esp_gcov_dump**(void)

Triggers gcov info dump. This function waits for the host to connect to target before dumping data.

Enumerations

enum esp_apptrace_dest_t

Application trace data destinations bits.

Values:

ESP_APPTRACE_DEST_TRAX = 0x1

JTAG destination.

ESP_APPTRACE_DEST_UART0 = 0x2

UART destination.

Header File

- app_trace/include/esp_sysview_trace.h

Functions

static *esp_err_t* **esp_sysview_flush**(uint32_t *tmo*)

Flushes remaining data in SystemView trace buffer to host.

Return ESP_OK.

Parameters

- `tmo`: Timeout for operation (in us). Use ESP_APPTRACE_TMO_INFINITE to wait indefinetly.

int **esp_sysview_vprintf**(**const** char ***format*, va_list *args*)

vprintf-like function to sent log messages to the host.

Return Number of bytes written.

Parameters

- `format`: Address of format string.
- `args`: List of arguments.

esp_err_t **esp_sysview_heap_trace_start**(uint32_t *tmo*)

Starts SystemView heap tracing.

Return ESP_OK on success, ESP_ERR_TIMEOUT if operation has been timed out.

Parameters

- `tmo`: Timeout (in us) to wait for the host to be connected. Use -1 to wait forever.

esp_err_t **esp_sysview_heap_trace_stop**(void)

Stops SystemView heap tracing.

Return ESP_OK.

void **esp_sysview_heap_trace_alloc**(void **addr*, uint32_t *size*, **const** void **callers*)

Sends heap allocation event to the host.

Parameters

- `addr`: Address of allocated block.
- `size`: Size of allocated block.
- `callers`: Pointer to array with callstack addresses. Array size must be CONFIG_HEAP_TRACING_STACK_DEPTH.

void **esp_sysview_heap_trace_free**(void **addr*, **const** void **callers*)

Sends heap de-allocation event to the host.

Parameters

- `addr`: Address of de-allocated block.
- `callers`: Pointer to array with callstack addresses. Array size must be CONFIG_HEAP_TRACING_STACK_DEPTH.

2.7.3 eFuse Manager

Introduction

The eFuse Manager library is designed to structure access to eFuse bits and make using these easy. This library operates eFuse bits by a structure name which is assigned in eFuse table. This sections introduces some concepts used by eFuse Manager.

Hardware description

The ESP32 has a number of eFuses which can store system and user parameters. Each eFuse is a one-bit field which can be programmed to 1 after which it cannot be reverted back to 0. Some of system parameters are using these eFuse bits directly by hardware modules and have special place (for example EFUSE_BLK0). For more details see ESP32 Technical Reference Manual in part 20 eFuse controller. Some eFuse bits are available for user applications.

ESP32 has 4 eFuse blocks each of the size of 256 bits (not all bits are available):

- EFUSE_BLK0 is used entirely for system purposes;
- EFUSE_BLK1 is used for flash encrypt key. If not using that Flash Encryption feature, they can be used for another purpose;
- EFUSE_BLK2 is used for security boot key. If not using that Secure Boot feature, they can be used for another purpose;
- EFUSE_BLK3 can be partially reserved for the custom MAC address, or used entirely for user application. Note that some bits are already used in IDF.

Each block is divided into 8 32-bits registers.

eFuse Manager component

The component has API functions for reading and writing fields. Access to the fields is carried out through the structures that describe the location of the eFuse bits in the blocks. The component provides the ability to form fields of any length and from any number of individual bits. The description of the fields is made in a CSV file in a table form. To generate from a tabular form (CSV file) in the C-source uses the tool *efuse_table_gen.py*. The tool checks the CSV file for uniqueness of field names and bit intersection, in case of using a *custom* file from the user' s project directory, the utility will check with the *common* CSV file.

CSV files:

- common (*esp_efuse_table.csv*) - contains eFuse fields which are used inside the IDF. C-source generation should be done manually when changing this file (run command `idf.py efuse_common_table`). Note that changes in this file can lead to incorrect operation.
- custom - (optional and can be enabled by `CONFIG_EFUSE_CUSTOM_TABLE`) contains eFuse fields that are used by the user in their application. C-source generation should be done manually when changing this file and running `idf.py efuse_custom_table`.

Description CSV file

The CSV file contains a description of the eFuse fields. In the simple case, one field has one line of description. Table header:

```
# field_name,  efuse_block(EFUSE_BLK0..EFUSE_BLK3), bit_start(0..255),  bit_count(1..
↪256),              comment
```

Individual params in CSV file the following meanings:

field_name Name of field. The prefix *ESP_EFUSE_* will be added to the name, and this field name will be available in the code. This name will be used to access the fields. The name must be unique for all fields. If the line has an empty name, then this line is combined with the previous field. This allows you to set an arbitrary order of bits in the field, and expand the field as well (see `MAC_FACTORY` field in the common table).

efuse_block Block number. It determines where the eFuse bits will be placed for this field. Available EFUSE_BLK0..EFUSE_BLK3.

bit_start Start bit number (0..255). The bit_start field can be omitted. In this case, it will be set to bit_start + bit_count from the previous record, if it has the same efuse_block. Otherwise (if efuse_block is different, or this is the first entry), an error will be generated.

bit_count The number of bits to use in this field (1..-). This parameter can not be omitted. This field also may be `MAX_BLK_LEN` in this case, the field length will have the maximum block length, taking into account the coding scheme (applicable for `ESP_EFUSE_SECURE_BOOT_KEY` and `ESP_EFUSE_ENCRYPT_FLASH_KEY` fields). The value `MAX_BLK_LEN` depends on `CONFIG_EFUSE_MAX_BLK_LEN`, will be replaced with "None" - 256, "3/4" - 192, "REPEAT" - 128.

comment This param is using for comment field, it also move to C-header file. The comment field can be omitted.

If a non-sequential bit order is required to describe a field, then the field description in the following lines should be continued without specifying a name, this will indicate that it belongs to one field. For example two fields MAC_FACTORY and MAC_FACTORY_CRC:

```
# Factory MAC address #
#######################
MAC_FACTORY,            EFUSE_BLK0,    72,    8,    Factory MAC addr [0]
,                       EFUSE_BLK0,    64,    8,    Factory MAC addr [1]
,                       EFUSE_BLK0,    56,    8,    Factory MAC addr [2]
,                       EFUSE_BLK0,    48,    8,    Factory MAC addr [3]
,                       EFUSE_BLK0,    40,    8,    Factory MAC addr [4]
,                       EFUSE_BLK0,    32,    8,    Factory MAC addr [5]
MAC_FACTORY_CRC,        EFUSE_BLK0,    80,    8,    CRC8 for factory MAC address
```

This field will available in code as ESP_EFUSE_MAC_FACTORY and ESP_EFUSE_MAC_FACTORY_CRC.

efuse_table_gen.py tool

The tool is designed to generate C-source files from CSV file and validate fields. First of all, the check is carried out on the uniqueness of the names and overlaps of the field bits. If an additional *custom* file is used, it will be checked with the existing *common* file (esp_efuse_table.csv). In case of errors, a message will be displayed and the string that caused the error. C-source files contain structures of type *esp_efuse_desc_t*.

To generate a *common* files, use the following command `idf.py efuse_common_table` or:

```
cd $IDF_PATH/components/efuse/
./efuse_table_gen.py esp32/esp_efuse_table.csv
```

After generation in the folder *esp32* create:

- *esp_efuse_table.c* file.
- In *include* folder *esp_efuse_table.c* file.

To generate a *custom* files, use the following command `idf.py efuse_custom_table` or:

```
cd $IDF_PATH/components/efuse/
./efuse_table_gen.py esp32/esp_efuse_table.csv PROJECT_PATH/main/esp_efuse_custom_table.
↪csv
```

After generation in the folder PROJECT_PATH/main create:

- *esp_efuse_custom_table.c* file.
- In *include* folder *esp_efuse_custom_table.c* file.

To use the generated fields, you need to include two files:

```
#include "esp_efuse.h"
#include "esp_efuse_table.h" or "esp_efuse_custom_table.h"
```

Support coding scheme

eFuse have three coding schemes:

- `None` (value 0).
- `3/4` (value 1).
- `Repeat` (value 2).

The coding scheme affects only EFUSE_BLK1, EFUSE_BLK2 and EFUSE_BLK3 blocks. EUSE_BLK0 block always has a coding scheme `None`. Coding changes the number of bits that can be written into a block, the block length is constant 256, some of these bits are used for encoding and are not used.

When using a coding scheme, the length of the payload that can be written is limited (for more details `20.3.1.3 System Parameter coding_scheme`):

- None 256 bits.
- 3/4 192 bits.
- Repeat 128 bits.

You can find out the coding scheme of your chip:

- run a `espefuse.py -p COM4 summary` command.
- from `esptool` utility logs (during flashing).
- calling the function in the code *esp_efuse_get_coding_scheme()* for the EFUSE_BLK3 block.

eFuse tables must always comply with the coding scheme in the chip. There is an `EFUSE_CODE_SCHEME_SELECTOR` option to select the coding type for tables in a Kconfig. When generating source files, if your tables do not follow the coding scheme, an error message will be displayed. Adjust the length or offset fields. If your program was compiled with `None` encoding and `3/4` is used in the chip, then the `ESP_ERR_CODING` error may occur when calling the eFuse API (the field is outside the block boundaries). If the field matches the new block boundaries, then the API will work without errors.

Also, 3/4 coding scheme imposes restrictions on writing bits belonging to one coding unit. The whole block with a length of 256 bits is divided into 4 coding units, and in each coding unit there are 6 bytes of useful data and 2 service bytes. These 2 service bytes contain the checksum of the previous 6 data bytes.

It turns out that only one field can be written into one coding unit. Repeated rewriting in one coding unit is prohibited. But if the record was made in advance or through a *esp_efuse_write_block()* function, then reading the fields belonging to one coding unit is possible.

In case `3/4` coding scheme, the writing process is divided into the coding units and we can not use the usual mode of writing some fields. We can prepare all the data for writing and burn it in one time. You can also use this mode for `None` coding scheme but it is not necessary. It is important for `3/4` coding scheme. To write some fields in one time need to use `the batch writing mode`. Firstly set this mode through *esp_efuse_batch_write_begin()* function then write some fields as usual use the `esp_efuse_write_...` functions. At the end to burn they, need to call the *esp_efuse_batch_write_commit()* function. It burns prepared data to the efuse blocks and disable the `batch recording mode`. `The batch writing mode` blocks `esp_efuse_read_...` operations.

After changing the coding scheme, run `efuse_common_table` and `efuse_custom_table` commands to check the tables of the new coding scheme.

eFuse API

Access to the fields is via a pointer to the description structure. API functions have some basic operation:

- *esp_efuse_read_field_blob()* - returns an array of read eFuse bits.
- *esp_efuse_read_field_cnt()* - returns the number of bits programmed as "1".
- *esp_efuse_write_field_blob()* - writes an array.
- *esp_efuse_write_field_cnt()* - writes a required count of bits as "1".
- *esp_efuse_get_field_size()* - returns the number of bits by the field name.
- *esp_efuse_read_reg()* - returns value of eFuse register.
- *esp_efuse_write_reg()* - writes value to eFuse register.
- *esp_efuse_get_coding_scheme()* - returns eFuse coding scheme for blocks.
- *esp_efuse_read_block()* - reads key to eFuse block starting at the offset and the required size.
- *esp_efuse_write_block()* - writes key to eFuse block starting at the offset and the required size.
- *esp_efuse_batch_write_begin()* - set the batch mode of writing fields.
- *esp_efuse_batch_write_commit()* - writes all prepared data for batch writing mode and reset the batch writing mode.
- *esp_efuse_batch_write_cancel()* - reset the batch writing mode and prepared data.

For frequently used fields, special functions are made, like this *esp_efuse_get_chip_ver()*, *esp_efuse_get_pkg_ver()*.

How add a new field

1. Find a free bits for field. Show *esp_efuse_table.csv* file or run `idf.py show_efuse_table` or the next command:

```
$ ./efuse_table_gen.py esp32/esp_efuse_table.csv --info
eFuse coding scheme: NONE
#       field_name                      efuse_block     bit_start       bit_count
1       WR_DIS_FLASH_CRYPT_CNT          EFUSE_BLK0          2               1
2       WR_DIS_BLK1                     EFUSE_BLK0          7               1
3       WR_DIS_BLK2                     EFUSE_BLK0          8               1
4       WR_DIS_BLK3                     EFUSE_BLK0          9               1
5       RD_DIS_BLK1                     EFUSE_BLK0          16              1
6       RD_DIS_BLK2                     EFUSE_BLK0          17              1
7       RD_DIS_BLK3                     EFUSE_BLK0          18              1
8       FLASH_CRYPT_CNT                 EFUSE_BLK0          20              7
```

(下页继续)

(续上页)

```
9       MAC_FACTORY                 EFUSE_BLK0          32                  8
10      MAC_FACTORY                 EFUSE_BLK0          40                  8
11      MAC_FACTORY                 EFUSE_BLK0          48                  8
12      MAC_FACTORY                 EFUSE_BLK0          56                  8
13      MAC_FACTORY                 EFUSE_BLK0          64                  8
14      MAC_FACTORY                 EFUSE_BLK0          72                  8
15      MAC_FACTORY_CRC             EFUSE_BLK0          80                  8
16      CHIP_VER_DIS_APP_CPU        EFUSE_BLK0          96                  1
17      CHIP_VER_DIS_BT             EFUSE_BLK0          97                  1
18      CHIP_VER_PKG                EFUSE_BLK0          105                 3
19      CHIP_CPU_FREQ_LOW           EFUSE_BLK0          108                 1
20      CHIP_CPU_FREQ_RATED         EFUSE_BLK0          109                 1
21      CHIP_VER_REV1               EFUSE_BLK0          111                 1
22      ADC_VREF_AND_SDIO_DREF      EFUSE_BLK0          136                 6
23      XPD_SDIO_REG                EFUSE_BLK0          142                 1
24      SDIO_TIEH                   EFUSE_BLK0          143                 1
25      SDIO_FORCE                  EFUSE_BLK0          144                 1
26      ENCRYPT_CONFIG              EFUSE_BLK0          188                 4
27      CONSOLE_DEBUG_DISABLE       EFUSE_BLK0          194                 1
28      ABS_DONE_0                  EFUSE_BLK0          196                 1
29      DISABLE_JTAG                EFUSE_BLK0          198                 1
30      DISABLE_DL_ENCRYPT          EFUSE_BLK0          199                 1
31      DISABLE_DL_DECRYPT          EFUSE_BLK0          200                 1
32      DISABLE_DL_CACHE            EFUSE_BLK0          201                 1
33      ENCRYPT_FLASH_KEY           EFUSE_BLK1          0                   256
34      SECURE_BOOT_KEY             EFUSE_BLK2          0                   256
35      MAC_CUSTOM_CRC              EFUSE_BLK3          0                   8
36      MAC_CUSTOM                  EFUSE_BLK3          8                   48
37      ADC1_TP_LOW                 EFUSE_BLK3          96                  7
38      ADC1_TP_HIGH                EFUSE_BLK3          103                 9
39      ADC2_TP_LOW                 EFUSE_BLK3          112                 7
40      ADC2_TP_HIGH                EFUSE_BLK3          119                 9
41      SECURE_VERSION              EFUSE_BLK3          128                 32
42      MAC_CUSTOM_VER              EFUSE_BLK3          184                 8

Used bits in eFuse table:
EFUSE_BLK0
[2 2] [7 9] [16 18] [20 27] [32 87] [96 97] [105 109] [111 111] [136 144] [188 191] [194␣
↪194] [196 196] [198 201]
```

(下页继续)

(续上页)

```
EFUSE_BLK1
[0 255]

EFUSE_BLK2
[0 255]

EFUSE_BLK3
[0 55] [96 159] [184 191]

Note: Not printed ranges are free for using. (bits in EFUSE_BLK0 are reserved for␣
↪Espressif)

Parsing eFuse CSV input file $IDF_PATH/components/efuse/esp32/esp_efuse_table.csv ...
Verifying eFuse table...
```

The number of bits not included in square brackets is free (bits in EFUSE_BLK0 are reserved for Espressif). All fields are checked for overlapping.

2. Fill a line for field: field_name, efuse_block, bit_start, bit_count, comment.
3. Run a `show_efuse_table` command to check eFuse table. To generate source files run `efuse_common_table` or `efuse_custom_table` command.

Debug eFuse & Unit tests

Virtual eFuses

The Kconfig option `CONFIG_EFUSE_VIRTUAL` will virtualize eFuse values inside the eFuse Manager, so writes are emulated and no eFuse values are permanently changed. This can be useful for debugging app and unit tests.

espefuse.py

esptool includes a useful tool for reading/writing ESP32 eFuse bits - espefuse.py.

```
espefuse.py -p COM4 summary

espefuse.py v2.3.1
Connecting........_
Security fuses:
FLASH_CRYPT_CNT        Flash encryption mode counter                    = 0 R/W (0x0)
```

(下页继续)

(续上页)

```
FLASH_CRYPT_CONFIG      Flash encryption config (key tweak bits)          = 0 R/W (0x0)
CONSOLE_DEBUG_DISABLE   Disable ROM BASIC interpreter fallback            = 1 R/W (0x1)
ABS_DONE_0              secure boot enabled for bootloader                = 0 R/W (0x0)
ABS_DONE_1              secure boot abstract 1 locked                     = 0 R/W (0x0)
JTAG_DISABLE            Disable JTAG                                      = 0 R/W (0x0)
DISABLE_DL_ENCRYPT      Disable flash encryption in UART bootloader       = 0 R/W (0x0)
DISABLE_DL_DECRYPT      Disable flash decryption in UART bootloader       = 0 R/W (0x0)
DISABLE_DL_CACHE        Disable flash cache in UART bootloader            = 0 R/W (0x0)
BLK1                    Flash encryption key
  = 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00␣
↪00 00 00 00 R/W
BLK2                    Secure boot key
  = 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00␣
↪00 00 00 00 R/W
BLK3                    Variable Block 3
  = 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 fa 87 02 91 00 00 00 00 00 00 00 00␣
↪00 00 00 00 R/W

Efuse fuses:
WR_DIS                  Efuse write disable mask                          = 0 R/W (0x0)
RD_DIS                  Efuse read disablemask                            = 0 R/W (0x0)
CODING_SCHEME           Efuse variable block length scheme                = 1 R/W (0x1)␣
↪(3/4)
KEY_STATUS              Usage of efuse block 3 (reserved)                 = 0 R/W (0x0)

Config fuses:
XPD_SDIO_FORCE          Ignore MTDI pin (GPIO12) for VDD_SDIO on reset    = 0 R/W (0x0)
XPD_SDIO_REG            If XPD_SDIO_FORCE, enable VDD_SDIO reg on reset   = 0 R/W (0x0)
XPD_SDIO_TIEH           If XPD_SDIO_FORCE & XPD_SDIO_REG, 1=3.3V 0=1.8V   = 0 R/W (0x0)
SPI_PAD_CONFIG_CLK      Override SD_CLK pad (GPIO6/SPICLK)                = 0 R/W (0x0)
SPI_PAD_CONFIG_Q        Override SD_DATA_0 pad (GPIO7/SPIQ)               = 0 R/W (0x0)
SPI_PAD_CONFIG_D        Override SD_DATA_1 pad (GPIO8/SPID)               = 0 R/W (0x0)
SPI_PAD_CONFIG_HD       Override SD_DATA_2 pad (GPIO9/SPIHD)              = 0 R/W (0x0)
SPI_PAD_CONFIG_CS0      Override SD_CMD pad (GPIO11/SPICS0)               = 0 R/W (0x0)
DISABLE_SDIO_HOST       Disable SDIO host                                 = 0 R/W (0x0)

Identity fuses:
MAC                     MAC Address
  = 84:0d:8e:18:8e:44 (CRC ad OK) R/W
CHIP_VER_REV1           Silicon Revision 1                                = 1 R/W (0x1)
```

(下页继续)

(续上页)

```
CHIP_VERSION            Reserved for future chip versions                = 2 R/W (0x2)
CHIP_PACKAGE            Chip package identifier                          = 0 R/W (0x0)

Calibration fuses:
BLK3_PART_RESERVE       BLOCK3 partially served for ADC calibration data  = 1 R/W (0x1)
ADC_VREF                Voltage reference calibration                    = 1114 R/W (0x2)
ADC1_TP_LOW             ADC1 150mV reading                               = 346 R/W (0x11)
ADC1_TP_HIGH            ADC1 850mV reading                               = 3285 R/W (0x5)
ADC2_TP_LOW             ADC2 150mV reading                               = 449 R/W (0x7)
ADC2_TP_HIGH            ADC2 850mV reading                               = 3362 R/W␣
↪(0x1f5)

Flash voltage (VDD_SDIO) determined by GPIO12 on reset (High for 1.8V, Low/NC for 3.3V).
```

To get a dump for all eFuse registers.

```
espefuse.py -p COM4 dump

$ espefuse.py -p COM4 dump
espefuse.py v2.3.1
Connecting........__
EFUSE block 0:
00000000 c403bb68 0082240a 00000000 00000035 00000000 00000000
EFUSE block 1:
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
EFUSE block 2:
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
EFUSE block 3:
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

Header File

- efuse/include/esp_efuse.h

Functions

esp_err_t **esp_efuse_read_field_blob**(const *esp_efuse_desc_t* **field*[], void **dst*, size_t *dst_size_bits*)

Reads bits from EFUSE field and writes it into an array.

The number of read bits will be limited to the minimum value from the description of the bits in "field"

structure or "dst_size_bits" required size. Use "esp_efuse_get_field_size()" function to determine the length of the field.

Return

- ESP_OK: The operation was successfully completed.
- ESP_ERR_INVALID_ARG: Error in the passed arguments.

Parameters

- `field`: A pointer to the structure describing the fields of efuse.
- `dst`: A pointer to array that will contain the result of reading.
- `dst_size_bits`: The number of bits required to read. If the requested number of bits is greater than the field, the number will be limited to the field size.

esp_err_t **esp_efuse_read_field_cnt(const** *esp_efuse_desc_t* ***field**[], size_t ***out_cnt**)

Reads bits from EFUSE field and returns number of bits programmed as "1".

If the bits are set not sequentially, they will still be counted.

Return

- ESP_OK: The operation was successfully completed.
- ESP_ERR_INVALID_ARG: Error in the passed arguments.

Parameters

- `field`: A pointer to the structure describing the fields of efuse.
- `out_cnt`: A pointer that will contain the number of programmed as "1" bits.

esp_err_t **esp_efuse_write_field_blob(const** *esp_efuse_desc_t* ***field**[], **const** void ***src**, size_t *src_size_bits*)

Writes array to EFUSE field.

The number of write bits will be limited to the minimum value from the description of the bits in "field" structure or "src_size_bits" required size. Use "esp_efuse_get_field_size()" function to determine the length of the field. After the function is completed, the writing registers are cleared.

Return

- ESP_OK: The operation was successfully completed.
- ESP_ERR_INVALID_ARG: Error in the passed arguments.
- ESP_ERR_EFUSE_REPEATED_PROG: Error repeated programming of programmed bits is strictly forbidden.
- ESP_ERR_CODING: Error range of data does not match the coding scheme.

Parameters

- `field`: A pointer to the structure describing the fields of efuse.

- `src`: A pointer to array that contains the data for writing.
- `src_size_bits`: The number of bits required to write.

esp_err_t **esp_efuse_write_field_cnt**(**const** *esp_efuse_desc_t* ***field*[], size_t *cnt*)

Writes a required count of bits as "1" to EFUSE field.

If there are no free bits in the field to set the required number of bits to "1", ESP_ERR_EFUSE_CNT_IS_FULL error is returned, the field will not be partially recorded. After the function is completed, the writing registers are cleared.

Return

- ESP_OK: The operation was successfully completed.
- ESP_ERR_INVALID_ARG: Error in the passed arguments.
- ESP_ERR_EFUSE_CNT_IS_FULL: Not all requested cnt bits is set.

Parameters

- `field`: A pointer to the structure describing the fields of efuse.
- `cnt`: Required number of programmed as "1" bits.

esp_err_t **esp_efuse_set_write_protect**(esp_efuse_block_t *blk*)

Sets a write protection for the whole block.

After that, it is impossible to write to this block. The write protection does not apply to block 0.

Return

- ESP_OK: The operation was successfully completed.
- ESP_ERR_INVALID_ARG: Error in the passed arguments.
- ESP_ERR_EFUSE_CNT_IS_FULL: Not all requested cnt bits is set.
- ESP_ERR_NOT_SUPPORTED: The block does not support this command.

Parameters

- `blk`: Block number of eFuse. (EFUSE_BLK1, EFUSE_BLK2 and EFUSE_BLK3)

esp_err_t **esp_efuse_set_read_protect**(esp_efuse_block_t *blk*)

Sets a read protection for the whole block.

After that, it is impossible to read from this block. The read protection does not apply to block 0.

Return

- ESP_OK: The operation was successfully completed.
- ESP_ERR_INVALID_ARG: Error in the passed arguments.
- ESP_ERR_EFUSE_CNT_IS_FULL: Not all requested cnt bits is set.
- ESP_ERR_NOT_SUPPORTED: The block does not support this command.

Parameters

- `blk`: Block number of eFuse. (EFUSE_BLK1, EFUSE_BLK2 and EFUSE_BLK3)

int **esp_efuse_get_field_size**(**const** *esp_efuse_desc_t* **field*[])

Returns the number of bits used by field.

Return Returns the number of bits used by field.

Parameters

- `field`: A pointer to the structure describing the fields of efuse.

uint32_t **esp_efuse_read_reg**(esp_efuse_block_t *blk*, unsigned int *num_reg*)

Returns value of efuse register.

This is a thread-safe implementation. Example: EFUSE_BLK2_RDATA3_REG where (blk=2, num_reg=3)

Return Value of register

Parameters

- `blk`: Block number of eFuse.
- `num_reg`: The register number in the block.

esp_err_t **esp_efuse_write_reg**(esp_efuse_block_t *blk*, unsigned int *num_reg*, uint32_t *val*)

Write value to efuse register.

Apply a coding scheme if necessary. This is a thread-safe implementation. Example: EFUSE_BLK3_WDATA0_REG where (blk=3, num_reg=0)

Return

- ESP_OK: The operation was successfully completed.
- ESP_ERR_EFUSE_REPEATED_PROG: Error repeated programming of programmed bits is strictly forbidden.

Parameters

- `blk`: Block number of eFuse.
- `num_reg`: The register number in the block.
- `val`: Value to write.

esp_efuse_coding_scheme_t **esp_efuse_get_coding_scheme**(esp_efuse_block_t *blk*)

Return efuse coding scheme for blocks.

Note: The coding scheme is applicable only to 1, 2 and 3 blocks. For 0 block, the coding scheme is always `NONE`.

Return Return efuse coding scheme for blocks

Parameters

- `blk`: Block number of eFuse.

esp_err_t **esp_efuse_read_block**(esp_efuse_block_t *blk*, void **dst_key*, size_t *offset_in_bits*, size_t *size_bits*)

Read key to efuse block starting at the offset and the required size.

Return

- ESP_OK: The operation was successfully completed.
- ESP_ERR_INVALID_ARG: Error in the passed arguments.
- ESP_ERR_CODING: Error range of data does not match the coding scheme.

Parameters

- `blk`: Block number of eFuse.
- `dst_key`: A pointer to array that will contain the result of reading.
- `offset_in_bits`: Start bit in block.
- `size_bits`: The number of bits required to read.

esp_err_t **esp_efuse_write_block**(esp_efuse_block_t *blk*, **const** void **src_key*, size_t *offset_in_bits*, size_t *size_bits*)

Write key to efuse block starting at the offset and the required size.

Return

- ESP_OK: The operation was successfully completed.
- ESP_ERR_INVALID_ARG: Error in the passed arguments.
- ESP_ERR_CODING: Error range of data does not match the coding scheme.
- ESP_ERR_EFUSE_REPEATED_PROG: Error repeated programming of programmed bits

Parameters

- `blk`: Block number of eFuse.
- `src_key`: A pointer to array that contains the key for writing.
- `offset_in_bits`: Start bit in block.
- `size_bits`: The number of bits required to write.

uint8_t **esp_efuse_get_chip_ver**(void)

Returns chip version from efuse.

Return chip version

uint32_t **esp_efuse_get_pkg_ver**(void)

Returns chip package from efuse.

Return chip package

void **esp_efuse_burn_new_values**(void)

void **esp_efuse_reset**(void)

void **esp_efuse_disable_basic_rom_console**(void)

void **esp_efuse_write_random_key**(uint32_t *blk_wdata0_reg*)

uint32_t **esp_efuse_read_secure_version**(void)

bool **esp_efuse_check_secure_version**(uint32_t *secure_version*)

esp_err_t **esp_efuse_update_secure_version**(uint32_t *secure_version*)

void **esp_efuse_init**(uint32_t *offset*, uint32_t *size*)

esp_err_t **esp_efuse_batch_write_begin**(void)

esp_err_t **esp_efuse_batch_write_cancel**(void)

esp_err_t **esp_efuse_batch_write_commit**(void)

Structures

struct esp_efuse_desc_t

Structure eFuse field.

Public Members

esp_efuse_block_t **efuse_block**

Block of eFuse

uint8_t **bit_start**

Start bit [0..255]

uint16_t **bit_count**

Length of bit field [1..-]

Macros

ESP_ERR_EFUSE

Base error code for efuse api.

`ESP_OK_EFUSE_CNT`
: OK the required number of bits is set.

`ESP_ERR_EFUSE_CNT_IS_FULL`
: Error field is full.

`ESP_ERR_EFUSE_REPEATED_PROG`
: Error repeated programming of programmed bits is strictly forbidden.

`ESP_ERR_CODING`
: Error while a encoding operation.

2.7.4 Error Codes and Helper Functions

This section lists definitions of common ESP-IDF error codes and several helper functions related to error handling.

For general information about error codes in ESP-IDF, see *Error Handling*.

For the full list of error codes defined in ESP-IDF, see *Error Code Reference*.

API Reference

Header File

- esp_common/include/esp_err.h

Functions

`const` char *`esp_err_to_name`(*esp_err_t code*)
: Returns string for esp_err_t error codes.

  This function finds the error code in a pre-generated lookup-table and returns its string representation.

  The function is generated by the Python script tools/gen_esp_err_to_name.py which should be run each time an esp_err_t error is modified, created or removed from the IDF project.

  Return string error message

  Parameters

  - `code`: esp_err_t error code

`const` char *`esp_err_to_name_r`(*esp_err_t code*, char **buf*, size_t *buflen*)
: Returns string for esp_err_t and system error codes.

This function finds the error code in a pre-generated lookup-table of esp_err_t errors and returns its string representation. If the error code is not found then it is attempted to be found among system errors.

The function is generated by the Python script tools/gen_esp_err_to_name.py which should be run each time an esp_err_t error is modified, created or removed from the IDF project.

Return buf containing the string error message

Parameters

- `code`: esp_err_t error code
- `buf`: buffer where the error message should be written
- `buflen`: Size of buffer buf. At most buflen bytes are written into the buf buffer (including the terminating null byte).

Macros

`ESP_OK`
esp_err_t value indicating success (no error)

`ESP_FAIL`
Generic esp_err_t code indicating failure

`ESP_ERR_NO_MEM`
Out of memory

`ESP_ERR_INVALID_ARG`
Invalid argument

`ESP_ERR_INVALID_STATE`
Invalid state

`ESP_ERR_INVALID_SIZE`
Invalid size

`ESP_ERR_NOT_FOUND`
Requested resource not found

`ESP_ERR_NOT_SUPPORTED`
Operation or feature not supported

`ESP_ERR_TIMEOUT`
Operation timed out

`ESP_ERR_INVALID_RESPONSE`
Received response was invalid

ESP_ERR_INVALID_CRC
: CRC or checksum was invalid

ESP_ERR_INVALID_VERSION
: Version was invalid

ESP_ERR_INVALID_MAC
: MAC address was invalid

ESP_ERR_WIFI_BASE
: Starting number of WiFi error codes

ESP_ERR_MESH_BASE
: Starting number of MESH error codes

ESP_ERR_FLASH_BASE
: Starting number of flash error codes

ESP_ERROR_CHECK(x)
: Macro which can be used to check the error code, and terminate the program in case the code is not ESP_OK. Prints the error code, error location, and the failed statement to serial output.

  Disabled if assertions are disabled.

ESP_ERROR_CHECK_WITHOUT_ABORT(x)
: Macro which can be used to check the error code. Prints the error code, error location, and the failed statement to serial output. In comparison with ESP_ERROR_CHECK(), this prints the same error message but isn't terminating the program.

Type Definitions

typedef int32_t **esp_err_t**

2.7.5 ESP HTTPS OTA

Overview

esp_https_ota provides simplified APIs to perform firmware upgrades over HTTPS. It's an abstraction layer over existing OTA APIs.

Application Example

```
esp_err_t do_firmware_upgrade()
{
    esp_http_client_config_t config = {
```

(下页继续)

(续上页)

```
        .url = CONFIG_FIRMWARE_UPGRADE_URL,
        .cert_pem = (char *)server_cert_pem_start,
    };
    esp_err_t ret = esp_https_ota(&config);
    if (ret == ESP_OK) {
        esp_restart();
    } else {
        return ESP_FAIL;
    }
    return ESP_OK;
}
```

Signature Verification

For additional security, signature of OTA firmware images can be verified. For that, refer *Secure OTA Updates Without Secure boot*

API Reference

Header File

- esp_https_ota/include/esp_https_ota.h

Functions

esp_err_t **esp_https_ota**(**const** *esp_http_client_config_t* **config*)

HTTPS OTA Firmware upgrade.

This function allocates HTTPS OTA Firmware upgrade context, establishes HTTPS connection, reads image data from HTTP stream and writes it to OTA partition and finishes HTTPS OTA Firmware upgrade operation. This API supports URL redirection, but if CA cert of URLs differ then it should be appended to `cert_pem` member of `config`.

Note This API handles the entire OTA operation, so if this API is being used then no other APIs from `esp_https_ota` component should be called. If more information and control is needed during the HTTPS OTA process, then one can use `esp_https_ota_begin` and subsequent APIs. If this API returns successfully, esp_restart() must be called to boot from the new firmware image.

Return

- ESP_OK: OTA data updated, next reboot will use specified partition.

- ESP_FAIL: For generic failure.
- ESP_ERR_INVALID_ARG: Invalid argument
- ESP_ERR_OTA_VALIDATE_FAILED: Invalid app image
- ESP_ERR_NO_MEM: Cannot allocate memory for OTA operation.
- ESP_ERR_FLASH_OP_TIMEOUT or ESP_ERR_FLASH_OP_FAIL: Flash write failed.
- For other return codes, refer OTA documentation in esp-idf's app_update component.

Parameters

- `config`: pointer to *esp_http_client_config_t* structure.

esp_err_t **esp_https_ota_begin**(*esp_https_ota_config_t* **ota_config*, *esp_https_ota_handle_t* **handle*)

Start HTTPS OTA Firmware upgrade.

This function initializes ESP HTTPS OTA context and establishes HTTPS connection. This function must be invoked first. If this function returns successfully, then `esp_https_ota_perform` should be called to continue with the OTA process and there should be a call to `esp_https_ota_finish` on completion of OTA operation or on failure in subsequent operations. This API supports URL redirection, but if CA cert of URLs differ then it should be appended to `cert_pem` member of `http_config`, which is a part of `ota_config`. In case of error, this API explicitly sets `handle` to NULL.

Note This API is blocking, so setting `is_async` member of `http_config` structure will result in an error.

Return

- ESP_OK: HTTPS OTA Firmware upgrade context initialised and HTTPS connection established
- ESP_FAIL: For generic failure.
- ESP_ERR_INVALID_ARG: Invalid argument (missing/incorrect config, certificate, etc.)
- For other return codes, refer documentation in app_update component and esp_http_client component in esp-idf.

Parameters

- `ota_config`: pointer to *esp_https_ota_config_t* structure
- `handle`: pointer to an allocated data of type `esp_https_ota_handle_t` which will be initialised in this function

esp_err_t **esp_https_ota_perform**(*esp_https_ota_handle_t https_ota_handle*)

Read image data from HTTP stream and write it to OTA partition.

This function reads image data from HTTP stream and writes it to OTA partition. This function must be called only if esp_https_ota_begin() returns successfully. This function must be called in a loop since it returns after every HTTP read operation thus giving you the flexibility to stop OTA operation midway.

Return

- ESP_ERR_HTTPS_OTA_IN_PROGRESS: OTA update is in progress, call this API again to continue.
- ESP_OK: OTA update was successful
- ESP_FAIL: OTA update failed
- ESP_ERR_INVALID_ARG: Invalid argument
- ESP_ERR_OTA_VALIDATE_FAILED: Invalid app image
- ESP_ERR_NO_MEM: Cannot allocate memory for OTA operation.
- ESP_ERR_FLASH_OP_TIMEOUT or ESP_ERR_FLASH_OP_FAIL: Flash write failed.
- For other return codes, refer OTA documentation in esp-idf's app_update component.

Parameters

- `https_ota_handle`: pointer to esp_https_ota_handle_t structure

bool **esp_https_ota_is_complete_data_received**(*esp_https_ota_handle_t https_ota_handle*)

Checks if complete data was received or not.

Note This API can be called just before esp_https_ota_finish() to validate if the complete image was indeed received.

Return

- false
- true

Parameters

- `https_ota_handle`: pointer to esp_https_ota_handle_t structure

esp_err_t **esp_https_ota_finish**(*esp_https_ota_handle_t https_ota_handle*)

Clean-up HTTPS OTA Firmware upgrade and close HTTPS connection.

This function closes the HTTP connection and frees the ESP HTTPS OTA context. This function switches the boot partition to the OTA partition containing the new firmware image.

Note If this API returns successfully, esp_restart() must be called to boot from the new firmware image

Return

- ESP_OK: Clean-up successful
- ESP_ERR_INVALID_STATE
- ESP_ERR_INVALID_ARG: Invalid argument
- ESP_ERR_OTA_VALIDATE_FAILED: Invalid app image

Parameters

- `https_ota_handle`: pointer to esp_https_ota_handle_t structure

esp_err_t **esp_https_ota_get_img_desc**(*esp_https_ota_handle_t https_ota_handle*, *esp_app_desc_t *new_app_info*)

Reads app description from image header. The app description provides information like the "Firmware version" of the image.

Note This API can be called only after esp_https_ota_begin() and before esp_https_ota_perform(). Calling this API is not mandatory.

Return

- ESP_ERR_INVALID_ARG: Invalid arguments
- ESP_FAIL: Failed to read image descriptor
- ESP_OK: Successfully read image descriptor

Parameters

- `https_ota_handle`: pointer to esp_https_ota_handle_t structure
- `new_app_info`: pointer to an allocated *esp_app_desc_t* structure

int **esp_https_ota_get_image_len_read**(*esp_https_ota_handle_t https_ota_handle*)

Structures

struct esp_https_ota_config_t

ESP HTTPS OTA configuration.

Public Members

const *esp_http_client_config_t* ***http_config**

ESP HTTP client configuration

Macros

ESP_ERR_HTTPS_OTA_BASE

ESP_ERR_HTTPS_OTA_IN_PROGRESS

Type Definitions

typedef void *esp_https_ota_handle_t

2.7.6 ESP-pthread

Overview

This module offers Espressif specific extensions to the pthread library that can be used to influence the beh

- Stack size of the pthreads
- Priority of the created pthreads
- Inheriting this configuration across threads
- Thread name
- Core affinity / core pinning.

Example to tune the stack size of the pthread:

```
void * thread_func(void * p)
{
    printf("In thread_func\n");
    return NULL;
}

void app_main(void)
{
    pthread_t t1;

    esp_pthread_cfg_t cfg = esp_create_default_pthread_config();
    cfg.stack_size = (4 * 1024);
    esp_pthread_set_cfg(&cfg);

    pthread_create(&t1, NULL, thread_func);
}
```

The API can also be used for inheriting the settings across threads. For example:

```
void * my_thread2(void * p)
{
    /* This thread will inherit the stack size of 4K */
    printf("In my_thread2\n");

    return NULL;
}

void * my_thread1(void * p)
{
    printf("In my_thread1\n");
    pthread_t t2;
    pthread_create(&t2, NULL, my_thread2);

    return NULL;
}

void app_main(void)
{
    pthread_t t1;

    esp_pthread_cfg_t cfg = esp_create_default_pthread_config();
    cfg.stack_size = (4 * 1024);
    cfg.inherit_cfg = true;
    esp_pthread_set_cfg(&cfg);

    pthread_create(&t1, NULL, my_thread1);
}
```

API Reference

Header File

- pthread/include/esp_pthread.h

Functions

esp_pthread_cfg_t **esp_pthread_get_default_config**(void)

Creates a default pthread configuration based on the values set via menuconfig.

Return A default configuration structure.

esp_err_t **esp_pthread_set_cfg**(**const** *esp_pthread_cfg_t* **cfg*)

Configure parameters for creating pthread.

This API allows you to configure how the subsequent pthread_create() call will behave. This call can be used to setup configuration parameters like stack size, priority, configuration inheritance etc.

If the 'inherit' flag in the configuration structure is enabled, then the same configuration is also inherited in the thread subtree.

Note Passing non-NULL attributes to pthread_create() will override the stack_size parameter set using this API

Return

- ESP_OK if configuration was successfully set
- ESP_ERR_NO_MEM if out of memory
- ESP_ERR_INVALID_ARG if stack_size is less than PTHREAD_STACK_MIN

Parameters

- `cfg`: The pthread config parameters

esp_err_t **esp_pthread_get_cfg**(*esp_pthread_cfg_t* **p*)

Get current pthread creation configuration.

This will retrieve the current configuration that will be used for creating threads.

Return

- ESP_OK if the configuration was available
- ESP_ERR_NOT_FOUND if a configuration wasn't previously set

Parameters

- `p`: Pointer to the pthread config structure that will be updated with the currently configured parameters

Structures

struct esp_pthread_cfg_t

pthread configuration structure that influences pthread creation

Public Members

size_t **stack_size**
The stack size of the pthread.

size_t **prio**
The thread' s priority.

bool **inherit_cfg**
Inherit this configuration further.

const char ***thread_name**
The thread name.

int **pin_to_core**
The core id to pin the thread to. Has the same value range as xCoreId argument of xTaskCreatePinnedToCore.

Macros

`PTHREAD_STACK_MIN`

2.7.7 Event Loop Library

Overview

The event loop library allows components to declare events to which other components can register handlers – code which will execute when those events occur. This allows loosely coupled components to attach desired behavior to changes in state of other components without application involvement. For instance, a high level connection handling library may subscribe to events produced by the wifi subsystem directly and act on those events. This also simplifies event processing by serializing and deferring code execution to another context.

Using `esp_event` APIs

There are two objects of concern for users of this library: events and event loops.

Events are occurrences of note. For example, for WiFi, a successful connection to the access point may be an event. Events are referenced using a two part identifier which are discussed more *here*. Event loops are the vehicle by which events get posted by event sources and handled by event handler functions. These two appear prominently in the event loop library APIs.

Using this library roughly entails the following flow:

1. A user defines a function that should run when an event is posted to a loop. This function is referred to as the event handler. It should have the same signature as *esp_event_handler_t*.

2. An event loop is created using *esp_event_loop_create()*, which outputs a handle to the loop of type *esp_event_loop_handle_t*. Event loops created using this API are referred to as user event loops. There is, however, a special type of event loop called the default event loop which are discussed *here*.
3. Components register event handlers to the loop using *esp_event_handler_register_with()*. Handlers can be registered with multiple loops, more on that *here*.
4. Event sources post an event to the loop using *esp_event_post_to()*.
5. Components wanting to remove their handlers from being called can do so by unregistering from the loop using *esp_event_handler_unregister_with()*.
6. Event loops which are no longer needed can be deleted using *esp_event_loop_delete()*.

In code, the flow above may look like as follows:

```
// 1. Define the event handler
void run_on_event(void* handler_arg, esp_event_base_t base, int32_t id, void* event_data)
{
    // Event handler logic
}

void app_main()
{
    // 2. A configuration structure of type esp_event_loop_args_t is needed to specify␣
↪the properties of the loop to be
    // created. A handle of type esp_event_loop_handle_t is obtained, which is needed by␣
↪the other APIs to reference the loop
    // to perform their operations on.
    esp_event_loop_args_t loop_args = {
        .queue_size = ...,
        .task_name = ...
        .task_priority = ...,
        .task_stack_size = ...,
        .task_core_id = ...
    };

    esp_event_loop_handle_t loop_handle;

    esp_event_loop_create(&loop_args, &loop_handle);

    // 3. Register event handler defined in (1). MY_EVENT_BASE and MY_EVENT_ID specifies␣
↪a hypothetical
    // event that handler run_on_event should execute on when it gets posted to the loop.
```

(下页继续)

(续上页)

```
    esp_event_handler_register_with(loop_handle, MY_EVENT_BASE, MY_EVENT_ID, run_on_
↪event, ...);

    ...

    // 4. Post events to the loop. This queues the event on the event loop. At some␣
↪point in time
    // the event loop executes the event handler registered to the posted event, in this␣
↪case run_on_event.
    // For simplicity sake this example calls esp_event_post_to from app_main, but␣
↪posting can be done from
    // any other tasks (which is the more interesting use case).
    esp_event_post_to(loop_handle, MY_EVENT_BASE, MY_EVENT_ID, ...);

    ...

    // 5. Unregistering an unneeded handler
    esp_event_handler_unregister_with(loop_handle, MY_EVENT_BASE, MY_EVENT_ID, run_on_
↪event);

    ...

    // 6. Deleting an unneeded event loop
    esp_event_loop_delete(loop_handle);
}
```

Declaring and defining events

As mentioned previously, events consists of two-part identifers: the event base and the event ID. The event base identifies an independent group of events; the event ID identifies the event within that group. Think of the event base and event ID as a person' s last name and first name, respectively. A last name identifies a family, and the first name identifies a person within that family.

The event loop library provides macros to declare and define the event base easily.

Event base declaration:

```
ESP_EVENT_DECLARE_BASE(EVENT_BASE)
```

Event base definition:

```
ESP_EVENT_DEFINE_BASE(EVENT_BASE)
```

注解: In IDF, the base identifiers for system events are uppercase and are postfixed with `_EVENT`. For example, the base for wifi events is declared and defined as `WIFI_EVENT`, the ethernet event base `ETHERNET_EVENT`, and so on. The purpose is to have event bases look like constants (although they are global variables considering the defintions of macros `ESP_EVENT_DECLARE_BASE` and `ESP_EVENT_DEFINE_BASE`).

For event ID' s, declaring them as enumerations is recommended. Once again, for visibility, these are typically placed in public header files.

Event ID:

```
enum {
    EVENT_ID_1,
    EVENT_ID_2,
    EVENT_ID_3,
    ...
}
```

Default Event Loop

The default event loop is a special type of loop used for system events (WiFi events, for example). The handle for this loop is hidden from the user. The creation, deletion, handler registration/unregistration and posting of events is done through a variant of the APIs for user event loops. The table below enumerates those variants, and the user event loops equivalent.

User Event Loops	Default Event Loops
esp_event_loop_create()	*esp_event_loop_create_default()*
esp_event_loop_delete()	*esp_event_loop_delete_default()*
esp_event_handler_register_with()	*esp_event_handler_register()*
esp_event_handler_unregister_with()	*esp_event_handler_unregister()*
esp_event_post_to()	*esp_event_post()*

If you compare the signatures for both, they are mostly similar except the for the lack of loop handle specification for the default event loop APIs.

Other than the API difference and the special designation to which system events are posted to, there is no difference to how default event loops and user event loops behave. It is even possible for users to post their own events to the default event loop, should the user opt to not create their own loops to save memory.

Notes on Handler Registration

It is possible to register a single handler to multiple events individually, i.e. using multiple calls to *esp_event_handler_register_with()*. For those multiple calls, the specific event base and event ID can be specified with which the handler should execute.

However, in some cases it is desirable for a handler to execute on (1) all events that get posted to a loop or (2) all events of a particular base identifier. This is possible using the special event base identifier `ESP_EVENT_ANY_BASE` and special event ID `ESP_EVENT_ANY_ID`. These special identifiers may be passed as the event base and event ID arguments for *esp_event_handler_register_with()*.

Therefore, the valid arguments to *esp_event_handler_register_with()* are:

1. <event base>, <event ID> - handler executes when the event with base <event base> and event ID <event ID> gets posted to the loop
2. <event base>, ESP_EVENT_ANY_ID - handler executes when any event with base <event base> gets posted to the loop
3. ESP_EVENT_ANY_BASE, ESP_EVENT_ANY_ID - handler executes when any event gets posted to the loop

As an example, suppose the following handler registrations were performed:

```
esp_event_handler_register_with(loop_handle, MY_EVENT_BASE, MY_EVENT_ID, run_on_event_1,␣
↪...);
esp_event_handler_register_with(loop_handle, MY_EVENT_BASE, ESP_EVENT_ANY_ID, run_on_
↪event_2, ...);
esp_event_handler_register_with(loop_handle, ESP_EVENT_ANY_BASE, ESP_EVENT_ANY_ID, run_
↪on_event_3, ...);
```

If the hypothetical event `MY_EVENT_BASE`, `MY_EVENT_ID` is posted, all three handlers `run_on_event_1`, `run_on_event_2`, and `run_on_event_3` would execute.

If the hypothetical event `MY_EVENT_BASE`, `MY_OTHER_EVENT_ID` is posted, only `run_on_event_2` and `run_on_event_3` would execute.

If the hypothetical event `MY_OTHER_EVENT_BASE`, `MY_OTHER_EVENT_ID` is posted, only `run_on_event_3` would execute.

Handler Registration and Handler Dispatch Order

The general rule is that for handlers that match a certain posted event during dispatch, those which are registered first also gets executed first. The user can then control which handlers get executed first by registering them before other handlers, provided that all registrations are performed using a single task. If the user plans to take advantage of this behavior, caution must be exercised if there are multiple tasks registering handlers. While the 'first registered, first executed' behavior still holds true, the task which

gets executed first will also get their handlers registered first. Handlers registered one after the other by a single task will still be dispatched in the order relative to each other, but if that task gets pre-empted in between registration by another task which also registers handlers; then during dispatch those handlers will also get executed in between.

Event loop profiling

A configuration option *CONFIG_ESP_EVENT_LOOP_PROFILING* can be enabled in order to activate statistics collection for all event loops created. The function *esp_event_dump()* can be used to output the collected statistics to a file stream. More details on the information included in the dump can be found in the *esp_event_dump()* API Reference.

Application Example

Examples on using the `esp_event` library can be found in system/esp_event. The examples cover event declaration, loop creation, handler registration and unregistration and event posting.

Other examples which also adopt esp_event library:

- NMEA Parser , which will decode the statements received from GPS.

API Reference

Header File

- esp_event/include/esp_event.h

Functions

esp_err_t **esp_event_loop_create**(const *esp_event_loop_args_t* **event_loop_args*, *esp_event_loop_handle_t* **event_loop*)

Create a new event loop.

Return

- ESP_OK: Success
- ESP_ERR_NO_MEM: Cannot allocate memory for event loops list
- ESP_FAIL: Failed to create task loop
- Others: Fail

Parameters

- `event_loop_args`: configuration structure for the event loop to create

- `event_loop`: handle to the created event loop

esp_err_t **esp_event_loop_delete**(*esp_event_loop_handle_t event_loop*)

Delete an existing event loop.

Return

- ESP_OK: Success
- Others: Fail

Parameters

- `event_loop`: event loop to delete

esp_err_t **esp_event_loop_create_default**(void)

Create default event loop.

Return

- ESP_OK: Success
- ESP_ERR_NO_MEM: Cannot allocate memory for event loops list
- ESP_FAIL: Failed to create task loop
- Others: Fail

esp_err_t **esp_event_loop_delete_default**(void)

Delete the default event loop.

Return

- ESP_OK: Success
- Others: Fail

esp_err_t **esp_event_loop_run**(*esp_event_loop_handle_t event_loop*, TickType_t *ticks_to_run*)

Dispatch events posted to an event loop.

This function is used to dispatch events posted to a loop with no dedicated task, i.e task name was set to NULL in event_loop_args argument during loop creation. This function includes an argument to limit the amount of time it runs, returning control to the caller when that time expires (or some time afterwards). There is no guarantee that a call to this function will exit at exactly the time of expiry. There is also no guarantee that events have been dispatched during the call, as the function might have spent all of the alloted time waiting on the event queue. Once an event has been unqueued, however, it is guaranteed to be dispatched. This guarantee contributes to not being able to exit exactly at time of expiry as (1) blocking on internal mutexes is necessary for dispatching the unqueued event, and (2) during dispatch of the unqueued event there is no way to control the time occupied by handler code

execution. The guaranteed time of exit is therefore the alloted time + amount of time required to dispatch the last unqueued event.

In cases where waiting on the queue times out, ESP_OK is returned and not ESP_ERR_TIMEOUT, since it is normal behavior.

Note encountering an unknown event that has been posted to the loop will only generate a warning, not an error.

Return

- ESP_OK: Success
- Others: Fail

Parameters

- `event_loop`: event loop to dispatch posted events from
- `ticks_to_run`: number of ticks to run the loop

esp_err_t **esp_event_handler_register**(*esp_event_base_t event_base*, int32_t *event_id*, *esp_event_handler_t event_handler*, void **event_handler_arg*)

Register an event handler to the system event loop.

This function can be used to register a handler for either: (1) specific events, (2) all events of a certain event base, or (3) all events known by the system event loop.

- specific events: specify exact event_base and event_id
- all events of a certain base: specify exact event_base and use ESP_EVENT_ANY_ID as the event_id
- all events known by the loop: use ESP_EVENT_ANY_BASE for event_base and ESP_EVENT_ANY_ID as the event_id

Registering multiple handlers to events is possible. Registering a single handler to multiple events is also possible. However, registering the same handler to the same event multiple times would cause the previous registrations to be overwritten.

Note the event loop library does not maintain a copy of event_handler_arg, therefore the user should ensure that event_handler_arg still points to a valid location by the time the handler gets called

Return

- ESP_OK: Success
- ESP_ERR_NO_MEM: Cannot allocate memory for the handler
- ESP_ERR_INVALID_ARG: Invalid combination of event base and event id

- Others: Fail

Parameters

- `event_base`: the base id of the event to register the handler for
- `event_id`: the id of the event to register the handler for
- `event_handler`: the handler function which gets called when the event is dispatched
- `event_handler_arg`: data, aside from event data, that is passed to the handler when it is called

esp_err_t **esp_event_handler_register_with**(*esp_event_loop_handle_t event_loop*, *esp_event_base_t event_base*, int32_t *event_id*, *esp_event_handler_t event_handler*, void **event_handler_arg*)

Register an event handler to a specific loop.

This function behaves in the same manner as esp_event_handler_register, except the additional specification of the event loop to register the handler to.

Note the event loop library does not maintain a copy of event_handler_arg, therefore the user should ensure that event_handler_arg still points to a valid location by the time the handler gets called

Return

- ESP_OK: Success
- ESP_ERR_NO_MEM: Cannot allocate memory for the handler
- ESP_ERR_INVALID_ARG: Invalid combination of event base and event id
- Others: Fail

Parameters

- `event_loop`: the event loop to register this handler function to
- `event_base`: the base id of the event to register the handler for
- `event_id`: the id of the event to register the handler for
- `event_handler`: the handler function which gets called when the event is dispatched
- `event_handler_arg`: data, aside from event data, that is passed to the handler when it is called

esp_err_t **esp_event_handler_unregister**(*esp_event_base_t event_base*, int32_t *event_id*, *esp_event_handler_t event_handler*)

Unregister a handler with the system event loop.

This function can be used to unregister a handler so that it no longer gets called during dispatch. Handlers can be unregistered for either: (1) specific events, (2) all events of a certain event base, or (3) all events known by the system event loop

- specific events: specify exact event_base and event_id
- all events of a certain base: specify exact event_base and use ESP_EVENT_ANY_ID as the event_id
- all events known by the loop: use ESP_EVENT_ANY_BASE for event_base and ESP_EVENT_ANY_ID as the event_id

This function ignores unregistration of handlers that has not been previously registered.

Return ESP_OK success

Return ESP_ERR_INVALID_ARG invalid combination of event base and event id

Return others fail

Parameters

- `event_base`: the base of the event with which to unregister the handler
- `event_id`: the id of the event with which to unregister the handler
- `event_handler`: the handler to unregister

esp_err_t **esp_event_handler_unregister_with**(*esp_event_loop_handle_t* *event_loop*, *esp_event_base_t* *event_base*, int32_t *event_id*, *esp_event_handler_t* *event_handler*)

Unregister a handler with the system event loop.

This function behaves in the same manner as esp_event_handler_unregister, except the additional specification of the event loop to unregister the handler with.

Return

- ESP_OK: Success
- ESP_ERR_INVALID_ARG: Invalid combination of event base and event id
- Others: Fail

Parameters

- `event_loop`: the event loop with which to unregister this handler function
- `event_base`: the base of the event with which to unregister the handler
- `event_id`: the id of the event with which to unregister the handler
- `event_handler`: the handler to unregister

esp_err_t **esp_event_post**(*esp_event_base_t event_base*, int32_t *event_id*, void **event_data*, size_t *event_data_size*, TickType_t *ticks_to_wait*)

Posts an event to the system default event loop. The event loop library keeps a copy of event_data and manages the copy' s lifetime automatically (allocation + deletion); this ensures that the data the handler recieves is always valid.

Return

- ESP_OK: Success
- ESP_ERR_TIMEOUT: Time to wait for event queue to unblock expired, queue full when posting from ISR
- ESP_ERR_INVALID_ARG: Invalid combination of event base and event id
- Others: Fail

Parameters

- `event_base`: the event base that identifies the event
- `event_id`: the event id that identifies the event
- `event_data`: the data, specific to the event occurence, that gets passed to the handler
- `event_data_size`: the size of the event data
- `ticks_to_wait`: number of ticks to block on a full event queue

esp_err_t **esp_event_post_to**(*esp_event_loop_handle_t event_loop*, *esp_event_base_t event_base*, int32_t *event_id*, void **event_data*, size_t *event_data_size*, TickType_t *ticks_to_wait*)

Posts an event to the specified event loop. The event loop library keeps a copy of event_data and manages the copy' s lifetime automatically (allocation + deletion); this ensures that the data the handler recieves is always valid.

This function behaves in the same manner as esp_event_post_to, except the additional specification of the event loop to post the event to.

Return

- ESP_OK: Success
- ESP_ERR_TIMEOUT: Time to wait for event queue to unblock expired, queue full when posting from ISR
- ESP_ERR_INVALID_ARG: Invalid combination of event base and event id
- Others: Fail

Parameters

- `event_loop`: the event loop to post to

- `event_base`: the event base that identifies the event
- `event_id`: the event id that identifies the event
- `event_data`: the data, specific to the event occurence, that gets passed to the handler
- `event_data_size`: the size of the event data
- `ticks_to_wait`: number of ticks to block on a full event queue

esp_err_t **esp_event_dump**(FILE **file*)

Dumps statistics of all event loops.

Dumps event loop info in the format:

```
     event loop
         handler
         handler
         ...
     event loop
         handler
         handler
         ...

 where:

  event loop
     format: address,name rx:total_recieved dr:total_dropped
     where:
         address - memory address of the event loop
         name - name of the event loop, 'none' if no dedicated task
         total_recieved - number of successfully posted events
         total_dropped - number of events unsuccessfully posted due to queue being␣
↪full

  handler
     format: address ev:base,id inv:total_invoked run:total_runtime
     where:
         address - address of the handler function
         base,id - the event specified by event base and id this handler executes
         total_invoked - number of times this handler has been invoked
         total_runtime - total amount of time used for invoking this handler
```

Note this function is a noop when CONFIG_ESP_EVENT_LOOP_PROFILING is disabled

Return

- ESP_OK: Success
- ESP_ERR_NO_MEM: Cannot allocate memory for event loops list
- Others: Fail

Parameters

- `file`: the file stream to output to

Structures

`struct esp_event_loop_args_t`

Configuration for creating event loops.

Public Members

int32_t **queue_size**

size of the event loop queue

const char ***task_name**

name of the event loop task; if NULL, a dedicated task is not created for event loop

UBaseType_t **task_priority**

priority of the event loop task, ignored if task name is NULL

uint32_t **task_stack_size**

stack size of the event loop task, ignored if task name is NULL

BaseType_t **task_core_id**

core to which the event loop task is pinned to, ignored if task name is NULL

Header File

- esp_event/include/esp_event_base.h

Macros

`ESP_EVENT_DECLARE_BASE`(id)

`ESP_EVENT_DEFINE_BASE`(id)

`ESP_EVENT_ANY_BASE`

register handler for any event base

`ESP_EVENT_ANY_ID`

register handler for any event id

Type Definitions

typedef const char ***esp_event_base_t**

unique pointer to a subsystem that exposes events

typedef void ***esp_event_loop_handle_t**

a number that identifies an event with respect to a base

typedef void (***esp_event_handler_t**)(void *event_handler_arg, *esp_event_base_t* event_base, int32_t event_id, void *event_data)

function called when an event is posted to the queue

Related Documents

Legacy event loop

API Reference

Header File

- esp_event/include/esp_event_legacy.h

Functions

esp_err_t **esp_event_send**(*system_event_t *event*)

Send a event to event task.

Other task/modules, such as the tcpip_adapter, can call this API to send an event to event task

Note This API is part of the legacy event system. New code should use event library API in esp_event.h

Return ESP_OK : succeed

Return others : fail

Parameters

- `event`: Event to send

esp_err_t **esp_event_send_internal**(*esp_event_base_t event_base*, int32_t *event_id*, void **event_data*, size_t *event_data_size*, TickType_t *ticks_to_wait*)

Send a event to event task.

Other task/modules, such as the tcpip_adapter, can call this API to send an event to event task

Note This API is used by WiFi Driver only.

Return ESP_OK : succeed

Return others : fail

Parameters

- `event_base`: the event base that identifies the event
- `event_id`: the event id that identifies the event
- `event_data`: the data, specific to the event occurence, that gets passed to the handler
- `event_data_size`: the size of the event data
- `ticks_to_wait`: number of ticks to block on a full event queue

esp_err_t **esp_event_process_default**(*system_event_t* **event*)

Default event handler for system events.

This function performs default handling of system events. When using esp_event_loop APIs, it is called automatically before invoking the user-provided callback function.

Note This API is part of the legacy event system. New code should use event library API in esp_event.h

Applications which implement a custom event loop must call this function as part of event processing.

Return ESP_OK if an event was handled successfully

Parameters

- `event`: pointer to event to be handled

void **esp_event_set_default_eth_handlers**(void)

Install default event handlers for Ethernet interface.

Note This API is part of the legacy event system. New code should use event library API in esp_event.h

void **esp_event_set_default_wifi_handlers**(void)

Install default event handlers for Wi-Fi interfaces (station and AP)

Note This API is part of the legacy event system. New code should use event library API in esp_event.h

esp_err_t **esp_event_loop_init**(*system_event_cb_t* *cb*, void **ctx*)

Initialize event loop.

Create the event handler and task

Note This API is part of the legacy event system. New code should use event library API in esp_event.h

Return

- ESP_OK: succeed
- others: fail

Parameters

- `cb`: application specified event callback, it can be modified by call esp_event_set_cb
- `ctx`: reserved for user

system_event_cb_t **esp_event_loop_set_cb**(*system_event_cb_t cb*, void **ctx*)

Set application specified event callback function.

Note This API is part of the legacy event system. New code should use event library API in esp_event.h

Attention 1. If cb is NULL, means application don't need to handle If cb is not NULL, it will be call when an event is received, after the default event callback is completed

Return old callback

Parameters

- `cb`: application callback function
- `ctx`: argument to be passed to callback

Unions

union system_event_info_t

#include <esp_event_legacy.h> Union of all possible system_event argument structures

Public Members

system_event_sta_connected_t **connected**

ESP32 station connected to AP

system_event_sta_disconnected_t **disconnected**

ESP32 station disconnected to AP

system_event_sta_scan_done_t **scan_done**

ESP32 station scan (APs) done

system_event_sta_authmode_change_t **auth_change**

the auth mode of AP ESP32 station connected to changed

system_event_sta_got_ip_t **got_ip**

ESP32 station got IP, first time got IP or when IP is changed

system_event_sta_wps_er_pin_t **sta_er_pin**
ESP32 station WPS enrollee mode PIN code received

system_event_sta_wps_fail_reason_t **sta_er_fail_reason**
ESP32 station WPS enrollee mode failed reason code received

system_event_ap_staconnected_t **sta_connected**
a station connected to ESP32 soft-AP

system_event_ap_stadisconnected_t **sta_disconnected**
a station disconnected to ESP32 soft-AP

system_event_ap_probe_req_rx_t **ap_probereqrecved**
ESP32 soft-AP receive probe request packet

system_event_ap_staipassigned_t **ap_staipassigned**
ESP32 soft-AP assign an IP to the station

system_event_got_ip6_t **got_ip6**
ESP32 station or ap or ethernet ipv6 addr state change to preferred

Structures

struct system_event_t
Event, as a tagged enum

Public Members

system_event_id_t **event_id**
event ID

system_event_info_t **event_info**
event information

Macros

SYSTEM_EVENT_AP_STA_GOT_IP6

Type Definitions

typedef *wifi_event_sta_wps_fail_reason_t* **system_event_sta_wps_fail_reason_t**
Argument structure of SYSTEM_EVENT_STA_WPS_ER_FAILED event

typedef *wifi_event_sta_scan_done_t* **system_event_sta_scan_done_t**
Argument structure of SYSTEM_EVENT_SCAN_DONE event

typedef *wifi_event_sta_connected_t* **system_event_sta_connected_t**
Argument structure of SYSTEM_EVENT_STA_CONNECTED event

typedef *wifi_event_sta_disconnected_t* **system_event_sta_disconnected_t**
Argument structure of SYSTEM_EVENT_STA_DISCONNECTED event

typedef *wifi_event_sta_authmode_change_t* **system_event_sta_authmode_change_t**
Argument structure of SYSTEM_EVENT_STA_AUTHMODE_CHANGE event

typedef *wifi_event_sta_wps_er_pin_t* **system_event_sta_wps_er_pin_t**
Argument structure of SYSTEM_EVENT_STA_WPS_ER_PIN event

typedef *wifi_event_ap_staconnected_t* **system_event_ap_staconnected_t**
Argument structure of event

typedef *wifi_event_ap_stadisconnected_t* **system_event_ap_stadisconnected_t**
Argument structure of event

typedef *wifi_event_ap_probe_req_rx_t* **system_event_ap_probe_req_rx_t**
Argument structure of event

typedef ip_event_ap_staipassigned_t **system_event_ap_staipassigned_t**
Argument structure of event

typedef ip_event_got_ip_t **system_event_sta_got_ip_t**
Argument structure of event

typedef ip_event_got_ip6_t **system_event_got_ip6_t**
Argument structure of event

typedef *esp_err_t* (***system_event_handler_t**)(*esp_event_base_t* event_base, int32_t event_id, void *event_data, size_t event_data_size, TickType_t ticks_to_wait)
Event handler function type

typedef *esp_err_t* (***system_event_cb_t**)(void *ctx, *system_event_t* *event)
Application specified event callback function.

Note This API is part of the legacy event system. New code should use event library API in esp_event.h

Return

- ESP_OK: succeed
- others: fail

Parameters

- `ctx`: reserved for user
- `event`: event type defined in this file

Enumerations

enum system_event_id_t

System event types enumeration

Values:

SYSTEM_EVENT_WIFI_READY = 0

ESP32 WiFi ready

SYSTEM_EVENT_SCAN_DONE

ESP32 finish scanning AP

SYSTEM_EVENT_STA_START

ESP32 station start

SYSTEM_EVENT_STA_STOP

ESP32 station stop

SYSTEM_EVENT_STA_CONNECTED

ESP32 station connected to AP

SYSTEM_EVENT_STA_DISCONNECTED

ESP32 station disconnected from AP

SYSTEM_EVENT_STA_AUTHMODE_CHANGE

the auth mode of AP connected by ESP32 station changed

SYSTEM_EVENT_STA_GOT_IP

ESP32 station got IP from connected AP

SYSTEM_EVENT_STA_LOST_IP

ESP32 station lost IP and the IP is reset to 0

SYSTEM_EVENT_STA_WPS_ER_SUCCESS

ESP32 station wps succeeds in enrollee mode

SYSTEM_EVENT_STA_WPS_ER_FAILED

ESP32 station wps fails in enrollee mode

SYSTEM_EVENT_STA_WPS_ER_TIMEOUT

ESP32 station wps timeout in enrollee mode

SYSTEM_EVENT_STA_WPS_ER_PIN

ESP32 station wps pin code in enrollee mode

SYSTEM_EVENT_STA_WPS_ER_PBC_OVERLAP

ESP32 station wps overlap in enrollee mode

SYSTEM_EVENT_AP_START

ESP32 soft-AP start

`SYSTEM_EVENT_AP_STOP`
ESP32 soft-AP stop

`SYSTEM_EVENT_AP_STACONNECTED`
a station connected to ESP32 soft-AP

`SYSTEM_EVENT_AP_STADISCONNECTED`
a station disconnected from ESP32 soft-AP

`SYSTEM_EVENT_AP_STAIPASSIGNED`
ESP32 soft-AP assign an IP to a connected station

`SYSTEM_EVENT_AP_PROBEREQRECVED`
Receive probe request packet in soft-AP interface

`SYSTEM_EVENT_GOT_IP6`
ESP32 station or ap or ethernet interface v6IP addr is preferred

`SYSTEM_EVENT_ETH_START`
ESP32 ethernet start

`SYSTEM_EVENT_ETH_STOP`
ESP32 ethernet stop

`SYSTEM_EVENT_ETH_CONNECTED`
ESP32 ethernet phy link up

`SYSTEM_EVENT_ETH_DISCONNECTED`
ESP32 ethernet phy link down

`SYSTEM_EVENT_ETH_GOT_IP`
ESP32 ethernet got IP from connected AP

`SYSTEM_EVENT_MAX`
Number of members in this enum

2.7.8 FreeRTOS

Overview

This section contains documentation of FreeRTOS types, functions, and macros. It is automatically generated from FreeRTOS header files.

For more information about FreeRTOS features specific to ESP-IDF, see *ESP-IDF FreeRTOS SMP Changes* and *ESP-IDF FreeRTOS Additions*.

Task API

Header File

- freertos/include/freertos/task.h

Functions

BaseType_t **xTaskCreatePinnedToCore**(TaskFunction_t *pvTaskCode*, **const** char ***const** *pcName*, **const** uint32_t *usStackDepth*, void ***const** *pvParameters*, UBaseType_t *uxPriority*, *TaskHandle_t* ***const** *pvCreatedTask*, **const** BaseType_t *xCoreID*)

Create a new task with a specified affinity.

This function is similar to xTaskCreate, but allows setting task affinity in SMP system.

Return pdPASS if the task was successfully created and added to a ready list, otherwise an error code defined in the file projdefs.h

Parameters

- `pvTaskCode`: Pointer to the task entry function. Tasks must be implemented to never return (i.e. continuous loop).
- `pcName`: A descriptive name for the task. This is mainly used to facilitate debugging. Max length defined by configMAX_TASK_NAME_LEN - default is 16.
- `usStackDepth`: The size of the task stack specified as the number of bytes. Note that this differs from vanilla FreeRTOS.
- `pvParameters`: Pointer that will be used as the parameter for the task being created.
- `uxPriority`: The priority at which the task should run. Systems that include MPU support can optionally create tasks in a privileged (system) mode by setting bit portPRIVILEGE_BIT of the priority parameter. For example, to create a privileged task at priority 2 the uxPriority parameter should be set to (2 | portPRIVILEGE_BIT).
- `pvCreatedTask`: Used to pass back a handle by which the created task can be referenced.
- `xCoreID`: If the value is tskNO_AFFINITY, the created task is not pinned to any CPU, and the scheduler can run it on any core available. Values 0 or 1 indicate the index number of the CPU which the task should be pinned to. Specifying values larger than (portNUM_PROCESSORS - 1) will cause the function to fail.

static BaseType_t **xTaskCreate**(TaskFunction_t *pvTaskCode*, **const** char ***const** *pcName*, **const** uint32_t *usStackDepth*, void ***const** *pvParameters*, UBaseType_t *uxPriority*, *TaskHandle_t* ***const** *pvCreatedTask*)

Create a new task and add it to the list of tasks that are ready to run.

Internally, within the FreeRTOS implementation, tasks use two blocks of memory. The first block is used to hold the task' s data structures. The second block is used by the task as its stack. If a task

is created using xTaskCreate() then both blocks of memory are automatically dynamically allocated inside the xTaskCreate() function. (see http://www.freertos.org/a00111.html). If a task is created using xTaskCreateStatic() then the application writer must provide the required memory. xTaskCreateStatic() therefore allows a task to be created without using any dynamic memory allocation.

See xTaskCreateStatic() for a version that does not use any dynamic memory allocation.

xTaskCreate() can only be used to create a task that has unrestricted access to the entire microcontroller memory map. Systems that include MPU support can alternatively create an MPU constrained task using xTaskCreateRestricted().

Example usage:

```
// Task to be created.
void vTaskCode( void * pvParameters )
{
 for( ;; )
 {
     // Task code goes here.
 }
}

// Function that creates a task.
void vOtherFunction( void )
{
static uint8_t ucParameterToPass;
TaskHandle_t xHandle = NULL;

 // Create the task, storing the handle.  Note that the passed parameter␣
↪ucParameterToPass
 // must exist for the lifetime of the task, so in this case is declared static. ␣
↪If it was just an
 // an automatic stack variable it might no longer exist, or at least have been␣
↪corrupted, by the time
 // the new task attempts to access it.
 xTaskCreate( vTaskCode, "NAME", STACK_SIZE, &ucParameterToPass, tskIDLE_PRIORITY, &
↪xHandle );
    configASSERT( xHandle );

 // Use the handle to delete the task.
    if( xHandle != NULL )
    {
     vTaskDelete( xHandle );
```

(下页继续)

(续上页)

```
    }
}
```

Return pdPASS if the task was successfully created and added to a ready list, otherwise an error code defined in the file projdefs.h

Note If program uses thread local variables (ones specified with "__thread" keyword) then storage for them will be allocated on the task's stack.

Parameters

- `pvTaskCode`: Pointer to the task entry function. Tasks must be implemented to never return (i.e. continuous loop).
- `pcName`: A descriptive name for the task. This is mainly used to facilitate debugging. Max length defined by configMAX_TASK_NAME_LEN - default is 16.
- `usStackDepth`: The size of the task stack specified as the number of bytes. Note that this differs from vanilla FreeRTOS.
- `pvParameters`: Pointer that will be used as the parameter for the task being created.
- `uxPriority`: The priority at which the task should run. Systems that include MPU support can optionally create tasks in a privileged (system) mode by setting bit portPRIVILEGE_BIT of the priority parameter. For example, to create a privileged task at priority 2 the uxPriority parameter should be set to (2 | portPRIVILEGE_BIT).
- `pvCreatedTask`: Used to pass back a handle by which the created task can be referenced.

TaskHandle_t **xTaskCreateStaticPinnedToCore**(TaskFunction_t *pvTaskCode*, **const** char ***const** *pcName*, **const** uint32_t *ulStackDepth*, void ***const** *pvParameters*, UBaseType_t *uxPriority*, StackType_t ***const** *pxStackBuffer*, StaticTask_t ***const** *pxTaskBuffer*, **const** BaseType_t *xCoreID*)

Create a new task with a specified affinity.

This function is similar to xTaskCreateStatic, but allows specifying task affinity in an SMP system.

Return If neither pxStackBuffer or pxTaskBuffer are NULL, then the task will be created and a task handle will be returned by which the created task can be referenced. If either pxStackBuffer or pxTaskBuffer are NULL then the task will not be created and NULL is returned.

Parameters

- `pvTaskCode`: Pointer to the task entry function. Tasks must be implemented to never return (i.e. continuous loop).

- `pcName`: A descriptive name for the task. This is mainly used to facilitate debugging. The maximum length of the string is defined by configMAX_TASK_NAME_LEN in FreeRTOSConfig.h.
- `ulStackDepth`: The size of the task stack specified as the number of bytes. Note that this differs from vanilla FreeRTOS.
- `pvParameters`: Pointer that will be used as the parameter for the task being created.
- `uxPriority`: The priority at which the task will run.
- `pxStackBuffer`: Must point to a StackType_t array that has at least ulStackDepth indexes - the array will then be used as the task' s stack, removing the need for the stack to be allocated dynamically.
- `pxTaskBuffer`: Must point to a variable of type StaticTask_t, which will then be used to hold the task' s data structures, removing the need for the memory to be allocated dynamically.
- `xCoreID`: If the value is tskNO_AFFINITY, the created task is not pinned to any CPU, and the scheduler can run it on any core available. Values 0 or 1 indicate the index number of the CPU which the task should be pinned to. Specifying values larger than (portNUM_PROCESSORS - 1) will cause the function to fail.

static *TaskHandle_t* **xTaskCreateStatic**(TaskFunction_t *pvTaskCode*, **const** char ***const** *pcName*, **const** uint32_t *ulStackDepth*, void ***const** *pvParameters*, UBaseType_t *uxPriority*, StackType_t ***const** *pxStackBuffer*, StaticTask_t ***const** *pxTaskBuffer*)

Create a new task and add it to the list of tasks that are ready to run.

Internally, within the FreeRTOS implementation, tasks use two blocks of memory. The first block is used to hold the task' s data structures. The second block is used by the task as its stack. If a task is created using xTaskCreate() then both blocks of memory are automatically dynamically allocated inside the xTaskCreate() function. (see http://www.freertos.org/a00111.html). If a task is created using xTaskCreateStatic() then the application writer must provide the required memory. xTaskCreateStatic() therefore allows a task to be created without using any dynamic memory allocation.

Example usage:

```
// Dimensions the buffer that the task being created will use as its stack.
// NOTE:  This is the number of bytes the stack will hold, not the number of
// words as found in vanilla FreeRTOS.
#define STACK_SIZE 200

// Structure that will hold the TCB of the task being created.
StaticTask_t xTaskBuffer;

// Buffer that the task being created will use as its stack.  Note this is
```

(下页继续)

(续上页)

```
// an array of StackType_t variables.  The size of StackType_t is dependent on
// the RTOS port.
StackType_t xStack[ STACK_SIZE ];

// Function that implements the task being created.
void vTaskCode( void * pvParameters )
{
    // The parameter value is expected to be 1 as 1 is passed in the
    // pvParameters value in the call to xTaskCreateStatic().
    configASSERT( ( uint32_t ) pvParameters == 1UL );

    for( ;; )
    {
        // Task code goes here.
    }
}

// Function that creates a task.
void vOtherFunction( void )
{
    TaskHandle_t xHandle = NULL;

    // Create the task without using any dynamic memory allocation.
    xHandle = xTaskCreateStatic(
                  vTaskCode,       // Function that implements the task.
                  "NAME",          // Text name for the task.
                  STACK_SIZE,      // Stack size in bytes, not words.
                  ( void * ) 1,    // Parameter passed into the task.
                  tskIDLE_PRIORITY,// Priority at which the task is created.
                  xStack,          // Array to use as the task's stack.
                  &xTaskBuffer );  // Variable to hold the task's data structure.

    // puxStackBuffer and pxTaskBuffer were not NULL, so the task will have
    // been created, and xHandle will be the task's handle.  Use the handle
    // to suspend the task.
    vTaskSuspend( xHandle );
}
```

Return If neither pxStackBuffer or pxTaskBuffer are NULL, then the task will be created and a task handle will be returned by which the created task can be referenced. If either pxStackBuffer or

pxTaskBuffer are NULL then the task will not be created and NULL is returned.

Note If program uses thread local variables (ones specified with "__thread" keyword) then storage for them will be allocated on the task's stack.

Parameters

- `pvTaskCode`: Pointer to the task entry function. Tasks must be implemented to never return (i.e. continuous loop).
- `pcName`: A descriptive name for the task. This is mainly used to facilitate debugging. The maximum length of the string is defined by configMAX_TASK_NAME_LEN in FreeRTOSConfig.h.
- `ulStackDepth`: The size of the task stack specified as the number of bytes. Note that this differs from vanilla FreeRTOS.
- `pvParameters`: Pointer that will be used as the parameter for the task being created.
- `uxPriority`: The priority at which the task will run.
- `pxStackBuffer`: Must point to a StackType_t array that has at least ulStackDepth indexes - the array will then be used as the task's stack, removing the need for the stack to be allocated dynamically.
- `pxTaskBuffer`: Must point to a variable of type StaticTask_t, which will then be used to hold the task's data structures, removing the need for the memory to be allocated dynamically.

void **vTaskDelete**(*TaskHandle_t xTaskToDelete*)

Remove a task from the RTOS real time kernel's management.

The task being deleted will be removed from all ready, blocked, suspended and event lists.

INCLUDE_vTaskDelete must be defined as 1 for this function to be available. See the configuration section for more information.

See the demo application file death.c for sample code that utilises vTaskDelete ().

Note The idle task is responsible for freeing the kernel allocated memory from tasks that have been deleted. It is therefore important that the idle task is not starved of microcontroller processing time if your application makes any calls to vTaskDelete (). Memory allocated by the task code is not automatically freed, and should be freed before the task is deleted.

Example usage:

```
void vOtherFunction( void )
{
TaskHandle_t xHandle;

    // Create the task, storing the handle.
```

(下页继续)

(续上页)

```
    xTaskCreate( vTaskCode, "NAME", STACK_SIZE, NULL, tskIDLE_PRIORITY, &xHandle );

    // Use the handle to delete the task.
    vTaskDelete( xHandle );
}
```

Parameters

- `xTaskToDelete`: The handle of the task to be deleted. Passing NULL will cause the calling task to be deleted.

void **vTaskDelay**(**const** TickType_t *xTicksToDelay*)

Delay a task for a given number of ticks.

The actual time that the task remains blocked depends on the tick rate. The constant portTICK_PERIOD_MS can be used to calculate real time from the tick rate - with the resolution of one tick period.

INCLUDE_vTaskDelay must be defined as 1 for this function to be available. See the configuration section for more information.

vTaskDelay() specifies a time at which the task wishes to unblock relative to the time at which vTaskDelay() is called. For example, specifying a block period of 100 ticks will cause the task to unblock 100 ticks after vTaskDelay() is called. vTaskDelay() does not therefore provide a good method of controlling the frequency of a periodic task as the path taken through the code, as well as other task and interrupt activity, will effect the frequency at which vTaskDelay() gets called and therefore the time at which the task next executes. See vTaskDelayUntil() for an alternative API function designed to facilitate fixed frequency execution. It does this by specifying an absolute time (rather than a relative time) at which the calling task should unblock.

Example usage:

```
void vTaskFunction( void * pvParameters )
{
// Block for 500ms.
const TickType_t xDelay = 500 / portTICK_PERIOD_MS;

    for( ;; )
    {
        // Simply toggle the LED every 500ms, blocking between each toggle.
        vToggleLED();
        vTaskDelay( xDelay );
    }
```

(下页继续)

(续上页)

```
}
```

Parameters

- `xTicksToDelay`: The amount of time, in tick periods, that the calling task should block.

void **vTaskDelayUntil**(TickType_t *const *pxPreviousWakeTime*, const TickType_t *xTimeIncrement*)

Delay a task until a specified time.

INCLUDE_vTaskDelayUntil must be defined as 1 for this function to be available. See the configuration section for more information.

This function can be used by periodic tasks to ensure a constant execution frequency.

This function differs from vTaskDelay () in one important aspect: vTaskDelay () will cause a task to block for the specified number of ticks from the time vTaskDelay () is called. It is therefore difficult to use vTaskDelay () by itself to generate a fixed execution frequency as the time between a task starting to execute and that task calling vTaskDelay () may not be fixed [the task may take a different path though the code between calls, or may get interrupted or preempted a different number of times each time it executes].

Whereas vTaskDelay () specifies a wake time relative to the time at which the function is called, vTaskDelayUntil () specifies the absolute (exact) time at which it wishes to unblock.

The constant portTICK_PERIOD_MS can be used to calculate real time from the tick rate - with the resolution of one tick period.

Example usage:

```
// Perform an action every 10 ticks.
void vTaskFunction( void * pvParameters )
{
TickType_t xLastWakeTime;
const TickType_t xFrequency = 10;

    // Initialise the xLastWakeTime variable with the current time.
    xLastWakeTime = xTaskGetTickCount ();
    for( ;; )
    {
        // Wait for the next cycle.
        vTaskDelayUntil( &xLastWakeTime, xFrequency );

        // Perform action here.
```

(下页继续)

(续上页)

```
    }
}
```

Parameters

- `pxPreviousWakeTime`: Pointer to a variable that holds the time at which the task was last unblocked. The variable must be initialised with the current time prior to its first use (see the example below). Following this the variable is automatically updated within vTaskDelayUntil ().
- `xTimeIncrement`: The cycle time period. The task will be unblocked at time *pxPreviousWakeTime + xTimeIncrement. Calling vTaskDelayUntil with the same xTimeIncrement parameter value will cause the task to execute with a fixed interface period.

UBaseType_t **uxTaskPriorityGet**(*TaskHandle_t xTask*)

Obtain the priority of any task.

INCLUDE_uxTaskPriorityGet must be defined as 1 for this function to be available. See the configuration section for more information.

Example usage:

```
void vAFunction( void )
{
TaskHandle_t xHandle;

 // Create a task, storing the handle.
 xTaskCreate( vTaskCode, "NAME", STACK_SIZE, NULL, tskIDLE_PRIORITY, &xHandle );

 // ...

 // Use the handle to obtain the priority of the created task.
 // It was created with tskIDLE_PRIORITY, but may have changed
 // it itself.
 if( uxTaskPriorityGet( xHandle ) != tskIDLE_PRIORITY )
 {
     // The task has changed it's priority.
 }

 // ...

 // Is our priority higher than the created task?
```

(下页继续)

(续上页)

```
  if( uxTaskPriorityGet( xHandle ) < uxTaskPriorityGet( NULL ) )
  {
      // Our priority (obtained using NULL handle) is higher.
  }
}
```

Return The priority of xTask.

Parameters

- `xTask`: Handle of the task to be queried. Passing a NULL handle results in the priority of the calling task being returned.

UBaseType_t **uxTaskPriorityGetFromISR**(*TaskHandle_t xTask*)

A version of uxTaskPriorityGet() that can be used from an ISR.

Return The priority of xTask.

Parameters

- `xTask`: Handle of the task to be queried. Passing a NULL handle results in the priority of the calling task being returned.

eTaskState **eTaskGetState**(*TaskHandle_t xTask*)

Obtain the state of any task.

States are encoded by the eTaskState enumerated type.

INCLUDE_eTaskGetState must be defined as 1 for this function to be available. See the configuration section for more information.

Return The state of xTask at the time the function was called. Note the state of the task might change between the function being called, and the functions return value being tested by the calling task.

Parameters

- `xTask`: Handle of the task to be queried.

void **vTaskPrioritySet**(*TaskHandle_t xTask*, UBaseType_t *uxNewPriority*)

Set the priority of any task.

INCLUDE_vTaskPrioritySet must be defined as 1 for this function to be available. See the configuration section for more information.

A context switch will occur before the function returns if the priority being set is higher than the currently executing task.

Example usage:

```
void vAFunction( void )
{
TaskHandle_t xHandle;

 // Create a task, storing the handle.
 xTaskCreate( vTaskCode, "NAME", STACK_SIZE, NULL, tskIDLE_PRIORITY, &xHandle );

 // ...

 // Use the handle to raise the priority of the created task.
 vTaskPrioritySet( xHandle, tskIDLE_PRIORITY + 1 );

 // ...

 // Use a NULL handle to raise our priority to the same value.
 vTaskPrioritySet( NULL, tskIDLE_PRIORITY + 1 );
}
```

Parameters

- `xTask`: Handle to the task for which the priority is being set. Passing a NULL handle results in the priority of the calling task being set.
- `uxNewPriority`: The priority to which the task will be set.

void **vTaskSuspend**(*TaskHandle_t xTaskToSuspend*)

Suspend a task.

INCLUDE_vTaskSuspend must be defined as 1 for this function to be available. See the configuration section for more information.

When suspended, a task will never get any microcontroller processing time, no matter what its priority.

Calls to vTaskSuspend are not accumulative - i.e. calling vTaskSuspend () twice on the same task still only requires one call to vTaskResume () to ready the suspended task.

Example usage:

```
void vAFunction( void )
{
TaskHandle_t xHandle;

 // Create a task, storing the handle.
```

(下页继续)

(续上页)

```
xTaskCreate( vTaskCode, "NAME", STACK_SIZE, NULL, tskIDLE_PRIORITY, &xHandle );

// ...

// Use the handle to suspend the created task.
vTaskSuspend( xHandle );

// ...

// The created task will not run during this period, unless
// another task calls vTaskResume( xHandle ).

//...


// Suspend ourselves.
vTaskSuspend( NULL );

// We cannot get here unless another task calls vTaskResume
// with our handle as the parameter.
}
```

Parameters

- `xTaskToSuspend`: Handle to the task being suspended. Passing a NULL handle will cause the calling task to be suspended.

void **vTaskResume**(*TaskHandle_t xTaskToResume*)

Resumes a suspended task.

INCLUDE_vTaskSuspend must be defined as 1 for this function to be available. See the configuration section for more information.

A task that has been suspended by one or more calls to vTaskSuspend () will be made available for running again by a single call to vTaskResume ().

Example usage:

```
void vAFunction( void )
{
TaskHandle_t xHandle;
```

(下页继续)

(续上页)

```
// Create a task, storing the handle.
xTaskCreate( vTaskCode, "NAME", STACK_SIZE, NULL, tskIDLE_PRIORITY, &xHandle );

// ...

// Use the handle to suspend the created task.
vTaskSuspend( xHandle );

// ...

// The created task will not run during this period, unless
// another task calls vTaskResume( xHandle ).

//...


// Resume the suspended task ourselves.
vTaskResume( xHandle );

// The created task will once again get microcontroller processing
// time in accordance with its priority within the system.
}
```

Parameters

- `xTaskToResume`: Handle to the task being readied.

BaseType_t **xTaskResumeFromISR**(*TaskHandle_t xTaskToResume*)

An implementation of vTaskResume() that can be called from within an ISR.

INCLUDE_xTaskResumeFromISR must be defined as 1 for this function to be available. See the configuration section for more information.

A task that has been suspended by one or more calls to vTaskSuspend () will be made available for running again by a single call to xTaskResumeFromISR ().

xTaskResumeFromISR() should not be used to synchronise a task with an interrupt if there is a chance that the interrupt could arrive prior to the task being suspended - as this can lead to interrupts being missed. Use of a semaphore as a synchronisation mechanism would avoid this eventuality.

Return pdTRUE if resuming the task should result in a context switch, otherwise pdFALSE. This is used by the ISR to determine if a context switch may be required following the ISR.

Parameters

- `xTaskToResume`: Handle to the task being readied.

void **vTaskSuspendAll**(void)

Suspends the scheduler without disabling interrupts.

Context switches will not occur while the scheduler is suspended.

After calling vTaskSuspendAll () the calling task will continue to execute without risk of being swapped out until a call to xTaskResumeAll () has been made.

API functions that have the potential to cause a context switch (for example, vTaskDelayUntil(), xQueueSend(), etc.) must not be called while the scheduler is suspended.

Example usage:

```
void vTask1( void * pvParameters )
{
 for( ;; )
 {
     // Task code goes here.

     // ...

     // At some point the task wants to perform a long operation during
     // which it does not want to get swapped out.  It cannot use
     // taskENTER_CRITICAL ()/taskEXIT_CRITICAL () as the length of the
     // operation may cause interrupts to be missed - including the
     // ticks.

     // Prevent the real time kernel swapping out the task.
     vTaskSuspendAll ();

     // Perform the operation here.  There is no need to use critical
     // sections as we have all the microcontroller processing time.
     // During this time interrupts will still operate and the kernel
     // tick count will be maintained.

     // ...

     // The operation is complete.  Restart the kernel.
     xTaskResumeAll ();
 }
}
```

BaseType_t **xTaskResumeAll**(void)

Resumes scheduler activity after it was suspended by a call to vTaskSuspendAll().

xTaskResumeAll() only resumes the scheduler. It does not unsuspend tasks that were previously suspended by a call to vTaskSuspend().

Example usage:

```
void vTask1( void * pvParameters )
{
 for( ;; )
 {
     // Task code goes here.

     // ...

     // At some point the task wants to perform a long operation during
     // which it does not want to get swapped out.  It cannot use
     // taskENTER_CRITICAL ()/taskEXIT_CRITICAL () as the length of the
     // operation may cause interrupts to be missed - including the
     // ticks.

     // Prevent the real time kernel swapping out the task.
     vTaskSuspendAll ();

     // Perform the operation here.  There is no need to use critical
     // sections as we have all the microcontroller processing time.
     // During this time interrupts will still operate and the real
     // time kernel tick count will be maintained.

     // ...

     // The operation is complete.  Restart the kernel.  We want to force
     // a context switch - but there is no point if resuming the scheduler
     // caused a context switch already.
     if( !xTaskResumeAll () )
     {
          taskYIELD ();
     }
 }
}
```

Return If resuming the scheduler caused a context switch then pdTRUE is returned, otherwise pdFALSE is returned.

TickType_t **xTaskGetTickCount**(void)

Get tick count

Return The count of ticks since vTaskStartScheduler was called.

TickType_t **xTaskGetTickCountFromISR**(void)

Get tick count from ISR

This is a version of xTaskGetTickCount() that is safe to be called from an ISR - provided that TickType_t is the natural word size of the microcontroller being used or interrupt nesting is either not supported or not being used.

Return The count of ticks since vTaskStartScheduler was called.

UBaseType_t **uxTaskGetNumberOfTasks**(void)

Get current number of tasks

Return The number of tasks that the real time kernel is currently managing. This includes all ready, blocked and suspended tasks. A task that has been deleted but not yet freed by the idle task will also be included in the count.

char ***pcTaskGetTaskName**(*TaskHandle_t xTaskToQuery*)

Get task name

Return The text (human readable) name of the task referenced by the handle xTaskToQuery. A task can query its own name by either passing in its own handle, or by setting xTaskToQuery to NULL. INCLUDE_pcTaskGetTaskName must be set to 1 in FreeRTOSConfig.h for pcTaskGetTaskName() to be available.

UBaseType_t **uxTaskGetStackHighWaterMark**(*TaskHandle_t xTask*)

Returns the high water mark of the stack associated with xTask.

INCLUDE_uxTaskGetStackHighWaterMark must be set to 1 in FreeRTOSConfig.h for this function to be available.

High water mark is the minimum free stack space there has been (in bytes rather than words as found in vanilla FreeRTOS) since the task started. The smaller the returned number the closer the task has come to overflowing its stack.

Return The smallest amount of free stack space there has been (in bytes rather than words as found in vanilla FreeRTOS) since the task referenced by xTask was created.

Parameters

- `xTask`: Handle of the task associated with the stack to be checked. Set xTask to NULL to check the stack of the calling task.

uint8_t *pxTaskGetStackStart(*TaskHandle_t xTask*)

Returns the start of the stack associated with xTask.

INCLUDE_pxTaskGetStackStart must be set to 1 in FreeRTOSConfig.h for this function to be available.

Returns the highest stack memory address on architectures where the stack grows down from high memory, and the lowest memory address on architectures where the stack grows up from low memory.

Return A pointer to the start of the stack.

Parameters

- `xTask`: Handle of the task associated with the stack returned. Set xTask to NULL to return the stack of the calling task.

void **vTaskSetApplicationTaskTag**(*TaskHandle_t xTask*, *TaskHookFunction_t pxHookFunction*)

Sets pxHookFunction to be the task hook function used by the task xTask.

Parameters

- `xTask`: Handle of the task to set the hook function for Passing xTask as NULL has the effect of setting the calling tasks hook function.
- `pxHookFunction`: Pointer to the hook function.

TaskHookFunction_t **xTaskGetApplicationTaskTag**(*TaskHandle_t xTask*)

Get the hook function assigned to given task.

Return The pxHookFunction value assigned to the task xTask.

Parameters

- `xTask`: Handle of the task to get the hook function for Passing xTask as NULL has the effect of getting the calling tasks hook function.

void **vTaskSetThreadLocalStoragePointer**(*TaskHandle_t xTaskToSet*, BaseType_t *xIndex*, void **pvValue*)

Set local storage pointer specific to the given task.

Each task contains an array of pointers that is dimensioned by the configNUM_THREAD_LOCAL_STORAGE_POINTERS setting in FreeRTOSConfig.h. The kernel does not use the pointers itself, so the application writer can use the pointers for any purpose they wish.

Parameters

- `xTaskToSet`: Task to set thread local storage pointer for
- `xIndex`: The index of the pointer to set, from 0 to configNUM_THREAD_LOCAL_STORAGE_POINTERS - 1.

- `pvValue`: Pointer value to set.

void *`pvTaskGetThreadLocalStoragePointer`(*TaskHandle_t xTaskToQuery*, BaseType_t *xIndex*)

Get local storage pointer specific to the given task.

Each task contains an array of pointers that is dimensioned by the configNUM_THREAD_LOCAL_STORAGE_POINTERS setting in FreeRTOSConfig.h. The kernel does not use the pointers itself, so the application writer can use the pointers for any purpose they wish.

Return Pointer value

Parameters

- `xTaskToQuery`: Task to get thread local storage pointer for
- `xIndex`: The index of the pointer to get, from 0 to configNUM_THREAD_LOCAL_STORAGE_POINTERS - 1.

void `vTaskSetThreadLocalStoragePointerAndDelCallback`(*TaskHandle_t xTaskToSet*, BaseType_t *xIndex*, void **pvValue*, *TlsDeleteCallbackFunction_t pvDelCallback*)

Set local storage pointer and deletion callback.

Each task contains an array of pointers that is dimensioned by the configNUM_THREAD_LOCAL_STORAGE_POINTERS setting in FreeRTOSConfig.h. The kernel does not use the pointers itself, so the application writer can use the pointers for any purpose they wish.

Local storage pointers set for a task can reference dynamically allocated resources. This function is similar to vTaskSetThreadLocalStoragePointer, but provides a way to release these resources when the task gets deleted. For each pointer, a callback function can be set. This function will be called when task is deleted, with the local storage pointer index and value as arguments.

Parameters

- `xTaskToSet`: Task to set thread local storage pointer for
- `xIndex`: The index of the pointer to set, from 0 to configNUM_THREAD_LOCAL_STORAGE_POINTERS - 1.
- `pvValue`: Pointer value to set.
- `pvDelCallback`: Function to call to dispose of the local storage pointer when the task is deleted.

BaseType_t `xTaskCallApplicationTaskHook`(*TaskHandle_t xTask*, void **pvParameter*)

Calls the hook function associated with xTask. Passing xTask as NULL has the effect of calling the

Running tasks (the calling task) hook function.

Parameters

- `xTask`: Handle of the task to call the hook for.
- `pvParameter`: Parameter passed to the hook function for the task to interpret as it wants. The return value is the value returned by the task hook function registered by the user.

TaskHandle_t **xTaskGetIdleTaskHandle**(void)

Get the handle of idle task for the current CPU.

xTaskGetIdleTaskHandle() is only available if INCLUDE_xTaskGetIdleTaskHandle is set to 1 in FreeRTOSConfig.h.

Return The handle of the idle task. It is not valid to call xTaskGetIdleTaskHandle() before the scheduler has been started.

TaskHandle_t **xTaskGetIdleTaskHandleForCPU**(UBaseType_t *cpuid*)

Get the handle of idle task for the given CPU.

xTaskGetIdleTaskHandleForCPU() is only available if INCLUDE_xTaskGetIdleTaskHandle is set to 1 in FreeRTOSConfig.h.

Return Idle task handle of a given cpu. It is not valid to call xTaskGetIdleTaskHandleForCPU() before the scheduler has been started.

Parameters

- `cpuid`: The CPU to get the handle for

UBaseType_t **uxTaskGetSystemState**(*TaskStatus_t* ***const** *pxTaskStatusArray*, **const** UBaseType_t *uxArraySize*, uint32_t ***const** *pulTotalRunTime*)

Get the state of tasks in the system.

configUSE_TRACE_FACILITY must be defined as 1 in FreeRTOSConfig.h for uxTaskGetSystemState() to be available.

uxTaskGetSystemState() populates an TaskStatus_t structure for each task in the system. TaskStatus_t structures contain, among other things, members for the task handle, task name, task priority, task state, and total amount of run time consumed by the task. See the TaskStatus_t structure definition in this file for the full member list.

Example usage:

```
// This example demonstrates how a human readable table of run time stats
// information is generated from raw data provided by uxTaskGetSystemState().
// The human readable table is written to pcWriteBuffer
```

(下页继续)

(续上页)

```
void vTaskGetRunTimeStats( char *pcWriteBuffer )
{
TaskStatus_t *pxTaskStatusArray;
volatile UBaseType_t uxArraySize, x;
uint32_t ulTotalRunTime, ulStatsAsPercentage;

 // Make sure the write buffer does not contain a string.
 *pcWriteBuffer = 0x00;

 // Take a snapshot of the number of tasks in case it changes while this
 // function is executing.
 uxArraySize = uxTaskGetNumberOfTasks();

 // Allocate a TaskStatus_t structure for each task.  An array could be
 // allocated statically at compile time.
 pxTaskStatusArray = pvPortMalloc( uxArraySize * sizeof( TaskStatus_t ) );

 if( pxTaskStatusArray != NULL )
 {
     // Generate raw status information about each task.
     uxArraySize = uxTaskGetSystemState( pxTaskStatusArray, uxArraySize, &
↪ulTotalRunTime );

     // For percentage calculations.
     ulTotalRunTime /= 100UL;

     // Avoid divide by zero errors.
     if( ulTotalRunTime > 0 )
     {
         // For each populated position in the pxTaskStatusArray array,
         // format the raw data as human readable ASCII data
         for( x = 0; x < uxArraySize; x++ )
         {
             // What percentage of the total run time has the task used?
             // This will always be rounded down to the nearest integer.
             // ulTotalRunTimeDiv100 has already been divided by 100.
             ulStatsAsPercentage = pxTaskStatusArray[ x ].ulRunTimeCounter /␣
↪ulTotalRunTime;

             if( ulStatsAsPercentage > 0UL )
```

(下页继续)

(续上页)

```
            {
                sprintf( pcWriteBuffer, "%s\t\t%lu\t\t%lu%%\r\n",␣
↪pxTaskStatusArray[ x ].pcTaskName, pxTaskStatusArray[ x ].ulRunTimeCounter,␣
↪ulStatsAsPercentage );
            }
            else
            {
                // If the percentage is zero here then the task has
                // consumed less than 1% of the total run time.
                sprintf( pcWriteBuffer, "%s\t\t%lu\t\t<1%%\r\n",␣
↪pxTaskStatusArray[ x ].pcTaskName, pxTaskStatusArray[ x ].ulRunTimeCounter );
            }

            pcWriteBuffer += strlen( ( char * ) pcWriteBuffer );
        }
    }

    // The array is no longer needed, free the memory it consumes.
    vPortFree( pxTaskStatusArray );
 }
}
```

Note This function is intended for debugging use only as its use results in the scheduler remaining suspended for an extended period.

Return The number of TaskStatus_t structures that were populated by uxTaskGetSystemState(). This should equal the number returned by the uxTaskGetNumberOfTasks() API function, but will be zero if the value passed in the uxArraySize parameter was too small.

Parameters

- `pxTaskStatusArray`: A pointer to an array of TaskStatus_t structures. The array must contain at least one TaskStatus_t structure for each task that is under the control of the RTOS. The number of tasks under the control of the RTOS can be determined using the uxTaskGetNumberOfTasks() API function.
- `uxArraySize`: The size of the array pointed to by the pxTaskStatusArray parameter. The size is specified as the number of indexes in the array, or the number of TaskStatus_t structures contained in the array, not by the number of bytes in the array.
- `pulTotalRunTime`: If configGENERATE_RUN_TIME_STATS is set to 1 in FreeRTOSConfig.h then *pulTotalRunTime is set by uxTaskGetSystemState() to the total run time (as defined by the run time stats clock, see http://www.freertos.org/rtos-run-time-stats.html)

since the target booted. pulTotalRunTime can be set to NULL to omit the total run time information.

void **vTaskList**(char **pcWriteBuffer*)

List all the current tasks.

configUSE_TRACE_FACILITY and configUSE_STATS_FORMATTING_FUNCTIONS must both be defined as 1 for this function to be available. See the configuration section of the FreeRTOS.org website for more information.

Lists all the current tasks, along with their current state and stack usage high water mark.

Note This function will disable interrupts for its duration. It is not intended for normal application runtime use but as a debug aid.

Tasks are reported as blocked ('B'), ready ('R'), deleted ('D') or suspended ('S').

vTaskList() calls uxTaskGetSystemState(), then formats part of the uxTaskGetSystemState() output into a human readable table that displays task names, states and stack usage.

Note This function is provided for convenience only, and is used by many of the demo applications. Do not consider it to be part of the scheduler.

vTaskList() has a dependency on the sprintf() C library function that might bloat the code size, use a lot of stack, and provide different results on different platforms. An alternative, tiny, third party, and limited functionality implementation of sprintf() is provided in many of the FreeRTOS/Demo sub-directories in a file called printf-stdarg.c (note printf-stdarg.c does not provide a full snprintf() implementation!).

It is recommended that production systems call uxTaskGetSystemState() directly to get access to raw stats data, rather than indirectly through a call to vTaskList().

Parameters

- `pcWriteBuffer`: A buffer into which the above mentioned details will be written, in ASCII form. This buffer is assumed to be large enough to contain the generated report. Approximately 40 bytes per task should be sufficient.

void **vTaskGetRunTimeStats**(char **pcWriteBuffer*)

Get the state of running tasks as a string

configGENERATE_RUN_TIME_STATS and configUSE_STATS_FORMATTING_FUNCTIONS must both be defined as 1 for this function to be available. The application must also then provide definitions for portCONFIGURE_TIMER_FOR_RUN_TIME_STATS() and portGET_RUN_TIME_COUNTER_VALUE() to configure a peripheral timer/counter and return the timers current count value respectively. The counter should be at least 10 times the frequency of the tick count.

Setting configGENERATE_RUN_TIME_STATS to 1 will result in a total accumulated execution time being stored for each task. The resolution of the accumulated time value depends on the frequency of the timer configured by the portCONFIGURE_TIMER_FOR_RUN_TIME_STATS() macro. Calling vTaskGetRunTimeStats() writes the total execution time of each task into a buffer, both as an absolute count value and as a percentage of the total system execution time.

Note This function will disable interrupts for its duration. It is not intended for normal application runtime use but as a debug aid.

vTaskGetRunTimeStats() calls uxTaskGetSystemState(), then formats part of the uxTaskGetSystemState() output into a human readable table that displays the amount of time each task has spent in the Running state in both absolute and percentage terms.

Note This function is provided for convenience only, and is used by many of the demo applications. Do not consider it to be part of the scheduler.

vTaskGetRunTimeStats() has a dependency on the sprintf() C library function that might bloat the code size, use a lot of stack, and provide different results on different platforms. An alternative, tiny, third party, and limited functionality implementation of sprintf() is provided in many of the FreeRTOS/Demo sub-directories in a file called printf-stdarg.c (note printf-stdarg.c does not provide a full snprintf() implementation!).

It is recommended that production systems call uxTaskGetSystemState() directly to get access to raw stats data, rather than indirectly through a call to vTaskGetRunTimeStats().

Parameters

- `pcWriteBuffer`: A buffer into which the execution times will be written, in ASCII form. This buffer is assumed to be large enough to contain the generated report. Approximately 40 bytes per task should be sufficient.

BaseType_t **`xTaskNotify`**(*TaskHandle_t xTaskToNotify*, uint32_t *ulValue*, *eNotifyAction eAction*)

Send task notification.

configUSE_TASK_NOTIFICATIONS must be undefined or defined as 1 for this function to be available.

When configUSE_TASK_NOTIFICATIONS is set to one each task has its own private "notification value", which is a 32-bit unsigned integer (uint32_t).

Events can be sent to a task using an intermediary object. Examples of such objects are queues, semaphores, mutexes and event groups. Task notifications are a method of sending an event directly to a task without the need for such an intermediary object.

A notification sent to a task can optionally perform an action, such as update, overwrite or increment the task's notification value. In that way task notifications can be used to send data to a task, or be used as light weight and fast binary or counting semaphores.

A notification sent to a task will remain pending until it is cleared by the task calling xTaskNotifyWait() or ulTaskNotifyTake(). If the task was already in the Blocked state to wait for a notification when the notification arrives then the task will automatically be removed from the Blocked state (unblocked) and the notification cleared.

A task can use xTaskNotifyWait() to [optionally] block to wait for a notification to be pending, or ulTaskNotifyTake() to [optionally] block to wait for its notification value to have a non-zero value. The task does not consume any CPU time while it is in the Blocked state.

See http://www.FreeRTOS.org/RTOS-task-notifications.html for details.

Return Dependent on the value of eAction. See the description of the eAction parameter.

Parameters

- `xTaskToNotify`: The handle of the task being notified. The handle to a task can be returned from the xTaskCreate() API function used to create the task, and the handle of the currently running task can be obtained by calling xTaskGetCurrentTaskHandle().
- `ulValue`: Data that can be sent with the notification. How the data is used depends on the value of the eAction parameter.
- `eAction`: Specifies how the notification updates the task' s notification value, if at all. Valid values for eAction are as follows:
  - eSetBits: The task' s notification value is bitwise ORed with ulValue. xTaskNofify() always returns pdPASS in this case.
  - eIncrement: The task' s notification value is incremented. ulValue is not used and xTaskNotify() always returns pdPASS in this case.
  - eSetValueWithOverwrite: The task' s notification value is set to the value of ulValue, even if the task being notified had not yet processed the previous notification (the task already had a notification pending). xTaskNotify() always returns pdPASS in this case.
  - eSetValueWithoutOverwrite: If the task being notified did not already have a notification pending then the task' s notification value is set to ulValue and xTaskNotify() will return pdPASS. If the task being notified already had a notification pending then no action is performed and pdFAIL is returned.
  - eNoAction: The task receives a notification without its notification value being updated. ulValue is not used and xTaskNotify() always returns pdPASS in this case.

BaseType_t **xTaskNotifyFromISR**(*TaskHandle_t xTaskToNotify*, uint32_t *ulValue*, *eNotifyAction eAction*, BaseType_t **pxHigherPriorityTaskWoken*)

Send task notification from an ISR.

configUSE_TASK_NOTIFICATIONS must be undefined or defined as 1 for this function to be available.

When configUSE_TASK_NOTIFICATIONS is set to one each task has its own private "notification value", which is a 32-bit unsigned integer (uint32_t).

A version of xTaskNotify() that can be used from an interrupt service routine (ISR).

Events can be sent to a task using an intermediary object. Examples of such objects are queues, semaphores, mutexes and event groups. Task notifications are a method of sending an event directly to a task without the need for such an intermediary object.

A notification sent to a task can optionally perform an action, such as update, overwrite or increment the task's notification value. In that way task notifications can be used to send data to a task, or be used as light weight and fast binary or counting semaphores.

A notification sent to a task will remain pending until it is cleared by the task calling xTaskNotifyWait() or ulTaskNotifyTake(). If the task was already in the Blocked state to wait for a notification when the notification arrives then the task will automatically be removed from the Blocked state (unblocked) and the notification cleared.

A task can use xTaskNotifyWait() to [optionally] block to wait for a notification to be pending, or ulTaskNotifyTake() to [optionally] block to wait for its notification value to have a non-zero value. The task does not consume any CPU time while it is in the Blocked state.

See http://www.FreeRTOS.org/RTOS-task-notifications.html for details.

Return Dependent on the value of eAction. See the description of the eAction parameter.

Parameters

- `xTaskToNotify`: The handle of the task being notified. The handle to a task can be returned from the xTaskCreate() API function used to create the task, and the handle of the currently running task can be obtained by calling xTaskGetCurrentTaskHandle().
- `ulValue`: Data that can be sent with the notification. How the data is used depends on the value of the eAction parameter.
- `eAction`: Specifies how the notification updates the task's notification value, if at all. Valid values for eAction are as follows:
  - eSetBits: The task's notification value is bitwise ORed with ulValue. xTaskNofify() always returns pdPASS in this case.
  - eIncrement: The task's notification value is incremented. ulValue is not used and xTaskNotify() always returns pdPASS in this case.
  - eSetValueWithOverwrite: The task's notification value is set to the value of ulValue, even if the task being notified had not yet processed the previous notification (the task already had a notification pending). xTaskNotify() always returns pdPASS in this case.
  - eSetValueWithoutOverwrite: If the task being notified did not already have a notification pending then the task's notification value is set to ulValue and xTaskNotify() will return

pdPASS. If the task being notified already had a notification pending then no action is performed and pdFAIL is returned.

- eNoAction: The task receives a notification without its notification value being updated. ulValue is not used and xTaskNotify() always returns pdPASS in this case.

- `pxHigherPriorityTaskWoken`: xTaskNotifyFromISR() will set *pxHigherPriorityTaskWoken to pdTRUE if sending the notification caused the task to which the notification was sent to leave the Blocked state, and the unblocked task has a priority higher than the currently running task. If xTaskNotifyFromISR() sets this value to pdTRUE then a context switch should be requested before the interrupt is exited. How a context switch is requested from an ISR is dependent on the port - see the documentation page for the port in use.

BaseType_t **xTaskNotifyWait**(uint32_t *ulBitsToClearOnEntry*, uint32_t *ulBitsToClearOnExit*, uint32_t **pulNotificationValue*, TickType_t *xTicksToWait*)

Wait for task notification

configUSE_TASK_NOTIFICATIONS must be undefined or defined as 1 for this function to be available.

When configUSE_TASK_NOTIFICATIONS is set to one each task has its own private "notification value", which is a 32-bit unsigned integer (uint32_t).

Events can be sent to a task using an intermediary object. Examples of such objects are queues, semaphores, mutexes and event groups. Task notifications are a method of sending an event directly to a task without the need for such an intermediary object.

A notification sent to a task can optionally perform an action, such as update, overwrite or increment the task's notification value. In that way task notifications can be used to send data to a task, or be used as light weight and fast binary or counting semaphores.

A notification sent to a task will remain pending until it is cleared by the task calling xTaskNotifyWait() or ulTaskNotifyTake(). If the task was already in the Blocked state to wait for a notification when the notification arrives then the task will automatically be removed from the Blocked state (unblocked) and the notification cleared.

A task can use xTaskNotifyWait() to [optionally] block to wait for a notification to be pending, or ulTaskNotifyTake() to [optionally] block to wait for its notification value to have a non-zero value. The task does not consume any CPU time while it is in the Blocked state.

See http://www.FreeRTOS.org/RTOS-task-notifications.html for details.

Return If a notification was received (including notifications that were already pending when xTaskNotifyWait was called) then pdPASS is returned. Otherwise pdFAIL is returned.

Parameters

- `ulBitsToClearOnEntry`: Bits that are set in ulBitsToClearOnEntry value will be cleared in the calling task's notification value before the task checks to see if any notifications are

pending, and optionally blocks if no notifications are pending. Setting ulBitsToClearOnEntry to ULONG_MAX (if limits.h is included) or 0xffffffffUL (if limits.h is not included) will have the effect of resetting the task' s notification value to 0. Setting ulBitsToClearOnEntry to 0 will leave the task' s notification value unchanged.

- `ulBitsToClearOnExit`: If a notification is pending or received before the calling task exits the xTaskNotifyWait() function then the task's notification value (see the xTaskNotify() API function) is passed out using the pulNotificationValue parameter. Then any bits that are set in ulBitsToClearOnExit will be cleared in the task's notification value (note *pulNotification-Value is set before any bits are cleared). Setting ulBitsToClearOnExit to ULONG_MAX (if limits.h is included) or 0xffffffffUL (if limits.h is not included) will have the effect of resetting the task' s notification value to 0 before the function exits. Setting ulBitsToClearOnExit to 0 will leave the task' s notification value unchanged when the function exits (in which case the value passed out in pulNotificationValue will match the task' s notification value).
- `pulNotificationValue`: Used to pass the task' s notification value out of the function. Note the value passed out will not be effected by the clearing of any bits caused by ulBitsTo-ClearOnExit being non-zero.
- `xTicksToWait`: The maximum amount of time that the task should wait in the Blocked state for a notification to be received, should a notification not already be pending when xTaskNotifyWait() was called. The task will not consume any processing time while it is in the Blocked state. This is specified in kernel ticks, the macro pdMS_TO_TICSK(value_in_ms) can be used to convert a time specified in milliseconds to a time specified in ticks.

void **vTaskNotifyGiveFromISR**(*TaskHandle_t xTaskToNotify*, BaseType_t **pxHigherPriorityTaskWoken*)

Simplified macro for sending task notification from ISR.

configUSE_TASK_NOTIFICATIONS must be undefined or defined as 1 for this macro to be available.

When configUSE_TASK_NOTIFICATIONS is set to one each task has its own private "notification value" , which is a 32-bit unsigned integer (uint32_t).

A version of xTaskNotifyGive() that can be called from an interrupt service routine (ISR).

Events can be sent to a task using an intermediary object. Examples of such objects are queues, semaphores, mutexes and event groups. Task notifications are a method of sending an event directly to a task without the need for such an intermediary object.

A notification sent to a task can optionally perform an action, such as update, overwrite or increment the task' s notification value. In that way task notifications can be used to send data to a task, or be used as light weight and fast binary or counting semaphores.

vTaskNotifyGiveFromISR() is intended for use when task notifications are used as light weight and faster binary or counting semaphore equivalents. Actual FreeRTOS semaphores are given from an ISR using the xSemaphoreGiveFromISR() API function, the equivalent action that instead uses a task notification is vTaskNotifyGiveFromISR().

When task notifications are being used as a binary or counting semaphore equivalent then the task being notified should wait for the notification using the ulTaskNotificationTake() API function rather than the xTaskNotifyWait() API function.

See http://www.FreeRTOS.org/RTOS-task-notifications.html for more details.

Parameters

- `xTaskToNotify`: The handle of the task being notified. The handle to a task can be returned from the xTaskCreate() API function used to create the task, and the handle of the currently running task can be obtained by calling xTaskGetCurrentTaskHandle().
- `pxHigherPriorityTaskWoken`: vTaskNotifyGiveFromISR() will set *pxHigherPriorityTaskWoken to pdTRUE if sending the notification caused the task to which the notification was sent to leave the Blocked state, and the unblocked task has a priority higher than the currently running task. If vTaskNotifyGiveFromISR() sets this value to pdTRUE then a context switch should be requested before the interrupt is exited. How a context switch is requested from an ISR is dependent on the port - see the documentation page for the port in use.

uint32_t **ulTaskNotifyTake**(BaseType_t *xClearCountOnExit*, TickType_t *xTicksToWait*)

Simplified macro for receiving task notification.

configUSE_TASK_NOTIFICATIONS must be undefined or defined as 1 for this function to be available.

When configUSE_TASK_NOTIFICATIONS is set to one each task has its own private "notification value", which is a 32-bit unsigned integer (uint32_t).

Events can be sent to a task using an intermediary object. Examples of such objects are queues, semaphores, mutexes and event groups. Task notifications are a method of sending an event directly to a task without the need for such an intermediary object.

A notification sent to a task can optionally perform an action, such as update, overwrite or increment the task's notification value. In that way task notifications can be used to send data to a task, or be used as light weight and fast binary or counting semaphores.

ulTaskNotifyTake() is intended for use when a task notification is used as a faster and lighter weight binary or counting semaphore alternative. Actual FreeRTOS semaphores are taken using the xSemaphoreTake() API function, the equivalent action that instead uses a task notification is ulTaskNotifyTake().

When a task is using its notification value as a binary or counting semaphore other tasks should send notifications to it using the xTaskNotifyGive() macro, or xTaskNotify() function with the eAction parameter set to eIncrement.

ulTaskNotifyTake() can either clear the task's notification value to zero on exit, in which case the notification value acts like a binary semaphore, or decrement the task's notification value on exit, in

which case the notification value acts like a counting semaphore.

A task can use ulTaskNotifyTake() to [optionally] block to wait for a the task' s notification value to be non-zero. The task does not consume any CPU time while it is in the Blocked state.

Where as xTaskNotifyWait() will return when a notification is pending, ulTaskNotifyTake() will return when the task' s notification value is not zero.

See http://www.FreeRTOS.org/RTOS-task-notifications.html for details.

Return The task' s notification count before it is either cleared to zero or decremented (see the xClearCountOnExit parameter).

Parameters

- `xClearCountOnExit`: if xClearCountOnExit is pdFALSE then the task' s notification value is decremented when the function exits. In this way the notification value acts like a counting semaphore. If xClearCountOnExit is not pdFALSE then the task' s notification value is cleared to zero when the function exits. In this way the notification value acts like a binary semaphore.
- `xTicksToWait`: The maximum amount of time that the task should wait in the Blocked state for the task' s notification value to be greater than zero, should the count not already be greater than zero when ulTaskNotifyTake() was called. The task will not consume any processing time while it is in the Blocked state. This is specified in kernel ticks, the macro pdMS_TO_TICSK(value_in_ms) can be used to convert a time specified in milliseconds to a time specified in ticks.

Structures

`struct xTASK_STATUS`

Used with the uxTaskGetSystemState() function to return the state of each task in the system.

Public Members

TaskHandle_t **xHandle**

The handle of the task to which the rest of the information in the structure relates.

const char ***pcTaskName**

A pointer to the task' s name. This value will be invalid if the task was deleted since the structure was populated!

UBaseType_t **xTaskNumber**

A number unique to the task.

eTaskState **eCurrentState**

The state in which the task existed when the structure was populated.

UBaseType_t **uxCurrentPriority**

The priority at which the task was running (may be inherited) when the structure was populated.

UBaseType_t **uxBasePriority**

The priority to which the task will return if the task' s current priority has been inherited to avoid unbounded priority inversion when obtaining a mutex. Only valid if configUSE_MUTEXES is defined as 1 in FreeRTOSConfig.h.

uint32_t **ulRunTimeCounter**

The total run time allocated to the task so far, as defined by the run time stats clock. See http://www.freertos.org/rtos-run-time-stats.html. Only valid when configGENERATE_RUN_TIME_STATS is defined as 1 in FreeRTOSConfig.h.

StackType_t ***pxStackBase**

Points to the lowest address of the task' s stack area.

uint32_t **usStackHighWaterMark**

The minimum amount of stack space that has remained for the task since the task was created. The closer this value is to zero the closer the task has come to overflowing its stack.

BaseType_t **xCoreID**

Core this task is pinned to (0, 1, or -1 for tskNO_AFFINITY). This field is present if CONFIG_FREERTOS_VTASKLIST_INCLUDE_COREID is set.

struct xTASK_SNAPSHOT

Used with the uxTaskGetSnapshotAll() function to save memory snapshot of each task in the system. We need this struct because TCB_t is defined (hidden) in tasks.c.

Public Members

void ***pxTCB**

Address of task control block.

StackType_t ***pxTopOfStack**

Points to the location of the last item placed on the tasks stack.

StackType_t ***pxEndOfStack**

Points to the end of the stack. pxTopOfStack < pxEndOfStack, stack grows hi2lo pxTopOfStack > pxEndOfStack, stack grows lo2hi

Macros

tskKERNEL_VERSION_NUMBER

tskKERNEL_VERSION_MAJOR

tskKERNEL_VERSION_MINOR

tskKERNEL_VERSION_BUILD

tskNO_AFFINITY

Argument of xTaskCreatePinnedToCore indicating that task has no affinity.

tskIDLE_PRIORITY

Defines the priority used by the idle task. This must not be modified.

taskYIELD()

task. h

Macro for forcing a context switch.

taskENTER_CRITICAL(mux)

task. h

Macro to mark the start of a critical code region. Preemptive context switches cannot occur when in a critical region.

Note This may alter the stack (depending on the portable implementation) so must be used with care!

taskENTER_CRITICAL_ISR(mux)

taskEXIT_CRITICAL(mux)

task. h

Macro to mark the end of a critical code region. Preemptive context switches cannot occur when in a critical region.

Note This may alter the stack (depending on the portable implementation) so must be used with care!

taskEXIT_CRITICAL_ISR(mux)

taskDISABLE_INTERRUPTS()

task. h

Macro to disable all maskable interrupts.

taskENABLE_INTERRUPTS()

task. h

Macro to enable microcontroller interrupts.

taskSCHEDULER_SUSPENDED

taskSCHEDULER_NOT_STARTED

taskSCHEDULER_RUNNING

xTaskNotifyGive(xTaskToNotify)

Simplified macro for sending task notification.

configUSE_TASK_NOTIFICATIONS must be undefined or defined as 1 for this macro to be available.

When configUSE_TASK_NOTIFICATIONS is set to one each task has its own private "notification value", which is a 32-bit unsigned integer (uint32_t).

Events can be sent to a task using an intermediary object. Examples of such objects are queues, semaphores, mutexes and event groups. Task notifications are a method of sending an event directly to a task without the need for such an intermediary object.

A notification sent to a task can optionally perform an action, such as update, overwrite or increment the task's notification value. In that way task notifications can be used to send data to a task, or be used as light weight and fast binary or counting semaphores.

xTaskNotifyGive() is a helper macro intended for use when task notifications are used as light weight and faster binary or counting semaphore equivalents. Actual FreeRTOS semaphores are given using the xSemaphoreGive() API function, the equivalent action that instead uses a task notification is xTaskNotifyGive().

When task notifications are being used as a binary or counting semaphore equivalent then the task being notified should wait for the notification using the ulTaskNotificationTake() API function rather than the xTaskNotifyWait() API function.

See http://www.FreeRTOS.org/RTOS-task-notifications.html for more details.

Return xTaskNotifyGive() is a macro that calls xTaskNotify() with the eAction parameter set to eIncrement - so pdPASS is always returned.

Parameters

- **xTaskToNotify**: The handle of the task being notified. The handle to a task can be returned from the xTaskCreate() API function used to create the task, and the handle of the currently running task can be obtained by calling xTaskGetCurrentTaskHandle().

Type Definitions

typedef void ***TaskHandle_t**

task. h

Type by which tasks are referenced. For example, a call to xTaskCreate returns (via a pointer parameter) an TaskHandle_t variable that can then be used as a parameter to vTaskDelete to delete the task.

typedef BaseType_t (***TaskHookFunction_t**)(void *)

Defines the prototype to which the application task hook function must conform.

typedef struct *xTASK_STATUS* **TaskStatus_t**

Used with the uxTaskGetSystemState() function to return the state of each task in the system.

typedef struct *xTASK_SNAPSHOT* **TaskSnapshot_t**

Used with the uxTaskGetSnapshotAll() function to save memory snapshot of each task in the system. We need this struct because TCB_t is defined (hidden) in tasks.c.

typedef void (***TlsDeleteCallbackFunction_t**)(int, void *)

Prototype of local storage pointer deletion callback.

Enumerations

enum **eTaskState**

Task states returned by eTaskGetState.

Values:

eRunning = 0

A task is querying the state of itself, so must be running.

eReady

The task being queried is in a read or pending ready list.

eBlocked

The task being queried is in the Blocked state.

eSuspended

The task being queried is in the Suspended state, or is in the Blocked state with an infinite time out.

eDeleted

The task being queried has been deleted, but its TCB has not yet been freed.

enum **eNotifyAction**

Actions that can be performed when vTaskNotify() is called.

Values:

eNoAction = 0

Notify the task without updating its notify value.

eSetBits

Set bits in the task' s notification value.

eIncrement

Increment the task' s notification value.

eSetValueWithOverwrite

Set the task' s notification value to a specific value even if the previous value has not yet been read by the task.

eSetValueWithoutOverwrite

Set the task's notification value if the previous value has been read by the task.

enum eSleepModeStatus

Possible return values for eTaskConfirmSleepModeStatus().

Values:

eAbortSleep = 0

A task has been made ready or a context switch pended since portSUPPORESS_TICKS_AND_SLEEP() was called - abort entering a sleep mode.

eStandardSleep

Enter a sleep mode that will not last any longer than the expected idle time.

eNoTasksWaitingTimeout

No tasks are waiting for a timeout so it is safe to enter a sleep mode that can only be exited by an external interrupt.

Queue API

Header File

- freertos/include/freertos/queue.h

Functions

BaseType_t **xQueueGenericSendFromISR**(*QueueHandle_t* *xQueue*, **const** void *__const__ *pvItemToQueue*, BaseType_t *__const__ *pxHigherPriorityTaskWoken*, **const** BaseType_t *xCopyPosition*)

It is preferred that the macros xQueueSendFromISR(), xQueueSendToFrontFromISR() and xQueueSendToBackFromISR() be used in place of calling this function directly. xQueueGiveFromISR() is an equivalent for use by semaphores that don't actually copy any data.

Post an item on a queue. It is safe to use this function from within an interrupt service routine.

Items are queued by copy not reference so it is preferable to only queue small items, especially when called from an ISR. In most cases it would be preferable to store a pointer to the item being queued.

Example usage for buffered IO (where the ISR can obtain more than one value per call):

```
void vBufferISR( void )
{
char cIn;
BaseType_t xHigherPriorityTaskWokenByPost;
```

(下页继续)

(续上页)

```
    // We have not woken a task at the start of the ISR.
    xHigherPriorityTaskWokenByPost = pdFALSE;

    // Loop until the buffer is empty.
    do
    {
        // Obtain a byte from the buffer.
        cIn = portINPUT_BYTE( RX_REGISTER_ADDRESS );

        // Post each byte.
        xQueueGenericSendFromISR( xRxQueue, &cIn, &xHigherPriorityTaskWokenByPost,␣
↪queueSEND_TO_BACK );

    } while( portINPUT_BYTE( BUFFER_COUNT ) );

    // Now the buffer is empty we can switch context if necessary.  Note that the
    // name of the yield function required is port specific.
    if( xHigherPriorityTaskWokenByPost )
    {
        taskYIELD_YIELD_FROM_ISR();
    }
}
```

Return pdTRUE if the data was successfully sent to the queue, otherwise errQUEUE_FULL.

Parameters

- `xQueue`: The handle to the queue on which the item is to be posted.
- `pvItemToQueue`: A pointer to the item that is to be placed on the queue. The size of the items the queue will hold was defined when the queue was created, so this many bytes will be copied from pvItemToQueue into the queue storage area.
- `pxHigherPriorityTaskWoken`: xQueueGenericSendFromISR() will set *pxHigherPriorityTaskWoken to pdTRUE if sending to the queue caused a task to unblock, and the unblocked task has a priority higher than the currently running task. If xQueueGenericSendFromISR() sets this value to pdTRUE then a context switch should be requested before the interrupt is exited.
- `xCopyPosition`: Can take the value queueSEND_TO_BACK to place the item at the back of the queue, or queueSEND_TO_FRONT to place the item at the front of the queue (for high priority messages).

BaseType_t **xQueueGiveFromISR**(*QueueHandle_t xQueue*, BaseType_t *const *pxHigherPriorityTaskWoken*)

BaseType_t **xQueueIsQueueEmptyFromISR**(const *QueueHandle_t xQueue*)

Utilities to query queues that are safe to use from an ISR. These utilities should be used only from witin an ISR, or within a critical section.

BaseType_t **xQueueIsQueueFullFromISR**(const *QueueHandle_t xQueue*)

UBaseType_t **uxQueueMessagesWaitingFromISR**(const *QueueHandle_t xQueue*)

BaseType_t **xQueueGenericSend**(*QueueHandle_t xQueue*, const void *const *pvItemToQueue*, TickType_t *xTicksToWait*, const BaseType_t *xCopyPosition*)

It is preferred that the macros xQueueSend(), xQueueSendToFront() and xQueueSendToBack() are used in place of calling this function directly.

Post an item on a queue. The item is queued by copy, not by reference. This function must not be called from an interrupt service routine. See xQueueSendFromISR () for an alternative which may be used in an ISR.

Example usage:

```
struct AMessage
{
char ucMessageID;
char ucData[ 20 ];
} xMessage;

uint32_t ulVar = 10UL;

void vATask( void *pvParameters )
{
QueueHandle_t xQueue1, xQueue2;
struct AMessage *pxMessage;

// Create a queue capable of containing 10 uint32_t values.
xQueue1 = xQueueCreate( 10, sizeof( uint32_t ) );

// Create a queue capable of containing 10 pointers to AMessage structures.
// These should be passed by pointer as they contain a lot of data.
xQueue2 = xQueueCreate( 10, sizeof( struct AMessage * ) );

// ...

if( xQueue1 != 0 )
{
```

(下页继续)

(续上页)

```
    // Send an uint32_t.  Wait for 10 ticks for space to become
    // available if necessary.
    if( xQueueGenericSend( xQueue1, ( void * ) &ulVar, ( TickType_t ) 10, queueSEND_
↪TO_BACK ) != pdPASS )
    {
        // Failed to post the message, even after 10 ticks.
    }
}

if( xQueue2 != 0 )
{
    // Send a pointer to a struct AMessage object.  Don't block if the
    // queue is already full.
    pxMessage = & xMessage;
    xQueueGenericSend( xQueue2, ( void * ) &pxMessage, ( TickType_t ) 0, queueSEND_
↪TO_BACK );
}

// ... Rest of task code.
}
```

Return pdTRUE if the item was successfully posted, otherwise errQUEUE_FULL.

Parameters

- `xQueue`: The handle to the queue on which the item is to be posted.
- `pvItemToQueue`: A pointer to the item that is to be placed on the queue. The size of the items the queue will hold was defined when the queue was created, so this many bytes will be copied from pvItemToQueue into the queue storage area.
- `xTicksToWait`: The maximum amount of time the task should block waiting for space to become available on the queue, should it already be full. The call will return immediately if this is set to 0 and the queue is full. The time is defined in tick periods so the constant portTICK_PERIOD_MS should be used to convert to real time if this is required.
- `xCopyPosition`: Can take the value queueSEND_TO_BACK to place the item at the back of the queue, or queueSEND_TO_FRONT to place the item at the front of the queue (for high priority messages).

BaseType_t **xQueuePeekFromISR**(*QueueHandle_t* *xQueue*, void ***const** *pvBuffer*)

A version of xQueuePeek() that can be called from an interrupt service routine (ISR).

Receive an item from a queue without removing the item from the queue. The item is received by copy

so a buffer of adequate size must be provided. The number of bytes copied into the buffer was defined when the queue was created.

Successfully received items remain on the queue so will be returned again by the next call, or a call to xQueueReceive().

Return pdTRUE if an item was successfully received from the queue, otherwise pdFALSE.

Parameters

- `xQueue`: The handle to the queue from which the item is to be received.
- `pvBuffer`: Pointer to the buffer into which the received item will be copied.

BaseType_t **xQueueGenericReceive**(*QueueHandle_t* *xQueue*, void *const *pvBuffer*, TickType_t *xTicksToWait*, const BaseType_t *xJustPeek*)

It is preferred that the macro xQueueReceive() be used rather than calling this function directly.

Receive an item from a queue. The item is received by copy so a buffer of adequate size must be provided. The number of bytes copied into the buffer was defined when the queue was created.

This function must not be used in an interrupt service routine. See xQueueReceiveFromISR for an alternative that can.

Example usage:

```
struct AMessage
{
   char ucMessageID;
   char ucData[ 20 ];
} xMessage;

QueueHandle_t xQueue;

// Task to create a queue and post a value.
void vATask( void *pvParameters )
{
struct AMessage *pxMessage;

   // Create a queue capable of containing 10 pointers to AMessage structures.
   // These should be passed by pointer as they contain a lot of data.
   xQueue = xQueueCreate( 10, sizeof( struct AMessage * ) );
   if( xQueue == 0 )
   {
       // Failed to create the queue.
   }
```

(下页继续)

(续上页)

```
    // ...

    // Send a pointer to a struct AMessage object.  Don't block if the
    // queue is already full.
    pxMessage = & xMessage;
    xQueueSend( xQueue, ( void * ) &pxMessage, ( TickType_t ) 0 );

    // ... Rest of task code.
}

// Task to receive from the queue.
void vADifferentTask( void *pvParameters )
{
struct AMessage *pxRxedMessage;

    if( xQueue != 0 )
    {
        // Receive a message on the created queue.  Block for 10 ticks if a
        // message is not immediately available.
        if( xQueueGenericReceive( xQueue, &( pxRxedMessage ), ( TickType_t ) 10 ) )
        {
            // pcRxedMessage now points to the struct AMessage variable posted
            // by vATask.
        }
    }

    // ... Rest of task code.
}
```

Return pdTRUE if an item was successfully received from the queue, otherwise pdFALSE.

Parameters

- `xQueue`: The handle to the queue from which the item is to be received.
- `pvBuffer`: Pointer to the buffer into which the received item will be copied.
- `xTicksToWait`: The maximum amount of time the task should block waiting for an item to receive should the queue be empty at the time of the call. The time is defined in tick periods so the constant portTICK_PERIOD_MS should be used to convert to real time if this is required. xQueueGenericReceive() will return immediately if the queue is empty and xTicksToWait is 0.

- `xJustPeek`: When set to true, the item received from the queue is not actually removed from the queue - meaning a subsequent call to xQueueReceive() will return the same item. When set to false, the item being received from the queue is also removed from the queue.

UBaseType_t **uxQueueMessagesWaiting**(**const** *QueueHandle_t xQueue*)

Return the number of messages stored in a queue.

Return The number of messages available in the queue.

Parameters

- `xQueue`: A handle to the queue being queried.

UBaseType_t **uxQueueSpacesAvailable**(**const** *QueueHandle_t xQueue*)

Return the number of free spaces available in a queue. This is equal to the number of items that can be sent to the queue before the queue becomes full if no items are removed.

Return The number of spaces available in the queue.

Parameters

- `xQueue`: A handle to the queue being queried.

void **vQueueDelete**(*QueueHandle_t xQueue*)

Delete a queue - freeing all the memory allocated for storing of items placed on the queue.

Parameters

- `xQueue`: A handle to the queue to be deleted.

BaseType_t **xQueueReceiveFromISR**(*QueueHandle_t xQueue*, void ***const** *pvBuffer*, BaseType_t ***const** *pxHigherPriorityTaskWoken*)

Receive an item from a queue. It is safe to use this function from within an interrupt service routine.

Example usage:

```
QueueHandle_t xQueue;

// Function to create a queue and post some values.
void vAFunction( void *pvParameters )
{
char cValueToPost;
const TickType_t xTicksToWait = ( TickType_t )0xff;

   // Create a queue capable of containing 10 characters.
   xQueue = xQueueCreate( 10, sizeof( char ) );
   if( xQueue == 0 )
```

(下页继续)

(续上页)

```
    {
        // Failed to create the queue.
    }

    // ...

    // Post some characters that will be used within an ISR.  If the queue
    // is full then this task will block for xTicksToWait ticks.
    cValueToPost = 'a';
    xQueueSend( xQueue, ( void * ) &cValueToPost, xTicksToWait );
    cValueToPost = 'b';
    xQueueSend( xQueue, ( void * ) &cValueToPost, xTicksToWait );

    // ... keep posting characters ... this task may block when the queue
    // becomes full.

    cValueToPost = 'c';
    xQueueSend( xQueue, ( void * ) &cValueToPost, xTicksToWait );
}

// ISR that outputs all the characters received on the queue.
void vISR_Routine( void )
{
BaseType_t xTaskWokenByReceive = pdFALSE;
char cRxedChar;

    while( xQueueReceiveFromISR( xQueue, ( void * ) &cRxedChar, &
↪xTaskWokenByReceive) )
    {
        // A character was received.  Output the character now.
        vOutputCharacter( cRxedChar );

        // If removing the character from the queue woke the task that was
        // posting onto the queue cTaskWokenByReceive will have been set to
        // pdTRUE.  No matter how many times this loop iterates only one
        // task will be woken.
    }

    if( cTaskWokenByPost != ( char ) pdFALSE;
    {
```

(下页继续)

(续上页)

```
        taskYIELD ();
    }
}
```

Return pdTRUE if an item was successfully received from the queue, otherwise pdFALSE.

Parameters

- `xQueue`: The handle to the queue from which the item is to be received.
- `pvBuffer`: Pointer to the buffer into which the received item will be copied.
- `pxHigherPriorityTaskWoken`: A task may be blocked waiting for space to become available on the queue. If xQueueReceiveFromISR causes such a task to unblock *pxTaskWoken will get set to pdTRUE, otherwise *pxTaskWoken will remain unchanged.

void **vQueueAddToRegistry**(*QueueHandle_t xQueue*, **const** char **pcName*)

The registry is provided as a means for kernel aware debuggers to locate queues, semaphores and mutexes. Call vQueueAddToRegistry() add a queue, semaphore or mutex handle to the registry if you want the handle to be available to a kernel aware debugger. If you are not using a kernel aware debugger then this function can be ignored.

configQUEUE_REGISTRY_SIZE defines the maximum number of handles the registry can hold. configQUEUE_REGISTRY_SIZE must be greater than 0 within FreeRTOSConfig.h for the registry to be available. Its value does not effect the number of queues, semaphores and mutexes that can be created - just the number that the registry can hold.

Parameters

- `xQueue`: The handle of the queue being added to the registry. This is the handle returned by a call to xQueueCreate(). Semaphore and mutex handles can also be passed in here.
- `pcName`: The name to be associated with the handle. This is the name that the kernel aware debugger will display. The queue registry only stores a pointer to the string - so the string must be persistent (global or preferably in ROM/Flash), not on the stack.

void **vQueueUnregisterQueue**(*QueueHandle_t xQueue*)

The registry is provided as a means for kernel aware debuggers to locate queues, semaphores and mutexes. Call vQueueAddToRegistry() add a queue, semaphore or mutex handle to the registry if you want the handle to be available to a kernel aware debugger, and vQueueUnregisterQueue() to remove the queue, semaphore or mutex from the register. If you are not using a kernel aware debugger then this function can be ignored.

Parameters

- `xQueue`: The handle of the queue being removed from the registry.

`const` char *`pcQueueGetName`(*QueueHandle_t xQueue*)

The queue registry is provided as a means for kernel aware debuggers to locate queues, semaphores and mutexes. Call pcQueueGetName() to look up and return the name of a queue in the queue registry from the queue' s handle.

Note This function has been back ported from FreeRTOS v9.0.0

Return If the queue is in the registry then a pointer to the name of the queue is returned. If the queue is not in the registry then NULL is returned.

Parameters

- `xQueue`: The handle of the queue the name of which will be returned.

QueueHandle_t `xQueueGenericCreate`(`const` UBaseType_t *uxQueueLength*, `const` UBaseType_t *uxItemSize*, `const` uint8_t *ucQueueType*)

Generic version of the function used to creaet a queue using dynamic memory allocation. This is called by other functions and macros that create other RTOS objects that use the queue structure as their base.

QueueHandle_t `xQueueGenericCreateStatic`(`const` UBaseType_t *uxQueueLength*, `const` UBaseType_t *uxItemSize*, uint8_t **pucQueueStorage*, StaticQueue_t **pxStaticQueue*, `const` uint8_t *ucQueueType*)

Generic version of the function used to creaet a queue using dynamic memory allocation. This is called by other functions and macros that create other RTOS objects that use the queue structure as their base.

QueueSetHandle_t `xQueueCreateSet`(`const` UBaseType_t *uxEventQueueLength*)

Queue sets provide a mechanism to allow a task to block (pend) on a read operation from multiple queues or semaphores simultaneously.

See FreeRTOS/Source/Demo/Common/Minimal/QueueSet.c for an example using this function.

A queue set must be explicitly created using a call to xQueueCreateSet() before it can be used. Once created, standard FreeRTOS queues and semaphores can be added to the set using calls to xQueueAddToSet(). xQueueSelectFromSet() is then used to determine which, if any, of the queues or semaphores contained in the set is in a state where a queue read or semaphore take operation would be successful.

Note 1: See the documentation on http://wwwFreeRTOS.org/RTOS-queue-sets.html for reasons why queue sets are very rarely needed in practice as there are simpler methods of blocking on multiple objects.

Note 2: Blocking on a queue set that contains a mutex will not cause the mutex holder to inherit the priority of the blocked task.

Note 3: An additional 4 bytes of RAM is required for each space in a every queue added to a queue

set. Therefore counting semaphores that have a high maximum count value should not be added to a queue set.

Note 4: A receive (in the case of a queue) or take (in the case of a semaphore) operation must not be performed on a member of a queue set unless a call to xQueueSelectFromSet() has first returned a handle to that set member.

Return If the queue set is created successfully then a handle to the created queue set is returned. Otherwise NULL is returned.

Parameters

- `uxEventQueueLength`: Queue sets store events that occur on the queues and semaphores contained in the set. uxEventQueueLength specifies the maximum number of events that can be queued at once. To be absolutely certain that events are not lost uxEventQueueLength should be set to the total sum of the length of the queues added to the set, where binary semaphores and mutexes have a length of 1, and counting semaphores have a length set by their maximum count value. Examples:
  - If a queue set is to hold a queue of length 5, another queue of length 12, and a binary semaphore, then uxEventQueueLength should be set to (5 + 12 + 1), or 18.
  - If a queue set is to hold three binary semaphores then uxEventQueueLength should be set to (1 + 1 + 1), or 3.
  - If a queue set is to hold a counting semaphore that has a maximum count of 5, and a counting semaphore that has a maximum count of 3, then uxEventQueueLength should be set to (5 + 3), or 8.

BaseType_t **xQueueAddToSet**(*QueueSetMemberHandle_t xQueueOrSemaphore*, *QueueSetHandle_t xQueueSet*)

Adds a queue or semaphore to a queue set that was previously created by a call to xQueueCreateSet().

See FreeRTOS/Source/Demo/Common/Minimal/QueueSet.c for an example using this function.

Note 1: A receive (in the case of a queue) or take (in the case of a semaphore) operation must not be performed on a member of a queue set unless a call to xQueueSelectFromSet() has first returned a handle to that set member.

Return If the queue or semaphore was successfully added to the queue set then pdPASS is returned. If the queue could not be successfully added to the queue set because it is already a member of a different queue set then pdFAIL is returned.

Parameters

- `xQueueOrSemaphore`: The handle of the queue or semaphore being added to the queue set (cast to an QueueSetMemberHandle_t type).
- `xQueueSet`: The handle of the queue set to which the queue or semaphore is being added.

BaseType_t **xQueueRemoveFromSet**(*QueueSetMemberHandle_t xQueueOrSemaphore*, *QueueSetHandle_t xQueueSet*)

Removes a queue or semaphore from a queue set. A queue or semaphore can only be removed from a set if the queue or semaphore is empty.

See FreeRTOS/Source/Demo/Common/Minimal/QueueSet.c for an example using this function.

Return If the queue or semaphore was successfully removed from the queue set then pdPASS is returned. If the queue was not in the queue set, or the queue (or semaphore) was not empty, then pdFAIL is returned.

Parameters

- `xQueueOrSemaphore`: The handle of the queue or semaphore being removed from the queue set (cast to an QueueSetMemberHandle_t type).
- `xQueueSet`: The handle of the queue set in which the queue or semaphore is included.

QueueSetMemberHandle_t **xQueueSelectFromSet**(*QueueSetHandle_t xQueueSet*, **const** TickType_t *xTicksToWait*)

xQueueSelectFromSet() selects from the members of a queue set a queue or semaphore that either contains data (in the case of a queue) or is available to take (in the case of a semaphore). xQueueSelectFromSet() effectively allows a task to block (pend) on a read operation on all the queues and semaphores in a queue set simultaneously.

See FreeRTOS/Source/Demo/Common/Minimal/QueueSet.c for an example using this function.

Note 1: See the documentation on http://wwwFreeRTOS.org/RTOS-queue-sets.html for reasons why queue sets are very rarely needed in practice as there are simpler methods of blocking on multiple objects.

Note 2: Blocking on a queue set that contains a mutex will not cause the mutex holder to inherit the priority of the blocked task.

Note 3: A receive (in the case of a queue) or take (in the case of a semaphore) operation must not be performed on a member of a queue set unless a call to xQueueSelectFromSet() has first returned a handle to that set member.

Return xQueueSelectFromSet() will return the handle of a queue (cast to a QueueSetMemberHandle_t type) contained in the queue set that contains data, or the handle of a semaphore (cast to a QueueSetMemberHandle_t type) contained in the queue set that is available, or NULL if no such queue or semaphore exists before before the specified block time expires.

Parameters

- `xQueueSet`: The queue set on which the task will (potentially) block.
- `xTicksToWait`: The maximum time, in ticks, that the calling task will remain in the Blocked state (with other tasks executing) to wait for a member of the queue set to be ready for a successful queue read or semaphore take operation.

QueueSetMemberHandle_t **xQueueSelectFromSetFromISR**(*QueueSetHandle_t xQueueSet*)

A version of xQueueSelectFromSet() that can be used from an ISR.

Macros

xQueueCreate(uxQueueLength, uxItemSize)

Creates a new queue instance. This allocates the storage required by the new queue and returns a handle for the queue.

Example usage:

```
struct AMessage
{
char ucMessageID;
char ucData[ 20 ];
};

void vATask( void *pvParameters )
{
QueueHandle_t xQueue1, xQueue2;

// Create a queue capable of containing 10 uint32_t values.
xQueue1 = xQueueCreate( 10, sizeof( uint32_t ) );
if( xQueue1 == 0 )
{
    // Queue was not created and must not be used.
}

// Create a queue capable of containing 10 pointers to AMessage structures.
// These should be passed by pointer as they contain a lot of data.
xQueue2 = xQueueCreate( 10, sizeof( struct AMessage * ) );
if( xQueue2 == 0 )
{
    // Queue was not created and must not be used.
}

// ... Rest of task code.
}
```

Return If the queue is successfully create then a handle to the newly created queue is returned. If the queue cannot be created then 0 is returned.

Parameters

- `uxQueueLength`: The maximum number of items that the queue can contain.
- `uxItemSize`: The number of bytes each item in the queue will require. Items are queued by copy, not by reference, so this is the number of bytes that will be copied for each posted item. Each item on the queue must be the same size.

`xQueueCreateStatic`(uxQueueLength, uxItemSize, pucQueueStorage, pxQueueBuffer)

Creates a new queue instance, and returns a handle by which the new queue can be referenced.

Internally, within the FreeRTOS implementation, queues use two blocks of memory. The first block is used to hold the queue' s data structures. The second block is used to hold items placed into the queue. If a queue is created using xQueueCreate() then both blocks of memory are automatically dynamically allocated inside the xQueueCreate() function. (see http://www.freertos.org/a00111.html). If a queue is created using xQueueCreateStatic() then the application writer must provide the memory that will get used by the queue. xQueueCreateStatic() therefore allows a queue to be created without using any dynamic memory allocation.

http://www.FreeRTOS.org/Embedded-RTOS-Queues.html

Example usage:

```
struct AMessage
{
 char ucMessageID;
 char ucData[ 20 ];
};

#define QUEUE_LENGTH 10
#define ITEM_SIZE sizeof( uint32_t )

// xQueueBuffer will hold the queue structure.
StaticQueue_t xQueueBuffer;

// ucQueueStorage will hold the items posted to the queue.  Must be at least
// [(queue length) * ( queue item size)] bytes long.
uint8_t ucQueueStorage[ QUEUE_LENGTH * ITEM_SIZE ];

void vATask( void *pvParameters )
{
 QueueHandle_t xQueue1;

 // Create a queue capable of containing 10 uint32_t values.
 xQueue1 = xQueueCreate( QUEUE_LENGTH, // The number of items the queue can hold.
                         ITEM_SIZE     // The size of each item in the queue
```

(下页继续)

(续上页)

```
                        &( ucQueueStorage[ 0 ] ), // The buffer that will hold the␣
↪items in the queue.
                        &xQueueBuffer ); // The buffer that will hold the queue␣
↪structure.

 // The queue is guaranteed to be created successfully as no dynamic memory
 // allocation is used.  Therefore xQueue1 is now a handle to a valid queue.

 // ... Rest of task code.
}
```

Return If the queue is created then a handle to the created queue is returned. If pxQueueBuffer is NULL then NULL is returned.

Parameters

- `uxQueueLength`: The maximum number of items that the queue can contain.
- `uxItemSize`: The number of bytes each item in the queue will require. Items are queued by copy, not by reference, so this is the number of bytes that will be copied for each posted item. Each item on the queue must be the same size.
- `pucQueueStorage`: If uxItemSize is not zero then pucQueueStorageBuffer must point to a uint8_t array that is at least large enough to hold the maximum number of items that can be in the queue at any one time - which is (uxQueueLength * uxItemsSize) bytes. If uxItemSize is zero then pucQueueStorageBuffer can be NULL.
- `pxQueueBuffer`: Must point to a variable of type StaticQueue_t, which will be used to hold the queue' s data structure.

`xQueueSendToFront`(xQueue, pvItemToQueue, xTicksToWait)

This is a macro that calls xQueueGenericSend().

Post an item to the front of a queue. The item is queued by copy, not by reference. This function must not be called from an interrupt service routine. See xQueueSendFromISR () for an alternative which may be used in an ISR.

Example usage:

```
struct AMessage
{
char ucMessageID;
char ucData[ 20 ];
} xMessage;
```

(下页继续)

(续上页)

```
uint32_t ulVar = 10UL;

void vATask( void *pvParameters )
{
QueueHandle_t xQueue1, xQueue2;
struct AMessage *pxMessage;

// Create a queue capable of containing 10 uint32_t values.
xQueue1 = xQueueCreate( 10, sizeof( uint32_t ) );

// Create a queue capable of containing 10 pointers to AMessage structures.
// These should be passed by pointer as they contain a lot of data.
xQueue2 = xQueueCreate( 10, sizeof( struct AMessage * ) );

// ...

if( xQueue1 != 0 )
{
    // Send an uint32_t.  Wait for 10 ticks for space to become
    // available if necessary.
    if( xQueueSendToFront( xQueue1, ( void * ) &ulVar, ( TickType_t ) 10 ) !=␣
↪pdPASS )
    {
        // Failed to post the message, even after 10 ticks.
    }
}

if( xQueue2 != 0 )
{
    // Send a pointer to a struct AMessage object.  Don't block if the
    // queue is already full.
    pxMessage = & xMessage;
    xQueueSendToFront( xQueue2, ( void * ) &pxMessage, ( TickType_t ) 0 );
}

// ... Rest of task code.
}
```

Return pdTRUE if the item was successfully posted, otherwise errQUEUE_FULL.

Parameters

- `xQueue`: The handle to the queue on which the item is to be posted.
- `pvItemToQueue`: A pointer to the item that is to be placed on the queue. The size of the items the queue will hold was defined when the queue was created, so this many bytes will be copied from pvItemToQueue into the queue storage area.
- `xTicksToWait`: The maximum amount of time the task should block waiting for space to become available on the queue, should it already be full. The call will return immediately if this is set to 0 and the queue is full. The time is defined in tick periods so the constant portTICK_PERIOD_MS should be used to convert to real time if this is required.

xQueueSendToBack(xQueue, pvItemToQueue, xTicksToWait)

This is a macro that calls xQueueGenericSend().

Post an item to the back of a queue. The item is queued by copy, not by reference. This function must not be called from an interrupt service routine. See xQueueSendFromISR () for an alternative which may be used in an ISR.

Example usage:

```
struct AMessage
{
char ucMessageID;
char ucData[ 20 ];
} xMessage;

uint32_t ulVar = 10UL;

void vATask( void *pvParameters )
{
QueueHandle_t xQueue1, xQueue2;
struct AMessage *pxMessage;

// Create a queue capable of containing 10 uint32_t values.
xQueue1 = xQueueCreate( 10, sizeof( uint32_t ) );

// Create a queue capable of containing 10 pointers to AMessage structures.
// These should be passed by pointer as they contain a lot of data.
xQueue2 = xQueueCreate( 10, sizeof( struct AMessage * ) );

// ...
```

(下页继续)

(续上页)

```
if( xQueue1 != 0 )
{
    // Send an uint32_t.  Wait for 10 ticks for space to become
    // available if necessary.
    if( xQueueSendToBack( xQueue1, ( void * ) &ulVar, ( TickType_t ) 10 ) != pdPASS␣
↪)
    {
        // Failed to post the message, even after 10 ticks.
    }
}

if( xQueue2 != 0 )
{
    // Send a pointer to a struct AMessage object.  Don't block if the
    // queue is already full.
    pxMessage = & xMessage;
    xQueueSendToBack( xQueue2, ( void * ) &pxMessage, ( TickType_t ) 0 );
}

// ... Rest of task code.
}
```

Return pdTRUE if the item was successfully posted, otherwise errQUEUE_FULL.

Parameters

- `xQueue`: The handle to the queue on which the item is to be posted.
- `pvItemToQueue`: A pointer to the item that is to be placed on the queue. The size of the items the queue will hold was defined when the queue was created, so this many bytes will be copied from pvItemToQueue into the queue storage area.
- `xTicksToWait`: The maximum amount of time the task should block waiting for space to become available on the queue, should it already be full. The call will return immediately if this is set to 0 and the queue is full. The time is defined in tick periods so the constant portTICK_PERIOD_MS should be used to convert to real time if this is required.

xQueueSend(xQueue, pvItemToQueue, xTicksToWait)

This is a macro that calls xQueueGenericSend(). It is included for backward compatibility with versions of FreeRTOS.org that did not include the xQueueSendToFront() and xQueueSendToBack() macros. It is equivalent to xQueueSendToBack().

Post an item on a queue. The item is queued by copy, not by reference. This function must not be

called from an interrupt service routine. See xQueueSendFromISR () for an alternative which may be used in an ISR.

Example usage:

```
struct AMessage
{
char ucMessageID;
char ucData[ 20 ];
} xMessage;

uint32_t ulVar = 10UL;

void vATask( void *pvParameters )
{
QueueHandle_t xQueue1, xQueue2;
struct AMessage *pxMessage;

// Create a queue capable of containing 10 uint32_t values.
xQueue1 = xQueueCreate( 10, sizeof( uint32_t ) );

// Create a queue capable of containing 10 pointers to AMessage structures.
// These should be passed by pointer as they contain a lot of data.
xQueue2 = xQueueCreate( 10, sizeof( struct AMessage * ) );

// ...

if( xQueue1 != 0 )
{
    // Send an uint32_t.  Wait for 10 ticks for space to become
    // available if necessary.
    if( xQueueSend( xQueue1, ( void * ) &ulVar, ( TickType_t ) 10 ) != pdPASS )
    {
        // Failed to post the message, even after 10 ticks.
    }
}

if( xQueue2 != 0 )
{
    // Send a pointer to a struct AMessage object.  Don't block if the
    // queue is already full.
    pxMessage = & xMessage;
```

(下页继续)

(续上页)

```
    xQueueSend( xQueue2, ( void * ) &pxMessage, ( TickType_t ) 0 );
}

// ... Rest of task code.
}
```

Return pdTRUE if the item was successfully posted, otherwise errQUEUE_FULL.

Parameters

- `xQueue`: The handle to the queue on which the item is to be posted.
- `pvItemToQueue`: A pointer to the item that is to be placed on the queue. The size of the items the queue will hold was defined when the queue was created, so this many bytes will be copied from pvItemToQueue into the queue storage area.
- `xTicksToWait`: The maximum amount of time the task should block waiting for space to become available on the queue, should it already be full. The call will return immediately if this is set to 0 and the queue is full. The time is defined in tick periods so the constant portTICK_PERIOD_MS should be used to convert to real time if this is required.

`xQueueOverwrite`(xQueue, pvItemToQueue)

Only for use with queues that have a length of one - so the queue is either empty or full.

Post an item on a queue. If the queue is already full then overwrite the value held in the queue. The item is queued by copy, not by reference.

This function must not be called from an interrupt service routine. See xQueueOverwriteFromISR () for an alternative which may be used in an ISR.

Example usage:

```
void vFunction( void *pvParameters )
{
QueueHandle_t xQueue;
uint32_t ulVarToSend, ulValReceived;

// Create a queue to hold one uint32_t value.  It is strongly
// recommended *not* to use xQueueOverwrite() on queues that can
// contain more than one value, and doing so will trigger an assertion
// if configASSERT() is defined.
xQueue = xQueueCreate( 1, sizeof( uint32_t ) );

// Write the value 10 to the queue using xQueueOverwrite().
```

(下页继续)

(续上页)

```
 ulVarToSend = 10;
 xQueueOverwrite( xQueue, &ulVarToSend );

 // Peeking the queue should now return 10, but leave the value 10 in
 // the queue.  A block time of zero is used as it is known that the
 // queue holds a value.
 ulValReceived = 0;
 xQueuePeek( xQueue, &ulValReceived, 0 );

 if( ulValReceived != 10 )
 {
     // Error unless the item was removed by a different task.
 }

 // The queue is still full.  Use xQueueOverwrite() to overwrite the
 // value held in the queue with 100.
 ulVarToSend = 100;
 xQueueOverwrite( xQueue, &ulVarToSend );

 // This time read from the queue, leaving the queue empty once more.
 // A block time of 0 is used again.
 xQueueReceive( xQueue, &ulValReceived, 0 );

 // The value read should be the last value written, even though the
 // queue was already full when the value was written.
 if( ulValReceived != 100 )
 {
     // Error!
 }

 // ...
}
```

Return xQueueOverwrite() is a macro that calls xQueueGenericSend(), and therefore has the same return values as xQueueSendToFront(). However, pdPASS is the only value that can be returned because xQueueOverwrite() will write to the queue even when the queue is already full.

Parameters

- `xQueue`: The handle of the queue to which the data is being sent.
- `pvItemToQueue`: A pointer to the item that is to be placed on the queue. The size of the

items the queue will hold was defined when the queue was created, so this many bytes will be copied from pvItemToQueue into the queue storage area.

xQueuePeek(xQueue, pvBuffer, xTicksToWait)

This is a macro that calls the xQueueGenericReceive() function.

Receive an item from a queue without removing the item from the queue. The item is received by copy so a buffer of adequate size must be provided. The number of bytes copied into the buffer was defined when the queue was created.

Successfully received items remain on the queue so will be returned again by the next call, or a call to xQueueReceive().

This macro must not be used in an interrupt service routine. See xQueuePeekFromISR() for an alternative that can be called from an interrupt service routine.

Example usage:

```
struct AMessage
{
char ucMessageID;
char ucData[ 20 ];
} xMessage;

QueueHandle_t xQueue;

// Task to create a queue and post a value.
void vATask( void *pvParameters )
{
struct AMessage *pxMessage;

// Create a queue capable of containing 10 pointers to AMessage structures.
// These should be passed by pointer as they contain a lot of data.
xQueue = xQueueCreate( 10, sizeof( struct AMessage * ) );
if( xQueue == 0 )
{
    // Failed to create the queue.
}

// ...

// Send a pointer to a struct AMessage object.  Don't block if the
// queue is already full.
pxMessage = & xMessage;
```

(下页继续)

(续上页)

```
xQueueSend( xQueue, ( void * ) &pxMessage, ( TickType_t ) 0 );

// ... Rest of task code.
}

// Task to peek the data from the queue.
void vADifferentTask( void *pvParameters )
{
struct AMessage *pxRxedMessage;

if( xQueue != 0 )
{
    // Peek a message on the created queue.  Block for 10 ticks if a
    // message is not immediately available.
    if( xQueuePeek( xQueue, &( pxRxedMessage ), ( TickType_t ) 10 ) )
    {
        // pcRxedMessage now points to the struct AMessage variable posted
        // by vATask, but the item still remains on the queue.
    }
}

// ... Rest of task code.
}
```

Return pdTRUE if an item was successfully received from the queue, otherwise pdFALSE.

Parameters

- `xQueue`: The handle to the queue from which the item is to be received.
- `pvBuffer`: Pointer to the buffer into which the received item will be copied.
- `xTicksToWait`: The maximum amount of time the task should block waiting for an item to receive should the queue be empty at the time of the call. The time is defined in tick periods so the constant portTICK_PERIOD_MS should be used to convert to real time if this is required. xQueuePeek() will return immediately if xTicksToWait is 0 and the queue is empty.

xQueueReceive(xQueue, pvBuffer, xTicksToWait)

queue. h

This is a macro that calls the xQueueGenericReceive() function.

Receive an item from a queue. The item is received by copy so a buffer of adequate size must be provided. The number of bytes copied into the buffer was defined when the queue was created.

Successfully received items are removed from the queue.

This function must not be used in an interrupt service routine. See xQueueReceiveFromISR for an alternative that can.

Example usage:

```
struct AMessage
{
 char ucMessageID;
 char ucData[ 20 ];
} xMessage;

QueueHandle_t xQueue;

// Task to create a queue and post a value.
void vATask( void *pvParameters )
{
struct AMessage *pxMessage;

 // Create a queue capable of containing 10 pointers to AMessage structures.
 // These should be passed by pointer as they contain a lot of data.
 xQueue = xQueueCreate( 10, sizeof( struct AMessage * ) );
 if( xQueue == 0 )
 {
     // Failed to create the queue.
 }

 // ...

 // Send a pointer to a struct AMessage object.  Don't block if the
 // queue is already full.
 pxMessage = & xMessage;
 xQueueSend( xQueue, ( void * ) &pxMessage, ( TickType_t ) 0 );

 // ... Rest of task code.
}

// Task to receive from the queue.
void vADifferentTask( void *pvParameters )
{
struct AMessage *pxRxedMessage;
```

(下页继续)

(续上页)

```
 if( xQueue != 0 )
 {
     // Receive a message on the created queue.  Block for 10 ticks if a
     // message is not immediately available.
     if( xQueueReceive( xQueue, &( pxRxedMessage ), ( TickType_t ) 10 ) )
     {
         // pcRxedMessage now points to the struct AMessage variable posted
         // by vATask.
     }
 }

 // ... Rest of task code.
}
```

Return pdTRUE if an item was successfully received from the queue, otherwise pdFALSE.

Parameters

- `xQueue`: The handle to the queue from which the item is to be received.
- `pvBuffer`: Pointer to the buffer into which the received item will be copied.
- `xTicksToWait`: The maximum amount of time the task should block waiting for an item to receive should the queue be empty at the time of the call. xQueueReceive() will return immediately if xTicksToWait is zero and the queue is empty. The time is defined in tick periods so the constant portTICK_PERIOD_MS should be used to convert to real time if this is required.

`xQueueSendToFrontFromISR`(xQueue, pvItemToQueue, pxHigherPriorityTaskWoken)

This is a macro that calls xQueueGenericSendFromISR().

Post an item to the front of a queue. It is safe to use this macro from within an interrupt service routine.

Items are queued by copy not reference so it is preferable to only queue small items, especially when called from an ISR. In most cases it would be preferable to store a pointer to the item being queued.

Example usage for buffered IO (where the ISR can obtain more than one value per call):

```
void vBufferISR( void )
{
char cIn;
BaseType_t xHigherPrioritTaskWoken;
```

(下页继续)

(续上页)

```
    // We have not woken a task at the start of the ISR.
    xHigherPriorityTaskWoken = pdFALSE;

    // Loop until the buffer is empty.
    do
    {
        // Obtain a byte from the buffer.
        cIn = portINPUT_BYTE( RX_REGISTER_ADDRESS );

        // Post the byte.
        xQueueSendToFrontFromISR( xRxQueue, &cIn, &xHigherPriorityTaskWoken );

    } while( portINPUT_BYTE( BUFFER_COUNT ) );

    // Now the buffer is empty we can switch context if necessary.
    if( xHigherPriorityTaskWoken )
    {
        portYIELD_FROM_ISR ();
    }
}
```

Return pdTRUE if the data was successfully sent to the queue, otherwise errQUEUE_FULL.

Parameters

- `xQueue`: The handle to the queue on which the item is to be posted.
- `pvItemToQueue`: A pointer to the item that is to be placed on the queue. The size of the items the queue will hold was defined when the queue was created, so this many bytes will be copied from pvItemToQueue into the queue storage area.
- `pxHigherPriorityTaskWoken`: xQueueSendToFrontFromISR() will set *pxHigherPriorityTaskWoken to pdTRUE if sending to the queue caused a task to unblock, and the unblocked task has a priority higher than the currently running task. If xQueueSendToFromFromISR() sets this value to pdTRUE then a context switch should be requested before the interrupt is exited.

xQueueSendToBackFromISR(xQueue, pvItemToQueue, pxHigherPriorityTaskWoken)

This is a macro that calls xQueueGenericSendFromISR().

Post an item to the back of a queue. It is safe to use this macro from within an interrupt service routine.

Items are queued by copy not reference so it is preferable to only queue small items, especially when called from an ISR. In most cases it would be preferable to store a pointer to the item being queued.

Example usage for buffered IO (where the ISR can obtain more than one value per call):

```
void vBufferISR( void )
{
char cIn;
BaseType_t xHigherPriorityTaskWoken;

   // We have not woken a task at the start of the ISR.
   xHigherPriorityTaskWoken = pdFALSE;

   // Loop until the buffer is empty.
   do
   {
       // Obtain a byte from the buffer.
       cIn = portINPUT_BYTE( RX_REGISTER_ADDRESS );

       // Post the byte.
       xQueueSendToBackFromISR( xRxQueue, &cIn, &xHigherPriorityTaskWoken );

   } while( portINPUT_BYTE( BUFFER_COUNT ) );

   // Now the buffer is empty we can switch context if necessary.
   if( xHigherPriorityTaskWoken )
   {
       portYIELD_FROM_ISR ();
   }
}
```

Return pdTRUE if the data was successfully sent to the queue, otherwise errQUEUE_FULL.

Parameters

- `xQueue`: The handle to the queue on which the item is to be posted.
- `pvItemToQueue`: A pointer to the item that is to be placed on the queue. The size of the items the queue will hold was defined when the queue was created, so this many bytes will be copied from pvItemToQueue into the queue storage area.
- `pxHigherPriorityTaskWoken`: xQueueSendToBackFromISR() will set *pxHigherPriority-TaskWoken to pdTRUE if sending to the queue caused a task to unblock, and the unblocked task has a priority higher than the currently running task. If xQueueSendToBackFromISR() sets this value to pdTRUE then a context switch should be requested before the interrupt is

exited.

xQueueOverwriteFromISR(xQueue, pvItemToQueue, pxHigherPriorityTaskWoken)

A version of xQueueOverwrite() that can be used in an interrupt service routine (ISR).

Only for use with queues that can hold a single item - so the queue is either empty or full.

Post an item on a queue. If the queue is already full then overwrite the value held in the queue. The item is queued by copy, not by reference.

Example usage:

```
 QueueHandle_t xQueue;

 void vFunction( void *pvParameters )
 {
    // Create a queue to hold one uint32_t value.  It is strongly
    // recommended *not* to use xQueueOverwriteFromISR() on queues that can
    // contain more than one value, and doing so will trigger an assertion
    // if configASSERT() is defined.
    xQueue = xQueueCreate( 1, sizeof( uint32_t ) );
}

void vAnInterruptHandler( void )
{
// xHigherPriorityTaskWoken must be set to pdFALSE before it is used.
BaseType_t xHigherPriorityTaskWoken = pdFALSE;
uint32_t ulVarToSend, ulValReceived;

    // Write the value 10 to the queue using xQueueOverwriteFromISR().
    ulVarToSend = 10;
    xQueueOverwriteFromISR( xQueue, &ulVarToSend, &xHigherPriorityTaskWoken );

    // The queue is full, but calling xQueueOverwriteFromISR() again will still
    // pass because the value held in the queue will be overwritten with the
    // new value.
    ulVarToSend = 100;
    xQueueOverwriteFromISR( xQueue, &ulVarToSend, &xHigherPriorityTaskWoken );

    // Reading from the queue will now return 100.

    // ...
```

(下页继续)

(续上页)

```
    if( xHigherPrioritytaskWoken == pdTRUE )
    {
        // Writing to the queue caused a task to unblock and the unblocked task
        // has a priority higher than or equal to the priority of the currently
        // executing task (the task this interrupt interrupted).  Perform a context
        // switch so this interrupt returns directly to the unblocked task.
        portYIELD_FROM_ISR(); // or portEND_SWITCHING_ISR() depending on the port.
    }
}
```

Return xQueueOverwriteFromISR() is a macro that calls xQueueGenericSendFromISR(), and therefore has the same return values as xQueueSendToFrontFromISR(). However, pdPASS is the only value that can be returned because xQueueOverwriteFromISR() will write to the queue even when the queue is already full.

Parameters

- `xQueue`: The handle to the queue on which the item is to be posted.
- `pvItemToQueue`: A pointer to the item that is to be placed on the queue. The size of the items the queue will hold was defined when the queue was created, so this many bytes will be copied from pvItemToQueue into the queue storage area.
- `pxHigherPriorityTaskWoken`: xQueueOverwriteFromISR() will set *pxHigherPriorityTaskWoken to pdTRUE if sending to the queue caused a task to unblock, and the unblocked task has a priority higher than the currently running task. If xQueueOverwriteFromISR() sets this value to pdTRUE then a context switch should be requested before the interrupt is exited.

`xQueueSendFromISR`(xQueue, pvItemToQueue, pxHigherPriorityTaskWoken)

This is a macro that calls xQueueGenericSendFromISR(). It is included for backward compatibility with versions of FreeRTOS.org that did not include the xQueueSendToBackFromISR() and xQueueSendToFrontFromISR() macros.

Post an item to the back of a queue. It is safe to use this function from within an interrupt service routine.

Items are queued by copy not reference so it is preferable to only queue small items, especially when called from an ISR. In most cases it would be preferable to store a pointer to the item being queued.

Example usage for buffered IO (where the ISR can obtain more than one value per call):

```
void vBufferISR( void )
{
```

(下页继续)

(续上页)

```
char cIn;
BaseType_t xHigherPriorityTaskWoken;

   // We have not woken a task at the start of the ISR.
   xHigherPriorityTaskWoken = pdFALSE;

   // Loop until the buffer is empty.
   do
   {
       // Obtain a byte from the buffer.
       cIn = portINPUT_BYTE( RX_REGISTER_ADDRESS );

       // Post the byte.
       xQueueSendFromISR( xRxQueue, &cIn, &xHigherPriorityTaskWoken );

   } while( portINPUT_BYTE( BUFFER_COUNT ) );

   // Now the buffer is empty we can switch context if necessary.
   if( xHigherPriorityTaskWoken )
   {
       // Actual macro used here is port specific.
       portYIELD_FROM_ISR ();
   }
}
```

Return pdTRUE if the data was successfully sent to the queue, otherwise errQUEUE_FULL.

Parameters

- `xQueue`: The handle to the queue on which the item is to be posted.
- `pvItemToQueue`: A pointer to the item that is to be placed on the queue. The size of the items the queue will hold was defined when the queue was created, so this many bytes will be copied from pvItemToQueue into the queue storage area.
- `pxHigherPriorityTaskWoken`: xQueueSendFromISR() will set *pxHigherPriorityTaskWoken to pdTRUE if sending to the queue caused a task to unblock, and the unblocked task has a priority higher than the currently running task. If xQueueSendFromISR() sets this value to pdTRUE then a context switch should be requested before the interrupt is exited.

`xQueueReset`(xQueue)

Reset a queue back to its original empty state. pdPASS is returned if the queue is successfully reset. pdFAIL is returned if the queue could not be reset because there are tasks blocked on the queue waiting

to either receive from the queue or send to the queue.

Return always returns pdPASS

Parameters

- `xQueue`: The queue to reset

Type Definitions

`typedef` void *`QueueHandle_t`

Type by which queues are referenced. For example, a call to xQueueCreate() returns an QueueHandle_t variable that can then be used as a parameter to xQueueSend(), xQueueReceive(), etc.

`typedef` void *`QueueSetHandle_t`

Type by which queue sets are referenced. For example, a call to xQueueCreateSet() returns an xQueueSet variable that can then be used as a parameter to xQueueSelectFromSet(), xQueueAddToSet(), etc.

`typedef` void *`QueueSetMemberHandle_t`

Queue sets can contain both queues and semaphores, so the QueueSetMemberHandle_t is defined as a type to be used where a parameter or return value can be either an QueueHandle_t or an SemaphoreHandle_t.

Semaphore API

Header File

- freertos/include/freertos/semphr.h

Macros

`semBINARY_SEMAPHORE_QUEUE_LENGTH`

`semSEMAPHORE_QUEUE_ITEM_LENGTH`

`semGIVE_BLOCK_TIME`

`xSemaphoreCreateBinary()`

Creates a new binary semaphore instance, and returns a handle by which the new semaphore can be referenced.

In many usage scenarios it is faster and more memory efficient to use a direct to task notification in place of a binary semaphore! http://www.freertos.org/RTOS-task-notifications.html

Internally, within the FreeRTOS implementation, binary semaphores use a block of memory, in which the semaphore structure is stored. If a binary semaphore is created using xSemaphoreCreateBinary() then the required memory is automatically dynamically allocated inside the xSemaphoreCre-

ateBinary() function. (see http://www.freertos.org/a00111.html). If a binary semaphore is created using xSemaphoreCreateBinaryStatic() then the application writer must provide the memory. xSemaphoreCreateBinaryStatic() therefore allows a binary semaphore to be created without using any dynamic memory allocation.

The old vSemaphoreCreateBinary() macro is now deprecated in favour of this xSemaphoreCreateBinary() function. Note that binary semaphores created using the vSemaphoreCreateBinary() macro are created in a state such that the first call to 'take' the semaphore would pass, whereas binary semaphores created using xSemaphoreCreateBinary() are created in a state such that the the semaphore must first be 'given' before it can be 'taken'.

Function that creates a semaphore by using the existing queue mechanism. The queue length is 1 as this is a binary semaphore. The data size is 0 as nothing is actually stored - all that is important is whether the queue is empty or full (the binary semaphore is available or not).

This type of semaphore can be used for pure synchronisation between tasks or between an interrupt and a task. The semaphore need not be given back once obtained, so one task/interrupt can continuously 'give' the semaphore while another continuously 'takes' the semaphore. For this reason this type of semaphore does not use a priority inheritance mechanism. For an alternative that does use priority inheritance see xSemaphoreCreateMutex().

Example usage:

```
SemaphoreHandle_t xSemaphore = NULL;

void vATask( void * pvParameters )
{
   // Semaphore cannot be used before a call to vSemaphoreCreateBinary ().
   // This is a macro so pass the variable in directly.
   xSemaphore = xSemaphoreCreateBinary();

   if( xSemaphore != NULL )
   {
       // The semaphore was created successfully.
       // The semaphore can now be used.
   }
}
```

Return Handle to the created semaphore.

xSemaphoreCreateBinaryStatic(pxStaticSemaphore)

Creates a new binary semaphore instance, and returns a handle by which the new semaphore can be referenced.

NOTE: In many usage scenarios it is faster and more memory efficient to use a direct to task notification

in place of a binary semaphore! http://www.freertos.org/RTOS-task-notifications.html

Internally, within the FreeRTOS implementation, binary semaphores use a block of memory, in which the semaphore structure is stored. If a binary semaphore is created using xSemaphoreCreateBinary() then the required memory is automatically dynamically allocated inside the xSemaphoreCreateBinary() function. (see http://www.freertos.org/a00111.html). If a binary semaphore is created using xSemaphoreCreateBinaryStatic() then the application writer must provide the memory. xSemaphoreCreateBinaryStatic() therefore allows a binary semaphore to be created without using any dynamic memory allocation.

This type of semaphore can be used for pure synchronisation between tasks or between an interrupt and a task. The semaphore need not be given back once obtained, so one task/interrupt can continuously 'give' the semaphore while another continuously 'takes' the semaphore. For this reason this type of semaphore does not use a priority inheritance mechanism. For an alternative that does use priority inheritance see xSemaphoreCreateMutex().

Example usage:

```
SemaphoreHandle_t xSemaphore = NULL;
StaticSemaphore_t xSemaphoreBuffer;

void vATask( void * pvParameters )
{
   // Semaphore cannot be used before a call to xSemaphoreCreateBinary().
   // The semaphore's data structures will be placed in the xSemaphoreBuffer
   // variable, the address of which is passed into the function.  The
   // function's parameter is not NULL, so the function will not attempt any
   // dynamic memory allocation, and therefore the function will not return
   // return NULL.
   xSemaphore = xSemaphoreCreateBinary( &xSemaphoreBuffer );

   // Rest of task code goes here.
}
```

Return If the semaphore is created then a handle to the created semaphore is returned. If pxSemaphoreBuffer is NULL then NULL is returned.

Parameters

- `pxStaticSemaphore`: Must point to a variable of type StaticSemaphore_t, which will then be used to hold the semaphore' s data structure, removing the need for the memory to be allocated dynamically.

`xSemaphoreTake`(xSemaphore, xBlockTime)

Macro to obtain a semaphore. The semaphore must have previously been created with a call to

vSemaphoreCreateBinary(), xSemaphoreCreateMutex() or xSemaphoreCreateCounting().

Example usage:

```
SemaphoreHandle_t xSemaphore = NULL;

// A task that creates a semaphore.
void vATask( void * pvParameters )
{
   // Create the semaphore to guard a shared resource.
   vSemaphoreCreateBinary( xSemaphore );
}

// A task that uses the semaphore.
void vAnotherTask( void * pvParameters )
{
   // ... Do other things.

   if( xSemaphore != NULL )
   {
       // See if we can obtain the semaphore.  If the semaphore is not available
       // wait 10 ticks to see if it becomes free.
       if( xSemaphoreTake( xSemaphore, ( TickType_t ) 10 ) == pdTRUE )
       {
           // We were able to obtain the semaphore and can now access the
           // shared resource.

           // ...

           // We have finished accessing the shared resource.  Release the
           // semaphore.
           xSemaphoreGive( xSemaphore );
       }
       else
       {
           // We could not obtain the semaphore and can therefore not access
           // the shared resource safely.
       }
   }
}
```

Return pdTRUE if the semaphore was obtained. pdFALSE if xBlockTime expired without the

semaphore becoming available.

Parameters

- `xSemaphore`: A handle to the semaphore being taken - obtained when the semaphore was created.
- `xBlockTime`: The time in ticks to wait for the semaphore to become available. The macro portTICK_PERIOD_MS can be used to convert this to a real time. A block time of zero can be used to poll the semaphore. A block time of portMAX_DELAY can be used to block indefinitely (provided INCLUDE_vTaskSuspend is set to 1 in FreeRTOSConfig.h).

`xSemaphoreTakeRecursive`(xMutex, xBlockTime)

Macro to recursively obtain, or 'take', a mutex type semaphore. The mutex must have previously been created using a call to xSemaphoreCreateRecursiveMutex();

configUSE_RECURSIVE_MUTEXES must be set to 1 in FreeRTOSConfig.h for this macro to be available.

This macro must not be used on mutexes created using xSemaphoreCreateMutex().

A mutex used recursively can be 'taken' repeatedly by the owner. The mutex doesn't become available again until the owner has called xSemaphoreGiveRecursive() for each successful 'take' request. For example, if a task successfully 'takes' the same mutex 5 times then the mutex will not be available to any other task until it has also 'given' the mutex back exactly five times.

Example usage:

```
SemaphoreHandle_t xMutex = NULL;

// A task that creates a mutex.
void vATask( void * pvParameters )
{
   // Create the mutex to guard a shared resource.
   xMutex = xSemaphoreCreateRecursiveMutex();
}

// A task that uses the mutex.
void vAnotherTask( void * pvParameters )
{
   // ... Do other things.

   if( xMutex != NULL )
   {
       // See if we can obtain the mutex.  If the mutex is not available
       // wait 10 ticks to see if it becomes free.
```

(下页继续)

(续上页)

```
        if( xSemaphoreTakeRecursive( xSemaphore, ( TickType_t ) 10 ) == pdTRUE )
        {
            // We were able to obtain the mutex and can now access the
            // shared resource.

            // ...
            // For some reason due to the nature of the code further calls to
            // xSemaphoreTakeRecursive() are made on the same mutex.  In real
            // code these would not be just sequential calls as this would make
            // no sense.  Instead the calls are likely to be buried inside
            // a more complex call structure.
            xSemaphoreTakeRecursive( xMutex, ( TickType_t ) 10 );
            xSemaphoreTakeRecursive( xMutex, ( TickType_t ) 10 );

            // The mutex has now been 'taken' three times, so will not be
            // available to another task until it has also been given back
            // three times.  Again it is unlikely that real code would have
            // these calls sequentially, but instead buried in a more complex
            // call structure.  This is just for illustrative purposes.
            xSemaphoreGiveRecursive( xMutex );
            xSemaphoreGiveRecursive( xMutex );
            xSemaphoreGiveRecursive( xMutex );

            // Now the mutex can be taken by other tasks.
        }
        else
        {
            // We could not obtain the mutex and can therefore not access
            // the shared resource safely.
        }
    }
}
```

Return pdTRUE if the semaphore was obtained. pdFALSE if xBlockTime expired without the semaphore becoming available.

Parameters

- `xMutex`: A handle to the mutex being obtained. This is the handle returned by xSemaphoreCreateRecursiveMutex();
- `xBlockTime`: The time in ticks to wait for the semaphore to become available. The macro

portTICK_PERIOD_MS can be used to convert this to a real time. A block time of zero can be used to poll the semaphore. If the task already owns the semaphore then xSemaphoreTakeRecursive() will return immediately no matter what the value of xBlockTime.

xSemaphoreGive(xSemaphore)

Macro to release a semaphore. The semaphore must have previously been created with a call to vSemaphoreCreateBinary(), xSemaphoreCreateMutex() or xSemaphoreCreateCounting(). and obtained using sSemaphoreTake().

This macro must not be used from an ISR. See xSemaphoreGiveFromISR () for an alternative which can be used from an ISR.

This macro must also not be used on semaphores created using xSemaphoreCreateRecursiveMutex().

Example usage:

```
SemaphoreHandle_t xSemaphore = NULL;

void vATask( void * pvParameters )
{
   // Create the semaphore to guard a shared resource.
   vSemaphoreCreateBinary( xSemaphore );

   if( xSemaphore != NULL )
   {
       if( xSemaphoreGive( xSemaphore ) != pdTRUE )
       {
           // We would expect this call to fail because we cannot give
           // a semaphore without first "taking" it!
       }

       // Obtain the semaphore - don't block if the semaphore is not
       // immediately available.
       if( xSemaphoreTake( xSemaphore, ( TickType_t ) 0 ) )
       {
           // We now have the semaphore and can access the shared resource.

           // ...

           // We have finished accessing the shared resource so can free the
           // semaphore.
           if( xSemaphoreGive( xSemaphore ) != pdTRUE )
           {
```

(下页继续)

(续上页)

```
                // We would not expect this call to fail because we must have
                // obtained the semaphore to get here.
            }
        }
    }
}
```

Return pdTRUE if the semaphore was released. pdFALSE if an error occurred. Semaphores are implemented using queues. An error can occur if there is no space on the queue to post a message - indicating that the semaphore was not first obtained correctly.

Parameters

- `xSemaphore`: A handle to the semaphore being released. This is the handle returned when the semaphore was created.

`xSemaphoreGiveRecursive`(xMutex)

Macro to recursively release, or 'give', a mutex type semaphore. The mutex must have previously been created using a call to xSemaphoreCreateRecursiveMutex();

configUSE_RECURSIVE_MUTEXES must be set to 1 in FreeRTOSConfig.h for this macro to be available.

This macro must not be used on mutexes created using xSemaphoreCreateMutex().

A mutex used recursively can be 'taken' repeatedly by the owner. The mutex doesn't become available again until the owner has called xSemaphoreGiveRecursive() for each successful 'take' request. For example, if a task successfully 'takes' the same mutex 5 times then the mutex will not be available to any other task until it has also 'given' the mutex back exactly five times.

Example usage:

```
SemaphoreHandle_t xMutex = NULL;

// A task that creates a mutex.
void vATask( void * pvParameters )
{
   // Create the mutex to guard a shared resource.
   xMutex = xSemaphoreCreateRecursiveMutex();
}

// A task that uses the mutex.
void vAnotherTask( void * pvParameters )
{
```

(下页继续)

(续上页)

```
    // ... Do other things.

    if( xMutex != NULL )
    {
        // See if we can obtain the mutex.  If the mutex is not available
        // wait 10 ticks to see if it becomes free.
        if( xSemaphoreTakeRecursive( xMutex, ( TickType_t ) 10 ) == pdTRUE )
        {
            // We were able to obtain the mutex and can now access the
            // shared resource.

            // ...
            // For some reason due to the nature of the code further calls to
            // xSemaphoreTakeRecursive() are made on the same mutex.  In real
            // code these would not be just sequential calls as this would make
            // no sense.  Instead the calls are likely to be buried inside
            // a more complex call structure.
            xSemaphoreTakeRecursive( xMutex, ( TickType_t ) 10 );
            xSemaphoreTakeRecursive( xMutex, ( TickType_t ) 10 );

            // The mutex has now been 'taken' three times, so will not be
            // available to another task until it has also been given back
            // three times.  Again it is unlikely that real code would have
            // these calls sequentially, it would be more likely that the calls
            // to xSemaphoreGiveRecursive() would be called as a call stack
            // unwound.  This is just for demonstrative purposes.
            xSemaphoreGiveRecursive( xMutex );
            xSemaphoreGiveRecursive( xMutex );
            xSemaphoreGiveRecursive( xMutex );

            // Now the mutex can be taken by other tasks.
        }
        else
        {
            // We could not obtain the mutex and can therefore not access
            // the shared resource safely.
        }
    }
}
```

Return pdTRUE if the semaphore was given.

Parameters

- `xMutex`: A handle to the mutex being released, or 'given'. This is the handle returned by xSemaphoreCreateMutex();

`xSemaphoreGiveFromISR`(xSemaphore, pxHigherPriorityTaskWoken)

Macro to release a semaphore. The semaphore must have previously been created with a call to vSemaphoreCreateBinary() or xSemaphoreCreateCounting().

Mutex type semaphores (those created using a call to xSemaphoreCreateMutex()) must not be used with this macro.

This macro can be used from an ISR.

Example usage:

```
\#define LONG_TIME 0xffff
\#define TICKS_TO_WAIT 10
SemaphoreHandle_t xSemaphore = NULL;

// Repetitive task.
void vATask( void * pvParameters )
{
   for( ;; )
   {
       // We want this task to run every 10 ticks of a timer.  The semaphore
       // was created before this task was started.

       // Block waiting for the semaphore to become available.
       if( xSemaphoreTake( xSemaphore, LONG_TIME ) == pdTRUE )
       {
           // It is time to execute.

           // ...

           // We have finished our task.  Return to the top of the loop where
           // we will block on the semaphore until it is time to execute
           // again.  Note when using the semaphore for synchronisation with an
           // ISR in this manner there is no need to 'give' the semaphore back.
       }
   }
}
```

(下页继续)

(续上页)

```
// Timer ISR
void vTimerISR( void * pvParameters )
{
static uint8_t ucLocalTickCount = 0;
static BaseType_t xHigherPriorityTaskWoken;

   // A timer tick has occurred.

   // ... Do other time functions.

   // Is it time for vATask () to run?
   xHigherPriorityTaskWoken = pdFALSE;
   ucLocalTickCount++;
   if( ucLocalTickCount >= TICKS_TO_WAIT )
   {
       // Unblock the task by releasing the semaphore.
       xSemaphoreGiveFromISR( xSemaphore, &xHigherPriorityTaskWoken );

       // Reset the count so we release the semaphore again in 10 ticks time.
       ucLocalTickCount = 0;
   }

   if( xHigherPriorityTaskWoken != pdFALSE )
   {
       // We can force a context switch here.  Context switching from an
       // ISR uses port specific syntax.  Check the demo task for your port
       // to find the syntax required.
   }
}
```

Return pdTRUE if the semaphore was successfully given, otherwise errQUEUE_FULL.

Parameters

- `xSemaphore`: A handle to the semaphore being released. This is the handle returned when the semaphore was created.
- `pxHigherPriorityTaskWoken`: xSemaphoreGiveFromISR() will set *pxHigherPriorityTaskWoken to pdTRUE if giving the semaphore caused a task to unblock, and the unblocked task has a priority higher than the currently running task. If xSemaphoreGiveFromISR() sets this value to pdTRUE then a context switch should be requested before the interrupt is

exited.

`xSemaphoreTakeFromISR`(xSemaphore, pxHigherPriorityTaskWoken)

Macro to take a semaphore from an ISR. The semaphore must have previously been created with a call to vSemaphoreCreateBinary() or xSemaphoreCreateCounting().

Mutex type semaphores (those created using a call to xSemaphoreCreateMutex()) must not be used with this macro.

This macro can be used from an ISR, however taking a semaphore from an ISR is not a common operation. It is likely to only be useful when taking a counting semaphore when an interrupt is obtaining an object from a resource pool (when the semaphore count indicates the number of resources available).

Return pdTRUE if the semaphore was successfully taken, otherwise pdFALSE

Parameters

- `xSemaphore`: A handle to the semaphore being taken. This is the handle returned when the semaphore was created.
- `pxHigherPriorityTaskWoken`: xSemaphoreTakeFromISR() will set *pxHigherPriorityTaskWoken to pdTRUE if taking the semaphore caused a task to unblock, and the unblocked task has a priority higher than the currently running task. If xSemaphoreTakeFromISR() sets this value to pdTRUE then a context switch should be requested before the interrupt is exited.

`xSemaphoreCreateMutex()`

Macro that implements a mutex semaphore by using the existing queue mechanism.

Internally, within the FreeRTOS implementation, mutex semaphores use a block of memory, in which the mutex structure is stored. If a mutex is created using xSemaphoreCreateMutex() then the required memory is automatically dynamically allocated inside the xSemaphoreCreateMutex() function. (see http://www.freertos.org/a00111.html). If a mutex is created using xSemaphoreCreateMutexStatic() then the application writer must provided the memory. xSemaphoreCreateMutexStatic() therefore allows a mutex to be created without using any dynamic memory allocation.

Mutexes created using this function can be accessed using the xSemaphoreTake() and xSemaphoreGive() macros. The xSemaphoreTakeRecursive() and xSemaphoreGiveRecursive() macros must not be used.

This type of semaphore uses a priority inheritance mechanism so a task ‘taking’ a semaphore MUST ALWAYS ‘give’ the semaphore back once the semaphore it is no longer required.

Mutex type semaphores cannot be used from within interrupt service routines.

See vSemaphoreCreateBinary() for an alternative implementation that can be used for pure synchronisation (where one task or interrupt always ‘gives’ the semaphore and another always ‘takes’ the

semaphore) and from within interrupt service routines.

Example usage:

```
SemaphoreHandle_t xSemaphore;

void vATask( void * pvParameters )
{
   // Semaphore cannot be used before a call to xSemaphoreCreateMutex().
   // This is a macro so pass the variable in directly.
   xSemaphore = xSemaphoreCreateMutex();

   if( xSemaphore != NULL )
   {
       // The semaphore was created successfully.
       // The semaphore can now be used.
   }
}
```

Return If the mutex was successfully created then a handle to the created semaphore is returned. If there was not enough heap to allocate the mutex data structures then NULL is returned.

xSemaphoreCreateMutexStatic(pxMutexBuffer)

Creates a new mutex type semaphore instance, and returns a handle by which the new mutex can be referenced.

Internally, within the FreeRTOS implementation, mutex semaphores use a block of memory, in which the mutex structure is stored. If a mutex is created using xSemaphoreCreateMutex() then the required memory is automatically dynamically allocated inside the xSemaphoreCreateMutex() function. (see http://www.freertos.org/a00111.html). If a mutex is created using xSemaphoreCreateMutexStatic() then the application writer must provided the memory. xSemaphoreCreateMutexStatic() therefore allows a mutex to be created without using any dynamic memory allocation.

Mutexes created using this function can be accessed using the xSemaphoreTake() and xSemaphoreGive() macros. The xSemaphoreTakeRecursive() and xSemaphoreGiveRecursive() macros must not be used.

This type of semaphore uses a priority inheritance mechanism so a task ‘taking’ a semaphore MUST ALWAYS ‘give’ the semaphore back once the semaphore it is no longer required.

Mutex type semaphores cannot be used from within interrupt service routines.

See xSemaphoreCreateBinary() for an alternative implementation that can be used for pure synchronisation (where one task or interrupt always ‘gives’ the semaphore and another always ‘takes’ the semaphore) and from within interrupt service routines.

Example usage:

```
SemaphoreHandle_t xSemaphore;
StaticSemaphore_t xMutexBuffer;

void vATask( void * pvParameters )
{
   // A mutex cannot be used before it has been created.  xMutexBuffer is
   // into xSemaphoreCreateMutexStatic() so no dynamic memory allocation is
   // attempted.
   xSemaphore = xSemaphoreCreateMutexStatic( &xMutexBuffer );

   // As no dynamic memory allocation was performed, xSemaphore cannot be NULL,
   // so there is no need to check it.
}
```

Return If the mutex was successfully created then a handle to the created mutex is returned. If pxMutexBuffer was NULL then NULL is returned.

Parameters

- `pxMutexBuffer`: Must point to a variable of type StaticSemaphore_t, which will be used to hold the mutex' s data structure, removing the need for the memory to be allocated dynamically.

`xSemaphoreCreateRecursiveMutex()`

Creates a new recursive mutex type semaphore instance, and returns a handle by which the new recursive mutex can be referenced.

Internally, within the FreeRTOS implementation, recursive mutexs use a block of memory, in which the mutex structure is stored. If a recursive mutex is created using xSemaphoreCreateRecursiveMutex() then the required memory is automatically dynamically allocated inside the xSemaphoreCreateRecursiveMutex() function. (see http://www.freertos.org/a00111.html). If a recursive mutex is created using xSemaphoreCreateRecursiveMutexStatic() then the application writer must provide the memory that will get used by the mutex. xSemaphoreCreateRecursiveMutexStatic() therefore allows a recursive mutex to be created without using any dynamic memory allocation.

Mutexes created using this macro can be accessed using the xSemaphoreTakeRecursive() and xSemaphoreGiveRecursive() macros. The xSemaphoreTake() and xSemaphoreGive() macros must not be used.

A mutex used recursively can be 'taken' repeatedly by the owner. The mutex doesn't become available again until the owner has called xSemaphoreGiveRecursive() for each successful 'take' request. For example, if a task successfully 'takes' the same mutex 5 times then the mutex will not be available to any other task until it has also 'given' the mutex back exactly five times.

This type of semaphore uses a priority inheritance mechanism so a task 'taking' a semaphore MUST ALWAYS 'give' the semaphore back once the semaphore it is no longer required.

Mutex type semaphores cannot be used from within interrupt service routines.

See vSemaphoreCreateBinary() for an alternative implementation that can be used for pure synchronisation (where one task or interrupt always 'gives' the semaphore and another always 'takes' the semaphore) and from within interrupt service routines.

Example usage:

```
SemaphoreHandle_t xSemaphore;

void vATask( void * pvParameters )
{
   // Semaphore cannot be used before a call to xSemaphoreCreateMutex().
   // This is a macro so pass the variable in directly.
   xSemaphore = xSemaphoreCreateRecursiveMutex();

   if( xSemaphore != NULL )
   {
       // The semaphore was created successfully.
       // The semaphore can now be used.
   }
}
```

Return xSemaphore Handle to the created mutex semaphore. Should be of type SemaphoreHandle_t.

xSemaphoreCreateRecursiveMutexStatic(pxStaticSemaphore)

Creates a new recursive mutex type semaphore instance, and returns a handle by which the new recursive mutex can be referenced.

Internally, within the FreeRTOS implementation, recursive mutexs use a block of memory, in which the mutex structure is stored. If a recursive mutex is created using xSemaphoreCreateRecursiveMutex() then the required memory is automatically dynamically allocated inside the xSemaphoreCreateRecursiveMutex() function. (see http://www.freertos.org/a00111.html). If a recursive mutex is created using xSemaphoreCreateRecursiveMutexStatic() then the application writer must provide the memory that will get used by the mutex. xSemaphoreCreateRecursiveMutexStatic() therefore allows a recursive mutex to be created without using any dynamic memory allocation.

Mutexes created using this macro can be accessed using the xSemaphoreTakeRecursive() and xSemaphoreGiveRecursive() macros. The xSemaphoreTake() and xSemaphoreGive() macros must not be used.

A mutex used recursively can be 'taken' repeatedly by the owner. The mutex doesn't become available again until the owner has called xSemaphoreGiveRecursive() for each successful 'take' request. For

example, if a task successfully ‘takes’ the same mutex 5 times then the mutex will not be available to any other task until it has also ‘given’ the mutex back exactly five times.

This type of semaphore uses a priority inheritance mechanism so a task ‘taking’ a semaphore MUST ALWAYS ‘give’ the semaphore back once the semaphore it is no longer required.

Mutex type semaphores cannot be used from within interrupt service routines.

See xSemaphoreCreateBinary() for an alternative implementation that can be used for pure synchronisation (where one task or interrupt always ‘gives’ the semaphore and another always ‘takes’ the semaphore) and from within interrupt service routines.

Example usage:

```
SemaphoreHandle_t xSemaphore;
StaticSemaphore_t xMutexBuffer;

void vATask( void * pvParameters )
{
   // A recursive semaphore cannot be used before it is created.  Here a
   // recursive mutex is created using xSemaphoreCreateRecursiveMutexStatic().
   // The address of xMutexBuffer is passed into the function, and will hold
   // the mutexes data structures - so no dynamic memory allocation will be
   // attempted.
   xSemaphore = xSemaphoreCreateRecursiveMutexStatic( &xMutexBuffer );

   // As no dynamic memory allocation was performed, xSemaphore cannot be NULL,
   // so there is no need to check it.
}
```

Return If the recursive mutex was successfully created then a handle to the created recursive mutex is returned. If pxMutexBuffer was NULL then NULL is returned.

Parameters

- `pxStaticSemaphore`: Must point to a variable of type StaticSemaphore_t, which will then be used to hold the recursive mutex’ s data structure, removing the need for the memory to be allocated dynamically.

`xSemaphoreCreateCounting`(uxMaxCount, uxInitialCount)

Creates a new counting semaphore instance, and returns a handle by which the new counting semaphore can be referenced.

In many usage scenarios it is faster and more memory efficient to use a direct to task notification in place of a counting semaphore! http://www.freertos.org/RTOS-task-notifications.html

Internally, within the FreeRTOS implementation, counting semaphores use a block of memory, in which the counting semaphore structure is stored. If a counting semaphore is created using xSemaphoreCreateCounting() then the required memory is automatically dynamically allocated inside the xSemaphoreCreateCounting() function. (see http://www.freertos.org/a00111.html). If a counting semaphore is created using xSemaphoreCreateCountingStatic() then the application writer can instead optionally provide the memory that will get used by the counting semaphore. xSemaphoreCreateCountingStatic() therefore allows a counting semaphore to be created without using any dynamic memory allocation.

Counting semaphores are typically used for two things:

1) Counting events.

In this usage scenario an event handler will 'give' a semaphore each time an event occurs (incrementing the semaphore count value), and a handler task will 'take' a semaphore each time it processes an event (decrementing the semaphore count value). The count value is therefore the difference between the number of events that have occurred and the number that have been processed. In this case it is desirable for the initial count value to be zero.

2) Resource management.

In this usage scenario the count value indicates the number of resources available. To obtain control of a resource a task must first obtain a semaphore - decrementing the semaphore count value. When the count value reaches zero there are no free resources. When a task finishes with the resource it 'gives' the semaphore back - incrementing the semaphore count value. In this case it is desirable for the initial count value to be equal to the maximum count value, indicating that all resources are free.

Example usage:

```
SemaphoreHandle_t xSemaphore;

void vATask( void * pvParameters )
{
SemaphoreHandle_t xSemaphore = NULL;

   // Semaphore cannot be used before a call to xSemaphoreCreateCounting().
   // The max value to which the semaphore can count should be 10, and the
   // initial value assigned to the count should be 0.
   xSemaphore = xSemaphoreCreateCounting( 10, 0 );

   if( xSemaphore != NULL )
   {
       // The semaphore was created successfully.
       // The semaphore can now be used.
   }
```

(下页继续)

(续上页)

```
}
```

Return Handle to the created semaphore. Null if the semaphore could not be created.

Parameters

- `uxMaxCount`: The maximum count value that can be reached. When the semaphore reaches this value it can no longer be 'given'.
- `uxInitialCount`: The count value assigned to the semaphore when it is created.

`xSemaphoreCreateCountingStatic`(uxMaxCount, uxInitialCount, pxSemaphoreBuffer)

Creates a new counting semaphore instance, and returns a handle by which the new counting semaphore can be referenced.

In many usage scenarios it is faster and more memory efficient to use a direct to task notification in place of a counting semaphore! http://www.freertos.org/RTOS-task-notifications.html

Internally, within the FreeRTOS implementation, counting semaphores use a block of memory, in which the counting semaphore structure is stored. If a counting semaphore is created using xSemaphoreCreateCounting() then the required memory is automatically dynamically allocated inside the xSemaphoreCreateCounting() function. (see http://www.freertos.org/a00111.html). If a counting semaphore is created using xSemaphoreCreateCountingStatic() then the application writer must provide the memory. xSemaphoreCreateCountingStatic() therefore allows a counting semaphore to be created without using any dynamic memory allocation.

Counting semaphores are typically used for two things:

1) Counting events.

In this usage scenario an event handler will 'give' a semaphore each time an event occurs (incrementing the semaphore count value), and a handler task will 'take' a semaphore each time it processes an event (decrementing the semaphore count value). The count value is therefore the difference between the number of events that have occurred and the number that have been processed. In this case it is desirable for the initial count value to be zero.

2) Resource management.

In this usage scenario the count value indicates the number of resources available. To obtain control of a resource a task must first obtain a semaphore - decrementing the semaphore count value. When the count value reaches zero there are no free resources. When a task finishes with the resource it 'gives' the semaphore back - incrementing the semaphore count value. In this case it is desirable for the initial count value to be equal to the maximum count value, indicating that all resources are free.

Example usage:

```
SemaphoreHandle_t xSemaphore;
StaticSemaphore_t xSemaphoreBuffer;

void vATask( void * pvParameters )
{
SemaphoreHandle_t xSemaphore = NULL;

   // Counting semaphore cannot be used before they have been created.  Create
   // a counting semaphore using xSemaphoreCreateCountingStatic().  The max
   // value to which the semaphore can count is 10, and the initial value
   // assigned to the count will be 0.  The address of xSemaphoreBuffer is
   // passed in and will be used to hold the semaphore structure, so no dynamic
   // memory allocation will be used.
   xSemaphore = xSemaphoreCreateCounting( 10, 0, &xSemaphoreBuffer );

   // No memory allocation was attempted so xSemaphore cannot be NULL, so there
   // is no need to check its value.
}
```

Return If the counting semaphore was successfully created then a handle to the created counting semaphore is returned. If pxSemaphoreBuffer was NULL then NULL is returned.

Parameters

- `uxMaxCount`: The maximum count value that can be reached. When the semaphore reaches this value it can no longer be ‘given’.
- `uxInitialCount`: The count value assigned to the semaphore when it is created.
- `pxSemaphoreBuffer`: Must point to a variable of type StaticSemaphore_t, which will then be used to hold the semaphore’s data structure, removing the need for the memory to be allocated dynamically.

`vSemaphoreDelete`(xSemaphore)

Delete a semaphore. This function must be used with care. For example, do not delete a mutex type semaphore if the mutex is held by a task.

Parameters

- `xSemaphore`: A handle to the semaphore to be deleted.

`xSemaphoreGetMutexHolder`(xSemaphore)

If xMutex is indeed a mutex type semaphore, return the current mutex holder. If xMutex is not a mutex type semaphore, or the mutex is available (not held by a task), return NULL.

Note: This is a good way of determining if the calling task is the mutex holder, but not a good way of determining the identity of the mutex holder as the holder may change between the function exiting and the returned value being tested.

`uxSemaphoreGetCount`(xSemaphore)

If the semaphore is a counting semaphore then uxSemaphoreGetCount() returns its current count value. If the semaphore is a binary semaphore then uxSemaphoreGetCount() returns 1 if the semaphore is available, and 0 if the semaphore is not available.

Type Definitions

`typedef` *QueueHandle_t* `SemaphoreHandle_t`

Timer API

Header File

- freertos/include/freertos/timers.h

Functions

TimerHandle_t `xTimerCreate(const` char *`const` *pcTimerName*, `const` TickType_t *xTimerPeriodInTicks*, `const` UBaseType_t *uxAutoReload*, void *`const` *pvTimerID*, *TimerCallbackFunction_t pxCallbackFunction*)

Creates a new software timer instance, and returns a handle by which the created software timer can be referenced.

Internally, within the FreeRTOS implementation, software timers use a block of memory, in which the timer data structure is stored. If a software timer is created using xTimerCreate() then the required memory is automatically dynamically allocated inside the xTimerCreate() function. (see http://www.freertos.org/a00111.html). If a software timer is created using xTimerCreateStatic() then the application writer must provide the memory that will get used by the software timer. xTimerCreateStatic() therefore allows a software timer to be created without using any dynamic memory allocation.

Timers are created in the dormant state. The xTimerStart(), xTimerReset(), xTimerStartFromISR(), xTimerResetFromISR(), xTimerChangePeriod() and xTimerChangePeriodFromISR() API functions can all be used to transition a timer into the active state.

Example usage:

```
#define NUM_TIMERS 5
```

(下页继续)

(续上页)

```
// An array to hold handles to the created timers.
TimerHandle_t xTimers[ NUM_TIMERS ];

// An array to hold a count of the number of times each timer expires.
int32_t lExpireCounters[ NUM_TIMERS ] = { 0 };

// Define a callback function that will be used by multiple timer instances.
// The callback function does nothing but count the number of times the
// associated timer expires, and stop the timer once the timer has expired
// 10 times.
void vTimerCallback( TimerHandle_t pxTimer )
{
int32_t lArrayIndex;
const int32_t xMaxExpiryCountBeforeStopping = 10;

       // Optionally do something if the pxTimer parameter is NULL.
       configASSERT( pxTimer );

    // Which timer expired?
    lArrayIndex = ( int32_t ) pvTimerGetTimerID( pxTimer );

    // Increment the number of times that pxTimer has expired.
    lExpireCounters[ lArrayIndex ] += 1;

    // If the timer has expired 10 times then stop it from running.
    if( lExpireCounters[ lArrayIndex ] == xMaxExpiryCountBeforeStopping )
    {
        // Do not use a block time if calling a timer API function from a
        // timer callback function, as doing so could cause a deadlock!
        xTimerStop( pxTimer, 0 );
    }
}

void main( void )
{
int32_t x;

    // Create then start some timers.  Starting the timers before the scheduler
    // has been started means the timers will start running immediately that
    // the scheduler starts.
```

(下页继续)

(续上页)

```
    for( x = 0; x < NUM_TIMERS; x++ )
    {
        xTimers[ x ] = xTimerCreate(    "Timer",       // Just a text name, not␣
↪used by the kernel.
                                        ( 100 * x ),   // The timer period in ticks.
                                        pdTRUE,        // The timers will auto-
↪reload themselves when they expire.
                                        ( void * ) x,  // Assign each timer a␣
↪unique id equal to its array index.
                                        vTimerCallback // Each timer calls the same␣
↪callback when it expires.
                                    );

        if( xTimers[ x ] == NULL )
        {
            // The timer was not created.
        }
        else
        {
            // Start the timer.  No block time is specified, and even if one was
            // it would be ignored because the scheduler has not yet been
            // started.
            if( xTimerStart( xTimers[ x ], 0 ) != pdPASS )
            {
                // The timer could not be set into the Active state.
            }
        }
    }

    // ...
    // Create tasks here.
    // ...

    // Starting the scheduler will start the timers running as they have already
    // been set into the active state.
    vTaskStartScheduler();

    // Should not reach here.
    for( ;; );
}
```

Return If the timer is successfully created then a handle to the newly created timer is returned. If the timer cannot be created (because either there is insufficient FreeRTOS heap remaining to allocate the timer structures, or the timer period was set to 0) then NULL is returned.

Parameters

- `pcTimerName`: A text name that is assigned to the timer. This is done purely to assist debugging. The kernel itself only ever references a timer by its handle, and never by its name.
- `xTimerPeriodInTicks`: The timer period. The time is defined in tick periods so the constant portTICK_PERIOD_MS can be used to convert a time that has been specified in milliseconds. For example, if the timer must expire after 100 ticks, then xTimerPeriodInTicks should be set to 100. Alternatively, if the timer must expire after 500ms, then xPeriod can be set to (500 / portTICK_PERIOD_MS) provided configTICK_RATE_HZ is less than or equal to 1000.
- `uxAutoReload`: If uxAutoReload is set to pdTRUE then the timer will expire repeatedly with a frequency set by the xTimerPeriodInTicks parameter. If uxAutoReload is set to pdFALSE then the timer will be a one-shot timer and enter the dormant state after it expires.
- `pvTimerID`: An identifier that is assigned to the timer being created. Typically this would be used in the timer callback function to identify which timer expired when the same callback function is assigned to more than one timer.
- `pxCallbackFunction`: The function to call when the timer expires. Callback functions must have the prototype defined by TimerCallbackFunction_t, which is "void vCallbackFunction(TimerHandle_t xTimer);" .

TimerHandle_t **xTimerCreateStatic(const** char ***const** *pcTimerName*, **const** TickType_t *xTimerPeriodInTicks*, **const** UBaseType_t *uxAutoReload*, void ***const** *pvTimerID*, *TimerCallbackFunction_t* *pxCallbackFunction*, StaticTimer_t **pxTimerBuffer*)

Creates a new software timer instance, and returns a handle by which the created software timer can be referenced.

Internally, within the FreeRTOS implementation, software timers use a block of memory, in which the timer data structure is stored. If a software timer is created using xTimerCreate() then the required memory is automatically dynamically allocated inside the xTimerCreate() function. (see http://www.freertos.org/a00111.html). If a software timer is created using xTimerCreateStatic() then the application writer must provide the memory that will get used by the software timer. xTimerCreateStatic() therefore allows a software timer to be created without using any dynamic memory allocation.

Timers are created in the dormant state. The xTimerStart(), xTimerReset(), xTimerStartFromISR(), xTimerResetFromISR(), xTimerChangePeriod() and xTimerChangePeriodFromISR() API functions can all be used to transition a timer into the active state.

Example usage:

```
// The buffer used to hold the software timer's data structure.
static StaticTimer_t xTimerBuffer;

// A variable that will be incremented by the software timer's callback
// function.
UBaseType_t uxVariableToIncrement = 0;

// A software timer callback function that increments a variable passed to
// it when the software timer was created.  After the 5th increment the
// callback function stops the software timer.
static void prvTimerCallback( TimerHandle_t xExpiredTimer )
{
UBaseType_t *puxVariableToIncrement;
BaseType_t xReturned;

    // Obtain the address of the variable to increment from the timer ID.
    puxVariableToIncrement = ( UBaseType_t * ) pvTimerGetTimerID( xExpiredTimer );

    // Increment the variable to show the timer callback has executed.
    ( *puxVariableToIncrement )++;

    // If this callback has executed the required number of times, stop the
    // timer.
    if( *puxVariableToIncrement == 5 )
    {
        // This is called from a timer callback so must not block.
        xTimerStop( xExpiredTimer, staticDONT_BLOCK );
    }
}



void main( void )
{
    // Create the software time.  xTimerCreateStatic() has an extra parameter
    // than the normal xTimerCreate() API function.  The parameter is a pointer
    // to the StaticTimer_t structure that will hold the software timer
    // structure.  If the parameter is passed as NULL then the structure will be
    // allocated dynamically, just as if xTimerCreate() had been called.
    xTimer = xTimerCreateStatic( "T1",             // Text name for the task.
 ␣Helps debugging only.  Not used by FreeRTOS.
```

(下页继续)

(续上页)

```
                                xTimerPeriod,     // The period of the timer in␣
↪ticks.
                                pdTRUE,           // This is an auto-reload timer.
                                ( void * ) &uxVariableToIncrement,    // A␣
↪variable incremented by the software timer's callback function
                                prvTimerCallback, // The function to execute when␣
↪the timer expires.
                                &xTimerBuffer );  // The buffer that will hold the␣
↪software timer structure.

    // The scheduler has not started yet so a block time is not used.
    xReturned = xTimerStart( xTimer, 0 );

    // ...
    // Create tasks here.
    // ...

    // Starting the scheduler will start the timers running as they have already
    // been set into the active state.
    vTaskStartScheduler();

    // Should not reach here.
    for( ;; );
}
```

Return If the timer is created then a handle to the created timer is returned. If pxTimerBuffer was NULL then NULL is returned.

Parameters

- `pcTimerName`: A text name that is assigned to the timer. This is done purely to assist debugging. The kernel itself only ever references a timer by its handle, and never by its name.
- `xTimerPeriodInTicks`: The timer period. The time is defined in tick periods so the constant portTICK_PERIOD_MS can be used to convert a time that has been specified in milliseconds. For example, if the timer must expire after 100 ticks, then xTimerPeriodInTicks should be set to 100. Alternatively, if the timer must expire after 500ms, then xPeriod can be set to (500 / portTICK_PERIOD_MS) provided configTICK_RATE_HZ is less than or equal to 1000.
- `uxAutoReload`: If uxAutoReload is set to pdTRUE then the timer will expire repeatedly with a frequency set by the xTimerPeriodInTicks parameter. If uxAutoReload is set to pdFALSE

then the timer will be a one-shot timer and enter the dormant state after it expires.

- `pvTimerID`: An identifier that is assigned to the timer being created. Typically this would be used in the timer callback function to identify which timer expired when the same callback function is assigned to more than one timer.
- `pxCallbackFunction`: The function to call when the timer expires. Callback functions must have the prototype defined by TimerCallbackFunction_t, which is "void vCallbackFunction(TimerHandle_t xTimer);" .
- `pxTimerBuffer`: Must point to a variable of type StaticTimer_t, which will be then be used to hold the software timer' s data structures, removing the need for the memory to be allocated dynamically.

void *`pvTimerGetTimerID`(*TimerHandle_t xTimer*)

Returns the ID assigned to the timer.

IDs are assigned to timers using the pvTimerID parameter of the call to xTimerCreated() that was used to create the timer.

If the same callback function is assigned to multiple timers then the timer ID can be used within the callback function to identify which timer actually expired.

Example usage:

Return The ID assigned to the timer being queried.

Parameters

- `xTimer`: The timer being queried.

See the xTimerCreate() API function example usage scenario.

void `vTimerSetTimerID`(*TimerHandle_t xTimer*, void **pvNewID*)

Sets the ID assigned to the timer.

IDs are assigned to timers using the pvTimerID parameter of the call to xTimerCreated() that was used to create the timer.

If the same callback function is assigned to multiple timers then the timer ID can be used as time specific (timer local) storage.

Example usage:

Parameters

- `xTimer`: The timer being updated.
- `pvNewID`: The ID to assign to the timer.

See the xTimerCreate() API function example usage scenario.

BaseType_t **xTimerIsTimerActive**(*TimerHandle_t xTimer*)

Queries a timer to see if it is active or dormant.

A timer will be dormant if:

```
1) It has been created but not started, or

2) It is an expired one-shot timer that has not been restarted.
```

Timers are created in the dormant state. The xTimerStart(), xTimerReset(), xTimerStartFromISR(), xTimerResetFromISR(), xTimerChangePeriod() and xTimerChangePeriodFromISR() API functions can all be used to transition a timer into the active state.

Example usage:

```
// This function assumes xTimer has already been created.
void vAFunction( TimerHandle_t xTimer )
{
    if( xTimerIsTimerActive( xTimer ) != pdFALSE ) // or more simply and␣
↪equivalently "if( xTimerIsTimerActive( xTimer ) )"
    {
        // xTimer is active, do something.
    }
    else
    {
        // xTimer is not active, do something else.
    }
}
```

Return pdFALSE will be returned if the timer is dormant. A value other than pdFALSE will be returned if the timer is active.

Parameters

- `xTimer`: The timer being queried.

TaskHandle_t **xTimerGetTimerDaemonTaskHandle**(void)

xTimerGetTimerDaemonTaskHandle() is only available if INCLUDE_xTimerGetTimerDaemonTaskHandle is set to 1 in FreeRTOSConfig.h.

Simply returns the handle of the timer service/daemon task. It it not valid to call xTimerGetTimerDaemonTaskHandle() before the scheduler has been started.

TickType_t **xTimerGetPeriod**(*TimerHandle_t xTimer*)

Returns the period of a timer.

Return The period of the timer in ticks.

Parameters

- `xTimer`: The handle of the timer being queried.

TickType_t **xTimerGetExpiryTime**(*TimerHandle_t xTimer*)

Returns the time in ticks at which the timer will expire. If this is less than the current tick count then the expiry time has overflowed from the current time.

Return If the timer is running then the time in ticks at which the timer will next expire is returned. If the timer is not running then the return value is undefined.

Parameters

- `xTimer`: The handle of the timer being queried.

BaseType_t **xTimerPendFunctionCallFromISR**(*PendedFunction_t xFunctionToPend*, void **pvParameter1*, uint32_t *ulParameter2*, BaseType_t **pxHigherPriorityTaskWoken*)

Used from application interrupt service routines to defer the execution of a function to the RTOS daemon task (the timer service task, hence this function is implemented in timers.c and is prefixed with 'Timer').

Ideally an interrupt service routine (ISR) is kept as short as possible, but sometimes an ISR either has a lot of processing to do, or needs to perform processing that is not deterministic. In these cases xTimerPendFunctionCallFromISR() can be used to defer processing of a function to the RTOS daemon task.

A mechanism is provided that allows the interrupt to return directly to the task that will subsequently execute the pended callback function. This allows the callback function to execute contiguously in time with the interrupt - just as if the callback had executed in the interrupt itself.

Example usage:

```
// The callback function that will execute in the context of the daemon task.
// Note callback functions must all use this same prototype.
void vProcessInterface( void *pvParameter1, uint32_t ulParameter2 )
{
   BaseType_t xInterfaceToService;

   // The interface that requires servicing is passed in the second
    // parameter.  The first parameter is not used in this case.
   xInterfaceToService = ( BaseType_t ) ulParameter2;

   // ...Perform the processing here...
```

(下页继续)

(续上页)

```
}

// An ISR that receives data packets from multiple interfaces
void vAnISR( void )
{
   BaseType_t xInterfaceToService, xHigherPriorityTaskWoken;

   // Query the hardware to determine which interface needs processing.
   xInterfaceToService = prvCheckInterfaces();

    // The actual processing is to be deferred to a task.  Request the
    // vProcessInterface() callback function is executed, passing in the
   // number of the interface that needs processing.  The interface to
   // service is passed in the second parameter.  The first parameter is
   // not used in this case.
   xHigherPriorityTaskWoken = pdFALSE;
   xTimerPendFunctionCallFromISR( vProcessInterface, NULL, ( uint32_t )␣
↪xInterfaceToService, &xHigherPriorityTaskWoken );

   // If xHigherPriorityTaskWoken is now set to pdTRUE then a context
   // switch should be requested.  The macro used is port specific and will
   // be either portYIELD_FROM_ISR() or portEND_SWITCHING_ISR() - refer to
   // the documentation page for the port being used.
   portYIELD_FROM_ISR( xHigherPriorityTaskWoken );

}
```

Return pdPASS is returned if the message was successfully sent to the timer daemon task, otherwise pdFALSE is returned.

Parameters

- `xFunctionToPend`: The function to execute from the timer service/ daemon task. The function must conform to the PendedFunction_t prototype.
- `pvParameter1`: The value of the callback function' s first parameter. The parameter has a void * type to allow it to be used to pass any type. For example, unsigned longs can be cast to a void *, or the void * can be used to point to a structure.
- `ulParameter2`: The value of the callback function' s second parameter.
- `pxHigherPriorityTaskWoken`: As mentioned above, calling this function will result in a message being sent to the timer daemon task. If the priority of the timer daemon task (which is set

using configTIMER_TASK_PRIORITY in FreeRTOSConfig.h) is higher than the priority of the currently running task (the task the interrupt interrupted) then *pxHigherPriorityTaskWoken will be set to pdTRUE within xTimerPendFunctionCallFromISR(), indicating that a context switch should be requested before the interrupt exits. For that reason *pxHigherPriorityTaskWoken must be initialised to pdFALSE. See the example code below.

BaseType_t **xTimerPendFunctionCall**(*PendedFunction_t xFunctionToPend*, void **pvParameter1*, uint32_t *ulParameter2*, TickType_t *xTicksToWait*)

Used to defer the execution of a function to the RTOS daemon task (the timer service task, hence this function is implemented in timers.c and is prefixed with 'Timer').

Return pdPASS is returned if the message was successfully sent to the timer daemon task, otherwise pdFALSE is returned.

Parameters

- `xFunctionToPend`: The function to execute from the timer service/ daemon task. The function must conform to the PendedFunction_t prototype.
- `pvParameter1`: The value of the callback function's first parameter. The parameter has a void * type to allow it to be used to pass any type. For example, unsigned longs can be cast to a void *, or the void * can be used to point to a structure.
- `ulParameter2`: The value of the callback function's second parameter.
- `xTicksToWait`: Calling this function will result in a message being sent to the timer daemon task on a queue. xTicksToWait is the amount of time the calling task should remain in the Blocked state (so not using any processing time) for space to become available on the timer queue if the queue is found to be full.

`const` char *`pcTimerGetTimerName`(*TimerHandle_t xTimer*)

Returns the name that was assigned to a timer when the timer was created.

Return The name assigned to the timer specified by the xTimer parameter.

Parameters

- `xTimer`: The handle of the timer being queried.

Macros

`tmrCOMMAND_EXECUTE_CALLBACK_FROM_ISR`

`tmrCOMMAND_EXECUTE_CALLBACK`

`tmrCOMMAND_START_DONT_TRACE`

`tmrCOMMAND_START`

tmrCOMMAND_RESET

tmrCOMMAND_STOP

tmrCOMMAND_CHANGE_PERIOD

tmrCOMMAND_DELETE

tmrFIRST_FROM_ISR_COMMAND

tmrCOMMAND_START_FROM_ISR

tmrCOMMAND_RESET_FROM_ISR

tmrCOMMAND_STOP_FROM_ISR

tmrCOMMAND_CHANGE_PERIOD_FROM_ISR

xTimerStart(xTimer, xTicksToWait)

Timer functionality is provided by a timer service/daemon task. Many of the public FreeRTOS timer API functions send commands to the timer service task through a queue called the timer command queue. The timer command queue is private to the kernel itself and is not directly accessible to application code. The length of the timer command queue is set by the configTIMER_QUEUE_LENGTH configuration constant.

xTimerStart() starts a timer that was previously created using the xTimerCreate() API function. If the timer had already been started and was already in the active state, then xTimerStart() has equivalent functionality to the xTimerReset() API function.

Starting a timer ensures the timer is in the active state. If the timer is not stopped, deleted, or reset in the mean time, the callback function associated with the timer will get called 'n' ticks after xTimerStart() was called, where 'n' is the timers defined period.

It is valid to call xTimerStart() before the scheduler has been started, but when this is done the timer will not actually start until the scheduler is started, and the timers expiry time will be relative to when the scheduler is started, not relative to when xTimerStart() was called.

The configUSE_TIMERS configuration constant must be set to 1 for xTimerStart() to be available.

Example usage:

Return pdFAIL will be returned if the start command could not be sent to the timer command queue even after xTicksToWait ticks had passed. pdPASS will be returned if the command was successfully sent to the timer command queue. When the command is actually processed will depend on the priority of the timer service/daemon task relative to other tasks in the system, although the timers expiry time is relative to when xTimerStart() is actually called. The timer service/daemon task priority is set by the configTIMER_TASK_PRIORITY configuration constant.

Parameters

- `xTimer`: The handle of the timer being started/restarted.

- `xTicksToWait`: Specifies the time, in ticks, that the calling task should be held in the Blocked state to wait for the start command to be successfully sent to the timer command queue, should the queue already be full when xTimerStart() was called. xTicksToWait is ignored if xTimerStart() is called before the scheduler is started.

See the xTimerCreate() API function example usage scenario.

`xTimerStop`(xTimer, xTicksToWait)

Timer functionality is provided by a timer service/daemon task. Many of the public FreeRTOS timer API functions send commands to the timer service task through a queue called the timer command queue. The timer command queue is private to the kernel itself and is not directly accessible to application code. The length of the timer command queue is set by the configTIMER_QUEUE_LENGTH configuration constant.

xTimerStop() stops a timer that was previously started using either of the The xTimerStart(), xTimerReset(), xTimerStartFromISR(), xTimerResetFromISR(), xTimerChangePeriod() or xTimerChangePeriodFromISR() API functions.

Stopping a timer ensures the timer is not in the active state.

The configUSE_TIMERS configuration constant must be set to 1 for xTimerStop() to be available.

Example usage:

Return pdFAIL will be returned if the stop command could not be sent to the timer command queue even after xTicksToWait ticks had passed. pdPASS will be returned if the command was successfully sent to the timer command queue. When the command is actually processed will depend on the priority of the timer service/daemon task relative to other tasks in the system. The timer service/daemon task priority is set by the configTIMER_TASK_PRIORITY configuration constant.

Parameters

- `xTimer`: The handle of the timer being stopped.
- `xTicksToWait`: Specifies the time, in ticks, that the calling task should be held in the Blocked state to wait for the stop command to be successfully sent to the timer command queue, should the queue already be full when xTimerStop() was called. xTicksToWait is ignored if xTimerStop() is called before the scheduler is started.

See the xTimerCreate() API function example usage scenario.

`xTimerChangePeriod`(xTimer, xNewPeriod, xTicksToWait)

Timer functionality is provided by a timer service/daemon task. Many of the public FreeRTOS timer API functions send commands to the timer service task through a queue called the timer command queue. The timer command queue is private to the kernel itself and is not directly accessible to application code. The length of the timer command queue is set by the configTIMER_QUEUE_LENGTH configuration constant.

xTimerChangePeriod() changes the period of a timer that was previously created using the xTimerCreate() API function.

xTimerChangePeriod() can be called to change the period of an active or dormant state timer.

The configUSE_TIMERS configuration constant must be set to 1 for xTimerChangePeriod() to be available.

Example usage:

```
// This function assumes xTimer has already been created.  If the timer
// referenced by xTimer is already active when it is called, then the timer
// is deleted.  If the timer referenced by xTimer is not active when it is
// called, then the period of the timer is set to 500ms and the timer is
// started.
void vAFunction( TimerHandle_t xTimer )
{
    if( xTimerIsTimerActive( xTimer ) != pdFALSE ) // or more simply and␣
↪equivalently "if( xTimerIsTimerActive( xTimer ) )"
    {
        // xTimer is already active - delete it.
        xTimerDelete( xTimer );
    }
    else
    {
        // xTimer is not active, change its period to 500ms.  This will also
        // cause the timer to start.  Block for a maximum of 100 ticks if the
        // change period command cannot immediately be sent to the timer
        // command queue.
        if( xTimerChangePeriod( xTimer, 500 / portTICK_PERIOD_MS, 100 ) == pdPASS )
        {
            // The command was successfully sent.
        }
        else
        {
            // The command could not be sent, even after waiting for 100 ticks
            // to pass.  Take appropriate action here.
        }
    }
}
```

Return pdFAIL will be returned if the change period command could not be sent to the timer command queue even after xTicksToWait ticks had passed. pdPASS will be returned if the command

was successfully sent to the timer command queue. When the command is actually processed will depend on the priority of the timer service/daemon task relative to other tasks in the system. The timer service/daemon task priority is set by the configTIMER_TASK_PRIORITY configuration constant.

Parameters

- `xTimer`: The handle of the timer that is having its period changed.
- `xNewPeriod`: The new period for xTimer. Timer periods are specified in tick periods, so the constant portTICK_PERIOD_MS can be used to convert a time that has been specified in milliseconds. For example, if the timer must expire after 100 ticks, then xNewPeriod should be set to 100. Alternatively, if the timer must expire after 500ms, then xNewPeriod can be set to (500 / portTICK_PERIOD_MS) provided configTICK_RATE_HZ is less than or equal to 1000.
- `xTicksToWait`: Specifies the time, in ticks, that the calling task should be held in the Blocked state to wait for the change period command to be successfully sent to the timer command queue, should the queue already be full when xTimerChangePeriod() was called. xTicksToWait is ignored if xTimerChangePeriod() is called before the scheduler is started.

`xTimerDelete`(xTimer, xTicksToWait)

Timer functionality is provided by a timer service/daemon task. Many of the public FreeRTOS timer API functions send commands to the timer service task through a queue called the timer command queue. The timer command queue is private to the kernel itself and is not directly accessible to application code. The length of the timer command queue is set by the configTIMER_QUEUE_LENGTH configuration constant.

xTimerDelete() deletes a timer that was previously created using the xTimerCreate() API function.

The configUSE_TIMERS configuration constant must be set to 1 for xTimerDelete() to be available.

Example usage:

Return pdFAIL will be returned if the delete command could not be sent to the timer command queue even after xTicksToWait ticks had passed. pdPASS will be returned if the command was successfully sent to the timer command queue. When the command is actually processed will depend on the priority of the timer service/daemon task relative to other tasks in the system. The timer service/daemon task priority is set by the configTIMER_TASK_PRIORITY configuration constant.

Parameters

- `xTimer`: The handle of the timer being deleted.
- `xTicksToWait`: Specifies the time, in ticks, that the calling task should be held in the Blocked state to wait for the delete command to be successfully sent to the timer command queue, should the queue already be full when xTimerDelete() was called. xTicksToWait is ignored if xTimerDelete() is called before the scheduler is started.

See the xTimerChangePeriod() API function example usage scenario.

xTimerReset(xTimer, xTicksToWait)

Timer functionality is provided by a timer service/daemon task. Many of the public FreeRTOS timer API functions send commands to the timer service task through a queue called the timer command queue. The timer command queue is private to the kernel itself and is not directly accessible to application code. The length of the timer command queue is set by the configTIMER_QUEUE_LENGTH configuration constant.

xTimerReset() re-starts a timer that was previously created using the xTimerCreate() API function. If the timer had already been started and was already in the active state, then xTimerReset() will cause the timer to re-evaluate its expiry time so that it is relative to when xTimerReset() was called. If the timer was in the dormant state then xTimerReset() has equivalent functionality to the xTimerStart() API function.

Resetting a timer ensures the timer is in the active state. If the timer is not stopped, deleted, or reset in the mean time, the callback function associated with the timer will get called 'n' ticks after xTimerReset() was called, where 'n' is the timers defined period.

It is valid to call xTimerReset() before the scheduler has been started, but when this is done the timer will not actually start until the scheduler is started, and the timers expiry time will be relative to when the scheduler is started, not relative to when xTimerReset() was called.

The configUSE_TIMERS configuration constant must be set to 1 for xTimerReset() to be available.

Example usage:

```
// When a key is pressed, an LCD back-light is switched on.  If 5 seconds pass
// without a key being pressed, then the LCD back-light is switched off.  In
// this case, the timer is a one-shot timer.

TimerHandle_t xBacklightTimer = NULL;

// The callback function assigned to the one-shot timer.  In this case the
// parameter is not used.
void vBacklightTimerCallback( TimerHandle_t pxTimer )
{
    // The timer expired, therefore 5 seconds must have passed since a key
    // was pressed.  Switch off the LCD back-light.
    vSetBacklightState( BACKLIGHT_OFF );
}

// The key press event handler.
void vKeyPressEventHandler( char cKey )
{
```

(下页继续)

(续上页)

```
    // Ensure the LCD back-light is on, then reset the timer that is
    // responsible for turning the back-light off after 5 seconds of
    // key inactivity.  Wait 10 ticks for the command to be successfully sent
    // if it cannot be sent immediately.
    vSetBacklightState( BACKLIGHT_ON );
    if( xTimerReset( xBacklightTimer, 100 ) != pdPASS )
    {
        // The reset command was not executed successfully.  Take appropriate
        // action here.
    }

    // Perform the rest of the key processing here.
}

void main( void )
{
int32_t x;

    // Create then start the one-shot timer that is responsible for turning
    // the back-light off if no keys are pressed within a 5 second period.
    xBacklightTimer = xTimerCreate( "BacklightTimer",           // Just a text name,
↪ not used by the kernel.
                                    ( 5000 / portTICK_PERIOD_MS), // The timer␣
↪period in ticks.
                                    pdFALSE,                    // The timer is a␣
↪one-shot timer.
                                    0,                          // The id is not␣
↪used by the callback so can take any value.
                                    vBacklightTimerCallback     // The callback␣
↪function that switches the LCD back-light off.
                                  );

    if( xBacklightTimer == NULL )
    {
        // The timer was not created.
    }
    else
    {
        // Start the timer.  No block time is specified, and even if one was
        // it would be ignored because the scheduler has not yet been
```

(下页继续)

(续上页)

```
        // started.
        if( xTimerStart( xBacklightTimer, 0 ) != pdPASS )
        {
            // The timer could not be set into the Active state.
        }
    }

    // ...
    // Create tasks here.
    // ...

    // Starting the scheduler will start the timer running as it has already
    // been set into the active state.
    xTaskStartScheduler();

    // Should not reach here.
    for( ;; );
}
```

Return pdFAIL will be returned if the reset command could not be sent to the timer command queue even after xTicksToWait ticks had passed. pdPASS will be returned if the command was successfully sent to the timer command queue. When the command is actually processed will depend on the priority of the timer service/daemon task relative to other tasks in the system, although the timers expiry time is relative to when xTimerStart() is actually called. The timer service/daemon task priority is set by the configTIMER_TASK_PRIORITY configuration constant.

Parameters

- `xTimer`: The handle of the timer being reset/started/restarted.
- `xTicksToWait`: Specifies the time, in ticks, that the calling task should be held in the Blocked state to wait for the reset command to be successfully sent to the timer command queue, should the queue already be full when xTimerReset() was called. xTicksToWait is ignored if xTimerReset() is called before the scheduler is started.

xTimerStartFromISR(xTimer, pxHigherPriorityTaskWoken)

A version of xTimerStart() that can be called from an interrupt service routine.

Example usage:

```
// This scenario assumes xBacklightTimer has already been created.  When a
// key is pressed, an LCD back-light is switched on.  If 5 seconds pass
```

(下页继续)

(续上页)

```
// without a key being pressed, then the LCD back-light is switched off.  In
// this case, the timer is a one-shot timer, and unlike the example given for
// the xTimerReset() function, the key press event handler is an interrupt
// service routine.

// The callback function assigned to the one-shot timer.  In this case the
// parameter is not used.
void vBacklightTimerCallback( TimerHandle_t pxTimer )
{
    // The timer expired, therefore 5 seconds must have passed since a key
    // was pressed.  Switch off the LCD back-light.
    vSetBacklightState( BACKLIGHT_OFF );
}

// The key press interrupt service routine.
void vKeyPressEventInterruptHandler( void )
{
BaseType_t xHigherPriorityTaskWoken = pdFALSE;

    // Ensure the LCD back-light is on, then restart the timer that is
    // responsible for turning the back-light off after 5 seconds of
    // key inactivity.  This is an interrupt service routine so can only
    // call FreeRTOS API functions that end in "FromISR".
    vSetBacklightState( BACKLIGHT_ON );

    // xTimerStartFromISR() or xTimerResetFromISR() could be called here
    // as both cause the timer to re-calculate its expiry time.
    // xHigherPriorityTaskWoken was initialised to pdFALSE when it was
    // declared (in this function).
    if( xTimerStartFromISR( xBacklightTimer, &xHigherPriorityTaskWoken ) != pdPASS )
    {
        // The start command was not executed successfully.  Take appropriate
        // action here.
    }

    // Perform the rest of the key processing here.

    // If xHigherPriorityTaskWoken equals pdTRUE, then a context switch
    // should be performed.  The syntax required to perform a context switch
    // from inside an ISR varies from port to port, and from compiler to
```

(下页继续)

(续上页)

```
    // compiler.  Inspect the demos for the port you are using to find the
    // actual syntax required.
    if( xHigherPriorityTaskWoken != pdFALSE )
    {
        // Call the interrupt safe yield function here (actual function
        // depends on the FreeRTOS port being used).
    }
}
```

Return pdFAIL will be returned if the start command could not be sent to the timer command queue. pdPASS will be returned if the command was successfully sent to the timer command queue. When the command is actually processed will depend on the priority of the timer service/daemon task relative to other tasks in the system, although the timers expiry time is relative to when xTimerStartFromISR() is actually called. The timer service/daemon task priority is set by the configTIMER_TASK_PRIORITY configuration constant.

Parameters

- `xTimer`: The handle of the timer being started/restarted.
- `pxHigherPriorityTaskWoken`: The timer service/daemon task spends most of its time in the Blocked state, waiting for messages to arrive on the timer command queue. Calling xTimerStartFromISR() writes a message to the timer command queue, so has the potential to transition the timer service/daemon task out of the Blocked state. If calling xTimerStartFromISR() causes the timer service/daemon task to leave the Blocked state, and the timer service/ daemon task has a priority equal to or greater than the currently executing task (the task that was interrupted), then *pxHigherPriorityTaskWoken will get set to pdTRUE internally within the xTimerStartFromISR() function. If xTimerStartFromISR() sets this value to pdTRUE then a context switch should be performed before the interrupt exits.

`xTimerStopFromISR`(xTimer, pxHigherPriorityTaskWoken)

A version of xTimerStop() that can be called from an interrupt service routine.

Example usage:

```
// This scenario assumes xTimer has already been created and started.  When
// an interrupt occurs, the timer should be simply stopped.

// The interrupt service routine that stops the timer.
void vAnExampleInterruptServiceRoutine( void )
{
BaseType_t xHigherPriorityTaskWoken = pdFALSE;
```

(下页继续)

(续上页)

```
    // The interrupt has occurred - simply stop the timer.
    // xHigherPriorityTaskWoken was set to pdFALSE where it was defined
    // (within this function).  As this is an interrupt service routine, only
    // FreeRTOS API functions that end in "FromISR" can be used.
    if( xTimerStopFromISR( xTimer, &xHigherPriorityTaskWoken ) != pdPASS )
    {
        // The stop command was not executed successfully.  Take appropriate
        // action here.
    }

    // If xHigherPriorityTaskWoken equals pdTRUE, then a context switch
    // should be performed.  The syntax required to perform a context switch
    // from inside an ISR varies from port to port, and from compiler to
    // compiler.  Inspect the demos for the port you are using to find the
    // actual syntax required.
    if( xHigherPriorityTaskWoken != pdFALSE )
    {
        // Call the interrupt safe yield function here (actual function
        // depends on the FreeRTOS port being used).
    }
}
```

Return pdFAIL will be returned if the stop command could not be sent to the timer command queue. pdPASS will be returned if the command was successfully sent to the timer command queue. When the command is actually processed will depend on the priority of the timer service/daemon task relative to other tasks in the system. The timer service/daemon task priority is set by the configTIMER_TASK_PRIORITY configuration constant.

Parameters

- `xTimer`: The handle of the timer being stopped.
- `pxHigherPriorityTaskWoken`: The timer service/daemon task spends most of its time in the Blocked state, waiting for messages to arrive on the timer command queue. Calling xTimerStopFromISR() writes a message to the timer command queue, so has the potential to transition the timer service/daemon task out of the Blocked state. If calling xTimerStopFromISR() causes the timer service/daemon task to leave the Blocked state, and the timer service/ daemon task has a priority equal to or greater than the currently executing task (the task that was interrupted), then *pxHigherPriorityTaskWoken will get set to pdTRUE internally within the xTimerStopFromISR() function. If xTimerStopFromISR() sets this value to pdTRUE then a context switch should be performed before the interrupt

exits.

xTimerChangePeriodFromISR(xTimer, xNewPeriod, pxHigherPriorityTaskWoken)

A version of xTimerChangePeriod() that can be called from an interrupt service routine.

Example usage:

```
// This scenario assumes xTimer has already been created and started.  When
// an interrupt occurs, the period of xTimer should be changed to 500ms.

// The interrupt service routine that changes the period of xTimer.
void vAnExampleInterruptServiceRoutine( void )
{
BaseType_t xHigherPriorityTaskWoken = pdFALSE;

    // The interrupt has occurred - change the period of xTimer to 500ms.
    // xHigherPriorityTaskWoken was set to pdFALSE where it was defined
    // (within this function).  As this is an interrupt service routine, only
    // FreeRTOS API functions that end in "FromISR" can be used.
    if( xTimerChangePeriodFromISR( xTimer, &xHigherPriorityTaskWoken ) != pdPASS )
    {
        // The command to change the timers period was not executed
        // successfully.  Take appropriate action here.
    }

    // If xHigherPriorityTaskWoken equals pdTRUE, then a context switch
    // should be performed.  The syntax required to perform a context switch
    // from inside an ISR varies from port to port, and from compiler to
    // compiler.  Inspect the demos for the port you are using to find the
    // actual syntax required.
    if( xHigherPriorityTaskWoken != pdFALSE )
    {
        // Call the interrupt safe yield function here (actual function
        // depends on the FreeRTOS port being used).
    }
}
```

Return pdFAIL will be returned if the command to change the timers period could not be sent to the timer command queue. pdPASS will be returned if the command was successfully sent to the timer command queue. When the command is actually processed will depend on the priority of the timer service/daemon task relative to other tasks in the system. The timer service/daemon task priority is set by the configTIMER_TASK_PRIORITY configuration constant.

Parameters

- `xTimer`: The handle of the timer that is having its period changed.
- `xNewPeriod`: The new period for xTimer. Timer periods are specified in tick periods, so the constant portTICK_PERIOD_MS can be used to convert a time that has been specified in milliseconds. For example, if the timer must expire after 100 ticks, then xNewPeriod should be set to 100. Alternatively, if the timer must expire after 500ms, then xNewPeriod can be set to (500 / portTICK_PERIOD_MS) provided configTICK_RATE_HZ is less than or equal to 1000.
- `pxHigherPriorityTaskWoken`: The timer service/daemon task spends most of its time in the Blocked state, waiting for messages to arrive on the timer command queue. Calling xTimerChangePeriodFromISR() writes a message to the timer command queue, so has the potential to transition the timer service/ daemon task out of the Blocked state. If calling xTimerChangePeriodFromISR() causes the timer service/daemon task to leave the Blocked state, and the timer service/daemon task has a priority equal to or greater than the currently executing task (the task that was interrupted), then *pxHigherPriorityTaskWoken will get set to pdTRUE internally within the xTimerChangePeriodFromISR() function. If xTimerChangePeriodFromISR() sets this value to pdTRUE then a context switch should be performed before the interrupt exits.

`xTimerResetFromISR`(xTimer, pxHigherPriorityTaskWoken)

A version of xTimerReset() that can be called from an interrupt service routine.

Example usage:

```
// This scenario assumes xBacklightTimer has already been created.  When a
// key is pressed, an LCD back-light is switched on.  If 5 seconds pass
// without a key being pressed, then the LCD back-light is switched off.  In
// this case, the timer is a one-shot timer, and unlike the example given for
// the xTimerReset() function, the key press event handler is an interrupt
// service routine.

// The callback function assigned to the one-shot timer.  In this case the
// parameter is not used.
void vBacklightTimerCallback( TimerHandle_t pxTimer )
{
    // The timer expired, therefore 5 seconds must have passed since a key
    // was pressed.  Switch off the LCD back-light.
    vSetBacklightState( BACKLIGHT_OFF );
}

// The key press interrupt service routine.
```

(下页继续)

(续上页)

```
void vKeyPressEventInterruptHandler( void )
{
BaseType_t xHigherPriorityTaskWoken = pdFALSE;

    // Ensure the LCD back-light is on, then reset the timer that is
    // responsible for turning the back-light off after 5 seconds of
    // key inactivity.  This is an interrupt service routine so can only
    // call FreeRTOS API functions that end in "FromISR".
    vSetBacklightState( BACKLIGHT_ON );

    // xTimerStartFromISR() or xTimerResetFromISR() could be called here
    // as both cause the timer to re-calculate its expiry time.
    // xHigherPriorityTaskWoken was initialised to pdFALSE when it was
    // declared (in this function).
    if( xTimerResetFromISR( xBacklightTimer, &xHigherPriorityTaskWoken ) != pdPASS )
    {
        // The reset command was not executed successfully.  Take appropriate
        // action here.
    }

    // Perform the rest of the key processing here.

    // If xHigherPriorityTaskWoken equals pdTRUE, then a context switch
    // should be performed.  The syntax required to perform a context switch
    // from inside an ISR varies from port to port, and from compiler to
    // compiler.  Inspect the demos for the port you are using to find the
    // actual syntax required.
    if( xHigherPriorityTaskWoken != pdFALSE )
    {
        // Call the interrupt safe yield function here (actual function
        // depends on the FreeRTOS port being used).
    }
}
```

Return pdFAIL will be returned if the reset command could not be sent to the timer command queue. pdPASS will be returned if the command was successfully sent to the timer command queue. When the command is actually processed will depend on the priority of the timer service/daemon task relative to other tasks in the system, although the timers expiry time is relative to when xTimerResetFromISR() is actually called. The timer service/daemon task priority is set by the configTIMER_TASK_PRIORITY configuration constant.

Parameters

- `xTimer`: The handle of the timer that is to be started, reset, or restarted.
- `pxHigherPriorityTaskWoken`: The timer service/daemon task spends most of its time in the Blocked state, waiting for messages to arrive on the timer command queue. Calling xTimerResetFromISR() writes a message to the timer command queue, so has the potential to transition the timer service/daemon task out of the Blocked state. If calling xTimerResetFromISR() causes the timer service/daemon task to leave the Blocked state, and the timer service/ daemon task has a priority equal to or greater than the currently executing task (the task that was interrupted), then *pxHigherPriorityTaskWoken will get set to pdTRUE internally within the xTimerResetFromISR() function. If xTimerResetFromISR() sets this value to pdTRUE then a context switch should be performed before the interrupt exits.

Type Definitions

`typedef` void *`TimerHandle_t`

Type by which software timers are referenced. For example, a call to xTimerCreate() returns an TimerHandle_t variable that can then be used to reference the subject timer in calls to other software timer API functions (for example, xTimerStart(), xTimerReset(), etc.).

`typedef` void (*`TimerCallbackFunction_t`)(*TimerHandle_t* xTimer)

Defines the prototype to which timer callback functions must conform.

`typedef` void (*`PendedFunction_t`)(void *, uint32_t)

Defines the prototype to which functions used with the xTimerPendFunctionCallFromISR() function must conform.

Event Group API

Header File

- freertos/include/freertos/event_groups.h

Functions

EventGroupHandle_t `xEventGroupCreate`(void)

Create a new event group.

Internally, within the FreeRTOS implementation, event groups use a [small] block of memory, in which the event group' s structure is stored. If an event groups is created using xEventGroupCreate() then the required memory is automatically dynamically allocated inside the xEventGroupCreate() function. (see http://www.freertos.org/a00111.html). If an event group is created using xEventGropuCreateStatic() then the application writer must instead provide the memory that will get used by the event group.

xEventGroupCreateStatic() therefore allows an event group to be created without using any dynamic memory allocation.

Although event groups are not related to ticks, for internal implementation reasons the number of bits available for use in an event group is dependent on the configUSE_16_BIT_TICKS setting in FreeRTOSConfig.h. If configUSE_16_BIT_TICKS is 1 then each event group contains 8 usable bits (bit 0 to bit 7). If configUSE_16_BIT_TICKS is set to 0 then each event group has 24 usable bits (bit 0 to bit 23). The EventBits_t type is used to store event bits within an event group.

Example usage:

```
// Declare a variable to hold the created event group.
EventGroupHandle_t xCreatedEventGroup;

// Attempt to create the event group.
xCreatedEventGroup = xEventGroupCreate();

// Was the event group created successfully?
if( xCreatedEventGroup == NULL )
{
    // The event group was not created because there was insufficient
    // FreeRTOS heap available.
}
else
{
    // The event group was created.
}
```

Return If the event group was created then a handle to the event group is returned. If there was insufficient FreeRTOS heap available to create the event group then NULL is returned. See http://www.freertos.org/a00111.html

EventGroupHandle_t **xEventGroupCreateStatic**(StaticEventGroup_t **pxEventGroupBuffer*)

Create a new event group.

Internally, within the FreeRTOS implementation, event groups use a [small] block of memory, in which the event group' s structure is stored. If an event groups is created using xEventGropuCreate() then the required memory is automatically dynamically allocated inside the xEventGroupCreate() function. (see http://www.freertos.org/a00111.html). If an event group is created using xEventGropuCreateStatic() then the application writer must instead provide the memory that will get used by the event group. xEventGroupCreateStatic() therefore allows an event group to be created without using any dynamic memory allocation.

Although event groups are not related to ticks, for internal implementation reasons the number of

bits available for use in an event group is dependent on the configUSE_16_BIT_TICKS setting in FreeRTOSConfig.h. If configUSE_16_BIT_TICKS is 1 then each event group contains 8 usable bits (bit 0 to bit 7). If configUSE_16_BIT_TICKS is set to 0 then each event group has 24 usable bits (bit 0 to bit 23). The EventBits_t type is used to store event bits within an event group.

Example usage:

```
// StaticEventGroup_t is a publicly accessible structure that has the same
// size and alignment requirements as the real event group structure.  It is
// provided as a mechanism for applications to know the size of the event
// group (which is dependent on the architecture and configuration file
// settings) without breaking the strict data hiding policy by exposing the
// real event group internals.  This StaticEventGroup_t variable is passed
// into the xSemaphoreCreateEventGroupStatic() function and is used to store
// the event group's data structures
StaticEventGroup_t xEventGroupBuffer;

// Create the event group without dynamically allocating any memory.
xEventGroup = xEventGroupCreateStatic( &xEventGroupBuffer );
```

Return If the event group was created then a handle to the event group is returned. If pxEventGroupBuffer was NULL then NULL is returned.

Parameters

- `pxEventGroupBuffer`: pxEventGroupBuffer must point to a variable of type StaticEventGroup_t, which will be then be used to hold the event group' s data structures, removing the need for the memory to be allocated dynamically.

EventBits_t **xEventGroupWaitBits**(*EventGroupHandle_t* *xEventGroup*, **const** *EventBits_t* *uxBitsToWaitFor*, **const** BaseType_t *xClearOnExit*, **const** BaseType_t *xWaitForAllBits*, TickType_t *xTicksToWait*)

[Potentially] block to wait for one or more bits to be set within a previously created event group.

This function cannot be called from an interrupt.

Example usage:

```
#define BIT_0    ( 1 << 0 )
#define BIT_4    ( 1 << 4 )

void aFunction( EventGroupHandle_t xEventGroup )
{
EventBits_t uxBits;
const TickType_t xTicksToWait = 100 / portTICK_PERIOD_MS;
```

(下页继续)

(续上页)

```
    // Wait a maximum of 100ms for either bit 0 or bit 4 to be set within
    // the event group.  Clear the bits before exiting.
    uxBits = xEventGroupWaitBits(
                xEventGroup,    // The event group being tested.
                BIT_0 | BIT_4,  // The bits within the event group to wait for.
                pdTRUE,         // BIT_0 and BIT_4 should be cleared before␣
↪returning.
                pdFALSE,        // Don't wait for both bits, either bit will do.
                xTicksToWait ); // Wait a maximum of 100ms for either bit to be␣
↪set.

    if( ( uxBits & ( BIT_0 | BIT_4 ) ) == ( BIT_0 | BIT_4 ) )
    {
        // xEventGroupWaitBits() returned because both bits were set.
    }
    else if( ( uxBits & BIT_0 ) != 0 )
    {
        // xEventGroupWaitBits() returned because just BIT_0 was set.
    }
    else if( ( uxBits & BIT_4 ) != 0 )
    {
        // xEventGroupWaitBits() returned because just BIT_4 was set.
    }
    else
    {
        // xEventGroupWaitBits() returned because xTicksToWait ticks passed
        // without either BIT_0 or BIT_4 becoming set.
    }
}
```

{c}

Return The value of the event group at the time either the bits being waited for became set, or the block time expired. Test the return value to know which bits were set. If xEventGroupWaitBits() returned because its timeout expired then not all the bits being waited for will be set. If xEventGroupWaitBits() returned because the bits it was waiting for were set then the returned value is the event group value before any bits were automatically cleared in the case that xClearOnExit parameter was set to pdTRUE.

Parameters

- `xEventGroup`: The event group in which the bits are being tested. The event group must have previously been created using a call to xEventGroupCreate().
- `uxBitsToWaitFor`: A bitwise value that indicates the bit or bits to test inside the event group. For example, to wait for bit 0 and/or bit 2 set uxBitsToWaitFor to 0x05. To wait for bits 0 and/or bit 1 and/or bit 2 set uxBitsToWaitFor to 0x07. Etc.
- `xClearOnExit`: If xClearOnExit is set to pdTRUE then any bits within uxBitsToWaitFor that are set within the event group will be cleared before xEventGroupWaitBits() returns if the wait condition was met (if the function returns for a reason other than a timeout). If xClearOnExit is set to pdFALSE then the bits set in the event group are not altered when the call to xEventGroupWaitBits() returns.
- `xWaitForAllBits`: If xWaitForAllBits is set to pdTRUE then xEventGroupWaitBits() will return when either all the bits in uxBitsToWaitFor are set or the specified block time expires. If xWaitForAllBits is set to pdFALSE then xEventGroupWaitBits() will return when any one of the bits set in uxBitsToWaitFor is set or the specified block time expires. The block time is specified by the xTicksToWait parameter.
- `xTicksToWait`: The maximum amount of time (specified in 'ticks') to wait for one/all (depending on the xWaitForAllBits value) of the bits specified by uxBitsToWaitFor to become set.

EventBits_t **xEventGroupClearBits**(*EventGroupHandle_t xEventGroup*, **const** *EventBits_t uxBitsToClear*)

Clear bits within an event group. This function cannot be called from an interrupt.

Example usage:

```
#define BIT_0    ( 1 << 0 )
#define BIT_4    ( 1 << 4 )

void aFunction( EventGroupHandle_t xEventGroup )
{
EventBits_t uxBits;

    // Clear bit 0 and bit 4 in xEventGroup.
    uxBits = xEventGroupClearBits(
                          xEventGroup,    // The event group being updated.
                          BIT_0 | BIT_4 );// The bits being cleared.

    if( ( uxBits & ( BIT_0 | BIT_4 ) ) == ( BIT_0 | BIT_4 ) )
    {
        // Both bit 0 and bit 4 were set before xEventGroupClearBits() was
        // called.  Both will now be clear (not set).
```

(下页继续)

(续上页)

```
    }
    else if( ( uxBits & BIT_0 ) != 0 )
    {
        // Bit 0 was set before xEventGroupClearBits() was called.  It will
        // now be clear.
    }
    else if( ( uxBits & BIT_4 ) != 0 )
    {
        // Bit 4 was set before xEventGroupClearBits() was called.  It will
        // now be clear.
    }
    else
    {
        // Neither bit 0 nor bit 4 were set in the first place.
    }
}
```

Return The value of the event group before the specified bits were cleared.

Parameters

- `xEventGroup`: The event group in which the bits are to be cleared.
- `uxBitsToClear`: A bitwise value that indicates the bit or bits to clear in the event group. For example, to clear bit 3 only, set uxBitsToClear to 0x08. To clear bit 3 and bit 0 set uxBitsToClear to 0x09.

EventBits_t **xEventGroupSetBits**(*EventGroupHandle_t xEventGroup*, **const** *EventBits_t uxBitsToSet*)

Set bits within an event group. This function cannot be called from an interrupt. xEventGroupSetBitsFromISR() is a version that can be called from an interrupt.

Setting bits in an event group will automatically unblock tasks that are blocked waiting for the bits.

Example usage:

```
#define BIT_0    ( 1 << 0 )
#define BIT_4    ( 1 << 4 )

void aFunction( EventGroupHandle_t xEventGroup )
{
EventBits_t uxBits;
```

(下页继续)

(续上页)

```
    // Set bit 0 and bit 4 in xEventGroup.
    uxBits = xEventGroupSetBits(
                        xEventGroup,    // The event group being updated.
                        BIT_0 | BIT_4 );// The bits being set.

    if( ( uxBits & ( BIT_0 | BIT_4 ) ) == ( BIT_0 | BIT_4 ) )
    {
        // Both bit 0 and bit 4 remained set when the function returned.
    }
    else if( ( uxBits & BIT_0 ) != 0 )
    {
        // Bit 0 remained set when the function returned, but bit 4 was
        // cleared.  It might be that bit 4 was cleared automatically as a
        // task that was waiting for bit 4 was removed from the Blocked
        // state.
    }
    else if( ( uxBits & BIT_4 ) != 0 )
    {
        // Bit 4 remained set when the function returned, but bit 0 was
        // cleared.  It might be that bit 0 was cleared automatically as a
        // task that was waiting for bit 0 was removed from the Blocked
        // state.
    }
    else
    {
        // Neither bit 0 nor bit 4 remained set.  It might be that a task
        // was waiting for both of the bits to be set, and the bits were
        // cleared as the task left the Blocked state.
    }
}
```

{c}

Return The value of the event group at the time the call to xEventGroupSetBits() returns. There are two reasons why the returned value might have the bits specified by the uxBitsToSet parameter cleared. First, if setting a bit results in a task that was waiting for the bit leaving the blocked state then it is possible the bit will be cleared automatically (see the xClearBitOnExit parameter of xEventGroupWaitBits()). Second, any unblocked (or otherwise Ready state) task that has a priority above that of the task that called xEventGroupSetBits() will execute and may change the event group value before the call to xEventGroupSetBits() returns.

Parameters

- `xEventGroup`: The event group in which the bits are to be set.
- `uxBitsToSet`: A bitwise value that indicates the bit or bits to set. For example, to set bit 3 only, set uxBitsToSet to 0x08. To set bit 3 and bit 0 set uxBitsToSet to 0x09.

EventBits_t **xEventGroupSync**(*EventGroupHandle_t xEventGroup*, **const** *EventBits_t uxBitsToSet*, **const** *EventBits_t uxBitsToWaitFor*, TickType_t *xTicksToWait*)

Atomically set bits within an event group, then wait for a combination of bits to be set within the same event group. This functionality is typically used to synchronise multiple tasks, where each task has to wait for the other tasks to reach a synchronisation point before proceeding.

This function cannot be used from an interrupt.

The function will return before its block time expires if the bits specified by the uxBitsToWait parameter are set, or become set within that time. In this case all the bits specified by uxBitsToWait will be automatically cleared before the function returns.

Example usage:

```
// Bits used by the three tasks.
#define TASK_0_BIT     ( 1 << 0 )
#define TASK_1_BIT     ( 1 << 1 )
#define TASK_2_BIT     ( 1 << 2 )

#define ALL_SYNC_BITS ( TASK_0_BIT | TASK_1_BIT | TASK_2_BIT )

// Use an event group to synchronise three tasks.  It is assumed this event
// group has already been created elsewhere.
EventGroupHandle_t xEventBits;

void vTask0( void *pvParameters )
{
EventBits_t uxReturn;
TickType_t xTicksToWait = 100 / portTICK_PERIOD_MS;

    for( ;; )
    {
       // Perform task functionality here.

       // Set bit 0 in the event flag to note this task has reached the
       // sync point.  The other two tasks will set the other two bits defined
       // by ALL_SYNC_BITS.  All three tasks have reached the synchronisation
       // point when all the ALL_SYNC_BITS are set.  Wait a maximum of 100ms
       // for this to happen.
       uxReturn = xEventGroupSync( xEventBits, TASK_0_BIT, ALL_SYNC_BITS,␣
↪xTicksToWait );
```

(下页继续)

(续上页)

```
        if( ( uxReturn & ALL_SYNC_BITS ) == ALL_SYNC_BITS )
        {
            // All three tasks reached the synchronisation point before the call
            // to xEventGroupSync() timed out.
        }
    }
}

void vTask1( void *pvParameters )
{
    for( ;; )
    {
        // Perform task functionality here.

        // Set bit 1 in the event flag to note this task has reached the
        // synchronisation point.  The other two tasks will set the other two
        // bits defined by ALL_SYNC_BITS.  All three tasks have reached the
        // synchronisation point when all the ALL_SYNC_BITS are set.  Wait
        // indefinitely for this to happen.
        xEventGroupSync( xEventBits, TASK_1_BIT, ALL_SYNC_BITS, portMAX_DELAY );

        // xEventGroupSync() was called with an indefinite block time, so
        // this task will only reach here if the syncrhonisation was made by all
        // three tasks, so there is no need to test the return value.
    }
}

void vTask2( void *pvParameters )
{
    for( ;; )
    {
        // Perform task functionality here.

        // Set bit 2 in the event flag to note this task has reached the
        // synchronisation point.  The other two tasks will set the other two
        // bits defined by ALL_SYNC_BITS.  All three tasks have reached the
        // synchronisation point when all the ALL_SYNC_BITS are set.  Wait
        // indefinitely for this to happen.
        xEventGroupSync( xEventBits, TASK_2_BIT, ALL_SYNC_BITS, portMAX_DELAY );
```

(下页继续)

(续上页)

```
        // xEventGroupSync() was called with an indefinite block time, so
        // this task will only reach here if the syncrhonisation was made by all
        // three tasks, so there is no need to test the return value.
    }
}
```

Return The value of the event group at the time either the bits being waited for became set, or the block time expired. Test the return value to know which bits were set. If xEventGroupSync() returned because its timeout expired then not all the bits being waited for will be set. If xEventGroupSync() returned because all the bits it was waiting for were set then the returned value is the event group value before any bits were automatically cleared.

Parameters

- `xEventGroup`: The event group in which the bits are being tested. The event group must have previously been created using a call to xEventGroupCreate().
- `uxBitsToSet`: The bits to set in the event group before determining if, and possibly waiting for, all the bits specified by the uxBitsToWait parameter are set.
- `uxBitsToWaitFor`: A bitwise value that indicates the bit or bits to test inside the event group. For example, to wait for bit 0 and bit 2 set uxBitsToWaitFor to 0x05. To wait for bits 0 and bit 1 and bit 2 set uxBitsToWaitFor to 0x07. Etc.
- `xTicksToWait`: The maximum amount of time (specified in 'ticks') to wait for all of the bits specified by uxBitsToWaitFor to become set.

EventBits_t **xEventGroupGetBitsFromISR**(*EventGroupHandle_t xEventGroup*)

A version of xEventGroupGetBits() that can be called from an ISR.

Return The event group bits at the time xEventGroupGetBitsFromISR() was called.

Parameters

- `xEventGroup`: The event group being queried.

void **vEventGroupDelete**(*EventGroupHandle_t xEventGroup*)

Delete an event group that was previously created by a call to xEventGroupCreate(). Tasks that are blocked on the event group will be unblocked and obtain 0 as the event group's value.

Parameters

- `xEventGroup`: The event group being deleted.

Macros

xEventGroupClearBitsFromISR(xEventGroup, uxBitsToClear)

A version of xEventGroupClearBits() that can be called from an interrupt.

Setting bits in an event group is not a deterministic operation because there are an unknown number of tasks that may be waiting for the bit or bits being set. FreeRTOS does not allow nondeterministic operations to be performed while interrupts are disabled, so protects event groups that are accessed from tasks by suspending the scheduler rather than disabling interrupts. As a result event groups cannot be accessed directly from an interrupt service routine. Therefore xEventGroupClearBitsFromISR() sends a message to the timer task to have the clear operation performed in the context of the timer task.

Example usage:

```
#define BIT_0    ( 1 << 0 )
#define BIT_4    ( 1 << 4 )

// An event group which it is assumed has already been created by a call to
// xEventGroupCreate().
EventGroupHandle_t xEventGroup;

void anInterruptHandler( void )
{
     // Clear bit 0 and bit 4 in xEventGroup.
     xResult = xEventGroupClearBitsFromISR(
                          xEventGroup,     // The event group being updated.
                          BIT_0 | BIT_4 ); // The bits being set.

     if( xResult == pdPASS )
     {
         // The message was posted successfully.
     }
}
```

Return If the request to execute the function was posted successfully then pdPASS is returned, otherwise pdFALSE is returned. pdFALSE will be returned if the timer service queue was full.

Parameters

- `xEventGroup`: The event group in which the bits are to be cleared.
- `uxBitsToClear`: A bitwise value that indicates the bit or bits to clear. For example, to clear bit 3 only, set uxBitsToClear to 0x08. To clear bit 3 and bit 0 set uxBitsToClear to 0x09.

xEventGroupSetBitsFromISR(xEventGroup, uxBitsToSet, pxHigherPriorityTaskWoken)

A version of xEventGroupSetBits() that can be called from an interrupt.

Setting bits in an event group is not a deterministic operation because there are an unknown number of tasks that may be waiting for the bit or bits being set. FreeRTOS does not allow nondeterministic operations to be performed in interrupts or from critical sections. Therefore xEventGroupSetBitFromISR() sends a message to the timer task to have the set operation performed in the context of the timer task - where a scheduler lock is used in place of a critical section.

Example usage:

```
#define BIT_0    ( 1 << 0 )
#define BIT_4    ( 1 << 4 )

// An event group which it is assumed has already been created by a call to
// xEventGroupCreate().
EventGroupHandle_t xEventGroup;

void anInterruptHandler( void )
{
BaseType_t xHigherPriorityTaskWoken, xResult;

     // xHigherPriorityTaskWoken must be initialised to pdFALSE.
     xHigherPriorityTaskWoken = pdFALSE;

     // Set bit 0 and bit 4 in xEventGroup.
     xResult = xEventGroupSetBitsFromISR(
                         xEventGroup,    // The event group being updated.
                         BIT_0 | BIT_4   // The bits being set.
                         &xHigherPriorityTaskWoken );

     // Was the message posted successfully?
     if( xResult == pdPASS )
     {
         // If xHigherPriorityTaskWoken is now set to pdTRUE then a context
         // switch should be requested.  The macro used is port specific and
         // will be either portYIELD_FROM_ISR() or portEND_SWITCHING_ISR() -
         // refer to the documentation page for the port being used.
         portYIELD_FROM_ISR( xHigherPriorityTaskWoken );
     }
}
```

Return If the request to execute the function was posted successfully then pdPASS is returned,

otherwise pdFALSE is returned. pdFALSE will be returned if the timer service queue was full.

Parameters

- `xEventGroup`: The event group in which the bits are to be set.
- `uxBitsToSet`: A bitwise value that indicates the bit or bits to set. For example, to set bit 3 only, set uxBitsToSet to 0x08. To set bit 3 and bit 0 set uxBitsToSet to 0x09.
- `pxHigherPriorityTaskWoken`: As mentioned above, calling this function will result in a message being sent to the timer daemon task. If the priority of the timer daemon task is higher than the priority of the currently running task (the task the interrupt interrupted) then *pxHigherPriorityTaskWoken will be set to pdTRUE by xEventGroupSetBitsFromISR(), indicating that a context switch should be requested before the interrupt exits. For that reason *pxHigherPriorityTaskWoken must be initialised to pdFALSE. See the example code below.

`xEventGroupGetBits`(xEventGroup)

Returns the current value of the bits in an event group. This function cannot be used from an interrupt.

Return The event group bits at the time xEventGroupGetBits() was called.

Parameters

- `xEventGroup`: The event group being queried.

Type Definitions

`typedef` void ***EventGroupHandle_t**

An event group is a collection of bits to which an application can assign a meaning. For example, an application may create an event group to convey the status of various CAN bus related events in which bit 0 might mean "A CAN

message has been received and is ready for processing" , bit 1 might mean "The application has queued a message that is ready for sending onto the CAN network" , and bit 2 might mean "It is time to send a SYNC message onto the CAN network" etc. A task can then test the bit values to see which events are active, and optionally enter the Blocked state to wait for a specified bit or a group of specified bits to be active. To continue the CAN bus example, a CAN controlling task can enter the Blocked state (and therefore not consume any processing time) until either bit 0, bit 1 or bit 2 are active, at which time the bit that was actually active would inform the task which action it had to take (process a received message, send a message, or send a SYNC).

The event groups implementation contains intelligence to avoid race conditions that would otherwise occur were an application to use a simple variable for the same purpose. This is particularly important with respect to when a bit within an event group is to be cleared, and when bits have to be set and then tested atomically - as is the case where event groups are used to create a synchronisation point between multiple tasks (a 'rendezvous'). event_groups.h

Type by which event groups are referenced. For example, a call to xEventGroupCreate() returns an EventGroupHandle_t variable that can then be used as a parameter to other event group functions.

typedef TickType_t **EventBits_t**

2.7.9 FreeRTOS Additions

Overview

ESP-IDF FreeRTOS is based on the Xtensa port of FreeRTOS v8.2.0 with significant modifications for SMP compatibility (see *ESP-IDF FreeRTOS SMP Changes*). However various features specific to ESP-IDF FreeRTOS have been added. The features are as follows:

Ring Buffers: Ring buffers were added to provide a form of buffer that could accept entries of arbitrary lengths.

Hooks: ESP-IDF FreeRTOS hooks provides support for registering extra Idle and Tick hooks at run time. Moreover, the hooks can be asymmetric amongst both CPUs.

Ring Buffers

The ESP-IDF FreeRTOS ring buffer is a strictly FIFO buffer that supports arbitrarily sized items. Ring buffers are a more memory efficient alternative to FreeRTOS queues in situations where the size of items is variable. The capacity of a ring buffer is not measured by the number of items it can store, but rather by the amount of memory used for storing items. You may apply for a piece of memory on the ring buffer to send an item, or just use the API to copy your data and send (according to the send API you call). For efficiency reasons, **items are always retrieved from the ring buffer by reference**. As a result, all retrieved items *must also be returned* in order for them to be removed from the ring buffer completely. The ring buffers are split into the three following types:

No-Split buffers will guarantee that an item is stored in contiguous memory and will not attempt to split an item under any circumstances. Use no-split buffers when items must occupy contiguous memory. *Only this buffer type allows you getting the data item address and writting to the item by yourself.*

Allow-Split buffers will allow an item to be split when wrapping around if doing so will allow the item to be stored. Allow-split buffers are more memory efficient than no-split buffers but can return an item in two parts when retrieving.

Byte buffers do not store data as separate items. All data is stored as a sequence of bytes, and any number of bytes and be sent or retrieved each time. Use byte buffers when separate items do not need to be maintained (e.g. a byte stream).

注解: No-split/allow-split buffers will always store items at 32-bit aligned addresses. Therefore when retrieving an item, the item pointer is guaranteed to be 32-bit aligned. This is useful especially when you

need to send some data to the DMA.

注解: Each item stored in no-split/allow-split buffers will **require an additional 8 bytes for a header**. Item sizes will also be rounded up to a 32-bit aligned size (multiple of 4 bytes), however the true item size is recorded within the header. The sizes of no-split/allow-split buffers will also be rounded up when created.

Usage

The following example demonstrates the usage of *xRingbufferCreate()* and *xRingbufferSend()* to create a ring buffer then send an item to it.

```
#include "freertos/ringbuf.h"
static char tx_item[] = "test_item";

...

    //Create ring buffer
    RingbufHandle_t buf_handle;
    buf_handle = xRingbufferCreate(1028, RINGBUF_TYPE_NOSPLIT);
    if (buf_handle == NULL) {
        printf("Failed to create ring buffer\n");
    }

    //Send an item
    UBaseType_t res =  xRingbufferSend(buf_handle, tx_item, sizeof(tx_item), pdMS_TO_
↪TICKS(1000));
    if (res != pdTRUE) {
        printf("Failed to send item\n");
    }
```

The following example demonstrates the usage of *xRingbufferSendAcquire()* and *xRingbufferSendComplete()* instead of *xRingbufferSend()* to apply for the memory on the ring buffer (of type *RINGBUF_TYPE_NOSPLIT*) and then send an item to it. This way adds one more step, but allows getting the address of the memory to write to, and writing to the memory yourself.

```
#include "freertos/ringbuf.h"
#include "soc/lldesc.h"

typedef struct {
```

(下页继续)

(续上页)

```
    lldesc_t dma_desc;
    uint8_t buf[1];
} dma_item_t;

#define DMA_ITEM_SIZE(N) (sizeof(lldesc_t)+(((N)+3)&(~3)))

...

    //Retrieve space for DMA descriptor and corresponding data buffer
    //This has to be done with SendAcquire, or the address may be different when copy
    dma_item_t item;
    UBaseType_t res =  xRingbufferSendAcquire(buf_handle,
                       &item, DMA_ITEM_SIZE(buffer_size), pdMS_TO_TICKS(1000));
    if (res != pdTRUE) {
        printf("Failed to acquire memory for item\n");
    }
    item->dma_desc = (lldesc_t) {
        .size = buffer_size,
        .length = buffer_size,
        .eof = 0,
        .owner = 1,
        .buf = &item->buf,
    };
    //Actually send to the ring buffer for consumer to use
    res = xRingbufferSendComplete(buf_handle, &item);
    if (res != pdTRUE) {
        printf("Failed to send item\n");
    }
```

The following example demonstrates retrieving and returning an item from a **no-split ring buffer** using *xRingbufferReceive()* and *vRingbufferReturnItem()*

```
...

    //Receive an item from no-split ring buffer
    size_t item_size;
    char *item = (char *)xRingbufferReceive(buf_handle, &item_size, pdMS_TO_TICKS(1000));

    //Check received item
    if (item != NULL) {
```

(下页继续)

(续上页)

```
        //Print item
        for (int i = 0; i < item_size; i++) {
            printf("%c", item[i]);
        }
        printf("\n");
        //Return Item
        vRingbufferReturnItem(buf_handle, (void *)item);
    } else {
        //Failed to receive item
        printf("Failed to receive item\n");
    }
```

The following example demonstrates retrieving and returning an item from an **allow-split ring buffer** using *xRingbufferReceiveSplit()* and *vRingbufferReturnItem()*

```
...

    //Receive an item from allow-split ring buffer
    size_t item_size1, item_size2;
    char *item1, *item2;
    BaseType_t ret = xRingbufferReceiveSplit(buf_handle, (void **)&item1, (void **)&
↪item2, &item_size1, &item_size2, pdMS_TO_TICKS(1000));

    //Check received item
    if (ret == pdTRUE && item1 != NULL) {
        for (int i = 0; i < item_size1; i++) {
            printf("%c", item1[i]);
        }
        vRingbufferReturnItem(buf_handle, (void *)item1);
        //Check if item was split
        if (item2 != NULL) {
            for (int i = 0; i < item_size2; i++) {
                printf("%c", item2[i]);
            }
            vRingbufferReturnItem(buf_handle, (void *)item2);
        }
        printf("\n");
    } else {
        //Failed to receive item
        printf("Failed to receive item\n");
```

(下页继续)

(续上页)

```
    }
```

The following example demonstrates retrieving and returning an item from a **byte buffer** using *xRingbufferReceiveUpTo()* and *vRingbufferReturnItem()*

```
...

    //Receive data from byte buffer
    size_t item_size;
    char *item = (char *)xRingbufferReceiveUpTo(buf_handle, &item_size, pdMS_TO_
↪TICKS(1000), sizeof(tx_item));

    //Check received data
    if (item != NULL) {
        //Print item
        for (int i = 0; i < item_size; i++) {
            printf("%c", item[i]);
        }
        printf("\n");
        //Return Item
        vRingbufferReturnItem(buf_handle, (void *)item);
    } else {
        //Failed to receive item
        printf("Failed to receive item\n");
    }
```

For ISR safe versions of the functions used above, call *xRingbufferSendFromISR()*, *xRingbufferReceiveFromISR()*, *xRingbufferReceiveSplitFromISR()*, *xRingbufferReceiveUpToFromISR()*, and *vRingbufferReturnItemFromISR()*

Sending to Ring Buffer

The following diagrams illustrate the differences between no-split/allow-split buffers and byte buffers with regards to sending items/data. The diagrams assume that three items of sizes **18, 3, and 27 bytes** are sent respectively to a **buffer of 128 bytes**.

For no-split/allow-split buffers, a header of 8 bytes precedes every data item. Furthermore, the space occupied by each item is **rounded up to the nearest 32-bit aligned size** in order to maintain overall 32-bit alignment. However the true size of the item is recorded inside the header which will be returned when the item is retrieved.

Referring to the diagram above, the 18, 3, and 27 byte items are **rounded up to 20, 4, and 28 bytes**

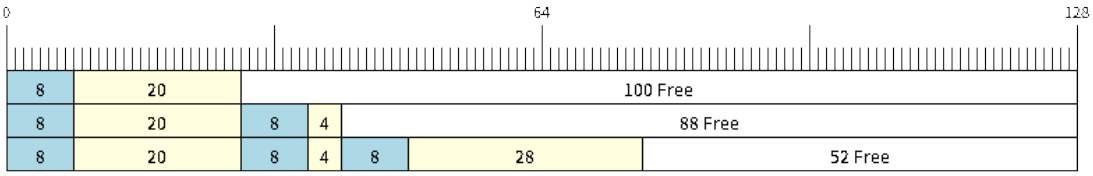


图 26: Sending items to no-split/allow-split ring buffers

respectively. An 8 byte header is then added in front of each item.

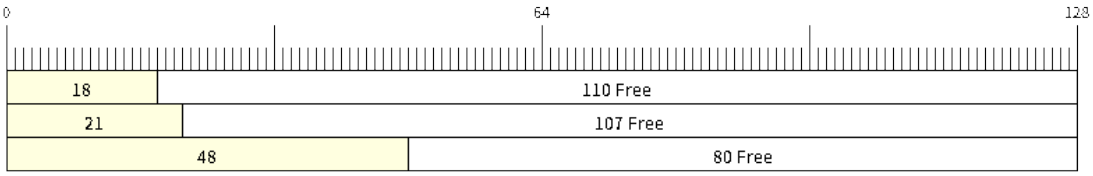


图 27: Sending items to byte buffers

Byte buffers treat data as a sequence of bytes and does not incur any overhead (no headers). As a result, all data sent to a byte buffer is merged into a single item.

Referring to the diagram above, the 18, 3, and 27 byte items are sequentially written to the byte buffer and **merged into a single item of 48 bytes**.

Using SendAcquire and SendComplete

Items in no-split buffers are acquired (by SendAcquire) in strict FIFO order and must be sent to the buffer by SendComplete for the data to be accessible by the consumer. Multiple items can be sent or acquired without calling SendComplete, and the items do not necessarily need to be completed in the order they were acquired. However the receiving of data items must occur in FIFO order, therefore not calling SendComplete the earliest acquired item will prevent the subsequent items from being received.

The following diagrams illustrate what will happen when SendAcquire/SendComplete don't happen in the same order. At the beginning, there is already an data item of 16 bytes sent to the ring buffer. Then SendAcquire is called to acquire space of 20, 8, 24 bytes on the ring buffer.

After that, we fill (use) the buffers, and send them to the ring buffer by SendComplete in the order of 8, 24, 20. When 8 bytes and 24 bytes data are sent, the consumer still can only get the 16 bytes data item. Due to the usage if 20 bytes item is not complete, it's not available, nor the following data items.

When the 20 bytes item is finally completed, all the 3 data items can be received now, in the order of 20, 8, 24 bytes, right after the 16 bytes item existing in the buffer at the beginning.

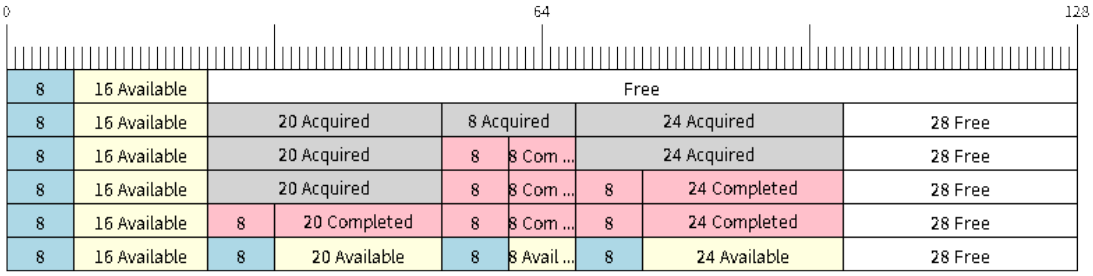


图 28: SendAcquire/SendComplete items in no-split ring buffers

Allow-split/byte buffers do not allow using SendAcquire/SendComplete since acquired buffers are required to be complete (not wrapped).

Wrap around

The following diagrams illustrate the differences between no-split, allow-split, and byte buffers when a sent item requires a wrap around. The diagrams assumes a buffer of **128 bytes** with **56 bytes of free space that wraps around** and a sent item of **28 bytes**.

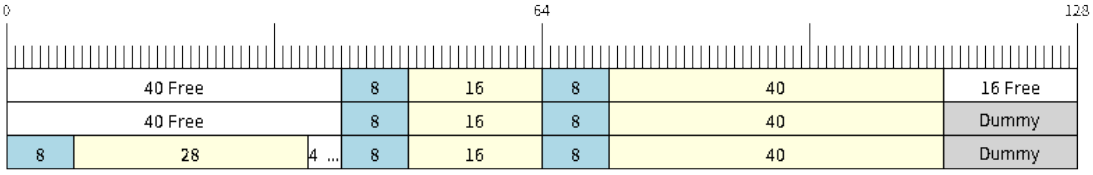


图 29: Wrap around in no-split buffers

No-split buffers will **only store an item in continuous free space and will not split an item under any circumstances**. When the free space at the tail of the buffer is insufficient to completely store the item and its header, the free space at the tail will be **marked as dummy data**. The buffer will then wrap around and store the item in the free space at the head of the buffer.

Referring to the diagram above, the 16 bytes of free space at the tail of the buffer is insufficient to store the 28 byte item. Therefore the 16 bytes is marked as dummy data and the item is written to the free space at the head of the buffer instead.

Allow-split buffers will attempt to **split the item into two parts** when the free space at the tail of the buffer is insufficient to store the item data and its header. Both parts of the split item will have their own headers (therefore incurring an extra 8 bytes of overhead).

Referring to the diagram above, the 16 bytes of free space at the tail of the buffer is insufficient to store the

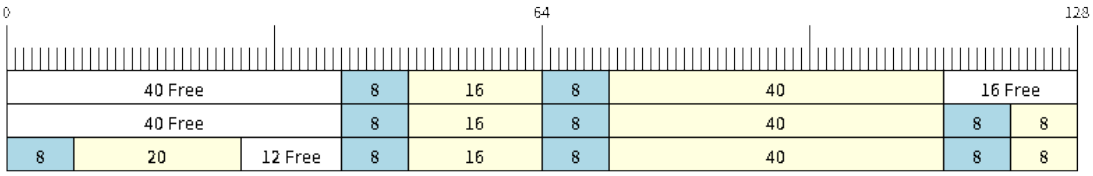


图 30: Wrap around in allow-split buffers

28 byte item. Therefore the item is split into two parts (8 and 20 bytes) and written as two parts to the buffer.

注解: Allow-split buffers treats the both parts of the split item as two separate items, therefore call *xRingbufferReceiveSplit()* instead of *xRingbufferReceive()* to receive both parts of a split item in a thread safe manner.

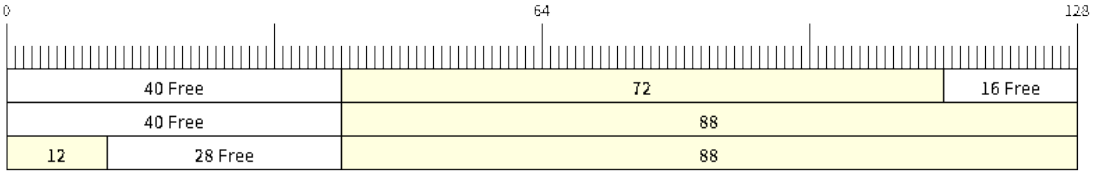


图 31: Wrap around in byte buffers

Byte buffers will **store as much data as possible into the free space at the tail of buffer**. The remaining data will then be stored in the free space at the head of the buffer. No overhead is incurred when wrapping around in byte buffers.

Referring to the diagram above, the 16 bytes of free space at the tail of the buffer is insufficient to completely store the 28 bytes of data. Therefore the 16 bytes of free space is filled with data, and the remaining 12 bytes are written to the free space at the head of the buffer. The buffer now contains data in two separate continuous parts, and each part continuous will be treated as a separate item by the byte buffer.

Retrieving/Returning

The following diagrams illustrates the differences between no-split/allow-split and byte buffers in retrieving and returning data.

Items in no-split/allow-split buffers are **retrieved in strict FIFO order** and **must be returned** for the occupied space to be freed. Multiple items can be retrieved before returning, and the items do not

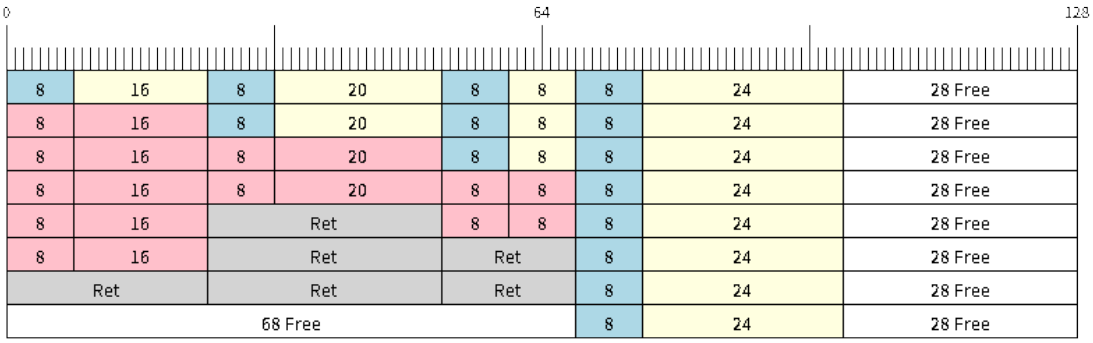


图 32: Retrieving/Returning items in no-split/allow-split ring buffers

necessarily need to be returned in the order they were retrieved. However the freeing of space must occur in FIFO order, therefore not returning the earliest retrieved item will prevent the space of subsequent items from being freed.

Referring to the diagram above, the **16, 20, and 8 byte items are retrieved in FIFO order**. However the items are not returned in they were retrieved (20, 8, 16). As such, the space is not freed until the first item (16 byte) is returned.

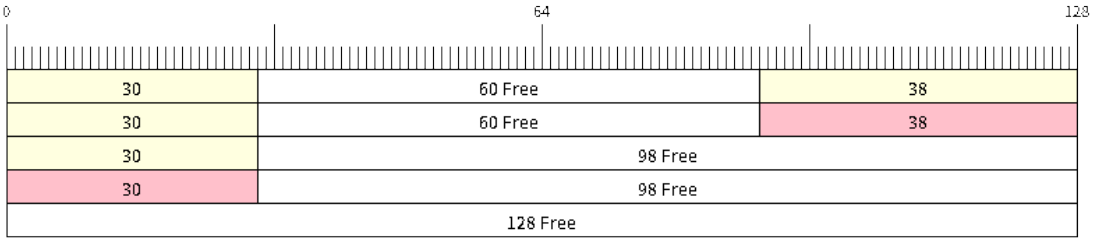


图 33: Retrieving/Returning data in byte buffers

Byte buffers **do not allow multiple retrievals before returning** (every retrieval must be followed by a return before another retrieval is permitted). When using *xRingbufferReceive()* or *xRingbufferReceiveFromISR()*, all continuous stored data will be retrieved. *xRingbufferReceiveUpTo()* or *xRingbufferReceiveUpToFromISR()* can be used to restrict the maximum number of bytes retrieved. Since every retrieval must be followed by a return, the space will be freed as soon as the data is returned.

Referring to the diagram above, the 38 bytes of continuous stored data at the tail of the buffer is retrieved, returned, and freed. The next call to *xRingbufferReceive()* or *xRingbufferReceiveFromISR()* then wraps around and does the same to the 30 bytes of continuous stored data at the head of the buffer.

Ring Buffers with Queue Sets

Ring buffers can be added to FreeRTOS queue sets using *xRingbufferAddToQueueSetRead()* such that every time a ring buffer receives an item or data, the queue set is notified. Once added to a queue set, every attempt to retrieve an item from a ring buffer should be preceded by a call to *xQueueSelectFromSet()*. To check whether the selected queue set member is the ring buffer, call *xRingbufferCanRead()*.

The following example demonstrates queue set usage with ring buffers.

```
#include "freertos/queue.h"
#include "freertos/ringbuf.h"

...

    //Create ring buffer and queue set
    RingbufHandle_t buf_handle = xRingbufferCreate(1028, RINGBUF_TYPE_NOSPLIT);
    QueueSetHandle_t queue_set = xQueueCreateSet(3);

    //Add ring buffer to queue set
    if (xRingbufferAddToQueueSetRead(buf_handle, queue_set) != pdTRUE) {
        printf("Failed to add to queue set\n");
    }

...

    //Block on queue set
    xQueueSetMemberHandle member = xQueueSelectFromSet(queue_set, pdMS_TO_TICKS(1000));

    //Check if member is ring buffer
    if (member != NULL && xRingbufferCanRead(buf_handle, member) == pdTRUE) {
        //Member is ring buffer, receive item from ring buffer
        size_t item_size;
        char *item = (char *)xRingbufferReceive(buf_handle, &item_size, 0);

        //Handle item
        ...

    } else {
        ...
    }
```

Ring Buffers with Static Allocation

The *xRingbufferCreateStatic()* can be used to create ring buffers with specific memory requirements (such as a ring buffer being allocated in external RAM). All blocks of memory used by a ring buffer must be manually allocated beforehand then passed to the *xRingbufferCreateStatic()* to be initialized as a ring buffer. These blocks include the following:

- The ring buffer's data structure of type *StaticRingbuffer_t*
- The ring buffer's storage area of size `xBufferSize`. Note that `xBufferSize` must be 32-bit aligned for no-split/allow-split buffers.

The manner in which these blocks are allocated will depend on the users requirements (e.g. all blocks being statically declared, or dynamically allocated with specific capabilities such as external RAM).

注解: The *CONFIG_FREERTOS_SUPPORT_STATIC_ALLOCATION* option must be enabled in *menuconfig* for statically allocated ring buffers to be available.

注解: When deleting a ring buffer created via *xRingbufferCreateStatic()*, the function *vRingbufferDelete()* will not free any of the memory blocks. This must be done manually by the user after *vRingbufferDelete()* is called.

The code snippet below demonstrates a ring buffer being allocated entirely in external RAM.

```
#include "freertos/ringbuf.h"
#include "freertos/semphr.h"
#include "esp_heap_caps.h"

#define BUFFER_SIZE     400     //32-bit aligned size
#define BUFFER_TYPE     RINGBUF_TYPE_NOSPLIT
...

//Allocate ring buffer data structure and storage area into external RAM
StaticRingbuffer_t *buffer_struct = (StaticRingbuffer_t *)heap_caps_
↪malloc(sizeof(StaticRingbuffer_t), MALLOC_CAP_SPIRAM);
uint8_t *buffer_storage = (uint8_t *)heap_caps_malloc(sizeof(uint8_t)*BUFFER_SIZE,␣
↪MALLOC_CAP_SPIRAM);

//Create a ring buffer with manually allocated memory
RingbufHandle_t handle = xRingbufferCreateStatic(BUFFER_SIZE, BUFFER_TYPE, buffer_
↪storage, buffer_struct);
```

(下页继续)

(续上页)

```
...

//Delete the ring buffer after used
vRingbufferDelete(handle);

//Manually free all blocks of memory
free(buffer_struct);
free(buffer_storage);
```

Ring Buffer API Reference

注解: Ideally, ring buffers can be used with multiple tasks in an SMP fashion where the **highest priority task will always be serviced first.** However due to the usage of binary semaphores in the ring buffer's underlying implementation, priority inversion may occur under very specific circumstances.

The ring buffer governs sending by a binary semaphore which is given whenever space is freed on the ring buffer. The highest priority task waiting to send will repeatedly take the semaphore until sufficient free space becomes available or until it times out. Ideally this should prevent any lower priority tasks from being serviced as the semaphore should always be given to the highest priority task.

However in between iterations of acquiring the semaphore, there is a **gap in the critical section** which may permit another task (on the other core or with an even higher priority) to free some space on the ring buffer and as a result give the semaphore. Therefore the semaphore will be given before the highest priority task can re-acquire the semaphore. This will result in the **semaphore being acquired by the second highest priority task** waiting to send, hence causing priority inversion.

This side effect will not affect ring buffer performance drastically given if the number of tasks using the ring buffer simultaneously is low, and the ring buffer is not operating near maximum capacity.

Header File

- esp_ringbuf/include/freertos/ringbuf.h

Functions

RingbufHandle_t **xRingbufferCreate**(size_t *xBufferSize*, *RingbufferType_t* *xBufferType*)

Create a ring buffer.

Note xBufferSize of no-split/allow-split buffers will be rounded up to the nearest 32-bit aligned size.

Return A handle to the created ring buffer, or NULL in case of error.

Parameters

- `xBufferSize`: Size of the buffer in bytes. Note that items require space for overhead in no-split/allow-split buffers
- `xBufferType`: Type of ring buffer, see documentation.

RingbufHandle_t **xRingbufferCreateNoSplit**(size_t *xItemSize*, size_t *xItemNum*)

Create a ring buffer of type RINGBUF_TYPE_NOSPLIT for a fixed item_size.

This API is similar to xRingbufferCreate(), but it will internally allocate additional space for the headers.

Return A RingbufHandle_t handle to the created ring buffer, or NULL in case of error.

Parameters

- `xItemSize`: Size of each item to be put into the ring buffer
- `xItemNum`: Maximum number of items the buffer needs to hold simultaneously

RingbufHandle_t **xRingbufferCreateStatic**(size_t *xBufferSize*, *RingbufferType_t* *xBufferType*, uint8_t **pucRingbufferStorage*, *StaticRingbuffer_t* **pxStaticRingbuffer*)

Create a ring buffer but manually provide the required memory.

Note The CONFIG_FREERTOS_SUPPORT_STATIC_ALLOCATION option must be enabled for this to be available

Note xBufferSize of no-split/allow-split buffers MUST be 32-bit aligned.

Return A handle to the created ring buffer

Parameters

- `xBufferSize`: Size of the buffer in bytes.
- `xBufferType`: Type of ring buffer, see documentation
- `pucRingbufferStorage`: Pointer to the ring buffer's storage area. Storage area must of the same size as specified by xBufferSize
- `pxStaticRingbuffer`: Pointed to a struct of type StaticRingbuffer_t which will be used to hold the ring buffer's data structure

BaseType_t **xRingbufferSend**(*RingbufHandle_t* *xRingbuffer*, **const** void **pvItem*, size_t *xItemSize*, TickType_t *xTicksToWait*)

Insert an item into the ring buffer.

Attempt to insert an item into the ring buffer. This function will block until enough free space is available or until it times out.

Note For no-split/allow-split ring buffers, the actual size of memory that the item will occupy will be rounded up to the nearest 32-bit aligned size. This is done to ensure all items are always stored in 32-bit aligned fashion.

Return

- pdTRUE if succeeded
- pdFALSE on time-out or when the data is larger than the maximum permissible size of the buffer

Parameters

- `xRingbuffer`: Ring buffer to insert the item into
- `pvItem`: Pointer to data to insert. NULL is allowed if xItemSize is 0.
- `xItemSize`: Size of data to insert.
- `xTicksToWait`: Ticks to wait for room in the ring buffer.

BaseType_t **xRingbufferSendFromISR**(*RingbufHandle_t xRingbuffer*, **const** void **pvItem*, size_t *xItemSize*, BaseType_t **pxHigherPriorityTaskWoken*)

Insert an item into the ring buffer in an ISR.

Attempt to insert an item into the ring buffer from an ISR. This function will return immediately if there is insufficient free space in the buffer.

Note For no-split/allow-split ring buffers, the actual size of memory that the item will occupy will be rounded up to the nearest 32-bit aligned size. This is done to ensure all items are always stored in 32-bit aligned fashion.

Return

- pdTRUE if succeeded
- pdFALSE when the ring buffer does not have space.

Parameters

- `xRingbuffer`: Ring buffer to insert the item into
- `pvItem`: Pointer to data to insert. NULL is allowed if xItemSize is 0.
- `xItemSize`: Size of data to insert.
- `pxHigherPriorityTaskWoken`: Value pointed to will be set to pdTRUE if the function woke up a higher priority task.

BaseType_t **xRingbufferSendAcquire**(*RingbufHandle_t xRingbuffer*, void **ppvItem*, size_t *xItemSize*, TickType_t *xTicksToWait*)

Acquire memory from the ring buffer to be written to by an external source and to be sent later.

Attempt to allocate buffer for an item to be sent into the ring buffer. This function will block until enough free space is available or until it timesout.

The item, as well as the following items `SendAcquire` or `Send` after it, will not be able to be read from the ring buffer until this item is actually sent into the ring buffer.

Note Only applicable for no-split ring buffers now, the actual size of memory that the item will occupy will be rounded up to the nearest 32-bit aligned size. This is done to ensure all items are always stored in 32-bit aligned fashion.

Return

- pdTRUE if succeeded
- pdFALSE on time-out or when the data is larger than the maximum permissible size of the buffer

Parameters

- `xRingbuffer`: Ring buffer to allocate the memory
- `ppvItem`: Double pointer to memory acquired (set to NULL if no memory were retrieved)
- `xItemSize`: Size of item to acquire.
- `xTicksToWait`: Ticks to wait for room in the ring buffer.

BaseType_t **xRingbufferSendComplete**(*RingbufHandle_t xRingbuffer*, void **pvItem*)

Actually send an item into the ring buffer allocated before by `xRingbufferSendAcquire`.

Note Only applicable for no-split ring buffers. Only call for items allocated by `xRingbufferSendAcquire`.

Return

- pdTRUE if succeeded
- pdFALSE if fail for some reason.

Parameters

- `xRingbuffer`: Ring buffer to insert the item into
- `pvItem`: Pointer to item in allocated memory to insert.

void ***xRingbufferReceive**(*RingbufHandle_t xRingbuffer*, size_t **pxItemSize*, TickType_t *xTicksToWait*)

Retrieve an item from the ring buffer.

Attempt to retrieve an item from the ring buffer. This function will block until an item is available or until it times out.

Note A call to vRingbufferReturnItem() is required after this to free the item retrieved.

Return

- Pointer to the retrieved item on success; *pxItemSize filled with the length of the item.
- NULL on timeout, *pxItemSize is untouched in that case.

Parameters

- `xRingbuffer`: Ring buffer to retrieve the item from
- `pxItemSize`: Pointer to a variable to which the size of the retrieved item will be written.
- `xTicksToWait`: Ticks to wait for items in the ring buffer.

void *`xRingbufferReceiveFromISR`(*RingbufHandle_t xRingbuffer*, size_t **pxItemSize*)

Retrieve an item from the ring buffer in an ISR.

Attempt to retrieve an item from the ring buffer. This function returns immediately if there are no items available for retrieval

Note A call to vRingbufferReturnItemFromISR() is required after this to free the item retrieved.

Note Byte buffers do not allow multiple retrievals before returning an item

Return

- Pointer to the retrieved item on success; *pxItemSize filled with the length of the item.
- NULL when the ring buffer is empty, *pxItemSize is untouched in that case.

Parameters

- `xRingbuffer`: Ring buffer to retrieve the item from
- `pxItemSize`: Pointer to a variable to which the size of the retrieved item will be written.

BaseType_t `xRingbufferReceiveSplit`(*RingbufHandle_t xRingbuffer*, void ****ppvHeadItem*, void ****ppvTailItem*, size_t **pxHeadItemSize*, size_t **pxTailItemSize*, TickType_t *xTicksToWait*)

Retrieve a split item from an allow-split ring buffer.

Attempt to retrieve a split item from an allow-split ring buffer. If the item is not split, only a single item is retried. If the item is split, both parts will be retrieved. This function will block until an item is available or until it times out.

Note Call(s) to vRingbufferReturnItem() is required after this to free up the item(s) retrieved.

Note This function should only be called on allow-split buffers

Return

- pdTRUE if an item (split or unsplit) was retrieved
- pdFALSE when no item was retrieved

Parameters

- `xRingbuffer`: Ring buffer to retrieve the item from
- `ppvHeadItem`: Double pointer to first part (set to NULL if no items were retrieved)
- `ppvTailItem`: Double pointer to second part (set to NULL if item is not split)
- `pxHeadItemSize`: Pointer to size of first part (unmodified if no items were retrieved)
- `pxTailItemSize`: Pointer to size of second part (unmodified if item is not split)
- `xTicksToWait`: Ticks to wait for items in the ring buffer.

BaseType_t **xRingbufferReceiveSplitFromISR**(*RingbufHandle_t xRingbuffer*, void **ppvHeadItem*, void **ppvTailItem*, size_t **pxHeadItemSize*, size_t **pxTailItemSize*)

Retrieve a split item from an allow-split ring buffer in an ISR.

Attempt to retrieve a split item from an allow-split ring buffer. If the item is not split, only a single item is retried. If the item is split, both parts will be retrieved. This function returns immediately if there are no items available for retrieval

Note Calls to vRingbufferReturnItemFromISR() is required after this to free up the item(s) retrieved.

Note This function should only be called on allow-split buffers

Return

- pdTRUE if an item (split or unsplit) was retrieved
- pdFALSE when no item was retrieved

Parameters

- `xRingbuffer`: Ring buffer to retrieve the item from
- `ppvHeadItem`: Double pointer to first part (set to NULL if no items were retrieved)
- `ppvTailItem`: Double pointer to second part (set to NULL if item is not split)
- `pxHeadItemSize`: Pointer to size of first part (unmodified if no items were retrieved)
- `pxTailItemSize`: Pointer to size of second part (unmodified if item is not split)

void ***xRingbufferReceiveUpTo**(*RingbufHandle_t xRingbuffer*, size_t **pxItemSize*, TickType_t *xTicksToWait*, size_t *xMaxSize*)

Retrieve bytes from a byte buffer, specifying the maximum amount of bytes to retrieve.

Attempt to retrieve data from a byte buffer whilst specifying a maximum number of bytes to retrieve. This function will block until there is data available for retrieval or until it times out.

Note A call to vRingbufferReturnItem() is required after this to free up the data retrieved.

Note This function should only be called on byte buffers

Note Byte buffers do not allow multiple retrievals before returning an item

Return

- Pointer to the retrieved item on success; *pxItemSize filled with the length of the item.
- NULL on timeout, *pxItemSize is untouched in that case.

Parameters

- `xRingbuffer`: Ring buffer to retrieve the item from
- `pxItemSize`: Pointer to a variable to which the size of the retrieved item will be written.
- `xTicksToWait`: Ticks to wait for items in the ring buffer.
- `xMaxSize`: Maximum number of bytes to return.

void *`xRingbufferReceiveUpToFromISR`(*RingbufHandle_t xRingbuffer*, size_t *pxItemSize*, size_t *xMaxSize*)

Retrieve bytes from a byte buffer, specifying the maximum amount of bytes to retrieve. Call this from an ISR.

Attempt to retrieve bytes from a byte buffer whilst specifying a maximum number of bytes to retrieve. This function will return immediately if there is no data available for retrieval.

Note A call to vRingbufferReturnItemFromISR() is required after this to free up the data received.

Note This function should only be called on byte buffers

Note Byte buffers do not allow multiple retrievals before returning an item

Return

- Pointer to the retrieved item on success; *pxItemSize filled with the length of the item.
- NULL when the ring buffer is empty, *pxItemSize is untouched in that case.

Parameters

- `xRingbuffer`: Ring buffer to retrieve the item from
- `pxItemSize`: Pointer to a variable to which the size of the retrieved item will be written.
- `xMaxSize`: Maximum number of bytes to return.

void `vRingbufferReturnItem`(*RingbufHandle_t xRingbuffer*, void *pvItem)

Return a previously-retrieved item to the ring buffer.

Note If a split item is retrieved, both parts should be returned by calling this function twice

Parameters

- `xRingbuffer`: Ring buffer the item was retrieved from
- `pvItem`: Item that was received earlier

void **vRingbufferReturnItemFromISR**(*RingbufHandle_t xRingbuffer*, void **pvItem*, BaseType_t **pxHigherPriorityTaskWoken*)

Return a previously-retrieved item to the ring buffer from an ISR.

Note If a split item is retrieved, both parts should be returned by calling this function twice

Parameters

- `xRingbuffer`: Ring buffer the item was retrieved from
- `pvItem`: Item that was received earlier
- `pxHigherPriorityTaskWoken`: Value pointed to will be set to pdTRUE if the function woke up a higher priority task.

void **vRingbufferDelete**(*RingbufHandle_t xRingbuffer*)

Delete a ring buffer.

Note This function will not deallocate any memory if the ring buffer was created using xRingbufferCreateStatic(). Deallocation must be done manually be the user.

Parameters

- `xRingbuffer`: Ring buffer to delete

size_t **xRingbufferGetMaxItemSize**(*RingbufHandle_t xRingbuffer*)

Get maximum size of an item that can be placed in the ring buffer.

This function returns the maximum size an item can have if it was placed in an empty ring buffer.

Note The max item size for a no-split buffer is limited to ((buffer_size/2)-header_size). This limit is imposed so that an item of max item size can always be sent to the an empty no-split buffer regardless of the internal positions of the buffer' s read/write/free pointers.

Return Maximum size, in bytes, of an item that can be placed in a ring buffer.

Parameters

- `xRingbuffer`: Ring buffer to query

size_t **xRingbufferGetCurFreeSize**(*RingbufHandle_t xRingbuffer*)

Get current free size available for an item/data in the buffer.

This gives the real time free space available for an item/data in the ring buffer. This represents the maximum size an item/data can have if it was currently sent to the ring buffer.

Warning This API is not thread safe. So, if multiple threads are accessing the same ring buffer, it is the application's responsibility to ensure atomic access to this API and the subsequent Send

Note An empty no-split buffer has a max current free size for an item that is limited to ((buffer_size/2)-header_size). See API reference for xRingbufferGetMaxItemSize().

Return Current free size, in bytes, available for an entry

Parameters

- `xRingbuffer`: Ring buffer to query

BaseType_t **xRingbufferAddToQueueSetRead**(*RingbufHandle_t xRingbuffer*, *QueueSetHandle_t xQueueSet*)

Add the ring buffer's read semaphore to a queue set.

The ring buffer's read semaphore indicates that data has been written to the ring buffer. This function adds the ring buffer's read semaphore to a queue set.

Return

- pdTRUE on success, pdFALSE otherwise

Parameters

- `xRingbuffer`: Ring buffer to add to the queue set
- `xQueueSet`: Queue set to add the ring buffer's read semaphore to

BaseType_t **xRingbufferCanRead**(*RingbufHandle_t xRingbuffer*, *QueueSetMemberHandle_t xMember*)

Check if the selected queue set member is the ring buffer's read semaphore.

This API checks if queue set member returned from xQueueSelectFromSet() is the read semaphore of this ring buffer. If so, this indicates the ring buffer has items waiting to be retrieved.

Return

- pdTRUE when semaphore belongs to ring buffer
- pdFALSE otherwise.

Parameters

- `xRingbuffer`: Ring buffer which should be checked
- `xMember`: Member returned from xQueueSelectFromSet

BaseType_t **xRingbufferRemoveFromQueueSetRead**(*RingbufHandle_t xRingbuffer*, *QueueSetHandle_t xQueueSet*)

Remove the ring buffer's read semaphore from a queue set.

This specifically removes a ring buffer's read semaphore from a queue set. The read semaphore is used to indicate when data has been written to the ring buffer

Return

- pdTRUE on success
- pdFALSE otherwise

Parameters

- `xRingbuffer`: Ring buffer to remove from the queue set
- `xQueueSet`: Queue set to remove the ring buffer's read semaphore from

void **vRingbufferGetInfo**(*RingbufHandle_t xRingbuffer*, UBaseType_t **uxFree*, UBaseType_t **uxRead*, UBaseType_t **uxWrite*, UBaseType_t **uxAcquire*, UBaseType_t **uxItemsWaiting*)

Get information about ring buffer status.

Get information of the a ring buffer's current status such as free/read/write pointer positions, and number of items waiting to be retrieved. Arguments can be set to NULL if they are not required.

Parameters

- `xRingbuffer`: Ring buffer to remove from the queue set
- `uxFree`: Pointer use to store free pointer position
- `uxRead`: Pointer use to store read pointer position
- `uxWrite`: Pointer use to store write pointer position
- `uxAcquire`: Pointer use to store acquire pointer position
- `uxItemsWaiting`: Pointer use to store number of items (bytes for byte buffer) waiting to be retrieved

void **xRingbufferPrintInfo**(*RingbufHandle_t xRingbuffer*)

Debugging function to print the internal pointers in the ring buffer.

Parameters

- `xRingbuffer`: Ring buffer to show

Structures

struct xSTATIC_RINGBUFFER

Struct that is equivalent in size to the ring buffer's data structure.

The contents of this struct are not meant to be used directly. This structure is meant to be used when creating a statically allocated ring buffer where this struct is of the exact size required to store a ring buffer's control data structure.

Note The CONFIG_FREERTOS_SUPPORT_STATIC_ALLOCATION option must be enabled for this structure to be available.

Type Definitions

typedef void ***RingbufHandle_t**

Type by which ring buffers are referenced. For example, a call to xRingbufferCreate() returns a RingbufHandle_t variable that can then be used as a parameter to xRingbufferSend(), xRingbufferReceive(), etc.

typedef struct *xSTATIC_RINGBUFFER* **StaticRingbuffer_t**

Struct that is equivalent in size to the ring buffer's data structure.

The contents of this struct are not meant to be used directly. This structure is meant to be used when creating a statically allocated ring buffer where this struct is of the exact size required to store a ring buffer's control data structure.

Note The CONFIG_FREERTOS_SUPPORT_STATIC_ALLOCATION option must be enabled for this structure to be available.

Enumerations

enum RingbufferType_t

Values:

RINGBUF_TYPE_NOSPLIT = 0

No-split buffers will only store an item in contiguous memory and will never split an item. Each item requires an 8 byte overhead for a header and will always internally occupy a 32-bit aligned size of space.

RINGBUF_TYPE_ALLOWSPLIT

Allow-split buffers will split an item into two parts if necessary in order to store it. Each item requires an 8 byte overhead for a header, splitting incurs an extra header. Each item will always internally occupy a 32-bit aligned size of space.

`RINGBUF_TYPE_BYTEBUF`
: Byte buffers store data as a sequence of bytes and do not maintain separate items, therefore byte buffers have no overhead. All data is stored as a sequence of byte and any number of bytes can be sent or retrieved each time.

`RINGBUF_TYPE_MAX`

Hooks

FreeRTOS consists of Idle Hooks and Tick Hooks which allow for application specific functionality to be added to the Idle Task and Tick Interrupt. ESP-IDF provides its own Idle and Tick Hook API in addition to the hooks provided by Vanilla FreeRTOS. ESP-IDF hooks have the added benefit of being run time configurable and asymmetrical.

Vanilla FreeRTOS Hooks

Idle and Tick Hooks in vanilla FreeRTOS are implemented by the user defining the functions `vApplicationIdleHook()` and `vApplicationTickHook()` respectively somewhere in the application. Vanilla FreeRTOS will run the user defined Idle Hook and Tick Hook on every iteration of the Idle Task and Tick Interrupt respectively.

Vanilla FreeRTOS hooks are referred to as **Legacy Hooks** in ESP-IDF FreeRTOS. To enable legacy hooks, *CONFIG_FREERTOS_LEGACY_HOOKS* should be enabled in *project configuration menu*.

Due to vanilla FreeRTOS being designed for single core, `vApplicationIdleHook()` and `vApplicationTickHook()` can only be defined once. However, the ESP32 is dual core in nature, therefore same Idle Hook and Tick Hook are used for both cores (in other words, the hooks are symmetrical for both cores).

ESP-IDF Idle and Tick Hooks

Due to the the dual core nature of the ESP32, it may be necessary for some applications to have separate hooks for each core. Furthermore, it may be necessary for the Idle Tasks or Tick Interrupts to execute multiple hooks that are configurable at run time. Therefore the ESP-IDF provides it' s own hooks API in addition to the legacy hooks provided by Vanilla FreeRTOS.

The ESP-IDF tick/idle hooks are registered at run time, and each tick/idle hook must be registered to a specific CPU. When the idle task runs/tick Interrupt occurs on a particular CPU, the CPU will run each of its registered idle/tick hooks in turn.

Hooks API Reference

Header File

- esp_common/include/esp_freertos_hooks.h

Functions

esp_err_t **esp_register_freertos_idle_hook_for_cpu**(*esp_freertos_idle_cb_t new_idle_cb*, UBaseType_t *cpuid*)

Register a callback to be called from the specified core's idle hook. The callback should return true if it should be called by the idle hook once per interrupt (or FreeRTOS tick), and return false if it should be called repeatedly as fast as possible by the idle hook.

Warning Idle callbacks MUST NOT, UNDER ANY CIRCUMSTANCES, CALL A FUNCTION THAT MIGHT BLOCK.

Return

- ESP_OK: Callback registered to the specified core's idle hook
- ESP_ERR_NO_MEM: No more space on the specified core's idle hook to register callback
- ESP_ERR_INVALID_ARG: cpuid is invalid

Parameters

- `new_idle_cb`: Callback to be called
- `cpuid`: id of the core

esp_err_t **esp_register_freertos_idle_hook**(*esp_freertos_idle_cb_t new_idle_cb*)

Register a callback to the idle hook of the core that calls this function. The callback should return true if it should be called by the idle hook once per interrupt (or FreeRTOS tick), and return false if it should be called repeatedly as fast as possible by the idle hook.

Warning Idle callbacks MUST NOT, UNDER ANY CIRCUMSTANCES, CALL A FUNCTION THAT MIGHT BLOCK.

Return

- ESP_OK: Callback registered to the calling core's idle hook
- ESP_ERR_NO_MEM: No more space on the calling core's idle hook to register callback

Parameters

- `new_idle_cb`: Callback to be called

esp_err_t **esp_register_freertos_tick_hook_for_cpu**(*esp_freertos_tick_cb_t new_tick_cb*, UBaseType_t *cpuid*)

Register a callback to be called from the specified core's tick hook.

Return

- ESP_OK: Callback registered to specified core’ s tick hook
- ESP_ERR_NO_MEM: No more space on the specified core’ s tick hook to register the callback
- ESP_ERR_INVALID_ARG: cpuid is invalid

Parameters

- `new_tick_cb`: Callback to be called
- `cpuid`: id of the core

esp_err_t **esp_register_freertos_tick_hook**(*esp_freertos_tick_cb_t new_tick_cb*)

Register a callback to be called from the calling core’ s tick hook.

Return

- ESP_OK: Callback registered to the calling core’ s tick hook
- ESP_ERR_NO_MEM: No more space on the calling core’s tick hook to register the callback

Parameters

- `new_tick_cb`: Callback to be called

void **esp_deregister_freertos_idle_hook_for_cpu**(*esp_freertos_idle_cb_t old_idle_cb*, UBaseType_t *cpuid*)

Unregister an idle callback from the idle hook of the specified core.

Parameters

- `old_idle_cb`: Callback to be unregistered
- `cpuid`: id of the core

void **esp_deregister_freertos_idle_hook**(*esp_freertos_idle_cb_t old_idle_cb*)

Unregister an idle callback. If the idle callback is registered to the idle hooks of both cores, the idle hook will be unregistered from both cores.

Parameters

- `old_idle_cb`: Callback to be unregistered

void **esp_deregister_freertos_tick_hook_for_cpu**(*esp_freertos_tick_cb_t old_tick_cb*, UBaseType_t *cpuid*)

Unregister a tick callback from the tick hook of the specified core.

Parameters

- `old_tick_cb`: Callback to be unregistered

- `cpuid`: id of the core

void **esp_deregister_freertos_tick_hook**(*esp_freertos_tick_cb_t old_tick_cb*)

Unregister a tick callback. If the tick callback is registered to the tick hooks of both cores, the tick hook will be unregistered from both cores.

Parameters

- `old_tick_cb`: Callback to be unregistered

Type Definitions

typedef bool (***esp_freertos_idle_cb_t**)(void)

typedef void (***esp_freertos_tick_cb_t**)(void)

2.7.10 Heap Memory Allocation

Stack and Heap

ESP-IDF applications use the common computer architecture patterns of *stack* (dynamic memory allocated by program control flow) and *heap* (dynamic memory allocated by function calls), as well as statically allocated memory (allocated at compile time).

Because ESP-IDF is a multi-threaded RTOS environment, each RTOS task has its own stack. By default, each of these stacks is allocated from the heap when the task is created. (See *xTaskCreateStatic()* for the alternative where stacks are statically allocated.)

Because ESP32 uses multiple types of RAM, it also contains multiple heaps with different capabilities. A capabilities-based memory allocator allows apps to make heap allocations for different purposes.

For most purposes, the standard libc `malloc()` and `free()` functions can be used for heap allocation without any special consideration.

However, in order to fully make use of all of the memory types and their characteristics, ESP-IDF also has a capabilities-based heap memory allocator. If you want to have memory with certain properties (for example, *DMA-Capable Memory* or executable-memory), you can create an OR-mask of the required capabilities and pass that to *heap_caps_malloc()*.

Memory Capabilities

The ESP32 contains multiple types of RAM:

- DRAM (Data RAM) is memory used to hold data. This is the most common kind of memory accessed as heap.

- IRAM (Instruction RAM) usually holds executable data only. If accessed as generic memory, all accesses must be *32-bit aligned*.
- D/IRAM is RAM which can be used as either Instruction or Data RAM.

For more details on these internal memory types, see *应用程序的内存布局*.

It's also possible to connect external SPI RAM to the ESP32 - *external RAM* can be integrated into the ESP32's memory map using the flash cache, and accessed similarly to DRAM.

DRAM uses capability `MALLOC_CAP_8BIT` (accessible in single byte reads and writes). When calling `malloc()`, the ESP-IDF `malloc()` implementation internally calls `heap_caps_malloc(size, MALLOC_CAP_8BIT)` in order to allocate DRAM that is byte-addressable. To test the free DRAM heap size at runtime, call cpp:func:*heap_caps_get_free_size(MALLOC_CAP_8BIT)*.

Because malloc uses the capabilities-based allocation system, memory allocated using *`heap_caps_malloc()`* can be freed by calling the standard `free()` function.

Available Heap

DRAM

At startup, the DRAM heap contains all data memory which is not statically allocated by the app. Reducing statically allocated buffers will increase the amount of available free heap.

To find the amount of statically allocated memory, use the *idf.py size* command.

注解: Due to a technical limitation, the maximum statically allocated DRAM usage is 160KB. The remaining 160KB (for a total of 320KB of DRAM) can only be allocated at runtime as heap.

注解: At runtime, the available heap DRAM may be less than calculated at compile time, because at startup some memory is allocated from the heap before the FreeRTOS scheduler is started (including memory for the stacks of initial FreeRTOS tasks).

IRAM

At startup, the IRAM heap contains all instruction memory which is not used by the app executable code.

The *idf.py size* command can be used to find the amount of IRAM used by the app.

D/IRAM

Some memory in the ESP32 is available as either DRAM or IRAM. If memory is allocated from a D/IRAM region, the free heap size for both types of memory will decrease.

Heap Sizes

At startup, all ESP-IDF apps log a summary of all heap addresses (and sizes) at level Info:

```
I (252) heap_init: Initializing. RAM available for dynamic allocation:
I (259) heap_init: At 3FFAE6E0 len 00001920 (6 KiB): DRAM
I (265) heap_init: At 3FFB2EC8 len 0002D138 (180 KiB): DRAM
I (272) heap_init: At 3FFE0440 len 00003AE0 (14 KiB): D/IRAM
I (278) heap_init: At 3FFE4350 len 0001BCB0 (111 KiB): D/IRAM
I (284) heap_init: At 4008944C len 00016BB4 (90 KiB): IRAM
```

Finding available heap

See *Heap Information*.

Special Capabilities

DMA-Capable Memory

Use the `MALLOC_CAP_DMA` flag to allocate memory which is suitable for use with hardware DMA engines (for example SPI and I2S). This capability flag excludes any external PSRAM.

32-Bit Accessible Memory

If a certain memory structure is only addressed in 32-bit units, for example an array of ints or pointers, it can be useful to allocate it with the `MALLOC_CAP_32BIT` flag. This also allows the allocator to give out IRAM memory; something which it can't do for a normal malloc() call. This can help to use all the available memory in the ESP32.

Memory allocated with `MALLOC_CAP_32BIT` can *only* be accessed via 32-bit reads and writes, any other type of access will generate a fatal LoadStoreError exception.

External SPI Memory

When *external RAM* is enabled, external SPI RAM under 4MiB in size can be allocated using standard `malloc` calls, or via `heap_caps_malloc(MALLOC_CAP_SPIRAM)`, depending on configuration. See 配置片外

RAM for more details.

To use the region above the 4MiB limit, you can use the *himem API*.

API Reference - Heap Allocation

Header File

- heap/include/esp_heap_caps.h

Functions

void *`heap_caps_malloc`(size_t *size*, uint32_t *caps*)

Allocate a chunk of memory which has the given capabilities.

Equivalent semantics to libc malloc(), for capability-aware memory.

In IDF, `malloc(p)` is equivalent to `heap_caps_malloc(p, MALLOC_CAP_8BIT)`.

Return A pointer to the memory allocated on success, NULL on failure

Parameters

- `size`: Size, in bytes, of the amount of memory to allocate
- `caps`: Bitwise OR of MALLOC_CAP_* flags indicating the type of memory to be returned

void `heap_caps_free`(void *_ptr_)

Free memory previously allocated via heap_caps_malloc() or heap_caps_realloc().

Equivalent semantics to libc free(), for capability-aware memory.

In IDF, `free(p)` is equivalent to `heap_caps_free(p)`.

Parameters

- `ptr`: Pointer to memory previously returned from heap_caps_malloc() or heap_caps_realloc(). Can be NULL.

void *`heap_caps_realloc`(void *_ptr_, size_t *size*, int *caps*)

Reallocate memory previously allocated via heap_caps_malloc() or heap_caps_realloc().

Equivalent semantics to libc realloc(), for capability-aware memory.

In IDF, `realloc(p, s)` is equivalent to `heap_caps_realloc(p, s, MALLOC_CAP_8BIT)`.

'caps' parameter can be different to the capabilities that any original 'ptr' was allocated with. In this way, realloc can be used to "move" a buffer if necessary to ensure it meets a new set of capabilities.

Return Pointer to a new buffer of size 'size' with capabilities 'caps', or NULL if allocation failed.

Parameters

- `ptr`: Pointer to previously allocated memory, or NULL for a new allocation.
- `size`: Size of the new buffer requested, or 0 to free the buffer.
- `caps`: Bitwise OR of MALLOC_CAP_* flags indicating the type of memory desired for the new allocation.

void *`heap_caps_aligned_alloc`(size_t *alignment*, size_t *size*, int *caps*)

Allocate a aligned chunk of memory which has the given capabilities.

Equivalent semantics to libc aligned_alloc(), for capability-aware memory.

Return A pointer to the memory allocated on success, NULL on failure

Note Any memory allocated with heaps_caps_aligned_alloc() MUST be freed with heap_caps_aligned_free() and CANNOT be passed to free()

Parameters

- `alignment`: How the pointer received needs to be aligned must be a power of two
- `size`: Size, in bytes, of the amount of memory to allocate
- `caps`: Bitwise OR of MALLOC_CAP_* flags indicating the type of memory to be returned

void *`heap_caps_aligned_calloc`(size_t *alignment*, size_t *n*, size_t *size*, uint32_t *caps*)

Allocate a aligned chunk of memory which has the given capabilities. The initialized value in the memory is set to zero.

Return A pointer to the memory allocated on success, NULL on failure

Note Any memory allocated with heap_caps_aligned_calloc() MUST be freed with heap_caps_aligned_free() and CANNOT be passed to free()

Parameters

- `alignment`: How the pointer received needs to be aligned must be a power of two
- `n`: Number of continuing chunks of memory to allocate
- `size`: Size, in bytes, of a chunk of memory to allocate
- `caps`: Bitwise OR of MALLOC_CAP_* flags indicating the type of memory to be returned

void `heap_caps_aligned_free`(void **ptr*)

Used to deallocate memory previously allocated with heap_caps_aligned_alloc.

Note This function is aimed to deallocate only memory allocated with heap_caps_aligned_alloc, memory allocated with heap_caps_malloc MUST not be passed to this function

Parameters

- `ptr`: Pointer to the memory allocated

void *`heap_caps_calloc`(size_t *n*, size_t *size*, uint32_t *caps*)

Allocate a chunk of memory which has the given capabilities. The initialized value in the memory is set to zero.

Equivalent semantics to libc calloc(), for capability-aware memory.

In IDF, `calloc(p)` is equivalent to `heap_caps_calloc(p, MALLOC_CAP_8BIT)`.

Return A pointer to the memory allocated on success, NULL on failure

Parameters

- `n`: Number of continuing chunks of memory to allocate
- `size`: Size, in bytes, of a chunk of memory to allocate
- `caps`: Bitwise OR of MALLOC_CAP_* flags indicating the type of memory to be returned

size_t `heap_caps_get_total_size`(uint32_t *caps*)

Get the total size of all the regions that have the given capabilities.

This function takes all regions capable of having the given capabilities allocated in them and adds up the total space they have.

Return total size in bytes

Parameters

- `caps`: Bitwise OR of MALLOC_CAP_* flags indicating the type of memory

size_t `heap_caps_get_free_size`(uint32_t *caps*)

Get the total free size of all the regions that have the given capabilities.

This function takes all regions capable of having the given capabilities allocated in them and adds up the free space they have.

Note that because of heap fragmentation it is probably not possible to allocate a single block of memory of this size. Use heap_caps_get_largest_free_block() for this purpose.

Return Amount of free bytes in the regions

Parameters

- `caps`: Bitwise OR of MALLOC_CAP_* flags indicating the type of memory

size_t `heap_caps_get_minimum_free_size`(uint32_t *caps*)

Get the total minimum free memory of all regions with the given capabilities.

This adds all the low water marks of the regions capable of delivering the memory with the given capabilities.

Note the result may be less than the global all-time minimum available heap of this kind, as "low water marks" are tracked per-region. Individual regions' heaps may have reached their "low water marks" at different points in time. However this result still gives a "worst case" indication for all-time minimum free heap.

Return Amount of free bytes in the regions

Parameters

- `caps`: Bitwise OR of MALLOC_CAP_* flags indicating the type of memory

size_t **heap_caps_get_largest_free_block**(uint32_t *caps*)

Get the largest free block of memory able to be allocated with the given capabilities.

Returns the largest value of `s` for which `heap_caps_malloc(s, caps)` will succeed.

Return Size of largest free block in bytes.

Parameters

- `caps`: Bitwise OR of MALLOC_CAP_* flags indicating the type of memory

void **heap_caps_get_info**(*multi_heap_info_t* **info*, uint32_t *caps*)

Get heap info for all regions with the given capabilities.

Calls multi_heap_info() on all heaps which share the given capabilities. The information returned is an aggregate across all matching heaps. The meanings of fields are the same as defined for *multi_heap_info_t*, except that `minimum_free_bytes` has the same caveats described in heap_caps_get_minimum_free_size().

Parameters

- `info`: Pointer to a structure which will be filled with relevant heap metadata.
- `caps`: Bitwise OR of MALLOC_CAP_* flags indicating the type of memory

void **heap_caps_print_heap_info**(uint32_t *caps*)

Print a summary of all memory with the given capabilities.

Calls multi_heap_info on all heaps which share the given capabilities, and prints a two-line summary for each, then a total summary.

Parameters

- `caps`: Bitwise OR of MALLOC_CAP_* flags indicating the type of memory

bool **heap_caps_check_integrity_all**(bool *print_errors*)

Check integrity of all heap memory in the system.

Calls multi_heap_check on all heaps. Optionally print errors if heaps are corrupt.

Calling this function is equivalent to calling heap_caps_check_integrity with the caps argument set to MALLOC_CAP_INVALID.

Return True if all heaps are valid, False if at least one heap is corrupt.

Parameters

- `print_errors`: Print specific errors if heap corruption is found.

bool **heap_caps_check_integrity**(uint32_t *caps*, bool *print_errors*)

Check integrity of all heaps with the given capabilities.

Calls multi_heap_check on all heaps which share the given capabilities. Optionally print errors if the heaps are corrupt.

See also heap_caps_check_integrity_all to check all heap memory in the system and heap_caps_check_integrity_addr to check memory around a single address.

Return True if all heaps are valid, False if at least one heap is corrupt.

Parameters

- `caps`: Bitwise OR of MALLOC_CAP_* flags indicating the type of memory
- `print_errors`: Print specific errors if heap corruption is found.

bool **heap_caps_check_integrity_addr**(intptr_t *addr*, bool *print_errors*)

Check integrity of heap memory around a given address.

This function can be used to check the integrity of a single region of heap memory, which contains the given address.

This can be useful if debugging heap integrity for corruption at a known address, as it has a lower overhead than checking all heap regions. Note that if the corrupt address moves around between runs (due to timing or other factors) then this approach won't work and you should call heap_caps_check_integrity or heap_caps_check_integrity_all instead.

Note The entire heap region around the address is checked, not only the adjacent heap blocks.

Return True if the heap containing the specified address is valid, False if at least one heap is corrupt or the address doesn't belong to a heap region.

Parameters

- `addr`: Address in memory. Check for corruption in region containing this address.
- `print_errors`: Print specific errors if heap corruption is found.

void **heap_caps_malloc_extmem_enable**(size_t *limit*)

Enable malloc() in external memory and set limit below which malloc() attempts are placed in internal memory.

When external memory is in use, the allocation strategy is to initially try to satisfy smaller allocation requests with internal memory and larger requests with external memory. This sets the limit between the two, as well as generally enabling allocation in external memory.

Parameters

- `limit`: Limit, in bytes.

void *`heap_caps_malloc_prefer`(size_t *size*, size_t *num*, ...)

Allocate a chunk of memory as preference in decreasing order.

Attention The variable parameters are bitwise OR of MALLOC_CAP_* flags indicating the type of memory. This API prefers to allocate memory with the first parameter. If failed, allocate memory with the next parameter. It will try in this order until allocating a chunk of memory successfully or fail to allocate memories with any of the parameters.

Return A pointer to the memory allocated on success, NULL on failure

Parameters

- `size`: Size, in bytes, of the amount of memory to allocate
- `num`: Number of variable paramters

void *`heap_caps_realloc_prefer`(void **ptr*, size_t *size*, size_t *num*, ...)

Allocate a chunk of memory as preference in decreasing order.

Return Pointer to a new buffer of size ‘size’, or NULL if allocation failed.

Parameters

- `ptr`: Pointer to previously allocated memory, or NULL for a new allocation.
- `size`: Size of the new buffer requested, or 0 to free the buffer.
- `num`: Number of variable paramters

void *`heap_caps_calloc_prefer`(size_t *n*, size_t *size*, size_t *num*, ...)

Allocate a chunk of memory as preference in decreasing order.

Return A pointer to the memory allocated on success, NULL on failure

Parameters

- `n`: Number of continuing chunks of memory to allocate
- `size`: Size, in bytes, of a chunk of memory to allocate
- `num`: Number of variable paramters

void **heap_caps_dump**(uint32_t *caps*)

Dump the full structure of all heaps with matching capabilities.

Prints a large amount of output to serial (because of locking limitations, the output bypasses stdout/stderr). For each (variable sized) block in each matching heap, the following output is printed on a single line:

- Block address (the data buffer returned by malloc is 4 bytes after this if heap debugging is set to Basic, or 8 bytes otherwise).
- Data size (the data size may be larger than the size requested by malloc, either due to heap fragmentation or because of heap debugging level).
- Address of next block in the heap.
- If the block is free, the address of the next free block is also printed.

Parameters

- `caps`: Bitwise OR of MALLOC_CAP_* flags indicating the type of memory

void **heap_caps_dump_all**(void)

Dump the full structure of all heaps.

Covers all registered heaps. Prints a large amount of output to serial.

Output is the same as for heap_caps_dump.

size_t **heap_caps_get_allocated_size**(void **ptr*)

Return the size that a particular pointer was allocated with.

Note The app will crash with an assertion failure if the pointer is not valid.

Return Size of the memory allocated at this block.

Parameters

- `ptr`: Pointer to currently allocated heap memory. Must be a pointer value previously returned by heap_caps_malloc,malloc,calloc, etc. and not yet freed.

Macros

`MALLOC_CAP_EXEC`

Flags to indicate the capabilities of the various memory systems.

Memory must be able to run executable code

`MALLOC_CAP_32BIT`

Memory must allow for aligned 32-bit data accesses.

`MALLOC_CAP_8BIT`
: Memory must allow for 8/16/…-bit data accesses.

`MALLOC_CAP_DMA`
: Memory must be able to accessed by DMA.

`MALLOC_CAP_PID2`
: Memory must be mapped to PID2 memory space (PIDs are not currently used)

`MALLOC_CAP_PID3`
: Memory must be mapped to PID3 memory space (PIDs are not currently used)

`MALLOC_CAP_PID4`
: Memory must be mapped to PID4 memory space (PIDs are not currently used)

`MALLOC_CAP_PID5`
: Memory must be mapped to PID5 memory space (PIDs are not currently used)

`MALLOC_CAP_PID6`
: Memory must be mapped to PID6 memory space (PIDs are not currently used)

`MALLOC_CAP_PID7`
: Memory must be mapped to PID7 memory space (PIDs are not currently used)

`MALLOC_CAP_SPIRAM`
: Memory must be in SPI RAM.

`MALLOC_CAP_INTERNAL`
: Memory must be internal; specifically it should not disappear when flash/spiram cache is switched off.

`MALLOC_CAP_DEFAULT`
: Memory can be returned in a non-capability-specific memory allocation (e.g. malloc(), calloc()) call.

`MALLOC_CAP_INVALID`
: Memory can’ t be used / list end marker.

Thread Safety

Heap functions are thread safe, meaning they can be called from different tasks simultaneously without any limitations.

It is technically possible to call `malloc`, `free`, and related functions from interrupt handler (ISR) context. However this is not recommended, as heap function calls may delay other interrupts. It is strongly recommended to refactor applications so that any buffers used by an ISR are pre-allocated outside of the ISR. Support for calling heap functions from ISRs may be removed in a future update.

Heap Tracing & Debugging

The following features are documented on the *Heap Memory Debugging* page:

- *Heap Information* (free space, etc.)
- *Heap Corruption Detection*
- *Heap Tracing* (memory leak detection, monitoring, etc.)

API Reference - Initialisation

Header File

- heap/include/esp_heap_caps_init.h

Functions

void **heap_caps_init**(void)

Initialize the capability-aware heap allocator.

This is called once in the IDF startup code. Do not call it at other times.

void **heap_caps_enable_nonos_stack_heaps**(void)

Enable heap(s) in memory regions where the startup stacks are located.

On startup, the pro/app CPUs have a certain memory region they use as stack, so we cannot do allocations in the regions these stack frames are. When FreeRTOS is completely started, they do not use that memory anymore and heap(s) there can be enabled.

esp_err_t **heap_caps_add_region**(intptr_t *start*, intptr_t *end*)

Add a region of memory to the collection of heaps at runtime.

Most memory regions are defined in soc_memory_layout.c for the SoC, and are registered via heap_caps_init(). Some regions can't be used immediately and are later enabled via heap_caps_enable_nonos_stack_heaps().

Call this function to add a region of memory to the heap at some later time.

This function does not consider any of the "reserved" regions or other data in soc_memory_layout, caller needs to consider this themselves.

All memory within the region specified by start & end parameters must be otherwise unused.

The capabilities of the newly registered memory will be determined by the start address, as looked up in the regions specified in soc_memory_layout.c.

Use heap_caps_add_region_with_caps() to register a region with custom capabilities.

Return ESP_OK on success, ESP_ERR_INVALID_ARG if a parameter is invalid, ESP_ERR_NOT_FOUND if the specified start address doesn't reside in a known region, or any error returned by heap_caps_add_region_with_caps().

Parameters

- `start`: Start address of new region.
- `end`: End address of new region.

esp_err_t **heap_caps_add_region_with_caps(const** uint32_t *caps*[], intptr_t *start*, intptr_t *end*)

Add a region of memory to the collection of heaps at runtime, with custom capabilities.

Similar to heap_caps_add_region(), only custom memory capabilities are specified by the caller.

Return

- ESP_OK on success
- ESP_ERR_INVALID_ARG if a parameter is invalid
- ESP_ERR_NO_MEM if no memory to register new heap.
- ESP_ERR_INVALID_SIZE if the memory region is too small to fit a heap
- ESP_FAIL if region overlaps the start and/or end of an existing region

Parameters

- `caps`: Ordered array of capability masks for the new region, in order of priority. Must have length SOC_MEMORY_TYPE_NO_PRIOS. Does not need to remain valid after the call returns.
- `start`: Start address of new region.
- `end`: End address of new region.

Implementation Notes

Knowledge about the regions of memory in the chip comes from the "soc" component, which contains memory layout information for the chip, and the different capabilities of each region. Each region's capabilities are prioritised, so that (for example) dedicated DRAM and IRAM regions will be used for allocations ahead of the more versatile D/IRAM regions.

Each contiguous region of memory contains its own memory heap. The heaps are created using the multi_heap functionality. multi_heap allows any contiguous region of memory to be used as a heap.

The heap capabilities allocator uses knowledge of the memory regions to initialize each individual heap. Allocation functions in the heap capabilities API will find the most appropriate heap for the allocation (based on desired capabilities, available space, and preferences for each region's use) and then calling *multi_heap_malloc()* or `multi_heap_calloc()` for the heap situated in that particular region.

Calling `free()` involves finding the particular heap corresponding to the freed address, and then calling *multi_heap_free()* on that particular multi_heap instance.

API Reference - Multi Heap API

(Note: The multi heap API is used internally by the heap capabilities allocator. Most IDF programs will never need to call this API directly.)

Header File

- heap/include/multi_heap.h

Functions

void *`multi_heap_aligned_alloc`(*multi_heap_handle_t* *heap*, size_t *size*, size_t *alignment*)
allocate a chunk of memory with specific alignment

Return pointer to the memory allocated, NULL on failure

Parameters

- `heap`: Handle to a registered heap.
- `size`: size in bytes of memory chunk
- `alignment`: how the memory must be aligned

void *`multi_heap_malloc`(*multi_heap_handle_t* *heap*, size_t *size*)
malloc() a buffer in a given heap

Semantics are the same as standard malloc(), only the returned buffer will be allocated in the specified heap.

Return Pointer to new memory, or NULL if allocation fails.

Parameters

- `heap`: Handle to a registered heap.
- `size`: Size of desired buffer.

void `multi_heap_aligned_free`(*multi_heap_handle_t* *heap*, void *p)
free() a buffer aligned in a given heap.

Parameters

- `heap`: Handle to a registered heap.
- `p`: NULL, or a pointer previously returned from multi_heap_aligned_alloc() for the same heap.

void **multi_heap_free**(*multi_heap_handle_t* *heap*, void **p*)

free() a buffer in a given heap.

Semantics are the same as standard free(), only the argument 'p' must be NULL or have been allocated in the specified heap.

Parameters

- `heap`: Handle to a registered heap.
- `p`: NULL, or a pointer previously returned from multi_heap_malloc() or multi_heap_realloc() for the same heap.

void ***multi_heap_realloc**(*multi_heap_handle_t* *heap*, void **p*, size_t *size*)

realloc() a buffer in a given heap.

Semantics are the same as standard realloc(), only the argument 'p' must be NULL or have been allocated in the specified heap.

Return New buffer of 'size' containing contents of 'p', or NULL if reallocation failed.

Parameters

- `heap`: Handle to a registered heap.
- `p`: NULL, or a pointer previously returned from multi_heap_malloc() or multi_heap_realloc() for the same heap.
- `size`: Desired new size for buffer.

size_t **multi_heap_get_allocated_size**(*multi_heap_handle_t* *heap*, void **p*)

Return the size that a particular pointer was allocated with.

Return Size of the memory allocated at this block. May be more than the original size argument, due to padding and minimum block sizes.

Parameters

- `heap`: Handle to a registered heap.
- `p`: Pointer, must have been previously returned from multi_heap_malloc() or multi_heap_realloc() for the same heap.

multi_heap_handle_t **multi_heap_register**(void **start*, size_t *size*)

Register a new heap for use.

This function initialises a heap at the specified address, and returns a handle for future heap operations.

There is no equivalent function for deregistering a heap - if all blocks in the heap are free, you can immediately start using the memory for other purposes.

Return Handle of a new heap ready for use, or NULL if the heap region was too small to be initialised.

Parameters

- `start`: Start address of the memory to use for a new heap.
- `size`: Size (in bytes) of the new heap.

void **multi_heap_set_lock**(*multi_heap_handle_t heap*, void **lock*)

Associate a private lock pointer with a heap.

The lock argument is supplied to the MULTI_HEAP_LOCK() and MULTI_HEAP_UNLOCK() macros, defined in multi_heap_platform.h.

The lock in question must be recursive.

When the heap is first registered, the associated lock is NULL.

Parameters

- `heap`: Handle to a registered heap.
- `lock`: Optional pointer to a locking structure to associate with this heap.

void **multi_heap_dump**(*multi_heap_handle_t heap*)

Dump heap information to stdout.

For debugging purposes, this function dumps information about every block in the heap to stdout.

Parameters

- `heap`: Handle to a registered heap.

bool **multi_heap_check**(*multi_heap_handle_t heap*, bool *print_errors*)

Check heap integrity.

Walks the heap and checks all heap data structures are valid. If any errors are detected, an error-specific message can be optionally printed to stderr. Print behaviour can be overriden at compile time by defining MULTI_CHECK_FAIL_PRINTF in multi_heap_platform.h.

Return true if heap is valid, false otherwise.

Parameters

- `heap`: Handle to a registered heap.
- `print_errors`: If true, errors will be printed to stderr.

size_t **multi_heap_free_size**(*multi_heap_handle_t heap*)

Return free heap size.

Returns the number of bytes available in the heap.

Equivalent to the total_free_bytes member returned by multi_heap_get_heap_info().

Note that the heap may be fragmented, so the actual maximum size for a single malloc() may be lower. To know this size, see the largest_free_block member returned by multi_heap_get_heap_info().

Return Number of free bytes.

Parameters

- `heap`: Handle to a registered heap.

size_t **multi_heap_minimum_free_size**(*multi_heap_handle_t heap*)

Return the lifetime minimum free heap size.

Equivalent to the minimum_free_bytes member returned by multi_heap_get_info().

Returns the lifetime "low water mark" of possible values returned from multi_free_heap_size(), for the specified heap.

Return Number of free bytes.

Parameters

- `heap`: Handle to a registered heap.

void **multi_heap_get_info**(*multi_heap_handle_t heap*, *multi_heap_info_t *info*)

Return metadata about a given heap.

Fills a *multi_heap_info_t* structure with information about the specified heap.

Parameters

- `heap`: Handle to a registered heap.
- `info`: Pointer to a structure to fill with heap metadata.

Structures

struct multi_heap_info_t

Structure to access heap metadata via multi_heap_get_info.

Public Members

size_t **total_free_bytes**

Total free bytes in the heap. Equivalent to multi_free_heap_size().

size_t **total_allocated_bytes**

Total bytes allocated to data in the heap.

size_t **largest_free_block**
: Size of largest free block in the heap. This is the largest malloc-able size.

size_t **minimum_free_bytes**
: Lifetime minimum free heap size. Equivalent to multi_minimum_free_heap_size().

size_t **allocated_blocks**
: Number of (variable size) blocks allocated in the heap.

size_t **free_blocks**
: Number of (variable size) free blocks in the heap.

size_t **total_blocks**
: Total number of (variable size) blocks in the heap.

Type Definitions

typedef struct multi_heap_info ***multi_heap_handle_t**
: Opaque handle to a registered heap.

2.7.11 Heap Memory Debugging

Overview

ESP-IDF integrates tools for requesting *heap information*, *detecting heap corruption*, and *tracing memory leaks*. These can help track down memory-related bugs.

For general information about the heap memory allocator, see the *Heap Memory Allocation* page.

Heap Information

To obtain information about the state of the heap:

- `xPortGetFreeHeapSize()` is a FreeRTOS function which returns the number of free bytes in the (data memory) heap. This is equivalent to calling `heap_caps_get_free_size(MALLOC_CAP_8BIT)`.
- *`heap_caps_get_free_size()`* can also be used to return the current free memory for different memory capabilities.
- *`heap_caps_get_largest_free_block()`* can be used to return the largest free block in the heap. This is the largest single allocation which is currently possible. Tracking this value and comparing to total free heap allows you to detect heap fragmentation.
- `xPortGetMinimumEverFreeHeapSize()` and the related *`heap_caps_get_minimum_free_size()`* can be used to track the heap "low water mark" since boot.

- *heap_caps_get_info()* returns a *multi_heap_info_t* structure which contains the information from the above functions, plus some additional heap-specific data (number of allocations, etc.).
- *heap_caps_print_heap_info()* prints a summary to stdout of the information returned by *heap_caps_get_info()*.
- *heap_caps_dump()* and *heap_caps_dump_all()* will output detailed information about the structure of each block in the heap. Note that this can be large amount of output.

Heap Corruption Detection

Heap corruption detection allows you to detect various types of heap memory errors:

- Out of bounds writes & buffer overflow.
- Writes to freed memory.
- Reads from freed or uninitialized memory,

Assertions

The heap implementation (`multi_heap.c`, etc.) includes a lot of assertions which will fail if the heap memory is corrupted. To detect heap corruption most effectively, ensure that assertions are enabled in the project configuration menu under `Compiler options` -> *CONFIG_COMPILER_OPTIMIZATION_ASSERTION_LEVEL*.

If a heap integrity assertion fails, a line will be printed like `CORRUPT HEAP: multi_heap.c:225 detected at 0x3ffbb71c`. The memory address which is printed is the address of the heap structure which has corrupt content.

It's also possible to manually check heap integrity by calling *heap_caps_check_integrity_all()* or related functions. This function checks all of requested heap memory for integrity, and can be used even if assertions are disabled. If the integrity check prints an error, it will also contain the address(es) of corrupt heap structures.

Finding Heap Corruption

Memory corruption can be one of the hardest classes of bugs to find and fix, as one area of memory can be corrupted from a totally different place. Some tips:

- A crash with a `CORRUPT HEAP:` message will usually include a stack trace, but this stack trace is rarely useful. The crash is the symptom of memory corruption when the system realises the heap is corrupt, but usually the corruption happened elsewhere and earlier in time.
- Increasing the Heap memory debugging *Configuration* level to "Light impact" or "Comprehensive" can give you a more accurate message with the first corrupt memory address.

- Adding regular calls to *heap_caps_check_integrity_all()* or *heap_caps_check_integrity_addr()* in your code will help you pin down the exact time that the corruption happened. You can move these checks around to "close in on" the section of code that corrupted the heap.
- Based on the memory address which is being corrupted, you can use *JTAG debugging* to set a watchpoint on this address and have the CPU halt when it is written to.
- If you don't have JTAG, but you do know roughly when the corruption happens, then you can set a watchpoint in software just beforehand via `esp_set_watchpoint()`. A fatal exception will occur when the watchpoint triggers. For example `esp_set_watchpoint(0, (void *)addr, 4, ESP_WATCHPOINT_STORE`. Note that watchpoints are per-CPU and are set on the current running CPU only, so if you don't know which CPU is corrupting memory then you will need to call this function on both CPUs.
- For buffer overflows, *heap tracing* in `HEAP_TRACE_ALL` mode lets you see which callers are allocating which addresses from the heap. See *Heap Tracing To Find Heap Corruption* for more details. If you can find the function which allocates memory with an address immediately before the address which is corrupted, this will probably be the function which overflows the buffer.
- Calling *heap_caps_dump()* or *heap_caps_dump_all()* can give an indication of what heap blocks are surrounding the corrupted region and may have overflowed/underflowed/etc.

Configuration

Temporarily increasing the heap corruption detection level can give more detailed information about heap corruption errors.

In the project configuration menu, under `Component config` there is a menu `Heap memory debugging`. The setting *CONFIG_HEAP_CORRUPTION_DETECTION* can be set to one of three levels:

Basic (no poisoning)

This is the default level. No special heap corruption features are enabled, but provided assertions are enabled (the default configuration) then a heap corruption error will be printed if any of the heap's internal data structures appear overwritten or corrupted. This usually indicates a buffer overrun or out of bounds write.

If assertions are enabled, an assertion will also trigger if a double-free occurs (the same memory is freed twice).

Calling *heap_caps_check_integrity()* in Basic mode will check the integrity of all heap structures, and print errors if any appear to be corrupted.

Light Impact

At this level, heap memory is additionally "poisoned" with head and tail "canary bytes" before and after each block which is allocated. If an application writes outside the bounds of allocated buffers, the canary bytes will be corrupted and the integrity check will fail.

The head canary word is 0xABBA1234 (3412BAAB in byte order), and the tail canary word is 0xBAAD5678 (7856ADBA in byte order).

"Basic" heap corruption checks can also detect most out of bounds writes, but this setting is more precise as even a single byte overrun can be detected. With Basic heap checks, the number of overrun bytes before a failure is detected will depend on the properties of the heap.

Enabling "Light Impact" checking increases memory usage, each individual allocation will use 9 to 12 additional bytes of memory (depending on alignment).

Each time `free()` is called in Light Impact mode, the head and tail canary bytes of the buffer being freed are checked against the expected values.

When *heap_caps_check_integrity()* is called, all allocated blocks of heap memory have their canary bytes checked against the expected values.

In both cases, the check is that the first 4 bytes of an allocated block (before the buffer returned to the user) should be the word 0xABBA1234. Then the last 4 bytes of the allocated block (after the buffer returned to the user) should be the word 0xBAAD5678.

Different values usually indicate buffer underrun or overrun, respectively.

Comprehensive

This level incorporates the "light impact" detection features plus additional checks for uninitialised-access and use-after-free bugs. In this mode, all freshly allocated memory is filled with the pattern 0xCE, and all freed memory is filled with the pattern 0xFE.

Enabling "Comprehensive" detection has a substantial runtime performance impact (as all memory needs to be set to the allocation patterns each time a malloc/free completes, and the memory also needs to be checked each time.) However it allows easier detection of memory corruption bugs which are much more subtle to find otherwise. It is recommended to only enable this mode when debugging, not in production.

Crashes in Comprehensive Mode

If an application crashes reading/writing an address related to 0xCECECECE in Comprehensive mode, this indicates it has read uninitialized memory. The application should be changed to either use calloc() (which zeroes memory), or initialize the memory before using it. The value 0xCECECECE may also be seen in stack-allocated automatic variables, because in IDF most task stacks are originally allocated from the heap and in C stack memory is uninitialized by default.

If an application crashes and the exception register dump indicates that some addresses or values were 0xFEFEFEFE, this indicates it is reading heap memory after it has been freed (a "use after free bug".) The application should be changed to not access heap memory after it has been freed.

If a call to malloc() or realloc() causes a crash because it expected to find the pattern 0xFEFEFEFE in free memory and a different pattern was found, then this indicates the app has a use-after-free bug where it is writing to memory which has already been freed.

Manual Heap Checks in Comprehensive Mode

Calls to *heap_caps_check_integrity()* may print errors relating to 0xFEFEFEFE, 0xABBA1234 or 0xBAAD5678. In each case the checker is expecting to find a given pattern, and will error out if this is not found:

- For free heap blocks, the checker expects to find all bytes set to 0xFE. Any other values indicate a use-after-free bug where free memory has been incorrectly overwritten.
- For allocated heap blocks, the behaviour is the same as for *Light Impact* mode. The canary bytes 0xABBA1234 and 0xBAAD5678 are checked at the head and tail of each allocated buffer, and any variation indicates a buffer overrun/underrun.

Heap Tracing

Heap Tracing allows tracing of code which allocates/frees memory. Two tracing modes are supported:

- Standalone. In this mode trace data are kept on-board, so the size of gathered information is limited by the buffer assigned for that purposes. Analysis is done by the on-board code. There are a couple of APIs available for accessing and dumping collected info.
- Host-based. This mode does not have the limitation of the standalone mode, because trace data are sent to the host over JTAG connection using app_trace library. Later on they can be analysed using special tools.

Heap tracing can perform two functions:

- Leak checking: find memory which is allocated and never freed.
- Heap use analysis: show all functions that are allocating/freeing memory while the trace is running.

How To Diagnose Memory Leaks

If you suspect a memory leak, the first step is to figure out which part of the program is leaking memory. Use the `xPortGetFreeHeapSize()`, *heap_caps_get_free_size()*, or *related functions* to track memory use over the life of the application. Try to narrow the leak down to a single function or sequence of functions where free memory always decreases and never recovers.

Standalone Mode

Once you've identified the code which you think is leaking:

- In the project configuration menu, navigate to `Component settings` -> `Heap Memory Debugging` -> `Heap tracing` and select `Standalone` option (see *CONFIG_HEAP_TRACING_DEST*).
- Call the function *heap_trace_init_standalone()* early in the program, to register a buffer which can be used to record the memory trace.
- Call the function *heap_trace_start()* to begin recording all mallocs/frees in the system. Call this immediately before the piece of code which you suspect is leaking memory.
- Call the function *heap_trace_stop()* to stop the trace once the suspect piece of code has finished executing.
- Call the function *heap_trace_dump()* to dump the results of the heap trace.

An example:

```
#include "esp_heap_trace.h"

#define NUM_RECORDS 100
static heap_trace_record_t trace_record[NUM_RECORDS]; // This buffer must be in internal␣
↪RAM

...

void app_main()
{
    ...
    ESP_ERROR_CHECK( heap_trace_init_standalone(trace_record, NUM_RECORDS) );
    ...
}

void some_function()
{
    ESP_ERROR_CHECK( heap_trace_start(HEAP_TRACE_LEAKS) );

    do_something_you_suspect_is_leaking();

    ESP_ERROR_CHECK( heap_trace_stop() );
    heap_trace_dump();
    ...
}
```

The output from the heap trace will look something like this:

```
2 allocations trace (100 entry buffer)
32 bytes (@ 0x3ffaf214) allocated CPU 0 ccount 0x2e9b7384 caller 0x400d276d:0x400d27c1
0x400d276d: leak_some_memory at /path/to/idf/examples/get-started/blink/main/./blink.c:27

0x400d27c1: blink_task at /path/to/idf/examples/get-started/blink/main/./blink.c:52

8 bytes (@ 0x3ffaf804) allocated CPU 0 ccount 0x2e9b79c0 caller 0x400d2776:0x400d27c1
0x400d2776: leak_some_memory at /path/to/idf/examples/get-started/blink/main/./blink.c:29

0x400d27c1: blink_task at /path/to/idf/examples/get-started/blink/main/./blink.c:52

40 bytes 'leaked' in trace (2 allocations)
total allocations 2 total frees 0
```

(Above example output is using *IDF Monitor* to automatically decode PC addresses to their source files & line number.)

The first line indicates how many allocation entries are in the buffer, compared to its total size.

In `HEAP_TRACE_LEAKS` mode, for each traced memory allocation which has not already been freed a line is printed with:

- `XX bytes` is number of bytes allocated
- `@ 0x...` is the heap address returned from malloc/calloc.
- `CPU x` is the CPU (0 or 1) running when the allocation was made.
- `ccount 0x...` is the CCOUNT (CPU cycle count) register value when the allocation was mode. Is different for CPU 0 vs CPU 1.
- `caller 0x...` gives the call stack of the call to malloc()/free(), as a list of PC addresses. These can be decoded to source files and line numbers, as shown above.

The depth of the call stack recorded for each trace entry can be configured in the project configuration menu, under `Heap Memory Debugging` -> `Enable heap tracing` -> `Heap tracing stack depth`. Up to 10 stack frames can be recorded for each allocation (the default is 2). Each additional stack frame increases the memory usage of each `heap_trace_record_t` record by eight bytes.

Finally, the total number of 'leaked' bytes (bytes allocated but not freed while trace was running) is printed, and the total number of allocations this represents.

A warning will be printed if the trace buffer was not large enough to hold all the allocations which happened. If you see this warning, consider either shortening the tracing period or increasing the number of records in the trace buffer.

Host-Based Mode

Once you've identified the code which you think is leaking:

- In the project configuration menu, navigate to `Component settings` -> `Heap Memory Debugging` -> *CONFIG_HEAP_TRACING_DEST* and select `Host-Based`.
- In the project configuration menu, navigate to `Component settings` -> `Application Level Tracing` -> *CONFIG_APPTRACE_DESTINATION* and select `Trace memory`.
- In the project configuration menu, navigate to `Component settings` -> `Application Level Tracing` -> `FreeRTOS SystemView Tracing` and enable *CONFIG_SYSVIEW_ENABLE*.
- Call the function *heap_trace_init_tohost()* early in the program, to initialize JTAG heap tracing module.
- Call the function *heap_trace_start()* to begin recording all mallocs/frees in the system. Call this immediately before the piece of code which you suspect is leaking memory. In host-based mode argument to this function is ignored and heap tracing module behaves like `HEAP_TRACE_ALL` was passed: all allocations and deallocations are sent to the host.
- Call the function *heap_trace_stop()* to stop the trace once the suspect piece of code has finished executing.

An example:

```
#include "esp_heap_trace.h"

...

void app_main()
{
    ...
    ESP_ERROR_CHECK( heap_trace_init_tohost() );
    ...
}

void some_function()
{
    ESP_ERROR_CHECK( heap_trace_start(HEAP_TRACE_LEAKS) );

    do_something_you_suspect_is_leaking();

    ESP_ERROR_CHECK( heap_trace_stop() );
    ...
}
```

To gather and analyse heap trace do the following on the host:

1. Build the program and download it to the target as described in *Getting Started Guide*.
2. Run OpenOCD (see *JTAG Debugging*).

注解: In order to use this feature you need OpenOCD version *v0.10.0-esp32-20181105* or later.

---

3. You can use GDB to start and/or stop tracing automatically. To do this you need to prepare special `gdbinit` file:

```
target remote :3333

mon reset halt
flushregs

tb heap_trace_start
commands
mon esp32 sysview start file:///tmp/heap.svdat
c
end

tb heap_trace_stop
commands
mon esp32 sysview stop
end

c
```

Using this file GDB will connect to the target, reset it, and start tracing when program hits breakpoint at *heap_trace_start()*. Trace data will be saved to `/tmp/heap_log.svdat`. Tracing will be stopped when program hits breakpoint at *heap_trace_stop()*.

4. Run GDB using the following command `xtensa-esp32-elf-gdb -x gdbinit </path/to/program/elf>`
5. Quit GDB when program stops at *heap_trace_stop()*. Trace data are saved in `/tmp/heap.svdat`
6. Run processing script `$IDF_PATH/tools/esp_app_trace/sysviewtrace_proc.py -p -b </path/to/program/elf> /tmp/heap_log.svdat`

The output from the heap trace will look something like this:

```
Parse trace from '/tmp/heap.svdat'...
Stop parsing trace. (Timeout 0.000000 sec while reading 1 bytes!)
Process events from '['/tmp/heap.svdat']'...
[0.002244575] HEAP: Allocated 1 bytes @ 0x3ffaffd8 from task "alloc" on core 0 by:
/home/user/projects/esp/esp-idf/examples/system/sysview_tracing_heap_log/main/sysview_
↪heap_log.c:47
/home/user/projects/esp/esp-idf/components/freertos/port.c:355 (discriminator 1)

[0.002258425] HEAP: Allocated 2 bytes @ 0x3ffaffe0 from task "alloc" on core 0 by:
/home/user/projects/esp/esp-idf/examples/system/sysview_tracing_heap_log/main/sysview_
↪heap_log.c:48
/home/user/projects/esp/esp-idf/components/freertos/port.c:355 (discriminator 1)

[0.002563725] HEAP: Freed bytes @ 0x3ffaffe0 from task "free" on core 0 by:
/home/user/projects/esp/esp-idf/examples/system/sysview_tracing_heap_log/main/sysview_
↪heap_log.c:31 (discriminator 9)
/home/user/projects/esp/esp-idf/components/freertos/port.c:355 (discriminator 1)

[0.002782950] HEAP: Freed bytes @ 0x3ffb40b8 from task "main" on core 0 by:
/home/user/projects/esp/esp-idf/components/freertos/tasks.c:4590
/home/user/projects/esp/esp-idf/components/freertos/tasks.c:4590

[0.002798700] HEAP: Freed bytes @ 0x3ffb50bc from task "main" on core 0 by:
/home/user/projects/esp/esp-idf/components/freertos/tasks.c:4590
/home/user/projects/esp/esp-idf/components/freertos/tasks.c:4590

[0.102436025] HEAP: Allocated 2 bytes @ 0x3ffaffe0 from task "alloc" on core 0 by:
/home/user/projects/esp/esp-idf/examples/system/sysview_tracing_heap_log/main/sysview_
↪heap_log.c:47
/home/user/projects/esp/esp-idf/components/freertos/port.c:355 (discriminator 1)

[0.102449800] HEAP: Allocated 4 bytes @ 0x3ffaffe8 from task "alloc" on core 0 by:
/home/user/projects/esp/esp-idf/examples/system/sysview_tracing_heap_log/main/sysview_
↪heap_log.c:48
/home/user/projects/esp/esp-idf/components/freertos/port.c:355 (discriminator 1)

[0.102666150] HEAP: Freed bytes @ 0x3ffaffe8 from task "free" on core 0 by:
/home/user/projects/esp/esp-idf/examples/system/sysview_tracing_heap_log/main/sysview_
↪heap_log.c:31 (discriminator 9)
/home/user/projects/esp/esp-idf/components/freertos/port.c:355 (discriminator 1)
```

(下页继续)

(续上页)

```
[0.202436200] HEAP: Allocated 3 bytes @ 0x3ffaffe8 from task "alloc" on core 0 by:
/home/user/projects/esp/esp-idf/examples/system/sysview_tracing_heap_log/main/sysview_
↪heap_log.c:47
/home/user/projects/esp/esp-idf/components/freertos/port.c:355 (discriminator 1)

[0.202451725] HEAP: Allocated 6 bytes @ 0x3ffafff0 from task "alloc" on core 0 by:
/home/user/projects/esp/esp-idf/examples/system/sysview_tracing_heap_log/main/sysview_
↪heap_log.c:48
/home/user/projects/esp/esp-idf/components/freertos/port.c:355 (discriminator 1)

[0.202667075] HEAP: Freed bytes @ 0x3ffafff0 from task "free" on core 0 by:
/home/user/projects/esp/esp-idf/examples/system/sysview_tracing_heap_log/main/sysview_
↪heap_log.c:31 (discriminator 9)
/home/user/projects/esp/esp-idf/components/freertos/port.c:355 (discriminator 1)

[0.302436000] HEAP: Allocated 4 bytes @ 0x3ffafff0 from task "alloc" on core 0 by:
/home/user/projects/esp/esp-idf/examples/system/sysview_tracing_heap_log/main/sysview_
↪heap_log.c:47
/home/user/projects/esp/esp-idf/components/freertos/port.c:355 (discriminator 1)

[0.302451475] HEAP: Allocated 8 bytes @ 0x3ffb40b8 from task "alloc" on core 0 by:
/home/user/projects/esp/esp-idf/examples/system/sysview_tracing_heap_log/main/sysview_
↪heap_log.c:48
/home/user/projects/esp/esp-idf/components/freertos/port.c:355 (discriminator 1)

[0.302667500] HEAP: Freed bytes @ 0x3ffb40b8 from task "free" on core 0 by:
/home/user/projects/esp/esp-idf/examples/system/sysview_tracing_heap_log/main/sysview_
↪heap_log.c:31 (discriminator 9)
/home/user/projects/esp/esp-idf/components/freertos/port.c:355 (discriminator 1)

Processing completed.
Processed 1019 events
=============== HEAP TRACE REPORT ===============
Processed 14 heap events.
[0.002244575] HEAP: Allocated 1 bytes @ 0x3ffaffd8 from task "alloc" on core 0 by:
/home/user/projects/esp/esp-idf/examples/system/sysview_tracing_heap_log/main/sysview_
↪heap_log.c:47
/home/user/projects/esp/esp-idf/components/freertos/port.c:355 (discriminator 1)
```

(下页继续)

(续上页)

```
[0.102436025] HEAP: Allocated 2 bytes @ 0x3ffaffe0 from task "alloc" on core 0 by:
/home/user/projects/esp/esp-idf/examples/system/sysview_tracing_heap_log/main/sysview_
↪heap_log.c:47
/home/user/projects/esp/esp-idf/components/freertos/port.c:355 (discriminator 1)

[0.202436200] HEAP: Allocated 3 bytes @ 0x3ffaffe8 from task "alloc" on core 0 by:
/home/user/projects/esp/esp-idf/examples/system/sysview_tracing_heap_log/main/sysview_
↪heap_log.c:47
/home/user/projects/esp/esp-idf/components/freertos/port.c:355 (discriminator 1)

[0.302436000] HEAP: Allocated 4 bytes @ 0x3ffafff0 from task "alloc" on core 0 by:
/home/user/projects/esp/esp-idf/examples/system/sysview_tracing_heap_log/main/sysview_
↪heap_log.c:47
/home/user/projects/esp/esp-idf/components/freertos/port.c:355 (discriminator 1)

Found 10 leaked bytes in 4 blocks.
```

Heap Tracing To Find Heap Corruption

Heap tracing can also be used to help track down heap corruption. When a region in heap is corrupted, it may be from some other part of the program which allocated memory at a nearby address.

If you have some idea at what time the corruption occurred, enabling heap tracing in `HEAP_TRACE_ALL` mode allows you to record all of the functions which allocated memory, and the addresses of the allocations.

Using heap tracing in this way is very similar to memory leak detection as described above. For memory which is allocated and not freed, the output is the same. However, records will also be shown for memory which has been freed.

Performance Impact

Enabling heap tracing in menuconfig increases the code size of your program, and has a very small negative impact on performance of heap allocation/free operations even when heap tracing is not running.

When heap tracing is running, heap allocation/free operations are substantially slower than when heap tracing is stopped. Increasing the depth of stack frames recorded for each allocation (see above) will also increase this performance impact.

False-Positive Memory Leaks

Not everything printed by *heap_trace_dump()* is necessarily a memory leak. Among things which may show up here, but are not memory leaks:

- Any memory which is allocated after *heap_trace_start()* but then freed after *heap_trace_stop()* will appear in the leak dump.
- Allocations may be made by other tasks in the system. Depending on the timing of these tasks, it's quite possible this memory is freed after *heap_trace_stop()* is called.
- The first time a task uses stdio - for example, when it calls `printf()` - a lock (RTOS mutex semaphore) is allocated by the libc. This allocation lasts until the task is deleted.
- Certain uses of `printf()`, such as printing floating point numbers, will allocate some memory from the heap on demand. These allocations last until the task is deleted.
- The Bluetooth, WiFi, and TCP/IP libraries will allocate heap memory buffers to handle incoming or outgoing data. These memory buffers are usually short lived, but some may be shown in the heap leak trace if the data was received/transmitted by the lower levels of the network while the leak trace was running.
- TCP connections will continue to use some memory after they are closed, because of the `TIME_WAIT` state. After the `TIME_WAIT` period has completed, this memory will be freed.

One way to differentiate between "real" and "false positive" memory leaks is to call the suspect code multiple times while tracing is running, and look for patterns (multiple matching allocations) in the heap trace output.

API Reference - Heap Tracing

Header File

- heap/include/esp_heap_trace.h

Functions

esp_err_t **heap_trace_init_standalone**(*heap_trace_record_t* **record_buffer*, size_t *num_records*)

Initialise heap tracing in standalone mode.

This function must be called before any other heap tracing functions.

To disable heap tracing and allow the buffer to be freed, stop tracing and then call heap_trace_init_standalone(NULL, 0);

Return

- ESP_ERR_NOT_SUPPORTED Project was compiled without heap tracing enabled in menuconfig.
- ESP_ERR_INVALID_STATE Heap tracing is currently in progress.
- ESP_OK Heap tracing initialised successfully.

Parameters

- `record_buffer`: Provide a buffer to use for heap trace data. Must remain valid any time heap tracing is enabled, meaning it must be allocated from internal memory not in PSRAM.
- `num_records`: Size of the heap trace buffer, as number of record structures.

esp_err_t **heap_trace_init_tohost**(void)

Initialise heap tracing in host-based mode.

This function must be called before any other heap tracing functions.

Return

- ESP_ERR_INVALID_STATE Heap tracing is currently in progress.
- ESP_OK Heap tracing initialised successfully.

esp_err_t **heap_trace_start**(*heap_trace_mode_t mode*)

Start heap tracing. All heap allocations & frees will be traced, until heap_trace_stop() is called.

Note heap_trace_init_standalone() must be called to provide a valid buffer, before this function is called.

Note Calling this function while heap tracing is running will reset the heap trace state and continue tracing.

Return

- ESP_ERR_NOT_SUPPORTED Project was compiled without heap tracing enabled in menuconfig.
- ESP_ERR_INVALID_STATE A non-zero-length buffer has not been set via heap_trace_init_standalone().
- ESP_OK Tracing is started.

Parameters

- `mode`: Mode for tracing.
  - HEAP_TRACE_ALL means all heap allocations and frees are traced.
  - HEAP_TRACE_LEAKS means only suspected memory leaks are traced. (When memory is freed, the record is removed from the trace buffer.)

esp_err_t **heap_trace_stop**(void)

Stop heap tracing.

Return

- ESP_ERR_NOT_SUPPORTED Project was compiled without heap tracing enabled in menuconfig.
- ESP_ERR_INVALID_STATE Heap tracing was not in progress.
- ESP_OK Heap tracing stopped..

esp_err_t **heap_trace_resume**(void)

Resume heap tracing which was previously stopped.

Unlike heap_trace_start(), this function does not clear the buffer of any pre-existing trace records.

The heap trace mode is the same as when heap_trace_start() was last called (or HEAP_TRACE_ALL if heap_trace_start() was never called).

Return

- ESP_ERR_NOT_SUPPORTED Project was compiled without heap tracing enabled in menuconfig.
- ESP_ERR_INVALID_STATE Heap tracing was already started.
- ESP_OK Heap tracing resumed.

size_t **heap_trace_get_count**(void)

Return number of records in the heap trace buffer.

It is safe to call this function while heap tracing is running.

esp_err_t **heap_trace_get**(size_t *index*, *heap_trace_record_t* **record*)

Return a raw record from the heap trace buffer.

Note It is safe to call this function while heap tracing is running, however in HEAP_TRACE_LEAK mode record indexing may skip entries unless heap tracing is stopped first.

Return

- ESP_ERR_NOT_SUPPORTED Project was compiled without heap tracing enabled in menuconfig.
- ESP_ERR_INVALID_STATE Heap tracing was not initialised.
- ESP_ERR_INVALID_ARG Index is out of bounds for current heap trace record count.
- ESP_OK Record returned successfully.

Parameters

- `index`: Index (zero-based) of the record to return.
- `record`: Record where the heap trace record will be copied.

void **heap_trace_dump**(void)

Dump heap trace record data to stdout.

Note It is safe to call this function while heap tracing is running, however in HEAP_TRACE_LEAK mode the dump may skip entries unless heap tracing is stopped first.

Structures

struct heap_trace_record_t

Trace record data type. Stores information about an allocated region of memory.

Public Members

uint32_t **ccount**

CCOUNT of the CPU when the allocation was made. LSB (bit value 1) is the CPU number (0 or 1).

void ***address**

Address which was allocated.

size_t **size**

Size of the allocation.

void ***alloced_by**[CONFIG_HEAP_TRACING_STACK_DEPTH]

Call stack of the caller which allocated the memory.

void ***freed_by**[CONFIG_HEAP_TRACING_STACK_DEPTH]

Call stack of the caller which freed the memory (all zero if not freed.)

Macros

CONFIG_HEAP_TRACING_STACK_DEPTH

Enumerations

enum heap_trace_mode_t

Values:

HEAP_TRACE_ALL

HEAP_TRACE_LEAKS

2.7.12 High Resolution Timer

Overview

Although FreeRTOS provides software timers, these timers have a few limitations:

- Maximum resolution is equal to RTOS tick period
- Timer callbacks are dispatched from a low-priority task

Hardware timers are free from both of the limitations, but often they are less convenient to use. For example, application components may need timer events to fire at certain times in the future, but the hardware timer only contains one "compare" value used for interrupt generation. This means that some facility needs to be built on top of the hardware timer to manage the list of pending events can dispatch the callbacks for these events as corresponding hardware interrupts happen.

`esp_timer` set of APIs provide such facility. Internally, `esp_timer` uses a 32-bit hardware timer (FRC1, "legacy" timer). `esp_timer` provides one-shot and periodic timers, microsecond time resolution, and 64-bit range.

Timer callbacks are dispatched from a high-priority `esp_timer` task. Because all the callbacks are dispatched from the same task, it is recommended to only do the minimal possible amount of work from the callback itself, posting an event to a lower priority task using a queue instead.

If other tasks with priority higher than `esp_timer` are running, callback dispatching will be delayed until `esp_timer` task has a chance to run. For example, this will happen if a SPI Flash operation is in progress.

Creating and starting a timer, and dispatching the callback takes some time. Therefore there is a lower limit to the timeout value of one-shot `esp_timer`. If *esp_timer_start_once()* is called with a timeout value less than 20us, the callback will be dispatched only after approximately 20us.

Periodic `esp_timer` also imposes a 50us restriction on the minimal timer period. Periodic software timers with period of less than 50us are not practical since they would consume most of the CPU time. Consider using dedicated hardware peripherals or DMA features if you find that a timer with small period is required.

Using `esp_timer` APIs

Single timer is represented by *esp_timer_handle_t* type. Timer has a callback function associated with it. This callback function is called from the `esp_timer` task each time the timer elapses.

- To create a timer, call *esp_timer_create()*.
- To delete the timer when it is no longer needed, call *esp_timer_delete()*.

The timer can be started in one-shot mode or in periodic mode.

- To start the timer in one-shot mode, call *esp_timer_start_once()*, passing the time interval after which the callback should be called. When the callback gets called, the timer is considered to be stopped.

- To start the timer in periodic mode, call *esp_timer_start_periodic()*, passing the period with which the callback should be called. The timer keeps running until *esp_timer_stop()* is called.

Note that the timer must not be running when *esp_timer_start_once()* or *esp_timer_start_periodic()* is called. To restart a running timer, call *esp_timer_stop()* first, then call one of the start functions.

Obtaining Current Time

`esp_timer` also provides a convenience function to obtain the time passed since start-up, with microsecond precision: *esp_timer_get_time()*. This function returns the number of microseconds since `esp_timer` was initialized, which usually happens shortly before `app_main` function is called.

Unlike *gettimeofday* function, values returned by *esp_timer_get_time()*:

- Start from zero after the chip wakes up from deep sleep
- Do not have timezone or DST adjustments applied

Application Example

The following example illustrates usage of `esp_timer` APIs: system/esp_timer.

API Reference

Header File

- esp_common/include/esp_timer.h

Functions

esp_err_t **esp_timer_init**(void)

Initialize esp_timer library.

Note This function is called from startup code. Applications do not need to call this function before using other esp_timer APIs.

Return

- ESP_OK on success
- ESP_ERR_NO_MEM if allocation has failed
- ESP_ERR_INVALID_STATE if already initialized
- other errors from interrupt allocator

esp_err_t **esp_timer_deinit**(void)

De-initialize esp_timer library.

Note Normally this function should not be called from applications

Return

- ESP_OK on success
- ESP_ERR_INVALID_STATE if not yet initialized

esp_err_t **esp_timer_create**(**const** *esp_timer_create_args_t* **create_args*, *esp_timer_handle_t* **out_handle*)

Create an esp_timer instance.

Note When done using the timer, delete it with esp_timer_delete function.

Return

- ESP_OK on success
- ESP_ERR_INVALID_ARG if some of the create_args are not valid
- ESP_ERR_INVALID_STATE if esp_timer library is not initialized yet
- ESP_ERR_NO_MEM if memory allocation fails

Parameters

- `create_args`: Pointer to a structure with timer creation arguments. Not saved by the library, can be allocated on the stack.
- `out_handle`: Output, pointer to esp_timer_handle_t variable which will hold the created timer handle.

esp_err_t **esp_timer_start_once**(*esp_timer_handle_t timer*, uint64_t *timeout_us*)

Start one-shot timer.

Timer should not be running when this function is called.

Return

- ESP_OK on success
- ESP_ERR_INVALID_ARG if the handle is invalid
- ESP_ERR_INVALID_STATE if the timer is already running

Parameters

- `timer`: timer handle created using esp_timer_create
- `timeout_us`: timer timeout, in microseconds relative to the current moment

esp_err_t **esp_timer_start_periodic**(*esp_timer_handle_t timer*, uint64_t *period*)

Start a periodic timer.

Timer should not be running when this function is called. This function will start the timer which will trigger every 'period' microseconds.

Return

- ESP_OK on success
- ESP_ERR_INVALID_ARG if the handle is invalid
- ESP_ERR_INVALID_STATE if the timer is already running

Parameters

- `timer`: timer handle created using esp_timer_create
- `period`: timer period, in microseconds

esp_err_t **esp_timer_stop**(*esp_timer_handle_t timer*)

Stop the timer.

This function stops the timer previously started using esp_timer_start_once or esp_timer_start_periodic.

Return

- ESP_OK on success
- ESP_ERR_INVALID_STATE if the timer is not running

Parameters

- `timer`: timer handle created using esp_timer_create

esp_err_t **esp_timer_delete**(*esp_timer_handle_t timer*)

Delete an esp_timer instance.

The timer must be stopped before deleting. A one-shot timer which has expired does not need to be stopped.

Return

- ESP_OK on success
- ESP_ERR_INVALID_STATE if the timer is not running

Parameters

- `timer`: timer handle allocated using esp_timer_create

int64_t **esp_timer_get_time**(void)

Get time in microseconds since boot.

Return number of microseconds since esp_timer_init was called (this normally happens early during application startup).

int64_t **esp_timer_get_next_alarm**(void)

Get the timestamp when the next timeout is expected to occur.

Return Timestamp of the nearest timer event, in microseconds. The timebase is the same as for the values returned by esp_timer_get_time.

esp_err_t **esp_timer_dump**(FILE **stream*)

Dump the list of timers to a stream.

If CONFIG_ESP_TIMER_PROFILING option is enabled, this prints the list of all the existing timers. Otherwise, only the list active timers is printed.

The format is:

name period alarm times_armed times_triggered total_callback_run_time

where:

name —timer name (if CONFIG_ESP_TIMER_PROFILING is defined), or timer pointer period — period of timer, in microseconds, or 0 for one-shot timer alarm - time of the next alarm, in microseconds since boot, or 0 if the timer is not started

The following fields are printed if CONFIG_ESP_TIMER_PROFILING is defined:

times_armed —number of times the timer was armed via esp_timer_start_X times_triggered - number of times the callback was called total_callback_run_time - total time taken by callback to execute, across all calls

Return

- ESP_OK on success
- ESP_ERR_NO_MEM if can not allocate temporary buffer for the output

Parameters

- **stream**: stream (such as stdout) to dump the information to

Structures

struct esp_timer_create_args_t

Timer configuration passed to esp_timer_create.

Public Members

esp_timer_cb_t **callback**

Function to call when timer expires.

void ***arg**

Argument to pass to the callback.

esp_timer_dispatch_t **dispatch_method**

Call the callback from task or from ISR.

const char ***name**

Timer name, used in esp_timer_dump function.

Type Definitions

typedef struct esp_timer ***esp_timer_handle_t**

Opaque type representing a single esp_timer.

typedef void (***esp_timer_cb_t**)(void *arg)

Timer callback function type.

Parameters

- **arg**: pointer to opaque user-specific data

Enumerations

enum esp_timer_dispatch_t

Method for dispatching timer callback.

Values:

ESP_TIMER_TASK

Callback is called from timer task.

2.7.13 The himem allocation API

Overview

The ESP32 can access external SPI RAM transparently, so you can use it as normal memory in your program code. However, because the address space for external memory is limited in size, only the first 4MiB can be used as such. Access to the remaining memory is still possible, however this needs to go through a bankswitching scheme controlled by the himem API.

Specifically, what is implemented by the himem API is a bankswitching scheme. Hardware-wise, the 4MiB region for external SPI RAM is mapped into the CPU address space by a MMU, which maps a configurable 32K bank/page of external SPI RAM into each of the 32K pages in the 4MiB region accessed by the CPU. For external memories that are <=4MiB, this MMU is configured to unity mapping, effectively mapping each CPU address 1-to-1 to the external SPI RAM address.

In order to use the himem API, you have to enable it in the menuconfig using `CONFIG_SPIRAM_BANKSWITCH_ENABLE`, as well as set the amount of banks reserved for this in `CONFIG_SPIRAM_BANKSWITCH_RESERVE`. This decreases the amount of external memory allocated by functions like `malloc()`, but it allows you to use the himem api to map any of the remaining memory into the reserved banks.

The himem API is more-or-less an abstraction of the bankswitching scheme: it allows you to claim one or more banks of address space (called 'regions' in the API) as well as one or more of banks of memory to map into the ranges.

Example

An example doing a simple memory test of the high memory range is available in esp-idf: system/himem

API Reference

Header File

- esp32/include/esp32/himem.h

Functions

esp_err_t **esp_himem_alloc**(size_t *size*, *esp_himem_handle_t* **handle_out*)

Allocate a block in high memory.

Return - ESP_OK if succesful

- ESP_ERR_NO_MEM if out of memory
- ESP_ERR_INVALID_SIZE if size is not a multiple of 32K

Parameters

- `size`: Size of the to-be-allocated block, in bytes. Note that this needs to be a multiple of the external RAM mmu block size (32K).
- `handle_out`: Handle to be returned

esp_err_t **esp_himem_alloc_map_range**(size_t *size*, *esp_himem_rangehandle_t* **handle_out*)

Allocate a memory region to map blocks into.

This allocates a contiguous CPU memory region that can be used to map blocks of physical memory into.

Return - ESP_OK if succesful

- ESP_ERR_NO_MEM if out of memory or address space
- ESP_ERR_INVALID_SIZE if size is not a multiple of 32K

Parameters

- `size`: Size of the range to be allocated. Note this needs to be a multiple of the external RAM mmu block size (32K).
- `handle_out`: Handle to be returned

esp_err_t **esp_himem_map**(*esp_himem_handle_t handle*, *esp_himem_rangehandle_t range*, size_t *ram_offset*, size_t *range_offset*, size_t *len*, int *flags*, void **out_ptr*)

Map a block of high memory into the CPUs address space.

This effectively makes the block available for read/write operations.

Note The region to be mapped needs to have offsets and sizes that are aligned to the SPI RAM MMU block size (32K)

Return - ESP_OK if the memory could be mapped

- ESP_ERR_INVALID_ARG if offset, range or len aren't MMU-block-aligned (32K)
- ESP_ERR_INVALID_SIZE if the offsets/lengths don't fit in the allocated memory or range
- ESP_ERR_INVALID_STATE if a block in the selected ram offset/length is already mapped, or if a block in the selected range offset/length already has a mapping.

Parameters

- `handle`: Handle to the block of memory, as given by esp_himem_alloc
- `range`: Range handle to map the memory in
- `ram_offset`: Offset into the block of physical memory of the block to map
- `range_offset`: Offset into the address range where the block will be mapped
- `len`: Length of region to map
- `flags`: One of ESP_HIMEM_MAPFLAG_*
- `out_ptr`: Pointer to variable to store resulting memory pointer in

esp_err_t **esp_himem_free**(*esp_himem_handle_t handle*)

Free a block of physical memory.

This clears out the associated handle making the memory available for re-allocation again. This will only succeed if none of the memory blocks currently have a mapping.

Return - ESP_OK if the memory is succesfully freed

- ESP_ERR_INVALID_ARG if the handle still is (partially) mapped

Parameters

- `handle`: Handle to the block of memory, as given by esp_himem_alloc

esp_err_t **esp_himem_free_map_range**(*esp_himem_rangehandle_t handle*)

Free a mapping range.

This clears out the associated handle making the range available for re-allocation again. This will only succeed if none of the range blocks currently are used for a mapping.

Return - ESP_OK if the memory is succesfully freed

- ESP_ERR_INVALID_ARG if the handle still is (partially) mapped to

Parameters

- `handle`: Handle to the range block, as given by esp_himem_alloc_map_range

esp_err_t **esp_himem_unmap**(*esp_himem_rangehandle_t range*, void **ptr*, size_t *len*)

Unmap a region.

Return - ESP_OK if the memory is succesfully unmapped,

- ESP_ERR_INVALID_ARG if ptr or len are invalid.

Parameters

- `range`: Range handle
- `ptr`: Pointer returned by esp_himem_map
- `len`: Length of the block to be unmapped. Must be aligned to the SPI RAM MMU blocksize (32K)

size_t **esp_himem_get_phys_size**(void)

Get total amount of memory under control of himem API.

Return Amount of memory, in bytes

size_t **esp_himem_get_free_size**(void)

Get free amount of memory under control of himem API.

Return Amount of free memory, in bytes

size_t **esp_himem_reserved_area_size**(void)

Get amount of SPI memory address space needed for bankswitching.

Note This is also weakly defined in esp32/spiram.c and returns 0 there, so if no other function in this file is used, no memory is reserved.

Return Amount of reserved area, in bytes

Macros

`ESP_HIMEM_BLKSZ`

`ESP_HIMEM_MAPFLAG_RO`

Indicates that a mapping will only be read from. Note that this is unused for now.

Type Definitions

typedef struct esp_himem_ramdata_t ***esp_himem_handle_t**

typedef struct esp_himem_rangedata_t ***esp_himem_rangehandle_t**

2.7.14 Inter-Processor Call

Overview

Due to the dual core nature of the ESP32, there are instances where a certain function must be run in the context of a particular core (e.g. allocating ISR to an interrupt source of a particular core). The IPC (Inter-Processor Call) feature allows for the execution of functions on a particular CPU.

A given function can be executed on a particular core by calling *esp_ipc_call()* or *esp_ipc_call_blocking()*. IPC is implemented via two high priority FreeRTOS tasks pinned to each CPU known as the IPC Tasks. The two IPC Tasks remain inactive (blocked) until *esp_ipc_call()* or *esp_ipc_call_blocking()* is called. When an IPC Task of a particular core is unblocked, it will preempt the current running task on that core and execute a given function.

Usage

esp_ipc_call() unblocks the IPC task on a particular core to execute a given function. The task that calls *esp_ipc_call()* will be blocked until the IPC Task begins execution of the given function. *esp_ipc_call_blocking()* is similar but will block the calling task until the IPC Task has completed execution of the given function.

Functions executed by IPCs must be functions of type *void func(void *arg)*. To run more complex functions which require a larger stack, the IPC tasks' stack size can be configured by modifying *CONFIG_ESP_IPC_TASK_STACK_SIZE* in *menuconfig*. The IPC API is protected by a mutex hence simultaneous IPC calls are not possible.

Care should taken to avoid deadlock when writing functions to be executed by IPC, especially when attempting to take a mutex within the function.

API Reference

Header File

- esp_common/include/esp_ipc.h

Functions

esp_err_t **esp_ipc_call**(uint32_t *cpu_id*, esp_ipc_func_t *func*, void **arg*)

Execute a function on the given CPU.

Run a given function on a particular CPU. The given function must accept a void* argument and return void. The given function is run in the context of the IPC task of the CPU specified by the cpu_id parameter. The calling task will be blocked until the IPC task begins executing the given function. If another IPC call is ongoing, the calling task will block until the other IPC call completes. The stack size allocated for the IPC task can be configured in the "Inter-Processor Call (IPC) task stack size" setting in menuconfig. Increase this setting if the given function requires more stack than default.

Note In single-core mode, returns ESP_ERR_INVALID_ARG for cpu_id 1.

Return

- ESP_ERR_INVALID_ARG if cpu_id is invalid
- ESP_ERR_INVALID_STATE if the FreeRTOS scheduler is not running
- ESP_OK otherwise

Parameters

- `cpu_id`: CPU where the given function should be executed (0 or 1)
- `func`: Pointer to a function of type void func(void* arg) to be executed
- `arg`: Arbitrary argument of type void* to be passed into the function

esp_err_t **esp_ipc_call_blocking**(uint32_t *cpu_id*, esp_ipc_func_t *func*, void **arg*)

Execute a function on the given CPU and blocks until it completes.

Run a given function on a particular CPU. The given function must accept a void* argument and return void. The given function is run in the context of the IPC task of the CPU specified by the cpu_id parameter. The calling task will be blocked until the IPC task completes execution of the given function. If another IPC call is ongoing, the calling task will block until the other IPC call completes. The stack size allocated for the IPC task can be configured in the "Inter-Processor Call (IPC) task

stack size" setting in menuconfig. Increase this setting if the given function requires more stack than default.

Note In single-core mode, returns ESP_ERR_INVALID_ARG for cpu_id 1.

Return

- ESP_ERR_INVALID_ARG if cpu_id is invalid
- ESP_ERR_INVALID_STATE if the FreeRTOS scheduler is not running
- ESP_OK otherwise

Parameters

- `cpu_id`: CPU where the given function should be executed (0 or 1)
- `func`: Pointer to a function of type void func(void* arg) to be executed
- `arg`: Arbitrary argument of type void* to be passed into the function

2.7.15 Call function with external stack

Overview

A given function can be executed with a user allocated stack space which is independent of current task stack, this mechanism can be used to save stack space wasted by tasks which call a common function with intensive stack usage such as *printf*. The given function can be called inside the macro `ESP_EXECUTE_EXPRESSION_WITH_STACK()` it will redirect the target function to be executed using the space allocated by the user.

Usage

`ESP_EXECUTE_EXPRESSION_WITH_STACK()` takes three arguments, a mutex object allocated by the caller, which is used to protect if the same function shares its allocated stack, a pointer to the top of stack used to that fuction, and the function itself, note the function is passed exactly in the same way as do when you call it on a regular way.

The usage may looks like the code below:

```
//Let's suppose we wanting to call printf using a separated stack space
//allowing app to reduce its stack size.
void app_main()
{
    //Allocate a stack buffer, from heap or as a static form:
    portSTACK_TYPE *shared_stack = malloc(8192 * sizeof(portSTACK_TYPE));
    assert(shared_stack != NULL);
```

(下页继续)

(续上页)

```
    //Allocate a mutex to protect its usage:
    SemaphoreHandle_t printf_lock = xSemaphoreCreateMutex();
    assert(printf_lock != NULL);

    //Call the desired function using the macro helper:
    ESP_EXECUTE_EXPRESSION_WITH_STACK(printf_lock,
                                      shared_stack,
                                      printf("Executing this from external stack! \n"));
    vSemaphoreDelete(printf_lock);
    free(shared_stack);
}
```

API Reference

Header File

- esp_common/include/esp_expression_with_stack.h

Functions

StackType_t *`esp_switch_stack_setup`(StackType_t *_stack_, size_t _stack_size_)

Fill stack frame with CPU-specifics value before use.

Return New pointer to the top of stack

Note Application must not call this function directly

Parameters

- `stack`: Caller allocated stack pointer
- `stack_size`: Size of stack in bytes

void `esp_switch_stack_enter`(StackType_t *_stack_, uint32_t *_backup_stack_)

Changes CPU sp-register to use another stack space and save the previous one.

Note Application must not call this function directly

Parameters

- `stack`: Caller allocated stack pointer
- `backup_stack`: Pointer to a place to save the current stack

void **esp_switch_stack_exit**(uint32_t *backup_stack)

Restores the previous CPU sp-register.

Note Application must not call this function directly

Parameters

- `backup_stack`: Pointer to the place where stack was saved

Macros

ESP_EXECUTE_EXPRESSION_WITH_STACK(lock, stack, stack_size, expression)

Executes a 1-line expression with a application alocated stack.

Note if either lock, stack or stack size is invalid, the expression will be called using the current stack.

Parameters

- `lock`: Mutex object to protect in case of shared stack
- `stack`: Pointer to user alocated stack
- `stack_size`: Size of current stack in bytes
- `expression`: Expression or function to be executed using the stack

2.7.16 Interrupt allocation

Overview

The ESP32 has two cores, with 32 interrupts each. Each interrupt has a certain priority level, most (but not all) interrupts are connected to the interrupt mux. Because there are more interrupt sources than interrupts, sometimes it makes sense to share an interrupt in multiple drivers. The esp_intr_alloc abstraction exists to hide all these implementation details.

A driver can allocate an interrupt for a certain peripheral by calling esp_intr_alloc (or esp_intr_alloc_sintrstatus). It can use the flags passed to this function to set the type of interrupt allocated, specifying a specific level or trigger method. The interrupt allocation code will then find an applicable interrupt, use the interrupt mux to hook it up to the peripheral, and install the given interrupt handler and ISR to it.

This code has two different types of interrupts it handles differently: Shared interrupts and non-shared interrupts. The simplest of the two are non-shared interrupts: a separate interrupt is allocated per esp_intr_alloc call and this interrupt is solely used for the peripheral attached to it, with only one ISR that will get called. Shared interrupts can have multiple peripherals triggering it, with multiple ISRs being called when one of the peripherals attached signals an interrupt. Thus, ISRs that are intended for shared interrupts should check the interrupt status of the peripheral they service in order to see if any action is required.

Non-shared interrupts can be either level- or edge-triggered. Shared interrupts can only be level interrupts (because of the chance of missed interrupts when edge interrupts are used.) (The logic behind this: DevA and DevB share an int. DevB signals an int. Int line goes high. ISR handler calls code for DevA -> does nothing. ISR handler calls code for DevB, but while doing that, DevA signals an int. ISR DevB is done, clears int for DevB, exits interrupt code. Now an interrupt for DevA is still pending, but because the int line never went low (DevA kept it high even when the int for DevB was cleared) the interrupt is never serviced.)

Multicore issues

Peripherals that can generate interrupts can be divided in two types:

- External peripherals, within the ESP32 but outside the Xtensa cores themselves. Most ESP32 peripherals are of this type.
- Internal peripherals, part of the Xtensa CPU cores themselves.

Interrupt handling differs slightly between these two types of peripherals.

Internal peripheral interrupts

Each Xtensa CPU core has its own set of six internal peripherals:

- Three timer comparators
- A performance monitor
- Two software interrupts.

Internal interrupt sources are defined in esp_intr_alloc.h as `ETS_INTERNAL_*_INTR_SOURCE`.

These peripherals can only be configured from the core they are associated with. When generating an interrupt, the interrupt they generate is hard-wired to their associated core; it' s not possible to have e.g. an internal timer comparator of one core generate an interrupt on another core. That is why these sources can only be managed using a task running on that specific core. Internal interrupt sources are still allocatable using esp_intr_alloc as normal, but they cannot be shared and will always have a fixed interrupt level (namely, the one associated in hardware with the peripheral).

External Peripheral Interrupts

The remaining interrupt sources are from external peripherals. These are defined in soc/soc.h as `ETS_*_INTR_SOURCE`.

Non-internal interrupt slots in both CPU cores are wired to an interrupt multiplexer, which can be used to route any external interrupt source to any of these interrupt slots.

- Allocating an external interrupt will always allocate it on the core that does the allocation.
- Freeing an external interrupt must always happen on the same core it was allocated on.

- Disabling and enabling external interrupts from another core is allowed.
- Multiple external interrupt sources can share an interrupt slot by passing `ESP_INTR_FLAG_SHARED` as a flag to esp_intr_alloc().

Care should be taken when calling esp_intr_alloc() from a task which is not pinned to a core. During task switching, these tasks can migrate between cores. Therefore it is impossible to tell which CPU the interrupt is allocated on, which makes it difficult to free the interrupt handle and may also cause debugging difficulties. It is advised to use xTaskCreatePinnedToCore() with a specific CoreID argument to create tasks that will allocate interrupts. In the case of internal interrupt sources, this is required.

IRAM-Safe Interrupt Handlers

The `ESP_INTR_FLAG_IRAM` flag registers an interrupt handler that always runs from IRAM (and reads all its data from DRAM), and therefore does not need to be disabled during flash erase and write operations.

This is useful for interrupts which need a guaranteed minimum execution latency, as flash write and erase operations can be slow (erases can take tens or hundreds of milliseconds to complete).

It can also be useful to keep an interrupt handler in IRAM if it is called very frequently, to avoid flash cache misses.

Refer to the *SPI flash API documentation* for more details.

Multiple Handlers Sharing A Source

Several handlers can be assigned to a same source, given that all handlers are allocated using the `ESP_INTR_FLAG_SHARED` flag. They' ll be all allocated to the interrupt, which the source is attached to, and called sequentially when the source is active. The handlers can be disabled and freed individually. The source is attached to the interrupt (enabled), if one or more handlers are enabled, otherwise detached. A handler will never be called when disabled, while **its source may still be triggered** if any one of its handler enabled.

Sources attached to non-shared interrupt do not support this feature.

Though the framework support this feature, you have to use it *very carefully*. There usually exist 2 ways to stop a interrupt from being triggered: *disable the source* or *mask peripheral interrupt status*. IDF only handles the enabling and disabling of the source itself, leaving status and mask bits to be handled by users. **Status bits should always be masked before the handler responsible for it is disabled, or the status should be handled in other enabled interrupt properly**. You may leave some status bits unhandled if you just disable one of all the handlers without masking the status bits, which causes the interrupt to trigger infinitely resulting in a system crash.

API Reference

Header File

- esp32/include/esp_intr_alloc.h

Functions

esp_err_t **esp_intr_mark_shared**(int *intno*, int *cpu*, bool *is_in_iram*)

Mark an interrupt as a shared interrupt.

This will mark a certain interrupt on the specified CPU as an interrupt that can be used to hook shared interrupt handlers to.

Return ESP_ERR_INVALID_ARG if cpu or intno is invalid ESP_OK otherwise

Parameters

- `intno`: The number of the interrupt (0-31)
- `cpu`: CPU on which the interrupt should be marked as shared (0 or 1)
- `is_in_iram`: Shared interrupt is for handlers that reside in IRAM and the int can be left enabled while the flash cache is disabled.

esp_err_t **esp_intr_reserve**(int *intno*, int *cpu*)

Reserve an interrupt to be used outside of this framework.

This will mark a certain interrupt on the specified CPU as reserved, not to be allocated for any reason.

Return ESP_ERR_INVALID_ARG if cpu or intno is invalid ESP_OK otherwise

Parameters

- `intno`: The number of the interrupt (0-31)
- `cpu`: CPU on which the interrupt should be marked as shared (0 or 1)

esp_err_t **esp_intr_alloc**(int *source*, int *flags*, *intr_handler_t handler*, void **arg*, *intr_handle_t* **ret_handle*)

Allocate an interrupt with the given parameters.

This finds an interrupt that matches the restrictions as given in the flags parameter, maps the given interrupt source to it and hooks up the given interrupt handler (with optional argument) as well. If needed, it can return a handle for the interrupt as well.

The interrupt will always be allocated on the core that runs this function.

If ESP_INTR_FLAG_IRAM flag is used, and handler address is not in IRAM or RTC_FAST_MEM, then ESP_ERR_INVALID_ARG is returned.

Return ESP_ERR_INVALID_ARG if the combination of arguments is invalid. ESP_ERR_NOT_FOUND No free interrupt found with the specified flags ESP_OK otherwise

Parameters

- `source`: The interrupt source. One of the ETS_*_INTR_SOURCE interrupt mux sources, as defined in soc/soc.h, or one of the internal ETS_INTERNAL_*_INTR_SOURCE sources as defined in this header.
- `flags`: An ORred mask of the ESP_INTR_FLAG_* defines. These restrict the choice of interrupts that this routine can choose from. If this value is 0, it will default to allocating a non-shared interrupt of level 1, 2 or 3. If this is ESP_INTR_FLAG_SHARED, it will allocate a shared interrupt of level 1. Setting ESP_INTR_FLAG_INTRDISABLED will return from this function with the interrupt disabled.
- `handler`: The interrupt handler. Must be NULL when an interrupt of level >3 is requested, because these types of interrupts aren't C-callable.
- `arg`: Optional argument for passed to the interrupt handler
- `ret_handle`: Pointer to an intr_handle_t to store a handle that can later be used to request details or free the interrupt. Can be NULL if no handle is required.

esp_err_t **esp_intr_alloc_intrstatus**(int *source*, int *flags*, uint32_t *intrstatusreg*, uint32_t *intrstatusmask*, *intr_handler_t* *handler*, void *arg, *intr_handle_t* *ret_handle)

Allocate an interrupt with the given parameters.

This essentially does the same as esp_intr_alloc, but allows specifying a register and mask combo. For shared interrupts, the handler is only called if a read from the specified register, ANDed with the mask, returns non-zero. By passing an interrupt status register address and a fitting mask, this can be used to accelerate interrupt handling in the case a shared interrupt is triggered; by checking the interrupt statuses first, the code can decide which ISRs can be skipped

Return ESP_ERR_INVALID_ARG if the combination of arguments is invalid. ESP_ERR_NOT_FOUND No free interrupt found with the specified flags ESP_OK otherwise

Parameters

- `source`: The interrupt source. One of the ETS_*_INTR_SOURCE interrupt mux sources, as defined in soc/soc.h, or one of the internal ETS_INTERNAL_*_INTR_SOURCE sources as defined in this header.
- `flags`: An ORred mask of the ESP_INTR_FLAG_* defines. These restrict the choice of interrupts that this routine can choose from. If this value is 0, it will default to allocating a non-shared interrupt of level 1, 2 or 3. If this is ESP_INTR_FLAG_SHARED, it will

allocate a shared interrupt of level 1. Setting ESP_INTR_FLAG_INTRDISABLED will return from this function with the interrupt disabled.

- `intrstatusreg`: The address of an interrupt status register
- `intrstatusmask`: A mask. If a read of address intrstatusreg has any of the bits that are 1 in the mask set, the ISR will be called. If not, it will be skipped.
- `handler`: The interrupt handler. Must be NULL when an interrupt of level >3 is requested, because these types of interrupts aren't C-callable.
- `arg`: Optional argument for passed to the interrupt handler
- `ret_handle`: Pointer to an intr_handle_t to store a handle that can later be used to request details or free the interrupt. Can be NULL if no handle is required.

esp_err_t **esp_intr_free**(*intr_handle_t handle*)

Disable and free an interrupt.

Use an interrupt handle to disable the interrupt and release the resources associated with it. If the current core is not the core that registered this interrupt, this routine will be assigned to the core that allocated this interrupt, blocking and waiting until the resource is successfully released.

Note When the handler shares its source with other handlers, the interrupt status bits it's responsible for should be managed properly before freeing it. see `esp_intr_disable` for more details. Please do not call this function in `esp_ipc_call_blocking`.

Return ESP_ERR_INVALID_ARG the handle is NULL ESP_FAIL failed to release this handle ESP_OK otherwise

Parameters

- `handle`: The handle, as obtained by esp_intr_alloc or esp_intr_alloc_intrstatus

int **esp_intr_get_cpu**(*intr_handle_t handle*)

Get CPU number an interrupt is tied to.

Return The core number where the interrupt is allocated

Parameters

- `handle`: The handle, as obtained by esp_intr_alloc or esp_intr_alloc_intrstatus

int **esp_intr_get_intno**(*intr_handle_t handle*)

Get the allocated interrupt for a certain handle.

Return The interrupt number

Parameters

- `handle`: The handle, as obtained by esp_intr_alloc or esp_intr_alloc_intrstatus

esp_err_t **esp_intr_disable**(*intr_handle_t handle*)

Disable the interrupt associated with the handle.

Note

1. For local interrupts (ESP_INTERNAL_* sources), this function has to be called on the CPU the interrupt is allocated on. Other interrupts have no such restriction.
2. When several handlers sharing a same interrupt source, interrupt status bits, which are handled in the handler to be disabled, should be masked before the disabling, or handled in other enabled interrupts properly. Miss of interrupt status handling will cause infinite interrupt calls and finally system crash.

Return ESP_ERR_INVALID_ARG if the combination of arguments is invalid. ESP_OK otherwise

Parameters

- **handle**: The handle, as obtained by esp_intr_alloc or esp_intr_alloc_intrstatus

esp_err_t **esp_intr_enable**(*intr_handle_t handle*)

Enable the interrupt associated with the handle.

Note For local interrupts (ESP_INTERNAL_* sources), this function has to be called on the CPU the interrupt is allocated on. Other interrupts have no such restriction.

Return ESP_ERR_INVALID_ARG if the combination of arguments is invalid. ESP_OK otherwise

Parameters

- **handle**: The handle, as obtained by esp_intr_alloc or esp_intr_alloc_intrstatus

esp_err_t **esp_intr_set_in_iram**(*intr_handle_t handle*, bool *is_in_iram*)

Set the "in IRAM" status of the handler.

Note Does not work on shared interrupts.

Return ESP_ERR_INVALID_ARG if the combination of arguments is invalid. ESP_OK otherwise

Parameters

- **handle**: The handle, as obtained by esp_intr_alloc or esp_intr_alloc_intrstatus
- **is_in_iram**: Whether the handler associated with this handle resides in IRAM. Handlers residing in IRAM can be called when cache is disabled.

void **esp_intr_noniram_disable**(void)

Disable interrupts that aren't specifically marked as running from IRAM.

void **esp_intr_noniram_enable**(void)

Re-enable interrupts disabled by esp_intr_noniram_disable.

Macros

ESP_INTR_FLAG_LEVEL1

Interrupt allocation flags.

These flags can be used to specify which interrupt qualities the code calling esp_intr_alloc* needs.Accept a Level 1 interrupt vector (lowest priority)

ESP_INTR_FLAG_LEVEL2

Accept a Level 2 interrupt vector.

ESP_INTR_FLAG_LEVEL3

Accept a Level 3 interrupt vector.

ESP_INTR_FLAG_LEVEL4

Accept a Level 4 interrupt vector.

ESP_INTR_FLAG_LEVEL5

Accept a Level 5 interrupt vector.

ESP_INTR_FLAG_LEVEL6

Accept a Level 6 interrupt vector.

ESP_INTR_FLAG_NMI

Accept a Level 7 interrupt vector (highest priority)

ESP_INTR_FLAG_SHARED

Interrupt can be shared between ISRs.

ESP_INTR_FLAG_EDGE

Edge-triggered interrupt.

ESP_INTR_FLAG_IRAM

ISR can be called if cache is disabled.

ESP_INTR_FLAG_INTRDISABLED

Return with this interrupt disabled.

ESP_INTR_FLAG_LOWMED

Low and medium prio interrupts. These can be handled in C.

ESP_INTR_FLAG_HIGH

High level interrupts. Need to be handled in assembly.

ESP_INTR_FLAG_LEVELMASK

Mask for all level flags.

ETS_INTERNAL_TIMER0_INTR_SOURCE

Xtensa timer 0 interrupt source.

The esp_intr_alloc* functions can allocate an int for all ETS_*_INTR_SOURCE interrupt sources that are routed through the interrupt mux. Apart from these sources, each core also has some internal sources that do not pass through the interrupt mux. To allocate an interrupt for these sources, pass these pseudo-sources to the functions.

`ETS_INTERNAL_TIMER1_INTR_SOURCE`
Xtensa timer 1 interrupt source.

`ETS_INTERNAL_TIMER2_INTR_SOURCE`
Xtensa timer 2 interrupt source.

`ETS_INTERNAL_SW0_INTR_SOURCE`
Software int source 1.

`ETS_INTERNAL_SW1_INTR_SOURCE`
Software int source 2.

`ETS_INTERNAL_PROFILING_INTR_SOURCE`
Int source for profiling.

`ETS_INTERNAL_INTR_SOURCE_OFF`

`ESP_INTR_ENABLE`(inum)

`ESP_INTR_DISABLE`(inum)

Type Definitions

`typedef` void (*`intr_handler_t`)(void *arg)

`typedef struct` *intr_handle_data_t* `intr_handle_data_t`

`typedef` *intr_handle_data_t* *`intr_handle_t`

2.7.17 Logging library

Overview

The logging library provides two ways for setting log verbosity:

- **At compile time**: in menuconfig, set the verbosity level using the option `CONFIG_LOG_DEFAULT_LEVEL`. All logging statements for verbosity levels higher than `CONFIG_LOG_DEFAULT_LEVEL` will be removed by the preprocessor.
- **At runtime**: all logs for verbosity levels lower than `CONFIG_LOG_DEFAULT_LEVEL` are enabled by default. The function *esp_log_level_set()* can be used to set a logging level on a per module basis. Modules are identified by their tags, which are human-readable ASCII zero-terminated strings.

There are the following verbosity levels:

- Error (lowest)
- Warning
- Info
- Debug
- Verbose (highest)

注解: The function *esp_log_level_set()* cannot set logging levels higher than specified by CONFIG_LOG_DEFAULT_LEVEL. To increase log level for a specific file at compile time, use the macro *LOG_LOCAL_LEVEL* (see the details below).

How to use this library

In each C file that uses logging functionality, define the TAG variable as shown below:

```
static const char* TAG = "MyModule";
```

Then use one of logging macros to produce output, e.g:

```
ESP_LOGW(TAG, "Baud rate error %.1f%%. Requested: %d baud, actual: %d baud", error * 100,
↪ baud_req, baud_real);
```

Several macros are available for different verbosity levels:

- ESP_LOGE - error (lowest)
- ESP_LOGW - warning
- ESP_LOGI - info
- ESP_LOGD - debug
- ESP_LOGV - verbose (highest)

Additionally, there are ESP_EARLY_LOGx versions for each of these macros, e.g., *ESP_EARLY_LOGE*. These versions have to be used explicitly in the early startup code only, before heap allocator and syscalls have been initialized. Normal ESP_LOGx macros can also be used while compiling the bootloader, but they will fall back to the same implementation as ESP_EARLY_LOGx macros.

To override default verbosity level at file or component scope, define the LOG_LOCAL_LEVEL macro.

At file scope, define it before including esp_log.h, e.g.:

```
#define LOG_LOCAL_LEVEL ESP_LOG_VERBOSE
#include "esp_log.h"
```

At component scope, define it in the component makefile:

```
CFLAGS += -D LOG_LOCAL_LEVEL=ESP_LOG_DEBUG
```

To configure logging output per module at runtime, add calls to the function *esp_log_level_set()* as follows:

```
esp_log_level_set("*", ESP_LOG_ERROR);        // set all components to ERROR level
esp_log_level_set("wifi", ESP_LOG_WARN);      // enable WARN logs from WiFi stack
esp_log_level_set("dhcpc", ESP_LOG_INFO);     // enable INFO logs from DHCP client
```

Logging to Host via JTAG

By default, the logging library uses the vprintf-like function to write formatted output to the dedicated UART. By calling a simple API, all log output may be routed to JTAG instead, making logging several times faster. For details, please refer to Section 记录日志到主机.

Application Example

The logging library is commonly used by most esp-idf components and examples. For demonstration of log functionality, check ESP-IDF' s examples directory. The most revelant examples that deal with logging are the following:

- system/ota
- storage/sd_card
- protocols/https_request

API Reference

Header File

- log/include/esp_log.h

Functions

void **esp_log_level_set**(**const** char *_tag_, _esp_log_level_t_ _level_)

Set log level for given tag.

If logging for given component has already been enabled, changes previous setting.

Note that this function can not raise log level above the level set using CONFIG_LOG_DEFAULT_LEVEL setting in menuconfig.

To raise log level above the default one for a given file, define LOG_LOCAL_LEVEL to one of the ESP_LOG_* values, before including esp_log.h in this file.

Parameters

- `tag`: Tag of the log entries to enable. Must be a non-NULL zero terminated string. Value "*" resets log level for all tags to the given value.
- `level`: Selects log level to enable. Only logs at this and lower verbosity levels will be shown.

vprintf_like_t **esp_log_set_vprintf**(*vprintf_like_t func*)

Set function used to output log entries.

By default, log output goes to UART0. This function can be used to redirect log output to some other destination, such as file or network. Returns the original log handler, which may be necessary to return output to the previous destination.

Return func old Function used for output.

Parameters

- `func`: new Function used for output. Must have same signature as vprintf.

uint32_t **esp_log_timestamp**(void)

Function which returns timestamp to be used in log output.

This function is used in expansion of ESP_LOGx macros. In the 2nd stage bootloader, and at early application startup stage this function uses CPU cycle counter as time source. Later when FreeRTOS scheduler start running, it switches to FreeRTOS tick count.

For now, we ignore millisecond counter overflow.

Return timestamp, in milliseconds

char ***esp_log_system_timestamp**(void)

Function which returns system timestamp to be used in log output.

This function is used in expansion of ESP_LOGx macros to print the system time as "HH:MM:SS.sss" . The system time is initialized to 0 on startup, this can be set to the correct time with an SNTP sync, or manually with standard POSIX time functions.

Currently this will not get used in logging from binary blobs (i.e WiFi & Bluetooth libraries), these will still print the RTOS tick time.

Return timestamp, in "HH:MM:SS.sss"

uint32_t **esp_log_early_timestamp**(void)

Function which returns timestamp to be used in log output.

This function uses HW cycle counter and does not depend on OS, so it can be safely used after application crash.

Return timestamp, in milliseconds

void **esp_log_write**(*esp_log_level_t level*, **const** char ***tag*, **const** char ***format*, ...)

Write message into the log.

This function is not intended to be used directly. Instead, use one of ESP_LOGE, ESP_LOGW, ESP_LOGI, ESP_LOGD, ESP_LOGV macros.

This function or these macros should not be used from an interrupt.

void **esp_log_writev**(*esp_log_level_t level*, **const** char ***tag*, **const** char ***format*, va_list *args*)

Write message into the log, va_list variant.

This function is provided to ease integration toward other logging framework, so that esp_log can be used as a log sink.

See esp_log_write()

Macros

ESP_LOG_BUFFER_HEX_LEVEL(tag, buffer, buff_len, level)

Log a buffer of hex bytes at specified level, separated into 16 bytes each line.

Parameters

- `tag`: description tag
- `buffer`: Pointer to the buffer array
- `buff_len`: length of buffer in bytes
- `level`: level of the log

ESP_LOG_BUFFER_CHAR_LEVEL(tag, buffer, buff_len, level)

Log a buffer of characters at specified level, separated into 16 bytes each line. Buffer should contain only printable characters.

Parameters

- `tag`: description tag
- `buffer`: Pointer to the buffer array
- `buff_len`: length of buffer in bytes
- `level`: level of the log

ESP_LOG_BUFFER_HEXDUMP(tag, buffer, buff_len, level)

Dump a buffer to the log at specified level.

The dump log shows just like the one below:

```
W (195) log_example: 0x3ffb4280   45 53 50 33 32 20 69 73  20 67 72 65 61 74 2c 20␣
↪ |ESP32 is great, |
W (195) log_example: 0x3ffb4290   77 6f 72 6b 69 6e 67 20  61 6c 6f 6e 67 20 77 69␣
↪ |working along wi|
W (205) log_example: 0x3ffb42a0   74 68 20 74 68 65 20 49  44 46 2e 00            ␣
↪ |th the IDF..|
```

It is highly recommend to use terminals with over 102 text width.

Parameters

- tag: description tag
- buffer: Pointer to the buffer array
- buff_len: length of buffer in bytes
- level: level of the log

ESP_LOG_BUFFER_HEX(tag, buffer, buff_len)

Log a buffer of hex bytes at Info level.

See esp_log_buffer_hex_level

Parameters

- tag: description tag
- buffer: Pointer to the buffer array
- buff_len: length of buffer in bytes

ESP_LOG_BUFFER_CHAR(tag, buffer, buff_len)

Log a buffer of characters at Info level. Buffer should contain only printable characters.

See esp_log_buffer_char_level

Parameters

- tag: description tag
- buffer: Pointer to the buffer array
- buff_len: length of buffer in bytes

ESP_EARLY_LOGE(tag, format, ...)
macro to output logs in startup code, before heap allocator and syscalls have been initialized. log at ESP_LOG_ERROR level.

See printf,ESP_LOGE

ESP_EARLY_LOGW(tag, format, ...)
macro to output logs in startup code at ESP_LOG_WARN level.

See ESP_EARLY_LOGE,ESP_LOGE, printf

ESP_EARLY_LOGI(tag, format, ...)
macro to output logs in startup code at ESP_LOG_INFO level.

See ESP_EARLY_LOGE,ESP_LOGE, printf

ESP_EARLY_LOGD(tag, format, ...)
macro to output logs in startup code at ESP_LOG_DEBUG level.

See ESP_EARLY_LOGE,ESP_LOGE, printf

ESP_EARLY_LOGV(tag, format, ...)
macro to output logs in startup code at ESP_LOG_VERBOSE level.

See ESP_EARLY_LOGE,ESP_LOGE, printf

ESP_LOG_EARLY_IMPL(tag, format, log_level, log_tag_letter, ...)

ESP_LOGE(tag, format, ...)

ESP_LOGW(tag, format, ...)

ESP_LOGI(tag, format, ...)

ESP_LOGD(tag, format, ...)

ESP_LOGV(tag, format, ...)

ESP_LOG_LEVEL_LOCAL(level, tag, format, ...)
runtime macro to output logs at a specified level.

See printf runtime macro to output logs at a specified level. Also check the level with LOG_LOCAL_LEVEL.

See printf, ESP_LOG_LEVEL

Parameters

- tag: tag of the log, which can be used to change the log level by esp_log_level_set at runtime.

- `level`: level of the output log.
- `format`: format of the output log. see `printf`
- `...`: variables to be replaced into the log. see `printf`

Type Definitions

`typedef` int (*`vprintf_like_t`)(`const` char *, va_list)

Enumerations

`enum esp_log_level_t`

Log level.

Values:

`ESP_LOG_NONE`

No log output

`ESP_LOG_ERROR`

Critical errors, software module can not recover on its own

`ESP_LOG_WARN`

Error conditions from which recovery measures have been taken

`ESP_LOG_INFO`

Information messages which describe normal flow of events

`ESP_LOG_DEBUG`

Extra information which is not necessary for normal use (values, pointers, sizes, etc).

`ESP_LOG_VERBOSE`

Bigger chunks of debugging information, or frequent messages which can potentially flood the output.

2.7.18 Miscellaneous System APIs

Software reset

To perform software reset of the chip, *esp_restart()* function is provided. When the function is called, execution of the program will stop, both CPUs will be reset, application will be loaded by the bootloader and started again.

Additionally, *esp_register_shutdown_handler()* function is provided to register a routine which needs to be called prior to restart (when done by *esp_restart()*). This is similar to the functionality of `atexit` POSIX function.

Reset reason

ESP-IDF application can be started or restarted due to a variety of reasons. To get the last reset reason, call *esp_reset_reason()* function. See description of *esp_reset_reason_t* for the list of possible reset reasons.

Heap memory

Two heap memory related functions are provided:

- *esp_get_free_heap_size()* returns the current size of free heap memory
- *esp_get_minimum_free_heap_size()* returns the minimum size of free heap memory that was available during program execution.

Note that ESP-IDF supports multiple heaps with different capabilities. Functions mentioned in this section return the size of heap memory which can be allocated using `malloc` family of functions. For further information about heap memory see *Heap Memory Allocation*.

Random number generation

ESP32 contains a hardware random number generator, values from it can be obtained using *esp_random()*.

When Wi-Fi or Bluetooth are enabled, numbers returned by hardware random number generator (RNG) can be considered true random numbers. Without Wi-Fi or Bluetooth enabled, hardware RNG is a pseudo-random number generator. At startup, ESP-IDF bootloader seeds the hardware RNG with entropy, but care must be taken when reading random values between the start of `app_main` and initialization of Wi-Fi or Bluetooth drivers.

MAC Address

These APIs allow querying and customizing MAC addresses used by Wi-Fi, Bluetooth, and Ethernet drivers.

ESP32 has up to 4 network interfaces: Wi-Fi station, Wi-Fi AP, Ethernet, and Bluetooth. Each of these interfaces needs to have a MAC address assigned to it. In ESP-IDF these addresses are calculated from *Base MAC address*. Base MAC address can be initialized with factory-programmed value from EFUSE, or with a user-defined value. In addition to setting the base MAC address, applications can specify the way in which MAC addresses are allocated to devices. See *Number of universally administered MAC address* section for more details.

Interface	MAC address (4 universally administered)	MAC address (2 universally administered)
Wi-Fi Station	base_mac	base_mac
Wi-Fi SoftAP	base_mac, +1 to the last octet	base_mac, first octet randomized
Bluetooth	base_mac, +2 to the last octet	base_mac, +1 to the last octet
Ethernet	base_mac, +3 to the last octet	base_mac, +1 to the last octet, first octet randomized

Base MAC address

Wi-Fi, Bluetooth, and Ethernet drivers use *esp_read_mac()* function to get MAC address for a specific interface.

By default, this function will use MAC address factory programmed in BLK0 of EFUSE as the base MAC address. MAC addresses of each interface will be calculated according to the table above.

Applications which don't use MAC address factory programmed into BLK0 of EFUSE can modify base MAC address used by *esp_read_mac()* using a call to *esp_base_mac_addr_set()*. Custom value of MAC address can come from application defined storage, such as Flash, NVS, etc. Note that the call to *esp_base_mac_addr_set()* needs to happen before network protocol stacks are initialized, for example, early in `app_main`.

Custom MAC address in BLK3 of EFUSE

To facilitate usage of custom MAC addresses, ESP-IDF provides *esp_efuse_mac_get_custom()* function, which loads MAC address from BLK3 of EFUSE. This function assumes that custom MAC address is stored in BLK3 of EFUSE (EFUSE_BLK3_RDATA0, EFUSE_BLK3_RDATA1, EFUSE_BLK3_RDATA2, EFUSE_BLK3_RDATA3, EFUSE_BLK3_RDATA4, EFUSE_BLK3_RDATA5 registers) in the following format:

Field	# of bits	Range of bits	Notes
Version	8	191:184	0: invalid, others —valid
Reserved	128	183:56	
MAC address	48	55:8	
MAC address CRC	8	7:0	CRC-8-CCITT, polynomial 0x07

Once MAC address has been obtained using *esp_efuse_mac_get_custom()*, call *esp_base_mac_addr_set()* to set this MAC address as base MAC address.

Number of universally administered MAC address

Serveral MAC addresses (universally administered by IEEE) are uniquely assigned to the networking interfaces (Wi-Fi/BT/Ethernet). The final octet of each universally administered MAC address increases by one. Only the first one of them (which is called base MAC address) is stored in EFUSE or external storage, the others are generated from it. Here, 'generate' means adding 0, 1, 2 and 3 (respectively) to the final octet of the base MAC address.

If the universally administered MAC addresses are not enough for all of the networking interfaces, locally administered MAC addresses which are derived from universally administered MAC addresses are assigned to the rest of networking interfaces.

See this article for the definition of local and universally administered MAC addresses.

The number of universally administered MAC address can be configured using *CONFIG_ESP32_UNIVERSAL_MAC_ADDRESSES*.

If the number of universal MAC addresses is two, only two interfaces (Wi-Fi Station and Bluetooth) receive a universally administered MAC address. These are generated sequentially by adding 0 and 1 (respectively) to the base MAC address. The remaining two interfaces (Wi-Fi SoftAP and Ethernet) receive local MAC addresses. These are derived from the universal Wi-Fi station and Bluetooth MAC addresses, respectively.

If the number of universal MAC addresses is four, all four interfaces (Wi-Fi Station, Wi-Fi SoftAP, Bluetooth and Ethernet) receive a universally administered MAC address. These are generated sequentially by adding 0, 1, 2 and 3 (respectively) to the final octet of the base MAC address.

When using the default (Espressif-assigned) base MAC address, either setting can be used. When using a custom universal MAC address range, the correct setting will depend on the allocation of MAC addresses in this range (either 2 or 4 per device.)

Chip version

esp_chip_info() function fills *esp_chip_info_t* structure with information about the chip. This includes the chip revision, number of CPU cores, and a bit mask of features enabled in the chip.

SDK version

esp_get_idf_version() returns a string describing the IDF version which was used to compile the application. This is the same value as the one available through `IDF_VER` variable of the build system. The version string generally has the format of `git describe` output.

To get the version at build time, additional version macros are provided. They can be used to enable or disable parts of the program depending on IDF version.

- *ESP_IDF_VERSION_MAJOR*, *ESP_IDF_VERSION_MINOR*, *ESP_IDF_VERSION_PATCH* are defined to integers representing major, minor, and patch version.

- *ESP_IDF_VERSION_VAL* and *ESP_IDF_VERSION* can be used when implementing version checks:

```
#include "esp_idf_version.h"

#if ESP_IDF_VERSION >= ESP_IDF_VERSION_VAL(4, 0, 0)
    // enable functionality present in IDF v4.0
#endif
```

App version

Application version is stored in *esp_app_desc_t* structure. It is located in DROM sector and has a fixed offset from the beginning of the binary file. The structure is located after *esp_image_header_t* and *esp_image_segment_header_t* structures. The field version has string type and max length 32 chars.

To set version in your project manually you need to set `PROJECT_VER` variable in your project CMakeLists.txt/Makefile:

- In application CMakeLists.txt put `set(PROJECT_VER "0.1.0.1")` before including `project.cmake`.

(For legacy GNU Make build system: in application Makefile put `PROJECT_VER = "0.1.0.1"` before including `project.mk`.)

If `PROJECT_VER` variable is not set in the project then it will be retrieved from either `$(PROJECT_PATH)/version.txt` file (if present) else using git command `git describe`. If neither is available then `PROJECT_VER` will be set to "1". Application can make use of this by calling *esp_ota_get_app_description()* or *esp_ota_get_partition_description()* functions.

API Reference

Header File

- esp_common/include/esp_system.h

Functions

esp_err_t **esp_register_shutdown_handler**(*shutdown_handler_t handle*)

Register shutdown handler.

This function allows you to register a handler that gets invoked before the application is restarted using esp_restart function.

Return

- ESP_OK on success
- ESP_ERR_INVALID_STATE if the handler has already been registered

- ESP_ERR_NO_MEM if no more shutdown handler slots are available

Parameters

- `handle`: function to execute on restart

esp_err_t **esp_unregister_shutdown_handler**(*shutdown_handler_t handle*)

Unregister shutdown handler.

This function allows you to unregister a handler which was previously registered using esp_register_shutdown_handler function.

- ESP_OK on success
- ESP_ERR_INVALID_STATE if the given handler hasn't been registered before

void **esp_restart**(void)

Restart PRO and APP CPUs.

This function can be called both from PRO and APP CPUs. After successful restart, CPU reset reason will be SW_CPU_RESET. Peripherals (except for WiFi, BT, UART0, SPI1, and legacy timers) are not reset. This function does not return.

esp_reset_reason_t **esp_reset_reason**(void)

Get reason of last reset.

Return See description of esp_reset_reason_t for explanation of each value.

uint32_t **esp_get_free_heap_size**(void)

Get the size of available heap.

Note that the returned value may be larger than the maximum contiguous block which can be allocated.

Return Available heap size, in bytes.

uint32_t **esp_get_minimum_free_heap_size**(void)

Get the minimum heap that has ever been available.

Return Minimum free heap ever available

uint32_t **esp_random**(void)

Get one random 32-bit word from hardware RNG.

The hardware RNG is fully functional whenever an RF subsystem is running (ie Bluetooth or WiFi is enabled). For random values, call this function after WiFi or Bluetooth are started.

If the RF subsystem is not used by the program, the function bootloader_random_enable() can be called to enable an entropy source. bootloader_random_disable() must be called before RF subsystem or I2S peripheral are used. See these functions' documentation for more details.

Any time the app is running without an RF subsystem (or bootloader_random) enabled, RNG hardware should be considered a PRNG. A very small amount of entropy is available due to pre-seeding while the IDF bootloader is running, but this should not be relied upon for any use.

Return Random value between 0 and UINT32_MAX

void **esp_fill_random**(void *_buf_, size_t _len_)

Fill a buffer with random bytes from hardware RNG.

Note This function has the same restrictions regarding available entropy as esp_random()

Parameters

- `buf`: Pointer to buffer to fill with random numbers.
- `len`: Length of buffer in bytes

esp_err_t **esp_base_mac_addr_set**(**const** uint8_t *_mac_)

Set base MAC address with the MAC address which is stored in BLK3 of EFUSE or external storage e.g. flash and EEPROM.

Base MAC address is used to generate the MAC addresses used by the networking interfaces. If using base MAC address stored in BLK3 of EFUSE or external storage, call this API to set base MAC address with the MAC address which is stored in BLK3 of EFUSE or external storage before initializing WiFi/BT/Ethernet.

Note Base MAC must be a unicast MAC (least significant bit of first byte must be zero).

Note If not using a valid OUI, set the "locally administered" bit (bit value 0x02 in the first byte) to avoid collisions.

Return ESP_OK on success ESP_ERR_INVALID_ARG If mac is NULL or is not a unicast MAC

Parameters

- `mac`: base MAC address, length: 6 bytes.

esp_err_t **esp_base_mac_addr_get**(uint8_t *_mac_)

Return base MAC address which is set using esp_base_mac_addr_set.

Return ESP_OK on success ESP_ERR_INVALID_MAC base MAC address has not been set

Parameters

- `mac`: base MAC address, length: 6 bytes.

esp_err_t **esp_efuse_mac_get_custom**(uint8_t *_mac_)

Return base MAC address which was previously written to BLK3 of EFUSE.

Base MAC address is used to generate the MAC addresses used by the networking interfaces. This API returns the custom base MAC address which was previously written to BLK3 of EFUSE. Writing this EFUSE allows setting of a different (non-Espressif) base MAC address. It is also possible to store a custom base MAC address elsewhere, see esp_base_mac_addr_set() for details.

Return ESP_OK on success ESP_ERR_INVALID_VERSION An invalid MAC version field was read from BLK3 of EFUSE ESP_ERR_INVALID_CRC An invalid MAC CRC was read from BLK3 of EFUSE

Parameters

- `mac`: base MAC address, length: 6 bytes.

esp_err_t **esp_efuse_mac_get_default**(uint8_t *mac)

Return base MAC address which is factory-programmed by Espressif in BLK0 of EFUSE.

Return ESP_OK on success

Parameters

- `mac`: base MAC address, length: 6 bytes.

esp_err_t **esp_read_mac**(uint8_t *mac, *esp_mac_type_t type*)

Read base MAC address and set MAC address of the interface.

This function first get base MAC address using esp_base_mac_addr_get or reads base MAC address from BLK0 of EFUSE. Then set the MAC address of the interface including wifi station, wifi softap, bluetooth and ethernet.

Return ESP_OK on success

Parameters

- `mac`: MAC address of the interface, length: 6 bytes.
- `type`: type of MAC address, 0:wifi station, 1:wifi softap, 2:bluetooth, 3:ethernet.

esp_err_t **esp_derive_local_mac**(uint8_t *local_mac, **const** uint8_t *universal_mac)

Derive local MAC address from universal MAC address.

This function derives a local MAC address from an universal MAC address. A `definition of local vs universal MAC address can be found on Wikipedia <>`. In ESP32, universal MAC address is generated from base MAC address in EFUSE or other external storage. Local MAC address is derived from the universal MAC address.

Return ESP_OK on success

Parameters

- `local_mac`: Derived local MAC address, length: 6 bytes.

- `universal_mac`: Source universal MAC address, length: 6 bytes.

void **esp_chip_info**(*esp_chip_info_t* **out_info*)

Fill an *esp_chip_info_t* structure with information about the chip.

Parameters

- `out_info`: structure to be filled

Structures

struct esp_chip_info_t

The structure represents information about the chip.

Public Members

esp_chip_model_t **model**

chip model, one of esp_chip_model_t

uint32_t **features**

bit mask of CHIP_FEATURE_x feature flags

uint8_t **cores**

number of CPU cores

uint8_t **revision**

chip revision number

Macros

CHIP_FEATURE_EMB_FLASH

Chip has embedded flash memory.

CHIP_FEATURE_WIFI_BGN

Chip has 2.4GHz WiFi.

CHIP_FEATURE_BLE

Chip has Bluetooth LE.

CHIP_FEATURE_BT

Chip has Bluetooth Classic.

Type Definitions

typedef void (***shutdown_handler_t**)(void)

Shutdown handler type

Enumerations

enum esp_mac_type_t

Values:

ESP_MAC_WIFI_STA

ESP_MAC_WIFI_SOFTAP

ESP_MAC_BT

ESP_MAC_ETH

enum esp_reset_reason_t

Reset reasons.

Values:

ESP_RST_UNKNOWN

Reset reason can not be determined.

ESP_RST_POWERON

Reset due to power-on event.

ESP_RST_EXT

Reset by external pin (not applicable for ESP32)

ESP_RST_SW

Software reset via esp_restart.

ESP_RST_PANIC

Software reset due to exception/panic.

ESP_RST_INT_WDT

Reset (software or hardware) due to interrupt watchdog.

ESP_RST_TASK_WDT

Reset due to task watchdog.

ESP_RST_WDT

Reset due to other watchdogs.

ESP_RST_DEEPSLEEP

Reset after exiting deep sleep mode.

ESP_RST_BROWNOUT

Brownout reset (software or hardware)

ESP_RST_SDIO

Reset over SDIO.

enum esp_chip_model_t

Chip models.

Values:

`CHIP_ESP32` = 1
ESP32.

`CHIP_ESP32S2BETA` = 2
ESP32-S2 Beta.

Header File

- esp_common/include/esp_idf_version.h

Functions

`const` char *`esp_get_idf_version`(void)
Return full IDF version string, same as 'git describe' output.

Note If you are printing the ESP-IDF version in a log file or other information, this function provides more information than using the numerical version macros. For example, numerical version macros don't differentiate between development, pre-release and release versions, but the output of this function does.

Return constant string from IDF_VER

Macros

`ESP_IDF_VERSION_MAJOR`
Major version number (X.x.x)

`ESP_IDF_VERSION_MINOR`
Minor version number (x.X.x)

`ESP_IDF_VERSION_PATCH`
Patch version number (x.x.X)

`ESP_IDF_VERSION_VAL`(major, minor, patch)
Macro to convert IDF version number into an integer

To be used in comparisons, such as ESP_IDF_VERSION >= ESP_IDF_VERSION_VAL(4, 0, 0)

`ESP_IDF_VERSION`
Current IDF version, as an integer

To be used in comparisons, such as ESP_IDF_VERSION >= ESP_IDF_VERSION_VAL(4, 0, 0)

2.7.19 Over The Air Updates (OTA)

OTA Process Overview

The OTA update mechanism allows a device to update itself based on data received while the normal firmware is running (for example, over WiFi or Bluetooth.)

OTA requires configuring the *Partition Table* of the device with at least two "OTA app slot" partitions (ie *ota_0* and *ota_1*) and an "OTA Data Partition".

The OTA operation functions write a new app firmware image to whichever OTA app slot is not currently being used for booting. Once the image is verified, the OTA Data partition is updated to specify that this image should be used for the next boot.

OTA Data Partition

An OTA data partition (type `data`, subtype `ota`) must be included in the *Partition Table* of any project which uses the OTA functions.

For factory boot settings, the OTA data partition should contain no data (all bytes erased to 0xFF). In this case the esp-idf software bootloader will boot the factory app if it is present in the the partition table. If no factory app is included in the partition table, the first available OTA slot (usually `ota_0`) is booted.

After the first OTA update, the OTA data partition is updated to specify which OTA app slot partition should be booted next.

The OTA data partition is two flash sectors (0x2000 bytes) in size, to prevent problems if there is a power failure while it is being written. Sectors are independently erased and written with matching data, and if they disagree a counter field is used to determine which sector was written more recently.

App rollback

The main purpose of the application rollback is to keep the device working after the update. This feature allows you to roll back to the previous working application in case a new application has critical errors. When the rollback process is enabled and an OTA update provides a new version of the app, one of three things can happen:

- The application works fine, *esp_ota_mark_app_valid_cancel_rollback()* marks the running application with the state `ESP_OTA_IMG_VALID`. There are no restrictions on booting this application.
- The application has critical errors and further work is not possible, a rollback to the previous application is required, *esp_ota_mark_app_invalid_rollback_and_reboot()* marks the running application with the state `ESP_OTA_IMG_INVALID` and reset. This application will not be selected by the bootloader for boot and will boot the previously working application.
- If the *CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE* option is set, and a reset occurs without calling either function then the application is rolled back.

Note: The state is not written to the binary image of the application it is written to the `otadata` partition. The partition contains a `ota_seq` counter which is a pointer to the slot (ota_0, ota_1, ···) from which the application will be selected for boot.

App OTA State

States control the process of selecting a boot app:

States	Restriction of selecting a boot app in bootloader
ESP_OTA_IMG_VALID	No restriction. Will be selected.
ESP_OTA_IMG_UNDEFINED	No restriction. Will be selected.
ESP_OTA_IMG_INVALID	Will not be selected.
ESP_OTA_IMG_ABORTED	Will not be selected.
ESP_OTA_IMG_NEW	If *CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE* option is set it will be selected only once. In bootloader the state immediately changes to `ESP_OTA_IMG_PENDING_VERIFY`.
ESP_OTA_IMG_PENDING_VERIFY	If *CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE* option is set it will not be selected and the state will change to `ESP_OTA_IMG_ABORTED`.

If *CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE* option is not enabled (by default), then the use of the following functions *esp_ota_mark_app_valid_cancel_rollback()* and *esp_ota_mark_app_invalid_rollback_and_reboot()* are optional, and `ESP_OTA_IMG_NEW` and `ESP_OTA_IMG_PENDING_VERIFY` states are not used.

An option in Kconfig *CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE* allows you to track the first boot of a new application. In this case, the application must confirm its operability by calling *esp_ota_mark_app_valid_cancel_rollback()* function, otherwise the application will be rolled back upon reboot. It allows you to control the operability of the application during the boot phase. Thus, a new application has only one attempt to boot successfully.

Rollback Process

The description of the rollback process when *CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE* option is enabled:

- The new application successfully downloaded and *esp_ota_set_boot_partition()* function makes this partition bootable and sets the state `ESP_OTA_IMG_NEW`. This state means that the application is new and should be monitored for its first boot.
- Reboot *esp_restart()*.
- The bootloader checks for the `ESP_OTA_IMG_PENDING_VERIFY` state if it is set, then it will be written to `ESP_OTA_IMG_ABORTED`.

- The bootloader selects a new application to boot so that the state is not set as ESP_OTA_IMG_INVALID or ESP_OTA_IMG_ABORTED.
- The bootloader checks the selected application for ESP_OTA_IMG_NEW state if it is set, then it will be written to ESP_OTA_IMG_PENDING_VERIFY. This state means that the application requires confirmation of its operability, if this does not happen and a reboot occurs, this state will be overwritten to ESP_OTA_IMG_ABORTED (see above) and this application will no longer be able to start, i.e. there will be a rollback to the previous work application.
- A new application has started and should make a self-test.
- If the self-test has completed successfully, then you must call the function *esp_ota_mark_app_valid_cancel_rollback()* because the application is awaiting confirmation of operability (ESP_OTA_IMG_PENDING_VERIFY state).
- If the self-test fails then call *esp_ota_mark_app_invalid_rollback_and_reboot()* function to roll back to the previous working application, while the invalid application is set ESP_OTA_IMG_INVALID state.
- If the application has not been confirmed, the state remains ESP_OTA_IMG_PENDING_VERIFY, and the next boot it will be changed to ESP_OTA_IMG_ABORTED. That will prevent re-boot of this application. There will be a rollback to the previous working application.

Unexpected Reset

If a power loss or an unexpected crash occurs at the time of the first boot of a new application, it will roll back the application.

Recommendation: Perform the self-test procedure as quickly as possible, to prevent rollback due to power loss.

Only OTA partitions can be rolled back. Factory partition is not rolled back.

Booting invalid/aborted apps

Booting an application which was previously set to ESP_OTA_IMG_INVALID or ESP_OTA_IMG_ABORTED is possible:

- Get the last invalid application partition *esp_ota_get_last_invalid_partition()*.
- Pass the received partition to *esp_ota_set_boot_partition()*, this will update the otadata.
- Restart *esp_restart()*. The bootloader will boot the specified application.

To determine if self-tests should be run during startup of an application, call the *esp_ota_get_state_partition()* function. If result is ESP_OTA_IMG_PENDING_VERIFY then self-testing and subsequent confirmation of operability is required.

Where the states are set

A brief description of where the states are set:

- `ESP_OTA_IMG_VALID` state is set by *esp_ota_mark_app_valid_cancel_rollback()* function.
- `ESP_OTA_IMG_UNDEFINED` state is set by *esp_ota_set_boot_partition()* function if *CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE* option is not enabled.
- `ESP_OTA_IMG_NEW` state is set by *esp_ota_set_boot_partition()* function if *CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE* option is enabled.
- `ESP_OTA_IMG_INVALID` state is set by *esp_ota_mark_app_invalid_rollback_and_reboot()* function.
- `ESP_OTA_IMG_ABORTED` state is set if there was no confirmation of the application operability and occurs reboots (if *CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE* option is enabled).
- `ESP_OTA_IMG_PENDING_VERIFY` state is set in a bootloader if *CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE* option is enabled and selected app has `ESP_OTA_IMG_NEW` state.

Anti-rollback

Anti-rollback prevents rollback to application with security version lower than one programmed in eFuse of chip.

This function works if set *CONFIG_BOOTLOADER_APP_ANTI_ROLLBACK* option. In the bootloader, when selecting a bootable application, an additional security version check is added which is on the chip and in the application image. The version in the bootable firmware must be greater than or equal to the version in the chip.

CONFIG_BOOTLOADER_APP_ANTI_ROLLBACK and *CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE* options are used together. In this case, rollback is possible only on the security version which is equal or higher than the version in the chip.

A typical anti-rollback scheme is

- New firmware released with the elimination of vulnerabilities with the previous version of security.
- After the developer makes sure that this firmware is working. He can increase the security version and release a new firmware.
- Download new application.
- To make it bootable, run the function *esp_ota_set_boot_partition()*. If the security version of the new application is smaller than the version in the chip, the new application will be erased. Update to new firmware is not possible.
- Reboot.

- In the bootloader, an application with a security version greater than or equal to the version in the chip will be selected. If otadata is in the initial state, and one firmware was loaded via a serial channel, whose secure version is higher than the chip, then the secure version of efuse will be immediately updated in the bootloader.
- New application booted. Then the application should perform diagnostics of the operation and if it is completed successfully, you should call *esp_ota_mark_app_valid_cancel_rollback()* function to mark the running application with the ESP_OTA_IMG_VALID state and update the secure version on chip. Note that if was called *esp_ota_mark_app_invalid_rollback_and_reboot()* function a rollback may not happend due to the device may not have any bootable apps then it will return ESP_ERR_OTA_ROLLBACK_FAILED error and stay in the ESP_OTA_IMG_PENDING_VERIFY state.
- The next update of app is possible if a running app is in the ESP_OTA_IMG_VALID state.

Recommendation:

If you want to avoid the download/erase overhead in case of the app from the server has security version lower then running app you have to get `new_app_info.secure_version` from the first package of an image and compare it with the secure version of efuse. Use `esp_efuse_check_secure_version(new_app_info.secure_version)` function if it is true then continue downloading otherwise abort.

```
....
bool image_header_was_checked = false;
while (1) {
    int data_read = esp_http_client_read(client, ota_write_data, BUFFSIZE);
    ...
    if (data_read > 0) {
        if (image_header_was_checked == false) {
            esp_app_desc_t new_app_info;
            if (data_read > sizeof(esp_image_header_t) + sizeof(esp_image_segment_header_
↪t) + sizeof(esp_app_desc_t)) {
                // check current version with downloading
                if (esp_efuse_check_secure_version(new_app_info.secure_version) ==␣
↪false) {
                    ESP_LOGE(TAG, "This a new app can not be downloaded due to a secure␣
↪version is lower than stored in efuse.");
                    http_cleanup(client);
                    task_fatal_error();
                }

                image_header_was_checked = true;

                esp_ota_begin(update_partition, OTA_SIZE_UNKNOWN, &update_handle);
            }
```

(下页继续)

(续上页)

```
        }
        esp_ota_write( update_handle, (const void *)ota_write_data, data_read);
    }
}
...
```

Restrictions:

- The number of bits in the `secure_version` field is limited to 32 bits. This means that only 32 times you can do an anti-rollback. You can reduce the length of this efuse field use *CONFIG_BOOTLOADER_APP_SEC_VER_SIZE_EFUSE_FIELD* option.
- Anti-rollback only works if the encoding scheme for efuse is set to `NONE`.
- The partition table should not have a factory partition, only two of the app.

`security_version`:

- In application image it is stored in `esp_app_desc` structure. The number is set *CONFIG_BOOTLOADER_APP_SECURE_VERSION*.
- In ESP32 it is stored in efuse `EFUSE_BLK3_RDATA4_REG`. (when a eFuse bit is programmed to 1, it can never be reverted to 0). The number of bits set in this register is the `security_version` from app.

Secure OTA Updates Without Secure boot

The verification of signed OTA updates can be performed even without enabling hardware secure boot. For doing so, refer *Signed App Verification Without Hardware Secure Boot*

OTA Tool (otatool.py)

The component *app_update* provides a tool otatool.py for performing OTA partition-related operations on a target device. The following operations can be performed using the tool:

- read contents of otadata partition (read_otadata)
- erase otadata partition, effectively resetting device to factory app (erase_otadata)
- switch OTA partitions (switch_ota_partition)
- erasing OTA partition (erase_ota_partition)
- write to OTA partition (write_ota_partition)
- read contents of OTA partition (read_ota_partition)

The tool can either be imported and used from another Python script or invoked from shell script for users wanting to perform operation programmatically. This is facilitated by the tool's Python API and command-line interface, respectively.

Python API

Before anything else, make sure that the *otatool* module is imported.

```
import sys
import os

idf_path = os.environ["IDF_PATH"]  # get value of IDF_PATH from environment
otatool_dir = os.path.join(idf_path, "components", "app_update")  # otatool.py lives in
↪$IDF_PATH/components/app_update

sys.path.append(otatool_dir)  # this enables Python to find otatool module
from otatool import *  # import all names inside otatool module
```

The starting point for using the tool's Python API to do is create a *OtatoolTarget* object:

```
# Create a partool.py target device connected on serial port /dev/ttyUSB1
target = OtatoolTarget("/dev/ttyUSB1")
```

The created object can now be used to perform operations on the target device:

```
# Erase otadata, reseting the device to factory app
target.erase_otadata()

# Erase contents of OTA app slot 0
target.erase_ota_partition(0)

# Switch boot partition to that of app slot 1
target.switch_ota_partition(1)

# Read OTA partition 'ota_3' and save contents to a file named 'ota_3.bin'
target.read_ota_partition("ota_3", "ota_3.bin")
```

The OTA partition to operate on is specified using either the app slot number or the partition name.

More information on the Python API is available in the docstrings for the tool.

Command-line Interface

The command-line interface of *otatool.py* has the following structure:

```
otatool.py [command-args] [subcommand] [subcommand-args]

- command-args - these are arguments that are needed for executing the main command␣
↪(parttool.py), mostly pertaining to the target device
- subcommand - this is the operation to be performed
- subcommand-args - these are arguments that are specific to the chosen operation
```

```
# Erase otadata, resetting the device to factory app
otatool.py --port "/dev/ttyUSB1" erase_otadata

# Erase contents of OTA app slot 0
otatool.py --port "/dev/ttyUSB1" erase_ota_partition --slot 0

# Switch boot partition to that of app slot 1
otatool.py --port "/dev/ttyUSB1" switch_ota_partition --slot 1

# Read OTA partition 'ota_3' and save contents to a file named 'ota_3.bin'
otatool.py --port "/dev/ttyUSB1" read_ota_partition --name=ota_3
```

More information can be obtained by specifying *–help* as argument:

```
# Display possible subcommands and show main command argument descriptions
otatool.py --help

# Show descriptions for specific subcommand arguments
otatool.py [subcommand] --help
```

See also

- *Partition Table documentation*
- *Lower-Level SPI Flash/Partition API*
- *ESP HTTPS OTA*

Application Example

End-to-end example of OTA firmware update workflow: system/ota.

API Reference

Header File

- app_update/include/esp_ota_ops.h

Functions

`const` *esp_app_desc_t* *`esp_ota_get_app_description`(void)

Return esp_app_desc structure. This structure includes app version.

Return description for running app.

Return Pointer to esp_app_desc structure.

int `esp_ota_get_app_elf_sha256`(char **dst*, size_t *size*)

Fill the provided buffer with SHA256 of the ELF file, formatted as hexadecimal, null-terminated. If the buffer size is not sufficient to fit the entire SHA256 in hex plus a null terminator, the largest possible number of bytes will be written followed by a null.

Return Number of bytes written to dst (including null terminator)

Parameters

- `dst`: Destination buffer
- `size`: Size of the buffer

esp_err_t `esp_ota_begin`(`const` *esp_partition_t* **partition*, size_t *image_size*, *esp_ota_handle_t* **out_handle*)

Commence an OTA update writing to the specified partition.

The specified partition is erased to the specified image size.

If image size is not yet known, pass OTA_SIZE_UNKNOWN which will cause the entire partition to be erased.

On success, this function allocates memory that remains in use until esp_ota_end() is called with the returned handle.

Note: If the rollback option is enabled and the running application has the ESP_OTA_IMG_PENDING_VERIFY state then it will lead to the ESP_ERR_OTA_ROLLBACK_INVALID_STATE error. Confirm the running app before to run download a new app, use esp_ota_mark_app_valid_cancel_rollback() function for it (this should be done as early as possible when you first download a new application).

Return

- ESP_OK: OTA operation commenced successfully.

- ESP_ERR_INVALID_ARG: partition or out_handle arguments were NULL, or partition doesn't point to an OTA app partition.
- ESP_ERR_NO_MEM: Cannot allocate memory for OTA operation.
- ESP_ERR_OTA_PARTITION_CONFLICT: Partition holds the currently running firmware, cannot update in place.
- ESP_ERR_NOT_FOUND: Partition argument not found in partition table.
- ESP_ERR_OTA_SELECT_INFO_INVALID: The OTA data partition contains invalid data.
- ESP_ERR_INVALID_SIZE: Partition doesn't fit in configured flash size.
- ESP_ERR_FLASH_OP_TIMEOUT or ESP_ERR_FLASH_OP_FAIL: Flash write failed.
- ESP_ERR_OTA_ROLLBACK_INVALID_STATE: If the running app has not confirmed state. Before performing an update, the application must be valid.

Parameters

- `partition`: Pointer to info for partition which will receive the OTA update. Required.
- `image_size`: Size of new OTA app image. Partition will be erased in order to receive this size of image. If 0 or OTA_SIZE_UNKNOWN, the entire partition is erased.
- `out_handle`: On success, returns a handle which should be used for subsequent esp_ota_write() and esp_ota_end() calls.

esp_err_t **esp_ota_write**(*esp_ota_handle_t handle*, **const** void **data*, size_t *size*)

Write OTA update data to partition.

This function can be called multiple times as data is received during the OTA operation. Data is written sequentially to the partition.

Return

- ESP_OK: Data was written to flash successfully.
- ESP_ERR_INVALID_ARG: handle is invalid.
- ESP_ERR_OTA_VALIDATE_FAILED: First byte of image contains invalid app image magic byte.
- ESP_ERR_FLASH_OP_TIMEOUT or ESP_ERR_FLASH_OP_FAIL: Flash write failed.
- ESP_ERR_OTA_SELECT_INFO_INVALID: OTA data partition has invalid contents

Parameters

- `handle`: Handle obtained from esp_ota_begin

- `data`: Data buffer to write
- `size`: Size of data buffer in bytes.

esp_err_t **esp_ota_end**(*esp_ota_handle_t handle*)

Finish OTA update and validate newly written app image.

Note After calling esp_ota_end(), the handle is no longer valid and any memory associated with it is freed (regardless of result).

Return

- ESP_OK: Newly written OTA app image is valid.
- ESP_ERR_NOT_FOUND: OTA handle was not found.
- ESP_ERR_INVALID_ARG: Handle was never written to.
- ESP_ERR_OTA_VALIDATE_FAILED: OTA image is invalid (either not a valid app image, or - if secure boot is enabled - signature failed to verify.)
- ESP_ERR_INVALID_STATE: If flash encryption is enabled, this result indicates an internal error writing the final encrypted bytes to flash.

Parameters

- `handle`: Handle obtained from esp_ota_begin().

esp_err_t **esp_ota_set_boot_partition**(**const** *esp_partition_t* **partition*)

Configure OTA data for a new boot partition.

Note If this function returns ESP_OK, calling esp_restart() will boot the newly configured app partition.

Return

- ESP_OK: OTA data updated, next reboot will use specified partition.
- ESP_ERR_INVALID_ARG: partition argument was NULL or didn’ t point to a valid OTA partition of type “app” .
- ESP_ERR_OTA_VALIDATE_FAILED: Partition contained invalid app image. Also returned if secure boot is enabled and signature validation failed.
- ESP_ERR_NOT_FOUND: OTA data partition not found.
- ESP_ERR_FLASH_OP_TIMEOUT or ESP_ERR_FLASH_OP_FAIL: Flash erase or write failed.

Parameters

- `partition`: Pointer to info for partition containing app image to boot.

`const` *esp_partition_t* *`esp_ota_get_boot_partition`(void)

Get partition info of currently configured boot app.

If esp_ota_set_boot_partition() has been called, the partition which was set by that function will be returned.

If esp_ota_set_boot_partition() has not been called, the result is usually the same as esp_ota_get_running_partition(). The two results are not equal if the configured boot partition does not contain a valid app (meaning that the running partition will be an app that the bootloader chose via fallback).

If the OTA data partition is not present or not valid then the result is the first app partition found in the partition table. In priority order, this means: the factory app, the first OTA app slot, or the test app partition.

Note that there is no guarantee the returned partition is a valid app. Use esp_image_verify(ESP_IMAGE_VERIFY, ···) to verify if the returned partition contains a bootable image.

Return Pointer to info for partition structure, or NULL if partition table is invalid or a flash read operation failed. Any returned pointer is valid for the lifetime of the application.

`const` *esp_partition_t* *`esp_ota_get_running_partition`(void)

Get partition info of currently running app.

This function is different to esp_ota_get_boot_partition() in that it ignores any change of selected boot partition caused by esp_ota_set_boot_partition(). Only the app whose code is currently running will have its partition information returned.

The partition returned by this function may also differ from esp_ota_get_boot_partition() if the configured boot partition is somehow invalid, and the bootloader fell back to a different app partition at boot.

Return Pointer to info for partition structure, or NULL if no partition is found or flash read operation failed. Returned pointer is valid for the lifetime of the application.

`const` *esp_partition_t* *`esp_ota_get_next_update_partition`(`const *esp_partition_t* *start_from`)

Return the next OTA app partition which should be written with a new firmware.

Call this function to find an OTA app partition which can be passed to esp_ota_begin().

Finds next partition round-robin, starting from the current running partition.

Return Pointer to info for partition which should be updated next. NULL result indicates invalid OTA data partition, or that no eligible OTA app slot partition was found.

Parameters

- `start_from`: If set, treat this partition info as describing the current running partition. Can be NULL, in which case esp_ota_get_running_partition() is used to find the currently running partition. The result of this function is never the same as this argument.

esp_err_t **esp_ota_get_partition_description**(const *esp_partition_t* *partition, *esp_app_desc_t* *app_desc)

Returns esp_app_desc structure for app partition. This structure includes app version.

Returns a description for the requested app partition.

Return

- ESP_OK Successful.
- ESP_ERR_NOT_FOUND app_desc structure is not found. Magic word is incorrect.
- ESP_ERR_NOT_SUPPORTED Partition is not application.
- ESP_ERR_INVALID_ARG Arguments is NULL or if partition' s offset exceeds partition size.
- ESP_ERR_INVALID_SIZE Read would go out of bounds of the partition.
- or one of error codes from lower-level flash driver.

Parameters

- `partition`: Pointer to app partition. (only app partition)
- `app_desc`: Structure of info about app.

esp_err_t **esp_ota_mark_app_valid_cancel_rollback**(void)

This function is called to indicate that the running app is working well.

Return

- ESP_OK: if successful.

esp_err_t **esp_ota_mark_app_invalid_rollback_and_reboot**(void)

This function is called to roll back to the previously workable app with reboot.

If rollback is successful then device will reset else API will return with error code. Checks applications on a flash drive that can be booted in case of rollback. If the flash does not have at least one app (except the running app) then rollback is not possible.

Return

- ESP_FAIL: if not successful.
- ESP_ERR_OTA_ROLLBACK_FAILED: The rollback is not possible due to flash does not have any apps.

const *esp_partition_t* ***esp_ota_get_last_invalid_partition**(void)

Returns last partition with invalid state (ESP_OTA_IMG_INVALID or ESP_OTA_IMG_ABORTED).

Return partition.

esp_err_t **esp_ota_get_state_partition(const** *esp_partition_t* ***partition**, esp_ota_img_states_t ***ota_state**)

Returns state for given partition.

Return

- ESP_OK: Successful.
- ESP_ERR_INVALID_ARG: partition or ota_state arguments were NULL.
- ESP_ERR_NOT_SUPPORTED: partition is not ota.
- ESP_ERR_NOT_FOUND: Partition table does not have otadata or state was not found for given partition.

Parameters

- `partition`: Pointer to partition.
- `ota_state`: state of partition (if this partition has a record in otadata).

esp_err_t **esp_ota_erase_last_boot_app_partition**(void)

Erase previous boot app partition and corresponding otadata select for this partition.

When current app is marked to as valid then you can erase previous app partition.

Return

- ESP_OK: Successful, otherwise ESP_ERR.

bool **esp_ota_check_rollback_is_possible**(void)

Checks applications on the slots which can be booted in case of rollback.

These applications should be valid (marked in otadata as not UNDEFINED, INVALID or ABORTED and crc is good) and be able booted, and secure_version of app >= secure_version of efuse (if anti-rollback is enabled).

Return

- True: Returns true if the slots have at least one app (except the running app).
- False: The rollback is not possible.

Macros

`OTA_SIZE_UNKNOWN`
Used for esp_ota_begin() if new image size is unknown

`ESP_ERR_OTA_BASE`
Base error code for ota_ops api

`ESP_ERR_OTA_PARTITION_CONFLICT`
Error if request was to write or erase the current running partition

`ESP_ERR_OTA_SELECT_INFO_INVALID`
Error if OTA data partition contains invalid content

`ESP_ERR_OTA_VALIDATE_FAILED`
Error if OTA app image is invalid

`ESP_ERR_OTA_SMALL_SEC_VER`
Error if the firmware has a secure version less than the running firmware.

`ESP_ERR_OTA_ROLLBACK_FAILED`
Error if flash does not have valid firmware in passive partition and hence rollback is not possible

`ESP_ERR_OTA_ROLLBACK_INVALID_STATE`
Error if current active firmware is still marked in pending validation state (ESP_OTA_IMG_PENDING_VERIFY), essentially first boot of firmware image post upgrade and hence firmware upgrade is not possible

Type Definitions

`typedef` uint32_t **`esp_ota_handle_t`**
Opaque handle for an application OTA update.

esp_ota_begin() returns a handle which is then used for subsequent calls to esp_ota_write() and esp_ota_end().

2.7.20 Performance Monitor

The Performance Monitor component provides APIs to use ESP32 internal performance counters to profile functions and applications.

Application Example

An example which combines performance monitor is provided in `examples/system/perfmon` directory. This example initializes the performance monitor structure and execute them with printing the statistics.

High level API Reference

Header Files

- perfmon/include/perfmon.h

API Reference

Header File

- perfmon/include/xtensa_perfmon_access.h

Functions

esp_err_t **xtensa_perfmon_init**(int *id*, uint16_t *select*, uint16_t *mask*, int *kernelcnt*, int *tracelevel*)

Init Performance Monitoor.

Initialize performance monitor register with define values

Return

- ESP_OK on success
- ESP_ERR_INVALID_ARG if one of the arguments is not correct

Parameters

- `id`: performance counter number
- `select`: select value from PMCTRLx register
- `mask`: mask value from PMCTRLx register
- `kernelcnt`: kernelcnt value from PMCTRLx register
- `tracelevel`: tracelevel value from PMCTRLx register

esp_err_t **xtensa_perfmon_reset**(int *id*)

Reset PM counter.

Reset PM counter. Writes 0 to the PMx register.

Return

- ESP_OK on success
- ESP_ERR_INVALID_ARG if id out of range

Parameters

- `id`: performance counter number

void **xtensa_perfmon_start**(void)

Start PM counters.

Start all PM counters synchronously. Write 1 to the PGM register

void **xtensa_perfmon_stop**(void)

Stop PM counters.

Stop all PM counters synchronously. Write 0 to the PGM register

uint32_t **xtensa_perfmon_value**(int *id*)

Read PM counter.

Read value of defined PM counter.

Return

- Performance counter value

Parameters

- `id`: performance counter number

esp_err_t **xtensa_perfmon_overflow**(int *id*)

Read PM overflow state.

Read overflow value of defined PM counter.

Return

- ESP_OK if there is no overflow (overflow = 0)
- ESP_FAIL if overflow occure (overflow = 1)

Parameters

- `id`: performance counter number

void **xtensa_perfmon_dump**(void)

Dump PM values.

Dump all PM register to the console.

Header File

- perfmon/include/xtensa_perfmon_apis.h

Functions

esp_err_t **xtensa_perfmon_exec**(const *xtensa_perfmon_config_t* **config*)

Execute PM.

Execute performance counter for dedicated function with defined parameters

Return

- ESP_OK if no errors
- ESP_ERR_INVALID_ARG if one of the required parameters not defined
- ESP_FAIL - counter overflow

Parameters

- `config`: pointer to the configuration structure

void **xtensa_perfmon_view_cb**(void **params*, uint32_t *select*, uint32_t *mask*, uint32_t *value*)

Dump PM results.

Callback to dump perfmon result to a FILE* stream specified in perfmon_config_t::callback_params. If callback_params is set to NULL, will print to stdout

Parameters

- `params`: used parameters passed from configuration (callback_params). This parameter expected as FILE* hanle, where data will be stored. If this parameter NULL, then data will be stored to the stdout.
- `select`: select value for current counter
- `mask`: mask value for current counter
- `value`: counter value for current counter

Structures

struct xtensa_perfmon_config

Performance monitor configuration structure.

Structure to configure performance counter to measure dedicated function

Public Members

int **repeat_count**

how much times function will be called before the calback will be repeated

float **max_deviation**

Difference between min and max counter number 0..1, 0 - no difference, 1 - not used

void ***call_params**

This pointer will be passed to the call_function as a parameter

void (*call_function)(void *params)
pointer to the function that have to be called

void (*callback)(void *params, uint32_t select, uint32_t mask, uint32_t value)
pointer to the function that will be called with result parameters

void *callback_params
parameter that will be passed to the callback

int tracelevel
trace level for all counters. In case of negative value, the filter will be ignored. If it' s >=0, then the perfmon will count only when interrupt level > tracelevel. It' s useful to monitor interrupts.

uint32_t counters_size
amount of counter in the list

const uint32_t *select_mask
list of the select/mask parameters

Type Definitions

typedef struct *xtensa_perfmon_config* xtensa_perfmon_config_t
Performance monitor configuration structure.

Structure to configure performance counter to measure dedicated function

2.7.21 电源管理

[English]

概述

ESP-IDF 中集成的电源管理算法可以根据应用程序组件的需求，调整外围总线 (APB) 频率、CPU 频率，并使芯片进入 Light-sleep 模式，尽可能减少运行应用程序的功耗。

应用程序组件可以通过创建和获取电源管理锁来控制功耗。

例如：

- 对于从 APB 获得时钟频率的外设，其驱动可以要求在使用该外设时，将 APB 频率设置为 80 MHz。
- RTOS 可以要求 CPU 在有任务准备开始运行时以最高配置频率工作。
- 一些外设可能需要中断才能启用，因此其驱动也会要求禁用 Light-sleep 模式。

因为请求较高的 APB 频率或 CPU 频率，以及禁用 Light-sleep 模式会增加功耗，请将组件使用的电源管理锁降到最少。

电源管理配置

编译时可使用*CONFIG_PM_ENABLE* 选项启用电源管理功能。

启用电源管理功能将会增加中断延迟。额外延迟与多个因素有关，例如：CPU 频率、单/双核模式、是否需要进行频率切换等。CPU 频率为 240 MHz 且未启用频率调节时，最小额外延迟为 0.2 us；如果启用频率调节，且在中断入口将频率由 40 MHz 调节至 80 MHz，则最大额外延迟为 40 us。

应用程序可以通过调用*esp_pm_configure()* 函数启用动态调频 (DFS) 功能和自动 Light-sleep 模式。此函数的参数为*esp_pm_config_esp32_t*，定义了频率调节的相关设置。在此参数结构中，需要初始化下面三个字段：

- `max_freq_mhz`：最大 CPU 频率 (MHz)，即获取 `ESP_PM_CPU_FREQ_MAX` 锁后所使用的频率。该字段通常设置为*CONFIG_ESP32_DEFAULT_CPU_FREQ_MHZ*。
- `min_freq_mhz`：最小 CPU 频率 (MHz)，即仅获取 `ESP_PM_APB_FREQ_MAX` 锁后所使用的频率。该字段可设置为晶振 (XTAL) 频率值，或者 XTAL 频率值除以整数。注意，10 MHz 是生成 1 MHz 的 REF_TICK 默认时钟所需的最小频率。
- `light_sleep_enable`：没有获取任何管理锁时，决定系统是否需要自动进入 Light-sleep 状态 (`true`/`false`)。

或者，如果在 menuconfig 中启用了*CONFIG_PM_DFS_INIT_AUTO* 选项，最大 CPU 频率将由*CONFIG_ESP32_DEFAULT_CPU_FREQ_MHZ* 设置决定，最小 CPU 频率将锁定为 XTAL 频率。

注解:

1. 自动 Light-sleep 模式基于 FreeRTOS Tickless Idle 功能，因此如果在 menuconfig 中没有启用*CONFIG_FREERTOS_USE_TICKLESS_IDLE* 选项，在请求自动 Light-sleep 时，*esp_pm_configure()* 将会返回 *ESP_ERR_NOT_SUPPORTED* 错误。
2. 在 Light-sleep 状态下，外设设有时钟门控，不会产生来自 GPIO 和内部外设的中断。*Sleep Modes* 文档中所提到的唤醒源可用于从 Light-sleep 状态触发唤醒。例如，EXT0 和 EXT1 唤醒源就可以通过 GPIO 唤醒芯片。

电源管理锁

应用程序可以通过获取或释放管理锁来控制电源管理算法。应用程序获取电源管理锁后，电源管理算法的操作将受到下面的限制。释放电源管理锁后，限制解除。

电源管理锁设有获取/释放计数器，如果已多次获取电源管理锁，则需要将电源管理锁释放相同次数以解除限制。

ESP32 支持下表中所述的三种电源管理锁。

电源管理锁	描述
ESP_PM_CPU_FREQ_MAX	请求使用 *esp_pm_configure()* 将 CPU 频率设置为最大值。ESP32 可以将该值设置为 80 MHz、160 MHz 或 240 MHz。
ESP_PM_APB_FREQ_MAX	请求将 APB 频率设置为最大值，ESP32 支持的最大频率为 80 MHz。
ESP_PM_NO_LIGHT_SLEEP	禁止自动切换至 Light-sleep 模式。

电源管理算法

下表列出了启用动态调频时如何切换 CPU 频率和 APB 频率。您可以使用 *esp_pm_configure()* 或者 *CONFIG_ESP32_DEFAULT_CPU_FREQ_MHZ* 指定 CPU 最大频率。

CPU 最高频率	电源管理锁获取情况	CPU 频率和 APB 频率
240 MHz	获取 ESP_PM_CPU_FREQ_MAX 或 ESP_PM_APB_FREQ_MAX	CPU: 240 MHz APB: 80 MHz
	无	使用 *esp_pm_configure()* 为二者设置最小值
160 MHz	获取 ESP_PM_CPU_FREQ_MAX	CPU: 160 MHz APB: 80 MHz
	获取 ESP_PM_CPU_FREQ_MAX，未获取 ESP_PM_APB_FREQ_MAX	CPU: 80 MHz APB: 80 MHz
	无	使用 *esp_pm_configure()* 为二者设置最小值
80 MHz	获取 ESP_PM_CPU_FREQ_MAX 或 ESP_PM_APB_FREQ_MAX	CPU: 80 MHz APB: 80 MHz
	无	使用 *esp_pm_configure()* 为二者设置最小值

如果没有获取任何管理锁，调用 *esp_pm_configure()* 将启动 Light-sleep 模式。Light-sleep 模式持续时间由以下因素决定：

- 处于阻塞状态的 FreeRTOS 任务数（有限超时）
- 高分辨率定时器 API 注册的计数器数量

您也可以设置 Light-sleep 模式在最近事件（任务解除阻塞，或计时器超时）之前持续多久才唤醒芯片。

动态调频和外设驱动

启用动态调频后，APB 频率可在一个 RTOS 滴答周期内多次更改。有些外设不受 APB 频率变更的影响，但有些外设可能会出现问题。例如，Timer Group 外设定时器会继续计数，但定时器计数的速度将随 APB 频率的变更而变更。

下面的外设不受 APB 频率变更的影响：

- **UART**：如果 REF_TICK 用作时钟源，则 UART 不受 APB 频率变更影响。请查看*uart_config_t* 中的 *use_ref_tick*。
- **LEDC**：如果 REF_TICK 用作时钟源，则 LEDC 不受 APB 频率变更影响。请查看*ledc_timer_config()* 函数。
- **RMT**：如果 REF_TICK 用作时钟源，则 RMT 不受 APB 频率变更影响。此驱动程序尚不支持 REF_TICK，但可以清除相应通道的 RMT_REF_ALWAYS_ON_CHx 位来启用该功能。

目前以下外设驱动程序可感知动态调频，并在调频期间使用 ESP_PM_APB_FREQ_MAX 锁：

- SPI master
- I2C
- I2S（如果 APLL 锁在使用中，I2S 则会启用 ESP_PM_NO_LIGHT_SLEEP 锁）
- SDMMC

启用以下驱动程序时，将占用 ESP_PM_APB_FREQ_MAX 锁：

- **SPI slave**：从调用*spi_slave_initialize()* 至*spi_slave_free()* 期间。
- **Ethernet**：从调用*esp_eth_driver_install()* 至*esp_eth_driver_uninstall()* 期间。
- **WiFi**：从调用*esp_wifi_start()* 至*esp_wifi_stop()* 期间。如果启用了调制解调器睡眠模式，广播关闭时将释放此管理锁。
- **Bluetooth**：从调用*esp_bt_controller_enable()* 至*esp_bt_controller_disable()* 期间。如果启用了蓝牙调制解调器，广播关闭时将释放此管理锁。但依然占用 ESP_PM_NO_LIGHT_SLEEP 锁。
- **CAN**：从调用*can_driver_install()* 至*can_driver_uninstall()* 期间。

以下外设驱动程序无法感知动态调频，应用程序需自己获取/释放管理锁：

- MCPWM
- PCNT
- Sigma-delta
- Timer Group

API 参考

Header File

- esp_common/include/esp_pm.h

Functions

esp_err_t **esp_pm_configure**(**const** void *_config_)

Set implementation-specific power management configuration.

Return

- ESP_OK on success
- ESP_ERR_INVALID_ARG if the configuration values are not correct
- ESP_ERR_NOT_SUPPORTED if certain combination of values is not supported, or if CONFIG_PM_ENABLE is not enabled in sdkconfig

Parameters

- `config`: pointer to implementation-specific configuration structure (e.g. esp_pm_config_esp32)

esp_err_t **esp_pm_lock_create**(*esp_pm_lock_type_t* *lock_type*, int *arg*, **const** char *_name_, *esp_pm_lock_handle_t* *_out_handle_)

Initialize a lock handle for certain power management parameter.

When lock is created, initially it is not taken. Call esp_pm_lock_acquire to take the lock.

This function must not be called from an ISR.

Return

- ESP_OK on success
- ESP_ERR_NO_MEM if the lock structure can not be allocated
- ESP_ERR_INVALID_ARG if out_handle is NULL or type argument is not valid
- ESP_ERR_NOT_SUPPORTED if CONFIG_PM_ENABLE is not enabled in sdkconfig

Parameters

- `lock_type`: Power management constraint which the lock should control
- `arg`: argument, value depends on lock_type, see esp_pm_lock_type_t
- `name`: arbitrary string identifying the lock (e.g. "wifi" or "spi"). Used by the esp_pm_dump_locks function to list existing locks. May be set to NULL. If not set to NULL, must point to a string which is valid for the lifetime of the lock.
- `out_handle`: handle returned from this function. Use this handle when calling esp_pm_lock_delete, esp_pm_lock_acquire, esp_pm_lock_release. Must not be NULL.

esp_err_t **esp_pm_lock_acquire**(*esp_pm_lock_handle_t* *handle*)

Take a power management lock.

Once the lock is taken, power management algorithm will not switch to the mode specified in a call to esp_pm_lock_create, or any of the lower power modes (higher numeric values of 'mode').

The lock is recursive, in the sense that if esp_pm_lock_acquire is called a number of times, esp_pm_lock_release has to be called the same number of times in order to release the lock.

This function may be called from an ISR.

This function is not thread-safe w.r.t. calls to other esp_pm_lock_* functions for the same handle.

Return

- ESP_OK on success
- ESP_ERR_INVALID_ARG if the handle is invalid
- ESP_ERR_NOT_SUPPORTED if CONFIG_PM_ENABLE is not enabled in sdkconfig

Parameters

- `handle`: handle obtained from esp_pm_lock_create function

esp_err_t **esp_pm_lock_release**(*esp_pm_lock_handle_t handle*)

Release the lock taken using esp_pm_lock_acquire.

Call to this functions removes power management restrictions placed when taking the lock.

Locks are recursive, so if esp_pm_lock_acquire is called a number of times, esp_pm_lock_release has to be called the same number of times in order to actually release the lock.

This function may be called from an ISR.

This function is not thread-safe w.r.t. calls to other esp_pm_lock_* functions for the same handle.

Return

- ESP_OK on success
- ESP_ERR_INVALID_ARG if the handle is invalid
- ESP_ERR_INVALID_STATE if lock is not acquired
- ESP_ERR_NOT_SUPPORTED if CONFIG_PM_ENABLE is not enabled in sdkconfig

Parameters

- `handle`: handle obtained from esp_pm_lock_create function

esp_err_t **esp_pm_lock_delete**(*esp_pm_lock_handle_t handle*)

Delete a lock created using esp_pm_lock.

The lock must be released before calling this function.

This function must not be called from an ISR.

Return

- ESP_OK on success
- ESP_ERR_INVALID_ARG if the handle argument is NULL
- ESP_ERR_INVALID_STATE if the lock is still acquired
- ESP_ERR_NOT_SUPPORTED if CONFIG_PM_ENABLE is not enabled in sdkconfig

Parameters

- `handle`: handle obtained from esp_pm_lock_create function

esp_err_t **esp_pm_dump_locks**(FILE **stream*)

Dump the list of all locks to stderr

This function dumps debugging information about locks created using esp_pm_lock_create to an output stream.

This function must not be called from an ISR. If esp_pm_lock_acquire/release are called while this function is running, inconsistent results may be reported.

Return

- ESP_OK on success
- ESP_ERR_NOT_SUPPORTED if CONFIG_PM_ENABLE is not enabled in sdkconfig

Parameters

- `stream`: stream to print information to; use stdout or stderr to print to the console; use fmemopen/open_memstream to print to a string buffer.

Type Definitions

typedef struct esp_pm_lock ***esp_pm_lock_handle_t**

Opaque handle to the power management lock.

Enumerations

enum esp_pm_lock_type_t

Power management constraints.

Values:

ESP_PM_CPU_FREQ_MAX

Require CPU frequency to be at the maximum value set via esp_pm_configure. Argument is unused and should be set to 0.

`ESP_PM_APB_FREQ_MAX`
: Require APB frequency to be at the maximum value supported by the chip. Argument is unused and should be set to 0.

`ESP_PM_NO_LIGHT_SLEEP`
: Prevent the system from going into light sleep. Argument is unused and should be set to 0.

Header File

- esp32/include/esp32/pm.h

Structures

`struct esp_pm_config_esp32_t`
: Power management config for ESP32.

  Pass a pointer to this structure as an argument to esp_pm_configure function.

Public Members

int **`max_freq_mhz`**
: Maximum CPU frequency, in MHz

int **`min_freq_mhz`**
: Minimum CPU frequency to use when no locks are taken, in MHz

bool **`light_sleep_enable`**
: Enter light sleep when no locks are taken

2.7.22 Sleep Modes

Overview

ESP32 is capable of light sleep and deep sleep power saving modes.

In light sleep mode, digital peripherals, most of the RAM, and CPUs are clock-gated, and supply voltage is reduced. Upon exit from light sleep, peripherals and CPUs resume operation, their internal state is preserved.

In deep sleep mode, CPUs, most of the RAM, and all the digital peripherals which are clocked from APB_CLK are powered off. The only parts of the chip which can still be powered on are: RTC controller, RTC peripherals (including ULP coprocessor), and RTC memories (slow and fast).

Wakeup from deep and light sleep modes can be done using several sources. These sources can be combined, in this case the chip will wake up when any one of the sources is triggered. Wakeup sources can be enabled

using `esp_sleep_enable_X_wakeup` APIs and can be disabled using *esp_sleep_disable_wakeup_source()* API. Next section describes these APIs in detail. Wakeup sources can be configured at any moment before entering light or deep sleep mode.

Additionally, the application can force specific powerdown modes for the RTC peripherals and RTC memories using *esp_sleep_pd_config()* API.

Once wakeup sources are configured, application can enter sleep mode using *esp_light_sleep_start()* or *esp_deep_sleep_start()* APIs. At this point the hardware will be configured according to the requested wakeup sources, and RTC controller will either power down or power off the CPUs and digital peripherals.

WiFi/BT and sleep modes

In deep sleep and light sleep modes, wireless peripherals are powered down. Before entering deep sleep or light sleep modes, applications must disable WiFi and BT using appropriate calls (*esp_bluedroid_disable()*, *esp_bt_controller_disable()*, *esp_wifi_stop()*). WiFi and BT connections will not be maintained in deep sleep or light sleep, even if these functions are not called.

If WiFi connection needs to be maintained, enable WiFi modem sleep, and enable automatic light sleep feature (see *Power Management APIs*). This will allow the system to wake up from sleep automatically when required by WiFi driver, thereby maintaining connection to the AP.

Wakeup sources

Timer

RTC controller has a built in timer which can be used to wake up the chip after a predefined amount of time. Time is specified at microsecond precision, but the actual resolution depends on the clock source selected for RTC SLOW_CLK. See chapter "Reset and Clock" of the ESP32 Technical Reference Manual for details about RTC clock options.

This wakeup mode doesn't require RTC peripherals or RTC memories to be powered on during sleep.

esp_sleep_enable_timer_wakeup() function can be used to enable deep sleep wakeup using a timer.

Touch pad

RTC IO module contains logic to trigger wakeup when a touch sensor interrupt occurs. You need to configure the touch pad interrupt before the chip starts deep sleep.

Revisions 0 and 1 of the ESP32 only support this wakeup mode when RTC peripherals are not forced to be powered on (i.e. ESP_PD_DOMAIN_RTC_PERIPH should be set to ESP_PD_OPTION_AUTO).

esp_sleep_enable_touchpad_wakeup() function can be used to enable this wakeup source.

External wakeup (ext0)

RTC IO module contains logic to trigger wakeup when one of RTC GPIOs is set to a predefined logic level. RTC IO is part of RTC peripherals power domain, so RTC peripherals will be kept powered on during deep sleep if this wakeup source is requested.

Because RTC IO module is enabled in this mode, internal pullup or pulldown resistors can also be used. They need to be configured by the application using *rtc_gpio_pullup_en()* and *rtc_gpio_pulldown_en()* functions, before calling `esp_sleep_start()`.

In revisions 0 and 1 of the ESP32, this wakeup source is incompatible with ULP and touch wakeup sources.

esp_sleep_enable_ext0_wakeup() function can be used to enable this wakeup source.

警告: After wake up from sleep, IO pad used for wakeup will be configured as RTC IO. Before using this pad as digital GPIO, reconfigure it using `rtc_gpio_deinit(gpio_num)` function.

External wakeup (ext1)

RTC controller contains logic to trigger wakeup using multiple RTC GPIOs. One of the two logic functions can be used to trigger wakeup:

- wake up if any of the selected pins is high (`ESP_EXT1_WAKEUP_ANY_HIGH`)
- wake up if all the selected pins are low (`ESP_EXT1_WAKEUP_ALL_LOW`)

This wakeup source is implemented by the RTC controller. As such, RTC peripherals and RTC memories can be powered down in this mode. However, if RTC peripherals are powered down, internal pullup and pulldown resistors will be disabled. To use internal pullup or pulldown resistors, request RTC peripherals power domain to be kept on during sleep, and configure pullup/pulldown resistors using `rtc_gpio_` functions, before entering sleep:

```
esp_sleep_pd_config(ESP_PD_DOMAIN_RTC_PERIPH, ESP_PD_OPTION_ON);
gpio_pullup_dis(gpio_num);
gpio_pulldown_en(gpio_num);
```

警告: After wake up from sleep, IO pad(s) used for wakeup will be configured as RTC IO. Before using these pads as digital GPIOs, reconfigure them using `rtc_gpio_deinit(gpio_num)` function.

esp_sleep_enable_ext1_wakeup() function can be used to enable this wakeup source.

ULP coprocessor wakeup

ULP coprocessor can run while the chip is in sleep mode, and may be used to poll sensors, monitor ADC or touch sensor values, and wake up the chip when a specific event is detected. ULP coprocessor is part of RTC peripherals power domain, and it runs the program stored in RTC slow memory. RTC slow memory will be powered on during sleep if this wakeup mode is requested. RTC peripherals will be automatically powered on before ULP coprocessor starts running the program; once the program stops running, RTC peripherals are automatically powered down again.

Revisions 0 and 1 of the ESP32 only support this wakeup mode when RTC peripherals are not forced to be powered on (i.e. ESP_PD_DOMAIN_RTC_PERIPH should be set to ESP_PD_OPTION_AUTO).

esp_sleep_enable_ulp_wakeup() function can be used to enable this wakeup source.

GPIO wakeup (light sleep only)

In addition to EXT0 and EXT1 wakeup sources described above, one more method of wakeup from external inputs is available in light sleep mode. With this wakeup source, each pin can be individually configured to trigger wakeup on high or low level using *gpio_wakeup_enable()* function. Unlike EXT0 and EXT1 wakeup sources, which can only be used with RTC IOs, this wakeup source can be used with any IO (RTC or digital).

esp_sleep_enable_gpio_wakeup() function can be used to enable this wakeup source.

UART wakeup (light sleep only)

When ESP32 receives UART input from external devices, it is often required to wake up the chip when input data is available. UART peripheral contains a feature which allows waking up the chip from light sleep when a certain number of positive edges on RX pin are seen. This number of positive edges can be set using *uart_set_wakeup_threshold()* function. Note that the character which triggers wakeup (and any characters before it) will not be received by the UART after wakeup. This means that the external device typically needs to send an extra character to the ESP32 to trigger wakeup, before sending the data.

esp_sleep_enable_uart_wakeup() function can be used to enable this wakeup source.

Power-down of RTC peripherals and memories

By default, *esp_deep_sleep_start()* and *esp_light_sleep_start()* functions will power down all RTC power domains which are not needed by the enabled wakeup sources. To override this behaviour, *esp_sleep_pd_config()* function is provided.

Note: in revision 0 of the ESP32, RTC fast memory will always be kept enabled in deep sleep, so that the deep sleep stub can run after reset. This can be overridden, if the application doesn’ t need clean reset behaviour after deep sleep.

If some variables in the program are placed into RTC slow memory (for example, using `RTC_DATA_ATTR` attribute), RTC slow memory will be kept powered on by default. This can be overridden using *esp_sleep_pd_config()* function, if desired.

Entering light sleep

esp_light_sleep_start() function can be used to enter light sleep once wakeup sources are configured. It is also possible to go into light sleep with no wakeup sources configured, in this case the chip will be in light sleep mode indefinitely, until external reset is applied.

Entering deep sleep

esp_deep_sleep_start() function can be used to enter deep sleep once wakeup sources are configured. It is also possible to go into deep sleep with no wakeup sources configured, in this case the chip will be in deep sleep mode indefinitely, until external reset is applied.

Configuring IOs

Some ESP32 IOs have internal pullups or pulldowns, which are enabled by default. If an external circuit drives this pin in deep sleep mode, current consumption may increase due to current flowing through these pullups and pulldowns.

To isolate a pin, preventing extra current draw, call *rtc_gpio_isolate()* function.

For example, on ESP32-WROVER module, GPIO12 is pulled up externally. GPIO12 also has an internal pulldown in the ESP32 chip. This means that in deep sleep, some current will flow through these external and internal resistors, increasing deep sleep current above the minimal possible value. Add the following code before *esp_deep_sleep_start()* to remove this extra current:

```
rtc_gpio_isolate(GPIO_NUM_12);
```

UART output handling

Before entering sleep mode, *esp_deep_sleep_start()* will flush the contents of UART FIFOs.

When entering light sleep mode using *esp_light_sleep_start()*, UART FIFOs will not be flushed. Instead, UART output will be suspended, and remaining characters in the FIFO will be sent out after wakeup from light sleep.

Checking sleep wakeup cause

esp_sleep_get_wakeup_cause() function can be used to check which wakeup source has triggered wakeup from sleep mode.

For touch pad and ext1 wakeup sources, it is possible to identify pin or touch pad which has caused wakeup using *esp_sleep_get_touchpad_wakeup_status()* and *esp_sleep_get_ext1_wakeup_status()* functions.

Disable sleep wakeup source

Previously configured wakeup source can be disabled later using *esp_sleep_disable_wakeup_source()* API. This function deactivates trigger for the given wakeup source. Additionally it can disable all triggers if the argument is `ESP_SLEEP_WAKEUP_ALL`.

Application Example

Implementation of basic functionality of deep sleep is shown in protocols/sntp example, where ESP module is periodically waken up to retrieve time from NTP server.

More extensive example in system/deep_sleep illustrates usage of various deep sleep wakeup triggers and ULP coprocessor programming.

API Reference

Header File

- esp32/include/esp_sleep.h

Functions

esp_err_t **esp_sleep_disable_wakeup_source**(*esp_sleep_source_t source*)

Disable wakeup source.

This function is used to deactivate wake up trigger for source defined as parameter of the function.

See docs/sleep-modes.rst for details.

Note This function does not modify wake up configuration in RTC. It will be performed in esp_sleep_start function.

Return

- ESP_OK on success
- ESP_ERR_INVALID_STATE if trigger was not active

Parameters

- `source`: - number of source to disable of type esp_sleep_source_t

esp_err_t **esp_sleep_enable_ulp_wakeup**(void)

Enable wakeup by ULP coprocessor.

Note In revisions 0 and 1 of the ESP32, ULP wakeup source can not be used when RTC_PERIPH power domain is forced to be powered on (ESP_PD_OPTION_ON) or when ext0 wakeup source is used.

Return

- ESP_OK on success
- ESP_ERR_NOT_SUPPORTED if additional current by touch (CONFIG_ESP32_RTC_EXT_CRYST_ADDIT_CURRENT) is enabled.
- ESP_ERR_INVALID_STATE if ULP co-processor is not enabled or if wakeup triggers conflict

esp_err_t **esp_sleep_enable_timer_wakeup**(uint64_t *time_in_us*)

Enable wakeup by timer.

Return

- ESP_OK on success
- ESP_ERR_INVALID_ARG if value is out of range (TBD)

Parameters

- `time_in_us`: time before wakeup, in microseconds

esp_err_t **esp_sleep_enable_touchpad_wakeup**(void)

Enable wakeup by touch sensor.

Note In revisions 0 and 1 of the ESP32, touch wakeup source can not be used when RTC_PERIPH power domain is forced to be powered on (ESP_PD_OPTION_ON) or when ext0 wakeup source is used.

Note The FSM mode of the touch button should be configured as the timer trigger mode.

Return

- ESP_OK on success
- ESP_ERR_NOT_SUPPORTED if additional current by touch (CONFIG_ESP32_RTC_EXT_CRYST_ADDIT_CURRENT) is enabled.
- ESP_ERR_INVALID_STATE if wakeup triggers conflict

touch_pad_t **esp_sleep_get_touchpad_wakeup_status**(void)

Get the touch pad which caused wakeup.

If wakeup was caused by another source, this function will return TOUCH_PAD_MAX;

Return touch pad which caused wakeup

esp_err_t **esp_sleep_enable_ext0_wakeup**(*gpio_num_t gpio_num*, int *level*)

Enable wakeup using a pin.

This function uses external wakeup feature of RTC_IO peripheral. It will work only if RTC peripherals are kept on during sleep.

This feature can monitor any pin which is an RTC IO. Once the pin transitions into the state given by level argument, the chip will be woken up.

Note This function does not modify pin configuration. The pin is configured in esp_sleep_start, immediately before entering sleep mode.

Note In revisions 0 and 1 of the ESP32, ext0 wakeup source can not be used together with touch or ULP wakeup sources.

Return

- ESP_OK on success
- ESP_ERR_INVALID_ARG if the selected GPIO is not an RTC GPIO, or the mode is invalid
- ESP_ERR_INVALID_STATE if wakeup triggers conflict

Parameters

- `gpio_num`: GPIO number used as wakeup source. Only GPIOs which are have RTC functionality can be used: 0,2,4,12-15,25-27,32-39.
- `level`: input level which will trigger wakeup (0=low, 1=high)

esp_err_t **esp_sleep_enable_ext1_wakeup**(uint64_t *mask*, *esp_sleep_ext1_wakeup_mode_t mode*)

Enable wakeup using multiple pins.

This function uses external wakeup feature of RTC controller. It will work even if RTC peripherals are shut down during sleep.

This feature can monitor any number of pins which are in RTC IOs. Once any of the selected pins goes into the state given by mode argument, the chip will be woken up.

Note This function does not modify pin configuration. The pins are configured in esp_sleep_start, immediately before entering sleep mode.

Note internal pullups and pulldowns don’ t work when RTC peripherals are shut down. In this case, external resistors need to be added. Alternatively, RTC peripherals (and pullups/pulldowns) may be kept enabled using esp_sleep_pd_config function.

Return

- ESP_OK on success
- ESP_ERR_INVALID_ARG if any of the selected GPIOs is not an RTC GPIO, or mode is invalid

Parameters

- `mask`: bit mask of GPIO numbers which will cause wakeup. Only GPIOs which are have RTC functionality can be used in this bit map: 0,2,4,12-15,25-27,32-39.
- `mode`: select logic function used to determine wakeup condition:
  - ESP_EXT1_WAKEUP_ALL_LOW: wake up when all selected GPIOs are low
  - ESP_EXT1_WAKEUP_ANY_HIGH: wake up when any of the selected GPIOs is high

esp_err_t **esp_sleep_enable_gpio_wakeup**(void)

Enable wakeup from light sleep using GPIOs.

Each GPIO supports wakeup function, which can be triggered on either low level or high level. Unlike EXT0 and EXT1 wakeup sources, this method can be used both for all IOs: RTC IOs and digital IOs. It can only be used to wakeup from light sleep though.

To enable wakeup, first call gpio_wakeup_enable, specifying gpio number and wakeup level, for each GPIO which is used for wakeup. Then call this function to enable wakeup feature.

Note In revisions 0 and 1 of the ESP32, GPIO wakeup source can not be used together with touch or ULP wakeup sources.

Return

- ESP_OK on success
- ESP_ERR_INVALID_STATE if wakeup triggers conflict

esp_err_t **esp_sleep_enable_uart_wakeup**(int *uart_num*)

Enable wakeup from light sleep using UART.

Use uart_set_wakeup_threshold function to configure UART wakeup threshold.

Wakeup from light sleep takes some time, so not every character sent to the UART can be received by the application.

Note ESP32 does not support wakeup from UART2.

Return

- ESP_OK on success
- ESP_ERR_INVALID_ARG if wakeup from given UART is not supported

Parameters

- `uart_num`: UART port to wake up from

uint64_t **esp_sleep_get_ext1_wakeup_status**(void)

Get the bit mask of GPIOs which caused wakeup (ext1)

If wakeup was caused by another source, this function will return 0.

Return bit mask, if GPIOn caused wakeup, BIT(n) will be set

esp_err_t **esp_sleep_pd_config**(*esp_sleep_pd_domain_t domain*, *esp_sleep_pd_option_t option*)

Set power down mode for an RTC power domain in sleep mode.

If not set set using this API, all power domains default to ESP_PD_OPTION_AUTO.

Return

- ESP_OK on success
- ESP_ERR_INVALID_ARG if either of the arguments is out of range

Parameters

- `domain`: power domain to configure
- `option`: power down option (ESP_PD_OPTION_OFF, ESP_PD_OPTION_ON, or ESP_PD_OPTION_AUTO)

void **esp_deep_sleep_start**(void)

Enter deep sleep with the configured wakeup options.

This function does not return.

esp_err_t **esp_light_sleep_start**(void)

Enter light sleep with the configured wakeup options.

Return

- ESP_OK on success (returned after wakeup)
- ESP_ERR_INVALID_STATE if WiFi or BT is not stopped

void **esp_deep_sleep**(uint64_t *time_in_us*)

Enter deep-sleep mode.

The device will automatically wake up after the deep-sleep time Upon waking up, the device calls deep sleep wake stub, and then proceeds to load application.

Call to this function is equivalent to a call to esp_deep_sleep_enable_timer_wakeup followed by a call to esp_deep_sleep_start.

esp_deep_sleep does not shut down WiFi, BT, and higher level protocol connections gracefully. Make sure relevant WiFi and BT stack functions are called to close any connections and deinitialize the peripherals. These include:

- esp_bluedroid_disable
- esp_bt_controller_disable
- esp_wifi_stop

This function does not return.

Parameters

- `time_in_us`: deep-sleep time, unit: microsecond

esp_sleep_wakeup_cause_t **esp_sleep_get_wakeup_cause**(void)

Get the wakeup source which caused wakeup from sleep.

Return cause of wake up from last sleep (deep sleep or light sleep)

void **esp_wake_deep_sleep**(void)

Default stub to run on wake from deep sleep.

Allows for executing code immediately on wake from sleep, before the software bootloader or ESP-IDF app has started up.

This function is weak-linked, so you can implement your own version to run code immediately when the chip wakes from sleep.

See docs/deep-sleep-stub.rst for details.

void **esp_set_deep_sleep_wake_stub**(*esp_deep_sleep_wake_stub_fn_t new_stub*)

Install a new stub at runtime to run on wake from deep sleep.

If implementing esp_wake_deep_sleep() then it is not necessary to call this function.

However, it is possible to call this function to substitute a different deep sleep stub. Any function used as a deep sleep stub must be marked RTC_IRAM_ATTR, and must obey the same rules given for esp_wake_deep_sleep().

esp_deep_sleep_wake_stub_fn_t **esp_get_deep_sleep_wake_stub**(void)

Get current wake from deep sleep stub.

Return Return current wake from deep sleep stub, or NULL if no stub is installed.

void **esp_default_wake_deep_sleep**(void)

The default esp-idf-provided esp_wake_deep_sleep() stub.

See docs/deep-sleep-stub.rst for details.

void **esp_deep_sleep_disable_rom_logging**(void)

Disable logging from the ROM code after deep sleep.

Using LSB of RTC_STORE4.

Type Definitions

typedef *esp_sleep_source_t* **esp_sleep_wakeup_cause_t**

typedef void (***esp_deep_sleep_wake_stub_fn_t**)(void)

Function type for stub to run on wake from sleep.

Enumerations

enum **esp_sleep_ext1_wakeup_mode_t**

Logic function used for EXT1 wakeup mode.

Values:

ESP_EXT1_WAKEUP_ALL_LOW = 0

Wake the chip when all selected GPIOs go low.

ESP_EXT1_WAKEUP_ANY_HIGH = 1

Wake the chip when any of the selected GPIOs go high.

enum **esp_sleep_pd_domain_t**

Power domains which can be powered down in sleep mode.

Values:

ESP_PD_DOMAIN_RTC_PERIPH

RTC IO, sensors and ULP co-processor.

ESP_PD_DOMAIN_RTC_SLOW_MEM

RTC slow memory.

ESP_PD_DOMAIN_RTC_FAST_MEM

RTC fast memory.

ESP_PD_DOMAIN_XTAL

XTAL oscillator.

ESP_PD_DOMAIN_MAX

Number of domains.

enum **esp_sleep_pd_option_t**

Power down options.

Values:

ESP_PD_OPTION_OFF

Power down the power domain in sleep mode.

ESP_PD_OPTION_ON

Keep power domain enabled during sleep mode.

ESP_PD_OPTION_AUTO

Keep power domain enabled in sleep mode, if it is needed by one of the wakeup options. Otherwise power it down.

enum esp_sleep_source_t

Sleep wakeup cause.

Values:

ESP_SLEEP_WAKEUP_UNDEFINED

In case of deep sleep, reset was not caused by exit from deep sleep.

ESP_SLEEP_WAKEUP_ALL

Not a wakeup cause, used to disable all wakeup sources with esp_sleep_disable_wakeup_source.

ESP_SLEEP_WAKEUP_EXT0

Wakeup caused by external signal using RTC_IO.

ESP_SLEEP_WAKEUP_EXT1

Wakeup caused by external signal using RTC_CNTL.

ESP_SLEEP_WAKEUP_TIMER

Wakeup caused by timer.

ESP_SLEEP_WAKEUP_TOUCHPAD

Wakeup caused by touchpad.

ESP_SLEEP_WAKEUP_ULP

Wakeup caused by ULP program.

ESP_SLEEP_WAKEUP_GPIO

Wakeup caused by GPIO (light sleep only)

ESP_SLEEP_WAKEUP_UART

Wakeup caused by UART (light sleep only)

2.7.23 Watchdogs

Overview

The ESP-IDF has support for two types of watchdogs: The Interrupt Watchdog Timer and the Task Watchdog Timer (TWDT). The Interrupt Watchdog Timer and the TWDT can both be enabled using *Project Configuration Menu*, however the TWDT can also be enabled during runtime. The Interrupt Watchdog is responsible for detecting instances where FreeRTOS task switching is blocked for a prolonged period of time. The TWDT is responsible for detecting instances of tasks running without yielding for a prolonged period.

Interrupt watchdog

The interrupt watchdog makes sure the FreeRTOS task switching interrupt isn't blocked for a long time. This is bad because no other tasks, including potentially important ones like the WiFi task and the idle task, can't get any CPU runtime. A blocked task switching interrupt can happen because a program runs into an infinite loop with interrupts disabled or hangs in an interrupt.

The default action of the interrupt watchdog is to invoke the panic handler. causing a register dump and an opportunity for the programmer to find out, using either OpenOCD or gdbstub, what bit of code is stuck with interrupts disabled. Depending on the configuration of the panic handler, it can also blindly reset the CPU, which may be preferred in a production environment.

The interrupt watchdog is built around the hardware watchdog in timer group 1. If this watchdog for some reason cannot execute the NMI handler that invokes the panic handler (e.g. because IRAM is overwritten by garbage), it will hard-reset the SOC.

Task Watchdog Timer

The Task Watchdog Timer (TWDT) is responsible for detecting instances of tasks running for a prolonged period of time without yielding. This is a symptom of CPU starvation and is usually caused by a higher priority task looping without yielding to a lower-priority task thus starving the lower priority task from CPU time. This can be an indicator of poorly written code that spinloops on a peripheral, or a task that is stuck in an infinite loop.

By default the TWDT will watch the Idle Tasks of each CPU, however any task can elect to be watched by the TWDT. Each watched task must 'reset' the TWDT periodically to indicate that they have been allocated CPU time. If a task does not reset within the TWDT timeout period, a warning will be printed with information about which tasks failed to reset the TWDT in time and which tasks are currently running on the ESP32 CPUs. And also there is a possibility to redefine the function *esp_task_wdt_isr_user_handler* in the user code to receive this event.

The TWDT is built around the Hardware Watchdog Timer in Timer Group 0. The TWDT can be initialized by calling *esp_task_wdt_init()* which will configure the hardware timer. A task can then subscribe to the TWDT using *esp_task_wdt_add()* in order to be watched. Each subscribed task must periodically call *esp_task_wdt_reset()* to reset the TWDT. Failure by any subscribed tasks to periodically call *esp_task_wdt_reset()* indicates that one or more tasks have been starved of CPU time or are stuck in a loop somewhere.

A watched task can be unsubscribed from the TWDT using *esp_task_wdt_delete()*. A task that has been unsubscribed should no longer call *esp_task_wdt_reset()*. Once all tasks have unsubscribed form the TWDT, the TWDT can be deinitialized by calling *esp_task_wdt_deinit()*.

By default *CONFIG_ESP_TASK_WDT* in *Project Configuration Menu* be enabled causing the TWDT to be initialized automatically during startup. Likewise *CONFIG_ESP_TASK_WDT_CHECK_IDLE_TASK_CPU0* and *CON-*

FIG_ESP_TASK_WDT_CHECK_IDLE_TASK_CPU1 are also enabled by default causing the two Idle Tasks to be subscribed to the TWDT during startup.

JTAG and watchdogs

While debugging using OpenOCD, the CPUs will be halted every time a breakpoint is reached. However if the watchdog timers continue to run when a breakpoint is encountered, they will eventually trigger a reset making it very difficult to debug code. Therefore OpenOCD will disable the hardware timers of both the interrupt and task watchdogs at every breakpoint. Moreover, OpenOCD will not reenable them upon leaving the breakpoint. This means that interrupt watchdog and task watchdog functionality will essentially be disabled. No warnings or panics from either watchdogs will be generated when the ESP32 is connected to OpenOCD via JTAG.

Interrupt Watchdog API Reference

Header File

- esp32/include/esp_int_wdt.h

Functions

void **esp_int_wdt_init**(void)

Initialize the non-CPU-specific parts of interrupt watchdog. This is called in the init code if the interrupt watchdog is enabled in menuconfig.

Task Watchdog API Reference

A full example using the Task Watchdog is available in esp-idf: system/task_watchdog

Header File

- esp_common/include/esp_task_wdt.h

Functions

esp_err_t **esp_task_wdt_init**(uint32_t *timeout*, bool *panic*)

Initialize the Task Watchdog Timer (TWDT)

This function configures and initializes the TWDT. If the TWDT is already initialized when this function is called, this function will update the TWDT' s timeout period and panic configurations

instead. After initializing the TWDT, any task can elect to be watched by the TWDT by subscribing to it using esp_task_wdt_add().

Return

- ESP_OK: Initialization was successful
- ESP_ERR_NO_MEM: Initialization failed due to lack of memory

Note esp_task_wdt_init() must only be called after the scheduler started

Parameters

- `timeout`: Timeout period of TWDT in seconds
- `panic`: Flag that controls whether the panic handler will be executed when the TWDT times out

esp_err_t **esp_task_wdt_deinit**(void)

Deinitialize the Task Watchdog Timer (TWDT)

This function will deinitialize the TWDT. Calling this function whilst tasks are still subscribed to the TWDT, or when the TWDT is already deinitialized, will result in an error code being returned.

Return

- ESP_OK: TWDT successfully deinitialized
- ESP_ERR_INVALID_STATE: Error, tasks are still subscribed to the TWDT
- ESP_ERR_NOT_FOUND: Error, TWDT has already been deinitialized

esp_err_t **esp_task_wdt_add**(*TaskHandle_t handle*)

Subscribe a task to the Task Watchdog Timer (TWDT)

This function subscribes a task to the TWDT. Each subscribed task must periodically call esp_task_wdt_reset() to prevent the TWDT from elapsing its timeout period. Failure to do so will result in a TWDT timeout. If the task being subscribed is one of the Idle Tasks, this function will automatically enable esp_task_wdt_reset() to called from the Idle Hook of the Idle Task. Calling this function whilst the TWDT is uninitialized or attempting to subscribe an already subscribed task will result in an error code being returned.

Return

- ESP_OK: Successfully subscribed the task to the TWDT
- ESP_ERR_INVALID_ARG: Error, the task is already subscribed
- ESP_ERR_NO_MEM: Error, could not subscribe the task due to lack of memory
- ESP_ERR_INVALID_STATE: Error, the TWDT has not been initialized yet

Parameters

- `handle`: Handle of the task. Input NULL to subscribe the current running task to the TWDT

esp_err_t **esp_task_wdt_reset**(void)

Reset the Task Watchdog Timer (TWDT) on behalf of the currently running task.

This function will reset the TWDT on behalf of the currently running task. Each subscribed task must periodically call this function to prevent the TWDT from timing out. If one or more subscribed tasks fail to reset the TWDT on their own behalf, a TWDT timeout will occur. If the IDLE tasks have been subscribed to the TWDT, they will automatically call this function from their idle hooks. Calling this function from a task that has not subscribed to the TWDT, or when the TWDT is uninitialized will result in an error code being returned.

Return

- ESP_OK: Successfully reset the TWDT on behalf of the currently running task
- ESP_ERR_NOT_FOUND: Error, the current running task has not subscribed to the TWDT
- ESP_ERR_INVALID_STATE: Error, the TWDT has not been initialized yet

esp_err_t **esp_task_wdt_delete**(*TaskHandle_t handle*)

Unsubscribes a task from the Task Watchdog Timer (TWDT)

This function will unsubscribe a task from the TWDT. After being unsubscribed, the task should no longer call esp_task_wdt_reset(). If the task is an IDLE task, this function will automatically disable the calling of esp_task_wdt_reset() from the Idle Hook. Calling this function whilst the TWDT is uninitialized or attempting to unsubscribe an already unsubscribed task from the TWDT will result in an error code being returned.

Return

- ESP_OK: Successfully unsubscribed the task from the TWDT
- ESP_ERR_INVALID_ARG: Error, the task is already unsubscribed
- ESP_ERR_INVALID_STATE: Error, the TWDT has not been initialized yet

Parameters

- `handle`: Handle of the task. Input NULL to unsubscribe the current running task.

esp_err_t **esp_task_wdt_status**(*TaskHandle_t handle*)

Query whether a task is subscribed to the Task Watchdog Timer (TWDT)

This function will query whether a task is currently subscribed to the TWDT, or whether the TWDT is initialized.

Return :

- ESP_OK: The task is currently subscribed to the TWDT

- ESP_ERR_NOT_FOUND: The task is currently not subscribed to the TWDT
- ESP_ERR_INVALID_STATE: The TWDT is not initialized, therefore no tasks can be subscribed

Parameters

- `handle`: Handle of the task. Input NULL to query the current running task.

2.7.24 System Time

Overview

System time can be kept using either one time source or two time sources simultaneously. The choice depends on the application purpose and accuracy requirements for system time.

There are the following two time sources:

- **RTC timer**: Allows keeping the system time during any resets and sleep modes, only the power-up reset leads to resetting the RTC timer. The frequency deviation depends on an *RTC Clock Source* and affects accuracy only in sleep modes, in which case the time will be measured at 6.6667 us resolution.
- **High-resolution timer**: Not available during any reset and sleep modes. The reason for using this timer is to achieve greater accuracy. It uses the APB_CLK clock source (typically 80 MHz), which has a frequency deviation of less than ±10 ppm. Time will be measured at 1 us resolution.

The settings for the system time source are as follows:

- RTC and high-resolution timer (default)
- RTC
- High-resolution timer
- None

It is recommended to stick to the default setting which provides maximum accuracy. If you want to choose a different timer, configure *CONFIG_ESP32_TIME_SYSCALL* in project configuration.

RTC Clock Source

The RTC timer has the following clock sources:

- `Internal 150kHz RC oscillator` (default): Features lowest deep sleep current consumption and no dependence on any external components. However, as frequency stability is affected by temperature fluctuations, time may drift in both Deep and Light sleep modes.
- `External 32kHz crystal`: Requires a 32kHz crystal to be connected to the 32K_XP and 32K_NP pins. Provides better frequency stability at the expense of slightly higher (by 1 uA) Deep sleep current consumption.

- `External 32kHz oscillator at 32K_XP pin`: Allows using 32kHz clock generated by an external circuit. The external clock signal must be connected to the 32K_XP pin. The amplitude should be less than 1.2 V for sine wave signal and less than 1 V for square wave signal. Common mode voltage should be in the range of 0.1 < Vcm < 0.5xVamp, where Vamp is signal amplitude. Additionally, a 1 nF capacitor must be placed between the 32K_XN pin and ground. In this case, the 32K_XN pin cannot be used as a GPIO pin.
- `Internal 8.5MHz oscillator, divided by 256 (~33kHz)`. Provides better frequency stability than the `internal 150kHz RC oscillator` at the expense of higher (by 5 uA) deep sleep current consumption. It also does not require external components.

The choice depends on your requirements for system time accuracy and power consumption in sleep modes. To modify the RTC clock source, set *CONFIG_ESP32_RTC_CLK_SRC* in project configuration.

More details on wiring requirements for the `External 32kHz crystal` and `External 32kHz oscillator at 32K_XP pin` sources can be found in Section 2.1.4 *Crystal Oscillator* of Hardware Design Guidelines.

Get Current Time

To get the current time, use the POSIX function `gettimeofday()`. Additionally, you can use the following standard C library functions to obtain time and manipulate it:

```
gettimeofday
time
asctime
clock
ctime
difftime
gmtime
localtime
mktime
strftime
adjtime*
```

* – To stop smooth time adjustment and update the current time immediately, use the POSIX function `settimeofday()`.

If you need to obtain time with one second resolution, use the following method:

```
time_t now;
char strftime_buf[64];
struct tm timeinfo;

time(&now);
```

(下页继续)

(续上页)

```
// Set timezone to China Standard Time
setenv("TZ", "CST-8", 1);
tzset();

localtime_r(&now, &timeinfo);
strftime(strftime_buf, sizeof(strftime_buf), "%c", &timeinfo);
ESP_LOGI(TAG, "The current date/time in Shanghai is: %s", strftime_buf);
```

If you need to obtain time with one microsecond resolution, use the code snippet below:

```
struct timeval tv_now;
gettimeofday(&tv_now, NULL);
int64_t time_us = (int64_t)tv_now.tv_sec * 1000000L + (int64_t)tv_now.tv_usec;
```

SNTP Time Synchronization

To set the current time, you can use the POSIX functions `settimeofday()` and `adjtime()`. They are used internally in the lwIP SNTP library to set current time when a response from the NTP server is received. These functions can also be used separately from the lwIP SNTP library.

A function to use inside the lwIP SNTP library depends on a sync mode for system time. Use the function *sntp_set_sync_mode()* to set one of the following sync modes:

- `SNTP_SYNC_MODE_IMMED` (default) updates system time immediately upon receiving a response from the SNTP server after using `settimeofday()`.
- `SNTP_SYNC_MODE_SMOOTH` updates time smoothly by gradually reducing time error using the funcion `adjtime()`. If the difference between the SNTP response time and system time is more than 35 minutes, update system time immediately by using `settimeofday()`.

The lwIP SNTP library has API functions for setting a callback function for a certain event. You might need the following functions:

- `sntp_set_time_sync_notification_cb()` - use it for setting a callback function that will notify of the time synchronization process
- `sntp_get_sync_status()` and `sntp_set_sync_status()` - use it to get/set time synchronization status

To start synchronization via SNTP, just call the following three functions.

```
sntp_setoperatingmode(SNTP_OPMODE_POLL);
sntp_setservername(0, "pool.ntp.org");
sntp_init();
```

An application with this initialization code will periodically synchronize the time. The time synchronization period is determined by `CONFIG_LWIP_SNTP_UPDATE_DELAY` (default value is one hour). To modify the variable, set *CONFIG_LWIP_SNTP_UPDATE_DELAY* in project configuration.

A code example that demonstrates the implementation of time synchronization based on the lwIP SNTP library is provided in protocols/sntp directory.

Timezones

To set local timezone, use the following POSIX functions:

1. Call `setenv()` to set the `TZ` environment variable to the correct value depending on the device location. The format of the time string is the same as described in the GNU libc documentation (although the implementation is different).
2. Call `tzset()` to update C library runtime data for the new time zone.

Once these steps are completed, call the standard C library function `localtime()`, and it will return correct local time taking into account the time zone offset and daylight saving time.

API Reference

Header File

- lwip/include/apps/sntp/sntp.h

Functions

void **sntp_sync_time**(**struct** timeval *_tv_)

This function updates the system time.

This is a weak-linked function. It is possible to replace all SNTP update functionality by placing a sntp_sync_time() function in the app firmware source. If the default implementation is used, calling sntp_set_sync_mode() allows the time synchronization mode to be changed to instant or smooth. If a callback function is registered via sntp_set_time_sync_notification_cb(), it will be called following time synchronization.

Parameters

- `tv`: Time received from SNTP server.

void **sntp_set_sync_mode**(*sntp_sync_mode_t sync_mode*)

Set the sync mode.

Allowable two mode: SNTP_SYNC_MODE_IMMED and SNTP_SYNC_MODE_SMOOTH.

Parameters

- `sync_mode`: Sync mode.

sntp_sync_mode_t **sntp_get_sync_mode**(void)

Get set sync mode.

Return SNTP_SYNC_MODE_IMMED: Update time immediately. SNTP_SYNC_MODE_SMOOTH: Smooth time updating.

sntp_sync_status_t **sntp_get_sync_status**(void)

Get status of time sync.

After the update is completed, the status will be returned as SNTP_SYNC_STATUS_COMPLETED. After that, the status will be reset to SNTP_SYNC_STATUS_RESET. If the update operation is not completed yet, the status will be SNTP_SYNC_STATUS_RESET. If a smooth mode was chosen and the synchronization is still continuing (adjtime works), then it will be SNTP_SYNC_STATUS_IN_PROGRESS.

Return SNTP_SYNC_STATUS_RESET: Reset status. SNTP_SYNC_STATUS_COMPLETED: Time is synchronized. SNTP_SYNC_STATUS_IN_PROGRESS: Smooth time sync in progress.

void **sntp_set_sync_status**(*sntp_sync_status_t sync_status*)

Set status of time sync.

Parameters

- `sync_status`: status of time sync (see sntp_sync_status_t)

void **sntp_set_time_sync_notification_cb**(*sntp_sync_time_cb_t callback*)

Set a callback function for time synchronization notification.

Parameters

- `callback`: a callback function

void **sntp_set_sync_interval**(uint32_t *interval_ms*)

Set the sync interval of SNTP operation.

Note: SNTPv4 RFC 4330 enforces a minimum sync interval of 15 seconds. This sync interval will be used in the next attempt update time throught SNTP. To apply the new sync interval call the sntp_restart() function, otherwise, it will be applied after the last interval expired.

Parameters

- `interval_ms`: The sync interval in ms. It cannot be lower than 15 seconds, otherwise 15 seconds will be set.

uint32_t **sntp_get_sync_interval**(void)

Get the sync interval of SNTP operation.

Return the sync interval

bool **sntp_restart**(void)
Restart SNTP.

Return True - Restart False - SNTP was not initialized yet

Type Definitions

typedef void (***sntp_sync_time_cb_t**)(**struct** timeval *tv)
SNTP callback function for notifying about time sync event.

Parameters

- tv: Time received from SNTP server.

Enumerations

enum sntp_sync_mode_t
SNTP time update mode.

Values:

SNTP_SYNC_MODE_IMMED
Update system time immediately when receiving a response from the SNTP server.

SNTP_SYNC_MODE_SMOOTH
Smooth time updating. Time error is gradually reduced using adjtime function. If the difference between SNTP response time and system time is large (more than 35 minutes) then update immediately.

enum sntp_sync_status_t
SNTP sync status.

Values:

SNTP_SYNC_STATUS_RESET

SNTP_SYNC_STATUS_COMPLETED

SNTP_SYNC_STATUS_IN_PROGRESS

Code examples for this API section are provided in the system directory of ESP-IDF examples.

2.8 Project Configuration

2.8.1 Introduction

ESP-IDF uses kconfiglib which is a Python-based extension to the Kconfig system which provides a compile-time project configuration mechanism. Kconfig is based around options of several types: integer, string, boolean. Kconfig files specify dependencies between options, default values of the options, the way the options are grouped together, etc.

For the complete list of available features please see Kconfig and kconfiglib extentions.

2.8.2 Project Configuration Menu

Application developers can open a terminal-based project configuration menu with the `idf.py menuconfig` build target.

After being updated, this configuration is saved inside `sdkconfig` file in the project root directory. Based on `sdkconfig`, application build targets will generate `sdkconfig.h` file in the build directory, and will make sdkconfig options available to the project build system and source files.

(For the legacy GNU Make build system, the project configuration menu is opened with `make menuconfig`.)

2.8.3 Using sdkconfig.defaults

In some cases, such as when `sdkconfig` file is under revision control, the fact that `sdkconfig` file gets changed by the build system may be inconvenient. The build system offers a way to avoid this, in the form of `sdkconfig.defaults` file. This file is never touched by the build system, and must be created manually. It can contain all the options which matter for the given application. The format is the same as that of the `sdkconfig` file. Once `sdkconfig.defaults` is created, `sdkconfig` can be deleted and added to the ignore list of the revision control system (e.g. `.gitignore` file for git). Project build targets will automatically create `sdkconfig` file, populated with the settings from `sdkconfig.defaults` file, and the rest of the settings will be set to their default values. Note that the build process will not override settings that are already in `sdkconfig` by ones from `sdkconfig.defaults`. For more information, see *自定义 sdkconfig 的默认值*.

2.8.4 Kconfig Formatting Rules

The following attributes of `Kconfig` files are standardized:

- Within any menu, option names should have a consistent prefix. The prefix length is currently set to at least 3 characters.

- The indentation style is 4 characters created by spaces. All sub-items belonging to a parent item are indented by one level deeper. For example, `menu` is indented by 0 characters, the `config` inside of the menu by 4 characters, the help of the `config` by 8 characters and the text of the `help` by 12 characters.
- No trailing spaces are allowed at the end of the lines.
- The maximum length of options is set to 40 characters.
- The maximum length of lines is set to 120 characters.
- Lines cannot be wrapped by backslash (because there is a bug in earlier versions of `conf-idf` which causes that Windows line endings are not recognized after a backslash).

Format checker

`tools/check_kconfigs.py` is provided for checking the `Kconfig` formatting rules. The checker checks all `Kconfig` and `Kconfig.projbuild` files in the ESP-IDF directory and generates a new file with suffix `.new` with some recommendations how to fix issues (if there are any). Please note that the checker cannot correct all rules and the responsibility of the developer is to check and make final corrections in order to pass the tests. For example, indentations will be corrected if there isn' t some misleading previous formatting but it cannot come up with a common prefix for options inside a menu.

2.8.5 Backward Compatibility of Kconfig Options

The standard Kconfig tools ignore unknown options in `sdkconfig`. So if a developer has custom settings for options which are renamed in newer ESP-IDF releases then the given setting for the option would be silently ignored. Therefore, several features have been adopted to avoid this:

1. `confgen.py` is used by the tool chain to pre-process `sdkconfig` files before anything else, for example `menuconfig`, would read them. As the consequence, the settings for old options will be kept and not ignored.
2. `confgen.py` recursively finds all `sdkconfig.rename` files in ESP-IDF directory which contain old and new `Kconfig` option names. Old options are replaced by new ones in the `sdkconfig` file.
3. `confgen.py` post-processes `sdkconfig` files and generates all build outputs (`sdkconfig.h`, `sdkconfig.cmake`, `auto.conf`) by adding a list of compatibility statements, i.e. value of the old option is set the value of the new option (after modification). This is done in order to not break customer codes where old option might still be used.
4. *Deprecated options and their replacements* are automatically generated by `confgen.py`.

2.8.6 Configuration Options Reference

Subsequent sections contain the list of available ESP-IDF options, automatically generated from Kconfig files. Note that depending on the options selected, some options listed here may not be visible by default in

the interface of menuconfig.

By convention, all option names are upper case with underscores. When Kconfig generates sdkconfig and sdkconfig.h files, option names are prefixed with `CONFIG_`. So if an option `ENABLE_FOO` is defined in a Kconfig file and selected in menuconfig, then sdkconfig and sdkconfig.h files will have `CONFIG_ENABLE_FOO` defined. In this reference, option names are also prefixed with `CONFIG_`, same as in the source code.

SDK tool configuration

Contains:

- *CONFIG_SDK_TOOLPREFIX*
- *CONFIG_SDK_PYTHON*
- *CONFIG_SDK_MAKE_WARN_UNDEFINED_VARIABLES*
- *CONFIG_SDK_TOOLCHAIN_SUPPORTS_TIME_WIDE_64_BITS*

CONFIG_SDK_TOOLPREFIX

Compiler toolchain path/prefix

Found in: SDK tool configuration

The prefix/path that is used to call the toolchain. The default setting assumes a crosstool-ng gcc setup that is in your PATH.

CONFIG_SDK_PYTHON

Python interpreter

Found in: SDK tool configuration

The executable name/path that is used to run python.

(Note: This option is used with the legacy GNU Make build system only.)

CONFIG_SDK_MAKE_WARN_UNDEFINED_VARIABLES

'make' warns on undefined variables

Found in: SDK tool configuration

Adds –warn-undefined-variables to MAKEFLAGS. This causes make to print a warning any time an undefined variable is referenced.

This option helps find places where a variable reference is misspelled or otherwise missing, but it can be unwanted if you have Makefiles which depend on undefined variables expanding to an empty string.

(Note: this option is used with the legacy GNU Make build system only.)

CONFIG_SDK_TOOLCHAIN_SUPPORTS_TIME_WIDE_64_BITS

Toolchain supports time_t wide 64-bits

Found in: SDK tool configuration

Enable this option in case you have a custom toolchain which supports time_t wide 64-bits. This option checks time_t is 64-bits and disables ROM time functions to use the time functions from the toolchain instead. This option allows resolving the Y2K38 problem. See "Setup Linux Toolchain from Scratch" to build a custom toolchain which supports 64-bits time_t.

Note: ESP-IDF does not currently come with any pre-compiled toolchain that supports 64-bit wide time_t. This will change in a future major release, but currently 64-bit time_t requires a custom built toolchain.

Build type

Contains:

- *CONFIG_APP_BUILD_TYPE*

CONFIG_APP_BUILD_TYPE

Application build type

Found in: Build type

Select the way the application is built.

By default, the application is built as a binary file in a format compatible with the ESP32 bootloader. In addition to this application, 2nd stage bootloader is also built. Application and bootloader binaries can be written into flash and loaded/executed from there.

Another option, useful for only very small and limited applications, is to only link the .elf file of the application, such that it can be loaded directly into RAM over JTAG. Note that since IRAM and DRAM sizes are very limited, it is not possible to build any complex application this way. However for kinds of testing and debugging, this option may provide faster iterations, since the application does not need to be written into flash. Note that at the moment, ESP-IDF does not contain all the startup code required to initialize the CPUs and ROM memory (data/bss). Therefore it is necessary to execute a bit of ROM code prior to executing the application. A gdbinit file may look as follows:

```
# Connect to a running instance of OpenOCD target remote :3333 # Reset and halt
the target mon reset halt # Run to a specific point in ROM code, # where most of
initialization is complete. thb *0x40007901 c # Load the application into RAM load
# Run till app_main tb app_main c
```

Execute this gdbinit file as follows:

```
xtensa-esp32-elf-gdb build/app-name.elf -x gdbinit
```

Recommended sdkconfig.defaults for building loadable ELF files is as follows. CONFIG_APP_BUILD_TYPE_ELF_RAM is required, other options help reduce application memory footprint.

```
CONFIG_APP_BUILD_TYPE_ELF_RAM=y CONFIG_VFS_SUPPORT_TERMIOS= CONFIG_NEWLIB_NANO_FORMAT=y
CONFIG_ESP32_PANIC_PRINT_HALT=y CONFIG_ESP32_DEBUG_STUBS_ENABLE=
CONFIG_ESP_ERR_TO_NAME_LOOKUP=
```

Available options:

- Default (binary application + 2nd stage bootloader) (APP_BUILD_TYPE_APP_2NDBOOT)
- ELF file, loadable into RAM (EXPERIMENTAL)) (APP_BUILD_TYPE_ELF_RAM)

Application manager

Contains:

- *CONFIG_APP_COMPILE_TIME_DATE*
- *CONFIG_APP_EXCLUDE_PROJECT_VER_VAR*
- *CONFIG_APP_EXCLUDE_PROJECT_NAME_VAR*
- *CONFIG_APP_RETRIEVE_LEN_ELF_SHA*

CONFIG_APP_COMPILE_TIME_DATE

Use time/date stamp for app

Found in: Application manager

If set, then the app will be built with the current time/date stamp. It is stored in the app description structure. If not set, time/date stamp will be excluded from app image. This can be useful for getting the same binary image files made from the same source, but at different times.

CONFIG_APP_EXCLUDE_PROJECT_VER_VAR

Exclude PROJECT_VER from firmware image

Found in: Application manager

The PROJECT_VER variable from the build system will not affect the firmware image. This value will not be contained in the esp_app_desc structure.

CONFIG_APP_EXCLUDE_PROJECT_NAME_VAR

Exclude PROJECT_NAME from firmware image

Found in: Application manager

The PROJECT_NAME variable from the build system will not affect the firmware image. This value will not be contained in the esp_app_desc structure.

CONFIG_APP_RETRIEVE_LEN_ELF_SHA

The length of APP ELF SHA is stored in RAM(chars)

Found in: Application manager

At startup, the app will read this many hex characters from the embedded APP ELF SHA-256 hash value and store it in static RAM. This ensures the app ELF SHA-256 value is always available if it needs to be printed by the panic handler code. Changing this value will change the size of a static buffer, in bytes.

Bootloader config

Contains:

- *CONFIG_BOOTLOADER_COMPILER_OPTIMIZATION*
- *CONFIG_BOOTLOADER_LOG_LEVEL*
- *CONFIG_BOOTLOADER_SPI_WP_PIN*
- *CONFIG_BOOTLOADER_VDDSDIO_BOOST*
- *CONFIG_BOOTLOADER_FACTORY_RESET*
- *CONFIG_BOOTLOADER_APP_TEST*
- *CONFIG_BOOTLOADER_HOLD_TIME_GPIO*
- *CONFIG_BOOTLOADER_WDT_ENABLE*
- *CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE*

- *CONFIG_BOOTLOADER_SKIP_VALIDATE_IN_DEEP_SLEEP*
- *CONFIG_BOOTLOADER_CUSTOM_RESERVE_RTC*

CONFIG_BOOTLOADER_COMPILER_OPTIMIZATION

Bootloader optimization Level

Found in: Bootloader config

This option sets compiler optimization level (gcc -O argument) for the bootloader.

- The default "Size" setting will add the -0s flag to CFLAGS.
- The "Debug" setting will add the -Og flag to CFLAGS.
- The "Performance" setting will add the -O2 flag to CFLAGS.
- The "None" setting will add the -O0 flag to CFLAGS.

Note that custom optimization levels may be unsupported.

Available options:

- Size (-Os) (BOOTLOADER_COMPILER_OPTIMIZATION_SIZE)
- Debug (-Og) (BOOTLOADER_COMPILER_OPTIMIZATION_DEBUG)
- Optimize for performance (-O2) (BOOTLOADER_COMPILER_OPTIMIZATION_PERF)
- Debug without optimization (-O0) (BOOTLOADER_COMPILER_OPTIMIZATION_NONE)

CONFIG_BOOTLOADER_LOG_LEVEL

Bootloader log verbosity

Found in: Bootloader config

Specify how much output to see in bootloader logs.

Available options:

- No output (BOOTLOADER_LOG_LEVEL_NONE)
- Error (BOOTLOADER_LOG_LEVEL_ERROR)
- Warning (BOOTLOADER_LOG_LEVEL_WARN)
- Info (BOOTLOADER_LOG_LEVEL_INFO)
- Debug (BOOTLOADER_LOG_LEVEL_DEBUG)
- Verbose (BOOTLOADER_LOG_LEVEL_VERBOSE)

CONFIG_BOOTLOADER_SPI_WP_PIN

SPI Flash WP Pin when customising pins via eFuse (read help)

Found in: Bootloader config

This value is ignored unless flash mode is set to QIO or QOUT *and* the SPI flash pins have been overriden by setting the eFuses SPI_PAD_CONFIG_xxx.

When this is the case, the eFuse config only defines 3 of the 4 Quad I/O data pins. The WP pin (aka ESP32 pin "SD_DATA_3" or SPI flash pin "IO2") is not specified in eFuse. That pin number is compiled into the bootloader instead.

The default value (GPIO 7) is correct for WP pin on ESP32-D2WD integrated flash.

CONFIG_BOOTLOADER_VDDSDIO_BOOST

VDDSDIO LDO voltage

Found in: Bootloader config

If this option is enabled, and VDDSDIO LDO is set to 1.8V (using eFuse or MTDI bootstrapping pin), bootloader will change LDO settings to output 1.9V instead. This helps prevent flash chip from browning out during flash programming operations.

This option has no effect if VDDSDIO is set to 3.3V, or if the internal VDDSDIO regulator is disabled via eFuse.

Available options:

- 1.8V (BOOTLOADER_VDDSDIO_BOOST_1_8V)
- 1.9V (BOOTLOADER_VDDSDIO_BOOST_1_9V)

CONFIG_BOOTLOADER_FACTORY_RESET

GPIO triggers factory reset

Found in: Bootloader config

Allows to reset the device to factory settings: - clear one or more data partitions; - boot from "factory" partition. The factory reset will occur if there is a GPIO input pulled low while device starts up. See settings below.

CONFIG_BOOTLOADER_NUM_PIN_FACTORY_RESET

Number of the GPIO input for factory reset

Found in: Bootloader config > CONFIG_BOOTLOADER_FACTORY_RESET

The selected GPIO will be configured as an input with internal pull-up enabled. To trigger a factory reset, this GPIO must be pulled low on reset. Note that GPIO34-39 do not have an internal pullup and an external one must be provided.

CONFIG_BOOTLOADER_OTA_DATA_ERASE

Clear OTA data on factory reset (select factory partition)

Found in: Bootloader config > CONFIG_BOOTLOADER_FACTORY_RESET

The device will boot from "factory" partition (or OTA slot 0 if no factory partition is present) after a factory reset.

CONFIG_BOOTLOADER_DATA_FACTORY_RESET

Comma-separated names of partitions to clear on factory reset

Found in: Bootloader config > CONFIG_BOOTLOADER_FACTORY_RESET

Allows customers to select which data partitions will be erased while factory reset.

Specify the names of partitions as a comma-delimited with optional spaces for readability. (Like this: "nvs, phy_init, ⋯") Make sure that the name specified in the partition table and here are the same. Partitions of type "app" cannot be specified here.

CONFIG_BOOTLOADER_APP_TEST

GPIO triggers boot from test app partition

Found in: Bootloader config

Allows to run the test app from "TEST" partition. A boot from "test" partition will occur if there is a GPIO input pulled low while device starts up. See settings below.

CONFIG_BOOTLOADER_NUM_PIN_APP_TEST

Number of the GPIO input to boot TEST partition

Found in: Bootloader config > CONFIG_BOOTLOADER_APP_TEST

The selected GPIO will be configured as an input with internal pull-up enabled. To trigger a test app, this GPIO must be pulled low on reset. After the GPIO input is deactivated and the device reboots, the old application will boot. (factory or OTA[x]). Note that GPIO34-39 do not have an internal pullup and an external one must be provided.

CONFIG_BOOTLOADER_HOLD_TIME_GPIO

Hold time of GPIO for reset/test mode (seconds)

Found in: Bootloader config

The GPIO must be held low continuously for this period of time after reset before a factory reset or test partition boot (as applicable) is performed.

CONFIG_BOOTLOADER_WDT_ENABLE

Use RTC watchdog in start code

Found in: Bootloader config

Tracks the execution time of startup code. If the execution time is exceeded, the RTC_WDT will restart system. It is also useful to prevent a lock up in start code caused by an unstable power source. NOTE: Tracks the execution time starts from the bootloader code - re-set timeout, while selecting the source for slow_clk - and ends calling app_main. Re-set timeout is needed due to WDT uses a SLOW_CLK clock source. After changing a frequency slow_clk a time of WDT needs to re-set for new frequency. slow_clk depends on ESP32_RTC_CLK_SRC (INTERNAL_RC or EXTERNAL_CRYSTAL).

CONFIG_BOOTLOADER_WDT_DISABLE_IN_USER_CODE

Allows RTC watchdog disable in user code

Found in: Bootloader config > CONFIG_BOOTLOADER_WDT_ENABLE

If it is set, the client must itself reset or disable rtc_wdt in their code (app_main()). Otherwise rtc_wdt will be disabled before calling app_main function. Use function rtc_wdt_feed() for resetting counter of rtc_wdt. Use function rtc_wdt_disable() for disabling rtc_wdt.

CONFIG_BOOTLOADER_WDT_TIME_MS

Timeout for RTC watchdog (ms)

Found in: Bootloader config > CONFIG_BOOTLOADER_WDT_ENABLE

Verify that this parameter is correct and more then the execution time. Pay attention to options such as reset to factory, trigger test partition and encryption on boot - these options can increase the execution time. Note: RTC_WDT will reset while encryption operations will be performed.

CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE

Enable app rollback support

Found in: Bootloader config

After updating the app, the bootloader runs a new app with the "ESP_OTA_IMG_PENDING_VERIFY" state set. This state prevents the re-run of this app. After the first boot of the new app in the user code, the function should be called to confirm the operability of the app or vice versa about its non-operability. If the app is working, then it is marked as valid. Otherwise, it is marked as not valid and rolls back to the previous working app. A reboot is performed, and the app is booted before the software update. Note: If during the first boot a new app the power goes out or the WDT works, then roll back will happen. Rollback is possible only between the apps with the same security versions.

CONFIG_BOOTLOADER_APP_ANTI_ROLLBACK

Enable app anti-rollback support

Found in: Bootloader config > CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE

This option prevents rollback to previous firmware/application image with lower security version.

CONFIG_BOOTLOADER_APP_SECURE_VERSION

eFuse secure version of app

Found in: Bootloader config > CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE > CONFIG_BOOTLOADER_APP_ANTI_ROLLBACK

The secure version is the sequence number stored in the header of each firmware. The security version is set in the bootloader, version is recorded in the eFuse field as the number of set ones. The allocated number of bits in the efuse field for storing the security version is limited (see BOOTLOADER_APP_SEC_VER_SIZE_EFUSE_FIELD option).

Bootloader: When bootloader selects an app to boot, an app is selected that has a security version greater or equal that recorded in eFuse field. The app is booted with a higher (or equal) secure version.

The security version is worth increasing if in previous versions there is a significant vulnerability and their use is not acceptable.

Your partition table should has a scheme with ota_0 + ota_1 (without factory).

CONFIG_BOOTLOADER_APP_SEC_VER_SIZE_EFUSE_FIELD

Size of the efuse secure version field

Found in: Bootloader config > CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE > CONFIG_BOOTLOADER_APP_ANTI_ROLLBACK

The size of the efuse secure version field. Its length is limited to 32 bits for ESP32 and 16 bits for ESP32S2BETA. This determines how many times the security version can be increased.

CONFIG_BOOTLOADER_EFUSE_SECURE_VERSION_EMULATE

Emulate operations with efuse secure version(only test)

Found in: Bootloader config > CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE > CONFIG_BOOTLOADER_APP_ANTI_ROLLBACK

This option allow emulate read/write operations with efuse secure version. It allow to test anti-rollback implemention without permanent write eFuse bits. In partition table should be exist this partition *emul_efuse, data, 5, , 0x2000*.

CONFIG_BOOTLOADER_SKIP_VALIDATE_IN_DEEP_SLEEP

Skip image validation when exiting deep sleep

Found in: Bootloader config

This option disables the normal validation of an image coming out of deep sleep (checksums, SHA256, and signature). This is a trade-off between wakeup performance from deep sleep, and image integrity checks.

Only enable this if you know what you are doing. It should not be used in conjunction with using deep_sleep() entry and changing the active OTA partition as this would skip the validation upon first load of the new OTA partition.

CONFIG_BOOTLOADER_CUSTOM_RESERVE_RTC

Reserve RTC FAST memory for custom purposes

Found in: Bootloader config

This option allows the customer to place data in the RTC FAST memory, this area remains valid when rebooted, except for power loss. This memory is located at a fixed address and is available for both the bootloader and the application. (The application and bootoloader must be compiled with the same option). The RTC FAST memory has access only through PRO_CPU.

CONFIG_BOOTLOADER_CUSTOM_RESERVE_RTC_SIZE

Size in bytes for custom purposes

Found in: Bootloader config > CONFIG_BOOTLOADER_CUSTOM_RESERVE_RTC

This option reserves in RTC FAST memory the area for custom purposes. If you want to create your own bootloader and save more information in this area of memory, you can increase it. It must be a multiple of 4 bytes. This area (rtc_retain_mem_t) is reserved and has access from the bootloader and an application.

Security features

Contains:

- *CONFIG_SECURE_SIGNED_APPS_NO_SECURE_BOOT*
- *CONFIG_SECURE_SIGNED_APPS_SCHEME*
- *CONFIG_SECURE_SIGNED_ON_BOOT_NO_SECURE_BOOT*
- *CONFIG_SECURE_SIGNED_ON_UPDATE_NO_SECURE_BOOT*
- *CONFIG_SECURE_BOOT*
- *CONFIG_SECURE_BOOTLOADER_MODE*
- *CONFIG_SECURE_BOOT_BUILD_SIGNED_BINARIES*
- *CONFIG_SECURE_BOOT_VERIFICATION_KEY*
- *CONFIG_SECURE_BOOTLOADER_KEY_ENCODING*
- *CONFIG_SECURE_BOOT_INSECURE*
- *CONFIG_SECURE_FLASH_ENC_ENABLED*
- *Potentially insecure options*

CONFIG_SECURE_SIGNED_APPS_NO_SECURE_BOOT

Require signed app images

Found in: Security features

Require apps to be signed to verify their integrity.

This option uses the same app signature scheme as hardware secure boot, but unlike hardware secure boot it does not prevent the bootloader from being physically updated. This means that the device can be secured against remote network access, but not physical access. Compared to using hardware Secure Boot this option is much simpler to implement.

CONFIG_SECURE_SIGNED_APPS_SCHEME

App Signing Scheme

Found in: Security features

Select the Secure App signing scheme. Depends on the Chip Revision. There are two options: 1. ECDSA based secure boot scheme. (Only choice for Secure Boot V1) Supported in ESP32 and ESP32-ECO3. 2. The RSA based secure boot scheme. (Only choice for Secure Boot V2) Supported in ESP32-ECO3. (ESP32 Chip Revision 3 onwards)

Available options:

- ECDSA (SECURE_SIGNED_APPS_ECDSA_SCHEME)

  Embeds the ECDSA public key in the bootloader and signs the application with an ECDSA key.

  Refer to the documentation before enabling.

- RSA (SECURE_SIGNED_APPS_RSA_SCHEME)

  Appends the RSA-3072 based Signature block to the application. Refer to <Secure Boot Version 2 documentation link> before enabling.

CONFIG_SECURE_SIGNED_ON_BOOT_NO_SECURE_BOOT

Bootloader verifies app signatures

Found in: Security features

If this option is set, the bootloader will be compiled with code to verify that an app is signed before booting it.

If hardware secure boot is enabled, this option is always enabled and cannot be disabled. If hardware secure boot is not enabled, this option doesn't add significant security by itself so most users will want to leave it disabled.

CONFIG_SECURE_SIGNED_ON_UPDATE_NO_SECURE_BOOT

Verify app signature on update

Found in: Security features

If this option is set, any OTA updated apps will have the signature verified before being considered valid.

When enabled, the signature is automatically checked whenever the esp_ota_ops.h APIs are used for OTA updates, or esp_image_format.h APIs are used to verify apps.

If hardware secure boot is enabled, this option is always enabled and cannot be disabled. If hardware secure boot is not enabled, this option still adds significant security against network-based attackers by preventing spoofing of OTA updates.

CONFIG_SECURE_BOOT

Enable hardware Secure Boot in bootloader (READ DOCS FIRST)

Found in: Security features

Build a bootloader which enables Secure Boot on first boot.

Once enabled, Secure Boot will not boot a modified bootloader. The bootloader will only load a partition table or boot an app if the data has a verified digital signature. There are implications for reflashing updated apps once secure boot is enabled.

When enabling secure boot, JTAG and ROM BASIC Interpreter are permanently disabled by default.

CONFIG_SECURE_BOOT_VERSION

Select secure boot version

Found in: Security features > CONFIG_SECURE_BOOT

Select the Secure Boot Version. Depends on the Chip Revision. Secure Boot V2 is the new RSA based secure boot scheme. Supported in ESP32-ECO3. (ESP32 Chip Revision 3 onwards) Secure Boot V1 is the AES based secure boot scheme. Supported in ESP32 and ESP32-ECO3.

Available options:

- Enable Secure Boot version 1 (SECURE_BOOT_V1_ENABLED)

  Build a bootloader which enables secure boot version 1 on first boot. Refer to the Secure Boot section of the ESP-IDF Programmer' s Guide for this version before enabling.

- Enable Secure Boot version 2 (SECURE_BOOT_V2_ENABLED)

  Build a bootloader which enables Secure Boot version 2 on first boot. Refer to Secure Boot V2 section of the ESP-IDF Programmer' s Guide for this version before enabling.

CONFIG_SECURE_BOOTLOADER_MODE

Secure bootloader mode

Found in: Security features

Available options:

- One-time flash (SECURE_BOOTLOADER_ONE_TIME_FLASH)

  On first boot, the bootloader will generate a key which is not readable externally or by software. A digest is generated from the bootloader image itself. This digest will be verified on each subsequent boot.

  Enabling this option means that the bootloader cannot be changed after the first time it is booted.

- Reflashable (SECURE_BOOTLOADER_REFLASHABLE)

  Generate a reusable secure bootloader key, derived (via SHA-256) from the secure boot signing key.

  This allows the secure bootloader to be re-flashed by anyone with access to the secure boot signing key.

  This option is less secure than one-time flash, because a leak of the digest key from one device allows reflashing of any device that uses it.

CONFIG_SECURE_BOOT_BUILD_SIGNED_BINARIES

Sign binaries during build

Found in: Security features

Once secure boot or signed app requirement is enabled, app images are required to be signed.

If enabled (default), these binary files are signed as part of the build process. The file named in "Secure boot private signing key" will be used to sign the image.

If disabled, unsigned app/partition data will be built. They must be signed manually using espsecure.py. Version 1 to enable ECDSA Based Secure Boot and Version 2 to enable RSA based Secure Boot. (for example, on a remote signing server.)

CONFIG_SECURE_BOOT_SIGNING_KEY

Secure boot private signing key

Found in: Security features > CONFIG_SECURE_BOOT_BUILD_SIGNED_BINARIES

Path to the key file used to sign app images.

Key file is an ECDSA private key (NIST256p curve) in PEM format for Secure Boot V1. Key file is an RSA private key in PEM format for Secure Boot V2.

Path is evaluated relative to the project directory.

You can generate a new signing key by running the following command: espsecure.py generate_signing_key secure_boot_signing_key.pem

See the Secure Boot section of the ESP-IDF Programmer' s Guide for this version for details.

CONFIG_SECURE_BOOT_VERIFICATION_KEY

Secure boot public signature verification key

Found in: Security features

Path to a public key file used to verify signed images. Secure Boot V1: This ECDSA public key is compiled into the bootloader and/or app, to verify app images. Secure Boot V2: This RSA public key is compiled into the signature block at the end of the bootloader/app.

Key file is in raw binary format, and can be extracted from a PEM formatted private key using the espsecure.py extract_public_key command.

Refer to the Secure Boot section of the ESP-IDF Programmer' s Guide for this version before enabling.

CONFIG_SECURE_BOOTLOADER_KEY_ENCODING

Hardware Key Encoding

Found in: Security features

In reflashable secure bootloader mode, a hardware key is derived from the signing key (with SHA-256) and can be written to eFuse with espefuse.py.

Normally this is a 256-bit key, but if 3/4 Coding Scheme is used on the device then the eFuse key is truncated to 192 bits.

This configuration item doesn' t change any firmware code, it only changes the size of key binary which is generated at build time.

Available options:

- No encoding (256 bit key) (SECURE_BOOTLOADER_KEY_ENCODING_256BIT)
- 3/4 encoding (192 bit key) (SECURE_BOOTLOADER_KEY_ENCODING_192BIT)

CONFIG_SECURE_BOOT_INSECURE

Allow potentially insecure options

Found in: Security features

You can disable some of the default protections offered by secure boot, in order to enable testing or a custom combination of security features.

Only enable these options if you are very sure.

Refer to the Secure Boot section of the ESP-IDF Programmer' s Guide for this version before enabling.

CONFIG_SECURE_FLASH_ENC_ENABLED

Enable flash encryption on boot (READ DOCS FIRST)

Found in: Security features

If this option is set, flash contents will be encrypted by the bootloader on first boot.

Note: After first boot, the system will be permanently encrypted. Re-flashing an encrypted system is complicated and not always possible.

Read *Flash 加密* before enabling.

CONFIG_SECURE_FLASH_ENCRYPTION_KEYSIZE

Size of generated AES-XTS key

Found in: Security features > CONFIG_SECURE_FLASH_ENC_ENABLED

Size of generated AES-XTS key.

AES-128 uses a 256-bit key (32 bytes) which occupies one Efuse key block. AES-256 uses a 512-bit key (64 bytes) which occupies two Efuse key blocks.

This setting is ignored if either type of key is already burned to Efuse before the first boot. In this case, the pre-burned key is used and no new key is generated.

Available options:

- AES-128 (256-bit key) (SECURE_FLASH_ENCRYPTION_AES128)
- AES-256 (512-bit key) (SECURE_FLASH_ENCRYPTION_AES256)

CONFIG_SECURE_FLASH_ENCRYPTION_MODE

Enable usage mode

Found in: Security features > CONFIG_SECURE_FLASH_ENC_ENABLED

By default Development mode is enabled which allows UART bootloader to perform flash encryption operations

Select Release mode only for production or manufacturing. Once enabled you can not reflash using UART bootloader

Refer to the Secure Boot section of the ESP-IDF Programmer' s Guide for this version and *Flash 加密* for details.

Available options:

- Development(NOT SECURE) (SECURE_FLASH_ENCRYPTION_MODE_DEVELOPMENT)
- Release (SECURE_FLASH_ENCRYPTION_MODE_RELEASE)

Potentially insecure options

Contains:

- *CONFIG_SECURE_BOOT_ALLOW_ROM_BASIC*
- *CONFIG_SECURE_BOOT_ALLOW_JTAG*
- *CONFIG_SECURE_BOOT_ALLOW_SHORT_APP_PARTITION*
- *CONFIG_SECURE_BOOT_V2_ALLOW_EFUSE_RD_DIS*
- *CONFIG_SECURE_FLASH_UART_BOOTLOADER_ALLOW_ENC*
- *CONFIG_SECURE_FLASH_UART_BOOTLOADER_ALLOW_DEC*
- *CONFIG_SECURE_FLASH_UART_BOOTLOADER_ALLOW_CACHE*
- *CONFIG_SECURE_FLASH_REQUIRE_ALREADY_ENABLED*

CONFIG_SECURE_BOOT_ALLOW_ROM_BASIC

Leave ROM BASIC Interpreter available on reset

Found in: Security features > Potentially insecure options

By default, the BASIC ROM Console starts on reset if no valid bootloader is read from the flash.

When either flash encryption or secure boot are enabled, the default is to disable this BASIC fallback mode permanently via eFuse.

If this option is set, this eFuse is not burned and the BASIC ROM Console may remain accessible. Only set this option in testing environments.

CONFIG_SECURE_BOOT_ALLOW_JTAG

Allow JTAG Debugging

Found in: Security features > Potentially insecure options

If not set (default), the bootloader will permanently disable JTAG (across entire chip) on first boot when either secure boot or flash encryption is enabled.

Setting this option leaves JTAG on for debugging, which negates all protections of flash encryption and some of the protections of secure boot.

Only set this option in testing environments.

CONFIG_SECURE_BOOT_ALLOW_SHORT_APP_PARTITION

Allow app partition length not 64KB aligned

Found in: Security features > Potentially insecure options

If not set (default), app partition size must be a multiple of 64KB. App images are padded to 64KB length, and the bootloader checks any trailing bytes after the signature (before the next 64KB boundary) have not been written. This is because flash cache maps entire 64KB pages into the address space. This prevents an attacker from appending unverified data after the app image in the flash, causing it to be mapped into the address space.

Setting this option allows the app partition length to be unaligned, and disables padding of the app image to this length. It is generally not recommended to set this option, unless you have a legacy partitioning scheme which doesn't support 64KB aligned partition lengths.

CONFIG_SECURE_BOOT_V2_ALLOW_EFUSE_RD_DIS

Allow additional read protecting of efuses

Found in: Security features > Potentially insecure options

If not set (default, recommended), on first boot the bootloader will burn the WR_DIS_RD_DIS efuse when Secure Boot is enabled. This prevents any more efuses from being read protected.

If this option is set, it will remain possible to write the EFUSE_RD_DIS efuse field after Secure Boot is enabled. This may allow an attacker to read-protect the BLK2 efuse holding the public key digest, causing an immediate denial of service and possibly allowing an additional fault injection attack to bypass the signature protection.

CONFIG_SECURE_FLASH_UART_BOOTLOADER_ALLOW_ENC

Leave UART bootloader encryption enabled

Found in: Security features > Potentially insecure options

If not set (default), the bootloader will permanently disable UART bootloader encryption access on first boot. If set, the UART bootloader will still be able to access hardware encryption.

It is recommended to only set this option in testing environments.

CONFIG_SECURE_FLASH_UART_BOOTLOADER_ALLOW_DEC

Leave UART bootloader decryption enabled

Found in: Security features > Potentially insecure options

If not set (default), the bootloader will permanently disable UART bootloader decryption access on first boot. If set, the UART bootloader will still be able to access hardware decryption.

Only set this option in testing environments. Setting this option allows complete bypass of flash encryption.

CONFIG_SECURE_FLASH_UART_BOOTLOADER_ALLOW_CACHE

Leave UART bootloader flash cache enabled

Found in: Security features > Potentially insecure options

If not set (default), the bootloader will permanently disable UART bootloader flash cache access on first boot. If set, the UART bootloader will still be able to access the flash cache.

Only set this option in testing environments.

CONFIG_SECURE_FLASH_REQUIRE_ALREADY_ENABLED

Require flash encryption to be already enabled

Found in: Security features > Potentially insecure options

If not set (default), and flash encryption is not yet enabled in eFuses, the 2nd stage bootloader will enable flash encryption: generate the flash encryption key and program eFuses. If this option is set, and flash encryption is not yet enabled, the bootloader will error out and reboot. If flash encryption is enabled in eFuses, this option does not change the bootloader behavior.

Only use this option in testing environments, to avoid accidentally enabling flash encryption on the wrong device. The device needs to have flash encryption already enabled using espefuse.py.

Serial flasher config

Contains:

- *CONFIG_ESPTOOLPY_PORT*
- *CONFIG_ESPTOOLPY_BAUD*
- *CONFIG_ESPTOOLPY_BAUD_OTHER_VAL*
- *CONFIG_ESPTOOLPY_COMPRESSED*
- *CONFIG_ESPTOOLPY_FLASHMODE*
- *CONFIG_ESPTOOLPY_FLASHFREQ*
- *CONFIG_ESPTOOLPY_FLASHSIZE*

- *CONFIG_ESPTOOLPY_FLASHSIZE_DETECT*
- *CONFIG_ESPTOOLPY_BEFORE*
- *CONFIG_ESPTOOLPY_AFTER*
- *CONFIG_ESPTOOLPY_MONITOR_BAUD*
- *CONFIG_ESPTOOLPY_MONITOR_BAUD_OTHER_VAL*

CONFIG_ESPTOOLPY_PORT

Default serial port

Found in: Serial flasher config

The serial port that' s connected to the ESP chip. This can be overridden by setting the ESPPORT environment variable.

This value is ignored when using the CMake-based build system or idf.py.

CONFIG_ESPTOOLPY_BAUD

Default baud rate

Found in: Serial flasher config

Default baud rate to use while communicating with the ESP chip. Can be overridden by setting the ESPBAUD variable.

This value is ignored when using the CMake-based build system or idf.py.

Available options:

- 115200 baud (ESPTOOLPY_BAUD_115200B)
- 230400 baud (ESPTOOLPY_BAUD_230400B)
- 921600 baud (ESPTOOLPY_BAUD_921600B)
- 2Mbaud (ESPTOOLPY_BAUD_2MB)
- Other baud rate (ESPTOOLPY_BAUD_OTHER)

CONFIG_ESPTOOLPY_BAUD_OTHER_VAL

Other baud rate value

Found in: Serial flasher config

CONFIG_ESPTOOLPY_COMPRESSED

Use compressed upload

Found in: Serial flasher config

The flasher tool can send data compressed using zlib, letting the ROM on the ESP chip decompress it on the fly before flashing it. For most payloads, this should result in a speed increase.

CONFIG_ESPTOOLPY_FLASHMODE

Flash SPI mode

Found in: Serial flasher config

Mode the flash chip is flashed in, as well as the default mode for the binary to run in.

Available options:

- QIO (ESPTOOLPY_FLASHMODE_QIO)
- QOUT (ESPTOOLPY_FLASHMODE_QOUT)
- DIO (ESPTOOLPY_FLASHMODE_DIO)
- DOUT (ESPTOOLPY_FLASHMODE_DOUT)

CONFIG_ESPTOOLPY_FLASHFREQ

Flash SPI speed

Found in: Serial flasher config

The SPI flash frequency to be used.

Available options:

- 80 MHz (ESPTOOLPY_FLASHFREQ_80M)
- 40 MHz (ESPTOOLPY_FLASHFREQ_40M)
- 26 MHz (ESPTOOLPY_FLASHFREQ_26M)
- 20 MHz (ESPTOOLPY_FLASHFREQ_20M)

CONFIG_ESPTOOLPY_FLASHSIZE

Flash size

Found in: Serial flasher config

SPI flash size, in megabytes

Available options:

- 1 MB (ESPTOOLPY_FLASHSIZE_1MB)
- 2 MB (ESPTOOLPY_FLASHSIZE_2MB)
- 4 MB (ESPTOOLPY_FLASHSIZE_4MB)
- 8 MB (ESPTOOLPY_FLASHSIZE_8MB)
- 16 MB (ESPTOOLPY_FLASHSIZE_16MB)

CONFIG_ESPTOOLPY_FLASHSIZE_DETECT

Detect flash size when flashing bootloader

Found in: Serial flasher config

If this option is set, flashing the project will automatically detect the flash size of the target chip and update the bootloader image before it is flashed.

CONFIG_ESPTOOLPY_BEFORE

Before flashing

Found in: Serial flasher config

Configure whether esptool.py should reset the ESP32 before flashing.

Automatic resetting depends on the RTS & DTR signals being wired from the serial port to the ESP32. Most USB development boards do this internally.

Available options:

- Reset to bootloader (ESPTOOLPY_BEFORE_RESET)
- No reset (ESPTOOLPY_BEFORE_NORESET)

CONFIG_ESPTOOLPY_AFTER

After flashing

Found in: Serial flasher config

Configure whether esptool.py should reset the ESP32 after flashing.

Automatic resetting depends on the RTS & DTR signals being wired from the serial port to the ESP32. Most USB development boards do this internally.

Available options:

- Reset after flashing (ESPTOOLPY_AFTER_RESET)

- Stay in bootloader (ESPTOOLPY_AFTER_NORESET)

CONFIG_ESPTOOLPY_MONITOR_BAUD

'idf.py monitor' baud rate

Found in: Serial flasher config

Baud rate to use when running 'idf.py monitor' or 'make monitor' to view serial output from a running chip.

Can override by setting the MONITORBAUD environment variable.

Available options:

- 9600 bps (ESPTOOLPY_MONITOR_BAUD_9600B)
- 57600 bps (ESPTOOLPY_MONITOR_BAUD_57600B)
- 115200 bps (ESPTOOLPY_MONITOR_BAUD_115200B)
- 230400 bps (ESPTOOLPY_MONITOR_BAUD_230400B)
- 921600 bps (ESPTOOLPY_MONITOR_BAUD_921600B)
- 2 Mbps (ESPTOOLPY_MONITOR_BAUD_2MB)
- Custom baud rate (ESPTOOLPY_MONITOR_BAUD_OTHER)

CONFIG_ESPTOOLPY_MONITOR_BAUD_OTHER_VAL

Custom baud rate value

Found in: Serial flasher config

Partition Table

Contains:

- *CONFIG_PARTITION_TABLE_TYPE*
- *CONFIG_PARTITION_TABLE_CUSTOM_FILENAME*
- *CONFIG_PARTITION_TABLE_OFFSET*
- *CONFIG_PARTITION_TABLE_MD5*

CONFIG_PARTITION_TABLE_TYPE

Partition Table

Found in: Partition Table

The partition table to flash to the ESP32. The partition table determines where apps, data and other resources are expected to be found.

The predefined partition table CSV descriptions can be found in the components/partition_table directory. Otherwise it's possible to create a new custom partition CSV for your application.

Available options:

- Single factory app, no OTA (PARTITION_TABLE_SINGLE_APP)
- Factory app, two OTA definitions (PARTITION_TABLE_TWO_OTA)
- Custom partition table CSV (PARTITION_TABLE_CUSTOM)

CONFIG_PARTITION_TABLE_CUSTOM_FILENAME

Custom partition CSV file

Found in: Partition Table

Name of the custom partition CSV filename. This path is evaluated relative to the project root directory.

CONFIG_PARTITION_TABLE_OFFSET

Offset of partition table

Found in: Partition Table

The address of partition table (by default 0x8000). Allows you to move the partition table, it gives more space for the bootloader. Note that the bootloader and app will both need to be compiled with the same PARTITION_TABLE_OFFSET value.

This number should be a multiple of 0x1000.

Note that partition offsets in the partition table CSV file may need to be changed if this value is set to a higher value. To have each partition offset adapt to the configured partition table offset, leave all partition offsets blank in the CSV file.

CONFIG_PARTITION_TABLE_MD5

Generate an MD5 checksum for the partition table

Found in: Partition Table

Generate an MD5 checksum for the partition table for protecting the integrity of the table. The generation should be turned off for legacy bootloaders which cannot recognize the MD5 checksum in the partition table.

Compiler options

Contains:

- *CONFIG_COMPILER_OPTIMIZATION*
- *CONFIG_COMPILER_OPTIMIZATION_ASSERTION_LEVEL*
- *CONFIG_COMPILER_CXX_EXCEPTIONS*
- *CONFIG_COMPILER_CXX_RTTI*
- *CONFIG_COMPILER_STACK_CHECK_MODE*
- *CONFIG_COMPILER_WARN_WRITE_STRINGS*
- *CONFIG_COMPILER_DISABLE_GCC8_WARNINGS*

CONFIG_COMPILER_OPTIMIZATION

Optimization Level

Found in: Compiler options

This option sets compiler optimization level (gcc -O argument) for the app.

- The "Default" setting will add the -0g flag to CFLAGS.
- The "Size" setting will add the -0s flag to CFLAGS.
- The "Performance" setting will add the -O2 flag to CFLAGS.
- The "None" setting will add the -O0 flag to CFLAGS.

The "Size" setting cause the compiled code to be smaller and faster, but may lead to difficulties of correlating code addresses to source file lines when debugging.

The "Performance" setting causes the compiled code to be larger and faster, but will be easier to correlated code addresses to source file lines.

"None" with -O0 produces compiled code without optimization.

Note that custom optimization levels may be unsupported.

Compiler optimization for the IDF bootloader is set separately, see the BOOTLOADER_COMPILER_OPTIMIZATION setting.

Available options:

- Debug (-Og) (COMPILER_OPTIMIZATION_DEFAULT)
- Optimize for size (-Os) (COMPILER_OPTIMIZATION_SIZE)
- Optimize for performance (-O2) (COMPILER_OPTIMIZATION_PERF)
- Debug without optimization (-O0) (COMPILER_OPTIMIZATION_NONE)

CONFIG_COMPILER_OPTIMIZATION_ASSERTION_LEVEL

Assertion level

Found in: Compiler options

Assertions can be:

- Enabled. Failure will print verbose assertion details. This is the default.
- Set to "silent" to save code size (failed assertions will abort() but user needs to use the aborting address to find the line number with the failed assertion.)
- Disabled entirely (not recommended for most configurations.) -DNDEBUG is added to CPPFLAGS in this case.

Available options:

- Enabled (COMPILER_OPTIMIZATION_ASSERTIONS_ENABLE)

  Enable assertions. Assertion content and line number will be printed on failure.

- Silent (saves code size) (COMPILER_OPTIMIZATION_ASSERTIONS_SILENT)

  Enable silent assertions. Failed assertions will abort(), user needs to use the aborting address to find the line number with the failed assertion.

- Disabled (sets -DNDEBUG) (COMPILER_OPTIMIZATION_ASSERTIONS_DISABLE)

  If assertions are disabled, -DNDEBUG is added to CPPFLAGS.

CONFIG_COMPILER_CXX_EXCEPTIONS

Enable C++ exceptions

Found in: Compiler options

Enabling this option compiles all IDF C++ files with exception support enabled.

Disabling this option disables C++ exception support in all compiled files, and any libstdc++ code which throws an exception will abort instead.

Enabling this option currently adds an additional ~500 bytes of heap overhead when an exception is thrown in user code for the first time.

Contains:

- *CONFIG_COMPILER_CXX_EXCEPTIONS_EMG_POOL_SIZE*

CONFIG_COMPILER_CXX_EXCEPTIONS_EMG_POOL_SIZE

Emergency Pool Size

Found in: Compiler options > CONFIG_COMPILER_CXX_EXCEPTIONS

Size (in bytes) of the emergency memory pool for C++ exceptions. This pool will be used to allocate memory for thrown exceptions when there is not enough memory on the heap.

CONFIG_COMPILER_CXX_RTTI

Enable C++ run-time type info (RTTI)

Found in: Compiler options

Enabling this option compiles all C++ files with RTTI support enabled. This increases binary size (typically by tens of kB) but allows using dynamic_cast conversion and typeid operator.

CONFIG_COMPILER_STACK_CHECK_MODE

Stack smashing protection mode

Found in: Compiler options

Stack smashing protection mode. Emit extra code to check for buffer overflows, such as stack smashing attacks. This is done by adding a guard variable to functions with vulnerable objects. The guards are initialized when a function is entered and then checked when the function exits. If a guard check fails, program is halted. Protection has the following modes:

- In NORMAL mode (GCC flag: -fstack-protector) only functions that call alloca, and functions with buffers larger than 8 bytes are protected.
- STRONG mode (GCC flag: -fstack-protector-strong) is like NORMAL, but includes additional functions to be protected – those that have local array definitions, or have references to local frame addresses.
- In OVERALL mode (GCC flag: -fstack-protector-all) all functions are protected.

Modes have the following impact on code performance and coverage:

- performance: NORMAL > STRONG > OVERALL
- coverage: NORMAL < STRONG < OVERALL

Available options:

- None (COMPILER_STACK_CHECK_MODE_NONE)
- Normal (COMPILER_STACK_CHECK_MODE_NORM)
- Strong (COMPILER_STACK_CHECK_MODE_STRONG)
- Overall (COMPILER_STACK_CHECK_MODE_ALL)

CONFIG_COMPILER_WARN_WRITE_STRINGS

Enable -Wwrite-strings warning flag

Found in: Compiler options

Adds -Wwrite-strings flag for the C/C++ compilers.

For C, this gives string constants the type `const char[]` so that copying the address of one into a non-const `char \*` pointer produces a warning. This warning helps to find at compile time code that tries to write into a string constant.

For C++, this warns about the deprecated conversion from string literals to `char \*`.

CONFIG_COMPILER_DISABLE_GCC8_WARNINGS

Disable new warnings introduced in GCC 6 - 8

Found in: Compiler options

Enable this option if using GCC 6 or newer, and wanting to disable warnings which don' t appear with GCC 5.

Component config

Contains:

- *Application Level Tracing*
- *Bluetooth*
- *CONFIG_BLE_MESH*
- *CoAP Configuration*
- *Driver configurations*
- *eFuse Bit Manager*
- *ESP-TLS*
- *ESP32-specific*
- *Power Management*

- *ADC-Calibration*
- *Common ESP-related*
- *Ethernet*
- *Event Loop Library*
- *GDB Stub*
- *ESP HTTP client*
- *HTTP Server*
- *ESP HTTPS OTA*
- *ESP HTTPS server*
- *ESP NETIF Adapter*
- *Wi-Fi*
- *PHY*
- *Core dump*
- *FAT Filesystem support*
- *Modbus configuration*
- *FreeRTOS*
- *Heap memory debugging*
- *jsmn*
- *libsodium*
- *Log output*
- *LWIP*
- *mbedTLS*
- *mDNS*
- *ESP-MQTT Configurations*
- *Newlib*
- *NVS*
- *OpenSSL*
- *PThreads*
- *SPI Flash driver*
- *SPIFFS Configuration*

- *Unity unit testing library*
- *Virtual file system*
- *Wear Levelling*
- *Wi-Fi Provisioning Manager*
- *Supplicant*

Application Level Tracing

Contains:

- *CONFIG_APPTRACE_DESTINATION*
- *CONFIG_APPTRACE_ONPANIC_HOST_FLUSH_TMO*
- *CONFIG_APPTRACE_POSTMORTEM_FLUSH_THRESH*
- *CONFIG_APPTRACE_PENDING_DATA_SIZE_MAX*
- *FreeRTOS SystemView Tracing*
- *CONFIG_APPTRACE_GCOV_ENABLE*

CONFIG_APPTRACE_DESTINATION

Data Destination

Found in: Component config > Application Level Tracing

Select destination for application trace: trace memory or none (to disable).

Available options:

- Trace memory (APPTRACE_DEST_TRAX)
- None (APPTRACE_DEST_NONE)

CONFIG_APPTRACE_ONPANIC_HOST_FLUSH_TMO

Timeout for flushing last trace data to host on panic

Found in: Component config > Application Level Tracing

Timeout for flushing last trace data to host in case of panic. In ms. Use -1 to disable timeout and wait forever.

CONFIG_APPTRACE_POSTMORTEM_FLUSH_THRESH

Threshold for flushing last trace data to host on panic

Found in: Component config > Application Level Tracing

Threshold for flushing last trace data to host on panic in post-mortem mode. This is minimal amount of data needed to perform flush. In bytes.

CONFIG_APPTRACE_PENDING_DATA_SIZE_MAX

Size of the pending data buffer

Found in: Component config > Application Level Tracing

Size of the buffer for events in bytes. It is useful for buffering events from the time critical code (scheduler, ISRs etc). If this parameter is 0 then events will be discarded when main HW buffer is full.

FreeRTOS SystemView Tracing

Contains:

- *CONFIG_SYSVIEW_ENABLE*

CONFIG_SYSVIEW_ENABLE

SystemView Tracing Enable

Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing

Enables supporrt for SEGGER SystemView tracing functionality.

CONFIG_SYSVIEW_TS_SOURCE

Timer to use as timestamp source

Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing > CONFIG_SYSVIEW_ENABLE

SystemView needs to use a hardware timer as the source of timestamps when tracing. This option selects the timer for it.

Available options:

- CPU cycle counter (CCOUNT) (SYSVIEW_TS_SOURCE_CCOUNT)
- Timer 0, Group 0 (SYSVIEW_TS_SOURCE_TIMER_00)

- Timer 1, Group 0 (SYSVIEW_TS_SOURCE_TIMER_01)
- Timer 0, Group 1 (SYSVIEW_TS_SOURCE_TIMER_10)
- Timer 1, Group 1 (SYSVIEW_TS_SOURCE_TIMER_11)
- esp_timer high resolution timer (SYSVIEW_TS_SOURCE_ESP_TIMER)

CONFIG_SYSVIEW_MAX_TASKS

Maximum supported tasks

Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing > CONFIG_SYSVIEW_ENABLE

Configures maximum supported tasks in sysview debug

CONFIG_SYSVIEW_BUF_WAIT_TMO

Trace buffer wait timeout

Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing > CONFIG_SYSVIEW_ENABLE

Configures timeout (in us) to wait for free space in trace buffer. Set to -1 to wait forever and avoid lost events.

CONFIG_SYSVIEW_EVT_OVERFLOW_ENABLE

Trace Buffer Overflow Event

Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing > CONFIG_SYSVIEW_ENABLE

Enables "Trace Buffer Overflow" event.

CONFIG_SYSVIEW_EVT_ISR_ENTER_ENABLE

ISR Enter Event

Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing > CONFIG_SYSVIEW_ENABLE

Enables "ISR Enter" event.

CONFIG_SYSVIEW_EVT_ISR_EXIT_ENABLE

ISR Exit Event

Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing > CONFIG_SYSVIEW_ENABLE

Enables "ISR Exit" event.

CONFIG_SYSVIEW_EVT_ISR_TO_SCHEDULER_ENABLE

ISR Exit to Scheduler Event

Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing > CONFIG_SYSVIEW_ENABLE

Enables "ISR to Scheduler" event.

CONFIG_SYSVIEW_EVT_TASK_START_EXEC_ENABLE

Task Start Execution Event

Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing > CONFIG_SYSVIEW_ENABLE

Enables "Task Start Execution" event.

CONFIG_SYSVIEW_EVT_TASK_STOP_EXEC_ENABLE

Task Stop Execution Event

Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing > CONFIG_SYSVIEW_ENABLE

Enables "Task Stop Execution" event.

CONFIG_SYSVIEW_EVT_TASK_START_READY_ENABLE

Task Start Ready State Event

Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing > CONFIG_SYSVIEW_ENABLE

Enables "Task Start Ready State" event.

CONFIG_SYSVIEW_EVT_TASK_STOP_READY_ENABLE

Task Stop Ready State Event

Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing > CONFIG_SYSVIEW_ENABLE

Enables "Task Stop Ready State" event.

CONFIG_SYSVIEW_EVT_TASK_CREATE_ENABLE

Task Create Event

Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing > CONFIG_SYSVIEW_ENABLE

Enables "Task Create" event.

CONFIG_SYSVIEW_EVT_TASK_TERMINATE_ENABLE

Task Terminate Event

Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing > CONFIG_SYSVIEW_ENABLE

Enables "Task Terminate" event.

CONFIG_SYSVIEW_EVT_IDLE_ENABLE

System Idle Event

Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing > CONFIG_SYSVIEW_ENABLE

Enables "System Idle" event.

CONFIG_SYSVIEW_EVT_TIMER_ENTER_ENABLE

Timer Enter Event

Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing > CONFIG_SYSVIEW_ENABLE

Enables "Timer Enter" event.

CONFIG_SYSVIEW_EVT_TIMER_EXIT_ENABLE

Timer Exit Event

Found in: Component config > Application Level Tracing > FreeRTOS SystemView Tracing > CONFIG_SYSVIEW_ENABLE

Enables "Timer Exit" event.

CONFIG_APPTRACE_GCOV_ENABLE

GCOV to Host Enable

Found in: Component config > Application Level Tracing

Enables support for GCOV data transfer to host.

Bluetooth

Contains:

- *CONFIG_BT_ENABLED*
- *Bluetooth controller*
- *CONFIG_BT_HOST*
- *Bluedroid Options*
- *NimBLE Options*

CONFIG_BT_ENABLED

Bluetooth

Found in: Component config > Bluetooth

Select this option to enable Bluetooth and show the submenu with Bluetooth configuration choices.

Bluetooth controller

Contains:

- *CONFIG_BTDM_CTRL_MODE*
- *CONFIG_BTDM_CTRL_BLE_MAX_CONN*
- *CONFIG_BTDM_CTRL_BR_EDR_MAX_ACL_CONN*

- *CONFIG_BTDM_CTRL_BR_EDR_MAX_SYNC_CONN*
- *CONFIG_BTDM_CTRL_BR_EDR_SCO_DATA_PATH*
- *CONFIG_BTDM_CTRL_AUTO_LATENCY*
- *CONFIG_BTDM_CTRL_PINNED_TO_CORE_CHOICE*
- *CONFIG_BTDM_CTRL_HCI_MODE_CHOICE*
- *HCI UART(H4) Options*
- *MODEM SLEEP Options*
- *CONFIG_BTDM_BLE_SLEEP_CLOCK_ACCURACY*
- *CONFIG_BTDM_BLE_SCAN_DUPL*
- *CONFIG_BTDM_CTRL_FULL_SCAN_SUPPORTED*
- *CONFIG_BTDM_BLE_ADV_REPORT_FLOW_CTRL_SUPP*
- *CONFIG_BTDM_COEX_BT_OPTIONS*

CONFIG_BTDM_CTRL_MODE

Bluetooth controller mode (BR/EDR/BLE/DUALMODE)

Found in: Component config > Bluetooth > Bluetooth controller

Specify the bluetooth controller mode (BR/EDR, BLE or dual mode).

Available options:

- BLE Only (BTDM_CTRL_MODE_BLE_ONLY)
- BR/EDR Only (BTDM_CTRL_MODE_BR_EDR_ONLY)
- Bluetooth Dual Mode (BTDM_CTRL_MODE_BTDM)

CONFIG_BTDM_CTRL_BLE_MAX_CONN

BLE Max Connections

Found in: Component config > Bluetooth > Bluetooth controller

BLE maximum connections of bluetooth controller. Each connection uses 1KB static DRAM whenever the BT controller is enabled.

CONFIG_BTDM_CTRL_BR_EDR_MAX_ACL_CONN

BR/EDR ACL Max Connections

Found in: Component config > Bluetooth > Bluetooth controller

BR/EDR ACL maximum connections of bluetooth controller. Each connection uses 1.2KB static DRAM whenever the BT controller is enabled.

CONFIG_BTDM_CTRL_BR_EDR_MAX_SYNC_CONN

BR/EDR Sync(SCO/eSCO) Max Connections

Found in: Component config > Bluetooth > Bluetooth controller

BR/EDR Synchronize maximum connections of bluetooth controller. Each connection uses 2KB static DRAM whenever the BT controller is enabled.

CONFIG_BTDM_CTRL_BR_EDR_SCO_DATA_PATH

BR/EDR Sync(SCO/eSCO) default data path

Found in: Component config > Bluetooth > Bluetooth controller

SCO data path, i.e. HCI or PCM. SCO data can be sent/received through HCI synchronous packets, or the data can be routed to on-chip PCM module on ESP32. PCM input/output signals can be "matrixed" to GPIOs. The default data path can also be set using API "esp_bredr_sco_datapath_set"

Available options:

- HCI (BTDM_CTRL_BR_EDR_SCO_DATA_PATH_HCI)
- PCM (BTDM_CTRL_BR_EDR_SCO_DATA_PATH_PCM)

CONFIG_BTDM_CTRL_AUTO_LATENCY

Auto latency

Found in: Component config > Bluetooth > Bluetooth controller

BLE auto latency, used to enhance classic BT performance while classic BT and BLE are enabled at the same time.

CONFIG_BTDM_CTRL_PINNED_TO_CORE_CHOICE

The cpu core which bluetooth controller run

Found in: Component config > Bluetooth > Bluetooth controller

Specify the cpu core to run bluetooth controller. Can not specify no-affinity.

Available options:

- Core 0 (PRO CPU) (BTDM_CTRL_PINNED_TO_CORE_0)
- Core 1 (APP CPU) (BTDM_CTRL_PINNED_TO_CORE_1)

CONFIG_BTDM_CTRL_HCI_MODE_CHOICE

HCI mode

Found in: Component config > Bluetooth > Bluetooth controller

Speicify HCI mode as VHCI or UART(H4)

Available options:

- VHCI (BTDM_CTRL_HCI_MODE_VHCI)

  Normal option. Mostly, choose this VHCI when bluetooth host run on ESP32, too.

- UART(H4) (BTDM_CTRL_HCI_MODE_UART_H4)

  If use external bluetooth host which run on other hardware and use UART as the HCI interface, choose this option.

HCI UART(H4) Options

Contains:

- *CONFIG_BT_HCI_UART_NO*
- *CONFIG_BT_HCI_UART_BAUDRATE*

CONFIG_BT_HCI_UART_NO

UART Number for HCI

Found in: Component config > Bluetooth > Bluetooth controller > HCI UART(H4) Options

Uart number for HCI. The available uart is UART1 and UART2.

CONFIG_BT_HCI_UART_BAUDRATE

UART Baudrate for HCI

Found in: Component config > Bluetooth > Bluetooth controller > HCI UART(H4) Options

UART Baudrate for HCI. Please use standard baudrate.

MODEM SLEEP Options

Contains:

- *CONFIG_BTDM_MODEM_SLEEP*
- *CONFIG_BTDM_LOW_POWER_CLOCK*

CONFIG_BTDM_MODEM_SLEEP

Bluetooth modem sleep

Found in: Component config > Bluetooth > Bluetooth controller > MODEM SLEEP Options

Enable/disable bluetooth controller low power mode.

CONFIG_BTDM_MODEM_SLEEP_MODE

Bluetooth Modem sleep mode

Found in: Component config > Bluetooth > Bluetooth controller > MODEM SLEEP Options > CONFIG_BTDM_MODEM_SLEEP

To select which strategy to use for modem sleep

Available options:

- ORIG Mode(sleep with low power clock) (BTDM_MODEM_SLEEP_MODE_ORIG)

  ORIG mode is a bluetooth sleep mode that can be used for dual mode controller. In this mode, bluetooth controller sleeps between BR/EDR frames and BLE events. A low power clock is used to maintain bluetooth reference clock.

- EVED Mode(For internal test only) (BTDM_MODEM_SLEEP_MODE_EVED)

  EVED mode is for BLE only and is only for internal test. Do not use it for production. this mode is not compatible with DFS nor light sleep

CONFIG_BTDM_LOW_POWER_CLOCK

Bluetooth low power clock

Found in: Component config > Bluetooth > Bluetooth controller > MODEM SLEEP Options

Select the low power clock source for bluetooth controller. Bluetooth low power clock is the clock source to maintain time in sleep mode.

- "Main crystal" option provides good accuracy and can support Dynamic Frequency Scaling to be used with Bluetooth modem sleep. Light sleep is not supported.

- "External 32kHz crystal" option allows user to use a 32.768kHz crystal as Bluetooth low power clock. This option is allowed as long as External 32kHz crystal is configured as the system RTC clock source. This option provides good accuracy and supports Bluetooth modem sleep to be used alongside Dynamic Frequency Scaling or light sleep.

Available options:

- Main crystal (BTDM_LPCLK_SEL_MAIN_XTAL)

  Main crystal can be used as low power clock for bluetooth modem sleep. If this option is selected, bluetooth modem sleep can work under Dynamic Frequency Scaling(DFS) enabled, but cannot work when light sleep is enabled. Main crystal has a good performance in accuracy as the bluetooth low power clock source.

- External 32kHz crystal (BTDM_LPCLK_SEL_EXT_32K_XTAL)

  External 32kHz crystal has a nominal frequency of 32.768kHz and provides good frequency stability. If used as Bluetooth low power clock, External 32kHz can support Bluetooth modem sleep to be used with both DFS and light sleep.

CONFIG_BTDM_BLE_SLEEP_CLOCK_ACCURACY

BLE Sleep Clock Accuracy

Found in: Component config > Bluetooth > Bluetooth controller

BLE Sleep Clock Accuracy(SCA) for the local device is used to estimate window widening in BLE connection events. With a lower level of clock accuracy(e.g. 500ppm over 250ppm), the slave needs a larger RX window to synchronize with master in each anchor point, thus resulting in an increase of power consumption but a higher level of robustness in keeping connected. According to the requirements of Bluetooth Core specification 4.2, the worst-case accuracy of Classic Bluetooth low power oscialltor(LPO) is +/-250ppm in STANDBY and in low power modes such as sniff. For BLE the worst-case SCA is +/-500ppm.

- "151ppm to 250ppm" option is the default value for Bluetooth Dual mode
- **"251ppm to 500ppm" option can be used in BLE only mode when using external 32kHz crystal a** low power clock. This option is provided in case that BLE sleep clock has a lower level of accuracy, or other error sources contribute to the inaccurate timing during sleep.

Available options:

- 251ppm to 500ppm (BTDM_BLE_DEFAULT_SCA_500PPM)
- 151ppm to 250ppm (BTDM_BLE_DEFAULT_SCA_250PPM)

CONFIG_BTDM_BLE_SCAN_DUPL

BLE Scan Duplicate Options

Found in: Component config > Bluetooth > Bluetooth controller

This select enables parameters setting of BLE scan duplicate.

CONFIG_BTDM_SCAN_DUPL_TYPE

Scan Duplicate Type

Found in: Component config > Bluetooth > Bluetooth controller > CONFIG_BTDM_BLE_SCAN_DUPL

Scan duplicate have three ways. one is "Scan Duplicate By Device Address" , This way is to use advertiser address filtering. The adv packet of the same address is only allowed to be reported once. Another way is "Scan Duplicate By Device Address And Advertising Data" . This way is to use advertising data and device address filtering. All different adv packets with the same address are allowed to be reported. The last way is "Scan Duplicate By Advertising Data" . This way is to use advertising data filtering. All same advertising data only allow to be reported once even though they are from different devices.

Available options:

- Scan Duplicate By Device Address (BTDM_SCAN_DUPL_TYPE_DEVICE)

  This way is to use advertiser address filtering. The adv packet of the same address is only allowed to be reported once

- Scan Duplicate By Advertising Data (BTDM_SCAN_DUPL_TYPE_DATA)

  This way is to use advertising data filtering. All same advertising data only allow to be reported once even though they are from different devices.

- Scan Duplicate By Device Address And Advertising Data (BTDM_SCAN_DUPL_TYPE_DATA_DEVICE)

  This way is to use advertising data and device address filtering. All different adv packets with the same address are allowed to be reported.

CONFIG_BTDM_SCAN_DUPL_CACHE_SIZE

Maximum number of devices in scan duplicate filter

Found in: Component config > Bluetooth > Bluetooth controller > CONFIG_BTDM_BLE_SCAN_DUPL

Maximum number of devices which can be recorded in scan duplicate filter. When the maximum amount of device in the filter is reached, the cache will be refreshed.

CONFIG_BTDM_BLE_MESH_SCAN_DUPL_EN

Special duplicate scan mechanism for BLE Mesh scan

Found in: Component config > Bluetooth > Bluetooth controller > CONFIG_BTDM_BLE_SCAN_DUPL

This enables the BLE scan duplicate for special BLE Mesh scan.

CONFIG_BTDM_MESH_DUPL_SCAN_CACHE_SIZE

Maximum number of Mesh adv packets in scan duplicate filter

Found in: Component config > Bluetooth > Bluetooth controller > CONFIG_BTDM_BLE_SCAN_DUPL > CONFIG_BTDM_BLE_MESH_SCAN_DUPL_EN

Maximum number of adv packets which can be recorded in duplicate scan cache for BLE Mesh. When the maximum amount of device in the filter is reached, the cache will be refreshed.

CONFIG_BTDM_CTRL_FULL_SCAN_SUPPORTED

BLE full scan feature supported

Found in: Component config > Bluetooth > Bluetooth controller

The full scan function is mainly used to provide BLE scan performance. This is required for scenes with high scan performance requirements, such as BLE Mesh scenes.

CONFIG_BTDM_BLE_ADV_REPORT_FLOW_CTRL_SUPP

BLE adv report flow control supported

Found in: Component config > Bluetooth > Bluetooth controller

The function is mainly used to enable flow control for advertising reports. When it is enabled, advertising reports will be discarded by the controller if the number of unprocessed advertising reports exceeds the size of BLE adv report flow control.

CONFIG_BTDM_BLE_ADV_REPORT_FLOW_CTRL_NUM

BLE adv report flow control number

Found in: Component config > Bluetooth > Bluetooth controller > CONFIG_BTDM_BLE_ADV_REPORT_FLOW_CTRL_SUPP

The number of unprocessed advertising report that Bluedroid can save.If you set *BTDM_BLE_ADV_REPORT_FLOW_CTRL_NUM* to a small value, this may cause adv packets lost. If you set *BTDM_BLE_ADV_REPORT_FLOW_CTRL_NUM* to a large value, Bluedroid may cache a lot of adv packets and this may cause system memory run out. For example, if you set it to 50, the maximum memory consumed by host is 35 * 50 bytes. Please set *BTDM_BLE_ADV_REPORT_FLOW_CTRL_NUM* according to your system free memory and handle adv packets as fast as possible, otherwise it will cause adv packets lost.

CONFIG_BTDM_BLE_ADV_REPORT_DISCARD_THRSHOLD

BLE adv lost event threshold value

Found in: Component config > Bluetooth > Bluetooth controller > CONFIG_BTDM_BLE_ADV_REPORT_FLOW_CTRL_SUPP

When adv report flow control is enabled, The ADV lost event will be generated when the number of ADV packets lost in the controller reaches this threshold. It is better to set a larger value. If you set *BTDM_BLE_ADV_REPORT_DISCARD_THRSHOLD* to a small value or printf every adv lost event, it may cause adv packets lost more.

CONFIG_BTDM_COEX_BT_OPTIONS

Coexistence Bluetooth Side Options

Found in: Component config > Bluetooth > Bluetooth controller

Options of Bluetooth Side of WiFi and bluetooth coexistence.

Contains:

- *CONFIG_BTDM_COEX_BLE_ADV_HIGH_PRIORITY*

CONFIG_BTDM_COEX_BLE_ADV_HIGH_PRIORITY

Improve BLE ADV priority for WiFi & BLE coexistence

Found in: Component config > Bluetooth > Bluetooth controller > CONFIG_BTDM_COEX_BT_OPTIONS

Improve BLE ADV coexistence priority to make it better performance. For example, BLE mesh need to enable this option to improve BLE adv performance.

CONFIG_BT_HOST

Bluetooth Host

Found in: Component config > Bluetooth

This helps to choose Bluetooth host stack

Available options:

- Bluedroid - Dual-mode (BT_BLUEDROID_ENABLED)

  This option is recommended for classic Bluetooth or for dual-mode usecases

- NimBLE - BLE only (BT_NIMBLE_ENABLED)

  This option is recommended for BLE only usecases to save on memory

- Controller Only (BT_CONTROLLER_ONLY)

  This option is recommended when you want to communicate directly with the controller (without any host) or when you are using any other host stack not supported by Espressif (not mentioned here).

Bluedroid Options

Contains:

- *CONFIG_BT_BTC_TASK_STACK_SIZE*
- *CONFIG_BT_BLUEDROID_PINNED_TO_CORE_CHOICE*
- *CONFIG_BT_BTU_TASK_STACK_SIZE*
- *CONFIG_BT_BLUEDROID_MEM_DEBUG*
- *CONFIG_BT_CLASSIC_ENABLED*
- *CONFIG_BT_HFP_WBS_ENABLE*
- *CONFIG_BT_SSP_ENABLED*
- *CONFIG_BT_BLE_ENABLED*
- *CONFIG_BT_STACK_NO_LOG*
- *BT DEBUG LOG LEVEL*
- *CONFIG_BT_ACL_CONNECTIONS*
- *CONFIG_BT_ALLOCATION_FROM_SPIRAM_FIRST*
- *CONFIG_BT_BLE_DYNAMIC_ENV_MEMORY*
- *CONFIG_BT_BLE_HOST_QUEUE_CONG_CHECK*

- *CONFIG_BT_BLE_ACT_SCAN_REP_ADV_SCAN*
- *CONFIG_BT_BLE_ESTAB_LINK_CONN_TOUT*

CONFIG_BT_BTC_TASK_STACK_SIZE

Bluetooth event (callback to application) task stack size

Found in: Component config > Bluetooth > Bluedroid Options

This select btc task stack size

CONFIG_BT_BLUEDROID_PINNED_TO_CORE_CHOICE

The cpu core which Bluedroid run

Found in: Component config > Bluetooth > Bluedroid Options

Which the cpu core to run Bluedroid. Can choose core0 and core1. Can not specify no-affinity.

Available options:

- Core 0 (PRO CPU) (BT_BLUEDROID_PINNED_TO_CORE_0)
- Core 1 (APP CPU) (BT_BLUEDROID_PINNED_TO_CORE_1)

CONFIG_BT_BTU_TASK_STACK_SIZE

Bluetooth Bluedroid Host Stack task stack size

Found in: Component config > Bluetooth > Bluedroid Options

This select btu task stack size

CONFIG_BT_BLUEDROID_MEM_DEBUG

Bluedroid memory debug

Found in: Component config > Bluetooth > Bluedroid Options

Bluedroid memory debug

CONFIG_BT_CLASSIC_ENABLED

Classic Bluetooth

Found in: Component config > Bluetooth > Bluedroid Options

For now this option needs "SMP_ENABLE" to be set to yes

CONFIG_BT_A2DP_ENABLE

A2DP

Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_CLASSIC_ENABLED

Advanced Audio Distrubution Profile

CONFIG_BT_SPP_ENABLED

SPP

Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_CLASSIC_ENABLED

This enables the Serial Port Profile

CONFIG_BT_HFP_ENABLE

Hands Free/Handset Profile

Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_CLASSIC_ENABLED

CONFIG_BT_HFP_ROLE

Hands-free Profile Role configuration

Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_CLASSIC_ENABLED > CONFIG_BT_HFP_ENABLE

Available options:

- Hands Free Unit (BT_HFP_CLIENT_ENABLE)
- Audio Gateway (BT_HFP_AG_ENABLE)

CONFIG_BT_HFP_AUDIO_DATA_PATH

audio(SCO) data path

Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_CLASSIC_ENABLED > CONFIG_BT_HFP_ENABLE

SCO data path, i.e. HCI or PCM. This option is set using API "esp_bredr_sco_datapath_set" in Bluetooth host. Default SCO data path can also be set in Bluetooth Controller.

Available options:

- PCM (BT_HFP_AUDIO_DATA_PATH_PCM)
- HCI (BT_HFP_AUDIO_DATA_PATH_HCI)

CONFIG_BT_HFP_WBS_ENABLE

Wide Band Speech

Found in: Component config > Bluetooth > Bluedroid Options

This enables Wide Band Speech. Should disable it when SCO data path is PCM. Otherwise there will be no data transmited via GPIOs.

CONFIG_BT_SSP_ENABLED

Secure Simple Pairing

Found in: Component config > Bluetooth > Bluedroid Options

This enables the Secure Simple Pairing. If disable this option, Bluedroid will only support Legacy Pairing

CONFIG_BT_BLE_ENABLED

Bluetooth Low Energy

Found in: Component config > Bluetooth > Bluedroid Options

This enables Bluetooth Low Energy

CONFIG_BT_GATTS_ENABLE

Include GATT server module(GATTS)

Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_BLE_ENABLED

This option can be disabled when the app work only on gatt client mode

CONFIG_BT_GATTS_PPCP_CHAR_GAP

Enable Peripheral Preferred Connection Parameters characteristic in GAP service

Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_BLE_ENABLED > CONFIG_BT_GATTS_ENABLE

This enables "Peripheral Preferred Connection Parameters" characteristic (UUID: 0x2A04) in GAP service that has connection parameters like min/max connection interval, slave latency and supervision timeout multiplier

CONFIG_BT_GATTS_SEND_SERVICE_CHANGE_MODE

GATTS Service Change Mode

Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_BLE_ENABLED > CONFIG_BT_GATTS_ENABLE

Service change indication mode for GATT Server.

Available options:

- GATTS manually send service change indication (BT_GATTS_SEND_SERVICE_CHANGE_MANUAL)

  Manually send service change indication through API esp_ble_gatts_send_service_change_indication()

- GATTS automatically send service change indication (BT_GATTS_SEND_SERVICE_CHANGE_AUTO)

  Let Bluedroid handle the service change indication internally

CONFIG_BT_GATTC_ENABLE

Include GATT client module(GATTC)

Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_BLE_ENABLED

This option can be close when the app work only on gatt server mode

CONFIG_BT_GATTC_CACHE_NVS_FLASH

Save gattc cache data to nvs flash

Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_BLE_ENABLED > CONFIG_BT_GATTC_ENABLE

This select can save gattc cache data to nvs flash

CONFIG_BT_BLE_SMP_ENABLE

Include BLE security module(SMP)

Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_BLE_ENABLED

This option can be close when the app not used the ble security connect.

CONFIG_BT_SMP_SLAVE_CON_PARAMS_UPD_ENABLE

Slave enable connection parameters update during pairing

Found in: Component config > Bluetooth > Bluedroid Options > CONFIG_BT_BLE_ENABLED > CONFIG_BT_BLE_SMP_ENABLE

In order to reduce the pairing time, slave actively initiates connection parameters update during pairing.

CONFIG_BT_STACK_NO_LOG

Disable BT debug logs (minimize bin size)

Found in: Component config > Bluetooth > Bluedroid Options

This select can save the rodata code size

BT DEBUG LOG LEVEL

Contains:

- *CONFIG_BT_LOG_HCI_TRACE_LEVEL*
- *CONFIG_BT_LOG_BTM_TRACE_LEVEL*
- *CONFIG_BT_LOG_L2CAP_TRACE_LEVEL*
- *CONFIG_BT_LOG_RFCOMM_TRACE_LEVEL*
- *CONFIG_BT_LOG_SDP_TRACE_LEVEL*
- *CONFIG_BT_LOG_GAP_TRACE_LEVEL*
- *CONFIG_BT_LOG_BNEP_TRACE_LEVEL*
- *CONFIG_BT_LOG_PAN_TRACE_LEVEL*
- *CONFIG_BT_LOG_A2D_TRACE_LEVEL*
- *CONFIG_BT_LOG_AVDT_TRACE_LEVEL*
- *CONFIG_BT_LOG_AVCT_TRACE_LEVEL*
- *CONFIG_BT_LOG_AVRC_TRACE_LEVEL*
- *CONFIG_BT_LOG_MCA_TRACE_LEVEL*

- *CONFIG_BT_LOG_HID_TRACE_LEVEL*
- *CONFIG_BT_LOG_APPL_TRACE_LEVEL*
- *CONFIG_BT_LOG_GATT_TRACE_LEVEL*
- *CONFIG_BT_LOG_SMP_TRACE_LEVEL*
- *CONFIG_BT_LOG_BTIF_TRACE_LEVEL*
- *CONFIG_BT_LOG_BTC_TRACE_LEVEL*
- *CONFIG_BT_LOG_OSI_TRACE_LEVEL*
- *CONFIG_BT_LOG_BLUFI_TRACE_LEVEL*

CONFIG_BT_LOG_HCI_TRACE_LEVEL

HCI layer

Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL

Define BT trace level for HCI layer

Available options:

- NONE (BT_LOG_HCI_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_HCI_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_HCI_TRACE_LEVEL_WARNING)
- API (BT_LOG_HCI_TRACE_LEVEL_API)
- EVENT (BT_LOG_HCI_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_HCI_TRACE_LEVEL_DEBUG)
- VERBOSE (BT_LOG_HCI_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_BTM_TRACE_LEVEL

BTM layer

Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL

Define BT trace level for BTM layer

Available options:

- NONE (BT_LOG_BTM_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_BTM_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_BTM_TRACE_LEVEL_WARNING)

- API (BT_LOG_BTM_TRACE_LEVEL_API)
- EVENT (BT_LOG_BTM_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_BTM_TRACE_LEVEL_DEBUG)
- VERBOSE (BT_LOG_BTM_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_L2CAP_TRACE_LEVEL

L2CAP layer

Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL

Define BT trace level for L2CAP layer

Available options:

- NONE (BT_LOG_L2CAP_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_L2CAP_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_L2CAP_TRACE_LEVEL_WARNING)
- API (BT_LOG_L2CAP_TRACE_LEVEL_API)
- EVENT (BT_LOG_L2CAP_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_L2CAP_TRACE_LEVEL_DEBUG)
- VERBOSE (BT_LOG_L2CAP_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_RFCOMM_TRACE_LEVEL

RFCOMM layer

Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL

Define BT trace level for RFCOMM layer

Available options:

- NONE (BT_LOG_RFCOMM_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_RFCOMM_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_RFCOMM_TRACE_LEVEL_WARNING)
- API (BT_LOG_RFCOMM_TRACE_LEVEL_API)
- EVENT (BT_LOG_RFCOMM_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_RFCOMM_TRACE_LEVEL_DEBUG)
- VERBOSE (BT_LOG_RFCOMM_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_SDP_TRACE_LEVEL

SDP layer

Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL

Define BT trace level for SDP layer

Available options:

- NONE (BT_LOG_SDP_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_SDP_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_SDP_TRACE_LEVEL_WARNING)
- API (BT_LOG_SDP_TRACE_LEVEL_API)
- EVENT (BT_LOG_SDP_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_SDP_TRACE_LEVEL_DEBUG)
- VERBOSE (BT_LOG_SDP_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_GAP_TRACE_LEVEL

GAP layer

Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL

Define BT trace level for GAP layer

Available options:

- NONE (BT_LOG_GAP_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_GAP_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_GAP_TRACE_LEVEL_WARNING)
- API (BT_LOG_GAP_TRACE_LEVEL_API)
- EVENT (BT_LOG_GAP_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_GAP_TRACE_LEVEL_DEBUG)
- VERBOSE (BT_LOG_GAP_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_BNEP_TRACE_LEVEL

BNEP layer

Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL

Define BT trace level for BNEP layer

Available options:

- NONE (BT_LOG_BNEP_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_BNEP_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_BNEP_TRACE_LEVEL_WARNING)
- API (BT_LOG_BNEP_TRACE_LEVEL_API)
- EVENT (BT_LOG_BNEP_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_BNEP_TRACE_LEVEL_DEBUG)
- VERBOSE (BT_LOG_BNEP_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_PAN_TRACE_LEVEL

PAN layer

Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL

Define BT trace level for PAN layer

Available options:

- NONE (BT_LOG_PAN_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_PAN_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_PAN_TRACE_LEVEL_WARNING)
- API (BT_LOG_PAN_TRACE_LEVEL_API)
- EVENT (BT_LOG_PAN_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_PAN_TRACE_LEVEL_DEBUG)
- VERBOSE (BT_LOG_PAN_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_A2D_TRACE_LEVEL

A2D layer

Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL

Define BT trace level for A2D layer

Available options:

- NONE (BT_LOG_A2D_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_A2D_TRACE_LEVEL_ERROR)

- WARNING (BT_LOG_A2D_TRACE_LEVEL_WARNING)
- API (BT_LOG_A2D_TRACE_LEVEL_API)
- EVENT (BT_LOG_A2D_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_A2D_TRACE_LEVEL_DEBUG)
- VERBOSE (BT_LOG_A2D_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_AVDT_TRACE_LEVEL

AVDT layer

Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL

Define BT trace level for AVDT layer

Available options:

- NONE (BT_LOG_AVDT_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_AVDT_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_AVDT_TRACE_LEVEL_WARNING)
- API (BT_LOG_AVDT_TRACE_LEVEL_API)
- EVENT (BT_LOG_AVDT_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_AVDT_TRACE_LEVEL_DEBUG)
- VERBOSE (BT_LOG_AVDT_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_AVCT_TRACE_LEVEL

AVCT layer

Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL

Define BT trace level for AVCT layer

Available options:

- NONE (BT_LOG_AVCT_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_AVCT_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_AVCT_TRACE_LEVEL_WARNING)
- API (BT_LOG_AVCT_TRACE_LEVEL_API)
- EVENT (BT_LOG_AVCT_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_AVCT_TRACE_LEVEL_DEBUG)

- VERBOSE (BT_LOG_AVCT_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_AVRC_TRACE_LEVEL

AVRC layer

Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL

Define BT trace level for AVRC layer

Available options:

- NONE (BT_LOG_AVRC_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_AVRC_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_AVRC_TRACE_LEVEL_WARNING)
- API (BT_LOG_AVRC_TRACE_LEVEL_API)
- EVENT (BT_LOG_AVRC_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_AVRC_TRACE_LEVEL_DEBUG)
- VERBOSE (BT_LOG_AVRC_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_MCA_TRACE_LEVEL

MCA layer

Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL

Define BT trace level for MCA layer

Available options:

- NONE (BT_LOG_MCA_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_MCA_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_MCA_TRACE_LEVEL_WARNING)
- API (BT_LOG_MCA_TRACE_LEVEL_API)
- EVENT (BT_LOG_MCA_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_MCA_TRACE_LEVEL_DEBUG)
- VERBOSE (BT_LOG_MCA_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_HID_TRACE_LEVEL

HID layer

Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL

Define BT trace level for HID layer

Available options:

- NONE (BT_LOG_HID_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_HID_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_HID_TRACE_LEVEL_WARNING)
- API (BT_LOG_HID_TRACE_LEVEL_API)
- EVENT (BT_LOG_HID_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_HID_TRACE_LEVEL_DEBUG)
- VERBOSE (BT_LOG_HID_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_APPL_TRACE_LEVEL

APPL layer

Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL

Define BT trace level for APPL layer

Available options:

- NONE (BT_LOG_APPL_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_APPL_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_APPL_TRACE_LEVEL_WARNING)
- API (BT_LOG_APPL_TRACE_LEVEL_API)
- EVENT (BT_LOG_APPL_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_APPL_TRACE_LEVEL_DEBUG)
- VERBOSE (BT_LOG_APPL_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_GATT_TRACE_LEVEL

GATT layer

Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL

Define BT trace level for GATT layer

Available options:

- NONE (BT_LOG_GATT_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_GATT_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_GATT_TRACE_LEVEL_WARNING)
- API (BT_LOG_GATT_TRACE_LEVEL_API)
- EVENT (BT_LOG_GATT_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_GATT_TRACE_LEVEL_DEBUG)
- VERBOSE (BT_LOG_GATT_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_SMP_TRACE_LEVEL

SMP layer

Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL

Define BT trace level for SMP layer

Available options:

- NONE (BT_LOG_SMP_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_SMP_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_SMP_TRACE_LEVEL_WARNING)
- API (BT_LOG_SMP_TRACE_LEVEL_API)
- EVENT (BT_LOG_SMP_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_SMP_TRACE_LEVEL_DEBUG)
- VERBOSE (BT_LOG_SMP_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_BTIF_TRACE_LEVEL

BTIF layer

Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL

Define BT trace level for BTIF layer

Available options:

- NONE (BT_LOG_BTIF_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_BTIF_TRACE_LEVEL_ERROR)

- WARNING (BT_LOG_BTIF_TRACE_LEVEL_WARNING)
- API (BT_LOG_BTIF_TRACE_LEVEL_API)
- EVENT (BT_LOG_BTIF_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_BTIF_TRACE_LEVEL_DEBUG)
- VERBOSE (BT_LOG_BTIF_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_BTC_TRACE_LEVEL

BTC layer

Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL

Define BT trace level for BTC layer

Available options:

- NONE (BT_LOG_BTC_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_BTC_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_BTC_TRACE_LEVEL_WARNING)
- API (BT_LOG_BTC_TRACE_LEVEL_API)
- EVENT (BT_LOG_BTC_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_BTC_TRACE_LEVEL_DEBUG)
- VERBOSE (BT_LOG_BTC_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_OSI_TRACE_LEVEL

OSI layer

Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL

Define BT trace level for OSI layer

Available options:

- NONE (BT_LOG_OSI_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_OSI_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_OSI_TRACE_LEVEL_WARNING)
- API (BT_LOG_OSI_TRACE_LEVEL_API)
- EVENT (BT_LOG_OSI_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_OSI_TRACE_LEVEL_DEBUG)

- VERBOSE (BT_LOG_OSI_TRACE_LEVEL_VERBOSE)

CONFIG_BT_LOG_BLUFI_TRACE_LEVEL

BLUFI layer

Found in: Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL

Define BT trace level for BLUFI layer

Available options:

- NONE (BT_LOG_BLUFI_TRACE_LEVEL_NONE)
- ERROR (BT_LOG_BLUFI_TRACE_LEVEL_ERROR)
- WARNING (BT_LOG_BLUFI_TRACE_LEVEL_WARNING)
- API (BT_LOG_BLUFI_TRACE_LEVEL_API)
- EVENT (BT_LOG_BLUFI_TRACE_LEVEL_EVENT)
- DEBUG (BT_LOG_BLUFI_TRACE_LEVEL_DEBUG)
- VERBOSE (BT_LOG_BLUFI_TRACE_LEVEL_VERBOSE)

CONFIG_BT_ACL_CONNECTIONS

BT/BLE MAX ACL CONNECTIONS(1~7)

Found in: Component config > Bluetooth > Bluedroid Options

Maximum BT/BLE connection count

CONFIG_BT_ALLOCATION_FROM_SPIRAM_FIRST

BT/BLE will first malloc the memory from the PSRAM

Found in: Component config > Bluetooth > Bluedroid Options

This select can save the internal RAM if there have the PSRAM

CONFIG_BT_BLE_DYNAMIC_ENV_MEMORY

Use dynamic memory allocation in BT/BLE stack

Found in: Component config > Bluetooth > Bluedroid Options

This select can make the allocation of memory will become more flexible

CONFIG_BT_BLE_HOST_QUEUE_CONG_CHECK

BLE queue congestion check

Found in: Component config > Bluetooth > Bluedroid Options

When scanning and scan duplicate is not enabled, if there are a lot of adv packets around or application layer handling adv packets is slow, it will cause the controller memory to run out. if enabled, adv packets will be lost when host queue is congested.

CONFIG_BT_BLE_ACT_SCAN_REP_ADV_SCAN

Report adv data and scan response individually when BLE active scan

Found in: Component config > Bluetooth > Bluedroid Options

Originally, when doing BLE active scan, Bluedroid will not report adv to application layer until receive scan response. This option is used to disable the behavior. When enable this option, Bluedroid will report adv data or scan response to application layer immediately.

Memory reserved at start of DRAM for Bluetooth stack

CONFIG_BT_BLE_ESTAB_LINK_CONN_TOUT

Timeout of BLE connection establishment

Found in: Component config > Bluetooth > Bluedroid Options

Bluetooth Connection establishment maximum time, if connection time exceeds this value, the connection establishment fails, ESP_GATTC_OPEN_EVT or ESP_GATTS_OPEN_EVT is triggered.

NimBLE Options

Contains:

- *CONFIG_BT_NIMBLE_MEM_ALLOC_MODE*
- *CONFIG_BT_NIMBLE_MAX_CONNECTIONS*
- *CONFIG_BT_NIMBLE_MAX_BONDS*
- *CONFIG_BT_NIMBLE_MAX_CCCDS*
- *CONFIG_BT_NIMBLE_L2CAP_COC_MAX_NUM*
- *CONFIG_BT_NIMBLE_PINNED_TO_CORE_CHOICE*
- *CONFIG_BT_NIMBLE_TASK_STACK_SIZE*

- *CONFIG_BT_NIMBLE_ROLE_CENTRAL*
- *CONFIG_BT_NIMBLE_ROLE_PERIPHERAL*
- *CONFIG_BT_NIMBLE_ROLE_BROADCASTER*
- *CONFIG_BT_NIMBLE_ROLE_OBSERVER*
- *CONFIG_BT_NIMBLE_NVS_PERSIST*
- *CONFIG_BT_NIMBLE_SM_LEGACY*
- *CONFIG_BT_NIMBLE_SM_SC*
- *CONFIG_BT_NIMBLE_DEBUG*
- *CONFIG_BT_NIMBLE_SM_SC_DEBUG_KEYS*
- *CONFIG_BT_NIMBLE_SVC_GAP_DEVICE_NAME*
- *CONFIG_BT_NIMBLE_GAP_DEVICE_NAME_MAX_LEN*
- *CONFIG_BT_NIMBLE_ATT_PREFERRED_MTU*
- *CONFIG_BT_NIMBLE_SVC_GAP_APPEARANCE*
- *CONFIG_BT_NIMBLE_ACL_BUF_COUNT*
- *CONFIG_BT_NIMBLE_ACL_BUF_SIZE*
- *CONFIG_BT_NIMBLE_HCI_EVT_BUF_SIZE*
- *CONFIG_BT_NIMBLE_HCI_EVT_HI_BUF_COUNT*
- *CONFIG_BT_NIMBLE_HCI_EVT_LO_BUF_COUNT*
- *CONFIG_BT_NIMBLE_MSYS1_BLOCK_COUNT*
- *CONFIG_BT_NIMBLE_HS_FLOW_CTRL*
- *CONFIG_BT_NIMBLE_RPA_TIMEOUT*
- *CONFIG_BT_NIMBLE_MESH*
- *CONFIG_BT_NIMBLE_CRYPTO_STACK_MBEDTLS*

CONFIG_BT_NIMBLE_MEM_ALLOC_MODE

Memory allocation strategy

Found in: Component config > Bluetooth > NimBLE Options

Allocation strategy for NimBLE host stack, essentially provides ability to allocate all required dynamic allocations from,

- Internal DRAM memory only

- External SPIRAM memory only
- Either internal or external memory based on default malloc() behavior in ESP-IDF

Recommended mode here is always internal, since that is most preferred from security perspective. But if application requirement does not allow sufficient free internal memory then alternate mode can be selected.

Available options:

- Internal memory (BT_NIMBLE_MEM_ALLOC_MODE_INTERNAL)
- External SPIRAM (BT_NIMBLE_MEM_ALLOC_MODE_EXTERNAL)
- Default alloc mode (BT_NIMBLE_MEM_ALLOC_MODE_DEFAULT)

CONFIG_BT_NIMBLE_MAX_CONNECTIONS

Maximum number of concurrent connections

Found in: Component config > Bluetooth > NimBLE Options

Defines maximum number of concurrent BLE connections

CONFIG_BT_NIMBLE_MAX_BONDS

Maximum number of bonds to save across reboots

Found in: Component config > Bluetooth > NimBLE Options

Defines maximum number of bonds to save for peer security and our security

CONFIG_BT_NIMBLE_MAX_CCCDS

Maximum number of CCC descriptors to save across reboots

Found in: Component config > Bluetooth > NimBLE Options

Defines maximum number of CCC descriptors to save

CONFIG_BT_NIMBLE_L2CAP_COC_MAX_NUM

Maximum number of connection oriented channels

Found in: Component config > Bluetooth > NimBLE Options

Defines maximum number of BLE Connection Oriented Channels. When set to (0), BLE COC is not compiled in

CONFIG_BT_NIMBLE_PINNED_TO_CORE_CHOICE

The CPU core on which NimBLE host will run

Found in: Component config > Bluetooth > NimBLE Options

The CPU core on which NimBLE host will run. You can choose Core 0 or Core 1. Cannot specify no-affinity

Available options:

- Core 0 (PRO CPU) (BT_NIMBLE_PINNED_TO_CORE_0)
- Core 1 (APP CPU) (BT_NIMBLE_PINNED_TO_CORE_1)

CONFIG_BT_NIMBLE_TASK_STACK_SIZE

NimBLE Host task stack size

Found in: Component config > Bluetooth > NimBLE Options

This configures stack size of NimBLE host task

CONFIG_BT_NIMBLE_ROLE_CENTRAL

Enable BLE Central role

Found in: Component config > Bluetooth > NimBLE Options

CONFIG_BT_NIMBLE_ROLE_PERIPHERAL

Enable BLE Peripheral role

Found in: Component config > Bluetooth > NimBLE Options

CONFIG_BT_NIMBLE_ROLE_BROADCASTER

Enable BLE Broadcaster role

Found in: Component config > Bluetooth > NimBLE Options

CONFIG_BT_NIMBLE_ROLE_OBSERVER

Enable BLE Observer role

Found in: Component config > Bluetooth > NimBLE Options

CONFIG_BT_NIMBLE_NVS_PERSIST

Persist the BLE Bonding keys in NVS

Found in: Component config > Bluetooth > NimBLE Options

Enable this flag to make bonding persistent across device reboots

CONFIG_BT_NIMBLE_SM_LEGACY

Security manager legacy pairing

Found in: Component config > Bluetooth > NimBLE Options

Enable security manager legacy pairing

CONFIG_BT_NIMBLE_SM_SC

Security manager secure connections (4.2)

Found in: Component config > Bluetooth > NimBLE Options

Enable security manager secure connections

CONFIG_BT_NIMBLE_DEBUG

Enable extra runtime asserts and host debugging

Found in: Component config > Bluetooth > NimBLE Options

This enables extra runtime asserts and host debugging

CONFIG_BT_NIMBLE_SM_SC_DEBUG_KEYS

Use predefined public-private key pair

Found in: Component config > Bluetooth > NimBLE Options

If this option is enabled, SM uses predefined DH key pair as described in Core Specification, Vol. 3, Part H, 2.3.5.6.1. This allows to decrypt air traffic easily and thus should only be used for debugging.

CONFIG_BT_NIMBLE_SVC_GAP_DEVICE_NAME

BLE GAP default device name

Found in: Component config > Bluetooth > NimBLE Options

The Device Name characteristic shall contain the name of the device as an UTF-8 string. This name can be changed by using API ble_svc_gap_device_name_set()

CONFIG_BT_NIMBLE_GAP_DEVICE_NAME_MAX_LEN

Maximum length of BLE device name in octets

Found in: *Component config* > *Bluetooth* > *NimBLE Options*

Device Name characteristic value shall be 0 to 248 octets in length

CONFIG_BT_NIMBLE_ATT_PREFERRED_MTU

Preferred MTU size in octets

Found in: *Component config* > *Bluetooth* > *NimBLE Options*

This is the default value of ATT MTU indicated by the device during an ATT MTU exchange. This value can be changed using API ble_att_set_preferred_mtu()

CONFIG_BT_NIMBLE_SVC_GAP_APPEARANCE

External appearance of the device

Found in: *Component config* > *Bluetooth* > *NimBLE Options*

Standard BLE GAP Appearance value in HEX format e.g. 0x02C0

CONFIG_BT_NIMBLE_ACL_BUF_COUNT

ACL Buffer count

Found in: *Component config* > *Bluetooth* > *NimBLE Options*

The number of ACL data buffers.

CONFIG_BT_NIMBLE_ACL_BUF_SIZE

ACL Buffer size

Found in: *Component config* > *Bluetooth* > *NimBLE Options*

This is the maximum size of the data portion of HCI ACL data packets. It does not include the HCI data header (of 4 bytes)

CONFIG_BT_NIMBLE_HCI_EVT_BUF_SIZE

HCI Event Buffer size

Found in: Component config > Bluetooth > NimBLE Options

This is the size of each HCI event buffer in bytes

CONFIG_BT_NIMBLE_HCI_EVT_HI_BUF_COUNT

High Priority HCI Event Buffer count

Found in: Component config > Bluetooth > NimBLE Options

This is the high priority HCI events' buffer size. High-priority event buffers are for everything except advertising reports. If there are no free high-priority event buffers then host will try to allocate a low-priority buffer instead

CONFIG_BT_NIMBLE_HCI_EVT_LO_BUF_COUNT

Low Priority HCI Event Buffer count

Found in: Component config > Bluetooth > NimBLE Options

This is the low priority HCI events' buffer size. Low-priority event buffers are only used for advertising reports. If there are no free low-priority event buffers, then an incoming advertising report will get dropped

CONFIG_BT_NIMBLE_MSYS1_BLOCK_COUNT

MSYS_1 Block Count

Found in: Component config > Bluetooth > NimBLE Options

MSYS is a system level mbuf registry. For prepare write & prepare responses MBUFs are allocated out of msys_1 pool. For NIMBLE_MESH enabled cases, this block count is increased by 8 than user defined count.

CONFIG_BT_NIMBLE_HS_FLOW_CTRL

Enable Host Flow control

Found in: Component config > Bluetooth > NimBLE Options

Enable Host Flow control

CONFIG_BT_NIMBLE_HS_FLOW_CTRL_ITVL

Host Flow control interval

Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_HS_FLOW_CTRL

Host flow control interval in msecs

CONFIG_BT_NIMBLE_HS_FLOW_CTRL_THRESH

Host Flow control threshold

Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_HS_FLOW_CTRL

Host flow control threshold, if the number of free buffers are at or below this threshold, send an immediate number-of-completed-packets event

CONFIG_BT_NIMBLE_HS_FLOW_CTRL_TX_ON_DISCONNECT

Host Flow control on disconnect

Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_HS_FLOW_CTRL

Enable this option to send number-of-completed-packets event to controller after disconnection

CONFIG_BT_NIMBLE_RPA_TIMEOUT

RPA timeout in seconds

Found in: Component config > Bluetooth > NimBLE Options

Time interval between RPA address change. This is applicable in case of Host based RPA

CONFIG_BT_NIMBLE_MESH

Enable BLE mesh functionality

Found in: Component config > Bluetooth > NimBLE Options

Enable BLE Mesh functionality

Contains:

- *CONFIG_BT_NIMBLE_MESH_PROXY*

- *CONFIG_BT_NIMBLE_MESH_PROV*
- *CONFIG_BT_NIMBLE_MESH_GATT_PROXY*
- *CONFIG_BT_NIMBLE_MESH_RELAY*
- *CONFIG_BT_NIMBLE_MESH_LOW_POWER*
- *CONFIG_BT_NIMBLE_MESH_FRIEND*
- *CONFIG_BT_NIMBLE_MESH_DEVICE_NAME*

CONFIG_BT_NIMBLE_MESH_PROXY

Enable mesh proxy functionality

Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_MESH

Enable proxy. This is automatically set whenever NIMBLE_MESH_PB_GATT or NIMBLE_MESH_GATT_PROXY is set

CONFIG_BT_NIMBLE_MESH_PROV

Enable BLE mesh provisioning

Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_MESH

Enable mesh provisioning

CONFIG_BT_NIMBLE_MESH_PB_ADV

Enable mesh provisioning over advertising bearer

Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_MESH > CONFIG_BT_NIMBLE_MESH_PROV

Enable this option to allow the device to be provisioned over the advertising bearer

CONFIG_BT_NIMBLE_MESH_PB_GATT

Enable mesh provisioning over GATT bearer

Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_MESH > CONFIG_BT_NIMBLE_MESH_PROV

Enable this option to allow the device to be provisioned over the GATT bearer

CONFIG_BT_NIMBLE_MESH_GATT_PROXY

Enable GATT Proxy functionality

Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_MESH

This option enables support for the Mesh GATT Proxy Service, i.e. the ability to act as a proxy between a Mesh GATT Client and a Mesh network

CONFIG_BT_NIMBLE_MESH_RELAY

Enable mesh relay functionality

Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_MESH

Support for acting as a Mesh Relay Node

CONFIG_BT_NIMBLE_MESH_LOW_POWER

Enable mesh low power mode

Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_MESH

Enable this option to be able to act as a Low Power Node

CONFIG_BT_NIMBLE_MESH_FRIEND

Enable mesh friend functionality

Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_MESH

Enable this option to be able to act as a Friend Node

CONFIG_BT_NIMBLE_MESH_DEVICE_NAME

Set mesh device name

Found in: Component config > Bluetooth > NimBLE Options > CONFIG_BT_NIMBLE_MESH

This value defines Bluetooth Mesh device/node name

CONFIG_BT_NIMBLE_CRYPTO_STACK_MBEDTLS

Override TinyCrypt with mbedTLS for crypto computations

Found in: Component config > Bluetooth > NimBLE Options

Enable this option to choose mbedTLS instead of TinyCrypt for crypto computations.

CONFIG_BLE_MESH

ESP BLE Mesh Support

Found in: Component config

This option enables ESP BLE Mesh support. The specific features that are available may depend on other features that have been enabled in the stack, such as Bluetooth Support, Bluedroid Support & GATT support.

Contains:

- *CONFIG_BLE_MESH_HCI_5_0*
- *CONFIG_BLE_MESH_USE_DUPLICATE_SCAN*
- *CONFIG_BLE_MESH_ALLOC_FROM_PSRAM_FIRST*
- *CONFIG_BLE_MESH_FAST_PROV*
- *CONFIG_BLE_MESH_NODE*
- *CONFIG_BLE_MESH_PROVISIONER*
- *CONFIG_BLE_MESH_PROV*
- *CONFIG_BLE_MESH_PB_ADV*
- *CONFIG_BLE_MESH_PB_GATT*
- *CONFIG_BLE_MESH_PROXY*
- *CONFIG_BLE_MESH_GATT_PROXY_SERVER*
- *CONFIG_BLE_MESH_GATT_PROXY_CLIENT*
- *CONFIG_BLE_MESH_NET_BUF_POOL_USAGE*
- *CONFIG_BLE_MESH_SETTINGS*
- *CONFIG_BLE_MESH_SUBNET_COUNT*
- *CONFIG_BLE_MESH_APP_KEY_COUNT*
- *CONFIG_BLE_MESH_MODEL_KEY_COUNT*
- *CONFIG_BLE_MESH_MODEL_GROUP_COUNT*

- *CONFIG_BLE_MESH_LABEL_COUNT*
- *CONFIG_BLE_MESH_CRPL*
- *CONFIG_BLE_MESH_MSG_CACHE_SIZE*
- *CONFIG_BLE_MESH_ADV_BUF_COUNT*
- *CONFIG_BLE_MESH_SUPPORT_BLE_ADV*
- *CONFIG_BLE_MESH_IVU_DIVIDER*
- *CONFIG_BLE_MESH_TX_SEG_MSG_COUNT*
- *CONFIG_BLE_MESH_RX_SEG_MSG_COUNT*
- *CONFIG_BLE_MESH_RX_SDU_MAX*
- *CONFIG_BLE_MESH_TX_SEG_MAX*
- *CONFIG_BLE_MESH_RELAY*
- *CONFIG_BLE_MESH_LOW_POWER*
- *CONFIG_BLE_MESH_FRIEND*
- *CONFIG_BLE_MESH_NO_LOG*
- *BLE Mesh STACK DEBUG LOG LEVEL*
- *BLE Mesh NET BUF DEBUG LOG LEVEL*
- *CONFIG_BLE_MESH_CLIENT_MSG_TIMEOUT*
- *Support for BLE Mesh Client Models*
- *CONFIG_BLE_MESH_IV_UPDATE_TEST*
- *BLE Mesh specific test option*

CONFIG_BLE_MESH_HCI_5_0

Support sending 20ms non-connectable adv packets

Found in: Component config > CONFIG_BLE_MESH

It is a temporary solution and needs further modifications.

CONFIG_BLE_MESH_USE_DUPLICATE_SCAN

Support Duplicate Scan in BLE Mesh

Found in: Component config > CONFIG_BLE_MESH

Enable this option to allow using specific duplicate scan filter in BLE Mesh, and Scan Duplicate Type must be set by choosing the option in the Bluetooth Controller section in menuconfig, which is "Scan Duplicate By Device Address and Advertising Data".

CONFIG_BLE_MESH_ALLOC_FROM_PSRAM_FIRST

BLE Mesh will first allocate memory from PSRAM

Found in: Component config > CONFIG_BLE_MESH

When this option is enabled, BLE Mesh stack will try to allocate memory from PSRAM firstly. This will save the internal RAM if PSRAM exists.

CONFIG_BLE_MESH_FAST_PROV

Enable BLE Mesh Fast Provisioning

Found in: Component config > CONFIG_BLE_MESH

Enable this option to allow BLE Mesh fast provisioning solution to be used. When there are multiple unprovisioned devices around, fast provisioning can greatly reduce the time consumption of the whole provisioning process. When this option is enabled, and after an unprovisioned device is provisioned into a node successfully, it can be changed to a temporary Provisioner.

CONFIG_BLE_MESH_NODE

Support for BLE Mesh Node

Found in: Component config > CONFIG_BLE_MESH

Enable the device to be provisioned into a node. This option should be enabled when an unprovisioned device is going to be provisioned into a node and communicate with other nodes in the BLE Mesh network.

CONFIG_BLE_MESH_PROVISIONER

Support for BLE Mesh Provisioner

Found in: Component config > CONFIG_BLE_MESH

Enable the device to be a Provisioner. The option should be enabled when a device is going to act as a Provisioner and provision unprovisioned devices into the BLE Mesh network.

CONFIG_BLE_MESH_WAIT_FOR_PROV_MAX_DEV_NUM

Maximum number of unprovisioned devices that can be added to device queue

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_PROVISIONER

This option specifies how many unprovisioned devices can be added to device queue for provisioning. Users can use this option to define the size of the queue in the bottom layer which is used to store unprovisioned device information (e.g. Device UUID, address).

CONFIG_BLE_MESH_MAX_PROV_NODES

Maximum number of devices that can be provisioned by Provisioner

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_PROVISIONER

This option specifies how many devices can be provisioned by a Provisioner. This value indicates the maximum number of unprovisioned devices which can be provisioned by a Provisioner. For instance, if the value is 6, it means the Provisioner can provision up to 6 unprovisioned devices. Theoretically a Provisioner without the limitation of its memory can provision up to 32766 unprovisioned devices, here we limit the maximum number to 100 just to limit the memory used by a Provisioner. The bigger the value is, the more memory it will cost by a Provisioner to store the information of nodes.

CONFIG_BLE_MESH_PBA_SAME_TIME

Maximum number of PB-ADV running at the same time by Provisioner

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_PROVISIONER

This option specifies how many devices can be provisioned at the same time using PB-ADV. For examples, if the value is 2, it means a Provisioner can provision two unprovisioned devices with PB-ADV at the same time.

CONFIG_BLE_MESH_PBG_SAME_TIME

Maximum number of PB-GATT running at the same time by Provisioner

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_PROVISIONER

This option specifies how many devices can be provisioned at the same time using PB-GATT. For example, if the value is 2, it means a Provisioner can provision two unprovisioned devices with PB-GATT at the same time.

CONFIG_BLE_MESH_PROVISIONER_SUBNET_COUNT

Maximum number of mesh subnets that can be created by Provisioner

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_PROVISIONER

This option specifies how many subnets per network a Provisioner can create. Indeed, this value decides the number of network keys which can be added by a Provisioner.

CONFIG_BLE_MESH_PROVISIONER_APP_KEY_COUNT

Maximum number of application keys that can be owned by Provisioner

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_PROVISIONER

This option specifies how many application keys the Provisioner can have. Indeed, this value decides the number of the application keys which can be added by a Provisioner.

CONFIG_BLE_MESH_PROV

BLE Mesh Provisioning support

Found in: Component config > CONFIG_BLE_MESH

Enable this option to support BLE Mesh Provisioning functionality. For BLE Mesh, this option should be always enabled.

CONFIG_BLE_MESH_PB_ADV

Provisioning support using the advertising bearer (PB-ADV)

Found in: Component config > CONFIG_BLE_MESH

Enable this option to allow the device to be provisioned over the advertising bearer. This option should be enabled if PB-ADV is going to be used during provisioning procedure.

CONFIG_BLE_MESH_PB_GATT

Provisioning support using GATT (PB-GATT)

Found in: Component config > CONFIG_BLE_MESH

Enable this option to allow the device to be provisioned over GATT. This option should be enabled if PB-GATT is going to be used during provisioning procedure.

Virtual option enabled whenever any Proxy protocol is needed

CONFIG_BLE_MESH_PROXY

BLE Mesh Proxy protocol support

Found in: Component config > CONFIG_BLE_MESH

Enable this option to support BLE Mesh Proxy protocol used by PB-GATT and other proxy pdu transmission.

CONFIG_BLE_MESH_GATT_PROXY_SERVER

BLE Mesh GATT Proxy Server

Found in: Component config > CONFIG_BLE_MESH

This option enables support for Mesh GATT Proxy Service, i.e. the ability to act as a proxy between a Mesh GATT Client and a Mesh network. This option should be enabled if a node is going to be a Proxy Server.

CONFIG_BLE_MESH_NODE_ID_TIMEOUT

Node Identity advertising timeout

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_GATT_PROXY_SERVER

This option determines for how long the local node advertises using Node Identity. The given value is in seconds. The specification limits this to 60 seconds and lists it as the recommended value as well. So leaving the default value is the safest option. When an unprovisioned device is provisioned successfully and becomes a node, it will start to advertise using Node Identity during the time set by this option. And after that, Network ID will be advertised.

CONFIG_BLE_MESH_PROXY_FILTER_SIZE

Maximum number of filter entries per Proxy Client

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_GATT_PROXY_SERVER

This option specifies how many Proxy Filter entries the local node supports. The entries of Proxy filter (whitelist or blacklist) are used to store a list of addresses which can be used to decide which messages will be forwarded to the Proxy Client by the Proxy Server.

CONFIG_BLE_MESH_GATT_PROXY_CLIENT

BLE Mesh GATT Proxy Client

Found in: Component config > CONFIG_BLE_MESH

This option enables support for Mesh GATT Proxy Client. The Proxy Client can use the GATT bearer to send mesh messages to a node that supports the advertising bearer.

CONFIG_BLE_MESH_NET_BUF_POOL_USAGE

BLE Mesh net buffer pool usage tracking

Found in: Component config > CONFIG_BLE_MESH

Enable BLE Mesh net buffer pool tracking. This option is used to introduce another variable in the bottom layer to record the usage of advertising buffers of BLE Mesh devices. Recommend to enable this option as default.

CONFIG_BLE_MESH_SETTINGS

Store BLE Mesh configuration persistently

Found in: Component config > CONFIG_BLE_MESH

When selected, the BLE Mesh stack will take care of storing/restoring the BLE Mesh configuration persistently in flash. If the device is a BLE Mesh node, when this option is enabled, the configuration of the device will be stored persistently, including unicast address, NetKey, AppKey, etc. And if the device is a BLE Mesh Provisioner, the information of the device will be stored persistently, including the information of provisioned nodes, NetKey, AppKey, etc.

CONFIG_BLE_MESH_SPECIFIC_PARTITION

Use a specific NVS partition for BLE Mesh

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_SETTINGS

When selected, the mesh stack will use a specified NVS partition instead of default NVS partition. Note that the specified partition must be registered with NVS using nvs_flash_init_partition() API, and the partition must exists in the csv file. When Provisioner needs to store a large amount of nodes' information in the flash (e.g. more than 20), this option is recommended to be enabled.

CONFIG_BLE_MESH_PARTITION_NAME

Name of the NVS partition for BLE Mesh

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_SETTINGS > CONFIG_BLE_MESH_SPECIFIC_PARTITION

This value defines the name of the specified NVS partition used by the mesh stack.

CONFIG_BLE_MESH_STORE_TIMEOUT

Delay (in seconds) before storing anything persistently

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_SETTINGS

This value defines in seconds how soon any pending changes are actually written into persistent storage (flash) after a change occurs. The option allows nodes to delay a certain period of time to save proper information to flash. The default value is 0, which means information will be stored immediately once there are updates.

CONFIG_BLE_MESH_SEQ_STORE_RATE

How often the sequence number gets updated in storage

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_SETTINGS

This value defines how often the local sequence number gets updated in persistent storage (i.e. flash). e.g. a value of 100 means that the sequence number will be stored to flash on every 100th increment. If the node sends messages very frequently a higher value makes more sense, whereas if the node sends infrequently a value as low as 0 (update storage for every increment) can make sense. When the stack gets initialized it will add sequence number to the last stored one, so that it starts off with a value that' s guaranteed to be larger than the last one used before power off.

CONFIG_BLE_MESH_RPL_STORE_TIMEOUT

Minimum frequency that the RPL gets updated in storage

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_SETTINGS

This value defines in seconds how soon the RPL (Replay Protection List) gets written to persistent storage after a change occurs. If the node receives messages frequently, then a large value is recommended. If the node receives messages rarely, then the value can be as low as 0 (which means the RPL is written into the storage immediately). Note that if the node operates in a security-sensitive case, and there is a risk of sudden power-off, then a value of 0 is strongly recommended. Otherwise, a power loss before RPL being written into the storage may introduce message replay attacks and system security will be in a vulnerable state.

CONFIG_BLE_MESH_SETTINGS_BACKWARD_COMPATIBILITY

A specific option for settings backward compatibility

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_SETTINGS

This option is created to solve the issue of failure in recovering node information after mesh stack updates. In the old version mesh stack, there is no key of "mesh/role" in nvs. In the new version mesh stack, key of "mesh/role" is added in nvs, recovering node information needs to check "mesh/role" key in nvs and implements selective recovery of mesh node information. Therefore, there may be failure in recovering node information during node restarting after OTA.

The new version mesh stack adds the option of "mesh/role" because we have added the support of storing Provisioner information, while the old version only supports storing node information.

If users are updating their nodes from old version to new version, we recommend enabling this option, so that system could set the flag in advance before recovering node information and make sure the node information recovering could work as expected.

CONFIG_BLE_MESH_SUBNET_COUNT

Maximum number of mesh subnets per network

Found in: Component config > CONFIG_BLE_MESH

This option specifies how many subnets a Mesh network can have at the same time. Indeed, this value decides the number of the network keys which can be owned by a node.

CONFIG_BLE_MESH_APP_KEY_COUNT

Maximum number of application keys per network

Found in: Component config > CONFIG_BLE_MESH

This option specifies how many application keys the device can store per network. Indeed, this value decides the number of the application keys which can be owned by a node.

CONFIG_BLE_MESH_MODEL_KEY_COUNT

Maximum number of application keys per model

Found in: Component config > CONFIG_BLE_MESH

This option specifies the maximum number of application keys to which each model can be bound.

CONFIG_BLE_MESH_MODEL_GROUP_COUNT

Maximum number of group address subscriptions per model

Found in: Component config > CONFIG_BLE_MESH

This option specifies the maximum number of addresses to which each model can be subscribed.

CONFIG_BLE_MESH_LABEL_COUNT

Maximum number of Label UUIDs used for Virtual Addresses

Found in: Component config > CONFIG_BLE_MESH

This option specifies how many Label UUIDs can be stored. Indeed, this value decides the number of the Virtual Addresses can be supported by a node.

CONFIG_BLE_MESH_CRPL

Maximum capacity of the replay protection list

Found in: Component config > CONFIG_BLE_MESH

This option specifies the maximum capacity of the replay protection list. It is similar to Network message cache size, but has a different purpose. The replay protection list is used to prevent a node from replay attack, which will store the source address and sequence number of the received mesh messages. For Provisioner, the replay protection list size should not be smaller than the maximum number of nodes whose information can be stored. And the element number of each node should also be taken into consideration. For example, if Provisioner can provision up to 20 nodes and each node contains two elements, then the replay protection list size of Provisioner should be at least 40.

CONFIG_BLE_MESH_MSG_CACHE_SIZE

Network message cache size

Found in: Component config > CONFIG_BLE_MESH

Number of messages that are cached for the network. This helps prevent unnecessary decryption operations and unnecessary relays. This option is similar to Replay protection list, but has a different purpose. A node is not required to cache the entire Network PDU and may cache only part of it for tracking, such as values for SRC/SEQ or others.

CONFIG_BLE_MESH_ADV_BUF_COUNT

Number of advertising buffers

Found in: Component config > CONFIG_BLE_MESH

Number of advertising buffers available. The transport layer reserves ADV_BUF_COUNT - 3 buffers for outgoing segments. The maximum outgoing SDU size is 12 times this value (out of which 4 or 8 bytes are used for the Transport Layer MIC). For example, 5 segments means the maximum SDU size is 60 bytes, which leaves 56 bytes for application layer data using a 4-byte MIC, or 52 bytes using an 8-byte MIC.

CONFIG_BLE_MESH_SUPPORT_BLE_ADV

Support sending normal BLE advertising packets

Found in: Component config > CONFIG_BLE_MESH

When selected, users can send normal BLE advertising packets with specific API.

CONFIG_BLE_MESH_BLE_ADV_BUF_COUNT

Number of advertising buffers for BLE advertising packets

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_SUPPORT_BLE_ADV

Number of advertising buffers for BLE packets available.

CONFIG_BLE_MESH_IVU_DIVIDER

Divider for IV Update state refresh timer

Found in: Component config > CONFIG_BLE_MESH

When the IV Update state enters Normal operation or IV Update in Progress, we need to keep track of how many hours has passed in the state, since the specification requires us to remain in the state at least for 96 hours (Update in Progress has an additional upper limit of 144 hours).

In order to fulfill the above requirement, even if the node might be powered off once in a while, we need to store persistently how many hours the node has been in the state. This doesn' t necessarily need to happen every hour (thanks to the flexible duration range). The exact cadence will depend a lot on the ways that the node will be used and what kind of power source it has.

Since there is no single optimal answer, this configuration option allows specifying a divider, i.e. how many intervals the 96 hour minimum gets split into. After each interval the duration that

the node has been in the current state gets stored to flash. E.g. the default value of 4 means that the state is saved every 24 hours (96 / 4).

CONFIG_BLE_MESH_TX_SEG_MSG_COUNT

Maximum number of simultaneous outgoing segmented messages

Found in: *Component config* > *CONFIG_BLE_MESH*

Maximum number of simultaneous outgoing multi-segment and/or reliable messages. The default value is 1, which means the device can only send one segmented message at a time. And if another segmented message is going to be sent, it should wait for the completion of the previous one. If users are going to send multiple segmented messages at the same time, this value should be configured properly.

CONFIG_BLE_MESH_RX_SEG_MSG_COUNT

Maximum number of simultaneous incoming segmented messages

Found in: *Component config* > *CONFIG_BLE_MESH*

Maximum number of simultaneous incoming multi-segment and/or reliable messages. The default value is 1, which means the device can only receive one segmented message at a time. And if another segmented message is going to be received, it should wait for the completion of the previous one. If users are going to receive multiple segmented messages at the same time, this value should be configured properly.

CONFIG_BLE_MESH_RX_SDU_MAX

Maximum incoming Upper Transport Access PDU length

Found in: *Component config* > *CONFIG_BLE_MESH*

Maximum incoming Upper Transport Access PDU length. Leave this to the default value, unless you really need to optimize memory usage.

CONFIG_BLE_MESH_TX_SEG_MAX

Maximum number of segments in outgoing messages

Found in: *Component config* > *CONFIG_BLE_MESH*

Maximum number of segments supported for outgoing messages. This value should typically be fine-tuned based on what models the local node supports, i.e. what' s the largest message payload that the node needs to be able to send. This value affects memory and call stack consumption,

which is why the default is lower than the maximum that the specification would allow (32 segments).

The maximum outgoing SDU size is 12 times this number (out of which 4 or 8 bytes is used for the Transport Layer MIC). For example, 5 segments means the maximum SDU size is 60 bytes, which leaves 56 bytes for application layer data using a 4-byte MIC and 52 bytes using an 8-byte MIC.

Be sure to specify a sufficient number of advertising buffers when setting this option to a higher value. There must be at least three more advertising buffers (BLE_MESH_ADV_BUF_COUNT) as there are outgoing segments.

CONFIG_BLE_MESH_RELAY

Relay support

Found in: Component config > CONFIG_BLE_MESH

Support for acting as a Mesh Relay Node. Enabling this option will allow a node to support the Relay feature, and the Relay feature can still be enabled or disabled by proper configuration messages. Disabling this option will let a node not support the Relay feature.

CONFIG_BLE_MESH_RELAY_ADV_BUF

Use separate advertising buffers for relay packets

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_RELAY

When selected, self-send packets will be put in a high-priority queue and relay packets will be put in a low-priority queue.

CONFIG_BLE_MESH_RELAY_ADV_BUF_COUNT

Number of advertising buffers for relay packets

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_RELAY > CONFIG_BLE_MESH_RELAY_ADV_BUF

Number of advertising buffers for relay packets available.

CONFIG_BLE_MESH_LOW_POWER

Support for Low Power features

Found in: Component config > CONFIG_BLE_MESH

Enable this option to operate as a Low Power Node. If low power consumption is required by a node, this option should be enabled. And once the node enters the mesh network, it will try to find a Friend node and establish a friendship.

CONFIG_BLE_MESH_LPN_ESTABLISHMENT

Perform Friendship establishment using low power

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_LOW_POWER

Perform the Friendship establishment using low power with the help of a reduced scan duty cycle. The downside of this is that the node may miss out on messages intended for it until it has successfully set up Friendship with a Friend node. When this option is enabled, the node will stop scanning for a period of time after a Friend Request or Friend Poll is sent, so as to reduce more power consumption.

CONFIG_BLE_MESH_LPN_AUTO

Automatically start looking for Friend nodes once provisioned

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_LOW_POWER

Once provisioned, automatically enable LPN functionality and start looking for Friend nodes. If this option is disabled LPN mode needs to be manually enabled by calling bt_mesh_lpn_set(true). When an unprovisioned device is provisioned successfully and becomes a node, enabling this option will trigger the node starts to send Friend Request at a certain period until it finds a proper Friend node.

CONFIG_BLE_MESH_LPN_AUTO_TIMEOUT

Time from last received message before going to LPN mode

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_LOW_POWER > CONFIG_BLE_MESH_LPN_AUTO

Time in seconds from the last received message, that the node waits out before starting to look for Friend nodes.

CONFIG_BLE_MESH_LPN_RETRY_TIMEOUT

Retry timeout for Friend requests

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_LOW_POWER

Time in seconds between Friend Requests, if a previous Friend Request did not yield any acceptable Friend Offers.

CONFIG_BLE_MESH_LPN_RSSI_FACTOR

RSSIFactor, used in Friend Offer Delay calculation

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_LOW_POWER

The contribution of the RSSI, measured by the Friend node, used in Friend Offer Delay calculations. 0 = 1, 1 = 1.5, 2 = 2, 3 = 2.5. RSSIFactor, one of the parameters carried by Friend Request sent by Low Power node, which is used to calculate the Friend Offer Delay.

CONFIG_BLE_MESH_LPN_RECV_WIN_FACTOR

ReceiveWindowFactor, used in Friend Offer Delay calculation

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_LOW_POWER

The contribution of the supported Receive Window used in Friend Offer Delay calculations. 0 = 1, 1 = 1.5, 2 = 2, 3 = 2.5. ReceiveWindowFactor, one of the parameters carried by Friend Request sent by Low Power node, which is used to calculate the Friend Offer Delay.

CONFIG_BLE_MESH_LPN_MIN_QUEUE_SIZE

Minimum size of the acceptable friend queue (MinQueueSizeLog)

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_LOW_POWER

The MinQueueSizeLog field is defined as log_2(N), where N is the minimum number of maximum size Lower Transport PDUs that the Friend node can store in its Friend Queue. As an example, MinQueueSizeLog value 1 gives N = 2, and value 7 gives N = 128.

CONFIG_BLE_MESH_LPN_RECV_DELAY

Receive delay requested by the local node

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_LOW_POWER

The ReceiveDelay is the time between the Low Power node sending a request and listening for a response. This delay allows the Friend node time to prepare the response. The value is in units of milliseconds.

CONFIG_BLE_MESH_LPN_POLL_TIMEOUT

The value of the PollTimeout timer

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_LOW_POWER

PollTimeout timer is used to measure time between two consecutive requests sent by a Low Power node. If no requests are received the Friend node before the PollTimeout timer expires, then the friendship is considered terminated. The value is in units of 100 milliseconds, so e.g. a value of 300 means 30 seconds. The smaller the value, the faster the Low Power node tries to get messages from corresponding Friend node and vice versa.

CONFIG_BLE_MESH_LPN_INIT_POLL_TIMEOUT

The starting value of the PollTimeout timer

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_LOW_POWER

The initial value of the PollTimeout timer when Friendship is to be established for the first time. After this, the timeout gradually grows toward the actual PollTimeout, doubling in value for each iteration. The value is in units of 100 milliseconds, so e.g. a value of 300 means 30 seconds.

CONFIG_BLE_MESH_LPN_SCAN_LATENCY

Latency for enabling scanning

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_LOW_POWER

Latency (in milliseconds) is the time it takes to enable scanning. In practice, it means how much time in advance of the Receive Window, the request to enable scanning is made.

CONFIG_BLE_MESH_LPN_GROUPS

Number of groups the LPN can subscribe to

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_LOW_POWER

Maximum number of groups to which the LPN can subscribe.

CONFIG_BLE_MESH_FRIEND

Support for Friend feature

Found in: *Component config* > *CONFIG_BLE_MESH*

Enable this option to be able to act as a Friend Node.

CONFIG_BLE_MESH_FRIEND_RECV_WIN

Friend Receive Window

Found in: *Component config* > *CONFIG_BLE_MESH* > *CONFIG_BLE_MESH_FRIEND*

Receive Window in milliseconds supported by the Friend node.

CONFIG_BLE_MESH_FRIEND_QUEUE_SIZE

Minimum number of buffers supported per Friend Queue

Found in: *Component config* > *CONFIG_BLE_MESH* > *CONFIG_BLE_MESH_FRIEND*

Minimum number of buffers available to be stored for each local Friend Queue. This option decides the size of each buffer which can be used by a Friend node to store messages for each Low Power node.

CONFIG_BLE_MESH_FRIEND_SUB_LIST_SIZE

Friend Subscription List Size

Found in: *Component config* > *CONFIG_BLE_MESH* > *CONFIG_BLE_MESH_FRIEND*

Size of the Subscription List that can be supported by a Friend node for a Low Power node. And Low Power node can send Friend Subscription List Add or Friend Subscription List Remove messages to the Friend node to add or remove subscription addresses.

CONFIG_BLE_MESH_FRIEND_LPN_COUNT

Number of supported LPN nodes

Found in: *Component config* > *CONFIG_BLE_MESH* > *CONFIG_BLE_MESH_FRIEND*

Number of Low Power Nodes with which a Friend can have Friendship simultaneously. A Friend node can have friendship with multiple Low Power nodes at the same time, while a Low Power node can only establish friendship with only one Friend node at the same time.

CONFIG_BLE_MESH_FRIEND_SEG_RX

Number of incomplete segment lists per LPN

Found in: Component config > CONFIG_BLE_MESH > CONFIG_BLE_MESH_FRIEND

Number of incomplete segment lists tracked for each Friends' LPN. In other words, this determines from how many elements can segmented messages destined for the Friend queue be received simultaneously.

CONFIG_BLE_MESH_NO_LOG

Disable BLE Mesh debug logs (minimize bin size)

Found in: Component config > CONFIG_BLE_MESH

Select this to save the BLE Mesh related rodata code size. Enabling this option will disable the output of BLE Mesh debug log.

BLE Mesh STACK DEBUG LOG LEVEL

Contains:

- *CONFIG_BLE_MESH_STACK_TRACE_LEVEL*

CONFIG_BLE_MESH_STACK_TRACE_LEVEL

BLE_MESH_STACK

Found in: Component config > CONFIG_BLE_MESH > BLE Mesh STACK DEBUG LOG LEVEL

Define BLE Mesh trace level for BLE Mesh stack.

Available options:

- NONE (BLE_MESH_TRACE_LEVEL_NONE)
- ERROR (BLE_MESH_TRACE_LEVEL_ERROR)
- WARNING (BLE_MESH_TRACE_LEVEL_WARNING)
- INFO (BLE_MESH_TRACE_LEVEL_INFO)

- DEBUG (BLE_MESH_TRACE_LEVEL_DEBUG)
- VERBOSE (BLE_MESH_TRACE_LEVEL_VERBOSE)

BLE Mesh NET BUF DEBUG LOG LEVEL

Contains:

- *CONFIG_BLE_MESH_NET_BUF_TRACE_LEVEL*

CONFIG_BLE_MESH_NET_BUF_TRACE_LEVEL

BLE_MESH_NET_BUF

Found in: Component config > CONFIG_BLE_MESH > BLE Mesh NET BUF DEBUG LOG LEVEL

Define BLE Mesh trace level for BLE Mesh net buffer.

Available options:

- NONE (BLE_MESH_NET_BUF_TRACE_LEVEL_NONE)
- ERROR (BLE_MESH_NET_BUF_TRACE_LEVEL_ERROR)
- WARNING (BLE_MESH_NET_BUF_TRACE_LEVEL_WARNING)
- INFO (BLE_MESH_NET_BUF_TRACE_LEVEL_INFO)
- DEBUG (BLE_MESH_NET_BUF_TRACE_LEVEL_DEBUG)
- VERBOSE (BLE_MESH_NET_BUF_TRACE_LEVEL_VERBOSE)

CONFIG_BLE_MESH_CLIENT_MSG_TIMEOUT

Timeout(ms) for client message response

Found in: Component config > CONFIG_BLE_MESH

Timeout value used by the node to get response of the acknowledged message which is sent by the client model. This value indicates the maximum time that a client model waits for the response of the sent acknowledged messages. If a client model uses 0 as the timeout value when sending acknowledged messages, then the default value will be used which is four seconds.

Support for BLE Mesh Client Models

Contains:

- *CONFIG_BLE_MESH_CFG_CLI*

- *CONFIG_BLE_MESH_HEALTH_CLI*
- *CONFIG_BLE_MESH_GENERIC_ONOFF_CLI*
- *CONFIG_BLE_MESH_GENERIC_LEVEL_CLI*
- *CONFIG_BLE_MESH_GENERIC_DEF_TRANS_TIME_CLI*
- *CONFIG_BLE_MESH_GENERIC_POWER_ONOFF_CLI*
- *CONFIG_BLE_MESH_GENERIC_POWER_LEVEL_CLI*
- *CONFIG_BLE_MESH_GENERIC_BATTERY_CLI*
- *CONFIG_BLE_MESH_GENERIC_LOCATION_CLI*
- *CONFIG_BLE_MESH_GENERIC_PROPERTY_CLI*
- *CONFIG_BLE_MESH_SENSOR_CLI*
- *CONFIG_BLE_MESH_TIME_CLI*
- *CONFIG_BLE_MESH_SCENE_CLI*
- *CONFIG_BLE_MESH_SCHEDULER_CLI*
- *CONFIG_BLE_MESH_LIGHT_LIGHTNESS_CLI*
- *CONFIG_BLE_MESH_LIGHT_CTL_CLI*
- *CONFIG_BLE_MESH_LIGHT_HSL_CLI*
- *CONFIG_BLE_MESH_LIGHT_XYL_CLI*
- *CONFIG_BLE_MESH_LIGHT_LC_CLI*

CONFIG_BLE_MESH_CFG_CLI

Configuration Client Model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client Models

Enable support for Configuration client model.

CONFIG_BLE_MESH_HEALTH_CLI

Health Client Model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client Models

Enable support for Health client model.

CONFIG_BLE_MESH_GENERIC_ONOFF_CLI

Generic OnOff Client Model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client Models

Enable support for Generic OnOff client model.

CONFIG_BLE_MESH_GENERIC_LEVEL_CLI

Generic Level Client Model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client Models

Enable support for Generic Level client model.

CONFIG_BLE_MESH_GENERIC_DEF_TRANS_TIME_CLI

Generic Default Transition Time Client Model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client Models

Enable support for Generic Default Transition Time client model.

CONFIG_BLE_MESH_GENERIC_POWER_ONOFF_CLI

Generic Power OnOff Client Model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client Models

Enable support for Generic Power OnOff client model.

CONFIG_BLE_MESH_GENERIC_POWER_LEVEL_CLI

Generic Power Level Client Model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client Models

Enable support for Generic Power Level client model.

CONFIG_BLE_MESH_GENERIC_BATTERY_CLI

Generic Battery Client Model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client Models

Enable support for Generic Battery client model.

CONFIG_BLE_MESH_GENERIC_LOCATION_CLI

Generic Location Client Model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client Models

Enable support for Generic Location client model.

CONFIG_BLE_MESH_GENERIC_PROPERTY_CLI

Generic Property Client Model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client Models

Enable support for Generic Property client model.

CONFIG_BLE_MESH_SENSOR_CLI

Sensor Client Model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client Models

Enable support for Sensor client model.

CONFIG_BLE_MESH_TIME_CLI

Time Client Model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client Models

Enable support for Time client model.

CONFIG_BLE_MESH_SCENE_CLI

Scene Client Model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client Models

Enable support for Scene client model.

CONFIG_BLE_MESH_SCHEDULER_CLI

Scheduler Client Model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client Models

Enable support for Scheduler client model.

CONFIG_BLE_MESH_LIGHT_LIGHTNESS_CLI

Light Lightness Client Model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client Models

Enable support for Light Lightness client model.

CONFIG_BLE_MESH_LIGHT_CTL_CLI

Light CTL Client Model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client Models

Enable support for Light CTL client model.

CONFIG_BLE_MESH_LIGHT_HSL_CLI

Light HSL Client Model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client Models

Enable support for Light HSL client model.

CONFIG_BLE_MESH_LIGHT_XYL_CLI

Light XYL Client Model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client Models

Enable support for Light XYL client model.

CONFIG_BLE_MESH_LIGHT_LC_CLI

Light LC Client Model

Found in: Component config > CONFIG_BLE_MESH > Support for BLE Mesh Client Models

Enable support for Light LC client model.

CONFIG_BLE_MESH_IV_UPDATE_TEST

Test the IV Update Procedure

Found in: Component config > CONFIG_BLE_MESH

This option removes the 96 hour limit of the IV Update Procedure and lets the state to be changed at any time. If IV Update test mode is going to be used, this option should be enabled.

BLE Mesh specific test option

Contains:

- *CONFIG_BLE_MESH_SELF_TEST*
- *CONFIG_BLE_MESH_SHELL*
- *CONFIG_BLE_MESH_DEBUG*

CONFIG_BLE_MESH_SELF_TEST

Perform BLE Mesh self-tests

Found in: Component config > CONFIG_BLE_MESH > BLE Mesh specific test option

This option adds extra self-tests which are run every time BLE Mesh networking is initialized.

CONFIG_BLE_MESH_TEST_AUTO_ENTER_NETWORK

Unprovisioned device enters mesh network automatically

Found in: Component config > CONFIG_BLE_MESH > BLE Mesh specific test option > CONFIG_BLE_MESH_SELF_TEST

With this option enabled, an unprovisioned device can automatically enters mesh network using a specific test function without the pro- visioning procedure. And on the Provisioner side, a test function needs to be invoked to add the node information into the mesh stack.

CONFIG_BLE_MESH_TEST_USE_WHITE_LIST

Use white list to filter mesh advertising packets

Found in: Component config > CONFIG_BLE_MESH > BLE Mesh specific test option > CONFIG_BLE_MESH_SELF_TEST

With this option enabled, users can use white list to filter mesh advertising packets while scanning.

CONFIG_BLE_MESH_SHELL

Enable BLE Mesh shell

Found in: Component config > CONFIG_BLE_MESH > BLE Mesh specific test option

Activate shell module that provides BLE Mesh commands to the console.

CONFIG_BLE_MESH_DEBUG

Enable BLE Mesh debug logs

Found in: Component config > CONFIG_BLE_MESH > BLE Mesh specific test option

Enable debug logs for the BLE Mesh functionality.

CONFIG_BLE_MESH_DEBUG_NET

Network layer debug

Found in: Component config > CONFIG_BLE_MESH > BLE Mesh specific test option > CONFIG_BLE_MESH_DEBUG

Enable Network layer debug logs for the BLE Mesh functionality.

CONFIG_BLE_MESH_DEBUG_TRANS

Transport layer debug

Found in: Component config > CONFIG_BLE_MESH > BLE Mesh specific test option > CONFIG_BLE_MESH_DEBUG

Enable Transport layer debug logs for the BLE Mesh functionality.

CONFIG_BLE_MESH_DEBUG_BEACON

Beacon debug

Found in: Component config > CONFIG_BLE_MESH > BLE Mesh specific test option > CONFIG_BLE_MESH_DEBUG

Enable Beacon-related debug logs for the BLE Mesh functionality.

CONFIG_BLE_MESH_DEBUG_CRYPTO

Crypto debug

Found in: Component config > CONFIG_BLE_MESH > BLE Mesh specific test option > CONFIG_BLE_MESH_DEBUG

Enable cryptographic debug logs for the BLE Mesh functionality.

CONFIG_BLE_MESH_DEBUG_PROV

Provisioning debug

Found in: *Component config* > *CONFIG_BLE_MESH* > *BLE Mesh specific test option* > *CONFIG_BLE_MESH_DEBUG*

Enable Provisioning debug logs for the BLE Mesh functionality.

CONFIG_BLE_MESH_DEBUG_ACCESS

Access layer debug

Found in: *Component config* > *CONFIG_BLE_MESH* > *BLE Mesh specific test option* > *CONFIG_BLE_MESH_DEBUG*

Enable Access layer debug logs for the BLE Mesh functionality.

CONFIG_BLE_MESH_DEBUG_MODEL

Foundation model debug

Found in: *Component config* > *CONFIG_BLE_MESH* > *BLE Mesh specific test option* > *CONFIG_BLE_MESH_DEBUG*

Enable Foundation Models debug logs for the BLE Mesh functionality.

CONFIG_BLE_MESH_DEBUG_ADV

Advertising debug

Found in: *Component config* > *CONFIG_BLE_MESH* > *BLE Mesh specific test option* > *CONFIG_BLE_MESH_DEBUG*

Enable advertising debug logs for the BLE Mesh functionality.

CONFIG_BLE_MESH_DEBUG_LOW_POWER

Low Power debug

Found in: *Component config* > *CONFIG_BLE_MESH* > *BLE Mesh specific test option* > *CONFIG_BLE_MESH_DEBUG*

Enable Low Power debug logs for the BLE Mesh functionality.

CONFIG_BLE_MESH_DEBUG_FRIEND

Friend debug

Found in: *Component config* > *CONFIG_BLE_MESH* > *BLE Mesh specific test option* > *CONFIG_BLE_MESH_DEBUG*

Enable Friend debug logs for the BLE Mesh functionality.

CONFIG_BLE_MESH_DEBUG_PROXY

Proxy debug

Found in: *Component config* > *CONFIG_BLE_MESH* > *BLE Mesh specific test option* > *CONFIG_BLE_MESH_DEBUG*

Enable Proxy protocol debug logs for the BLE Mesh functionality.

CoAP Configuration

Contains:

- *CONFIG_COAP_MBEDTLS_ENCRYPTION_MODE*
- *CONFIG_COAP_MBEDTLS_DEBUG*

CONFIG_COAP_MBEDTLS_ENCRYPTION_MODE

CoAP Encryption method

Found in: *Component config* > *CoAP Configuration*

If the CoAP information is to be encrypted, the encryption environment can be set up in one of two ways (default being Pre-Shared key mode)

- Encrypt using defined Pre-Shared Keys (PSK if uri includes coaps://)
- Encrypt using defined Public Key Infrastructure (PKI if uri includes coaps://)

Available options:

- Pre-Shared Keys (COAP_MBEDTLS_PSK)
- PKI Certificates (COAP_MBEDTLS_PKI)

CONFIG_COAP_MBEDTLS_DEBUG

Enable CoAP debugging

Found in: Component config > CoAP Configuration

Enable CoAP debugging functions at compile time for the example code.

If this option is enabled, call coap_set_log_level() at runtime in order to enable CoAP debug output via the ESP log mechanism.

CONFIG_COAP_MBEDTLS_DEBUG_LEVEL

Set CoAP debugging level

Found in: Component config > CoAP Configuration > CONFIG_COAP_MBEDTLS_DEBUG

Set CoAP debugging level

Available options:

- Emergency (COAP_LOG_EMERG)
- Alert (COAP_LOG_ALERT)
- Critical (COAP_LOG_CRIT)
- Error (COAP_LOG_ERROR)
- Warning (COAP_LOG_WARNING)
- Notice (COAP_LOG_NOTICE)
- Info (COAP_LOG_INFO)
- Debug (COAP_LOG_DEBUG)

Driver configurations

Contains:

- *ADC configuration*
- *SPI configuration*
- *UART configuration*
- *RTCIO configuration*

ADC configuration

Contains:

- *CONFIG_ADC_FORCE_XPD_FSM*
- *CONFIG_ADC_DISABLE_DAC*

CONFIG_ADC_FORCE_XPD_FSM

Use the FSM to control ADC power

Found in: Component config > Driver configurations > ADC configuration

ADC power can be controlled by the FSM instead of software. This allows the ADC to be shut off when it is not working leading to lower power consumption. However using the FSM control ADC power will increase the noise of ADC.

CONFIG_ADC_DISABLE_DAC

Disable DAC when ADC2 is used on GPIO 25 and 26

Found in: Component config > Driver configurations > ADC configuration

If this is set, the ADC2 driver will disable the output of the DAC corresponding to the specified channel. This is the default value.

For testing, disable this option so that we can measure the output of DAC by internal ADC.

SPI configuration

Contains:

- *CONFIG_SPI_MASTER_IN_IRAM*
- *CONFIG_SPI_MASTER_ISR_IN_IRAM*
- *CONFIG_SPI_SLAVE_IN_IRAM*
- *CONFIG_SPI_SLAVE_ISR_IN_IRAM*

CONFIG_SPI_MASTER_IN_IRAM

Place transmitting functions of SPI master into IRAM

Found in: Component config > Driver configurations > SPI configuration

Normally only the ISR of SPI master is placed in the IRAM, so that it can work without the flash when interrupt is triggered. For other functions, there' s some possibility that the flash cache miss when running inside and out of SPI functions, which may increase the interval of SPI transactions. Enable this to put `queue\_trans`, `get\_trans\_result` and `transmit` functions into the IRAM to avoid possible cache miss.

During unit test, this is enabled to measure the ideal case of api.

CONFIG_SPI_MASTER_ISR_IN_IRAM

Place SPI master ISR function into IRAM

Found in: Component config > Driver configurations > SPI configuration

Place the SPI master ISR in to IRAM to avoid possible cache miss.

Also you can forbid the ISR being disabled during flash writing access, by add ESP_INTR_FLAG_IRAM when initializing the driver.

CONFIG_SPI_SLAVE_IN_IRAM

Place transmitting functions of SPI slave into IRAM

Found in: Component config > Driver configurations > SPI configuration

Normally only the ISR of SPI slave is placed in the IRAM, so that it can work without the flash when interrupt is triggered. For other functions, there' s some possibility that the flash cache miss when running inside and out of SPI functions, which may increase the interval of SPI transactions. Enable this to put `queue\_trans`, `get\_trans\_result` and `transmit` functions into the IRAM to avoid possible cache miss.

CONFIG_SPI_SLAVE_ISR_IN_IRAM

Place SPI slave ISR function into IRAM

Found in: Component config > Driver configurations > SPI configuration

Place the SPI slave ISR in to IRAM to avoid possible cache miss.

Also you can forbid the ISR being disabled during flash writing access, by add ESP_INTR_FLAG_IRAM when initializing the driver.

UART configuration

Contains:

- *CONFIG_UART_ISR_IN_IRAM*

CONFIG_UART_ISR_IN_IRAM

Place UART ISR function into IRAM

Found in: Component config > Driver configurations > UART configuration

If this option is not selected, UART interrupt will be disabled for a long time and may cause data lost when doing spi flash operation.

RTCIO configuration

Contains:

- *CONFIG_RTCIO_SUPPORT_RTC_GPIO_DESC*

CONFIG_RTCIO_SUPPORT_RTC_GPIO_DESC

Support array *rtc_gpio_desc* for ESP32

Found in: Component config > Driver configurations > RTCIO configuration

The the array *rtc_gpio_desc* will don't compile by default. If this option is selected, the array *rtc_gpio_desc* can be compile. If user use this array, please enable this configuration.

eFuse Bit Manager

Contains:

- *CONFIG_EFUSE_CUSTOM_TABLE*
- *CONFIG_EFUSE_VIRTUAL*
- *CONFIG_EFUSE_CODE_SCHEME_SELECTOR*

CONFIG_EFUSE_CUSTOM_TABLE

Use custom eFuse table

Found in: Component config > eFuse Bit Manager

Allows to generate a structure for eFuse from the CSV file.

CONFIG_EFUSE_CUSTOM_TABLE_FILENAME

Custom eFuse CSV file

Found in: Component config > eFuse Bit Manager > CONFIG_EFUSE_CUSTOM_TABLE

Name of the custom eFuse CSV filename. This path is evaluated relative to the project root directory.

CONFIG_EFUSE_VIRTUAL

Simulate eFuse operations in RAM

Found in: Component config > eFuse Bit Manager

All read and writes operations are redirected to RAM instead of eFuse registers. If this option is set, all permanent changes (via eFuse) are disabled. Log output will state changes which would be applied, but they will not be.

CONFIG_EFUSE_CODE_SCHEME_SELECTOR

Coding Scheme Compatibility

Found in: Component config > eFuse Bit Manager

Selector eFuse code scheme.

Available options:

- None Only (EFUSE_CODE_SCHEME_COMPAT_NONE)
- 3/4 and None (EFUSE_CODE_SCHEME_COMPAT_3_4)
- Repeat, 3/4 and None (common table does not support it) (EFUSE_CODE_SCHEME_COMPAT_REPEAT)

ESP-TLS

Contains:

- *CONFIG_ESP_TLS_LIBRARY_CHOOSE*
- *CONFIG_ESP_TLS_SERVER*
- *CONFIG_ESP_TLS_PSK_VERIFICATION*
- *CONFIG_ESP_WOLFSSL_SMALL_CERT_VERIFY*
- *CONFIG_ESP_DEBUG_WOLFSSL*

CONFIG_ESP_TLS_LIBRARY_CHOOSE

Choose SSL/TLS library for ESP-TLS (See help for more Info)

Found in: Component config > ESP-TLS

The ESP-TLS APIs support multiple backend TLS libraries. Currently mbedTLS and WolfSSL are supported. Different TLS libraries may support different features and have different resource usage. Consult the ESP-TLS documentation in ESP-IDF Programming guide for more details.

Available options:

- mbedTLS (ESP_TLS_USING_MBEDTLS)
- wolfSSL (License info in wolfSSL directory README) (ESP_TLS_USING_WOLFSSL)

CONFIG_ESP_TLS_SERVER

Enable ESP-TLS Server

Found in: Component config > ESP-TLS

Enable support for creating server side SSL/TLS session, uses the mbedtls crypto library

CONFIG_ESP_TLS_PSK_VERIFICATION

Enable PSK verification

Found in: Component config > ESP-TLS

Enable support for pre shared key ciphers, uses the mbedtls crypto library

CONFIG_ESP_WOLFSSL_SMALL_CERT_VERIFY

Enable SMALL_CERT_VERIFY

Found in: Component config > ESP-TLS

Enables server verification with Intermediate CA cert, does not authenticate full chain of trust upto the root CA cert (After Enabling this option client only needs to have Intermediate CA certificate of the server to authenticate server, root CA cert is not necessary).

CONFIG_ESP_DEBUG_WOLFSSL

Enable debug logs for wolfSSL

Found in: Component config > ESP-TLS

Enable detailed debug prints for wolfSSL SSL library.

ESP32-specific

Contains:

- *CONFIG_ESP32_REV_MIN*
- *CONFIG_ESP32_DEFAULT_CPU_FREQ_MHZ*
- *CONFIG_ESP32_SPIRAM_SUPPORT*
- *CONFIG_ESP32_TRAX*
- *CONFIG_ESP32_UNIVERSAL_MAC_ADDRESSES*
- *CONFIG_ESP32_ULP_COPROC_ENABLED*
- *CONFIG_ESP32_PANIC*
- *CONFIG_ESP32_DEBUG_OCDAWARE*
- *CONFIG_ESP32_BROWNOUT_DET*
- *CONFIG_ESP32_REDUCE_PHY_TX_POWER*
- *CONFIG_ESP32_TIME_SYSCALL*
- *CONFIG_ESP32_RTC_CLK_SRC*
- *CONFIG_ESP32_RTC_EXT_CRYST_ADDIT_CURRENT*
- *CONFIG_ESP32_RTC_CLK_CAL_CYCLES*
- *CONFIG_ESP32_RTC_XTAL_CAL_RETRY*
- *CONFIG_ESP32_RTC_XTAL_BOOTSTRAP_CYCLES*
- *CONFIG_ESP32_DEEP_SLEEP_WAKEUP_DELAY*
- *CONFIG_ESP32_XTAL_FREQ_SEL*
- *CONFIG_ESP32_DISABLE_BASIC_ROM_CONSOLE*
- *CONFIG_ESP32_NO_BLOBS*
- *CONFIG_ESP32_COMPATIBLE_PRE_V2_1_BOOTLOADERS*
- *CONFIG_ESP32_RTCDATA_IN_FAST_MEM*
- *CONFIG_ESP32_USE_FIXED_STATIC_RAM_SIZE*
- *CONFIG_ESP32_DPORT_DIS_INTERRUPT_LVL*

CONFIG_ESP32_REV_MIN

Minimum Supported ESP32 Revision

Found in: Component config > ESP32-specific

Minimum revision that ESP-IDF would support. ESP-IDF performs different strategy on different esp32 revision.

Available options:

- Rev 0 (ESP32_REV_MIN_0)
- Rev 1 (ESP32_REV_MIN_1)
- Rev 2 (ESP32_REV_MIN_2)
- Rev 3 (ESP32_REV_MIN_3)

CONFIG_ESP32_DEFAULT_CPU_FREQ_MHZ

CPU frequency

Found in: Component config > ESP32-specific

CPU frequency to be set on application startup.

Available options:

- 80 MHz (ESP32_DEFAULT_CPU_FREQ_80)
- 160 MHz (ESP32_DEFAULT_CPU_FREQ_160)
- 240 MHz (ESP32_DEFAULT_CPU_FREQ_240)

CONFIG_ESP32_SPIRAM_SUPPORT

Support for external, SPI-connected RAM

Found in: Component config > ESP32-specific

This enables support for an external SPI RAM chip, connected in parallel with the main SPI flash chip.

SPI RAM config

Contains:

- *CONFIG_SPIRAM_TYPE*
- *CONFIG_SPIRAM_SPEED*
- *CONFIG_SPIRAM_BOOT_INIT*
- *CONFIG_SPIRAM_USE*
- *CONFIG_SPIRAM_MEMTEST*

- *CONFIG_SPIRAM_MALLOC_ALWAYSINTERNAL*
- *CONFIG_SPIRAM_TRY_ALLOCATE_WIFI_LWIP*
- *CONFIG_SPIRAM_MALLOC_RESERVE_INTERNAL*
- *CONFIG_SPIRAM_ALLOW_BSS_SEG_EXTERNAL_MEMORY*
- *CONFIG_SPIRAM_CACHE_WORKAROUND*
- *SPIRAM cache workaround debugging*
- *CONFIG_SPIRAM_BANKSWITCH_ENABLE*
- *CONFIG_SPIRAM_ALLOW_STACK_EXTERNAL_MEMORY*
- *CONFIG_SPIRAM_OCCUPY_SPI_HOST*
- *PSRAM clock and cs IO for ESP32-DOWD*
- *PSRAM clock and cs IO for ESP32-D2WD*
- *PSRAM clock and cs IO for ESP32-PICO*
- *CONFIG_SPIRAM_SPIWP_SD3_PIN*
- *CONFIG_SPIRAM_2T_MODE*

CONFIG_SPIRAM_TYPE

Type of SPI RAM chip in use

Found in: Component config > ESP32-specific > CONFIG_ESP32_SPIRAM_SUPPORT > SPI RAM config

Available options:

- Auto-detect (SPIRAM_TYPE_AUTO)
- ESP-PSRAM32 or IS25WP032 (SPIRAM_TYPE_ESPPSRAM32)
- ESP-PSRAM64 or LY68L6400 (SPIRAM_TYPE_ESPPSRAM64)

CONFIG_SPIRAM_SPEED

Set RAM clock speed

Found in: Component config > ESP32-specific > CONFIG_ESP32_SPIRAM_SUPPORT > SPI RAM config

Select the speed for the SPI RAM chip. If SPI RAM is enabled, we only support three combinations of SPI speed mode we supported now:

1. Flash SPI running at 40Mhz and RAM SPI running at 40Mhz

2. Flash SPI running at 80Mhz and RAM SPI running at 40Mhz
3. Flash SPI running at 80Mhz and RAM SPI running at 80Mhz

Note: If the third mode(80Mhz+80Mhz) is enabled for SPI RAM of type 32MBit, one of the HSPI/VSPI host will be occupied by the system. Which SPI host to use can be selected by the config item SPIRAM_OCCUPY_SPI_HOST. Application code should never touch HSPI/VSPI hardware in this case. The option to select 80MHz will only be visible if the flash SPI speed is also 80MHz. (ESPTOOLPY_FLASHFREQ_80M is true)

Available options:

- 40MHz clock speed (SPIRAM_SPEED_40M)
- 80MHz clock speed (SPIRAM_SPEED_80M)

CONFIG_SPIRAM_BOOT_INIT

Initialize SPI RAM during startup

Found in: Component config > ESP32-specific > CONFIG_ESP32_SPIRAM_SUPPORT > SPI RAM config

If this is enabled, the SPI RAM will be enabled during initial boot. Unless you have specific requirements, you' ll want to leave this enabled so memory allocated during boot-up can also be placed in SPI RAM.

CONFIG_SPIRAM_IGNORE_NOTFOUND

Ignore PSRAM when not found

Found in: Component config > ESP32-specific > CONFIG_ESP32_SPIRAM_SUPPORT > SPI RAM config > CONFIG_SPIRAM_BOOT_INIT

Normally, if psram initialization is enabled during compile time but not found at runtime, it is seen as an error making the CPU panic. If this is enabled, booting will complete but no PSRAM will be available.

CONFIG_SPIRAM_USE

SPI RAM access method

Found in: Component config > ESP32-specific > CONFIG_ESP32_SPIRAM_SUPPORT > SPI RAM config

The SPI RAM can be accessed in multiple methods: by just having it available as an unmanaged memory region in the CPU' s memory map, by integrating it in the heap as 'special' memory

needing heap_caps_malloc to allocate, or by fully integrating it making malloc() also able to return SPI RAM pointers.

Available options:

- Integrate RAM into memory map (SPIRAM_USE_MEMMAP)
- Make RAM allocatable using heap_caps_malloc(…, MALLOC_CAP_SPIRAM) (SPIRAM_USE_CAPS_ALLOC)
- Make RAM allocatable using malloc() as well (SPIRAM_USE_MALLOC)

CONFIG_SPIRAM_MEMTEST

Run memory test on SPI RAM initialization

Found in: Component config > ESP32-specific > CONFIG_ESP32_SPIRAM_SUPPORT > SPI RAM config

Runs a rudimentary memory test on initialization. Aborts when memory test fails. Disable this for slightly faster startup.

CONFIG_SPIRAM_MALLOC_ALWAYSINTERNAL

Maximum malloc() size, in bytes, to always put in internal memory

Found in: Component config > ESP32-specific > CONFIG_ESP32_SPIRAM_SUPPORT > SPI RAM config

If malloc() is capable of also allocating SPI-connected ram, its allocation strategy will prefer to allocate chunks less than this size in internal memory, while allocations larger than this will be done from external RAM. If allocation from the preferred region fails, an attempt is made to allocate from the non-preferred region instead, so malloc() will not suddenly fail when either internal or external memory is full.

CONFIG_SPIRAM_TRY_ALLOCATE_WIFI_LWIP

Try to allocate memories of WiFi and LWIP in SPIRAM firstly. If failed, allocate internal memory

Found in: Component config > ESP32-specific > CONFIG_ESP32_SPIRAM_SUPPORT > SPI RAM config

Try to allocate memories of WiFi and LWIP in SPIRAM firstly. If failed, try to allocate internal memory then.

CONFIG_SPIRAM_MALLOC_RESERVE_INTERNAL

Reserve this amount of bytes for data that specifically needs to be in DMA or internal memory

Found in: Component config > ESP32-specific > CONFIG_ESP32_SPIRAM_SUPPORT > SPI RAM config

Because the external/internal RAM allocation strategy is not always perfect, it sometimes may happen that the internal memory is entirely filled up. This causes allocations that are specifically done in internal memory, for example the stack for new tasks or memory to service DMA or have memory that' s also available when SPI cache is down, to fail. This option reserves a pool specifically for requests like that; the memory in this pool is not given out when a normal malloc() is called.

Set this to 0 to disable this feature.

Note that because FreeRTOS stacks are forced to internal memory, they will also use this memory pool; be sure to keep this in mind when adjusting this value.

Note also that the DMA reserved pool may not be one single contiguous memory region, depending on the configured size and the static memory usage of the app.

CONFIG_SPIRAM_ALLOW_BSS_SEG_EXTERNAL_MEMORY

Allow .bss segment placed in external memory

Found in: Component config > ESP32-specific > CONFIG_ESP32_SPIRAM_SUPPORT > SPI RAM config

If enabled the option,and add EXT_RAM_ATTR defined your variable,then your variable will be placed in PSRAM instead of internal memory, and placed most of variables of lwip,net802.11,pp,bluedroid library to external memory defaultly.

CONFIG_SPIRAM_CACHE_WORKAROUND

Enable workaround for bug in SPI RAM cache for Rev1 ESP32s

Found in: Component config > ESP32-specific > CONFIG_ESP32_SPIRAM_SUPPORT > SPI RAM config

Revision 1 of the ESP32 has a bug that can cause a write to PSRAM not to take place in some situations when the cache line needs to be fetched from external RAM and an interrupt occurs. This enables a fix in the compiler (-mfix-esp32-psram-cache-issue) that makes sure the specific code that is vulnerable to this will not be emitted.

This will also not use any bits of newlib that are located in ROM, opting for a version that is compiled with the workaround and located in flash instead.

The workaround is not required for ESP32 revision 3 and above.

SPIRAM cache workaround debugging

Contains:

- *CONFIG_SPIRAM_CACHE_WORKAROUND_STRATEGY*

CONFIG_SPIRAM_CACHE_WORKAROUND_STRATEGY

Workaround strategy

Found in: Component config > ESP32-specific > CONFIG_ESP32_SPIRAM_SUPPORT > SPI RAM config > SPIRAM cache workaround debugging

Select the workaround strategy. Note that the strategy for precompiled libraries (libgcc, newlib, bt, wifi) is not affected by this selection.

Unless you know you need a different strategy, it' s suggested you stay with the default MEMW strategy. Note that DUPLDST can interfere with hardware encryption and this will be automatically disabled if this workaround is selected. 'Insert nops' is the workaround that was used in older esp-idf versions. This workaround still can cause faulty data transfers from/to SPI RAM in some situation.

Available options:

- Insert memw after vulnerable instructions (default) (SPIRAM_CACHE_WORKAROUND_STRATEGY_MEMW)
- Duplicate LD/ST for 32-bit, memw for 8/16 bit (SPIRAM_CACHE_WORKAROUND_STRATEGY_DUPLDST)
- Insert nops between vulnerable loads/stores (old strategy, obsolete) (SPIRAM_CACHE_WORKAROUND_STRATEGY_NOPS)

CONFIG_SPIRAM_BANKSWITCH_ENABLE

Enable bank switching for >4MiB external RAM

Found in: Component config > ESP32-specific > CONFIG_ESP32_SPIRAM_SUPPORT > SPI RAM config

The ESP32 only supports 4MiB of external RAM in its address space. The hardware does support larger memories, but these have to be bank-switched in and out of this address space. Enabling this allows you to reserve some MMU pages for this, which allows the use of the esp_himem api to manage these banks.

#Note that this is limited to 62 banks, as esp_spiram_writeback_cache needs some kind of mapping of #some banks below that mark to work. We cannot at this moment guarantee this to exist when himem is #enabled.

If spiram 2T mode is enabled, the size of 64Mbit psram will be changed as 32Mbit, so himem will be unusable.

CONFIG_SPIRAM_BANKSWITCH_RESERVE

Amount of 32K pages to reserve for bank switching

Found in: Component config > ESP32-specific > CONFIG_ESP32_SPIRAM_SUPPORT > SPI RAM config > CONFIG_SPIRAM_BANKSWITCH_ENABLE

Select the amount of banks reserved for bank switching. Note that the amount of RAM allocatable with malloc/esp_heap_alloc_caps will decrease by 32K for each page reserved here.

Note that this reservation is only actually done if your program actually uses the himem API. Without any himem calls, the reservation is not done and the original amount of memory will be available to malloc/esp_heap_alloc_caps.

CONFIG_SPIRAM_ALLOW_STACK_EXTERNAL_MEMORY

Allow external memory as an argument to xTaskCreateStatic

Found in: Component config > ESP32-specific > CONFIG_ESP32_SPIRAM_SUPPORT > SPI RAM config

Because some bits of the ESP32 code environment cannot be recompiled with the cache workaround, normally tasks cannot be safely run with their stack residing in external memory; for this reason xTaskCreate and friends always allocate stack in internal memory and xTaskCreateStatic will check if the memory passed to it is in internal memory. If you have a task that needs a large amount of stack and does not call on ROM code in any way (no direct calls, but also no Bluetooth/WiFi), you can try to disable this and use xTaskCreateStatic to create the tasks stack in external memory.

CONFIG_SPIRAM_OCCUPY_SPI_HOST

SPI host to use for 32MBit PSRAM

Found in: Component config > ESP32-specific > CONFIG_ESP32_SPIRAM_SUPPORT > SPI RAM config

When both flash and PSRAM is working under 80MHz, and the PSRAM is of type 32MBit, one of the HSPI/VSPI host will be used to output the clock. Select which one to use here.

Available options:

- HSPI host (SPI2) (SPIRAM_OCCUPY_HSPI_HOST)
- VSPI host (SPI3) (SPIRAM_OCCUPY_VSPI_HOST)
- Will not try to use any host, will abort if not able to use the PSRAM (SPIRAM_OCCUPY_NO_HOST)

PSRAM clock and cs IO for ESP32-DOWD

Contains:

- *CONFIG_D0WD_PSRAM_CLK_IO*
- *CONFIG_D0WD_PSRAM_CS_IO*

CONFIG_D0WD_PSRAM_CLK_IO

PSRAM CLK IO number

Found in: Component config > ESP32-specific > CONFIG_ESP32_SPIRAM_SUPPORT > SPI RAM config > PSRAM clock and cs IO for ESP32-DOWD

The PSRAM CLOCK IO can be any unused GPIO, user can config it based on hardware design. If user use 1.8V flash and 1.8V psram, this value can only be one of 6, 7, 8, 9, 10, 11, 16, 17.

CONFIG_D0WD_PSRAM_CS_IO

PSRAM CS IO number

Found in: Component config > ESP32-specific > CONFIG_ESP32_SPIRAM_SUPPORT > SPI RAM config > PSRAM clock and cs IO for ESP32-DOWD

The PSRAM CS IO can be any unused GPIO, user can config it based on hardware design. If user use 1.8V flash and 1.8V psram, this value can only be one of 6, 7, 8, 9, 10, 11, 16, 17.

PSRAM clock and cs IO for ESP32-D2WD

Contains:

- *CONFIG_D2WD_PSRAM_CLK_IO*
- *CONFIG_D2WD_PSRAM_CS_IO*

CONFIG_D2WD_PSRAM_CLK_IO

PSRAM CLK IO number

Found in: *Component config* > *ESP32-specific* > *CONFIG_ESP32_SPIRAM_SUPPORT* > *SPI RAM config* > *PSRAM clock and cs IO for ESP32-D2WD*

User can config it based on hardware design. For ESP32-D2WD chip, the psram can only be 1.8V psram, so this value can only be one of 6, 7, 8, 9, 10, 11, 16, 17.

CONFIG_D2WD_PSRAM_CS_IO

PSRAM CS IO number

Found in: *Component config* > *ESP32-specific* > *CONFIG_ESP32_SPIRAM_SUPPORT* > *SPI RAM config* > *PSRAM clock and cs IO for ESP32-D2WD*

User can config it based on hardware design. For ESP32-D2WD chip, the psram can only be 1.8V psram, so this value can only be one of 6, 7, 8, 9, 10, 11, 16, 17.

PSRAM clock and cs IO for ESP32-PICO

Contains:

- *CONFIG_PICO_PSRAM_CS_IO*

CONFIG_PICO_PSRAM_CS_IO

PSRAM CS IO number

Found in: *Component config* > *ESP32-specific* > *CONFIG_ESP32_SPIRAM_SUPPORT* > *SPI RAM config* > *PSRAM clock and cs IO for ESP32-PICO*

The PSRAM CS IO can be any unused GPIO, user can config it based on hardware design.

For ESP32-PICO chip, the psram share clock with flash, so user do not need to configure the clock IO. For the reference hardware design, please refer to https://www.espressif.com/sites/default/files/documentation/esp32-pico-d4_datasheet_en.pdf

CONFIG_SPIRAM_SPIWP_SD3_PIN

SPI PSRAM WP(SD3) Pin when customising pins via eFuse (read help)

Found in: *Component config* > *ESP32-specific* > *CONFIG_ESP32_SPIRAM_SUPPORT* > *SPI RAM config*

This value is ignored unless flash mode is set to DIO or DOUT and the SPI flash pins have been overriden by setting the eFuses SPI_PAD_CONFIG_xxx.

When this is the case, the eFuse config only defines 3 of the 4 Quad I/O data pins. The WP pin (aka ESP32 pin "SD_DATA_3" or SPI flash pin "IO2") is not specified in eFuse. And the psram only has QPI mode, the WP pin is necessary, so we need to configure this value here.

When flash mode is set to QIO or QOUT, the PSRAM WP pin will be set as the value configured in bootloader.

For ESP32-PICO chip, the default value of this config should be 7.

CONFIG_SPIRAM_2T_MODE

Enable SPI PSRAM 2T mode

Found in: Component config > ESP32-specific > CONFIG_ESP32_SPIRAM_SUPPORT > SPI RAM config

Enable this option to fix single bit errors inside 64Mbit PSRAM.

Some 64Mbit PSRAM chips have a hardware issue in the RAM which causes bit errors at multiple fixed bit positions.

Note: If this option is enabled, the 64Mbit PSRAM chip will appear to be 32Mbit in size. Applications will not be affected unless the use the esp_himem APIs, which are not supported in 2T mode.

CONFIG_ESP32_TRAX

Use TRAX tracing feature

Found in: Component config > ESP32-specific

The ESP32 contains a feature which allows you to trace the execution path the processor has taken through the program. This is stored in a chunk of 32K (16K for single-processor) of memory that can't be used for general purposes anymore. Disable this if you do not know what this is.

CONFIG_ESP32_TRAX_TWOBANKS

Reserve memory for tracing both pro as well as app cpu execution

Found in: Component config > ESP32-specific > CONFIG_ESP32_TRAX

The ESP32 contains a feature which allows you to trace the execution path the processor has taken through the program. This is stored in a chunk of 32K (16K for single-processor) of memory that can't be used for general purposes anymore. Disable this if you do not know what this is.

Memory to reverse for trace, used in linker script

CONFIG_ESP32_UNIVERSAL_MAC_ADDRESSES

Number of universally administered (by IEEE) MAC address

Found in: Component config > ESP32-specific

Configure the number of universally administered (by IEEE) MAC addresses. During initialisation, MAC addresses for each network interface are generated or derived from a single base MAC address. If the number of universal MAC addresses is four, all four interfaces (WiFi station, WiFi softap, Bluetooth and Ethernet) receive a universally administered MAC address. These are generated sequentially by adding 0, 1, 2 and 3 (respectively) to the final octet of the base MAC address. If the number of universal MAC addresses is two, only two interfaces (WiFi station and Bluetooth) receive a universally administered MAC address. These are generated sequentially by adding 0 and 1 (respectively) to the base MAC address. The remaining two interfaces (WiFi softap and Ethernet) receive local MAC addresses. These are derived from the universal WiFi station and Bluetooth MAC addresses, respectively. When using the default (Espressif-assigned) base MAC address, either setting can be used. When using a custom universal MAC address range, the correct setting will depend on the allocation of MAC addresses in this range (either 2 or 4 per device.)

Available options:

- Two (ESP32_UNIVERSAL_MAC_ADDRESSES_TWO)
- Four (ESP32_UNIVERSAL_MAC_ADDRESSES_FOUR)

CONFIG_ESP32_ULP_COPROC_ENABLED

Enable Ultra Low Power (ULP) Coprocessor

Found in: Component config > ESP32-specific

Set to 'y' if you plan to load a firmware for the coprocessor.

If this option is enabled, further coprocessor configuration will appear in the Components menu.

CONFIG_ESP32_ULP_COPROC_RESERVE_MEM

RTC slow memory reserved for coprocessor

Found in: Component config > ESP32-specific > CONFIG_ESP32_ULP_COPROC_ENABLED

Bytes of memory to reserve for ULP coprocessor firmware & data.

Data is reserved at the beginning of RTC slow memory.

CONFIG_ESP32_PANIC

Panic handler behaviour

Found in: Component config > ESP32-specific

If FreeRTOS detects unexpected behaviour or an unhandled exception, the panic handler is invoked. Configure the panic handlers action here.

Available options:

- Print registers and halt (ESP32_PANIC_PRINT_HALT)

  Outputs the relevant registers over the serial port and halt the processor. Needs a manual reset to restart.

- Print registers and reboot (ESP32_PANIC_PRINT_REBOOT)

  Outputs the relevant registers over the serial port and immediately reset the processor.

- Silent reboot (ESP32_PANIC_SILENT_REBOOT)

  Just resets the processor without outputting anything

- Invoke GDBStub (ESP32_PANIC_GDBSTUB)

  Invoke gdbstub on the serial port, allowing for gdb to attach to it to do a postmortem of the crash.

CONFIG_ESP32_DEBUG_OCDAWARE

Make exception and panic handlers JTAG/OCD aware

Found in: Component config > ESP32-specific

The FreeRTOS panic and unhandled exception handers can detect a JTAG OCD debugger and instead of panicking, have the debugger stop on the offending instruction.

CONFIG_ESP32_BROWNOUT_DET

Hardware brownout detect & reset

Found in: Component config > ESP32-specific

The ESP32 has a built-in brownout detector which can detect if the voltage is lower than a specific value. If this happens, it will reset the chip in order to prevent unintended behaviour.

CONFIG_ESP32_BROWNOUT_DET_LVL_SEL

Brownout voltage level

Found in: Component config > ESP32-specific > CONFIG_ESP32_BROWNOUT_DET

The brownout detector will reset the chip when the supply voltage is approximately below this level. Note that there may be some variation of brownout voltage level between each ESP32 chip.

#The voltage levels here are estimates, more work needs to be done to figure out the exact voltages #of the brownout threshold levels.

Available options:

- 2.43V +/- 0.05 (ESP32_BROWNOUT_DET_LVL_SEL_0)
- 2.48V +/- 0.05 (ESP32_BROWNOUT_DET_LVL_SEL_1)
- 2.58V +/- 0.05 (ESP32_BROWNOUT_DET_LVL_SEL_2)
- 2.62V +/- 0.05 (ESP32_BROWNOUT_DET_LVL_SEL_3)
- 2.67V +/- 0.05 (ESP32_BROWNOUT_DET_LVL_SEL_4)
- 2.70V +/- 0.05 (ESP32_BROWNOUT_DET_LVL_SEL_5)
- 2.77V +/- 0.05 (ESP32_BROWNOUT_DET_LVL_SEL_6)
- 2.80V +/- 0.05 (ESP32_BROWNOUT_DET_LVL_SEL_7)

CONFIG_ESP32_REDUCE_PHY_TX_POWER

Reduce PHY TX power when brownout reset

Found in: Component config > ESP32-specific

When brownout reset occurs, reduce PHY TX power to keep the code running

Note about the use of "FRC1" name: currently FRC1 timer is not used for # high resolution timekeeping anymore. Instead the esp_timer API, implemented # using FRC2 timer, is used. # FRC1 name in the option name is kept for compatibility.

CONFIG_ESP32_TIME_SYSCALL

Timers used for gettimeofday function

Found in: Component config > ESP32-specific

This setting defines which hardware timers are used to implement 'gettimeofday' and 'time' functions in C library.

- If both high-resolution and RTC timers are used, timekeeping will continue in deep sleep. Time will be reported at 1 microsecond resolution. This is the default, and the recommended option.
- If only high-resolution timer is used, gettimeofday will provide time at microsecond resolution. Time will not be preserved when going into deep sleep mode.
- If only RTC timer is used, timekeeping will continue in deep sleep, but time will be measured at 6.(6) microsecond resolution. Also the gettimeofday function itself may take longer to run.
- If no timers are used, gettimeofday and time functions return -1 and set errno to ENOSYS.
- When RTC is used for timekeeping, two RTC_STORE registers are used to keep time in deep sleep mode.

Available options:

- RTC and high-resolution timer (ESP32_TIME_SYSCALL_USE_RTC_FRC1)
- RTC (ESP32_TIME_SYSCALL_USE_RTC)
- High-resolution timer (ESP32_TIME_SYSCALL_USE_FRC1)
- None (ESP32_TIME_SYSCALL_USE_NONE)

CONFIG_ESP32_RTC_CLK_SRC

RTC clock source

Found in: Component config > ESP32-specific

Choose which clock is used as RTC clock source.

- "Internal 150kHz oscillator" option provides lowest deep sleep current consumption, and does not require extra external components. However frequency stability with respect to temperature is poor, so time may drift in deep/light sleep modes.
- "External 32kHz crystal" provides better frequency stability, at the expense of slightly higher (1uA) deep sleep current consumption.
- "External 32kHz oscillator" allows using 32kHz clock generated by an external circuit. In this case, external clock signal must be connected to 32K_XP pin. Amplitude should be <1.2V in case of sine wave signal, and <1V in case of square wave signal. Common mode voltage should be 0.1 < Vcm < 0.5Vamp, where Vamp is the signal amplitude. Additionally, 1nF capacitor must be connected between 32K_XN pin and ground. 32K_XN pin can not be used as a GPIO in this case.
- "Internal 8.5MHz oscillator divided by 256" option results in higher deep sleep current (by 5uA) but has better frequency stability than the internal 150kHz oscillator. It does not require external components.

Available options:

- Internal 150kHz RC oscillator (ESP32_RTC_CLK_SRC_INT_RC)
- External 32kHz crystal (ESP32_RTC_CLK_SRC_EXT_CRYS)
- External 32kHz oscillator at 32K_XP pin (ESP32_RTC_CLK_SRC_EXT_OSC)
- Internal 8.5MHz oscillator, divided by 256 (~33kHz) (ESP32_RTC_CLK_SRC_INT_8MD256)

CONFIG_ESP32_RTC_EXT_CRYST_ADDIT_CURRENT

Additional current for external 32kHz crystal

Found in: Component config > ESP32-specific

Choose which additional current is used for rtc external crystal.

- With some 32kHz crystal configurations, the X32N and X32P pins may not have enough drive strength to keep the crystal oscillating during deep sleep. If this option is enabled, additional current from touchpad 9 is provided internally to drive the 32kHz crystal. If this option is enabled, deep sleep current is slightly higher (4-5uA) and the touchpad and ULP wakeup sources are not available.

CONFIG_ESP32_RTC_CLK_CAL_CYCLES

Number of cycles for RTC_SLOW_CLK calibration

Found in: Component config > ESP32-specific

When the startup code initializes RTC_SLOW_CLK, it can perform calibration by comparing the RTC_SLOW_CLK frequency with main XTAL frequency. This option sets the number of RTC_SLOW_CLK cycles measured by the calibration routine. Higher numbers increase calibration precision, which may be important for applications which spend a lot of time in deep sleep. Lower numbers reduce startup time.

When this option is set to 0, clock calibration will not be performed at startup, and approximate clock frequencies will be assumed:

- 150000 Hz if internal RC oscillator is used as clock source. For this use value 1024.
- 32768 Hz if the 32k crystal oscillator is used. For this use value 3000 or more. In case more value will help improve the definition of the launch of the crystal. If the crystal could not start, it will be switched to internal RC.

CONFIG_ESP32_RTC_XTAL_CAL_RETRY

Number of attempts to repeat 32k XTAL calibration

Found in: Component config > ESP32-specific

Number of attempts to repeat 32k XTAL calibration before giving up and switching to the internal RC. Increase this option if the 32k crystal oscillator does not start and switches to internal RC.

CONFIG_ESP32_RTC_XTAL_BOOTSTRAP_CYCLES

Bootstrap cycles for external 32kHz crystal

Found in: Component config > ESP32-specific

To reduce the startup time of an external RTC crystal, we bootstrap it with a 32kHz square wave for a fixed number of cycles. Setting 0 will disable bootstrapping (if disabled, the crystal may take longer to start up or fail to oscillate under some conditions).

If this value is too high, a faulty crystal may initially start and then fail. If this value is too low, an otherwise good crystal may not start.

To accurately determine if the crystal has started, set a larger "Number of cycles for RTC_SLOW_CLK calibration" (about 3000).

CONFIG_ESP32_DEEP_SLEEP_WAKEUP_DELAY

Extra delay in deep sleep wake stub (in us)

Found in: Component config > ESP32-specific

When ESP32 exits deep sleep, the CPU and the flash chip are powered on at the same time. CPU will run deep sleep stub first, and then proceed to load code from flash. Some flash chips need sufficient time to pass between power on and first read operation. By default, without any extra delay, this time is approximately 900us, although some flash chip types need more than that.

By default extra delay is set to 2000us. When optimizing startup time for applications which require it, this value may be reduced.

If you are seeing "flash read err, 1000" message printed to the console after deep sleep reset, try increasing this value.

CONFIG_ESP32_XTAL_FREQ_SEL

Main XTAL frequency

Found in: Component config > ESP32-specific

ESP32 currently supports the following XTAL frequencies:

- 26 MHz
- 40 MHz

Startup code can automatically estimate XTAL frequency. This feature uses the internal 8MHz oscillator as a reference. Because the internal oscillator frequency is temperature dependent, it is not recommended to use automatic XTAL frequency detection in applications which need to work at high ambient temperatures and use high-temperature qualified chips and modules.

Available options:

- 40 MHz (ESP32_XTAL_FREQ_40)
- 26 MHz (ESP32_XTAL_FREQ_26)
- Autodetect (ESP32_XTAL_FREQ_AUTO)

CONFIG_ESP32_DISABLE_BASIC_ROM_CONSOLE

Permanently disable BASIC ROM Console

Found in: Component config > ESP32-specific

If set, the first time the app boots it will disable the BASIC ROM Console permanently (by burning an eFuse).

Otherwise, the BASIC ROM Console starts on reset if no valid bootloader is read from the flash.

(Enabling secure boot also disables the BASIC ROM Console by default.)

CONFIG_ESP32_NO_BLOBS

No Binary Blobs

Found in: Component config > ESP32-specific

If enabled, this disables the linking of binary libraries in the application build. Note that after enabling this Wi-Fi/Bluetooth will not work.

CONFIG_ESP32_COMPATIBLE_PRE_V2_1_BOOTLOADERS

App compatible with bootloaders before IDF v2.1

Found in: Component config > ESP32-specific

Bootloaders before IDF v2.1 did less initialisation of the system clock. This setting needs to be enabled to build an app which can be booted by these older bootloaders.

If this setting is enabled, the app can be booted by any bootloader from IDF v1.0 up to the current version.

If this setting is disabled, the app can only be booted by bootloaders from IDF v2.1 or newer.

Enabling this setting adds approximately 1KB to the app' s IRAM usage.

CONFIG_ESP32_RTCDATA_IN_FAST_MEM

Place RTC_DATA_ATTR and RTC_RODATA_ATTR variables into RTC fast memory segment

Found in: Component config > ESP32-specific

This option allows to place .rtc_data and .rtc_rodata sections into RTC fast memory segment to free the slow memory region for ULP programs. This option depends on the CONFIG_FREERTOS_UNICORE option because RTC fast memory can be accessed only by PRO_CPU core.

CONFIG_ESP32_USE_FIXED_STATIC_RAM_SIZE

Use fixed static RAM size

Found in: Component config > ESP32-specific

If this option is disabled, the DRAM part of the heap starts right after the .bss section, within the dram0_0 region. As a result, adding or removing some static variables will change the available heap size.

If this option is enabled, the DRAM part of the heap starts right after the dram0_0 region, where its length is set with ESP32_FIXED_STATIC_RAM_SIZE

CONFIG_ESP32_FIXED_STATIC_RAM_SIZE

Fixed Static RAM size

Found in: Component config > ESP32-specific > CONFIG_ESP32_USE_FIXED_STATIC_RAM_SIZE

RAM size dedicated for static variables (.data & .bss sections). Please note that the actual length will be reduced by BT_RESERVE_DRAM if Bluetooth controller is enabled.

CONFIG_ESP32_DPORT_DIS_INTERRUPT_LVL

Disable the interrupt level for the DPORT workarounds

Found in: Component config > ESP32-specific

To prevent interrupting DPORT workarounds, need to disable interrupt with a maximum used level in the system.

Power Management

Contains:

- *CONFIG_PM_ENABLE*

CONFIG_PM_ENABLE

Support for power management

Found in: Component config > Power Management

If enabled, application is compiled with support for power management. This option has run-time overhead (increased interrupt latency, longer time to enter idle state), and it also reduces accuracy of RTOS ticks and timers used for timekeeping. Enable this option if application uses power management APIs.

CONFIG_PM_DFS_INIT_AUTO

Enable dynamic frequency scaling (DFS) at startup

Found in: Component config > Power Management > CONFIG_PM_ENABLE

If enabled, startup code configures dynamic frequency scaling. Max CPU frequency is set to CONFIG_ESP32_DEFAULT_CPU_FREQ_MHZ setting, min frequency is set to XTAL frequency. If disabled, DFS will not be active until the application configures it using esp_pm_configure function.

CONFIG_PM_USE_RTC_TIMER_REF

Use RTC timer to prevent time drift (EXPERIMENTAL)

Found in: Component config > Power Management > CONFIG_PM_ENABLE

When APB clock frequency changes, high-resolution timer (esp_timer) scale and base value need to be adjusted. Each adjustment may cause small error, and over time such small errors may cause time drift. If this option is enabled, RTC timer will be used as a reference to compensate for the drift. It is recommended that this option is only used if 32k XTAL is selected as RTC clock source.

CONFIG_PM_PROFILING

Enable profiling counters for PM locks

Found in: Component config > Power Management > CONFIG_PM_ENABLE

If enabled, esp_pm_* functions will keep track of the amount of time each of the power management locks has been held, and esp_pm_dump_locks function will print this information. This feature can be used to analyze which locks are preventing the chip from going into a lower power state, and see what time the chip spends in each power saving mode. This feature does incur some run-time overhead, so should typically be disabled in production builds.

CONFIG_PM_TRACE

Enable debug tracing of PM using GPIOs

Found in: Component config > Power Management > CONFIG_PM_ENABLE

If enabled, some GPIOs will be used to signal events such as RTOS ticks, frequency switching, entry/exit from idle state. Refer to pm_trace.c file for the list of GPIOs. This feature is intended to be used when analyzing/debugging behavior of power management implementation, and should be kept disabled in applications.

ADC-Calibration

Contains:

- *CONFIG_ADC_CAL_EFUSE_TP_ENABLE*
- *CONFIG_ADC_CAL_EFUSE_VREF_ENABLE*
- *CONFIG_ADC_CAL_LUT_ENABLE*

CONFIG_ADC_CAL_EFUSE_TP_ENABLE

Use Two Point Values

Found in: Component config > ADC-Calibration

Some ESP32s have Two Point calibration values burned into eFuse BLOCK3. This option will allow the ADC calibration component to characterize the ADC-Voltage curve using Two Point values if they are available.

CONFIG_ADC_CAL_EFUSE_VREF_ENABLE

Use eFuse Vref

Found in: Component config > ADC-Calibration

Some ESP32s have Vref burned into eFuse BLOCK0. This option will allow the ADC calibration component to characterize the ADC-Voltage curve using eFuse Vref if it is available.

CONFIG_ADC_CAL_LUT_ENABLE

Use Lookup Tables

Found in: Component config > ADC-Calibration

This option will allow the ADC calibration component to use Lookup Tables to correct for non-linear behavior in 11db attenuation. Other attenuations do not exhibit non-linear behavior hence will not be affected by this option.

Common ESP-related

Contains:

- *CONFIG_ESP_TIMER_PROFILING*
- *CONFIG_ESP_ERR_TO_NAME_LOOKUP*
- *CONFIG_ESP_SYSTEM_EVENT_QUEUE_SIZE*
- *CONFIG_ESP_SYSTEM_EVENT_TASK_STACK_SIZE*
- *CONFIG_ESP_MAIN_TASK_STACK_SIZE*
- *CONFIG_ESP_IPC_TASK_STACK_SIZE*
- *CONFIG_ESP_IPC_USES_CALLERS_PRIORITY*
- *CONFIG_ESP_TIMER_TASK_STACK_SIZE*
- *CONFIG_ESP_MINIMAL_SHARED_STACK_SIZE*
- *CONFIG_ESP_CONSOLE_UART*
- *CONFIG_ESP_CONSOLE_UART_NUM*
- *CONFIG_ESP_CONSOLE_UART_TX_GPIO*
- *CONFIG_ESP_CONSOLE_UART_RX_GPIO*
- *CONFIG_ESP_CONSOLE_UART_BAUDRATE*
- *CONFIG_ESP_INT_WDT*
- *CONFIG_ESP_TASK_WDT*
- *CONFIG_ESP_PANIC_HANDLER_IRAM*

CONFIG_ESP_TIMER_PROFILING

Enable esp_timer profiling features

Found in: Component config > Common ESP-related

If enabled, esp_timer_dump will dump information such as number of times the timer was started, number of times the timer has triggered, and the total time it took for the callback to run. This option has some effect on timer performance and the amount of memory used for timer storage, and should only be used for debugging/testing purposes.

CONFIG_ESP_ERR_TO_NAME_LOOKUP

Enable lookup of error code strings

Found in: Component config > Common ESP-related

Functions esp_err_to_name() and esp_err_to_name_r() return string representations of error codes from a pre-generated lookup table. This option can be used to turn off the use of the look-up table in order to save memory but this comes at the price of sacrificing distinguishable (meaningful) output string representations.

CONFIG_ESP_SYSTEM_EVENT_QUEUE_SIZE

System event queue size

Found in: Component config > Common ESP-related

Config system event queue size in different application.

CONFIG_ESP_SYSTEM_EVENT_TASK_STACK_SIZE

Event loop task stack size

Found in: Component config > Common ESP-related

Config system event task stack size in different application.

CONFIG_ESP_MAIN_TASK_STACK_SIZE

Main task stack size

Found in: Component config > Common ESP-related

Configure the "main task" stack size. This is the stack of the task which calls app_main(). If app_main() returns then this task is deleted and its stack memory is freed.

CONFIG_ESP_IPC_TASK_STACK_SIZE

Inter-Processor Call (IPC) task stack size

Found in: Component config > Common ESP-related

Configure the IPC tasks stack size. One IPC task runs on each core (in dual core mode), and allows for cross-core function calls.

See IPC documentation for more details.

The default stack size should be enough for most common use cases. It can be shrunk if you are sure that you do not use any custom IPC functionality.

CONFIG_ESP_IPC_USES_CALLERS_PRIORITY

IPC runs at caller' s priority

Found in: Component config > Common ESP-related

If this option is not enabled then the IPC task will keep behavior same as prior to that of ESP-IDF v4.0, and hence IPC task will run at (configMAX_PRIORITIES - 1) priority.

CONFIG_ESP_TIMER_TASK_STACK_SIZE

High-resolution timer task stack size

Found in: Component config > Common ESP-related

Configure the stack size of esp_timer/ets_timer task. This task is used to dispatch callbacks of timers created using ets_timer and esp_timer APIs. If you are seing stack overflow errors in timer task, increase this value.

Note that this is not the same as FreeRTOS timer task. To configure FreeRTOS timer task size, see "FreeRTOS timer task stack size" option in "FreeRTOS" menu.

CONFIG_ESP_MINIMAL_SHARED_STACK_SIZE

Minimal allowed size for shared stack

Found in: Component config > Common ESP-related

Minimal value of size, in bytes, accepted to execute a expression with shared stack.

CONFIG_ESP_CONSOLE_UART

UART for console output

Found in: Component config > Common ESP-related

Select whether to use UART for console output (through stdout and stderr).

- Default is to use UART0 on pins GPIO1(TX) and GPIO3(RX).
- If "Custom" is selected, UART0 or UART1 can be chosen, and any pins can be selected.
- If "None" is selected, there will be no console output on any UART, except for initial output from ROM bootloader. This output can be further suppressed by bootstrapping GPIO13 pin to low logic level.

Available options:

- Default: UART0, TX=GPIO1, RX=GPIO3 (ESP_CONSOLE_UART_DEFAULT)
- Custom (ESP_CONSOLE_UART_CUSTOM)
- None (ESP_CONSOLE_UART_NONE)

CONFIG_ESP_CONSOLE_UART_NUM

UART peripheral to use for console output (0-1)

Found in: Component config > Common ESP-related

Due of a ROM bug, UART2 is not supported for console output via ets_printf.

Available options:

- UART0 (ESP_CONSOLE_UART_CUSTOM_NUM_0)
- UART1 (ESP_CONSOLE_UART_CUSTOM_NUM_1)

CONFIG_ESP_CONSOLE_UART_TX_GPIO

UART TX on GPIO#

Found in: Component config > Common ESP-related

CONFIG_ESP_CONSOLE_UART_RX_GPIO

UART RX on GPIO#

Found in: Component config > Common ESP-related

CONFIG_ESP_CONSOLE_UART_BAUDRATE

UART console baud rate

Found in: Component config > Common ESP-related

CONFIG_ESP_INT_WDT

Interrupt watchdog

Found in: Component config > Common ESP-related

This watchdog timer can detect if the FreeRTOS tick interrupt has not been called for a certain time, either because a task turned off interrupts and did not turn them on for a long time, or because an interrupt handler did not return. It will try to invoke the panic handler first and failing that reset the SoC.

CONFIG_ESP_INT_WDT_TIMEOUT_MS

Interrupt watchdog timeout (ms)

Found in: Component config > Common ESP-related > CONFIG_ESP_INT_WDT

The timeout of the watchdog, in miliseconds. Make this higher than the FreeRTOS tick rate.

CONFIG_ESP_INT_WDT_CHECK_CPU1

Also watch CPU1 tick interrupt

Found in: Component config > Common ESP-related > CONFIG_ESP_INT_WDT

Also detect if interrupts on CPU 1 are disabled for too long.

CONFIG_ESP_TASK_WDT

Initialize Task Watchdog Timer on startup

Found in: Component config > Common ESP-related

The Task Watchdog Timer can be used to make sure individual tasks are still running. Enabling this option will cause the Task Watchdog Timer to be initialized automatically at startup. The Task Watchdog timer can be initialized after startup as well (see Task Watchdog Timer API Reference)

CONFIG_ESP_TASK_WDT_PANIC

Invoke panic handler on Task Watchdog timeout

Found in: Component config > Common ESP-related > CONFIG_ESP_TASK_WDT

If this option is enabled, the Task Watchdog Timer will be configured to trigger the panic handler when it times out. This can also be configured at run time (see Task Watchdog Timer API Reference)

CONFIG_ESP_TASK_WDT_TIMEOUT_S

Task Watchdog timeout period (seconds)

Found in: Component config > Common ESP-related > CONFIG_ESP_TASK_WDT

Timeout period configuration for the Task Watchdog Timer in seconds. This is also configurable at run time (see Task Watchdog Timer API Reference)

CONFIG_ESP_TASK_WDT_CHECK_IDLE_TASK_CPU0

Watch CPU0 Idle Task

Found in: Component config > Common ESP-related > CONFIG_ESP_TASK_WDT

If this option is enabled, the Task Watchdog Timer will watch the CPU0 Idle Task. Having the Task Watchdog watch the Idle Task allows for detection of CPU starvation as the Idle Task not being called is usually a symptom of CPU starvation. Starvation of the Idle Task is detrimental as FreeRTOS household tasks depend on the Idle Task getting some runtime every now and then.

CONFIG_ESP_TASK_WDT_CHECK_IDLE_TASK_CPU1

Watch CPU1 Idle Task

Found in: Component config > Common ESP-related > CONFIG_ESP_TASK_WDT

If this option is enabled, the Task Wtachdog Timer will wach the CPU1 Idle Task.

CONFIG_ESP_PANIC_HANDLER_IRAM

Place panic handler code in IRAM

Found in: Component config > Common ESP-related

If this option is disabled (default), the panic handler code is placed in flash not IRAM. This means that if ESP-IDF crashes while flash cache is disabled, the panic handler will automatically re-enable flash cache before running GDB Stub or Core Dump. This adds some minor risk, if the flash cache status is also corrupted during the crash.

If this option is enabled, the panic handler code is placed in IRAM. This allows the panic handler to run without needing to re-enable cache first. This may be necessary to debug some complex issues with crashes while flash cache is disabled (for example, when writing to SPI flash.)

Ethernet

Contains:

- *CONFIG_ETH_USE_ESP32_EMAC*
- *CONFIG_ETH_USE_SPI_ETHERNET*
- *CONFIG_ETH_USE_OPENETH*

CONFIG_ETH_USE_ESP32_EMAC

Support ESP32 internal EMAC controller

Found in: Component config > Ethernet

ESP32 integrates a 10/100M Ethernet MAC controller.

Contains:

- *CONFIG_ETH_PHY_INTERFACE*
- *CONFIG_ETH_RMII_CLK_MODE*
- *CONFIG_ETH_RMII_CLK_OUTPUT_GPIO0*
- *CONFIG_ETH_RMII_CLK_OUT_GPIO*
- *CONFIG_ETH_DMA_BUFFER_SIZE*
- *CONFIG_ETH_DMA_RX_BUFFER_NUM*
- *CONFIG_ETH_DMA_TX_BUFFER_NUM*

CONFIG_ETH_PHY_INTERFACE

PHY interface

Found in: Component config > Ethernet > CONFIG_ETH_USE_ESP32_EMAC

Select the communication interface between MAC and PHY chip.

Available options:

- Reduced Media Independent Interface (RMII) (ETH_PHY_INTERFACE_RMII)
- Media Independent Interface (MII) (ETH_PHY_INTERFACE_MII)

CONFIG_ETH_RMII_CLK_MODE

RMII clock mode

Found in: Component config > Ethernet > CONFIG_ETH_USE_ESP32_EMAC

Select external or internal RMII clock.

Available options:

- Input RMII clock from external (ETH_RMII_CLK_INPUT)

  MAC will get RMII clock from outside. Note that ESP32 only supports GPIO0 to input the RMII clock.

- Output RMII clock from internal (ETH_RMII_CLK_OUTPUT)

  ESP32 can generate RMII clock by internal APLL. This clock can be routed to the external PHY device. ESP32 supports to route the RMII clock to GPIO0/16/17.

CONFIG_ETH_RMII_CLK_OUTPUT_GPIO0

Output RMII clock from GPIO0 (Experimental!)

Found in: Component config > Ethernet > CONFIG_ETH_USE_ESP32_EMAC

GPIO0 can be set to output a pre-divided PLL clock (test only!). Enabling this option will configure GPIO0 to output a 50MHz clock. In fact this clock doesn't have directly relationship with EMAC peripheral. Sometimes this clock won't work well with your PHY chip. You might need to add some extra devices after GPIO0 (e.g. inverter). Note that outputting RMII clock on GPIO0 is an experimental practice. If you want the Ethernet to work with WiFi, don't select GPIO0 output mode for stability.

CONFIG_ETH_RMII_CLK_OUT_GPIO

RMII clock GPIO number

Found in: Component config > Ethernet > CONFIG_ETH_USE_ESP32_EMAC

Set the GPIO number to output RMII Clock.

CONFIG_ETH_DMA_BUFFER_SIZE

Ethernet DMA buffer size (Byte)

Found in: Component config > Ethernet > CONFIG_ETH_USE_ESP32_EMAC

Set the size of each buffer used by Ethernet MAC DMA.

CONFIG_ETH_DMA_RX_BUFFER_NUM

Amount of Ethernet DMA Rx buffers

Found in: *Component config* > *Ethernet* > *CONFIG_ETH_USE_ESP32_EMAC*

Number of DMA receive buffers. Each buffer' s size is ETH_DMA_BUFFER_SIZE. Larger number of buffers could increase throughput somehow.

CONFIG_ETH_DMA_TX_BUFFER_NUM

Amount of Ethernet DMA Tx buffers

Found in: *Component config* > *Ethernet* > *CONFIG_ETH_USE_ESP32_EMAC*

Number of DMA transmit buffers. Each buffer' s size is ETH_DMA_BUFFER_SIZE. Larger number of buffers could increase throughput somehow.

CONFIG_ETH_USE_SPI_ETHERNET

Support SPI to Ethernet Module

Found in: *Component config* > *Ethernet*

ESP-IDF can also support some SPI-Ethernet modules.

Contains:

- *CONFIG_ETH_SPI_ETHERNET_DM9051*

CONFIG_ETH_SPI_ETHERNET_DM9051

Use DM9051

Found in: *Component config* > *Ethernet* > *CONFIG_ETH_USE_SPI_ETHERNET*

DM9051 is a fast Ethernet controller with an SPI interface. It' s also integrated with a 10/100M PHY and MAC. Set true to enable DM9051 driver.

CONFIG_ETH_USE_OPENETH

Support OpenCores Ethernet MAC (for use with QEMU)

Found in: *Component config* > *Ethernet*

OpenCores Ethernet MAC driver can be used when an ESP-IDF application is executed in QEMU. This driver is not supported when running on a real chip.

Contains:

- *CONFIG_ETH_OPENETH_DMA_RX_BUFFER_NUM*
- *CONFIG_ETH_OPENETH_DMA_TX_BUFFER_NUM*

CONFIG_ETH_OPENETH_DMA_RX_BUFFER_NUM

Number of Ethernet DMA Rx buffers

Found in: *Component config* > *Ethernet* > *CONFIG_ETH_USE_OPENETH*

Number of DMA receive buffers, each buffer is 1600 bytes.

CONFIG_ETH_OPENETH_DMA_TX_BUFFER_NUM

Number of Ethernet DMA Tx buffers

Found in: *Component config* > *Ethernet* > *CONFIG_ETH_USE_OPENETH*

Number of DMA transmit buffers, each buffer is 1600 bytes.

Event Loop Library

Contains:

- *CONFIG_ESP_EVENT_LOOP_PROFILING*
- *CONFIG_ESP_EVENT_POST_FROM_ISR*

CONFIG_ESP_EVENT_LOOP_PROFILING

Enable event loop profiling

Found in: *Component config* > *Event Loop Library*

Enables collections of statistics in the event loop library such as the number of events posted to/recieved by an event loop, number of callbacks involved, number of events dropped to to a full event loop queue, run time of event handlers, and number of times/run time of each event handler.

CONFIG_ESP_EVENT_POST_FROM_ISR

Support posting events from ISRs

Found in: *Component config* > *Event Loop Library*

Enable posting events from interrupt handlers.

CONFIG_ESP_EVENT_POST_FROM_IRAM_ISR

Support posting events from ISRs placed in IRAM

Found in: Component config > Event Loop Library > CONFIG_ESP_EVENT_POST_FROM_ISR

Enable posting events from interrupt handlers placed in IRAM. Enabling this option places API functions esp_event_post and esp_event_post_to in IRAM.

GDB Stub

Contains:

- *CONFIG_ESP_GDBSTUB_SUPPORT_TASKS*

CONFIG_ESP_GDBSTUB_SUPPORT_TASKS

Enable listing FreeRTOS tasks through GDB Stub

Found in: Component config > GDB Stub

If enabled, GDBStub can supply the list of FreeRTOS tasks to GDB. Thread list can be queried from GDB using 'info threads' command. Note that if GDB task lists were corrupted, this feature may not work. If GDBStub fails, try disabling this feature.

CONFIG_ESP_GDBSTUB_MAX_TASKS

Maximum number of tasks supported by GDB Stub

Found in: Component config > GDB Stub > CONFIG_ESP_GDBSTUB_SUPPORT_TASKS

Set the number of tasks which GDB Stub will support.

ESP HTTP client

Contains:

- *CONFIG_ESP_HTTP_CLIENT_ENABLE_HTTPS*
- *CONFIG_ESP_HTTP_CLIENT_ENABLE_BASIC_AUTH*

CONFIG_ESP_HTTP_CLIENT_ENABLE_HTTPS

Enable https

Found in: Component config > ESP HTTP client

This option will enable https protocol by linking mbedtls library and initializing SSL transport

CONFIG_ESP_HTTP_CLIENT_ENABLE_BASIC_AUTH

Enable HTTP Basic Authentication

Found in: Component config > ESP HTTP client

This option will enable HTTP Basic Authentication. It is disabled by default as Basic auth uses unencrypted encoding, so it introduces a vulnerability when not using TLS

HTTP Server

Contains:

- *CONFIG_HTTPD_MAX_REQ_HDR_LEN*
- *CONFIG_HTTPD_MAX_URI_LEN*
- *CONFIG_HTTPD_ERR_RESP_NO_DELAY*
- *CONFIG_HTTPD_PURGE_BUF_LEN*
- *CONFIG_HTTPD_LOG_PURGE_DATA*

CONFIG_HTTPD_MAX_REQ_HDR_LEN

Max HTTP Request Header Length

Found in: Component config > HTTP Server

This sets the maximum supported size of headers section in HTTP request packet to be processed by the server

CONFIG_HTTPD_MAX_URI_LEN

Max HTTP URI Length

Found in: Component config > HTTP Server

This sets the maximum supported size of HTTP request URI to be processed by the server

CONFIG_HTTPD_ERR_RESP_NO_DELAY

Use TCP_NODELAY socket option when sending HTTP error responses

Found in: Component config > HTTP Server

Using TCP_NODEALY socket option ensures that HTTP error response reaches the client before the underlying socket is closed. Please note that turning this off may cause multiple test failures

CONFIG_HTTPD_PURGE_BUF_LEN

Length of temporary buffer for purging data

Found in: Component config > HTTP Server

This sets the size of the temporary buffer used to receive and discard any remaining data that is received from the HTTP client in the request, but not processed as part of the server HTTP request handler.

If the remaining data is larger than the available buffer size, the buffer will be filled in multiple iterations. The buffer should be small enough to fit on the stack, but large enough to avoid excessive iterations.

CONFIG_HTTPD_LOG_PURGE_DATA

Log purged content data at Debug level

Found in: Component config > HTTP Server

Enabling this will log discarded binary HTTP request data at Debug level. For large content data this may not be desirable as it will clutter the log.

ESP HTTPS OTA

Contains:

- *CONFIG_OTA_ALLOW_HTTP*

CONFIG_OTA_ALLOW_HTTP

Allow HTTP for OTA (WARNING: ONLY FOR TESTING PURPOSE, READ HELP)

Found in: Component config > ESP HTTPS OTA

It is highly recommended to keep HTTPS (along with server certificate validation) enabled. Enabling this option comes with potential risk of: - Non-encrypted communication channel with server - Accepting firmware upgrade image from server with fake identity

ESP HTTPS server

Contains:

- *CONFIG_ESP_HTTPS_SERVER_ENABLE*

CONFIG_ESP_HTTPS_SERVER_ENABLE

Enable ESP_HTTPS_SERVER component

Found in: Component config > ESP HTTPS server

Enable ESP HTTPS server component

ESP NETIF Adapter

Contains:

- *CONFIG_ESP_NETIF_IP_LOST_TIMER_INTERVAL*
- *CONFIG_ESP_NETIF_USE_TCPIP_STACK_LIB*
- *CONFIG_ESP_NETIF_TCPIP_ADAPTER_COMPATIBLE_LAYER*

CONFIG_ESP_NETIF_IP_LOST_TIMER_INTERVAL

IP Address lost timer interval (seconds)

Found in: Component config > ESP NETIF Adapter

The value of 0 indicates the IP lost timer is disabled, otherwise the timer is enabled.

The IP address may be lost because of some reasons, e.g. when the station disconnects from soft-AP, or when DHCP IP renew fails etc. If the IP lost timer is enabled, it will be started everytime the IP is lost. Event SYSTEM_EVENT_STA_LOST_IP will be raised if the timer expires. The IP lost timer is stopped if the station get the IP again before the timer expires.

CONFIG_ESP_NETIF_USE_TCPIP_STACK_LIB

TCP/IP Stack Library

Found in: Component config > ESP NETIF Adapter

Choose the TCP/IP Stack to work, for example, LwIP, uIP, etc.

Available options:

- LwIP (ESP_NETIF_TCPIP_LWIP)

  lwIP is a small independent implementation of the TCP/IP protocol suite.

- Loopback (ESP_NETIF_LOOPBACK)

  Dummy implementation of esp-netif functionality which connects driver transmit to receive function. This option is for testing purpose only

CONFIG_ESP_NETIF_TCPIP_ADAPTER_COMPATIBLE_LAYER

Enable backward compatible tcpip_adapter interface

Found in: Component config > ESP NETIF Adapter

Backward compatible interface to tcpip_adapter is enabled by default to support legacy TCP/IP stack initialisation code. Disable this option to use only esp-netif interface.

Wi-Fi

Contains:

- *CONFIG_ESP32_WIFI_SW_COEXIST_ENABLE*
- *CONFIG_ESP32_WIFI_STATIC_RX_BUFFER_NUM*
- *CONFIG_ESP32_WIFI_DYNAMIC_RX_BUFFER_NUM*
- *CONFIG_ESP32_WIFI_TX_BUFFER*
- *CONFIG_ESP32_WIFI_STATIC_TX_BUFFER_NUM*
- *CONFIG_ESP32_WIFI_DYNAMIC_TX_BUFFER_NUM*
- *CONFIG_ESP32_WIFI_CSI_ENABLED*
- *CONFIG_ESP32_WIFI_AMPDU_TX_ENABLED*
- *CONFIG_ESP32_WIFI_AMPDU_RX_ENABLED*
- *CONFIG_ESP32_WIFI_NVS_ENABLED*
- *CONFIG_ESP32_WIFI_TASK_CORE_ID*
- *CONFIG_ESP32_WIFI_SOFTAP_BEACON_MAX_LEN*
- *CONFIG_ESP32_WIFI_MGMT_SBUF_NUM*
- *CONFIG_ESP32_WIFI_DEBUG_LOG_ENABLE*
- *CONFIG_ESP32_WIFI_IRAM_OPT*
- *CONFIG_ESP32_WIFI_RX_IRAM_OPT*
- *CONFIG_ESP32_WIFI_ENABLE_WPA3_SAE*

CONFIG_ESP32_WIFI_SW_COEXIST_ENABLE

Software controls WiFi/Bluetooth coexistence

Found in: Component config > Wi-Fi

If enabled, WiFi & Bluetooth coexistence is controlled by software rather than hardware. Recommended for heavy traffic scenarios. Both coexistence configuration options are automatically managed, no user intervention is required. If only Bluetooth is used, it is recommended to disable this option to reduce binary file size.

CONFIG_ESP32_WIFI_STATIC_RX_BUFFER_NUM

Max number of WiFi static RX buffers

Found in: Component config > Wi-Fi

Set the number of WiFi static RX buffers. Each buffer takes approximately 1.6KB of RAM. The static rx buffers are allocated when esp_wifi_init is called, they are not freed until esp_wifi_deinit is called.

WiFi hardware use these buffers to receive all 802.11 frames. A higher number may allow higher throughput but increases memory use. If ESP32_WIFI_AMPDU_RX_ENABLED is enabled, this value is recommended to set equal or bigger than ESP32_WIFI_RX_BA_WIN in order to achieve better throughput and compatibility with both stations and APs.

CONFIG_ESP32_WIFI_DYNAMIC_RX_BUFFER_NUM

Max number of WiFi dynamic RX buffers

Found in: Component config > Wi-Fi

Set the number of WiFi dynamic RX buffers, 0 means unlimited RX buffers will be allocated (provided sufficient free RAM). The size of each dynamic RX buffer depends on the size of the received data frame.

For each received data frame, the WiFi driver makes a copy to an RX buffer and then delivers it to the high layer TCP/IP stack. The dynamic RX buffer is freed after the higher layer has successfully received the data frame.

For some applications, WiFi data frames may be received faster than the application can process them. In these cases we may run out of memory if RX buffer number is unlimited (0).

If a dynamic RX buffer limit is set, it should be at least the number of static RX buffers.

CONFIG_ESP32_WIFI_TX_BUFFER

Type of WiFi TX buffers

Found in: Component config > Wi-Fi

Select type of WiFi TX buffers:

If "Static" is selected, WiFi TX buffers are allocated when WiFi is initialized and released when WiFi is de-initialized. The size of each static TX buffer is fixed to about 1.6KB.

If "Dynamic" is selected, each WiFi TX buffer is allocated as needed when a data frame is delivered to the Wifi driver from the TCP/IP stack. The buffer is freed after the data frame has been sent by the WiFi driver. The size of each dynamic TX buffer depends on the length of each data frame sent by the TCP/IP layer.

If PSRAM is enabled, "Static" should be selected to guarantee enough WiFi TX buffers. If PSRAM is disabled, "Dynamic" should be selected to improve the utilization of RAM.

Available options:

- Static (ESP32_WIFI_STATIC_TX_BUFFER)
- Dynamic (ESP32_WIFI_DYNAMIC_TX_BUFFER)

CONFIG_ESP32_WIFI_STATIC_TX_BUFFER_NUM

Max number of WiFi static TX buffers

Found in: Component config > Wi-Fi

Set the number of WiFi static TX buffers. Each buffer takes approximately 1.6KB of RAM. The static RX buffers are allocated when esp_wifi_init() is called, they are not released until esp_wifi_deinit() is called.

For each transmitted data frame from the higher layer TCP/IP stack, the WiFi driver makes a copy of it in a TX buffer. For some applications especially UDP applications, the upper layer can deliver frames faster than WiFi layer can transmit. In these cases, we may run out of TX buffers.

CONFIG_ESP32_WIFI_DYNAMIC_TX_BUFFER_NUM

Max number of WiFi dynamic TX buffers

Found in: Component config > Wi-Fi

Set the number of WiFi dynamic TX buffers. The size of each dynamic TX buffer is not fixed, it depends on the size of each transmitted data frame.

For each transmitted frame from the higher layer TCP/IP stack, the WiFi driver makes a copy of it in a TX buffer. For some applications, especially UDP applications, the upper layer can deliver frames faster than WiFi layer can transmit. In these cases, we may run out of TX buffers.

CONFIG_ESP32_WIFI_CSI_ENABLED

WiFi CSI(Channel State Information)

Found in: Component config > Wi-Fi

Select this option to enable CSI(Channel State Information) feature. CSI takes about CONFIG_ESP32_WIFI_STATIC_RX_BUFFER_NUM KB of RAM. If CSI is not used, it is better to disable this feature in order to save memory.

CONFIG_ESP32_WIFI_AMPDU_TX_ENABLED

WiFi AMPDU TX

Found in: Component config > Wi-Fi

Select this option to enable AMPDU TX feature

CONFIG_ESP32_WIFI_TX_BA_WIN

WiFi AMPDU TX BA window size

Found in: Component config > Wi-Fi > CONFIG_ESP32_WIFI_AMPDU_TX_ENABLED

Set the size of WiFi Block Ack TX window. Generally a bigger value means higher throughput but more memory. Most of time we should NOT change the default value unless special reason, e.g. test the maximum UDP TX throughput with iperf etc. For iperf test in shieldbox, the recommended value is 9~12.

CONFIG_ESP32_WIFI_AMPDU_RX_ENABLED

WiFi AMPDU RX

Found in: Component config > Wi-Fi

Select this option to enable AMPDU RX feature

CONFIG_ESP32_WIFI_RX_BA_WIN

WiFi AMPDU RX BA window size

Found in: Component config > Wi-Fi > CONFIG_ESP32_WIFI_AMPDU_RX_ENABLED

Set the size of WiFi Block Ack RX window. Generally a bigger value means higher throughput and better compatibility but more memory. Most of time we should NOT change the default value unless special reason, e.g. test the maximum UDP RX throughput with iperf etc. For iperf test in shieldbox, the recommended value is 9~12. If PSRAM is used and WiFi memory is prefered to allocat in PSRAM first, the default and minimum value should be 16 to achieve better throughput and compatibility with both stations and APs.

CONFIG_ESP32_WIFI_NVS_ENABLED

WiFi NVS flash

Found in: Component config > Wi-Fi

Select this option to enable WiFi NVS flash

CONFIG_ESP32_WIFI_TASK_CORE_ID

WiFi Task Core ID

Found in: Component config > Wi-Fi

Pinned WiFi task to core 0 or core 1.

Available options:

- Core 0 (ESP32_WIFI_TASK_PINNED_TO_CORE_0)
- Core 1 (ESP32_WIFI_TASK_PINNED_TO_CORE_1)

CONFIG_ESP32_WIFI_SOFTAP_BEACON_MAX_LEN

Max length of WiFi SoftAP Beacon

Found in: Component config > Wi-Fi

ESP-MESH utilizes beacon frames to detect and resolve root node conflicts (see documentation). However the default length of a beacon frame can simultaneously hold only five root node identifier structures, meaning that a root node conflict of up to five nodes can be detected at one time. In the occurence of more root nodes conflict involving more than five root nodes, the conflict resolution process will detect five of the root nodes, resolve the conflict, and re-detect more root nodes. This process will repeat until all root node conflicts are resolved. However this process can generally take a very long time.

To counter this situation, the beacon frame length can be increased such that more root nodes can be detected simultaneously. Each additional root node will require 36 bytes and should be added ontop of the default beacon frame length of 752 bytes. For example, if you want to detect 10 root nodes simultaneously, you need to set the beacon frame length as 932 (752+36*5).

Setting a longer beacon length also assists with debugging as the conflicting root nodes can be identified more quickly.

CONFIG_ESP32_WIFI_MGMT_SBUF_NUM

WiFi mgmt short buffer number

Found in: Component config > Wi-Fi

Set the number of WiFi management short buffer.

CONFIG_ESP32_WIFI_DEBUG_LOG_ENABLE

Enable WiFi debug log

Found in: Component config > Wi-Fi

Select this option to enable WiFi debug log

CONFIG_ESP32_WIFI_DEBUG_LOG_LEVEL

WiFi debug log level

Found in: Component config > Wi-Fi > CONFIG_ESP32_WIFI_DEBUG_LOG_ENABLE

The WiFi log is divided into the following levels: ERROR,WARNING,INFO,DEBUG,VERBOSE. The ERROR,WARNING,INFO levels are enabled by default, and the DEBUG,VERBOSE levels can be enabled here.

Available options:

- WiFi Debug Log Debug (ESP32_WIFI_DEBUG_LOG_DEBUG)
- WiFi Debug Log Verbose (ESP32_WIFI_DEBUG_LOG_VERBOSE)

CONFIG_ESP32_WIFI_DEBUG_LOG_MODULE

WiFi debug log module

Found in: Component config > Wi-Fi > CONFIG_ESP32_WIFI_DEBUG_LOG_ENABLE

The WiFi log module contains three parts: WIFI,COEX,MESH. The WIFI module indicates the logs related to WiFi, the COEX module indicates the logs related to WiFi and BT(or BLE) coexist, the MESH module indicates the logs related to Mesh. When ESP32_WIFI_LOG_MODULE_ALL is enabled, all modules are selected.

Available options:

- WiFi Debug Log Module All (ESP32_WIFI_DEBUG_LOG_MODULE_ALL)
- WiFi Debug Log Module WiFi (ESP32_WIFI_DEBUG_LOG_MODULE_WIFI)
- WiFi Debug Log Module Coex (ESP32_WIFI_DEBUG_LOG_MODULE_COEX)
- WiFi Debug Log Module Mesh (ESP32_WIFI_DEBUG_LOG_MODULE_MESH)

CONFIG_ESP32_WIFI_DEBUG_LOG_SUBMODULE

WiFi debug log submodule

Found in: Component config > Wi-Fi > CONFIG_ESP32_WIFI_DEBUG_LOG_ENABLE

Enable this option to set the WiFi debug log submodule. Currently the log submodule contains the following parts: INIT,IOCTL,CONN,SCAN. The INIT submodule indicates the initialization process.The IOCTL submodule indicates the API calling process. The CONN submodule indicates the connecting process.The SCAN submodule indicates the scaning process.

CONFIG_ESP32_WIFI_DEBUG_LOG_SUBMODULE_ALL

WiFi Debug Log Submodule All

Found in: Component config > Wi-Fi > CONFIG_ESP32_WIFI_DEBUG_LOG_ENABLE > CONFIG_ESP32_WIFI_DEBUG_LOG_SUBMODULE

When this option is enabled, all debug submodules are selected.

CONFIG_ESP32_WIFI_DEBUG_LOG_SUBMODULE_INIT

WiFi Debug Log Submodule Init

Found in: Component config > Wi-Fi > CONFIG_ESP32_WIFI_DEBUG_LOG_ENABLE > CONFIG_ESP32_WIFI_DEBUG_LOG_SUBMODULE

CONFIG_ESP32_WIFI_DEBUG_LOG_SUBMODULE_IOCTL

WiFi Debug Log Submodule Ioctl

Found in: Component config > Wi-Fi > CONFIG_ESP32_WIFI_DEBUG_LOG_ENABLE > CONFIG_ESP32_WIFI_DEBUG_LOG_SUBMODULE

CONFIG_ESP32_WIFI_DEBUG_LOG_SUBMODULE_CONN

WiFi Debug Log Submodule Conn

Found in: Component config > Wi-Fi > CONFIG_ESP32_WIFI_DEBUG_LOG_ENABLE > CONFIG_ESP32_WIFI_DEBUG_LOG_SUBMODULE

CONFIG_ESP32_WIFI_DEBUG_LOG_SUBMODULE_SCAN

WiFi Debug Log Submodule Scan

Found in: Component config > Wi-Fi > CONFIG_ESP32_WIFI_DEBUG_LOG_ENABLE > CONFIG_ESP32_WIFI_DEBUG_LOG_SUBMODULE

CONFIG_ESP32_WIFI_IRAM_OPT

WiFi IRAM speed optimization

Found in: Component config > Wi-Fi

Select this option to place frequently called Wi-Fi library functions in IRAM. When this option is disabled, more than 10Kbytes of IRAM memory will be saved but Wi-Fi throughput will be reduced.

CONFIG_ESP32_WIFI_RX_IRAM_OPT

WiFi RX IRAM speed optimization

Found in: Component config > Wi-Fi

Select this option to place frequently called Wi-Fi library RX functions in IRAM. When this option is disabled, more than 17Kbytes of IRAM memory will be saved but Wi-Fi performance will be reduced.

CONFIG_ESP32_WIFI_ENABLE_WPA3_SAE

Enable WPA3-Personal

Found in: Component config > Wi-Fi

Select this option to allow the device to establish a WPA3-Personal connection with eligible AP's. PMF (Protected Management Frames) is a prerequisite feature for a WPA3 connection, it needs to be explicitly configured before attempting connection. Please refer to the Wi-Fi Driver API Guide for details.

PHY

Contains:

- *CONFIG_ESP32_PHY_CALIBRATION_AND_DATA_STORAGE*
- *CONFIG_ESP32_PHY_INIT_DATA_IN_PARTITION*
- *CONFIG_ESP32_PHY_MAX_WIFI_TX_POWER*

CONFIG_ESP32_PHY_CALIBRATION_AND_DATA_STORAGE

Store phy calibration data in NVS

Found in: Component config > PHY

If this option is enabled, NVS will be initialized and calibration data will be loaded from there. PHY calibration will be skipped on deep sleep wakeup. If calibration data is not found, full calibration will be performed and stored in NVS. Normally, only partial calibration will be performed. If this option is disabled, full calibration will be performed.

If it' s easy that your board calibrate bad data, choose 'n'. Two cases for example, you should choose 'n': 1.If your board is easy to be booted up with antenna disconnected. 2.Because of your board design, each time when you do calibration, the result are too unstable. If unsure, choose 'y'.

CONFIG_ESP32_PHY_INIT_DATA_IN_PARTITION

Use a partition to store PHY init data

Found in: Component config > PHY

If enabled, PHY init data will be loaded from a partition. When using a custom partition table, make sure that PHY data partition is included (type: 'data', subtype: 'phy'). With default partition tables, this is done automatically. If PHY init data is stored in a partition, it has to be flashed there, otherwise runtime error will occur.

If this option is not enabled, PHY init data will be embedded into the application binary.

If unsure, choose 'n'.

CONFIG_ESP32_PHY_MAX_WIFI_TX_POWER

Max WiFi TX power (dBm)

Found in: Component config > PHY

Set maximum transmit power for WiFi radio. Actual transmit power for high data rates may be lower than this setting.

Core dump

Contains:

- *CONFIG_ESP32_COREDUMP_TO_FLASH_OR_UART*
- *CONFIG_ESP32_COREDUMP_DATA_FORMAT*
- *CONFIG_ESP32_COREDUMP_CHECKSUM*
- *CONFIG_ESP32_CORE_DUMP_MAX_TASKS_NUM*
- *CONFIG_ESP32_CORE_DUMP_UART_DELAY*
- *CONFIG_ESP32_CORE_DUMP_STACK_SIZE*

CONFIG_ESP32_COREDUMP_TO_FLASH_OR_UART

Data destination

Found in: Component config > Core dump

Select place to store core dump: flash, uart or none (to disable core dumps generation).

Core dumps to Flash are not available if PSRAM is used for task stacks.

If core dump is configured to be stored in flash and custom partition table is used add corresponding entry to your CSV. For examples, please see predefined partition table CSV descriptions in the components/partition_table directory.

Available options:

- Flash (ESP32_ENABLE_COREDUMP_TO_FLASH)
- UART (ESP32_ENABLE_COREDUMP_TO_UART)
- None (ESP32_ENABLE_COREDUMP_TO_NONE)

CONFIG_ESP32_COREDUMP_DATA_FORMAT

Core dump data format

Found in: Component config > Core dump

Select the data format for core dump.

Available options:

- Binary format (ESP32_COREDUMP_DATA_FORMAT_BIN)
- ELF format (ESP32_COREDUMP_DATA_FORMAT_ELF)

CONFIG_ESP32_COREDUMP_CHECKSUM

Core dump data integrity check

Found in: Component config > Core dump

Select the integrity check for the core dump.

Available options:

- Use CRC32 for integrity verification (ESP32_COREDUMP_CHECKSUM_CRC32)
- Use SHA256 for integrity verification (ESP32_COREDUMP_CHECKSUM_SHA256)

CONFIG_ESP32_CORE_DUMP_MAX_TASKS_NUM

Maximum number of tasks

Found in: Component config > Core dump

Maximum number of tasks snapshots in core dump.

CONFIG_ESP32_CORE_DUMP_UART_DELAY

Delay before print to UART

Found in: Component config > Core dump

Config delay (in ms) before printing core dump to UART. Delay can be interrupted by pressing Enter key.

CONFIG_ESP32_CORE_DUMP_STACK_SIZE

Reserved stack size

Found in: Component config > Core dump

Size of the memory to be reserved for core dump stack. If 0 core dump process will run on the stack of crashed task/ISR, otherwise special stack will be allocated. To ensure that core dump itself will not overflow task/ISR stack set this to the value above 800. NOTE: It eats DRAM.

FAT Filesystem support

Contains:

- *CONFIG_FATFS_CHOOSE_CODEPAGE*
- *CONFIG_FATFS_LONG_FILENAMES*

- *CONFIG_FATFS_MAX_LFN*
- *CONFIG_FATFS_API_ENCODING*
- *CONFIG_FATFS_FS_LOCK*
- *CONFIG_FATFS_TIMEOUT_MS*
- *CONFIG_FATFS_PER_FILE_CACHE*
- *CONFIG_FATFS_ALLOC_PREFER_EXTRAM*

CONFIG_FATFS_CHOOSE_CODEPAGE

OEM Code Page

Found in: Component config > FAT Filesystem support

OEM code page used for file name encodings.

If "Dynamic" is selected, code page can be chosen at runtime using f_setcp function. Note that choosing this option will increase application size by ~480kB.

Available options:

- Dynamic (all code pages supported) (FATFS_CODEPAGE_DYNAMIC)
- US (CP437) (FATFS_CODEPAGE_437)
- Arabic (CP720) (FATFS_CODEPAGE_720)
- Greek (CP737) (FATFS_CODEPAGE_737)
- KBL (CP771) (FATFS_CODEPAGE_771)
- Baltic (CP775) (FATFS_CODEPAGE_775)
- Latin 1 (CP850) (FATFS_CODEPAGE_850)
- Latin 2 (CP852) (FATFS_CODEPAGE_852)
- Cyrillic (CP855) (FATFS_CODEPAGE_855)
- Turkish (CP857) (FATFS_CODEPAGE_857)
- Portugese (CP860) (FATFS_CODEPAGE_860)
- Icelandic (CP861) (FATFS_CODEPAGE_861)
- Hebrew (CP862) (FATFS_CODEPAGE_862)
- Canadian French (CP863) (FATFS_CODEPAGE_863)
- Arabic (CP864) (FATFS_CODEPAGE_864)
- Nordic (CP865) (FATFS_CODEPAGE_865)

- Russian (CP866) (FATFS_CODEPAGE_866)
- Greek 2 (CP869) (FATFS_CODEPAGE_869)
- Japanese (DBCS) (CP932) (FATFS_CODEPAGE_932)
- Simplified Chinese (DBCS) (CP936) (FATFS_CODEPAGE_936)
- Korean (DBCS) (CP949) (FATFS_CODEPAGE_949)
- Traditional Chinese (DBCS) (CP950) (FATFS_CODEPAGE_950)

CONFIG_FATFS_LONG_FILENAMES

Long filename support

Found in: Component config > FAT Filesystem support

Support long filenames in FAT. Long filename data increases memory usage. FATFS can be configured to store the buffer for long filename data in stack or heap.

Available options:

- No long filenames (FATFS_LFN_NONE)
- Long filename buffer in heap (FATFS_LFN_HEAP)
- Long filename buffer on stack (FATFS_LFN_STACK)

CONFIG_FATFS_MAX_LFN

Max long filename length

Found in: Component config > FAT Filesystem support

Maximum long filename length. Can be reduced to save RAM.

CONFIG_FATFS_API_ENCODING

API character encoding

Found in: Component config > FAT Filesystem support

Choose encoding for character and string arguments/returns when using FATFS APIs. The encoding of arguments will usually depend on text editor settings.

Available options:

- API uses ANSI/OEM encoding (FATFS_API_ENCODING_ANSI_OEM)
- API uses UTF-16 encoding (FATFS_API_ENCODING_UTF_16)

- API uses UTF-8 encoding (FATFS_API_ENCODING_UTF_8)

CONFIG_FATFS_FS_LOCK

Number of simultaneously open files protected by lock function

Found in: Component config > FAT Filesystem support

This option sets the FATFS configuration value _FS_LOCK. The option _FS_LOCK switches file lock function to control duplicated file open and illegal operation to open objects.

* 0: Disable file lock function. To avoid volume corruption, application should avoid illegal open, remove and rename to the open objects.

* >0: Enable file lock function. The value defines how many files/sub-directories can be opened simultaneously under file lock control.

Note that the file lock control is independent of re-entrancy.

CONFIG_FATFS_TIMEOUT_MS

Timeout for acquiring a file lock, ms

Found in: Component config > FAT Filesystem support

This option sets FATFS configuration value _FS_TIMEOUT, scaled to milliseconds. Sets the number of milliseconds FATFS will wait to acquire a mutex when operating on an open file. For example, if one task is performing a lenghty operation, another task will wait for the first task to release the lock, and time out after amount of time set by this option.

CONFIG_FATFS_PER_FILE_CACHE

Use separate cache for each file

Found in: Component config > FAT Filesystem support

This option affects FATFS configuration value _FS_TINY.

If this option is set, _FS_TINY is 0, and each open file has its own cache, size of the cache is equal to the _MAX_SS variable (512 or 4096 bytes). This option uses more RAM if more than 1 file is open, but needs less reads and writes to the storage for some operations.

If this option is not set, _FS_TINY is 1, and single cache is used for all open files, size is also equal to _MAX_SS variable. This reduces the amount of heap used when multiple files are open, but increases the number of read and write operations which FATFS needs to make.

CONFIG_FATFS_ALLOC_PREFER_EXTRAM

Perfer external RAM when allocating FATFS buffers

Found in: Component config > FAT Filesystem support

When the option is enabled, internal buffers used by FATFS will be allocated from external RAM. If the allocation from external RAM fails, the buffer will be allocated from the internal RAM. Disable this option if optimizing for performance. Enable this option if optimizing for internal memory size.

Modbus configuration

Contains:

- *CONFIG_FMB_COMM_MODE_RTU_EN*
- *CONFIG_FMB_COMM_MODE_ASCII_EN*
- *CONFIG_FMB_MASTER_TIMEOUT_MS_RESPOND*
- *CONFIG_FMB_MASTER_DELAY_MS_CONVERT*
- *CONFIG_FMB_QUEUE_LENGTH*
- *CONFIG_FMB_SERIAL_TASK_STACK_SIZE*
- *CONFIG_FMB_SERIAL_BUF_SIZE*
- *CONFIG_FMB_SERIAL_ASCII_BITS_PER_SYMB*
- *CONFIG_FMB_SERIAL_ASCII_TIMEOUT_RESPOND_MS*
- *CONFIG_FMB_SERIAL_TASK_PRIO*
- *CONFIG_FMB_CONTROLLER_SLAVE_ID_SUPPORT*
- *CONFIG_FMB_CONTROLLER_NOTIFY_TIMEOUT*
- *CONFIG_FMB_CONTROLLER_NOTIFY_QUEUE_SIZE*
- *CONFIG_FMB_CONTROLLER_STACK_SIZE*
- *CONFIG_FMB_EVENT_QUEUE_TIMEOUT*
- *CONFIG_FMB_TIMER_PORT_ENABLED*
- *CONFIG_FMB_TIMER_GROUP*
- *CONFIG_FMB_TIMER_INDEX*
- *CONFIG_FMB_TIMER_ISR_IN_IRAM*

CONFIG_FMB_COMM_MODE_RTU_EN

Enable Modbus stack support for RTU mode

Found in: Component config > Modbus configuration

Enable RTU Modbus communication mode option for Modbus serial stack.

CONFIG_FMB_COMM_MODE_ASCII_EN

Enable Modbus stack support for ASCII mode

Found in: Component config > Modbus configuration

Enable ASCII Modbus communication mode option for Modbus serial stack.

CONFIG_FMB_MASTER_TIMEOUT_MS_RESPOND

Slave respond timeout (Milliseconds)

Found in: Component config > Modbus configuration

If master sends a frame which is not broadcast, it has to wait sometime for slave response. if slave is not respond in this time, the master will process timeout error.

CONFIG_FMB_MASTER_DELAY_MS_CONVERT

Slave conversion delay (Milliseconds)

Found in: Component config > Modbus configuration

If master sends a broadcast frame, it has to wait conversion time to delay, then master can send next frame.

CONFIG_FMB_QUEUE_LENGTH

Modbus serial task queue length

Found in: Component config > Modbus configuration

Modbus serial driver queue length. It is used by event queue task. See the serial driver API for more information.

CONFIG_FMB_SERIAL_TASK_STACK_SIZE

Modbus serial task stack size

Found in: Component config > Modbus configuration

Modbus serial task stack size for event queue task. It may be adjusted when debugging is enabled (for example).

CONFIG_FMB_SERIAL_BUF_SIZE

Modbus serial task RX/TX buffer size

Found in: Component config > Modbus configuration

Modbus serial task RX and TX buffer size for UART driver initialization. This buffer is used for modbus frame transfer. The Modbus protocol maximum frame size is 256 bytes. Bigger size can be used for non standard implementations.

CONFIG_FMB_SERIAL_ASCII_BITS_PER_SYMB

Number of data bits per ASCII character

Found in: Component config > Modbus configuration

This option defines the number of data bits per ASCII character.

CONFIG_FMB_SERIAL_ASCII_TIMEOUT_RESPOND_MS

Response timeout for ASCII communication mode (ms)

Found in: Component config > Modbus configuration

This option defines response timeout of slave in milliseconds for ASCII communication mode. Thus the timeout will expire and allow the master' s program to handle the error.

CONFIG_FMB_SERIAL_TASK_PRIO

Modbus serial task priority

Found in: Component config > Modbus configuration

Modbus UART driver event task priority. The priority of Modbus controller task is equal to (CONFIG_FMB_SERIAL_TASK_PRIO - 1).

CONFIG_FMB_CONTROLLER_SLAVE_ID_SUPPORT

Modbus controller slave ID support

Found in: Component config > Modbus configuration

Modbus slave ID support enable. When enabled the Modbus <Report Slave ID> command is supported by stack.

CONFIG_FMB_CONTROLLER_SLAVE_ID

Modbus controller slave ID

Found in: Component config > Modbus configuration > CONFIG_FMB_CONTROLLER_SLAVE_ID_SUPPORT

Modbus slave ID value to identify modbus device in the network using <Report Slave ID> command. Most significant byte of ID is used as short device ID and other three bytes used as long ID.

CONFIG_FMB_CONTROLLER_NOTIFY_TIMEOUT

Modbus controller notification timeout (ms)

Found in: Component config > Modbus configuration

Modbus controller notification timeout in milliseconds. This timeout is used to send notification about accessed parameters.

CONFIG_FMB_CONTROLLER_NOTIFY_QUEUE_SIZE

Modbus controller notification queue size

Found in: Component config > Modbus configuration

Modbus controller notification queue size. The notification queue is used to get information about accessed parameters.

CONFIG_FMB_CONTROLLER_STACK_SIZE

Modbus controller stack size

Found in: Component config > Modbus configuration

Modbus controller task stack size. The Stack size may be adjusted when debug mode is used which requires more stack size (for example).

CONFIG_FMB_EVENT_QUEUE_TIMEOUT

Modbus stack event queue timeout (ms)

Found in: Component config > Modbus configuration

Modbus stack event queue timeout in milliseconds. This may help to optimize Modbus stack event processing time.

CONFIG_FMB_TIMER_PORT_ENABLED

Modbus slave stack use timer for 3.5T symbol time measurement

Found in: Component config > Modbus configuration

If this option is set the Modbus stack uses timer for T3.5 time measurement. Else the internal UART TOUT timeout is used for 3.5T symbol time measurement.

CONFIG_FMB_TIMER_GROUP

Modbus Timer group number

Found in: Component config > Modbus configuration

Modbus Timer group number that is used for timeout measurement.

CONFIG_FMB_TIMER_INDEX

Modbus Timer index in the group

Found in: Component config > Modbus configuration

Modbus Timer Index in the group that is used for timeout measurement.

CONFIG_FMB_TIMER_ISR_IN_IRAM

Place timer interrupt handler into IRAM

Found in: Component config > Modbus configuration

This option places Modbus timer IRQ handler into IRAM. This allows to avoid delays related to processing of non-IRAM-safe interrupts during a flash write operation (NVS updating a value, or some other flash API which has to perform an read/write operation and disable CPU cache). This option has dependency with the UART_ISR_IN_IRAM option which places UART interrupt handler into IRAM to prevent delays related to processing of UART events.

FreeRTOS

Contains:

- *CONFIG_FREERTOS_UNICORE*
- *CONFIG_FREERTOS_CORETIMER*
- *CONFIG_FREERTOS_HZ*
- *CONFIG_FREERTOS_ASSERT_ON_UNTESTED_FUNCTION*
- *CONFIG_FREERTOS_CHECK_STACKOVERFLOW*
- *CONFIG_FREERTOS_WATCHPOINT_END_OF_STACK*
- *CONFIG_FREERTOS_INTERRUPT_BACKTRACE*
- *CONFIG_FREERTOS_THREAD_LOCAL_STORAGE_POINTERS*
- *CONFIG_FREERTOS_ASSERT*
- *CONFIG_FREERTOS_IDLE_TASK_STACKSIZE*
- *CONFIG_FREERTOS_ISR_STACKSIZE*
- *CONFIG_FREERTOS_LEGACY_HOOKS*
- *CONFIG_FREERTOS_MAX_TASK_NAME_LEN*
- *CONFIG_FREERTOS_SUPPORT_STATIC_ALLOCATION*
- *CONFIG_FREERTOS_TIMER_TASK_PRIORITY*
- *CONFIG_FREERTOS_TIMER_TASK_STACK_DEPTH*
- *CONFIG_FREERTOS_TIMER_QUEUE_LENGTH*
- *CONFIG_FREERTOS_QUEUE_REGISTRY_SIZE*
- *CONFIG_FREERTOS_USE_TRACE_FACILITY*
- *CONFIG_FREERTOS_GENERATE_RUN_TIME_STATS*
- *CONFIG_FREERTOS_USE_TICKLESS_IDLE*
- *CONFIG_FREERTOS_DEBUG_INTERNALS*
- *CONFIG_FREERTOS_TASK_FUNCTION_WRAPPER*
- *CONFIG_FREERTOS_CHECK_MUTEX_GIVEN_BY_OWNER*
- *CONFIG_FREERTOS_CHECK_PORT_CRITICAL_COMPLIANCE*

CONFIG_FREERTOS_UNICORE

Run FreeRTOS only on first core

Found in: Component config > FreeRTOS

This version of FreeRTOS normally takes control of all cores of the CPU. Select this if you only want to start it on the first core. This is needed when e.g. another process needs complete control over the second core.

This invisible config value sets the value of tskNO_AFFINITY in task.h. # Intended to be used as a constant from other Kconfig files. # Value is (32-bit) INT_MAX.

CONFIG_FREERTOS_CORETIMER

Xtensa timer to use as the FreeRTOS tick source

Found in: Component config > FreeRTOS

FreeRTOS needs a timer with an associated interrupt to use as the main tick source to increase counters, run timers and do pre-emptive multitasking with. There are multiple timers available to do this, with different interrupt priorities. Check

Available options:

- Timer 0 (int 6, level 1) (FREERTOS_CORETIMER_0)

  Select this to use timer 0

- Timer 1 (int 15, level 3) (FREERTOS_CORETIMER_1)

  Select this to use timer 1

CONFIG_FREERTOS_HZ

Tick rate (Hz)

Found in: Component config > FreeRTOS

Select the tick rate at which FreeRTOS does pre-emptive context switching.

CONFIG_FREERTOS_ASSERT_ON_UNTESTED_FUNCTION

Halt when an SMP-untested function is called

Found in: Component config > FreeRTOS

Some functions in FreeRTOS have not been thoroughly tested yet when moving to the SMP implementation of FreeRTOS. When this option is enabled, these fuctions will throw an assert().

CONFIG_FREERTOS_CHECK_STACKOVERFLOW

Check for stack overflow

Found in: Component config > FreeRTOS

FreeRTOS can check for stack overflows in threads and trigger an user function called vApplicationStackOverflowHook when this happens.

Available options:

- No checking (FREERTOS_CHECK_STACKOVERFLOW_NONE)

  Do not check for stack overflows (configCHECK_FOR_STACK_OVERFLOW=0)

- Check by stack pointer value (FREERTOS_CHECK_STACKOVERFLOW_PTRVAL)

  Check for stack overflows on each context switch by checking if the stack pointer is in a valid range. Quick but does not detect stack overflows that happened between context switches (configCHECK_FOR_STACK_OVERFLOW=1)

- Check using canary bytes (FREERTOS_CHECK_STACKOVERFLOW_CANARY)

  Places some magic bytes at the end of the stack area and on each context switch, check if these bytes are still intact. More thorough than just checking the pointer, but also slightly slower. (configCHECK_FOR_STACK_OVERFLOW=2)

CONFIG_FREERTOS_WATCHPOINT_END_OF_STACK

Set a debug watchpoint as a stack overflow check

Found in: Component config > FreeRTOS

FreeRTOS can check if a stack has overflown its bounds by checking either the value of the stack pointer or by checking the integrity of canary bytes. (See FREERTOS_CHECK_STACKOVERFLOW for more information.) These checks only happen on a context switch, and the situation that caused the stack overflow may already be long gone by then. This option will use the debug memory watchpoint 1 (the second one) to allow breaking into the debugger (or panic' ing) as soon as any of the last 32 bytes on the stack of a task are overwritten. The side effect is that using gdb, you effectively only have one watchpoint; the 2nd one is overwritten as soon as a task switch happens.

This check only triggers if the stack overflow writes within 4 bytes of the end of the stack, rather than overshooting further, so it is worth combining this approach with one of the other stack overflow check methods.

When this watchpoint is hit, gdb will stop with a SIGTRAP message. When no JTAG OCD is attached, esp-idf will panic on an unhandled debug exception.

CONFIG_FREERTOS_INTERRUPT_BACKTRACE

Enable backtrace from interrupt to task context

Found in: Component config > FreeRTOS

If this option is enabled, interrupt stack frame will be modified to point to the code of the interrupted task as its return address. This helps the debugger (or the panic handler) show a backtrace from the interrupt to the task which was interrupted. This also works for nested interrupts: higer level interrupt stack can be traced back to the lower level interrupt. This option adds 4 instructions to the interrupt dispatching code.

CONFIG_FREERTOS_THREAD_LOCAL_STORAGE_POINTERS

Number of thread local storage pointers

Found in: Component config > FreeRTOS

FreeRTOS has the ability to store per-thread pointers in the task control block. This controls the number of pointers available.

This value must be at least 1. Index 0 is reserved for use by the pthreads API thread-local-storage. Other indexes can be used for any desired purpose.

CONFIG_FREERTOS_ASSERT

FreeRTOS assertions

Found in: Component config > FreeRTOS

Failed FreeRTOS configASSERT() assertions can be configured to behave in different ways.

Available options:

- abort() on failed assertions (FREERTOS_ASSERT_FAIL_ABORT)

  If a FreeRTOS configASSERT() fails, FreeRTOS will abort() and halt execution. The panic handler can be configured to handle the outcome of an abort() in different ways.

- Print and continue failed assertions (FREERTOS_ASSERT_FAIL_PRINT_CONTINUE)

  If a FreeRTOS assertion fails, print it out and continue.

- Disable FreeRTOS assertions (FREERTOS_ASSERT_DISABLE)

  FreeRTOS configASSERT() will not be compiled into the binary.

CONFIG_FREERTOS_IDLE_TASK_STACKSIZE

Idle Task stack size

Found in: Component config > FreeRTOS

The idle task has its own stack, sized in bytes. The default size is enough for most uses. Size can be reduced to 768 bytes if no (or simple) FreeRTOS idle hooks are used and pthread local storage or FreeRTOS local storage cleanup callbacks are not used.

The stack size may need to be increased above the default if the app installs idle or thread local storage cleanup hooks that use a lot of stack memory.

CONFIG_FREERTOS_ISR_STACKSIZE

ISR stack size

Found in: Component config > FreeRTOS

The interrupt handlers have their own stack. The size of the stack can be defined here. Each processor has its own stack, so the total size occupied will be twice this.

CONFIG_FREERTOS_LEGACY_HOOKS

Use FreeRTOS legacy hooks

Found in: Component config > FreeRTOS

FreeRTOS offers a number of hooks/callback functions that are called when a timer tick happens, the idle thread runs etc. esp-idf replaces these by runtime registerable hooks using the esp_register_freertos_xxx_hook system, but for legacy reasons the old hooks can also still be enabled. Please enable this only if you have code that for some reason can't be migrated to the esp_register_freertos_xxx_hook system.

CONFIG_FREERTOS_MAX_TASK_NAME_LEN

Maximum task name length

Found in: Component config > FreeRTOS

Changes the maximum task name length. Each task allocated will include this many bytes for a task name. Using a shorter value saves a small amount of RAM, a longer value allows more complex names.

For most uses, the default of 16 is OK.

CONFIG_FREERTOS_SUPPORT_STATIC_ALLOCATION

Enable FreeRTOS static allocation API

Found in: Component config > FreeRTOS

FreeRTOS gives the application writer the ability to instead provide the memory themselves, allowing the following objects to optionally be created without any memory being allocated dynamically:

- Tasks
- Software Timers (Daemon task is still dynamic. See documentation)
- Queues
- Event Groups
- Binary Semaphores
- Counting Semaphores
- Recursive Semaphores
- Mutexes

Whether it is preferable to use static or dynamic memory allocation is dependent on the application, and the preference of the application writer. Both methods have pros and cons, and both methods can be used within the same RTOS application.

Creating RTOS objects using statically allocated RAM has the benefit of providing the application writer with more control: RTOS objects can be placed at specific memory locations. The maximum RAM footprint can be determined at link time, rather than run time. The application writer does not need to concern themselves with graceful handling of memory allocation failures. It allows the RTOS to be used in applications that simply don' t allow any dynamic memory allocation (although FreeRTOS includes allocation schemes that can overcome most objections).

CONFIG_FREERTOS_ENABLE_STATIC_TASK_CLEAN_UP

Enable static task clean up hook

Found in: Component config > FreeRTOS > CONFIG_FREERTOS_SUPPORT_STATIC_ALLOCATION

Enable this option to make FreeRTOS call the static task clean up hook when a task is deleted.

Bear in mind that if this option is enabled you will need to implement the following function:

```
void vPortCleanUpTCB ( void \*pxTCB ) {
    // place clean up code here
}
```

CONFIG_FREERTOS_TIMER_TASK_PRIORITY

FreeRTOS timer task priority

Found in: Component config > FreeRTOS

The timer service task (primarily) makes use of existing FreeRTOS features, allowing timer functionality to be added to an application with minimal impact on the size of the application's executable binary.

Use this constant to define the priority that the timer task will run at.

CONFIG_FREERTOS_TIMER_TASK_STACK_DEPTH

FreeRTOS timer task stack size

Found in: Component config > FreeRTOS

The timer service task (primarily) makes use of existing FreeRTOS features, allowing timer functionality to be added to an application with minimal impact on the size of the application's executable binary.

Use this constant to define the size (in bytes) of the stack allocated for the timer task.

CONFIG_FREERTOS_TIMER_QUEUE_LENGTH

FreeRTOS timer queue length

Found in: Component config > FreeRTOS

FreeRTOS provides a set of timer related API functions. Many of these functions use a standard FreeRTOS queue to send commands to the timer service task. The queue used for this purpose is called the 'timer command queue'. The 'timer command queue' is private to the FreeRTOS timer implementation, and cannot be accessed directly.

For most uses the default value of 10 is OK.

CONFIG_FREERTOS_QUEUE_REGISTRY_SIZE

FreeRTOS queue registry size

Found in: Component config > FreeRTOS

FreeRTOS uses the queue registry as a means for kernel aware debuggers to locate queues, semaphores, and mutexes. The registry allows for a textual name to be associated with a queue for easy identification within a debugging GUI. A value of 0 will disable queue registry functionality,

and a value larger than 0 will specify the number of queues/semaphores/mutexes that the registry can hold.

CONFIG_FREERTOS_USE_TRACE_FACILITY

Enable FreeRTOS trace facility

Found in: Component config > FreeRTOS

If enabled, configUSE_TRACE_FACILITY will be defined as 1 in FreeRTOS. This will allow the usage of trace facility functions such as uxTaskGetSystemState().

CONFIG_FREERTOS_USE_STATS_FORMATTING_FUNCTIONS

Enable FreeRTOS stats formatting functions

Found in: Component config > FreeRTOS > CONFIG_FREERTOS_USE_TRACE_FACILITY

If enabled, configUSE_STATS_FORMATTING_FUNCTIONS will be defined as 1 in FreeRTOS. This will allow the usage of stats formatting functions such as vTaskList().

CONFIG_FREERTOS_VTASKLIST_INCLUDE_COREID

Enable display of xCoreID in vTaskList

Found in: Component config > FreeRTOS > CONFIG_FREERTOS_USE_TRACE_FACILITY > CONFIG_FREERTOS_USE_STATS_FORMATTING_FUNCTIONS

If enabled, this will include an extra column when vTaskList is called to display the CoreID the task is pinned to (0,1) or -1 if not pinned.

CONFIG_FREERTOS_GENERATE_RUN_TIME_STATS

Enable FreeRTOS to collect run time stats

Found in: Component config > FreeRTOS

If enabled, configGENERATE_RUN_TIME_STATS will be defined as 1 in FreeRTOS. This will allow FreeRTOS to collect information regarding the usage of processor time amongst FreeRTOS tasks. Run time stats are generated using either the ESP Timer or the CPU Clock as the clock source (Note that run time stats are only valid until the clock source overflows). The function vTaskGetRunTimeStats() will also be available if FREERTOS_USE_STATS_FORMATTING_FUNCTIONS and FREERTOS_USE_TRACE_FACILITY are enabled. vTaskGetRunTimeStats() will display the run

time of each task as a % of the total run time of all CPUs (task run time / no of CPUs) / (total run time / 100)

CONFIG_FREERTOS_RUN_TIME_STATS_CLK

Choose the clock source for run time stats

Found in: Component config > FreeRTOS > CONFIG_FREERTOS_GENERATE_RUN_TIME_STATS

Choose the clock source for FreeRTOS run time stats. Options are CPU0' s CPU Clock or the ESP Timer. Both clock sources are 32 bits. The CPU Clock can run at a higher frequency hence provide a finer resolution but will overflow much quicker. Note that run time stats are only valid until the clock source overflows.

Available options:

- Use ESP TIMER for run time stats (FREERTOS_RUN_TIME_STATS_USING_ESP_TIMER)

  ESP Timer will be used as the clock source for FreeRTOS run time stats. The ESP Timer runs at a frequency of 1MHz regardless of Dynamic Frequency Scaling. Therefore the ESP Timer will overflow in approximately 4290 seconds.

- Use CPU Clock for run time stats (FREERTOS_RUN_TIME_STATS_USING_CPU_CLK)

  CPU Clock will be used as the clock source for the generation of run time stats. The CPU Clock has a frequency dependent on ESP32_DEFAULT_CPU_FREQ_MHZ and Dynamic Frequency Scaling (DFS). Therefore the CPU Clock frequency can fluctuate between 80 to 240MHz. Run time stats generated using the CPU Clock represents the number of CPU cycles each task is allocated and DOES NOT reflect the amount of time each task runs for (as CPU clock frequency can change). If the CPU clock consistently runs at the maximum frequency of 240MHz, it will overflow in approximately 17 seconds.

CONFIG_FREERTOS_USE_TICKLESS_IDLE

Tickless idle support

Found in: Component config > FreeRTOS

If power management support is enabled, FreeRTOS will be able to put the system into light sleep mode when no tasks need to run for a number of ticks. This number can be set using FREERTOS_IDLE_TIME_BEFORE_SLEEP option. This feature is also known as "automatic light sleep" .

Note that timers created using esp_timer APIs may prevent the system from entering sleep mode, even when no tasks need to run.

If disabled, automatic light sleep support will be disabled.

CONFIG_FREERTOS_IDLE_TIME_BEFORE_SLEEP

Minimum number of ticks to enter sleep mode for

Found in: Component config > FreeRTOS > CONFIG_FREERTOS_USE_TICKLESS_IDLE

FreeRTOS will enter light sleep mode if no tasks need to run for this number of ticks.

CONFIG_FREERTOS_DEBUG_INTERNALS

Debug FreeRTOS internals

Found in: Component config > FreeRTOS

Enable this option to show the menu with internal FreeRTOS debugging features. This option does not change any code by itself, it just shows/hides some options.

Contains:

- *CONFIG_FREERTOS_PORTMUX_DEBUG*
- *CONFIG_FREERTOS_PORTMUX_DEBUG_RECURSIVE*

CONFIG_FREERTOS_PORTMUX_DEBUG

Debug portMUX portENTER_CRITICAL/portEXIT_CRITICAL

Found in: Component config > FreeRTOS > CONFIG_FREERTOS_DEBUG_INTERNALS

If enabled, debug information (including integrity checks) will be printed to UART for the port-specific MUX implementation.

CONFIG_FREERTOS_PORTMUX_DEBUG_RECURSIVE

Debug portMUX Recursion

Found in: Component config > FreeRTOS > CONFIG_FREERTOS_DEBUG_INTERNALS

If enabled, additional debug information will be printed for recursive portMUX usage.

CONFIG_FREERTOS_TASK_FUNCTION_WRAPPER

Enclose all task functions in a wrapper function

Found in: Component config > FreeRTOS

If enabled, all FreeRTOS task functions will be enclosed in a wrapper function. If a task function mistakenly returns (i.e. does not delete), the call flow will return to the wrapper function. The

wrapper function will then log an error and abort the application. This option is also required for GDB backtraces and C++ exceptions to work correctly inside top-level task functions.

CONFIG_FREERTOS_CHECK_MUTEX_GIVEN_BY_OWNER

Check that mutex semaphore is given by owner task

Found in: Component config > FreeRTOS

If enabled, assert that when a mutex semaphore is given, the task giving the semaphore is the task which is currently holding the mutex.

CONFIG_FREERTOS_CHECK_PORT_CRITICAL_COMPLIANCE

Tests compliance with Vanilla FreeRTOS port*_CRITICAL calls

Found in: Component config > FreeRTOS

If enabled, context of port*_CRITICAL calls (ISR or Non-ISR) would be checked to be in compliance with Vanilla FreeRTOS. e.g Calling port*_CRITICAL from ISR context would cause assert failure

Heap memory debugging

Contains:

- *CONFIG_HEAP_CORRUPTION_DETECTION*
- *CONFIG_HEAP_TRACING_DEST*
- *CONFIG_HEAP_TRACING_STACK_DEPTH*
- *CONFIG_HEAP_TASK_TRACKING*

CONFIG_HEAP_CORRUPTION_DETECTION

Heap corruption detection

Found in: Component config > Heap memory debugging

Enable heap poisoning features to detect heap corruption caused by out-of-bounds access to heap memory.

See the "Heap Memory Debugging" page of the IDF documentation for a description of each level of heap corruption detection.

Available options:

- Basic (no poisoning) (HEAP_POISONING_DISABLED)
- Light impact (HEAP_POISONING_LIGHT)
- Comprehensive (HEAP_POISONING_COMPREHENSIVE)

CONFIG_HEAP_TRACING_DEST

Heap tracing

Found in: Component config > Heap memory debugging

Enables the heap tracing API defined in esp_heap_trace.h.

This function causes a moderate increase in IRAM code side and a minor increase in heap function (malloc/free/realloc) CPU overhead, even when the tracing feature is not used. So it' s best to keep it disabled unless tracing is being used.

Available options:

- Disabled (HEAP_TRACING_OFF)
- Standalone (HEAP_TRACING_STANDALONE)
- Host-based (HEAP_TRACING_TOHOST)

CONFIG_HEAP_TRACING_STACK_DEPTH

Heap tracing stack depth

Found in: Component config > Heap memory debugging

Number of stack frames to save when tracing heap operation callers.

More stack frames uses more memory in the heap trace buffer (and slows down allocation), but can provide useful information.

CONFIG_HEAP_TASK_TRACKING

Enable heap task tracking

Found in: Component config > Heap memory debugging

Enables tracking the task responsible for each heap allocation.

This function depends on heap poisoning being enabled and adds four more bytes of overhead for each block allocated.

jsmn

Contains:

- *CONFIG_JSMN_PARENT_LINKS*
- *CONFIG_JSMN_STRICT*

CONFIG_JSMN_PARENT_LINKS

Enable parent links

Found in: Component config > jsmn

You can access to parent node of parsed json

CONFIG_JSMN_STRICT

Enable strict mode

Found in: Component config > jsmn

In strict mode primitives are: numbers and booleans

libsodium

Contains:

- *CONFIG_LIBSODIUM_USE_MBEDTLS_SHA*

CONFIG_LIBSODIUM_USE_MBEDTLS_SHA

Use mbedTLS SHA256 & SHA512 implementations

Found in: Component config > libsodium

If this option is enabled, libsodium will use thin wrappers around mbedTLS for SHA256 & SHA512 operations.

This saves some code size if mbedTLS is also used. However it is incompatible with hardware SHA acceleration (due to the way libsodium' s API manages SHA state).

Log output

Contains:

- *CONFIG_LOG_DEFAULT_LEVEL*

- *CONFIG_LOG_COLORS*
- *CONFIG_LOG_TIMESTAMP_SOURCE*

CONFIG_LOG_DEFAULT_LEVEL

Default log verbosity

Found in: Component config > Log output

Specify how much output to see in logs by default. You can set lower verbosity level at runtime using esp_log_level_set function.

Note that this setting limits which log statements are compiled into the program. So setting this to, say, "Warning" would mean that changing log level to "Debug" at runtime will not be possible.

Available options:

- No output (LOG_DEFAULT_LEVEL_NONE)
- Error (LOG_DEFAULT_LEVEL_ERROR)
- Warning (LOG_DEFAULT_LEVEL_WARN)
- Info (LOG_DEFAULT_LEVEL_INFO)
- Debug (LOG_DEFAULT_LEVEL_DEBUG)
- Verbose (LOG_DEFAULT_LEVEL_VERBOSE)

CONFIG_LOG_COLORS

Use ANSI terminal colors in log output

Found in: Component config > Log output

Enable ANSI terminal color codes in bootloader output.

In order to view these, your terminal program must support ANSI color codes.

CONFIG_LOG_TIMESTAMP_SOURCE

Log Timestamps

Found in: Component config > Log output

Choose what sort of timestamp is displayed in the log output:

- Milliseconds since boot is calulated from the RTOS tick count multiplied by the tick period. This time will reset after a software reboot. e.g. (90000)

- System time is taken from POSIX time functions which use the ESP32’ s RTC and FRC1 timers to maintain an accurate time. The system time is initialized to 0 on startup, it can be set with an SNTP sync, or with POSIX time functions. This time will not reset after a software reboot. e.g. (00:01:30.000)
- NOTE: Currently this will not get used in logging from binary blobs (i.e WiFi & Bluetooth libraries), these will always print milliseconds since boot.

Available options:

- Milliseconds Since Boot (LOG_TIMESTAMP_SOURCE_RTOS)
- System Time (LOG_TIMESTAMP_SOURCE_SYSTEM)

LWIP

Contains:

- *CONFIG_LWIP_LOCAL_HOSTNAME*
- *CONFIG_LWIP_DNS_SUPPORT_MDNS_QUERIES*
- *CONFIG_LWIP_L2_TO_L3_COPY*
- *CONFIG_LWIP_IRAM_OPTIMIZATION*
- *CONFIG_LWIP_TIMERS_ONDEMAND*
- *CONFIG_LWIP_MAX_SOCKETS*
- *CONFIG_LWIP_USE_ONLY_LWIP_SELECT*
- *CONFIG_LWIP_SO_LINGER*
- *CONFIG_LWIP_SO_REUSE*
- *CONFIG_LWIP_SO_RCVBUF*
- *CONFIG_LWIP_NETBUF_RECVINFO*
- *CONFIG_LWIP_IP_FRAG*
- *CONFIG_LWIP_IP_REASSEMBLY*
- *CONFIG_LWIP_STATS*
- *CONFIG_LWIP_ETHARP_TRUST_IP_MAC*
- *CONFIG_LWIP_ESP_GRATUITOUS_ARP*
- *CONFIG_LWIP_TCPIP_RECVMBOX_SIZE*
- *CONFIG_LWIP_DHCP_DOES_ARP_CHECK*
- *CONFIG_LWIP_DHCP_RESTORE_LAST_IP*

- *DHCP server*
- *CONFIG_LWIP_AUTOIP*
- *CONFIG_LWIP_IPV6_AUTOCONFIG*
- *CONFIG_LWIP_NETIF_LOOPBACK*
- *TCP*
- *UDP*
- *CONFIG_LWIP_TCPIP_TASK_STACK_SIZE*
- *CONFIG_LWIP_TCPIP_TASK_AFFINITY*
- *CONFIG_LWIP_PPP_SUPPORT*
- *ICMP*
- *LWIP RAW API*
- *SNTP*

CONFIG_LWIP_LOCAL_HOSTNAME

Local netif hostname

Found in: Component config > LWIP

The name this device will report to other devices on the network

CONFIG_LWIP_DNS_SUPPORT_MDNS_QUERIES

Enable mDNS queries in resolving host name

Found in: Component config > LWIP

If this feature is enabled, standard API such as gethostbyname support .local addresses by sending one shot multicast mDNS query

CONFIG_LWIP_L2_TO_L3_COPY

Enable copy between Layer2 and Layer3 packets

Found in: Component config > LWIP

If this feature is enabled, all traffic from layer2(WIFI Driver) will be copied to a new buffer before sending it to layer3(LWIP stack), freeing the layer2 buffer. Please be notified that the total layer2 receiving buffer is fixed and ESP32 currently supports 25 layer2 receiving buffer, when layer2 buffer runs out of memory, then the incoming packets will be dropped in hardware.

The layer3 buffer is allocated from the heap, so the total layer3 receiving buffer depends on the available heap size, when heap runs out of memory, no copy will be sent to layer3 and packet will be dropped in layer2. Please make sure you fully understand the impact of this feature before enabling it.

CONFIG_LWIP_IRAM_OPTIMIZATION

Enable LWIP IRAM optimization

Found in: Component config > LWIP

If this feature is enabled, some functions relating to RX/TX in LWIP will be put into IRAM, it can improve UDP/TCP throughput by >10% for single core mode, it doesn't help too much for dual core mode. On the other hand, it needs about 10KB IRAM for these optimizations.

If this feature is disabled, all lwip functions will be put into FLASH.

CONFIG_LWIP_TIMERS_ONDEMAND

Enable LWIP Timers on demand

Found in: Component config > LWIP

If this feature is enabled, IGMP and MLD6 timers will be activated only when joining groups or receiving QUERY packets.

This feature will reduce the power consumption for applications which do not use IGMP and MLD6.

CONFIG_LWIP_MAX_SOCKETS

Max number of open sockets

Found in: Component config > LWIP

Sockets take up a certain amount of memory, and allowing fewer sockets to be open at the same time conserves memory. Specify the maximum amount of sockets here. The valid value is from 1 to 16.

CONFIG_LWIP_USE_ONLY_LWIP_SELECT

Support LWIP socket select() only

Found in: Component config > LWIP

The virtual filesystem layer of select() redirects sockets to lwip_select() and non-socket file descriptors to their respective driver implementations. If this option is enabled then all calls of select() will be redirected to lwip_select(), therefore, select can be used for sockets only.

CONFIG_LWIP_SO_LINGER

Enable SO_LINGER processing

Found in: Component config > LWIP

Enabling this option allows SO_LINGER processing. l_onoff = 1,l_linger can set the timeout.

If l_linger=0, When a connection is closed, TCP will terminate the connection. This means that TCP will discard any data packets stored in the socket send buffer and send an RST to the peer.

If l_linger!=0,Then closesocket() calls to block the process until the remaining data packets has been sent or timed out.

CONFIG_LWIP_SO_REUSE

Enable SO_REUSEADDR option

Found in: Component config > LWIP

Enabling this option allows binding to a port which remains in TIME_WAIT.

CONFIG_LWIP_SO_REUSE_RXTOALL

SO_REUSEADDR copies broadcast/multicast to all matches

Found in: Component config > LWIP > CONFIG_LWIP_SO_REUSE

Enabling this option means that any incoming broadcast or multicast packet will be copied to all of the local sockets that it matches (may be more than one if SO_REUSEADDR is set on the socket.)

This increases memory overhead as the packets need to be copied, however they are only copied per matching socket. You can safely disable it if you don' t plan to receive broadcast or multicast traffic on more than one socket at a time.

CONFIG_LWIP_SO_RCVBUF

Enable SO_RCVBUF option

Found in: Component config > LWIP

Enabling this option allows checking for available data on a netconn.

CONFIG_LWIP_NETBUF_RECVINFO

Enable IP_PKTINFO option

Found in: Component config > LWIP

Enabling this option allows checking for the destination address of a received IPv4 Packet.

CONFIG_LWIP_IP_FRAG

Enable fragment outgoing IP packets

Found in: Component config > LWIP

Enabling this option allows fragmenting outgoing IP packets if their size exceeds MTU.

CONFIG_LWIP_IP_REASSEMBLY

Enable reassembly incoming fragmented IP packets

Found in: Component config > LWIP

Enabling this option allows reassemblying incoming fragmented IP packets.

CONFIG_LWIP_STATS

Enable LWIP statistics

Found in: Component config > LWIP

Enabling this option allows LWIP statistics

CONFIG_LWIP_ETHARP_TRUST_IP_MAC

Enable LWIP ARP trust

Found in: Component config > LWIP

Enabling this option allows ARP table to be updated.

If this option is enabled, the incoming IP packets cause the ARP table to be updated with the source MAC and IP addresses supplied in the packet. You may want to disable this if you do not trust LAN peers to have the correct addresses, or as a limited approach to attempt to handle spoofing. If disabled, lwIP will need to make a new ARP request if the peer is not already in the ARP table, adding a little latency. The peer *is* in the ARP table if it requested our address before. Also notice that this slows down input processing of every IP packet!

There are two known issues in real application if this feature is enabled: - The LAN peer may have bug to update the ARP table after the ARP entry is aged out. If the ARP entry on the LAN peer is aged out but failed to be updated, all IP packets sent from LWIP to the LAN peer will be dropped by LAN peer. - The LAN peer may not be trustful, the LAN peer may send IP packets to LWIP with two different MACs, but the same IP address. If this happens, the LWIP has problem to receive IP packets from LAN peer.

So the recommendation is to disable this option. Here the LAN peer means the other side to which the ESP station or soft-AP is connected.

CONFIG_LWIP_ESP_GRATUITOUS_ARP

Send gratuitous ARP periodically

Found in: Component config > LWIP

Enable this option allows to send gratuitous ARP periodically.

This option solve the compatibility issues.If the ARP table of the AP is old, and the AP doesn't send ARP request to update it's ARP table, this will lead to the STA sending IP packet fail. Thus we send gratuitous ARP periodically to let AP update it's ARP table.

CONFIG_LWIP_GARP_TMR_INTERVAL

GARP timer interval(seconds)

Found in: Component config > LWIP > CONFIG_LWIP_ESP_GRATUITOUS_ARP

Set the timer interval for gratuitous ARP. The default value is 60s

CONFIG_LWIP_TCPIP_RECVMBOX_SIZE

TCPIP task receive mail box size

Found in: Component config > LWIP

Set TCPIP task receive mail box size. Generally bigger value means higher throughput but more memory. The value should be bigger than UDP/TCP mail box size.

CONFIG_LWIP_DHCP_DOES_ARP_CHECK

DHCP: Perform ARP check on any offered address

Found in: Component config > LWIP

Enabling this option performs a check (via ARP request) if the offered IP address is not already in use by another host on the network.

CONFIG_LWIP_DHCP_RESTORE_LAST_IP

DHCP: Restore last IP obtained from DHCP server

Found in: Component config > LWIP

When this option is enabled, DHCP client tries to re-obtain last valid IP address obtained from DHCP server. Last valid DHCP configuration is stored in nvs and restored after reset/power-up. If IP is still available, there is no need for sending discovery message to DHCP server and save some time.

DHCP server

Contains:

- *CONFIG_LWIP_DHCPS_LEASE_UNIT*
- *CONFIG_LWIP_DHCPS_MAX_STATION_NUM*

CONFIG_LWIP_DHCPS_LEASE_UNIT

Multiplier for lease time, in seconds

Found in: Component config > LWIP > DHCP server

The DHCP server is calculating lease time multiplying the sent and received times by this number of seconds per unit. The default is 60, that equals one minute.

CONFIG_LWIP_DHCPS_MAX_STATION_NUM

Maximum number of stations

Found in: Component config > LWIP > DHCP server

The maximum number of DHCP clients that are connected to the server. After this number is exceeded, DHCP server removes of the oldest device from it' s address pool, without notification.

CONFIG_LWIP_AUTOIP

Enable IPV4 Link-Local Addressing (AUTOIP)

Found in: Component config > LWIP

Enabling this option allows the device to self-assign an address in the 169.256/16 range if none is assigned statically or via DHCP.

See RFC 3927.

Contains:

- *CONFIG_LWIP_AUTOIP_TRIES*
- *CONFIG_LWIP_AUTOIP_MAX_CONFLICTS*
- *CONFIG_LWIP_AUTOIP_RATE_LIMIT_INTERVAL*

CONFIG_LWIP_AUTOIP_TRIES

DHCP Probes before self-assigning IPv4 LL address

Found in: Component config > LWIP > CONFIG_LWIP_AUTOIP

DHCP client will send this many probes before self-assigning a link local address.

From LWIP help: "This can be set as low as 1 to get an AutoIP address very quickly, but you should be prepared to handle a changing IP address when DHCP overrides AutoIP." (In the case of ESP-IDF, this means multiple SYSTEM_EVENT_STA_GOT_IP events.)

CONFIG_LWIP_AUTOIP_MAX_CONFLICTS

Max IP conflicts before rate limiting

Found in: Component config > LWIP > CONFIG_LWIP_AUTOIP

If the AUTOIP functionality detects this many IP conflicts while self-assigning an address, it will go into a rate limited mode.

CONFIG_LWIP_AUTOIP_RATE_LIMIT_INTERVAL

Rate limited interval (seconds)

Found in: Component config > LWIP > CONFIG_LWIP_AUTOIP

If rate limiting self-assignment requests, wait this long between each request.

CONFIG_LWIP_IPV6_AUTOCONFIG

Enable IPV6 stateless address autoconfiguration

Found in: Component config > LWIP

Enabling this option allows the devices to IPV6 stateless address autoconfiguration.

See RFC 4862.

CONFIG_LWIP_NETIF_LOOPBACK

Support per-interface loopback

Found in: Component config > LWIP

Enabling this option means that if a packet is sent with a destination address equal to the interface' s own IP address, it will "loop back" and be received by this interface.

Contains:

- *CONFIG_LWIP_LOOPBACK_MAX_PBUFS*

CONFIG_LWIP_LOOPBACK_MAX_PBUFS

Max queued loopback packets per interface

Found in: Component config > LWIP > CONFIG_LWIP_NETIF_LOOPBACK

Configure the maximum number of packets which can be queued for loopback on a given interface. Reducing this number may cause packets to be dropped, but will avoid filling memory with queued packet data.

TCP

Contains:

- *CONFIG_LWIP_MAX_ACTIVE_TCP*
- *CONFIG_LWIP_MAX_LISTENING_TCP*
- *CONFIG_LWIP_TCP_MAXRTX*
- *CONFIG_LWIP_TCP_SYNMAXRTX*
- *CONFIG_LWIP_TCP_MSS*
- *CONFIG_LWIP_TCP_TMR_INTERVAL*
- *CONFIG_LWIP_TCP_MSL*
- *CONFIG_LWIP_TCP_SND_BUF_DEFAULT*
- *CONFIG_LWIP_TCP_WND_DEFAULT*
- *CONFIG_LWIP_TCP_RECVMBOX_SIZE*
- *CONFIG_LWIP_TCP_QUEUE_OOSEQ*
- *CONFIG_LWIP_TCP_SACK_OUT*
- *CONFIG_LWIP_TCP_KEEP_CONNECTION_WHEN_IP_CHANGES*

- *CONFIG_LWIP_TCP_OVERSIZE*
- *CONFIG_LWIP_WND_SCALE*

CONFIG_LWIP_MAX_ACTIVE_TCP

Maximum active TCP Connections

Found in: Component config > LWIP > TCP

The maximum number of simultaneously active TCP connections. The practical maximum limit is determined by available heap memory at runtime.

Changing this value by itself does not substantially change the memory usage of LWIP, except for preventing new TCP connections after the limit is reached.

CONFIG_LWIP_MAX_LISTENING_TCP

Maximum listening TCP Connections

Found in: Component config > LWIP > TCP

The maximum number of simultaneously listening TCP connections. The practical maximum limit is determined by available heap memory at runtime.

Changing this value by itself does not substantially change the memory usage of LWIP, except for preventing new listening TCP connections after the limit is reached.

CONFIG_LWIP_TCP_MAXRTX

Maximum number of retransmissions of data segments

Found in: Component config > LWIP > TCP

Set maximum number of retransmissions of data segments.

CONFIG_LWIP_TCP_SYNMAXRTX

Maximum number of retransmissions of SYN segments

Found in: Component config > LWIP > TCP

Set maximum number of retransmissions of SYN segments.

CONFIG_LWIP_TCP_MSS

Maximum Segment Size (MSS)

Found in: Component config > LWIP > TCP

Set maximum segment size for TCP transmission.

Can be set lower to save RAM, the default value 1460(ipv4)/1440(ipv6) will give best throughput. IPv4 TCP_MSS Range: 576 <= TCP_MSS <= 1460 IPv6 TCP_MSS Range: 1220<= TCP_mSS <= 1440

CONFIG_LWIP_TCP_TMR_INTERVAL

TCP timer interval(ms)

Found in: Component config > LWIP > TCP

Set TCP timer interval in milliseconds.

Can be used to speed connections on bad networks. A lower value will redeliver unacked packets faster.

CONFIG_LWIP_TCP_MSL

Maximum segment lifetime (MSL)

Found in: Component config > LWIP > TCP

Set maximum segment lifetime in in milliseconds.

CONFIG_LWIP_TCP_SND_BUF_DEFAULT

Default send buffer size

Found in: Component config > LWIP > TCP

Set default send buffer size for new TCP sockets.

Per-socket send buffer size can be changed at runtime with lwip_setsockopt(s, TCP_SNDBUF, ···).

This value must be at least 2x the MSS size, and the default is 4x the default MSS size.

Setting a smaller default SNDBUF size can save some RAM, but will decrease performance.

CONFIG_LWIP_TCP_WND_DEFAULT

Default receive window size

Found in: Component config > LWIP > TCP

Set default TCP receive window size for new TCP sockets.

Per-socket receive window size can be changed at runtime with lwip_setsockopt(s, TCP_WINDOW, ···).

Setting a smaller default receive window size can save some RAM, but will significantly decrease performance.

CONFIG_LWIP_TCP_RECVMBOX_SIZE

Default TCP receive mail box size

Found in: Component config > LWIP > TCP

Set TCP receive mail box size. Generally bigger value means higher throughput but more memory. The recommended value is: LWIP_TCP_WND_DEFAULT/TCP_MSS + 2, e.g. if LWIP_TCP_WND_DEFAULT=14360, TCP_MSS=1436, then the recommended receive mail box size is (14360/1436 + 2) = 12.

TCP receive mail box is a per socket mail box, when the application receives packets from TCP socket, LWIP core firstly posts the packets to TCP receive mail box and the application then fetches the packets from mail box. It means LWIP can caches maximum LWIP_TCP_RECCVMBOX_SIZE packets for each TCP socket, so the maximum possible cached TCP packets for all TCP sockets is LWIP_TCP_RECCVMBOX_SIZE multiples the maximum TCP socket number. In other words, the bigger LWIP_TCP_RECVMBOX_SIZE means more memory. On the other hand, if the receiv mail box is too small, the mail box may be full. If the mail box is full, the LWIP drops the packets. So generally we need to make sure the TCP receive mail box is big enough to avoid packet drop between LWIP core and application.

CONFIG_LWIP_TCP_QUEUE_OOSEQ

Queue incoming out-of-order segments

Found in: Component config > LWIP > TCP

Queue incoming out-of-order segments for later use.

Disable this option to save some RAM during TCP sessions, at the expense of increased retransmissions if segments arrive out of order.

CONFIG_LWIP_TCP_SACK_OUT

Support sending selective acknowledgements

Found in: Component config > LWIP > TCP

TCP will support sending selective acknowledgements (SACKs).

CONFIG_LWIP_TCP_KEEP_CONNECTION_WHEN_IP_CHANGES

Keep TCP connections when IP changed

Found in: Component config > LWIP > TCP

This option is enabled when the following scenario happen: network dropped and reconnected, IP changes is like: 192.168.0.2->0.0.0.0->192.168.0.2

Disable this option to keep consistent with the original LWIP code behavior.

CONFIG_LWIP_TCP_OVERSIZE

Pre-allocate transmit PBUF size

Found in: Component config > LWIP > TCP

Allows enabling "oversize" allocation of TCP transmission pbufs ahead of time, which can reduce the length of pbuf chains used for transmission.

This will not make a difference to sockets where Nagle' s algorithm is disabled.

Default value of MSS is fine for most applications, 25% MSS may save some RAM when only transmitting small amounts of data. Disabled will have worst performance and fragmentation characteristics, but uses least RAM overall.

Available options:

- MSS (LWIP_TCP_OVERSIZE_MSS)
- 25% MSS (LWIP_TCP_OVERSIZE_QUARTER_MSS)
- Disabled (LWIP_TCP_OVERSIZE_DISABLE)

CONFIG_LWIP_WND_SCALE

Support TCP window scale

Found in: Component config > LWIP > TCP

Enable this feature to support TCP window scaling.

CONFIG_LWIP_TCP_RCV_SCALE

Set TCP receiving window scaling factor

Found in: Component config > LWIP > TCP > CONFIG_LWIP_WND_SCALE

Enable this feature to support TCP window scaling.

UDP

Contains:

- *CONFIG_LWIP_MAX_UDP_PCBS*
- *CONFIG_LWIP_UDP_RECVMBOX_SIZE*

CONFIG_LWIP_MAX_UDP_PCBS

Maximum active UDP control blocks

Found in: Component config > LWIP > UDP

The maximum number of active UDP "connections" (ie UDP sockets sending/receiving data). The practical maximum limit is determined by available heap memory at runtime.

CONFIG_LWIP_UDP_RECVMBOX_SIZE

Default UDP receive mail box size

Found in: Component config > LWIP > UDP

Set UDP receive mail box size. The recommended value is 6.

UDP receive mail box is a per socket mail box, when the application receives packets from UDP socket, LWIP core firstly posts the packets to UDP receive mail box and the application then fetches the packets from mail box. It means LWIP can caches maximum UDP_RECCVMBOX_SIZE packets for each UDP socket, so the maximum possible cached UDP packets for all UDP sockets is UDP_RECCVMBOX_SIZE multiples the maximum UDP socket number. In other words, the bigger UDP_RECVMBOX_SIZE means more memory. On the other hand, if the receiv mail box is too small, the mail box may be full. If the mail box is full, the LWIP drops the packets. So generally we need to make sure the UDP receive mail box is big enough to avoid packet drop between LWIP core and application.

CONFIG_LWIP_TCPIP_TASK_STACK_SIZE

TCP/IP Task Stack Size

Found in: Component config > LWIP

Configure TCP/IP task stack size, used by LWIP to process multi-threaded TCP/IP operations. Setting this stack too small will result in stack overflow crashes.

CONFIG_LWIP_TCPIP_TASK_AFFINITY

TCP/IP task affinity

Found in: Component config > LWIP

Allows setting LwIP tasks affinity, i.e. whether the task is pinned to CPU0, pinned to CPU1, or allowed to run on any CPU. Currently this applies to "TCP/IP" task and "Ping" task.

Available options:

- No affinity (LWIP_TCPIP_TASK_AFFINITY_NO_AFFINITY)
- CPU0 (LWIP_TCPIP_TASK_AFFINITY_CPU0)
- CPU1 (LWIP_TCPIP_TASK_AFFINITY_CPU1)

CONFIG_LWIP_PPP_SUPPORT

Enable PPP support (new/experimental)

Found in: Component config > LWIP

Enable PPP stack. Now only PPP over serial is possible.

PPP over serial support is experimental and unsupported.

Contains:

- *CONFIG_LWIP_PPP_ENABLE_IPV6*
- *CONFIG_LWIP_PPP_NOTIFY_PHASE_SUPPORT*
- *CONFIG_LWIP_PPP_PAP_SUPPORT*
- *CONFIG_LWIP_PPP_CHAP_SUPPORT*
- *CONFIG_LWIP_PPP_MSCHAP_SUPPORT*
- *CONFIG_LWIP_PPP_MPPE_SUPPORT*
- *CONFIG_LWIP_PPP_DEBUG_ON*

CONFIG_LWIP_PPP_ENABLE_IPV6

Enable IPV6 support for PPP connections (IPV6CP)

Found in: Component config > LWIP > CONFIG_LWIP_PPP_SUPPORT

Enable IPV6 support in PPP for the local link between the DTE (processor) and DCE (modem). There are some modems which do not support the IPV6 addressing in the local link. If they are requested for IPV6CP negotiation, they may time out. This would in turn fail the configuration for the whole link. If your modem is not responding correctly to PPP Phase Network, try to disable IPV6 support.

CONFIG_LWIP_PPP_NOTIFY_PHASE_SUPPORT

Enable Notify Phase Callback

Found in: Component config > LWIP > CONFIG_LWIP_PPP_SUPPORT

Enable to set a callback which is called on change of the internal PPP state machine.

CONFIG_LWIP_PPP_PAP_SUPPORT

Enable PAP support

Found in: Component config > LWIP > CONFIG_LWIP_PPP_SUPPORT

Enable Password Authentication Protocol (PAP) support

CONFIG_LWIP_PPP_CHAP_SUPPORT

Enable CHAP support

Found in: Component config > LWIP > CONFIG_LWIP_PPP_SUPPORT

Enable Challenge Handshake Authentication Protocol (CHAP) support

CONFIG_LWIP_PPP_MSCHAP_SUPPORT

Enable MSCHAP support

Found in: Component config > LWIP > CONFIG_LWIP_PPP_SUPPORT

Enable Microsoft version of the Challenge-Handshake Authentication Protocol (MSCHAP) support

CONFIG_LWIP_PPP_MPPE_SUPPORT

Enable MPPE support

Found in: Component config > LWIP > CONFIG_LWIP_PPP_SUPPORT

Enable Microsoft Point-to-Point Encryption (MPPE) support

CONFIG_LWIP_PPP_DEBUG_ON

Enable PPP debug log output

Found in: Component config > LWIP > CONFIG_LWIP_PPP_SUPPORT

Enable PPP debug log output

ICMP

Contains:

- *CONFIG_LWIP_MULTICAST_PING*
- *CONFIG_LWIP_BROADCAST_PING*

CONFIG_LWIP_MULTICAST_PING

Respond to multicast pings

Found in: Component config > LWIP > ICMP

CONFIG_LWIP_BROADCAST_PING

Respond to broadcast pings

Found in: Component config > LWIP > ICMP

LWIP RAW API

Contains:

- *CONFIG_LWIP_MAX_RAW_PCBS*

CONFIG_LWIP_MAX_RAW_PCBS

Maximum LWIP RAW PCBs

Found in: Component config > LWIP > LWIP RAW API

The maximum number of simultaneously active LWIP RAW protocol control blocks. The practical maximum limit is determined by available heap memory at runtime.

SNTP

Contains:

- *CONFIG_LWIP_DHCP_MAX_NTP_SERVERS*
- *CONFIG_LWIP_SNTP_UPDATE_DELAY*

CONFIG_LWIP_DHCP_MAX_NTP_SERVERS

Maximum number of NTP servers

Found in: Component config > LWIP > SNTP

Set maximum number of NTP servers used by LwIP SNTP module. First argument of sntp_setserver/sntp_setservername functions is limited to this value.

CONFIG_LWIP_SNTP_UPDATE_DELAY

Request interval to update time (ms)

Found in: Component config > LWIP > SNTP

This option allows you to set the time update period via SNTP. Default is 1 hour. Must not be below 15 seconds by specification. (SNTPv4 RFC 4330 enforces a minimum update time of 15 seconds).

mbedTLS

Contains:

- *CONFIG_MBEDTLS_MEM_ALLOC_MODE*
- *CONFIG_MBEDTLS_SSL_MAX_CONTENT_LEN*
- *CONFIG_MBEDTLS_ASYMMETRIC_CONTENT_LEN*
- *CONFIG_MBEDTLS_DEBUG*
- *CONFIG_MBEDTLS_ECP_RESTARTABLE*
- *CONFIG_MBEDTLS_CMAC_C*
- *CONFIG_MBEDTLS_HARDWARE_AES*
- *CONFIG_MBEDTLS_HARDWARE_MPI*
- *CONFIG_MBEDTLS_HARDWARE_SHA*
- *CONFIG_MBEDTLS_HAVE_TIME*

- *CONFIG_MBEDTLS_TLS_MODE*
- *TLS Key Exchange Methods*
- *CONFIG_MBEDTLS_SSL_RENEGOTIATION*
- *CONFIG_MBEDTLS_SSL_PROTO_SSL3*
- *CONFIG_MBEDTLS_SSL_PROTO_TLS1*
- *CONFIG_MBEDTLS_SSL_PROTO_TLS1_1*
- *CONFIG_MBEDTLS_SSL_PROTO_TLS1_2*
- *CONFIG_MBEDTLS_SSL_PROTO_DTLS*
- *CONFIG_MBEDTLS_SSL_ALPN*
- *CONFIG_MBEDTLS_CLIENT_SSL_SESSION_TICKETS*
- *CONFIG_MBEDTLS_SERVER_SSL_SESSION_TICKETS*
- *Symmetric Ciphers*
- *CONFIG_MBEDTLS_RIPEMD160_C*
- *Certificates*
- *CONFIG_MBEDTLS_ECP_C*
- *CONFIG_MBEDTLS_SECURITY_RISKS*

CONFIG_MBEDTLS_MEM_ALLOC_MODE

Memory allocation strategy

Found in: Component config > mbedTLS

Allocation strategy for mbedTLS, essentially provides ability to allocate all required dynamic allocations from,

- Internal DRAM memory only
- External SPIRAM memory only
- Either internal or external memory based on default malloc() behavior in ESP-IDF
- Custom allocation mode, by overwriting calloc()/free() using mbedtls_platform_set_calloc_free() function

Recommended mode here is always internal, since that is most preferred from security perspective. But if application requirement does not allow sufficient free internal memory then alternate mode can be selected.

Available options:

- Internal memory (MBEDTLS_INTERNAL_MEM_ALLOC)
- External SPIRAM (MBEDTLS_EXTERNAL_MEM_ALLOC)
- Default alloc mode (MBEDTLS_DEFAULT_MEM_ALLOC)
- Custom alloc mode (MBEDTLS_CUSTOM_MEM_ALLOC)

CONFIG_MBEDTLS_SSL_MAX_CONTENT_LEN

TLS maximum message content length

Found in: Component config > mbedTLS

Maximum TLS message length (in bytes) supported by mbedTLS.

16384 is the default and this value is required to comply fully with TLS standards.

However you can set a lower value in order to save RAM. This is safe if the other end of the connection supports Maximum Fragment Length Negotiation Extension (max_fragment_length, see RFC6066) or you know for certain that it will never send a message longer than a certain number of bytes.

If the value is set too low, symptoms are a failed TLS handshake or a return value of MBEDTLS_ERR_SSL_INVALID_RECORD (-0x7200).

CONFIG_MBEDTLS_ASYMMETRIC_CONTENT_LEN

Asymmetric in/out fragment length

Found in: Component config > mbedTLS

If enabled, this option allows customizing TLS in/out fragment length in asymmetric way. Please note that enabling this with default values saves 12KB of dynamic memory per TLS connection.

CONFIG_MBEDTLS_SSL_IN_CONTENT_LEN

TLS maximum incoming fragment length

Found in: Component config > mbedTLS > CONFIG_MBEDTLS_ASYMMETRIC_CONTENT_LEN

This defines maximum incoming fragment length, overriding default maximum content length (MBEDTLS_SSL_MAX_CONTENT_LEN).

CONFIG_MBEDTLS_SSL_OUT_CONTENT_LEN

TLS maximum outgoing fragment length

Found in: Component config > mbedTLS > CONFIG_MBEDTLS_ASYMMETRIC_CONTENT_LEN

This defines maximum outgoing fragment length, overriding default maximum content length (MBEDTLS_SSL_MAX_CONTENT_LEN).

CONFIG_MBEDTLS_DEBUG

Enable mbedTLS debugging

Found in: Component config > mbedTLS

Enable mbedTLS debugging functions at compile time.

If this option is enabled, you can include "mbedtls/esp_debug.h" and call mbedtls_esp_enable_debug_log() at runtime in order to enable mbedTLS debug output via the ESP log mechanism.

CONFIG_MBEDTLS_DEBUG_LEVEL

Set mbedTLS debugging level

Found in: Component config > mbedTLS > CONFIG_MBEDTLS_DEBUG

Set mbedTLS debugging level

Available options:

- Warning (MBEDTLS_DEBUG_LEVEL_WARN)
- Info (MBEDTLS_DEBUG_LEVEL_INFO)
- Debug (MBEDTLS_DEBUG_LEVEL_DEBUG)
- Verbose (MBEDTLS_DEBUG_LEVEL_VERBOSE)

CONFIG_MBEDTLS_ECP_RESTARTABLE

Enable mbedTLS ecp restartable

Found in: Component config > mbedTLS

Enable "non-blocking" ECC operations that can return early and be resumed.

CONFIG_MBEDTLS_CMAC_C

Enable CMAC mode for block ciphers

Found in: Component config > mbedTLS

Enable the CMAC (Cipher-based Message Authentication Code) mode for block ciphers.

CONFIG_MBEDTLS_HARDWARE_AES

Enable hardware AES acceleration

Found in: Component config > mbedTLS

Enable hardware accelerated AES encryption & decryption.

Note that if the ESP32 CPU is running at 240MHz, hardware AES does not offer any speed boost over software AES.

CONFIG_MBEDTLS_HARDWARE_MPI

Enable hardware MPI (bignum) acceleration

Found in: Component config > mbedTLS

Enable hardware accelerated multiple precision integer operations.

Hardware accelerated multiplication, modulo multiplication, and modular exponentiation for up to 4096 bit results.

These operations are used by RSA.

CONFIG_MBEDTLS_HARDWARE_SHA

Enable hardware SHA acceleration

Found in: Component config > mbedTLS

Enable hardware accelerated SHA1, SHA256, SHA384 & SHA512 in mbedTLS.

Due to a hardware limitation, hardware acceleration is only guaranteed if SHA digests are calculated one at a time. If more than one SHA digest is calculated at the same time, one will be calculated fully in hardware and the rest will be calculated (at least partially calculated) in software. This happens automatically.

SHA hardware acceleration is faster than software in some situations but slower in others. You should benchmark to find the best setting for you.

CONFIG_MBEDTLS_HAVE_TIME

Enable mbedtls time

Found in: Component config > mbedTLS

System has time.h and time(). The time does not need to be correct, only time differences are used.

CONFIG_MBEDTLS_HAVE_TIME_DATE

Enable mbedtls certificate expiry check

Found in: Component config > mbedTLS > CONFIG_MBEDTLS_HAVE_TIME

System has time.h and time(), gmtime() and the clock is correct. The time needs to be correct (not necesarily very accurate, but at least the date should be correct). This is used to verify the validity period of X.509 certificates.

It is suggested that you should get the real time by "SNTP".

CONFIG_MBEDTLS_TLS_MODE

TLS Protocol Role

Found in: Component config > mbedTLS

mbedTLS can be compiled with protocol support for the TLS server, TLS client, or both server and client.

Reducing the number of TLS roles supported saves code size.

Available options:

- Server & Client (MBEDTLS_TLS_SERVER_AND_CLIENT)
- Server (MBEDTLS_TLS_SERVER_ONLY)
- Client (MBEDTLS_TLS_CLIENT_ONLY)
- None (MBEDTLS_TLS_DISABLED)

TLS Key Exchange Methods

Contains:

- *CONFIG_MBEDTLS_PSK_MODES*
- *CONFIG_MBEDTLS_KEY_EXCHANGE_RSA*
- *CONFIG_MBEDTLS_KEY_EXCHANGE_DHE_RSA*
- *CONFIG_MBEDTLS_KEY_EXCHANGE_ELLIPTIC_CURVE*

CONFIG_MBEDTLS_PSK_MODES

Enable pre-shared-key ciphersuites

Found in: Component config > mbedTLS > TLS Key Exchange Methods

Enable to show configuration for different types of pre-shared-key TLS authentatication methods.

Leaving this options disabled will save code size if they are not used.

CONFIG_MBEDTLS_KEY_EXCHANGE_PSK

Enable PSK based ciphersuite modes

Found in: Component config > mbedTLS > TLS Key Exchange Methods > CONFIG_MBEDTLS_PSK_MODES

Enable to support symmetric key PSK (pre-shared-key) TLS key exchange modes.

CONFIG_MBEDTLS_KEY_EXCHANGE_DHE_PSK

Enable DHE-PSK based ciphersuite modes

Found in: Component config > mbedTLS > TLS Key Exchange Methods > CONFIG_MBEDTLS_PSK_MODES

Enable to support Diffie-Hellman PSK (pre-shared-key) TLS authentication modes.

CONFIG_MBEDTLS_KEY_EXCHANGE_ECDHE_PSK

Enable ECDHE-PSK based ciphersuite modes

Found in: Component config > mbedTLS > TLS Key Exchange Methods > CONFIG_MBEDTLS_PSK_MODES

Enable to support Elliptic-Curve-Diffie-Hellman PSK (pre-shared-key) TLS authentication modes.

CONFIG_MBEDTLS_KEY_EXCHANGE_RSA_PSK

Enable RSA-PSK based ciphersuite modes

Found in: Component config > mbedTLS > TLS Key Exchange Methods > CONFIG_MBEDTLS_PSK_MODES

Enable to support RSA PSK (pre-shared-key) TLS authentication modes.

CONFIG_MBEDTLS_KEY_EXCHANGE_RSA

Enable RSA-only based ciphersuite modes

Found in: Component config > mbedTLS > TLS Key Exchange Methods

Enable to support ciphersuites with prefix TLS-RSA-WITH-

CONFIG_MBEDTLS_KEY_EXCHANGE_DHE_RSA

Enable DHE-RSA based ciphersuite modes

Found in: Component config > mbedTLS > TLS Key Exchange Methods

Enable to support ciphersuites with prefix TLS-DHE-RSA-WITH-

CONFIG_MBEDTLS_KEY_EXCHANGE_ELLIPTIC_CURVE

Support Elliptic Curve based ciphersuites

Found in: Component config > mbedTLS > TLS Key Exchange Methods

Enable to show Elliptic Curve based ciphersuite mode options.

Disabling all Elliptic Curve ciphersuites saves code size and can give slightly faster TLS handshakes, provided the server supports RSA-only ciphersuite modes.

CONFIG_MBEDTLS_KEY_EXCHANGE_ECDHE_RSA

Enable ECDHE-RSA based ciphersuite modes

Found in: Component config > mbedTLS > TLS Key Exchange Methods > CONFIG_MBEDTLS_KEY_EXCHANGE_ELLIPTIC_CURVE

Enable to support ciphersuites with prefix TLS-ECDHE-RSA-WITH-

CONFIG_MBEDTLS_KEY_EXCHANGE_ECDHE_ECDSA

Enable ECDHE-ECDSA based ciphersuite modes

Found in: Component config > mbedTLS > TLS Key Exchange Methods > CONFIG_MBEDTLS_KEY_EXCHANGE_ELLIPTIC_CURVE

Enable to support ciphersuites with prefix TLS-ECDHE-RSA-WITH-

CONFIG_MBEDTLS_KEY_EXCHANGE_ECDH_ECDSA

Enable ECDH-ECDSA based ciphersuite modes

Found in: Component config > mbedTLS > TLS Key Exchange Methods > CONFIG_MBEDTLS_KEY_EXCHANGE_ELLIPTIC_CURVE

Enable to support ciphersuites with prefix TLS-ECDHE-RSA-WITH-

CONFIG_MBEDTLS_KEY_EXCHANGE_ECDH_RSA

Enable ECDH-RSA based ciphersuite modes

Found in: Component config > mbedTLS > TLS Key Exchange Methods > CONFIG_MBEDTLS_KEY_EXCHANGE_ELLIPTIC_CURVE

Enable to support ciphersuites with prefix TLS-ECDHE-RSA-WITH-

CONFIG_MBEDTLS_SSL_RENEGOTIATION

Support TLS renegotiation

Found in: Component config > mbedTLS

The two main uses of renegotiation are (1) refresh keys on long-lived connections and (2) client authentication after the initial handshake. If you don' t need renegotiation, disabling it will save code size and reduce the possibility of abuse/vulnerability.

CONFIG_MBEDTLS_SSL_PROTO_SSL3

Legacy SSL 3.0 support

Found in: Component config > mbedTLS

Support the legacy SSL 3.0 protocol. Most servers will speak a newer TLS protocol these days.

CONFIG_MBEDTLS_SSL_PROTO_TLS1

Support TLS 1.0 protocol

Found in: Component config > mbedTLS

CONFIG_MBEDTLS_SSL_PROTO_TLS1_1

Support TLS 1.1 protocol

Found in: Component config > mbedTLS

CONFIG_MBEDTLS_SSL_PROTO_TLS1_2

Support TLS 1.2 protocol

Found in: Component config > mbedTLS

CONFIG_MBEDTLS_SSL_PROTO_DTLS

Support DTLS protocol (all versions)

Found in: Component config > mbedTLS

Requires TLS 1.1 to be enabled for DTLS 1.0 Requires TLS 1.2 to be enabled for DTLS 1.2

CONFIG_MBEDTLS_SSL_ALPN

Support ALPN (Application Layer Protocol Negotiation)

Found in: Component config > mbedTLS

Disabling this option will save some code size if it is not needed.

CONFIG_MBEDTLS_CLIENT_SSL_SESSION_TICKETS

TLS: Client Support for RFC 5077 SSL session tickets

Found in: Component config > mbedTLS

Client support for RFC 5077 session tickets. See mbedTLS documentation for more details. Disabling this option will save some code size.

CONFIG_MBEDTLS_SERVER_SSL_SESSION_TICKETS

TLS: Server Support for RFC 5077 SSL session tickets

Found in: Component config > mbedTLS

Server support for RFC 5077 session tickets. See mbedTLS documentation for more details. Disabling this option will save some code size.

Symmetric Ciphers

Contains:

- *CONFIG_MBEDTLS_AES_C*
- *CONFIG_MBEDTLS_CAMELLIA_C*
- *CONFIG_MBEDTLS_DES_C*
- *CONFIG_MBEDTLS_RC4_MODE*
- *CONFIG_MBEDTLS_BLOWFISH_C*
- *CONFIG_MBEDTLS_XTEA_C*

- *CONFIG_MBEDTLS_CCM_C*
- *CONFIG_MBEDTLS_GCM_C*

CONFIG_MBEDTLS_AES_C

AES block cipher

Found in: Component config > mbedTLS > Symmetric Ciphers

CONFIG_MBEDTLS_CAMELLIA_C

Camellia block cipher

Found in: Component config > mbedTLS > Symmetric Ciphers

CONFIG_MBEDTLS_DES_C

DES block cipher (legacy, insecure)

Found in: Component config > mbedTLS > Symmetric Ciphers

Enables the DES block cipher to support 3DES-based TLS ciphersuites.

3DES is vulnerable to the Sweet32 attack and should only be enabled if absolutely necessary.

CONFIG_MBEDTLS_RC4_MODE

RC4 Stream Cipher (legacy, insecure)

Found in: Component config > mbedTLS > Symmetric Ciphers

ARCFOUR (RC4) stream cipher can be disabled entirely, enabled but not added to default ciphersuites, or enabled completely.

Please consider the security implications before enabling RC4.

Available options:

- Disabled (MBEDTLS_RC4_DISABLED)
- Enabled, not in default ciphersuites (MBEDTLS_RC4_ENABLED_NO_DEFAULT)
- Enabled (MBEDTLS_RC4_ENABLED)

CONFIG_MBEDTLS_BLOWFISH_C

Blowfish block cipher (read help)

Found in: Component config > mbedTLS > Symmetric Ciphers

Enables the Blowfish block cipher (not used for TLS sessions.)

The Blowfish cipher is not used for mbedTLS TLS sessions but can be used for other purposes. Read up on the limitations of Blowfish (including Sweet32) before enabling.

CONFIG_MBEDTLS_XTEA_C

XTEA block cipher

Found in: Component config > mbedTLS > Symmetric Ciphers

Enables the XTEA block cipher.

CONFIG_MBEDTLS_CCM_C

CCM (Counter with CBC-MAC) block cipher modes

Found in: Component config > mbedTLS > Symmetric Ciphers

Enable Counter with CBC-MAC (CCM) modes for AES and/or Camellia ciphers.

Disabling this option saves some code size.

CONFIG_MBEDTLS_GCM_C

GCM (Galois/Counter) block cipher modes

Found in: Component config > mbedTLS > Symmetric Ciphers

Enable Galois/Counter Mode for AES and/or Camellia ciphers.

This option is generally faster than CCM.

CONFIG_MBEDTLS_RIPEMD160_C

Enable RIPEMD-160 hash algorithm

Found in: Component config > mbedTLS

Enable the RIPEMD-160 hash algorithm.

Certificates

Contains:

- *CONFIG_MBEDTLS_PEM_PARSE_C*
- *CONFIG_MBEDTLS_PEM_WRITE_C*
- *CONFIG_MBEDTLS_X509_CRL_PARSE_C*
- *CONFIG_MBEDTLS_X509_CSR_PARSE_C*

CONFIG_MBEDTLS_PEM_PARSE_C

Read & Parse PEM formatted certificates

Found in: Component config > mbedTLS > Certificates

Enable decoding/parsing of PEM formatted certificates.

If your certificates are all in the simpler DER format, disabling this option will save some code size.

CONFIG_MBEDTLS_PEM_WRITE_C

Write PEM formatted certificates

Found in: Component config > mbedTLS > Certificates

Enable writing of PEM formatted certificates.

If writing certificate data only in DER format, disabling this option will save some code size.

CONFIG_MBEDTLS_X509_CRL_PARSE_C

X.509 CRL parsing

Found in: Component config > mbedTLS > Certificates

Support for parsing X.509 Certifificate Revocation Lists.

CONFIG_MBEDTLS_X509_CSR_PARSE_C

X.509 CSR parsing

Found in: Component config > mbedTLS > Certificates

Support for parsing X.509 Certifificate Signing Requests

CONFIG_MBEDTLS_ECP_C

Elliptic Curve Ciphers

Found in: Component config > mbedTLS

Contains:

- *CONFIG_MBEDTLS_ECDH_C*
- *CONFIG_MBEDTLS_ECP_DP_SECP192R1_ENABLED*
- *CONFIG_MBEDTLS_ECP_DP_SECP224R1_ENABLED*
- *CONFIG_MBEDTLS_ECP_DP_SECP256R1_ENABLED*
- *CONFIG_MBEDTLS_ECP_DP_SECP384R1_ENABLED*
- *CONFIG_MBEDTLS_ECP_DP_SECP521R1_ENABLED*
- *CONFIG_MBEDTLS_ECP_DP_SECP192K1_ENABLED*
- *CONFIG_MBEDTLS_ECP_DP_SECP224K1_ENABLED*
- *CONFIG_MBEDTLS_ECP_DP_SECP256K1_ENABLED*
- *CONFIG_MBEDTLS_ECP_DP_BP256R1_ENABLED*
- *CONFIG_MBEDTLS_ECP_DP_BP384R1_ENABLED*
- *CONFIG_MBEDTLS_ECP_DP_BP512R1_ENABLED*
- *CONFIG_MBEDTLS_ECP_DP_CURVE25519_ENABLED*
- *CONFIG_MBEDTLS_ECP_NIST_OPTIM*

CONFIG_MBEDTLS_ECDH_C

Elliptic Curve Diffie-Hellman (ECDH)

Found in: Component config > mbedTLS > CONFIG_MBEDTLS_ECP_C

Enable ECDH. Needed to use ECDHE-xxx TLS ciphersuites.

CONFIG_MBEDTLS_ECDSA_C

Elliptic Curve DSA

Found in: Component config > mbedTLS > CONFIG_MBEDTLS_ECP_C > CONFIG_MBEDTLS_ECDH_C

Enable ECDSA. Needed to use ECDSA-xxx TLS ciphersuites.

CONFIG_MBEDTLS_ECP_DP_SECP192R1_ENABLED

Enable SECP192R1 curve

Found in: Component config > mbedTLS > CONFIG_MBEDTLS_ECP_C

Enable support for SECP192R1 Elliptic Curve.

CONFIG_MBEDTLS_ECP_DP_SECP224R1_ENABLED

Enable SECP224R1 curve

Found in: Component config > mbedTLS > CONFIG_MBEDTLS_ECP_C

Enable support for SECP224R1 Elliptic Curve.

CONFIG_MBEDTLS_ECP_DP_SECP256R1_ENABLED

Enable SECP256R1 curve

Found in: Component config > mbedTLS > CONFIG_MBEDTLS_ECP_C

Enable support for SECP256R1 Elliptic Curve.

CONFIG_MBEDTLS_ECP_DP_SECP384R1_ENABLED

Enable SECP384R1 curve

Found in: Component config > mbedTLS > CONFIG_MBEDTLS_ECP_C

Enable support for SECP384R1 Elliptic Curve.

CONFIG_MBEDTLS_ECP_DP_SECP521R1_ENABLED

Enable SECP521R1 curve

Found in: Component config > mbedTLS > CONFIG_MBEDTLS_ECP_C

Enable support for SECP521R1 Elliptic Curve.

CONFIG_MBEDTLS_ECP_DP_SECP192K1_ENABLED

Enable SECP192K1 curve

Found in: Component config > mbedTLS > CONFIG_MBEDTLS_ECP_C

Enable support for SECP192K1 Elliptic Curve.

CONFIG_MBEDTLS_ECP_DP_SECP224K1_ENABLED

Enable SECP224K1 curve

Found in: Component config > mbedTLS > CONFIG_MBEDTLS_ECP_C

Enable support for SECP224K1 Elliptic Curve.

CONFIG_MBEDTLS_ECP_DP_SECP256K1_ENABLED

Enable SECP256K1 curve

Found in: Component config > mbedTLS > CONFIG_MBEDTLS_ECP_C

Enable support for SECP256K1 Elliptic Curve.

CONFIG_MBEDTLS_ECP_DP_BP256R1_ENABLED

Enable BP256R1 curve

Found in: Component config > mbedTLS > CONFIG_MBEDTLS_ECP_C

support for DP Elliptic Curve.

CONFIG_MBEDTLS_ECP_DP_BP384R1_ENABLED

Enable BP384R1 curve

Found in: Component config > mbedTLS > CONFIG_MBEDTLS_ECP_C

support for DP Elliptic Curve.

CONFIG_MBEDTLS_ECP_DP_BP512R1_ENABLED

Enable BP512R1 curve

Found in: Component config > mbedTLS > CONFIG_MBEDTLS_ECP_C

support for DP Elliptic Curve.

CONFIG_MBEDTLS_ECP_DP_CURVE25519_ENABLED

Enable CURVE25519 curve

Found in: Component config > mbedTLS > CONFIG_MBEDTLS_ECP_C

Enable support for CURVE25519 Elliptic Curve.

CONFIG_MBEDTLS_ECP_NIST_OPTIM

NIST ‘modulo p’ optimisations

Found in: Component config > mbedTLS > CONFIG_MBEDTLS_ECP_C

NIST ‘modulo p’ optimisations increase Elliptic Curve operation performance.

Disabling this option saves some code size.

end of Elliptic Curve options

CONFIG_MBEDTLS_SECURITY_RISKS

Show configurations with potential security risks

Found in: Component config > mbedTLS

Contains:

- *CONFIG_MBEDTLS_ALLOW_UNSUPPORTED_CRITICAL_EXT*

CONFIG_MBEDTLS_ALLOW_UNSUPPORTED_CRITICAL_EXT

X.509 CRT parsing with unsupported critical extensions

Found in: Component config > mbedTLS > CONFIG_MBEDTLS_SECURITY_RISKS

Allow the X.509 certificate parser to load certificates with unsupported critical extensions

mDNS

Contains:

- *CONFIG_MDNS_MAX_SERVICES*
- *CONFIG_MDNS_TASK_PRIORITY*
- *CONFIG_MDNS_TASK_AFFINITY*
- *CONFIG_MDNS_SERVICE_ADD_TIMEOUT_MS*
- *CONFIG_MDNS_TIMER_PERIOD_MS*

CONFIG_MDNS_MAX_SERVICES

Max number of services

Found in: Component config > mDNS

Services take up a certain amount of memory, and allowing fewer services to be open at the same time conserves memory. Specify the maximum amount of services here. The valid value is from 1 to 64.

CONFIG_MDNS_TASK_PRIORITY

mDNS task priority

Found in: Component config > mDNS

Allows setting mDNS task priority. Please do not set the task priority higher than priorities of system tasks. Compile time warning/error would be emitted if the chosen task priority were too high.

CONFIG_MDNS_TASK_AFFINITY

mDNS task affinity

Found in: Component config > mDNS

Allows setting mDNS tasks affinity, i.e. whether the task is pinned to CPU0, pinned to CPU1, or allowed to run on any CPU.

Available options:

- No affinity (MDNS_TASK_AFFINITY_NO_AFFINITY)
- CPU0 (MDNS_TASK_AFFINITY_CPU0)
- CPU1 (MDNS_TASK_AFFINITY_CPU1)

CONFIG_MDNS_SERVICE_ADD_TIMEOUT_MS

mDNS adding service timeout (ms)

Found in: Component config > mDNS

Configures timeout for adding a new mDNS service. Adding a service fails if could not be completed within this time.

CONFIG_MDNS_TIMER_PERIOD_MS

mDNS timer period (ms)

Found in: Component config > mDNS

Configures period of mDNS timer, which periodically transmits packets and schedules mDNS searches.

ESP-MQTT Configurations

Contains:

- *CONFIG_MQTT_PROTOCOL_311*
- *CONFIG_MQTT_TRANSPORT_SSL*
- *CONFIG_MQTT_TRANSPORT_WEBSOCKET*
- *CONFIG_MQTT_USE_CUSTOM_CONFIG*
- *CONFIG_MQTT_TASK_CORE_SELECTION_ENABLED*
- *CONFIG_MQTT_CUSTOM_OUTBOX*

CONFIG_MQTT_PROTOCOL_311

Enable MQTT protocol 3.1.1

Found in: Component config > ESP-MQTT Configurations

If not, this library will use MQTT protocol 3.1

CONFIG_MQTT_TRANSPORT_SSL

Enable MQTT over SSL

Found in: Component config > ESP-MQTT Configurations

Enable MQTT transport over SSL with mbedtls

CONFIG_MQTT_TRANSPORT_WEBSOCKET

Enable MQTT over Websocket

Found in: Component config > ESP-MQTT Configurations

Enable MQTT transport over Websocket.

CONFIG_MQTT_TRANSPORT_WEBSOCKET_SECURE

Enable MQTT over Websocket Secure

Found in: Component config > ESP-MQTT Configurations > CONFIG_MQTT_TRANSPORT_WEBSOCKET

Enable MQTT transport over Websocket Secure.

CONFIG_MQTT_USE_CUSTOM_CONFIG

MQTT Using custom configurations

Found in: Component config > ESP-MQTT Configurations

Custom MQTT configurations.

CONFIG_MQTT_TCP_DEFAULT_PORT

Default MQTT over TCP port

Found in: Component config > ESP-MQTT Configurations > CONFIG_MQTT_USE_CUSTOM_CONFIG

Default MQTT over TCP port

CONFIG_MQTT_SSL_DEFAULT_PORT

Default MQTT over SSL port

Found in: Component config > ESP-MQTT Configurations > CONFIG_MQTT_USE_CUSTOM_CONFIG

Default MQTT over SSL port

CONFIG_MQTT_WS_DEFAULT_PORT

Default MQTT over Websocket port

Found in: Component config > ESP-MQTT Configurations > CONFIG_MQTT_USE_CUSTOM_CONFIG

Default MQTT over Websocket port

CONFIG_MQTT_WSS_DEFAULT_PORT

Default MQTT over Websocket Secure port

Found in: Component config > ESP-MQTT Configurations > CONFIG_MQTT_USE_CUSTOM_CONFIG

Default MQTT over Websocket Secure port

CONFIG_MQTT_BUFFER_SIZE

Default MQTT Buffer Size

Found in: Component config > ESP-MQTT Configurations > CONFIG_MQTT_USE_CUSTOM_CONFIG

This buffer size using for both transmit and receive

CONFIG_MQTT_TASK_STACK_SIZE

MQTT task stack size

Found in: Component config > ESP-MQTT Configurations > CONFIG_MQTT_USE_CUSTOM_CONFIG

MQTT task stack size

CONFIG_MQTT_DISABLE_API_LOCKS

Disable API locks

Found in: Component config > ESP-MQTT Configurations > CONFIG_MQTT_USE_CUSTOM_CONFIG

Default config employs API locks to protect internal structures. It is possible to disable these locks if the user code doesn't access MQTT API from multiple concurrent tasks

CONFIG_MQTT_TASK_PRIORITY

MQTT task priority

Found in: Component config > ESP-MQTT Configurations > CONFIG_MQTT_USE_CUSTOM_CONFIG

MQTT task priority. Higher number denotes higher priority.

CONFIG_MQTT_TASK_CORE_SELECTION_ENABLED

Enable MQTT task core selection

Found in: Component config > ESP-MQTT Configurations

This will enable core selection

CONFIG_MQTT_TASK_CORE_SELECTION

Core to use ?

Found in: Component config > ESP-MQTT Configurations > CONFIG_MQTT_TASK_CORE_SELECTION_ENABLED

Available options:

- Core 0 (MQTT_USE_CORE_0)
- Core 1 (MQTT_USE_CORE_1)

CONFIG_MQTT_CUSTOM_OUTBOX

Enable custom outbox implementation

Found in: Component config > ESP-MQTT Configurations

Set to true if a specific implementation of message outbox is needed (e.g. persistant outbox in NVM or similar).

Newlib

Contains:

- *CONFIG_NEWLIB_STDOUT_LINE_ENDING*
- *CONFIG_NEWLIB_STDIN_LINE_ENDING*
- *CONFIG_NEWLIB_NANO_FORMAT*

CONFIG_NEWLIB_STDOUT_LINE_ENDING

Line ending for UART output

Found in: Component config > Newlib

This option allows configuring the desired line endings sent to UART when a newline (‘n’ , LF) appears on stdout. Three options are possible:

CRLF: whenever LF is encountered, prepend it with CR

LF: no modification is applied, stdout is sent as is

CR: each occurence of LF is replaced with CR

This option doesn’ t affect behavior of the UART driver (drivers/uart.h).

Available options:

- CRLF (NEWLIB_STDOUT_LINE_ENDING_CRLF)
- LF (NEWLIB_STDOUT_LINE_ENDING_LF)
- CR (NEWLIB_STDOUT_LINE_ENDING_CR)

CONFIG_NEWLIB_STDIN_LINE_ENDING

Line ending for UART input

Found in: Component config > Newlib

This option allows configuring which input sequence on UART produces a newline ('n', LF) on stdin. Three options are possible:

CRLF: CRLF is converted to LF

LF: no modification is applied, input is sent to stdin as is

CR: each occurence of CR is replaced with LF

This option doesn't affect behavior of the UART driver (drivers/uart.h).

Available options:

- CRLF (NEWLIB_STDIN_LINE_ENDING_CRLF)
- LF (NEWLIB_STDIN_LINE_ENDING_LF)
- CR (NEWLIB_STDIN_LINE_ENDING_CR)

CONFIG_NEWLIB_NANO_FORMAT

Enable 'nano' formatting options for printf/scanf family

Found in: Component config > Newlib

ESP32 ROM contains parts of newlib C library, including printf/scanf family of functions. These functions have been compiled with so-called "nano" formatting option. This option doesn't support 64-bit integer formats and C99 features, such as positional arguments.

For more details about "nano" formatting option, please see newlib readme file, search for '–enable-newlib-nano-formatted-io': https://sourceware.org/newlib/README

If this option is enabled, build system will use functions available in ROM, reducing the application binary size. Functions available in ROM run faster than functions which run from flash. Functions available in ROM can also run when flash instruction cache is disabled.

If you need 64-bit integer formatting support or C99 features, keep this option disabled.

NVS

Contains:

- *CONFIG_NVS_ENCRYPTION*

CONFIG_NVS_ENCRYPTION

Enable NVS encryption

Found in: Component config > NVS

This option enables encryption for NVS. When enabled, AES-XTS is used to encrypt the complete NVS data, except the page headers. It requires XTS encryption keys to be stored in an encrypted partition. This means enabling flash encryption is a pre-requisite for this feature.

OpenSSL

Contains:

- *CONFIG_OPENSSL_DEBUG*
- *CONFIG_OPENSSL_ASSERT*

CONFIG_OPENSSL_DEBUG

Enable OpenSSL debugging

Found in: Component config > OpenSSL

Enable OpenSSL debugging function.

If the option is enabled, "SSL_DEBUG" works.

CONFIG_OPENSSL_DEBUG_LEVEL

OpenSSL debugging level

Found in: Component config > OpenSSL > CONFIG_OPENSSL_DEBUG

OpenSSL debugging level.

Only function whose debugging level is higher than "OPENSSL_DEBUG_LEVEL" works.

For example: If OPENSSL_DEBUG_LEVEL = 2, you use function "SSL_DEBUG(1, "malloc failed")". Because $1 < 2$, it will not print.

CONFIG_OPENSSL_LOWLEVEL_DEBUG

Enable OpenSSL low-level module debugging

Found in: Component config > OpenSSL > CONFIG_OPENSSL_DEBUG

If the option is enabled, low-level module debugging function of OpenSSL is enabled, e.g. mbedtls internal debugging function.

CONFIG_OPENSSL_ASSERT

Select OpenSSL assert function

Found in: Component config > OpenSSL

OpenSSL function needs "assert" function to check if input parameters are valid.

If you want to use assert debugging function, "OPENSSL_DEBUG" should be enabled.

Available options:

- Do nothing (OPENSSL_ASSERT_DO_NOTHING)

  Do nothing and "SSL_ASSERT" does not work.

- Check and exit (OPENSSL_ASSERT_EXIT)

  Enable assert exiting, it will check and return error code.

- Show debugging message (OPENSSL_ASSERT_DEBUG)

  Enable assert debugging, it will check and show debugging message.

- Show debugging message and exit (OPENSSL_ASSERT_DEBUG_EXIT)

  Enable assert debugging and exiting, it will check, show debugging message and return error code.

- Show debugging message and block (OPENSSL_ASSERT_DEBUG_BLOCK)

  Enable assert debugging and blocking, it will check, show debugging message and block by "while (1);" .

PThreads

Contains:

- *CONFIG_PTHREAD_TASK_PRIO_DEFAULT*
- *CONFIG_PTHREAD_TASK_STACK_SIZE_DEFAULT*
- *CONFIG_PTHREAD_STACK_MIN*

- *CONFIG_PTHREAD_TASK_CORE_DEFAULT*
- *CONFIG_PTHREAD_TASK_NAME_DEFAULT*

CONFIG_PTHREAD_TASK_PRIO_DEFAULT

Default task priority

Found in: Component config > PThreads

Priority used to create new tasks with default pthread parameters.

CONFIG_PTHREAD_TASK_STACK_SIZE_DEFAULT

Default task stack size

Found in: Component config > PThreads

Stack size used to create new tasks with default pthread parameters.

CONFIG_PTHREAD_STACK_MIN

Minimum allowed pthread stack size

Found in: Component config > PThreads

Minimum allowed pthread stack size set in attributes passed to pthread_create

CONFIG_PTHREAD_TASK_CORE_DEFAULT

Default pthread core affinity

Found in: Component config > PThreads

The default core to which pthreads are pinned.

Available options:

- No affinity (PTHREAD_DEFAULT_CORE_NO_AFFINITY)
- Core 0 (PTHREAD_DEFAULT_CORE_0)
- Core 1 (PTHREAD_DEFAULT_CORE_1)

CONFIG_PTHREAD_TASK_NAME_DEFAULT

Default name of pthreads

Found in: Component config > PThreads

The default name of pthreads.

SPI Flash driver

Contains:

- *CONFIG_SPI_FLASH_VERIFY_WRITE*
- *CONFIG_SPI_FLASH_ENABLE_COUNTERS*
- *CONFIG_SPI_FLASH_ROM_DRIVER_PATCH*
- *CONFIG_SPI_FLASH_DANGEROUS_WRITE*
- *CONFIG_SPI_FLASH_USE_LEGACY_IMPL*
- *CONFIG_SPI_FLASH_BYPASS_BLOCK_ERASE*
- *CONFIG_SPI_FLASH_YIELD_DURING_ERASE*
- *Auto-detect flash chips*

CONFIG_SPI_FLASH_VERIFY_WRITE

Verify SPI flash writes

Found in: Component config > SPI Flash driver

If this option is enabled, any time SPI flash is written then the data will be read back and verified. This can catch hardware problems with SPI flash, or flash which was not erased before verification.

CONFIG_SPI_FLASH_LOG_FAILED_WRITE

Log errors if verification fails

Found in: Component config > SPI Flash driver > CONFIG_SPI_FLASH_VERIFY_WRITE

If this option is enabled, if SPI flash write verification fails then a log error line will be written with the address, expected & actual values. This can be useful when debugging hardware SPI flash problems.

CONFIG_SPI_FLASH_WARN_SETTING_ZERO_TO_ONE

Log warning if writing zero bits to ones

Found in: Component config > SPI Flash driver > CONFIG_SPI_FLASH_VERIFY_WRITE

If this option is enabled, any SPI flash write which tries to set zero bits in the flash to ones will log a warning. Such writes will not result in the requested data appearing identically in flash once written, as SPI NOR flash can only set bits to one when an entire sector is erased. After erasing, individual bits can only be written from one to zero.

Note that some software (such as SPIFFS) which is aware of SPI NOR flash may write one bits as an optimisation, relying on the data in flash becoming a bitwise AND of the new data and any existing data. Such software will log spurious warnings if this option is enabled.

CONFIG_SPI_FLASH_ENABLE_COUNTERS

Enable operation counters

Found in: Component config > SPI Flash driver

This option enables the following APIs:

- spi_flash_reset_counters
- spi_flash_dump_counters
- spi_flash_get_counters

These APIs may be used to collect performance data for spi_flash APIs and to help understand behaviour of libraries which use SPI flash.

CONFIG_SPI_FLASH_ROM_DRIVER_PATCH

Enable SPI flash ROM driver patched functions

Found in: Component config > SPI Flash driver

Enable this flag to use patched versions of SPI flash ROM driver functions. This option is needed to write to flash on ESP32-D2WD, and any configuration where external SPI flash is connected to non-default pins.

CONFIG_SPI_FLASH_DANGEROUS_WRITE

Writing to dangerous flash regions

Found in: Component config > SPI Flash driver

SPI flash APIs can optionally abort or return a failure code if erasing or writing addresses that fall at the beginning of flash (covering the bootloader and partition table) or that overlap the app partition that contains the running app.

It is not recommended to ever write to these regions from an IDF app, and this check prevents logic errors or corrupted firmware memory from damaging these regions.

Note that this feature *does not* check calls to the esp_rom_xxx SPI flash ROM functions. These functions should not be called directly from IDF applications.

Available options:

- Aborts (SPI_FLASH_DANGEROUS_WRITE_ABORTS)
- Fails (SPI_FLASH_DANGEROUS_WRITE_FAILS)
- Allowed (SPI_FLASH_DANGEROUS_WRITE_ALLOWED)

CONFIG_SPI_FLASH_USE_LEGACY_IMPL

Use the legacy implementation before IDF v4.0

Found in: Component config > SPI Flash driver

The implementation of SPI flash has been greatly changed in IDF v4.0. Enable this option to use the legacy implementation.

CONFIG_SPI_FLASH_BYPASS_BLOCK_ERASE

Bypass a block erase and always do sector erase

Found in: Component config > SPI Flash driver

Some flash chips can have very high "max" erase times, especially for block erase (32KB or 64KB). This option allows to bypass "block erase" and always do sector erase commands. This will be much slower overall in most cases, but improves latency for other code to run.

CONFIG_SPI_FLASH_YIELD_DURING_ERASE

Enables yield operation during flash erase

Found in: Component config > SPI Flash driver

This allows to yield the CPUs between erase commands. Prevents starvation of other tasks.

CONFIG_SPI_FLASH_ERASE_YIELD_DURATION_MS

Duration of erasing to yield CPUs (ms)

Found in: Component config > SPI Flash driver > CONFIG_SPI_FLASH_YIELD_DURING_ERASE

If a duration of one erase command is large then it will yield CPUs after finishing a current command.

CONFIG_SPI_FLASH_ERASE_YIELD_TICKS

CPU release time (tick)

Found in: Component config > SPI Flash driver > CONFIG_SPI_FLASH_YIELD_DURING_ERASE

Defines how many ticks will be before returning to continue a erasing.

Auto-detect flash chips

Contains:

- *CONFIG_SPI_FLASH_SUPPORT_ISSI_CHIP*
- *CONFIG_SPI_FLASH_SUPPORT_GD_CHIP*

CONFIG_SPI_FLASH_SUPPORT_ISSI_CHIP

ISSI

Found in: Component config > SPI Flash driver > Auto-detect flash chips

Enable this to support auto detection of ISSI chips if chip vendor not directly given by `chip\_drv` member of the chip struct. This adds support for variant chips, however will extend detecting time.

CONFIG_SPI_FLASH_SUPPORT_GD_CHIP

GigaDevice

Found in: Component config > SPI Flash driver > Auto-detect flash chips

Enable this to support auto detection of GD (GigaDevice) chips if chip vendor not directly given by `chip\_drv` member of the chip struct. If you are using Wrover modules, please don't disable this, otherwise your flash may not work in 4-bit mode.

This adds support for variant chips, however will extend detecting time and image size. Note that the default chip driver supports the GD chips with product ID 60H.

SPIFFS Configuration

Contains:

- *CONFIG_SPIFFS_MAX_PARTITIONS*
- *SPIFFS Cache Configuration*

- *CONFIG_SPIFFS_PAGE_CHECK*
- *CONFIG_SPIFFS_GC_MAX_RUNS*
- *CONFIG_SPIFFS_GC_STATS*
- *CONFIG_SPIFFS_PAGE_SIZE*
- *CONFIG_SPIFFS_OBJ_NAME_LEN*
- *CONFIG_SPIFFS_FOLLOW_SYMLINKS*
- *CONFIG_SPIFFS_USE_MAGIC*
- *CONFIG_SPIFFS_META_LENGTH*
- *CONFIG_SPIFFS_USE_MTIME*
- *CONFIG_SPIFFS_MTIME_WIDE_64_BITS*
- *Debug Configuration*

CONFIG_SPIFFS_MAX_PARTITIONS

Maximum Number of Partitions

Found in: Component config > SPIFFS Configuration

Define maximum number of partitions that can be mounted.

SPIFFS Cache Configuration

Contains:

- *CONFIG_SPIFFS_CACHE*

CONFIG_SPIFFS_CACHE

Enable SPIFFS Cache

Found in: Component config > SPIFFS Configuration > SPIFFS Cache Configuration

Enables/disable memory read caching of nucleus file system operations.

CONFIG_SPIFFS_CACHE_WR

Enable SPIFFS Write Caching

Found in: Component config > SPIFFS Configuration > SPIFFS Cache Configuration > CONFIG_SPIFFS_CACHE

Enables memory write caching for file descriptors in hydrogen.

CONFIG_SPIFFS_CACHE_STATS

Enable SPIFFS Cache Statistics

Found in: Component config > SPIFFS Configuration > SPIFFS Cache Configuration > CONFIG_SPIFFS_CACHE

Enable/disable statistics on caching. Debug/test purpose only.

CONFIG_SPIFFS_PAGE_CHECK

Enable SPIFFS Page Check

Found in: Component config > SPIFFS Configuration

Always check header of each accessed page to ensure consistent state. If enabled it will increase number of reads from flash, especially if cache is disabled.

CONFIG_SPIFFS_GC_MAX_RUNS

Set Maximum GC Runs

Found in: Component config > SPIFFS Configuration

Define maximum number of GC runs to perform to reach desired free pages.

CONFIG_SPIFFS_GC_STATS

Enable SPIFFS GC Statistics

Found in: Component config > SPIFFS Configuration

Enable/disable statistics on gc. Debug/test purpose only.

CONFIG_SPIFFS_PAGE_SIZE

SPIFFS logical page size

Found in: Component config > SPIFFS Configuration

Logical page size of SPIFFS partition, in bytes. Must be multiple of flash page size (which is usually 256 bytes). Larger page sizes reduce overhead when storing large files, and improve filesystem performance when reading large files. Smaller page sizes reduce overhead when storing small (< page size) files.

CONFIG_SPIFFS_OBJ_NAME_LEN

Set SPIFFS Maximum Name Length

Found in: Component config > SPIFFS Configuration

Object name maximum length. Note that this length include the zero-termination character, meaning maximum string of characters can at most be SPIFFS_OBJ_NAME_LEN - 1.

SPIFFS_OBJ_NAME_LEN + SPIFFS_META_LENGTH should not exceed SPIFFS_PAGE_SIZE - 64.

CONFIG_SPIFFS_FOLLOW_SYMLINKS

Enable symbolic links for image creation

Found in: Component config > SPIFFS Configuration

If this option is enabled, symbolic links are taken into account during partition image creation.

CONFIG_SPIFFS_USE_MAGIC

Enable SPIFFS Filesystem Magic

Found in: Component config > SPIFFS Configuration

Enable this to have an identifiable spiffs filesystem. This will look for a magic in all sectors to determine if this is a valid spiffs system or not at mount time.

CONFIG_SPIFFS_USE_MAGIC_LENGTH

Enable SPIFFS Filesystem Length Magic

Found in: Component config > SPIFFS Configuration > CONFIG_SPIFFS_USE_MAGIC

If this option is enabled, the magic will also be dependent on the length of the filesystem. For example, a filesystem configured and formatted for 4 megabytes will not be accepted for mounting with a configuration defining the filesystem as 2 megabytes.

CONFIG_SPIFFS_META_LENGTH

Size of per-file metadata field

Found in: Component config > SPIFFS Configuration

This option sets the number of extra bytes stored in the file header. These bytes can be used in an application-specific manner. Set this to at least 4 bytes to enable support for saving file modification time.

SPIFFS_OBJ_NAME_LEN + SPIFFS_META_LENGTH should not exceed SPIFFS_PAGE_SIZE - 64.

CONFIG_SPIFFS_USE_MTIME

Save file modification time

Found in: Component config > SPIFFS Configuration

If enabled, then the first 4 bytes of per-file metadata will be used to store file modification time (mtime), accessible through stat/fstat functions. Modification time is updated when the file is opened.

CONFIG_SPIFFS_MTIME_WIDE_64_BITS

The time field occupies 64 bits in the image instead of 32 bits

Found in: Component config > SPIFFS Configuration

If this option is not set, the time field is 32 bits (up to 2106 year), otherwise it is 64 bits and make sure it matches SPIFFS_META_LENGTH. If the chip already has the spiffs image with the time field = 32 bits then this option cannot be applied in this case. Erase it first before using this option. To resolve the Y2K38 problem for the spiffs, use a toolchain with support time_t 64 bits (see SDK_TOOLCHAIN_SUPPORTS_TIME_WIDE_64_BITS).

Debug Configuration

Contains:

- *CONFIG_SPIFFS_DBG*
- *CONFIG_SPIFFS_API_DBG*
- *CONFIG_SPIFFS_GC_DBG*
- *CONFIG_SPIFFS_CACHE_DBG*
- *CONFIG_SPIFFS_CHECK_DBG*
- *CONFIG_SPIFFS_TEST_VISUALISATION*

CONFIG_SPIFFS_DBG

Enable general SPIFFS debug

Found in: Component config > SPIFFS Configuration > Debug Configuration

Enabling this option will print general debug mesages to the console.

CONFIG_SPIFFS_API_DBG

Enable SPIFFS API debug

Found in: Component config > SPIFFS Configuration > Debug Configuration

Enabling this option will print API debug mesages to the console.

CONFIG_SPIFFS_GC_DBG

Enable SPIFFS Garbage Cleaner debug

Found in: Component config > SPIFFS Configuration > Debug Configuration

Enabling this option will print GC debug mesages to the console.

CONFIG_SPIFFS_CACHE_DBG

Enable SPIFFS Cache debug

Found in: Component config > SPIFFS Configuration > Debug Configuration

Enabling this option will print cache debug mesages to the console.

CONFIG_SPIFFS_CHECK_DBG

Enable SPIFFS Filesystem Check debug

Found in: Component config > SPIFFS Configuration > Debug Configuration

Enabling this option will print Filesystem Check debug mesages to the console.

CONFIG_SPIFFS_TEST_VISUALISATION

Enable SPIFFS Filesystem Visualization

Found in: Component config > SPIFFS Configuration > Debug Configuration

Enable this option to enable SPIFFS_vis function in the API.

Unity unit testing library

Contains:

- *CONFIG_UNITY_ENABLE_FLOAT*
- *CONFIG_UNITY_ENABLE_DOUBLE*
- *CONFIG_UNITY_ENABLE_COLOR*
- *CONFIG_UNITY_ENABLE_IDF_TEST_RUNNER*
- *CONFIG_UNITY_ENABLE_FIXTURE*
- *CONFIG_UNITY_ENABLE_BACKTRACE_ON_FAIL*

CONFIG_UNITY_ENABLE_FLOAT

Support for float type

Found in: Component config > Unity unit testing library

If not set, assertions on float arguments will not be available.

CONFIG_UNITY_ENABLE_DOUBLE

Support for double type

Found in: Component config > Unity unit testing library

If not set, assertions on double arguments will not be available.

CONFIG_UNITY_ENABLE_COLOR

Colorize test output

Found in: Component config > Unity unit testing library

If set, Unity will colorize test results using console escape sequences.

CONFIG_UNITY_ENABLE_IDF_TEST_RUNNER

Include ESP-IDF test registration/running helpers

Found in: Component config > Unity unit testing library

If set, then the following features will be available:

- TEST_CASE macro which performs automatic registration of test functions

- Functions to run registered test functions: unity_run_all_tests, unity_run_tests_with_filter, unity_run_single_test_by_name.
- Interactive menu which lists test cases and allows choosing the tests to be run, available via unity_run_menu function.

Disable if a different test registration mechanism is used.

CONFIG_UNITY_ENABLE_FIXTURE

Include Unity test fixture

Found in: Component config > Unity unit testing library

If set, unity_fixture.h header file and associated source files are part of the build. These provide an optional set of macros and functions to implement test groups.

CONFIG_UNITY_ENABLE_BACKTRACE_ON_FAIL

Print a backtrace when a unit test fails

Found in: Component config > Unity unit testing library

If set, the unity framework will print the backtrace information before jumping back to the test menu. The jumping is usually occurs in assert functions such as TEST_ASSERT, TEST_FAIL etc.

Virtual file system

Contains:

- *CONFIG_VFS_SUPPRESS_SELECT_DEBUG_OUTPUT*
- *CONFIG_VFS_SUPPORT_TERMIOS*
- *Host File System I/O (Semihosting)*

CONFIG_VFS_SUPPRESS_SELECT_DEBUG_OUTPUT

Suppress select() related debug outputs

Found in: Component config > Virtual file system

Select() related functions might produce an unconveniently lot of debug outputs when one sets the default log level to DEBUG or higher. It is possible to suppress these debug outputs by enabling this option.

CONFIG_VFS_SUPPORT_TERMIOS

Add support for termios.h

Found in: Component config > Virtual file system

Disabling this option can save memory when the support for termios.h is not required.

Host File System I/O (Semihosting)

Contains:

- *CONFIG_SEMIHOSTFS_MAX_MOUNT_POINTS*
- *CONFIG_SEMIHOSTFS_HOST_PATH_MAX_LEN*

CONFIG_SEMIHOSTFS_MAX_MOUNT_POINTS

Maximum number of the host filesystem mount points

Found in: Component config > Virtual file system > Host File System I/O (Semihosting)

Define maximum number of host filesystem mount points.

CONFIG_SEMIHOSTFS_HOST_PATH_MAX_LEN

Maximum path length for the host base directory

Found in: Component config > Virtual file system > Host File System I/O (Semihosting)

Define maximum path length for the host base directory which is to be mounted. If host path passed to esp_vfs_semihost_register() is longer than this value it will be truncated.

Wear Levelling

Contains:

- *CONFIG_WL_SECTOR_SIZE*
- *CONFIG_WL_SECTOR_MODE*

CONFIG_WL_SECTOR_SIZE

Wear Levelling library sector size

Found in: Component config > Wear Levelling

Sector size used by wear levelling library. You can set default sector size or size that will fit to the flash device sector size.

With sector size set to 4096 bytes, wear levelling library is more efficient. However if FAT filesystem is used on top of wear levelling library, it will need more temporary storage: 4096 bytes for each mounted filesystem and 4096 bytes for each opened file.

With sector size set to 512 bytes, wear levelling library will perform more operations with flash memory, but less RAM will be used by FAT filesystem library (512 bytes for the filesystem and 512 bytes for each file opened).

Available options:

- 512 (WL_SECTOR_SIZE_512)
- 4096 (WL_SECTOR_SIZE_4096)

CONFIG_WL_SECTOR_MODE

Sector store mode

Found in: Component config > Wear Levelling

Specify the mode to store data into flash:

- In Performance mode a data will be stored to the RAM and then stored back to the flash. Compared to the Safety mode, this operation is faster, but if power will be lost when erase sector operation is in progress, then the data from complete flash device sector will be lost.
- In Safety mode data from complete flash device sector will be read from flash, modified, and then stored back to flash. Compared to the Performance mode, this operation is slower, but if power is lost during erase sector operation, then the data from full flash device sector will not be lost.

Available options:

- Perfomance (WL_SECTOR_MODE_PERF)
- Safety (WL_SECTOR_MODE_SAFE)

Wi-Fi Provisioning Manager

Contains:

- *CONFIG_WIFI_PROV_SCAN_MAX_ENTRIES*
- *CONFIG_WIFI_PROV_AUTOSTOP_TIMEOUT*

CONFIG_WIFI_PROV_SCAN_MAX_ENTRIES

Max Wi-Fi Scan Result Entries

Found in: *Component config* > *Wi-Fi Provisioning Manager*

This sets the maximum number of entries of Wi-Fi scan results that will be kept by the provisioning manager

CONFIG_WIFI_PROV_AUTOSTOP_TIMEOUT

Provisioning auto-stop timeout

Found in: *Component config* > *Wi-Fi Provisioning Manager*

Time (in seconds) after which the Wi-Fi provisioning manager will auto-stop after connecting to a Wi-Fi network successfully.

Supplicant

Contains:

- *CONFIG_WPA_MBEDTLS_CRYPTO*
- *CONFIG_WPA_TLS_V12*

CONFIG_WPA_MBEDTLS_CRYPTO

Use MbedTLS crypto API' s

Found in: *Component config* > *Supplicant*

Select this option to use MbedTLS crypto API' s which utilize hardware acceleration.

CONFIG_WPA_TLS_V12

Enable TLS v1.2

Found in: *Component config* > *Supplicant*

Select this to enable TLS v1.2 for WPA2-Enterprise Authentication.

Compatibility options

Contains:

- *CONFIG_LEGACY_INCLUDE_COMMON_HEADERS*

CONFIG_LEGACY_INCLUDE_COMMON_HEADERS

Include headers across components as before IDF v4.0

Found in: Compatibility options

Soc, esp32, and driver components, the most common components. Some header of these components are included implicitly by headers of other components before IDF v4.0. It' s not required for high-level components, but still included through long header chain everywhere.

This is harmful to the modularity. So it' s changed in IDF v4.0.

You can still include these headers in a legacy way until it is totally deprecated by enable this option.

Deprecated options and their replacements

- CONFIG_A2DP_ENABLE (*CONFIG_BT_A2DP_ENABLE*)
- CONFIG_ADC2_DISABLE_DAC (*CONFIG_ADC_DISABLE_DAC*)
- CONFIG_APP_ANTI_ROLLBACK (*CONFIG_BOOTLOADER_APP_ANTI_ROLLBACK*)
- CONFIG_APP_ROLLBACK_ENABLE (*CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE*)
- CONFIG_APP_SECURE_VERSION (*CONFIG_BOOTLOADER_APP_SECURE_VERSION*)
- CONFIG_APP_SECURE_VERSION_SIZE_EFUSE_FIELD (*CONFIG_BOOTLOADER_APP_SEC_VER_SIZE_EFUSE_FIELD*)
- CONFIG_BLE_ACTIVE_SCAN_REPORT_ADV_SCAN_RSP_INDIVIDUALLY (*CONFIG_BT_BLE_ACT_SCAN_REP_ADV_SCAN*)
- CONFIG_BLE_ADV_REPORT_DISCARD_THRSHOLD (*CONFIG_BTDM_BLE_ADV_REPORT_DISCARD_*
- CONFIG_BLE_ADV_REPORT_FLOW_CONTROL_NUM (*CONFIG_BTDM_BLE_ADV_REPORT_FLOW_CTRL_NUM*)
- CONFIG_BLE_ADV_REPORT_FLOW_CONTROL_SUPPORTED (*CONFIG_BTDM_BLE_ADV_REPORT_FLOW_CTRL_SUPP*)
- CONFIG_BLE_ESTABLISH_LINK_CONNECTION_TIMEOUT (*CONFIG_BT_BLE_ESTAB_LINK_CONN_TOUT*)
- CONFIG_BLE_HOST_QUEUE_CONGESTION_CHECK (*CONFIG_BT_BLE_HOST_QUEUE_CONG_CHEC*
- CONFIG_BLE_MESH_GATT_PROXY (*CONFIG_BLE_MESH_GATT_PROXY_SERVER*)
- CONFIG_BLE_MESH_SCAN_DUPLICATE_EN (*CONFIG_BTDM_BLE_MESH_SCAN_DUPL_EN*)
- CONFIG_BLE_SCAN_DUPLICATE (*CONFIG_BTDM_BLE_SCAN_DUPL*)
- CONFIG_BLE_SMP_ENABLE (*CONFIG_BT_BLE_SMP_ENABLE*)

- CONFIG_BLUEDROID_MEM_DEBUG (*CONFIG_BT_BLUEDROID_MEM_DEBUG*)
- **CONFIG_BLUEDROID_PINNED_TO_CORE_CHOICE (*CONFIG_BT_BLUEDROID_PINNED*)**
  - CONFIG_BLUEDROID_PINNED_TO_CORE_0
  - CONFIG_BLUEDROID_PINNED_TO_CORE_1
- CONFIG_BROWNOUT_DET (*CONFIG_ESP32_BROWNOUT_DET*)
- **CONFIG_BROWNOUT_DET_LVL_SEL (*CONFIG_ESP32_BROWNOUT_DET_LVL_SEL*)**
  - CONFIG_BROWNOUT_DET_LVL_SEL_0
  - CONFIG_BROWNOUT_DET_LVL_SEL_1
  - CONFIG_BROWNOUT_DET_LVL_SEL_2
  - CONFIG_BROWNOUT_DET_LVL_SEL_3
  - CONFIG_BROWNOUT_DET_LVL_SEL_4
  - CONFIG_BROWNOUT_DET_LVL_SEL_5
  - CONFIG_BROWNOUT_DET_LVL_SEL_6
  - CONFIG_BROWNOUT_DET_LVL_SEL_7
- CONFIG_BTC_TASK_STACK_SIZE (*CONFIG_BT_BTC_TASK_STACK_SIZE*)
- CONFIG_BTDM_CONTROLLER_BLE_MAX_CONN (*CONFIG_BTDM_CTRL_BLE_MAX_CONN*)
- CONFIG_BTDM_CONTROLLER_BR_EDR_MAX_ACL_CONN (*CONFIG_BTDM_CTRL_BR_EDR_MAX_ACL_CONN*)
- CONFIG_BTDM_CONTROLLER_BR_EDR_MAX_SYNC_CONN (*CONFIG_BTDM_CTRL_BR_EDR_MAX_SYNC_CONN*)
- CONFIG_BTDM_CONTROLLER_FULL_SCAN_SUPPORTED (*CONFIG_BTDM_CTRL_FULL_SCAN_SUPPORTED*)
- **CONFIG_BTDM_CONTROLLER_HCI_MODE_CHOICE (*CONFIG_BTDM_CTRL_HCI_MOD*)**
  - CONFIG_BTDM_CONTROLLER_HCI_MODE_VHCI
  - CONFIG_BTDM_CONTROLLER_HCI_MODE_UART_H4
- **CONFIG_BTDM_CONTROLLER_MODE (*CONFIG_BTDM_CTRL_MODE*)**
  - CONFIG_BTDM_CONTROLLER_MODE_BLE_ONLY
  - CONFIG_BTDM_CONTROLLER_MODE_BR_EDR_ONLY
  - CONFIG_BTDM_CONTROLLER_MODE_BTDM

- CONFIG_BTDM_CONTROLLER_MODEM_SLEEP (*CONFIG_BTDM_MODEM_SLEEP*)
- CONFIG_BTDM_CONTROLLER_PINNED_TO_CORE_CHOICE (*CONFIG_BTDM_CTRL_PINNED_TO_CORE_CHOICE*)
- CONFIG_BTU_TASK_STACK_SIZE (*CONFIG_BT_BTU_TASK_STACK_SIZE*)
- CONFIG_CLASSIC_BT_ENABLED (*CONFIG_BT_CLASSIC_ENABLED*)
- CONFIG_COMPATIBLE_PRE_V2_1_BOOTLOADERS (*CONFIG_ESP32_COMPATIBLE_PRE_V2_1_BOOT*
- **CONFIG_CONSOLE_UART (*CONFIG_ESP_CONSOLE_UART*)**
  - CONFIG_CONSOLE_UART_DEFAULT
  - CONFIG_CONSOLE_UART_CUSTOM
  - CONFIG_CONSOLE_UART_NONE
- CONFIG_CONSOLE_UART_BAUDRATE (*CONFIG_ESP_CONSOLE_UART_BAUDRATE*)
- CONFIG_CONSOLE_UART_RX_GPIO (*CONFIG_ESP_CONSOLE_UART_RX_GPIO*)
- CONFIG_CONSOLE_UART_TX_GPIO (*CONFIG_ESP_CONSOLE_UART_TX_GPIO*)
- CONFIG_CXX_EXCEPTIONS (*CONFIG_COMPILER_CXX_EXCEPTIONS*)
- CONFIG_CXX_EXCEPTIONS_EMG_POOL_SIZE (*CONFIG_COMPILER_CXX_EXCEPTIONS_EMG_POOL*
- CONFIG_DISABLE_BASIC_ROM_CONSOLE (*CONFIG_ESP32_DISABLE_BASIC_ROM_CONSOLE*)
- CONFIG_DISABLE_GCC8_WARNINGS (*CONFIG_COMPILER_DISABLE_GCC8_WARNINGS*)
- CONFIG_DUPLICATE_SCAN_CACHE_SIZE (*CONFIG_BTDM_SCAN_DUPL_CACHE_SIZE*)
- CONFIG_EFUSE_SECURE_VERSION_EMULATE (*CONFIG_BOOTLOADER_EFUSE_SECURE_VERSION_*
- CONFIG_ENABLE_STATIC_TASK_CLEAN_UP_HOOK (*CONFIG_FREERTOS_ENABLE_STATIC_TASK_*
- **CONFIG_ESP32_APPTRACE_DESTINATION (*CONFIG_APPTRACE_DESTINATION*)**
  - CONFIG_ESP32_APPTRACE_DEST_TRAX
  - CONFIG_ESP32_APPTRACE_DEST_NONE
- CONFIG_ESP32_APPTRACE_ONPANIC_HOST_FLUSH_TMO (*CONFIG_APPTRACE_ONPANIC_HOST_FLUSH_TMO*)
- CONFIG_ESP32_APPTRACE_PENDING_DATA_SIZE_MAX (*CONFIG_APPTRACE_PENDING_DATA_SIZE_MAX*)
- CONFIG_ESP32_APPTRACE_POSTMORTEM_FLUSH_TRAX_THRESH (*CONFIG_APPTRACE_POSTMORTEM_FLUSH_THRESH*)
- CONFIG_ESP32_GCOV_ENABLE (*CONFIG_APPTRACE_GCOV_ENABLE*)
- CONFIG_ESP32_PTHREAD_STACK_MIN (*CONFIG_PTHREAD_STACK_MIN*)

- CONFIG_ESP32_PTHREAD_TASK_NAME_DEFAULT (*CONFIG_PTHREAD_TASK_NAME_DEFAULT*)
- CONFIG_ESP32_PTHREAD_TASK_PRIO_DEFAULT (*CONFIG_PTHREAD_TASK_PRIO_DEFAULT*)
- CONFIG_ESP32_PTHREAD_TASK_STACK_SIZE_DEFAULT (*CONFIG_PTHREAD_TASK_STACK_SIZE_DEFAULT*)
- **CONFIG_ESP32_RTC_CLOCK_SOURCE (*CONFIG_ESP32_RTC_CLK_SRC*)**
  - CONFIG_ESP32_RTC_CLOCK_SOURCE_INTERNAL_RC
  - CONFIG_ESP32_RTC_CLOCK_SOURCE_EXTERNAL_CRYSTAL
  - CONFIG_ESP32_RTC_CLOCK_SOURCE_EXTERNAL_OSC
  - CONFIG_ESP32_RTC_CLOCK_SOURCE_INTERNAL_8MD256
- CONFIG_ESP32_RTC_EXTERNAL_CRYSTAL_ADDITIONAL_CURRENT (*CONFIG_ESP32_RTC_EXT_CRYST_ADDIT_CURRENT*)
- CONFIG_ESP_GRATUITOUS_ARP (*CONFIG_LWIP_ESP_GRATUITOUS_ARP*)
- CONFIG_ESP_TCP_KEEP_CONNECTION_WHEN_IP_CHANGES (*CONFIG_LWIP_TCP_KEEP_CONNECTION_WHEN_IP_CHANGES*)
- CONFIG_EVENT_LOOP_PROFILING (*CONFIG_ESP_EVENT_LOOP_PROFILING*)
- CONFIG_FLASH_ENCRYPTION_ENABLED (*CONFIG_SECURE_FLASH_ENC_ENABLED*)
- CONFIG_FLASH_ENCRYPTION_UART_BOOTLOADER_ALLOW_CACHE (*CONFIG_SECURE_FLASH_UART_BOOTLOADER_ALLOW_CACHE*)
- CONFIG_FLASH_ENCRYPTION_UART_BOOTLOADER_ALLOW_DECRYPT (*CONFIG_SECURE_FLASH_UART_BOOTLOADER_ALLOW_DEC*)
- CONFIG_FLASH_ENCRYPTION_UART_BOOTLOADER_ALLOW_ENCRYPT (*CONFIG_SECURE_FLASH_UART_BOOTLOADER_ALLOW_ENC*)
- CONFIG_GARP_TMR_INTERVAL (*CONFIG_LWIP_GARP_TMR_INTERVAL*)
- CONFIG_GATTC_CACHE_NVS_FLASH (*CONFIG_BT_GATTC_CACHE_NVS_FLASH*)
- CONFIG_GATTC_ENABLE (*CONFIG_BT_GATTC_ENABLE*)
- CONFIG_GATTS_ENABLE (*CONFIG_BT_GATTS_ENABLE*)
- CONFIG_GDBSTUB_MAX_TASKS (*CONFIG_ESP_GDBSTUB_MAX_TASKS*)
- CONFIG_GDBSTUB_SUPPORT_TASKS (*CONFIG_ESP_GDBSTUB_SUPPORT_TASKS*)
- **CONFIG_HFP_AUDIO_DATA_PATH (*CONFIG_BT_HFP_AUDIO_DATA_PATH*)**
  - CONFIG_HFP_AUDIO_DATA_PATH_PCM
  - CONFIG_HFP_AUDIO_DATA_PATH_HCI

- CONFIG_HFP_ENABLE (*CONFIG_BT_HFP_ENABLE*)
- **CONFIG_HFP_ROLE (*CONFIG_BT_HFP_ROLE*)**
  - CONFIG_HFP_CLIENT_ENABLE
  - CONFIG_HFP_AG_ENABLE
- CONFIG_INT_WDT (*CONFIG_ESP_INT_WDT*)
- CONFIG_INT_WDT_CHECK_CPU1 (*CONFIG_ESP_INT_WDT_CHECK_CPU1*)
- CONFIG_INT_WDT_TIMEOUT_MS (*CONFIG_ESP_INT_WDT_TIMEOUT_MS*)
- CONFIG_IPC_TASK_STACK_SIZE (*CONFIG_ESP_IPC_TASK_STACK_SIZE*)
- CONFIG_L2_TO_L3_COPY (*CONFIG_LWIP_L2_TO_L3_COPY*)
- CONFIG_MAIN_TASK_STACK_SIZE (*CONFIG_ESP_MAIN_TASK_STACK_SIZE*)
- CONFIG_MAKE_WARN_UNDEFINED_VARIABLES (*CONFIG_SDK_MAKE_WARN_UNDEFINED_VARIA*
- CONFIG_MB_CONTROLLER_NOTIFY_QUEUE_SIZE (*CONFIG_FMB_CONTROLLER_NOTIFY_QUEUE_*
- CONFIG_MB_CONTROLLER_NOTIFY_TIMEOUT (*CONFIG_FMB_CONTROLLER_NOTIFY_TIMEOUT*)
- CONFIG_MB_CONTROLLER_SLAVE_ID (*CONFIG_FMB_CONTROLLER_SLAVE_ID*)
- CONFIG_MB_CONTROLLER_SLAVE_ID_SUPPORT (*CONFIG_FMB_CONTROLLER_SLAVE_ID_SUPPO*
- CONFIG_MB_CONTROLLER_STACK_SIZE (*CONFIG_FMB_CONTROLLER_STACK_SIZE*)
- CONFIG_MB_EVENT_QUEUE_TIMEOUT (*CONFIG_FMB_EVENT_QUEUE_TIMEOUT*)
- CONFIG_MB_MASTER_DELAY_MS_CONVERT (*CONFIG_FMB_MASTER_DELAY_MS_CONVERT*)
- CONFIG_MB_MASTER_TIMEOUT_MS_RESPOND (*CONFIG_FMB_MASTER_TIMEOUT_MS_RESPOND*)
- CONFIG_MB_QUEUE_LENGTH (*CONFIG_FMB_QUEUE_LENGTH*)
- CONFIG_MB_SERIAL_BUF_SIZE (*CONFIG_FMB_SERIAL_BUF_SIZE*)
- CONFIG_MB_SERIAL_TASK_PRIO (*CONFIG_FMB_SERIAL_TASK_PRIO*)
- CONFIG_MB_SERIAL_TASK_STACK_SIZE (*CONFIG_FMB_SERIAL_TASK_STACK_SIZE*)
- CONFIG_MB_TIMER_GROUP (*CONFIG_FMB_TIMER_GROUP*)
- CONFIG_MB_TIMER_INDEX (*CONFIG_FMB_TIMER_INDEX*)
- CONFIG_MB_TIMER_PORT_ENABLED (*CONFIG_FMB_TIMER_PORT_ENABLED*)
- CONFIG_MESH_DUPLICATE_SCAN_CACHE_SIZE (*CONFIG_BTDM_MESH_DUPL_SCAN_CACHE_SIZE*
- CONFIG_MONITOR_BAUD_OTHER_VAL (*CONFIG_ESPTOOLPY_MONITOR_BAUD_OTHER_VAL*)
- CONFIG_NIMBLE_ACL_BUF_COUNT (*CONFIG_BT_NIMBLE_ACL_BUF_COUNT*)
- CONFIG_NIMBLE_ACL_BUF_SIZE (*CONFIG_BT_NIMBLE_ACL_BUF_SIZE*)

- CONFIG_NIMBLE_ATT_PREFERRED_MTU (*CONFIG_BT_NIMBLE_ATT_PREFERRED_MTU*)
- CONFIG_NIMBLE_CRYPTO_STACK_MBEDTLS (*CONFIG_BT_NIMBLE_CRYPTO_STACK_MBEDTLS*)
- CONFIG_NIMBLE_DEBUG (*CONFIG_BT_NIMBLE_DEBUG*)
- CONFIG_NIMBLE_GAP_DEVICE_NAME_MAX_LEN (*CONFIG_BT_NIMBLE_GAP_DEVICE_NAME_MA*
- CONFIG_NIMBLE_HCI_EVT_BUF_SIZE (*CONFIG_BT_NIMBLE_HCI_EVT_BUF_SIZE*)
- CONFIG_NIMBLE_HCI_EVT_HI_BUF_COUNT (*CONFIG_BT_NIMBLE_HCI_EVT_HI_BUF_COUNT*)
- CONFIG_NIMBLE_HCI_EVT_LO_BUF_COUNT (*CONFIG_BT_NIMBLE_HCI_EVT_LO_BUF_COUNT*)
- CONFIG_NIMBLE_HS_FLOW_CTRL (*CONFIG_BT_NIMBLE_HS_FLOW_CTRL*)
- CONFIG_NIMBLE_HS_FLOW_CTRL_ITVL (*CONFIG_BT_NIMBLE_HS_FLOW_CTRL_ITVL*)
- CONFIG_NIMBLE_HS_FLOW_CTRL_THRESH (*CONFIG_BT_NIMBLE_HS_FLOW_CTRL_THRESH*)
- CONFIG_NIMBLE_HS_FLOW_CTRL_TX_ON_DISCONNECT (*CONFIG_BT_NIMBLE_HS_FLOW_CTRL_TX_ON_DISCONNECT*)
- CONFIG_NIMBLE_L2CAP_COC_MAX_NUM (*CONFIG_BT_NIMBLE_L2CAP_COC_MAX_NUM*)
- CONFIG_NIMBLE_MAX_BONDS (*CONFIG_BT_NIMBLE_MAX_BONDS*)
- CONFIG_NIMBLE_MAX_CCCDS (*CONFIG_BT_NIMBLE_MAX_CCCDS*)
- CONFIG_NIMBLE_MAX_CONNECTIONS (*CONFIG_BT_NIMBLE_MAX_CONNECTIONS*)
- **CONFIG_NIMBLE_MEM_ALLOC_MODE (*CONFIG_BT_NIMBLE_MEM_ALLOC_MODE*)**
  - CONFIG_NIMBLE_MEM_ALLOC_MODE_INTERNAL
  - CONFIG_NIMBLE_MEM_ALLOC_MODE_EXTERNAL
  - CONFIG_NIMBLE_MEM_ALLOC_MODE_DEFAULT
- CONFIG_NIMBLE_MESH (*CONFIG_BT_NIMBLE_MESH*)
- CONFIG_NIMBLE_MESH_DEVICE_NAME (*CONFIG_BT_NIMBLE_MESH_DEVICE_NAME*)
- CONFIG_NIMBLE_MESH_FRIEND (*CONFIG_BT_NIMBLE_MESH_FRIEND*)
- CONFIG_NIMBLE_MESH_GATT_PROXY (*CONFIG_BT_NIMBLE_MESH_GATT_PROXY*)
- CONFIG_NIMBLE_MESH_LOW_POWER (*CONFIG_BT_NIMBLE_MESH_LOW_POWER*)
- CONFIG_NIMBLE_MESH_PB_ADV (*CONFIG_BT_NIMBLE_MESH_PB_ADV*)
- CONFIG_NIMBLE_MESH_PB_GATT (*CONFIG_BT_NIMBLE_MESH_PB_GATT*)
- CONFIG_NIMBLE_MESH_PROV (*CONFIG_BT_NIMBLE_MESH_PROV*)
- CONFIG_NIMBLE_MESH_PROXY (*CONFIG_BT_NIMBLE_MESH_PROXY*)
- CONFIG_NIMBLE_MESH_RELAY (*CONFIG_BT_NIMBLE_MESH_RELAY*)

- CONFIG_NIMBLE_MSYS1_BLOCK_COUNT (*CONFIG_BT_NIMBLE_MSYS1_BLOCK_COUNT*)
- CONFIG_NIMBLE_NVS_PERSIST (*CONFIG_BT_NIMBLE_NVS_PERSIST*)
- **CONFIG_NIMBLE_PINNED_TO_CORE_CHOICE (*CONFIG_BT_NIMBLE_PINNED_TO_C*

  - CONFIG_NIMBLE_PINNED_TO_CORE_0
  - CONFIG_NIMBLE_PINNED_TO_CORE_1
- CONFIG_NIMBLE_ROLE_BROADCASTER (*CONFIG_BT_NIMBLE_ROLE_BROADCASTER*)
- CONFIG_NIMBLE_ROLE_CENTRAL (*CONFIG_BT_NIMBLE_ROLE_CENTRAL*)
- CONFIG_NIMBLE_ROLE_OBSERVER (*CONFIG_BT_NIMBLE_ROLE_OBSERVER*)
- CONFIG_NIMBLE_ROLE_PERIPHERAL (*CONFIG_BT_NIMBLE_ROLE_PERIPHERAL*)
- CONFIG_NIMBLE_RPA_TIMEOUT (*CONFIG_BT_NIMBLE_RPA_TIMEOUT*)
- CONFIG_NIMBLE_SM_LEGACY (*CONFIG_BT_NIMBLE_SM_LEGACY*)
- CONFIG_NIMBLE_SM_SC (*CONFIG_BT_NIMBLE_SM_SC*)
- CONFIG_NIMBLE_SM_SC_DEBUG_KEYS (*CONFIG_BT_NIMBLE_SM_SC_DEBUG_KEYS*)
- CONFIG_NIMBLE_SVC_GAP_APPEARANCE (*CONFIG_BT_NIMBLE_SVC_GAP_APPEARANCE*)
- CONFIG_NIMBLE_SVC_GAP_DEVICE_NAME (*CONFIG_BT_NIMBLE_SVC_GAP_DEVICE_NAME*)
- CONFIG_NIMBLE_TASK_STACK_SIZE (*CONFIG_BT_NIMBLE_TASK_STACK_SIZE*)
- CONFIG_NO_BLOBS (*CONFIG_ESP32_NO_BLOBS*)
- **CONFIG_OPTIMIZATION_ASSERTION_LEVEL (*CONFIG_COMPILER_OPTIMIZATION_AS*

  - CONFIG_OPTIMIZATION_ASSERTIONS_ENABLED
  - CONFIG_OPTIMIZATION_ASSERTIONS_SILENT
  - CONFIG_OPTIMIZATION_ASSERTIONS_DISABLED
- **CONFIG_OPTIMIZATION_COMPILER (*CONFIG_COMPILER_OPTIMIZATION*)**

  - CONFIG_COMPILER_OPTIMIZATION_LEVEL_DEBUG
  - CONFIG_COMPILER_OPTIMIZATION_LEVEL_RELEASE
- CONFIG_POST_EVENTS_FROM_IRAM_ISR (*CONFIG_ESP_EVENT_POST_FROM_IRAM_ISR*)
- CONFIG_POST_EVENTS_FROM_ISR (*CONFIG_ESP_EVENT_POST_FROM_ISR*)
- CONFIG_PPP_CHAP_SUPPORT (*CONFIG_LWIP_PPP_CHAP_SUPPORT*)
- CONFIG_PPP_DEBUG_ON (*CONFIG_LWIP_PPP_DEBUG_ON*)

- CONFIG_PPP_MPPE_SUPPORT (*CONFIG_LWIP_PPP_MPPE_SUPPORT*)
- CONFIG_PPP_MSCHAP_SUPPORT (*CONFIG_LWIP_PPP_MSCHAP_SUPPORT*)
- CONFIG_PPP_NOTIFY_PHASE_SUPPORT (*CONFIG_LWIP_PPP_NOTIFY_PHASE_SUPPORT*)
- CONFIG_PPP_PAP_SUPPORT (*CONFIG_LWIP_PPP_PAP_SUPPORT*)
- CONFIG_PPP_SUPPORT (*CONFIG_LWIP_PPP_SUPPORT*)
- CONFIG_PYTHON (*CONFIG_SDK_PYTHON*)
- CONFIG_REDUCE_PHY_TX_POWER (*CONFIG_ESP32_REDUCE_PHY_TX_POWER*)
- CONFIG_SMP_SLAVE_CON_PARAMS_UPD_ENABLE (*CONFIG_BT_SMP_SLAVE_CON_PARAMS_UPD*
- CONFIG_SPIRAM_SUPPORT (*CONFIG_ESP32_SPIRAM_SUPPORT*)
- **CONFIG_SPI_FLASH_WRITING_DANGEROUS_REGIONS (*CONFIG_SPI_FLASH_DANGER***

  - CONFIG_SPI_FLASH_WRITING_DANGEROUS_REGIONS_ABORTS
  - CONFIG_SPI_FLASH_WRITING_DANGEROUS_REGIONS_FAILS
  - CONFIG_SPI_FLASH_WRITING_DANGEROUS_REGIONS_ALLOWED

- **CONFIG_STACK_CHECK_MODE (*CONFIG_COMPILER_STACK_CHECK_MODE*)**

  - CONFIG_STACK_CHECK_NONE
  - CONFIG_STACK_CHECK_NORM
  - CONFIG_STACK_CHECK_STRONG
  - CONFIG_STACK_CHECK_ALL

- CONFIG_SUPPORT_STATIC_ALLOCATION (*CONFIG_FREERTOS_SUPPORT_STATIC_ALLOCATION*)
- CONFIG_SUPPORT_TERMIOS (*CONFIG_VFS_SUPPORT_TERMIOS*)
- CONFIG_SUPPRESS_SELECT_DEBUG_OUTPUT (*CONFIG_VFS_SUPPRESS_SELECT_DEBUG_OUTPUT*
- CONFIG_SW_COEXIST_ENABLE (*CONFIG_ESP32_WIFI_SW_COEXIST_ENABLE*)
- CONFIG_SYSTEM_EVENT_QUEUE_SIZE (*CONFIG_ESP_SYSTEM_EVENT_QUEUE_SIZE*)
- CONFIG_SYSTEM_EVENT_TASK_STACK_SIZE (*CONFIG_ESP_SYSTEM_EVENT_TASK_STACK_SIZE*)
- CONFIG_TASK_WDT (*CONFIG_ESP_TASK_WDT*)
- CONFIG_TASK_WDT_CHECK_IDLE_TASK_CPU0 (*CONFIG_ESP_TASK_WDT_CHECK_IDLE_TASK_C*
- CONFIG_TASK_WDT_CHECK_IDLE_TASK_CPU1 (*CONFIG_ESP_TASK_WDT_CHECK_IDLE_TASK_C*
- CONFIG_TASK_WDT_PANIC (*CONFIG_ESP_TASK_WDT_PANIC*)
- CONFIG_TASK_WDT_TIMEOUT_S (*CONFIG_ESP_TASK_WDT_TIMEOUT_S*)

- CONFIG_TCPIP_RECVMBOX_SIZE (*CONFIG_LWIP_TCPIP_RECVMBOX_SIZE*)
- CONFIG_TCPIP_TASK_STACK_SIZE (*CONFIG_LWIP_TCPIP_TASK_STACK_SIZE*)
- CONFIG_TCP_MAXRTX (*CONFIG_LWIP_TCP_MAXRTX*)
- CONFIG_TCP_MSL (*CONFIG_LWIP_TCP_MSL*)
- CONFIG_TCP_MSS (*CONFIG_LWIP_TCP_MSS*)
- **CONFIG_TCP_OVERSIZE (*CONFIG_LWIP_TCP_OVERSIZE*)**
  - CONFIG_TCP_OVERSIZE_MSS
  - CONFIG_TCP_OVERSIZE_QUARTER_MSS
  - CONFIG_TCP_OVERSIZE_DISABLE
- CONFIG_TCP_QUEUE_OOSEQ (*CONFIG_LWIP_TCP_QUEUE_OOSEQ*)
- CONFIG_TCP_RECVMBOX_SIZE (*CONFIG_LWIP_TCP_RECVMBOX_SIZE*)
- CONFIG_TCP_SND_BUF_DEFAULT (*CONFIG_LWIP_TCP_SND_BUF_DEFAULT*)
- CONFIG_TCP_SYNMAXRTX (*CONFIG_LWIP_TCP_SYNMAXRTX*)
- CONFIG_TCP_WND_DEFAULT (*CONFIG_LWIP_TCP_WND_DEFAULT*)
- CONFIG_TIMER_QUEUE_LENGTH (*CONFIG_FREERTOS_TIMER_QUEUE_LENGTH*)
- CONFIG_TIMER_TASK_PRIORITY (*CONFIG_FREERTOS_TIMER_TASK_PRIORITY*)
- CONFIG_TIMER_TASK_STACK_DEPTH (*CONFIG_FREERTOS_TIMER_TASK_STACK_DEPTH*)
- CONFIG_TIMER_TASK_STACK_SIZE (*CONFIG_ESP_TIMER_TASK_STACK_SIZE*)
- CONFIG_TOOLPREFIX (*CONFIG_SDK_TOOLPREFIX*)
- CONFIG_UDP_RECVMBOX_SIZE (*CONFIG_LWIP_UDP_RECVMBOX_SIZE*)
- CONFIG_ULP_COPROC_ENABLED (*CONFIG_ESP32_ULP_COPROC_ENABLED*)
- CONFIG_ULP_COPROC_RESERVE_MEM (*CONFIG_ESP32_ULP_COPROC_RESERVE_MEM*)
- CONFIG_USE_ONLY_LWIP_SELECT (*CONFIG_LWIP_USE_ONLY_LWIP_SELECT*)
- CONFIG_WARN_WRITE_STRINGS (*CONFIG_COMPILER_WARN_WRITE_STRINGS*)
- CONFIG_WIFI_LWIP_ALLOCATION_FROM_SPIRAM_FIRST (*CONFIG_SPIRAM_TRY_ALLOCATE_WIFI_LWIP*)

2.8.7 Customisations

Because IDF builds by default with 警告未定义的变量, when the Kconfig tool generates Makefiles (the `auto.conf` file) its behaviour has been customised. In normal Kconfig, a variable which is set to "no" is undefined. In IDF's version of Kconfig, this variable is defined in the Makefile but has an empty value.

(Note that `ifdef` and `ifndef` can still be used in Makefiles, because they test if a variable is defined *and has a non-empty value*.)

When generating header files for C & C++, the behaviour is not customised - so `#ifdef` can be used to test if a boolean config item is set or not.

2.9 Error Codes Reference

This section lists various error code constants defined in ESP-IDF.

For general information about error codes in ESP-IDF, see *Error Handling*.

ESP_FAIL (-1): Generic esp_err_t code indicating failure

ESP_OK (0): esp_err_t value indicating success (no error)

ESP_ERR_NO_MEM **(0x101)**: Out of memory

ESP_ERR_INVALID_ARG **(0x102)**: Invalid argument

ESP_ERR_INVALID_STATE **(0x103)**: Invalid state

ESP_ERR_INVALID_SIZE **(0x104)**: Invalid size

ESP_ERR_NOT_FOUND **(0x105)**: Requested resource not found

ESP_ERR_NOT_SUPPORTED **(0x106)**: Operation or feature not supported

ESP_ERR_TIMEOUT **(0x107)**: Operation timed out

ESP_ERR_INVALID_RESPONSE **(0x108)**: Received response was invalid

ESP_ERR_INVALID_CRC **(0x109)**: CRC or checksum was invalid

ESP_ERR_INVALID_VERSION **(0x10a)**: Version was invalid

ESP_ERR_INVALID_MAC **(0x10b)**: MAC address was invalid

ESP_ERR_NOT_FINISHED **(0x201)**

ESP_ERR_NVS_BASE **(0x1100)**: Starting number of error codes

ESP_ERR_NVS_NOT_INITIALIZED **(0x1101)**: The storage driver is not initialized

ESP_ERR_NVS_NOT_FOUND **(0x1102)**: Id namespace doesn’t exist yet and mode is NVS_READONLY

ESP_ERR_NVS_TYPE_MISMATCH **(0x1103)**: The type of set or get operation doesn’t match the type of value stored in NVS

ESP_ERR_NVS_READ_ONLY **(0x1104)**: Storage handle was opened as read only

ESP_ERR_NVS_NOT_ENOUGH_SPACE **(0x1105)**: There is not enough space in the underlying storage to save the value

ESP_ERR_NVS_INVALID_NAME **(0x1106)**: Namespace name doesn’t satisfy constraints

ESP_ERR_NVS_INVALID_HANDLE **(0x1107)**: Handle has been closed or is NULL

ESP_ERR_NVS_REMOVE_FAILED **(0x1108)**: The value wasn't updated because flash write operation has failed. The value was written however, and update will be finished after re-initialization of nvs, provided that flash operation doesn't fail again.

ESP_ERR_NVS_KEY_TOO_LONG **(0x1109)**: Key name is too long

ESP_ERR_NVS_PAGE_FULL **(0x110a)**: Internal error; never returned by nvs API functions

ESP_ERR_NVS_INVALID_STATE **(0x110b)**: NVS is in an inconsistent state due to a previous error. Call nvs_flash_init and nvs_open again, then retry.

ESP_ERR_NVS_INVALID_LENGTH **(0x110c)**: String or blob length is not sufficient to store data

ESP_ERR_NVS_NO_FREE_PAGES **(0x110d)**: NVS partition doesn't contain any empty pages. This may happen if NVS partition was truncated. Erase the whole partition and call nvs_flash_init again.

ESP_ERR_NVS_VALUE_TOO_LONG **(0x110e)**: String or blob length is longer than supported by the implementation

ESP_ERR_NVS_PART_NOT_FOUND **(0x110f)**: Partition with specified name is not found in the partition table

ESP_ERR_NVS_NEW_VERSION_FOUND **(0x1110)**: NVS partition contains data in new format and cannot be recognized by this version of code

ESP_ERR_NVS_XTS_ENCR_FAILED **(0x1111)**: XTS encryption failed while writing NVS entry

ESP_ERR_NVS_XTS_DECR_FAILED **(0x1112)**: XTS decryption failed while reading NVS entry

ESP_ERR_NVS_XTS_CFG_FAILED **(0x1113)**: XTS configuration setting failed

ESP_ERR_NVS_XTS_CFG_NOT_FOUND **(0x1114)**: XTS configuration not found

ESP_ERR_NVS_ENCR_NOT_SUPPORTED **(0x1115)**: NVS encryption is not supported in this version

ESP_ERR_NVS_KEYS_NOT_INITIALIZED **(0x1116)**: NVS key partition is uninitialized

ESP_ERR_NVS_CORRUPT_KEY_PART **(0x1117)**: NVS key partition is corrupt

ESP_ERR_NVS_CONTENT_DIFFERS **(0x1118)**: Internal error; never returned by nvs API functions. NVS key is different in comparison

ESP_ERR_ULP_BASE **(0x1200)**: Offset for ULP-related error codes

ESP_ERR_ULP_SIZE_TOO_BIG **(0x1201)**: Program doesn't fit into RTC memory reserved for the ULP

ESP_ERR_ULP_INVALID_LOAD_ADDR **(0x1202)**: Load address is outside of RTC memory reserved for the ULP

ESP_ERR_ULP_DUPLICATE_LABEL **(0x1203)**: More than one label with the same number was defined

ESP_ERR_ULP_UNDEFINED_LABEL **(0x1204)**: Branch instructions references an undefined label

ESP_ERR_ULP_BRANCH_OUT_OF_RANGE **(0x1205)**: Branch target is out of range of B instruction (try replacing with BX)

ESP_ERR_OTA_BASE **(0x1500)**: Base error code for ota_ops api

ESP_ERR_OTA_PARTITION_CONFLICT **(0x1501)**: Error if request was to write or erase the current running partition

ESP_ERR_OTA_SELECT_INFO_INVALID **(0x1502)**: Error if OTA data partition contains invalid content

ESP_ERR_OTA_VALIDATE_FAILED **(0x1503)**: Error if OTA app image is invalid

ESP_ERR_OTA_SMALL_SEC_VER **(0x1504)**: Error if the firmware has a secure version less than the running firmware.

ESP_ERR_OTA_ROLLBACK_FAILED **(0x1505)**: Error if flash does not have valid firmware in passive partition and hence rollback is not possible

ESP_ERR_OTA_ROLLBACK_INVALID_STATE **(0x1506)**: Error if current active firmware is still marked in pending validation state (ESP_OTA_IMG_PENDING_VERIFY), essentially first boot of firmware image post upgrade and hence firmware upgrade is not possible

ESP_ERR_EFUSE **(0x1600)**: Base error code for efuse api.

ESP_OK_EFUSE_CNT **(0x1601)**: OK the required number of bits is set.

ESP_ERR_EFUSE_CNT_IS_FULL **(0x1602)**: Error field is full.

ESP_ERR_EFUSE_REPEATED_PROG **(0x1603)**: Error repeated programming of programmed bits is strictly forbidden.

ESP_ERR_CODING **(0x1604)**: Error while a encoding operation.

`ESP_ERR_IMAGE_BASE` **(0x2000)**

`ESP_ERR_IMAGE_FLASH_FAIL` **(0x2001)**

`ESP_ERR_IMAGE_INVALID` **(0x2002)**

ESP_ERR_WIFI_BASE **(0x3000)**: Starting number of WiFi error codes

ESP_ERR_WIFI_NOT_INIT **(0x3001)**: WiFi driver was not installed by esp_wifi_init

ESP_ERR_WIFI_NOT_STARTED **(0x3002)**: WiFi driver was not started by esp_wifi_start

ESP_ERR_WIFI_NOT_STOPPED **(0x3003)**: WiFi driver was not stopped by esp_wifi_stop

ESP_ERR_WIFI_IF **(0x3004)**: WiFi interface error

ESP_ERR_WIFI_MODE **(0x3005)**: WiFi mode error

ESP_ERR_WIFI_STATE **(0x3006)**: WiFi internal state error

ESP_ERR_WIFI_CONN **(0x3007)**: WiFi internal control block of station or soft-AP error

ESP_ERR_WIFI_NVS **(0x3008)**: WiFi internal NVS module error

ESP_ERR_WIFI_MAC **(0x3009)**: MAC address is invalid

ESP_ERR_WIFI_SSID **(0x300a)**: SSID is invalid

ESP_ERR_WIFI_PASSWORD (**0x300b**): Password is invalid

ESP_ERR_WIFI_TIMEOUT (**0x300c**): Timeout error

ESP_ERR_WIFI_WAKE_FAIL (**0x300d**): WiFi is in sleep state(RF closed) and wakeup fail

ESP_ERR_WIFI_WOULD_BLOCK (**0x300e**): The caller would block

ESP_ERR_WIFI_NOT_CONNECT (**0x300f**): Station still in disconnect status

ESP_ERR_WIFI_POST (**0x3012**): Failed to post the event to WiFi task

ESP_ERR_WIFI_INIT_STATE (**0x3013**): Invalod WiFi state when init/deinit is called

ESP_ERR_WIFI_STOP_STATE (**0x3014**): Returned when WiFi is stopping

ESP_ERR_WIFI_REGISTRAR (**0x3033**): WPS registrar is not supported

ESP_ERR_WIFI_WPS_TYPE (**0x3034**): WPS type error

ESP_ERR_WIFI_WPS_SM (**0x3035**): WPS state machine is not initialized

ESP_ERR_ESPNOW_BASE (**0x3064**): ESPNOW error number base.

ESP_ERR_ESPNOW_NOT_INIT (**0x3065**): ESPNOW is not initialized.

ESP_ERR_ESPNOW_ARG (**0x3066**): Invalid argument

ESP_ERR_ESPNOW_NO_MEM (**0x3067**): Out of memory

ESP_ERR_ESPNOW_FULL (**0x3068**): ESPNOW peer list is full

ESP_ERR_ESPNOW_NOT_FOUND (**0x3069**): ESPNOW peer is not found

ESP_ERR_ESPNOW_INTERNAL (**0x306a**): Internal error

ESP_ERR_ESPNOW_EXIST (**0x306b**): ESPNOW peer has existed

ESP_ERR_ESPNOW_IF (**0x306c**): Interface error

ESP_ERR_MESH_BASE (**0x4000**): Starting number of MESH error codes

ESP_ERR_MESH_WIFI_NOT_START (**0x4001**)

ESP_ERR_MESH_NOT_INIT (**0x4002**)

ESP_ERR_MESH_NOT_CONFIG (**0x4003**)

ESP_ERR_MESH_NOT_START (**0x4004**)

ESP_ERR_MESH_NOT_SUPPORT (**0x4005**)

ESP_ERR_MESH_NOT_ALLOWED (**0x4006**)

ESP_ERR_MESH_NO_MEMORY (**0x4007**)

ESP_ERR_MESH_ARGUMENT (**0x4008**)

ESP_ERR_MESH_EXCEED_MTU (**0x4009**)

ESP_ERR_MESH_TIMEOUT (0x400a)

ESP_ERR_MESH_DISCONNECTED (0x400b)

ESP_ERR_MESH_QUEUE_FAIL (0x400c)

ESP_ERR_MESH_QUEUE_FULL (0x400d)

ESP_ERR_MESH_NO_PARENT_FOUND (0x400e)

ESP_ERR_MESH_NO_ROUTE_FOUND (0x400f)

ESP_ERR_MESH_OPTION_NULL (0x4010)

ESP_ERR_MESH_OPTION_UNKNOWN (0x4011)

ESP_ERR_MESH_XON_NO_WINDOW (0x4012)

ESP_ERR_MESH_INTERFACE (0x4013)

ESP_ERR_MESH_DISCARD_DUPLICATE (0x4014)

ESP_ERR_MESH_DISCARD (0x4015)

ESP_ERR_MESH_VOTING (0x4016)

ESP_ERR_ESP_NETIF_BASE (0x5000)

ESP_ERR_ESP_NETIF_INVALID_PARAMS (0x5001)

ESP_ERR_ESP_NETIF_IF_NOT_READY (0x5002)

ESP_ERR_ESP_NETIF_DHCPC_START_FAILED (0x5003)

ESP_ERR_ESP_NETIF_DHCP_ALREADY_STARTED (0x5004)

ESP_ERR_ESP_NETIF_DHCP_ALREADY_STOPPED (0x5005)

ESP_ERR_ESP_NETIF_NO_MEM (0x5006)

ESP_ERR_ESP_NETIF_DHCP_NOT_STOPPED (0x5007)

ESP_ERR_ESP_NETIF_DRIVER_ATTACH_FAILED (0x5008)

ESP_ERR_ESP_NETIF_INIT_FAILED (0x5009)

ESP_ERR_ESP_NETIF_DNS_NOT_CONFIGURED (0x500a)

ESP_ERR_FLASH_BASE (0x6000): Starting number of flash error codes

ESP_ERR_FLASH_OP_FAIL (0x6001)

ESP_ERR_FLASH_OP_TIMEOUT (0x6002)

ESP_ERR_FLASH_NOT_INITIALISED (0x6003)

ESP_ERR_FLASH_UNSUPPORTED_HOST (0x6004)

ESP_ERR_FLASH_UNSUPPORTED_CHIP (0x6005)

ESP_ERR_FLASH_PROTECTED (**0x6006**)

ESP_ERR_HTTP_BASE (**0x7000**): Starting number of HTTP error codes

ESP_ERR_HTTP_MAX_REDIRECT (**0x7001**): The error exceeds the number of HTTP redirects

ESP_ERR_HTTP_CONNECT (**0x7002**): Error open the HTTP connection

ESP_ERR_HTTP_WRITE_DATA (**0x7003**): Error write HTTP data

ESP_ERR_HTTP_FETCH_HEADER (**0x7004**): Error read HTTP header from server

ESP_ERR_HTTP_INVALID_TRANSPORT (**0x7005**): There are no transport support for the input scheme

ESP_ERR_HTTP_CONNECTING (**0x7006**): HTTP connection hasn't been established yet

ESP_ERR_HTTP_EAGAIN (**0x7007**): Mapping of errno EAGAIN to esp_err_t

ESP_ERR_ESP_TLS_BASE (**0x8000**): Starting number of ESP-TLS error codes

ESP_ERR_ESP_TLS_CANNOT_RESOLVE_HOSTNAME (**0x8001**): Error if hostname couldn't be resolved upon tls connection

ESP_ERR_ESP_TLS_CANNOT_CREATE_SOCKET (**0x8002**): Failed to create socket

ESP_ERR_ESP_TLS_UNSUPPORTED_PROTOCOL_FAMILY (**0x8003**): Unsupported protocol family

ESP_ERR_ESP_TLS_FAILED_CONNECT_TO_HOST (**0x8004**): Failed to connect to host

ESP_ERR_ESP_TLS_SOCKET_SETOPT_FAILED (**0x8005**): failed to set socket option

ESP_ERR_MBEDTLS_CERT_PARTLY_OK (**0x8006**): mbedtls parse certificates was partly successful

ESP_ERR_MBEDTLS_CTR_DRBG_SEED_FAILED (**0x8007**): mbedtls api returned error

ESP_ERR_MBEDTLS_SSL_SET_HOSTNAME_FAILED (**0x8008**): mbedtls api returned error

ESP_ERR_MBEDTLS_SSL_CONFIG_DEFAULTS_FAILED (**0x8009**): mbedtls api returned error

ESP_ERR_MBEDTLS_SSL_CONF_ALPN_PROTOCOLS_FAILED (**0x800a**): mbedtls api returned error

ESP_ERR_MBEDTLS_X509_CRT_PARSE_FAILED (**0x800b**): mbedtls api returned error

ESP_ERR_MBEDTLS_SSL_CONF_OWN_CERT_FAILED (**0x800c**): mbedtls api returned error

ESP_ERR_MBEDTLS_SSL_SETUP_FAILED (**0x800d**): mbedtls api returned error

ESP_ERR_MBEDTLS_SSL_WRITE_FAILED (**0x800e**): mbedtls api returned error

ESP_ERR_MBEDTLS_PK_PARSE_KEY_FAILED (**0x800f**): mbedtls api returned failed

ESP_ERR_MBEDTLS_SSL_HANDSHAKE_FAILED (**0x8010**): mbedtls api returned failed

ESP_ERR_MBEDTLS_SSL_CONF_PSK_FAILED (**0x8011**): mbedtls api returned failed

ESP_ERR_ESP_TLS_CONNECTION_TIMEOUT (**0x8012**): new connection in esp_tls_low_level_conn connection timeouted

ESP_ERR_HTTPS_OTA_BASE (**0x9000**)

ESP_ERR_HTTPS_OTA_IN_PROGRESS (**0x9001**)

ESP_ERR_PING_BASE (**0xa000**)

ESP_ERR_PING_INVALID_PARAMS (**0xa001**)

ESP_ERR_PING_NO_MEM (**0xa002**)

ESP_ERR_HTTPD_BASE (**0xb000**): Starting number of HTTPD error codes

ESP_ERR_HTTPD_HANDLERS_FULL (**0xb001**): All slots for registering URI handlers have been consumed

ESP_ERR_HTTPD_HANDLER_EXISTS (**0xb002**): URI handler with same method and target URI already registered

ESP_ERR_HTTPD_INVALID_REQ (**0xb003**): Invalid request pointer

ESP_ERR_HTTPD_RESULT_TRUNC (**0xb004**): Result string truncated

ESP_ERR_HTTPD_RESP_HDR (**0xb005**): Response header field larger than supported

ESP_ERR_HTTPD_RESP_SEND (**0xb006**): Error occured while sending response packet

ESP_ERR_HTTPD_ALLOC_MEM (**0xb007**): Failed to dynamically allocate memory for resource

ESP_ERR_HTTPD_TASK (**0xb008**): Failed to launch server task/thread

CHAPTER 3

ESP32 H/W 硬件参考

[English]

3.1 ESP32 系列模组和开发板

[English]

乐鑫设计并提供多种模组和开发板以供用户体验 ESP32 系列芯片的强大功能。

本文档主要介绍了当前乐鑫所提供的各种模组和开发板。

注解： 如需了解较早版本或已停产的模组和开发板，请参考*Previous Versions of ESP32 Modules and Boards*。

3.1.1 WROOM、SOLO、WROVER 和 PICO 系列模组

WROOM、SOLO、WROVER 和 PICO 系列模组内置 ESP32 芯片并集成了晶振、天线匹配电路等重要组件，可直接集成到终端产品中。如果再结合一些其他组件，例如编程接口、Bootstrapping 电阻和排针，您就可以体验 ESP32 的强大功能了。

下表总结了上述系列模组的主要特点，详细信息见后续章节。

模组	芯片	Flash (MB)	PSRAM (MB)	天线	尺寸 (mm)
ESP32-WROOM-32	ESP32-D0WDQ6	4	–	MIFA	18 × 25.5 × 3.1
ESP32-WROOM-32D	ESP32-D0WD	4、8 或 16	–	MIFA	18 × 25.5 × 3.1
ESP32-WROOM-32U	ESP32-D0WD	4、8 或 16	–	U.FL	18 × 19.2 × 3.1
ESP32-SOLO-1	ESP32-S0WD	4	–	MIFA	18 × 25.5 × 3.1
ESP32-WROVER (PCB)	ESP32-D0WDQ6	4	8	MIFA	18 × 31.4 × 3.3
ESP32-WROVER (IPEX)	ESP32-D0WDQ6	4	8	U.FL	18 × 31.4 × 3.3
ESP32-WROVER-B	ESP32-D0WD	4、8 或 16	8	MIFA	18 × 31.4 × 3.3
ESP32-WROVER-IB	ESP32-D0WD	4、8 或 16	8	U.FL	18 × 31.4 × 3.3

- ESP32-**D**.. 代表双核芯片，ESP32-**S**.. 代表单核芯片；
- MIFA - 蛇形倒 F 天线；
- U.FL - U.FL/IPEX 天线连接器；
- ESP32-WROOM-32x、ESP32-WROVER-B 和 ESP32-WROVER-IB 模组默认内置 4 MB flash，客户可定制 8 MB 和 16 MB flash，详情见 乐鑫产品订购信息 (PDF) 和《ESP32 技术规格书》(PDF)；
- 最初发布的 ESP32-WROVER 模组内置 4 MB PSRAM；
- *ESP-WROOM-32* 是 *ESP32-WROOM-32* 的曾用名。

ESP32-WROOM-32 模组

ESP32-WROOM-32 模组是 WROOM/WROVER 系列最先发布的模组，内置 ESP32-D0WDQ6 芯片，是一款基础且常用的 ESP32 模组。

有关该模组的详细信息，请查看*WROOM、SOLO、WROVER 和 PICO 系列模组* 章节中的表格。

图 1: ESP32-WROOM-32 模组（正反面图）

相关文档

- 《ESP32-WROOM-32 技术规格书》(PDF)；
- ESP32-WROOM-32 参考设计，包括原理图（由 OrCAD Capture 绘制）、PCB 布局（由 Mentor PADS 绘制）、GERBER 文件和 BOM 清单。

ESP32-WROOM-32D/ESP32-WROOM-32U 模组

两款模组均集成了 ESP32-D0WD 芯片，与*ESP32-WROOM-32 模组* 集成的 ESP32-D0WDQ6 相比，ESP32-D0WD 芯片的封装更小，在 PCB 上占用的面积更小。

有关这两款模组的详细信息，请查看*WROOM、SOLO、WROVER 和 PICO 系列模组* 中的表格和 乐鑫产品订购信息。

ESP32-WROOM-32U 是整个 WROOM/WROVER 模组系列中最小的模组。

图 2: ESP32-WROOM-32D 模组（正反面图）

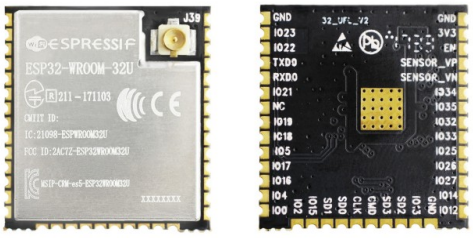


图 3: ESP32-WROOM-32U 模组（正反面图）

相关文档

- 《ESP32-WROOM-32D/ESP32-WROOM-32U 技术规格书》(PDF)

ESP32-SOLO-1 模组

ESP32-SOLO-1 模组是 ESP32-WROOM-32D 模组的简化版本，内置一个 ESP32 单核芯片，支持高达 160 MHz 的时钟频率。

有关此模组的详细信息，请查看 *WROOM、SOLO、WROVER 和 PICO 系列模组* 章节中的表格和 乐鑫产品订购信息。

图 4: ESP32-SOLO-1 模组（正反面图）

相关文档

- 《ESP32-SOLO-1 技术规格书》(PDF)

ESP32-WROVER 系列模组

ESP32-WROVER 系列模组在 ESP32-WROOM-32x 模组的基础上进行了一些修改，其中包含一些功能升级，并新增 8 MB SPI PSRAM（伪静态 RAM）。

有关该模组的详细信息，请查看 *WROOM、SOLO、WROVER 和 PICO 系列模组* 章节中的表格和 乐鑫产品订购信息。

- **ESP32-WROVER (PCB)** 模组和 **ESP32-WROVER (IPEX)** 模组内置 1.8 V PSRAM，支持 144 MHz 时钟频率。
- **ESP32-WROVER-B** 模组和 **ESP32-WROVER-IB** 模组内置 3.3 V PSRAM，支持 133 MHz 时钟频率。

下图为配备有 PCB 天线的 ESP32-WROVER 模组：

相关文档

- 《ESP32-WROVER 技术规格书》(PDF)
- 《ESP32-WROVER-B 技术规格书》(PDF)

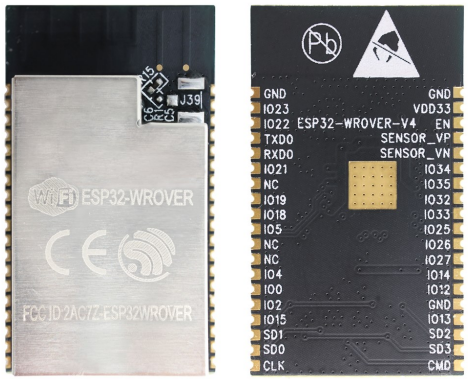


图 5: ESP32-WROVER 模组（正反面图）

- 《ESP-PSRAM64 & ESP-PSRAM64H 技术规格书》(PDF)
- ESP32-WROVER 参考设计 包含原理图（由 OrCAD Capture 绘制）、PCB 布局（由 Mentor PADS 绘制）、GERBER 文件和 BOM 清单。

ESP32-PICO-D4 模组

ESP32-PICO-D4 模组是一款 SiP 模组，无缝集成了所有外设，包括：

- 4 MB flash
- 晶振
- 滤波电容
- RF 匹配电路

有关该模组的详细信息，请查看 乐鑫产品订购信息。

相关文档

- 《ESP32-PICO-D4 技术规格书》(PDF)

3.1.2 ESP32 开发板

ESP32 系列开发板功能各异，具体有以下不同点：

- 访问的 ESP32 GPIO 管脚不同；
- 接口不同，包括 USB 和 JTAG；
- 外设不同，包括 TouchPad、LCD 显示屏、SD 卡槽和相机模组排母等。

ESP32-PICO-KIT V4.1 开发板

ESP32-PICO-KIT V4.1 开发板是基于 ESP32 的最小开发板，内置连接至电脑 USB 端口所需的所有组件，同时设有排针，可直接将此开发板插接于迷你面包板。

ESP32-PICO-KIT V4.1 开发板集成了*ESP32-PICO-D4 模组*，只需在 PCB (20 x 52 mm) 上添加少许外部组件即可构成一个功能齐全的开发板。这部分外部组件包括天线、LDO、USB 至 UART 桥接器、一个重置按钮和一个固件下载模式激活按钮。

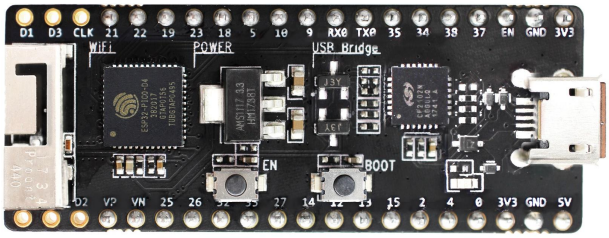


图 6: ESP32-PICO-KIT V4.1 开发板

与 ESP32-PICO-KIT V4 相比，ESP32-PICO-KIT V4.1 开发板支持 CP2102N USB 至 UART 桥接器，可提供高达 3 Mbps 的传输速率。

相关文档

- *ESP32-PICO-KIT V4/V4.1 入门指南*
- ESP32-PICO-KIT V4.1 原理图 (PDF)
- ESP32-PICO-KIT 参考设计，包含原理图（由 OrCAD Capture 绘制）、PCB 布局（由 Mentor PADS 绘制）、GERBER 文件和 BOM 清单。
- 《ESP32-PICO-D4 技术规格书》(PDF)

较早版本开发板

- *ESP32-PICO-KIT V4*
- *ESP32-PICO-KIT V3*

ESP32 DevKitC V4 开发板

ESP32 DevKitC V4 开发板是一款小巧实用的开发板，具备以下特色功能：

- 集成了*ESP32-WROOM-32 模组*
- USB 转串口编程接口同时可为开发板供电
- 设有排针

- 设有重置按钮和固件下载模式激活按钮
- 以及其他组件

与较早版本的*ESP32 Core Board V2 / ESP32 DevKitC* 相比，ESP32 DevKitC V4 开发板集成了*ESP32-WROVER 系列模组* （而非 ESP32-WROOM-32 模组），同时内置 CP2102N 芯片，支持更高波特率。

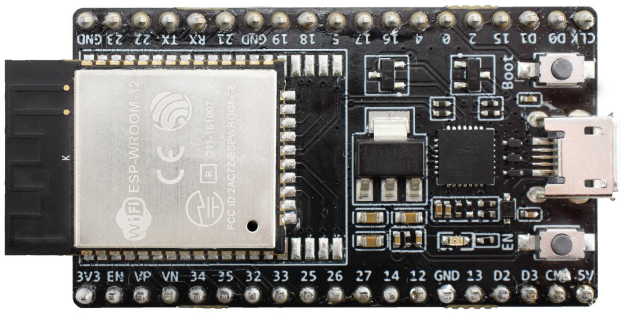


图 7: ESP32 DevKitC V4 开发板

相关文档

- *ESP32-DevKitC V4 入门指南*
- ESP32-DevKitC 原理图 (PDF)
- ESP32-DevKitC 参考设计，包含原理图（由 OrCAD Capture 绘制）、PCB 布局（由 Mentor PADS 绘制）、GERBER 文件和 BOM 清单。
- CP210x USB 至 UART 桥 VCP 驱动器

较早版本开发板

- *ESP32 Core Board V2 / ESP32 DevKitC*

ESP-WROVER-KIT V4.1 开发板

ESP-WROVER-KIT V4.1 开发板具备以下特色功能：

- USB 转双串口转换器（用于后续编程）
- JTAG 调试接口
- MicroSD 卡槽
- 3.2” SPI LCD 显示屏
- 相机模组排母
- RGB 发光二极管

- 支持 32.768 kHz 晶振输入用于 RTC 及低功耗模式操作

ESP-WROVER-KIT V4.1 开发板支持 USB 供电或标准的 5 毫米电源插座供电，可使用跳线选择电源，或使用独立的开关控制电源。

ESP-WROVER-KIT V4.1 开发板集成了 ESP-WROVER-B 模组，该模组集成了 8 MB PSRAM，方便用户灵活扩展存储空间，增强数据处理能力。ESP-WROVER-KIT V4.1 开发板还可以集成*WROOM、SOLO、WROVER 和 PICO 系列模组* 中所述的 ESP 其他版本模组。

与*ESP-WROVER-KIT V3* 相比，ESP-WROVER-KIT V4.1 开发板在设计上有以下改动：

- JP8、JP11 和 JP13 合并成了一个 JP2。
- USB 连接器的固定脚改为直插式，并移至板子右下角。
- R61 已变更为零欧姆电阻。
- 基于测试结果和采购选择，部分组件已由功能对等组件替代，例如 EN 和 Boot 按钮。

上图所示开发板集成了 ESP32-WROVER-B 模组。

相关文档

- *ESP-WROVER-KIT V4.1 入门指南*
- ESP-WROVER-KIT V4.1 原理图 (PDF)
- *JTAG 调试*
- FTDI 虚拟 COM 端口驱动

较早版本开发板

- *ESP-WROVER-KIT V3*
- *ESP-WROVER-KIT V2*
- *ESP-WROVER-KIT V1 / ESP32 DevKitJ V1*

3.1.3 相关文档

- *Previous Versions of ESP32 Modules and Boards*

3.2 Previous Versions of ESP32 Modules and Boards

This sections contains overview and links to documentation of previous version ESP32 Modules and Boards that have been replaced with newer versions or discontinued. It is maintained for convenience of users as previous versions of some modules and boards are still in use and some may still be available for purchase.

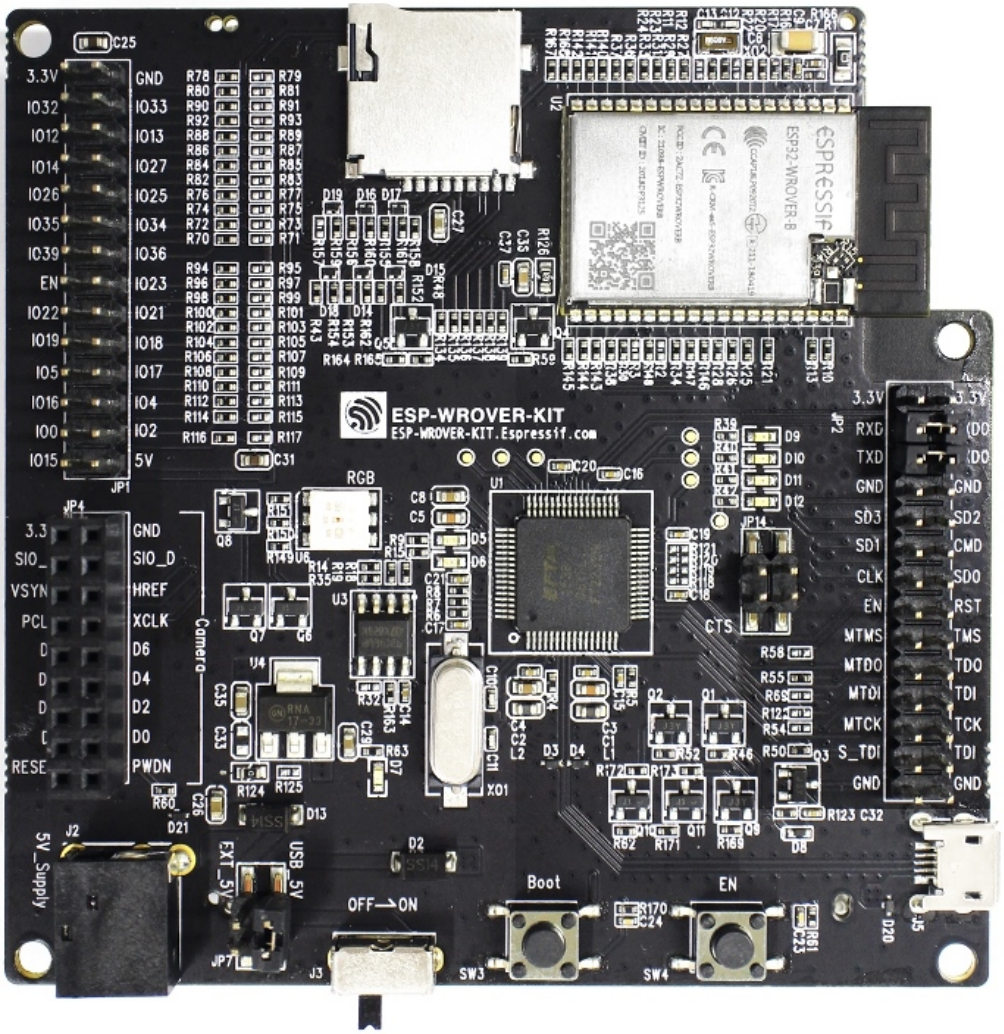


图 8: ESP-WROVER-KIT V4.1 开发板

3.2.1 Modules (No updated or discontinued modules)

So far, no modules have been updated or discontinued.

3.2.2 Development Boards

To see the latest development boards, please refer to section *ESP32 系列模组和开发板*.

ESP32-PICO-KIT V4

The smallest ESP32 development board with all the components required to connect it directly to a PC USB port, and pin headers to plug into a mini breadboard. It is equipped with ESP32-PICO-D4 module that integrates 4 MB flash memory, a crystal oscillator, filter capacitors and RF matching circuit in one single package. As result, the fully functional development board requires only a few external components that can easy fit on a 20 x 52 mm PCB including antenna, LDO, USB-UART bridge and two buttons to reset it and put into download mode.

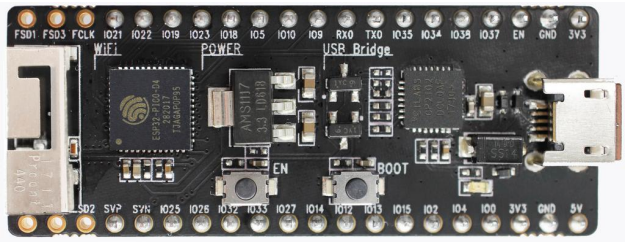


图 9: ESP32-PICO-KIT V4 board

Comparing to ESP32-PICO-KIT V3, this version has revised printout and reduced number of exposed pins. Instead of 20, only 17 header pins are populated, so V4 can fit into a mini breadboard.

Documentation

- *ESP32-PICO-KIT V4/V4.1 入门指南*
- ESP32-PICO-KIT V4 Schematic (PDF)
- ESP32-PICO-D4 Datasheet (PDF)

ESP32-PICO-KIT V3

The first public release of Espressif' s ESP32-PICO-D4 module on a mini development board. The board has a USB port for programming and debugging and two rows of 20 pin headers to plug into a breadboard. The ESP32-PICO-D4 module itself is small and requires only a few external components. Besides two core CPUs it integrates 4MB flash memory, a crystal oscillator and antenna matching components in one single

7 x 7 mm package. As a result the module and all the components making the complete development board fit into 20 x 52 mm PCB.

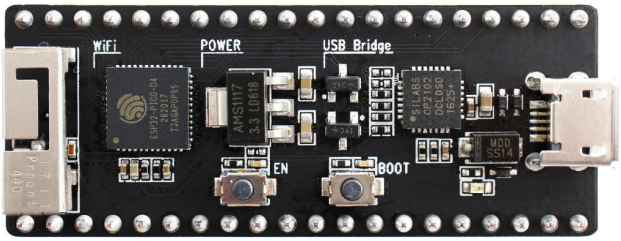


图 10: ESP32-PICO-KIT V3 board

Documentation

- *ESP32-PICO-KIT V3 入门指南*
- ESP32-PICO-KIT V3 Schematic (PDF)
- ESP32-PICO-D4 Datasheet (PDF)

ESP32 Core Board V2 / ESP32 DevKitC

Small and convenient development board with ESP-WROOM-32 module installed, break out pin headers and minimum additional components. Includes USB to serial programming interface, that also provides power supply for the board. Has pushbuttons to reset the board and put it in upload mode.

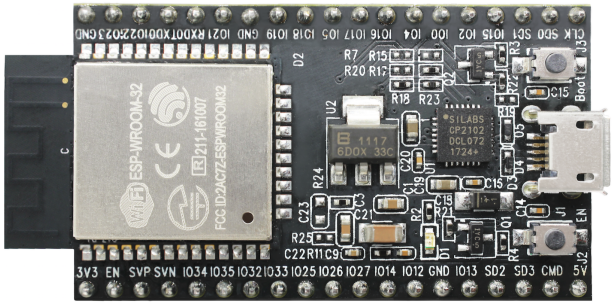


图 11: ESP32 Core Board V2 / ESP32 DevKitC board

Documentation

- *ESP32-DevKitC V2 入门指南*
- ESP32 DevKitC V2 Schematic (PDF)
- CP210x USB to UART Bridge VCP Drivers

ESP-WROVER-KIT V3

The ESP-WROVER-KIT V3 development board has dual port USB to serial converter for programming and JTAG interface for debugging. Power supply is provided by USB interface or from standard 5 mm power supply jack. Power supply selection is done with a jumper and may be put on/off with a separate switch. This board has MicroSD card slot, 3.2" SPI LCD screen and dedicated header to connect a camera. It provides RGB diode for diagnostics. Includes 32.768 kHz XTAL for internal RTC to operate it in low power modes.

As all previous versions of ESP-WROVER-KIT boards, it is ready to accommodate an *ESP32-WROOM-32 模组* or *ESP32-WROVER 系列模组* module.

This is the first release of ESP-WROVER-KIT shipped with *ESP32-WROVER 系列模组* module installed by default. This release also introduced several design changes to conditioning and interlocking of signals to the bootstrapping pins. Also, a zero Ohm resistor (R166) has been added between WROVER/WROOM module and VDD33 net, which can be desoldered, or replaced with a shunt resistor, for current measurement. This is intended to facilitate power consumption analysis in various operation modes of ESP32. Refer to schematic - the changes are enclosed in green border.

The camera header has been changed from male back to female. The board soldermask is matte black. The board on picture above has *ESP32-WROVER 系列模组* is installed.

Documentation

- *ESP-WROVER-KIT V3 入门指南*
- ESP-WROVER-KIT V3 Schematic (PDF)
- *JTAG 调试*
- FTDI Virtual COM Port Drivers

ESP-WROVER-KIT V2

This is updated version of ESP32 DevKitJ V1 described above with design improvements identified when DevKitJ was in use, e.g. improved support for SD card. By default board has ESP-WROOM-32 module installed.

Comparing to previous version, this board has a shiny black finish and a male camera header.

Documentation

- *ESP-WROVER-KIT V2 入门指南*
- ESP-WROVER-KIT V2 Schematic (PDF)
- *JTAG 调试*

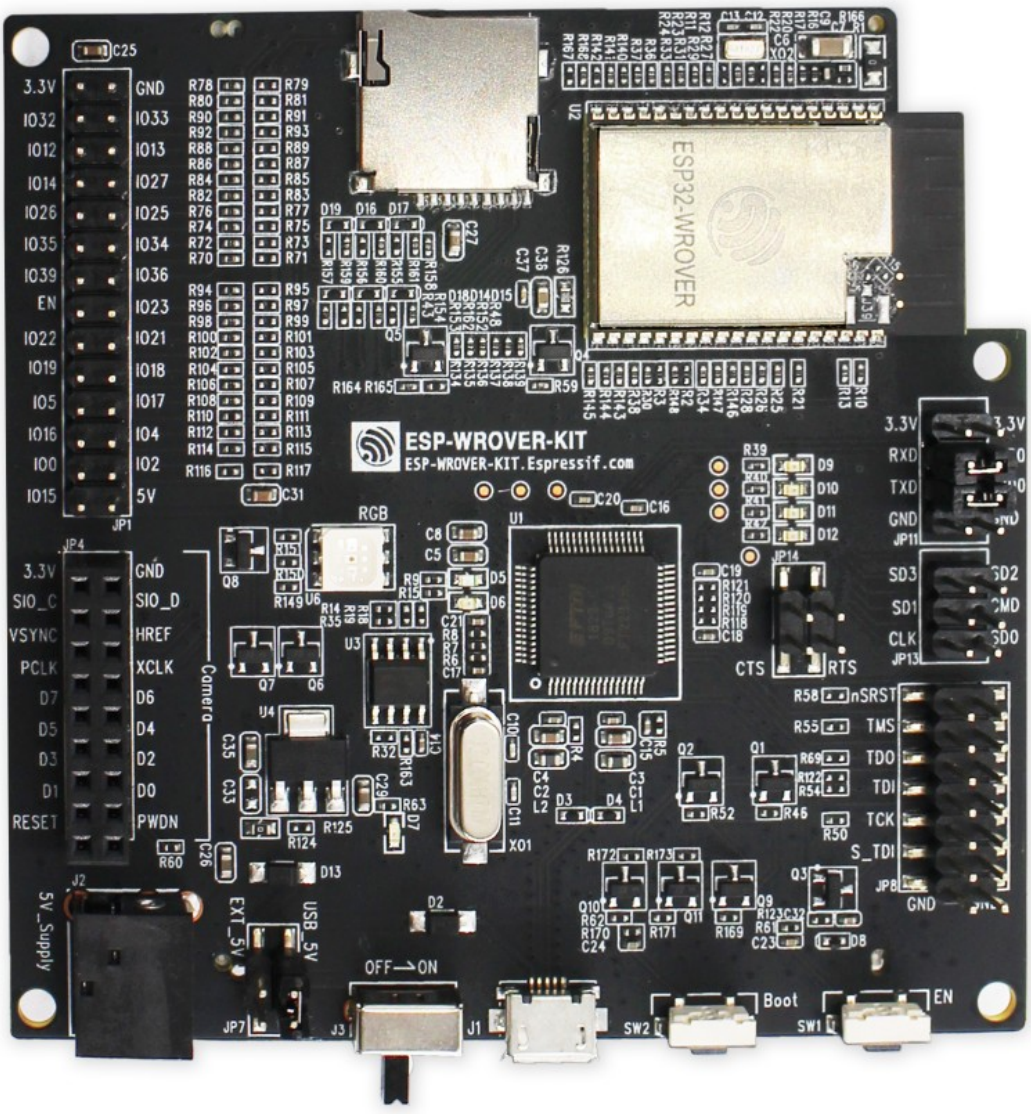


图 12: ESP-WROVER-KIT V3 board

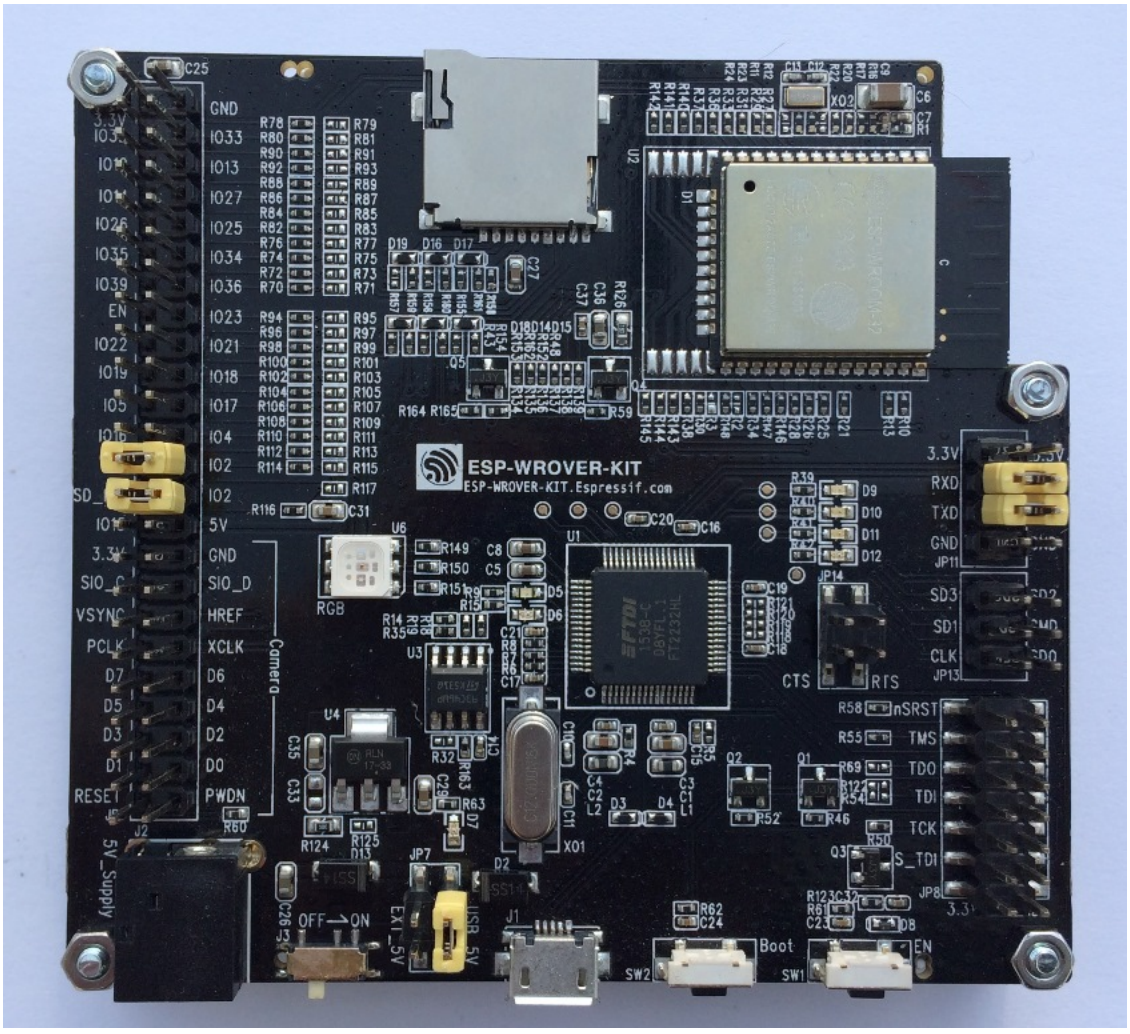


图 13: ESP-WROVER-KIT V2 board

- FTDI Virtual COM Port Drivers

ESP-WROVER-KIT V1 / ESP32 DevKitJ V1

The first version of ESP-WROVER-KIT development board. Shipped with ESP-WROOM-32 on board.

ESP-WROVER-KIT has dual port USB to serial converter for programming and JTAG interface for debugging. Power supply is provided by USB interface or from standard 5 mm power supply jack. Power supply selection is done with a jumper and may be put on/off with a separate switch. The board has MicroSD card slot, 3.2" SPI LCD screen and dedicated header to connect a camera. It provides RGB diode for diagnostics. Includes 32.768 kHz XTAL for internal RTC to operate it in low power modes.

All versions of ESP-WROVER-KIT are ready to accommodate an ESP-WROOM-32 or ESP32-WROVER module.

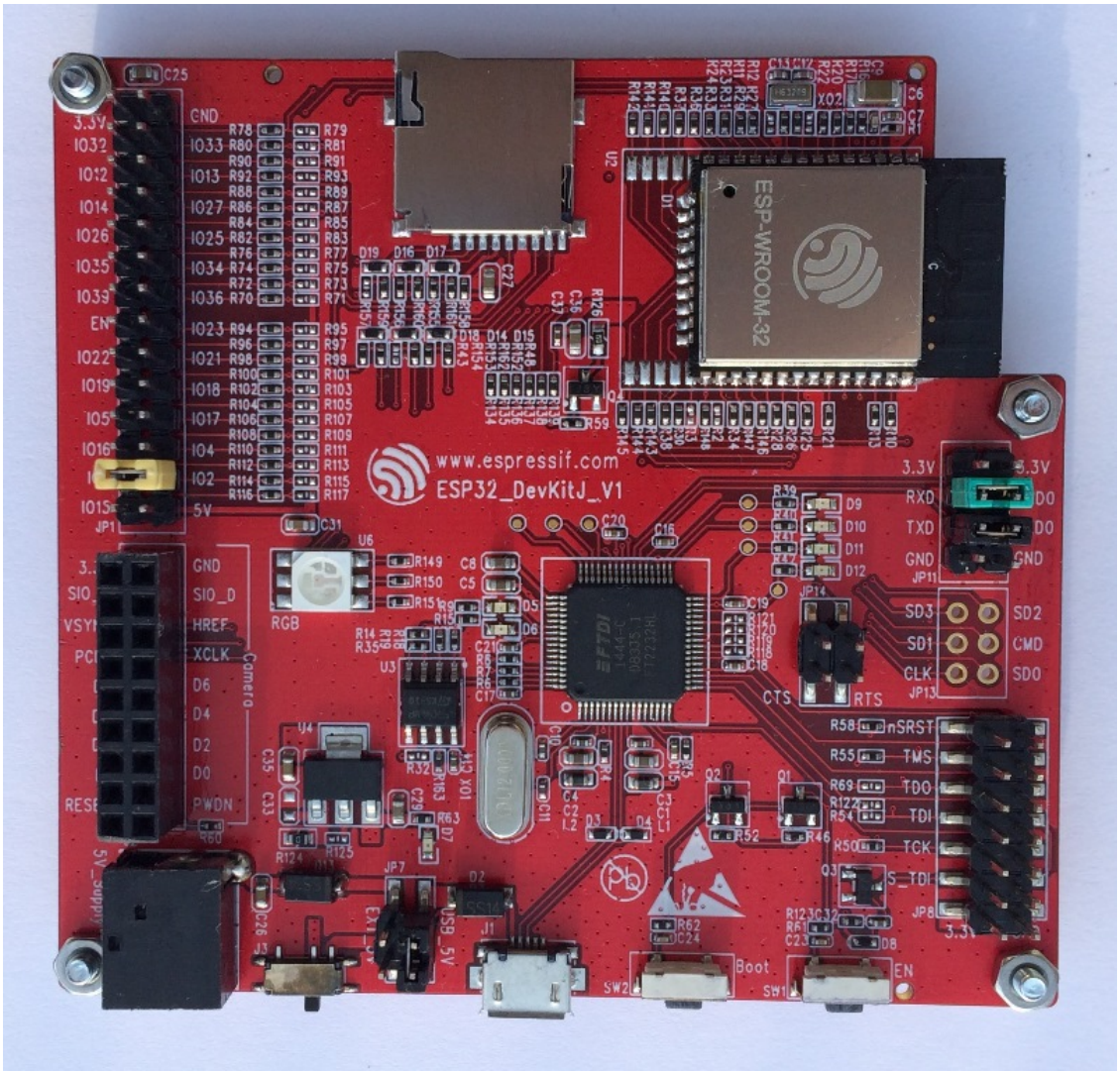


图 14: ESP-WROVER-KIT V1 / ESP32 DevKitJ V1 board

The board has red soldermask.

Documentation

- ESP-WROVER-KIT V1 Schematic (PDF)
- *JTAG 调试*
- FTDI Virtual COM Port Drivers

ESP32 Demo Board V2

One of first feature rich evaluation boards that contains several pin headers, dip switches, USB to serial programming interface, reset and boot mode press buttons, power switch, 10 touch pads and separate header to connect LCD screen.

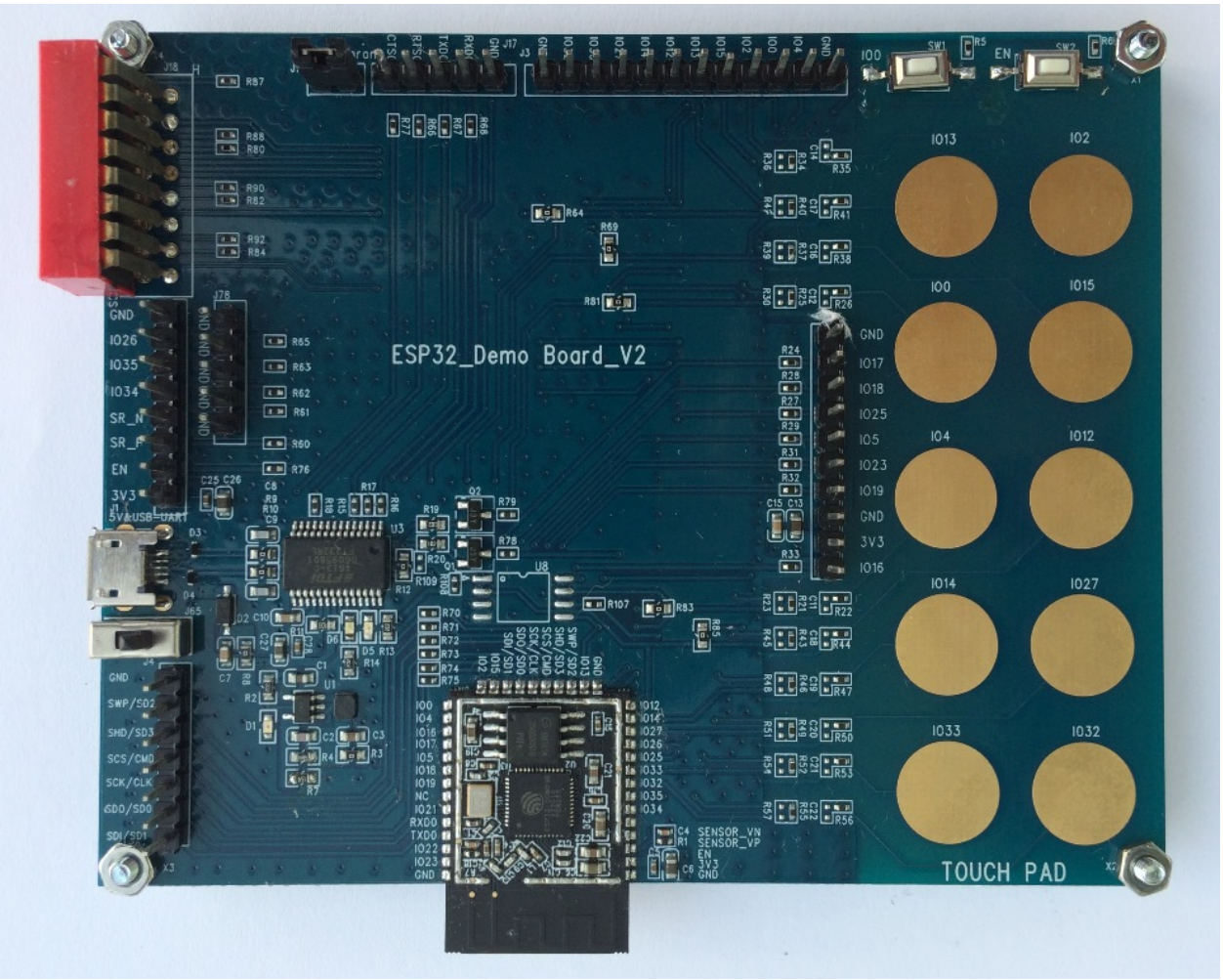


图 15: ESP32 Demo Board V2

Production of this board is discontinued.

Documentation

- ESP32 Demo Board V2 Schematic (PDF)
- FTDI Virtual COM Port Drivers

3.2.3 Related Documents

- *ESP32 系列模组和开发板*

CHAPTER 4

ESP32-S2 Beta Preview Support

The current master branch of ESP-IDF (pre-V4.1) contains preview support for ESP32-S2 Beta.

The ESP32-S2 Beta chip is not the same as the final ESP32-S2, the Beta chip is a limited engineering sample and as a result not all features are available.

Software support is still being developed ahead of the final ESP32-S2 release (see *Limitations* and *Roadmap* sections below).

4.1 Documentation

Except where it specifically mentions ESP32-S2, this version of the ESP-IDF Programming Guide is written for ESP32 only. Most software APIs and components are the same or very similar for ESP32-S2, but differences may not be documented yet.

4.2 Setting up for ESP32-S2 Beta

To start with, follow the 快速入门 guide to configure the CMake-based build system for ESP32.

When running `install.sh` or `install.bat` as part of the Get Started guide, the ESP32-S2 toolchain will be automatically downloaded and added to the IDF Tools directory. If you had already run `install.sh` or `install.bat` before updating to a version with ESP32-S2 support, you may need to re-run it.

The toolchain tuple is *xtensa-esp32s2-elf-*, for example the GCC compiler for this target is named *xtensa-esp32s2-elf-gcc*.

4.2.1 Manual toolchain URLs

If not using `install.sh` or `install.bat`, you can download the manual toolchain manually from here:

- Windows
- macOS
- Linux 64-bit x86
- Linux 32-bit x86

If installing the toolchain manually, unpack it somewhere and add the `bin` subdirectory to your *PATH*.

4.3 Building a project for ESP32-S2 Beta

重要: Not all ESP-IDF examples support ESP32-S2 Beta.

For some examples this is because required hardware is not included in ESP32-S2 so it cannot be supported. For some examples this is because the example has not yet been updated for ESP32-S2.

If building an example, please check the example CMakeLists.txt file for the clause `SUPPORTED_TARGETS esp32`. If this is present, the example won't work on ESP32-S2.

Before building an ESP-IDF project, run the following command to switch the target to ESP32-S2 Beta:

```
idf.py set-target esp32s2beta
```

重要: Running `set-target` will clear the project configuration and create a new empty `sdkconfig` file for the project. The old configuration will be saved in `sdkconfig.old`.

Then configure the project:

```
idf.py menuconfig
```

重要: There are 3 versions of ESP32-S2 Beta engineering sample available with different Wi-Fi PHYs. They can be identified by the marking on the chip itself. The project must be configured to match the chip via the project setting "ESP32-S2 Beta chip version" (`CONFIG_ESP32S2_BETA_CHIP_VERSION`). This can be found under Component Config -> ESP32S2-specific. If the project setting doesn't match the chip then Wi-Fi performance may be very poor, or the chip may crash when initializing Wi-Fi.

Build and flash the project as usual, for example:

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

Consult the 快速入门 guide for more details on building and flashing an ESP-IDF project.

注解: It is possible to set the project target back to ESP32 by running `idf.py set-target esp32`. This also clears the project configuration.

4.4 Limitations

ESP32-S2 Beta support is currently a preview and does not support all features. Espressif firmware developers are working hard on the remaining support, so please update the `master` branch regularly to receive the latest code.

- The integrated USB OTG peripheral is not supported
- Documentation not updated for ESP32-S2 in most places
- Peripheral drivers are a work in progress
- No power saving modes
- No hardware security features or cryptographic accelerator support
- Time-of-Flight (TOF) measurements with normal Wi-Fi packets is not supported

4.5 Roadmap

- ESP-IDF V4.1 will support ESP32-S2 Beta chip as a preview with some limitations.
- ESP-IDF V4.2 will support ESP32-S2.

Support for ESP32-S2 Beta will be removed from ESP-IDF once ESP32-S2 is available.

4.6 Issues

If you find issues which are not mentioned in *Limitations*, please open an issue on GitHub.

Please make sure to mention that you are using the ESP32-S2 Beta preview support and give your exact version of ESP-IDF.

CHAPTER 5

API 指南

[English]

5.1 ESP-IDF 编程注意事项

[English]

5.1.1 应用程序的启动流程

本文将会介绍 ESP32 从上电到运行 `app_main` 函数中间所经历的步骤（即启动流程）。

宏观上，该启动流程可以分为如下 3 个步骤：

1. 一级引导程序被固化在了 ESP32 内部的 ROM 中，它会从 Flash 的 `0x1000` 偏移地址处加载二级引导程序至 RAM(IRAM & DRAM) 中。
2. 二级引导程序从 Flash 中加载分区表和主程序镜像至内存中，主程序中包含了 RAM 段和通过 Flash 高速缓存映射的只读段。
3. 主程序运行，这时第二个 CPU 和 RTOS 的调度器可以开始运行。

下面会对上述过程进行更为详细的阐述。

一级引导程序

SoC 复位后，PRO CPU 会立即开始运行，执行复位向量代码，而 APP CPU 仍然保持复位状态。在启动过程中，PRO CPU 会执行所有的初始化操作。APP CPU 的复位状态会在应用程序启动代码的 `call_start_cpu0` 函数中失效。复位向量代码位于 ESP32 芯片掩膜 ROM 的 `0x40000400` 地址处，该地址不能被修改。

复位向量调用的启动代码会根据 `GPIO_STRAP_REG` 寄存器的值来确定 ESP32 的工作模式，该寄存器保存着复位后 bootstrap 引脚的电平状态。根据不同的复位原因，程序会执行不同的操作：

1. 从深度睡眠模式复位：如果 `RTC_CNTL_STORE6_REG` 寄存器的值非零，并且 `RTC_CNTL_STORE7_REG` 寄存器中的 RTC 内存的 CRC 校验值有效，那么程序会使用 `RTC_CNTL_STORE6_REG` 寄存器的值作为入口地址，并立即跳转到该地址运行。如果 `RTC_CNTL_STORE6_REG` 的值为零，或者 `RTC_CNTL_STORE7_REG` 中的 CRC 校验值无效，又或者跳转到 `RTC_CNTL_STORE6_REG` 地址处运行的程序返回，那么将会执行上电复位的相关操作。**注意**：如果想在这里运行自定义的代码，可以参考深度睡眠 文档里面介绍的方法。
2. 上电复位、软件 SoC 复位、看门狗 SoC 复位：检查 `GPIO_STRAP_REG` 寄存器，判断是否 UART 或 SDIO 请求进入下载模式。如果是，则配置好 UART 或者 SDIO，然后等待下载代码。否则程序将会执行软件 CPU 复位的相关操作。
3. 软件 CPU 复位、看门狗 CPU 复位：根据 EFUSE 中的值配置 SPI Flash，然后尝试从 Flash 中加载代码，这部分的内存将会在后面一小节详细介绍。如果从 Flash 中加载代码失败，就会将 BASIC 解析器加压缩到 RAM 中启动。需要注意的是，此时 RTC 看门狗还在使能状态，如果在几百毫秒内没有任何输入事件，那么看门狗会再次复位 SoC，重复整个过程。如果解析器收到了来自 UART 的输入，程序会关闭看门狗。

应用程序的二进制镜像会从 Flash 的 `0x1000` 地址处加载。Flash 的第一个 4kB 扇区用于存储安全引导程序和应用程序镜像的签名。有关详细信息，请查看安全启动文档。

二级引导程序

在 ESP-IDF 中，存放在 Flash 的 `0x1000` 偏移地址处的二进制镜像就是二级引导程序。二级引导程序的源码可以在 ESP-IDF 的 components/bootloader 目录下找到。请注意，对于 ESP32 芯片来说，这并不是唯一的安排程序镜像的方式。事实上用户完全可以把一个功能齐全的应用程序烧写到 Flash 的 `0x1000` 偏移地址处运行，但这超出本文档的范围。ESP-IDF 使用二级引导程序可以增加 Flash 分区的灵活性（使用分区表），并且方便实现 Flash 加密，安全引导和空中升级（OTA）等功能。

当一级引导程序校验并加载完二级引导程序后，它会从二进制镜像的头部找到二级引导程序的入口点，并跳转过去运行。

二级引导程序从 Flash 的 `0x8000` 偏移地址处读取分区表。详细信息请参阅分区表文档分区表 。二级引导程序会寻找出厂分区和 OTA 分区，然后根据 OTA 信息分区的数据决引导哪个分区。

对于选定的分区，二级引导程序将映射到 IRAM 和 DRAM 的数据和代码段复制到它们的加载地址处。对于一些加载地址位于 DROM 和 IROM 区域的段，会通过配置 Flash MMU 为其提供正确的映射。请注意，二级引导程序会为 PRO CPU 和 APP CPU 都配置 Flash MMU，但它只使能了 PRO CPU 的 Flash MMU。这么做的原因在于二级引导程序的代码被加载到了 APP CPU 的高速缓存使用的内存区域，因此使能 APP

CPU 高速缓存的任务就交给了应用程序。一旦代码加载完毕并且设置好 Flash MMU，二级引导程序会从应用程序二进制镜像文件的头部寻找入口地址，然后跳转到该地址处运行。

目前还不支持添加钩子函数到二级引导程序中以自定义应用程序分区选择的逻辑，但是可以通过别的途径实现这个需求，比如根据某个 GPIO 的不同状态来引导不同的应用程序镜像。此类自定义的功能将在未来添加到 ESP-IDF 中。目前，可以通过将 bootloader 组件复制到应用程序目录并在那里进行必要的更改来自定义引导程序。在这种情况下，ESP-IDF 的编译系统将编译应用程序目录中的组件而不是 ESP-IDF 组件目录。

应用程序启动阶段

ESP-IDF 应用程序的入口是 `components/esp32/cpu_start.c` 文件中的 `call_start_cpu0` 函数，该函数主要完成了两件事，一是启用堆分配器，二是使 APP CPU 跳转到其入口点——`call_start_cpu1` 函数。PRO CPU 上的代码会给 APP CPU 设置好入口地址，解除其复位状态，然后等待 APP CPU 上运行的代码设置一个全局标志，以表明 APP CPU 已经正常启动。完成后，PRO CPU 跳转到 `start_cpu0` 函数，APP CPU 跳转到 `start_cpu1` 函数。

`start_cpu0` 和 `start_cpu1` 这两个函数都是弱类型的，这意味着如果某些特定的应用程序需要修改初始化顺序，就可以通过重写这两个函数来实现。`start_cpu0` 默认的实现方式是初始化用户在 `menuconfig` 中选择的组件，具体实现步骤可以阅读 `components/esp32/cpu_start.c` 文件中的源码。请注意，此阶段会调用应用程序中存在的 C++ 全局构造函数。一旦所有必要的组件都初始化好，就会创建 *main task* ，并启动 FreeRTOS 的调度器。

当 PRO CPU 在 `start_cpu0` 函数中进行初始化的时候，APP CPU 在 `start_cpu1` 函数中自旋，等待 PRO CPU 上的调度器启动。一旦 PRO CPU 上的调度器启动后，APP CPU 上的代码也会启动调度器。

主任务是指运行 `app_main` 函数的任务，主任务的堆栈大小和优先级可以在 `menuconfig` 中进行配置。应用程序可以用此任务来完成用户程序相关的初始化设置，比如启动其他的任务。应用程序还可以将主任务用于事件循环和其他通用活动。如果 `app_main` 函数返回，那么主任务将会被删除。

5.1.2 应用程序的内存布局

ESP32 芯片具有灵活的内存映射功能，本小节将介绍 ESP-IDF 默认使用这些功能的方式。

ESP-IDF 应用程序的代码可以放在以下内存区域之一。

IRAM（指令 RAM）

ESP-IDF 将内部 SRAM0 区域（在技术参考手册中有定义）的一部分分配为指令 RAM。除了开始的 64kB 用作 PRO CPU 和 APP CPU 的高速缓存外，剩余内存区域（从 `0x40080000` 至 `0x400A0000` ）被用来存储应用程序中部分需要在 RAM 中运行的代码。

一些 ESP-IDF 的组件和 WiFi 协议栈的部分代码通过链接脚本文件被存放到了这块内存区域。

如果一些应用程序的代码需要放在 IRAM 中运行，可以使用 `IRAM_ATTR` 宏定义进行声明。

```
#include "esp_attr.h"

void IRAM_ATTR gpio_isr_handler(void* arg)
{
    // ...
}
```

下面列举了应用程序中可能或者应该放入 IRAM 中运行例子。

- 当注册中断处理程序的时候设置了 ESP_INTR_FLAG_IRAM ，那么中断处理程序就必须要放在 IRAM 中运行。这种情况下，ISR 只能调用存放在 IRAM 或者 ROM 中的函数。注意：目前所有 FreeRTOS 的 API 都已经存放到了 IRAM 中，所以在中断中调用 FreeRTOS 的中断专属 API 是安全的。如果将 ISR 放在 IRAM 中运行，那么必须使用宏定义 DRAM_ATTR 将该 ISR 用到所有常量数据和调用的函数（包括但不限于 const char 数组）放入 DRAM 中。
- 可以将一些时间关键的代码放在 IRAM 中，这样可以缩减从 Flash 加载代码所消耗的时间。ESP32 是通过 32kB 的高速缓存来从外部 Flash 中读取代码和数据的，将函数放在 IRAM 中运行可以减少由高速缓存未命中引起的时间延迟。

IROM（代码从 Flash 中运行）

如果一个函数没有被显式地声明放在 IRAM 或者 RTC 内存中，则将其置于 Flash 中。Flash 技术参考手册中介绍了 Flash MMU 允许代码从 Flash 执行的机制。ESP-IDF 将从 Flash 中执行的代码放在 0x400D0000—0x40400000 区域的开始，在启动阶段，二级引导程序会初始化 Flash MMU，将代码在 Flash 中的位置映射到这个区域的开头。对这个区域的访问会被透明地缓存到 0x40070000 —0x40080000 范围内的两个 32kB 的块中。

请注意，使用 Window ABI CALLx 指令可能无法访问 0x40000000 —0x40400000 区域以外的代码，所以要特别留意应用程序是否使用了 0x40400000 —0x40800000 或者 0x40800000 —0x40C00000 区域，ESP-IDF 默认不会使用这两个区域。

RTC 快速内存

从深度睡眠模式唤醒后必须要运行的代码要放在 RTC 内存中，更多信息请查阅文档深度睡眠 。

DRAM（数据 RAM）

链接器将非常量静态数据和零初始化数据放入 0x3FFB0000 — 0x3FFF0000 这 256kB 的区域。注意，如果使用蓝牙堆栈，此区域会减少 64kB（通过将起始地址移至 0x3FFC0000 ）。如果使用了内存跟踪的功能，该区域的长度还要减少 16kB 或者 32kB。放置静态数据后，留在此区域中的剩余空间都用作运行时堆。

常量数据也可以放在 DRAM 中，例如，用在 ISR 中的常量数据（参见上面 IRAM 部分的介绍），为此需要使用 DRAM_ATTR 宏来声明。

```
DRAM_ATTR const char[] format_string = "%p %x";
char buffer[64];
sprintf(buffer, format_string, ptr, val);
```

毋庸置疑，不建议在 ISR 中使用 printf 和其余输出函数。出于调试的目的，可以在 ISR 中使用 ESP_EARLY_LOGx 来输出日志，不过要确保将 TAG 和格式字符串都放在了 DRAM 中。

宏 __NOINIT_ATTR 可以用来声明将数据放在 .noinit 段中，放在此段中的数据不会在启动时被初始化，并且在软件重启后会保留原来的值。

例子：

```
__NOINIT_ATTR uint32_t noinit_data;
```

DROM（数据存储在 Flash 中）

默认情况下，链接器将常量数据放入一个 4MB 区域 (0x3F400000 — 0x3F800000)，该区域用于通过 Flash MMU 和高速缓存来访问外部 Flash。一种特例情况是，字面量会被编译器嵌入到应用程序代码中。

RTC 慢速内存

从 RTC 内存运行的代码（例如深度睡眠模块的代码）使用的全局和静态变量必须要放在 RTC 慢速内存中。更多详细说明请查看文档深度睡眠 。

宏 RTC_NOINIT_ATTR 用来声明将数据放入 RTC 慢速内存中，该数据在深度睡眠唤醒后将保持不变。

例子：

```
RTC_NOINIT_ATTR uint32_t rtc_noinit_data;
```

5.1.3 DMA 能力要求

大多数的 DMA 控制器（比如 SPI，SDMMC 等）都要求发送/接收缓冲区放在 DRAM 中，并且按字对齐。我们建议将 DMA 缓冲区放在静态变量中而不是堆栈中。使用 DMA_ATTR 宏可以声明该全局/本地的静态变量具备 DMA 能力，例如：

```
DMA_ATTR uint8_t buffer[]="I want to send something";

void app_main()
{
    // 初始化代码...
    spi_transaction_t temp = {
```

(下页继续)

(续上页)

```
        .tx_buffer = buffer,
        .length = 8*sizeof(buffer),
    };
    spi_device_transmit( spi, &temp );
    // 其他程序
}
```

或者：

```
void app_main()
{
    DMA_ATTR static uint8_t buffer[]="I want to send something";
    // 初始化代码...
    spi_transaction_t temp = {
        .tx_buffer = buffer,
        .length = 8*sizeof(buffer),
    };
    spi_device_transmit( spi, &temp );
    // 其他程序
}
```

在堆栈中放置 DMA 缓冲区仍然是允许的，但是你必须记住：

1. 如果堆栈在 pSRAM 中，切勿尝试这么做，因为堆栈在 pSRAM 中的话就要按照片外 *SRAM* 文档介绍的步骤来操作（至少要在 menuconfig 中使能 SPIRAM_ALLOW_STACK_EXTERNAL_MEMORY），所以请确保你的任务不在 pSRAM 中。
2. 在函数中使用 WORD_ALIGNED_ATTR 宏来修饰变量，将其放在适当的位置上，比如：

```
void app_main()
{
    uint8_t stuff;
    WORD_ALIGNED_ATTR uint8_t buffer[]="I want to send something";   //否则 buffer 数组会被存储在 stuff 变量的后面
    // 初始化代码...
    spi_transaction_t temp = {
        .tx_buffer = buffer,
        .length = 8*sizeof(buffer),
    };
    spi_device_transmit( spi, &temp );
    // 其他程序
}
```

5.2 构建系统（CMake 版）

[English]

本文档将介绍基于 CMake 的 ESP-IDF 构建系统的实现原理以及 组件等相关概念，此外 ESP-IDF 还支持基于 *GNU Make* 的构建系统。

如需您想了解如何使用 CMake 构建系统来组织和构建新的 ESP-IDF 项目或组件，请阅读本文档。

5.2.1 概述

一个 ESP-IDF 项目可以看作是多个不同组件的集合，例如一个显示当前湿度的网页服务器会包含以下组件：

- ESP32 基础库，包括 libc、ROM bindings 等
- Wi-Fi 驱动
- TCP/IP 协议栈
- FreeRTOS 操作系统
- 网页服务器
- 湿度传感器的驱动
- 负责将上述组件整合到一起的主程序

ESP-IDF 可以显式地指定和配置每个组件。在构建项目的时候，构建系统会前往 ESP-IDF 目录、项目目录和用户自定义目录（可选）中查找所有组件，允许用户通过文本菜单系统配置 ESP-IDF 项目中用到的每个组件。在所有组件配置结束后，构建系统开始编译整个项目。

概念

- 项目特指一个目录，其中包含了构建可执行应用程序所需的全部文件和配置，以及其他支持型文件，例如分区表、数据/文件系统分区和引导程序。
- 项目配置保存在项目根目录下名为 `sdkconfig` 的文件中，可以通过 `idf.py menuconfig` 进行修改，且一个项目只能包含一个项目配置。
- 应用程序是由 ESP-IDF 构建得到的可执行文件。一个项目通常会构建两个应用程序：项目应用程序（可执行的主文件，即用户自定义的固件）和引导程序（启动并初始化项目应用程序）。
- 组件是模块化且独立的代码，会被编译成静态库（.a 文件）并链接到应用程序。部分组件由 ESP-IDF 官方提供，其他组件则来源于其它开源项目。
- 目标特指运行构建后应用程序的硬件设备。ESP-IDF 当前仅支持 `ESP32` 这一个硬件目标。

请注意，以下内容并不属于项目的组成部分：

- `ESP-IDF` 并不是项目的一部分，它独立于项目，通过 `IDF_PATH` 环境变量（保存 `esp-idf` 目录的路径）链接到项目，从而将 IDF 框架与项目分离。

- 交叉编译工具链并不是项目的组成部分，它应该被安装在系统 PATH 环境变量中。

5.2.2 使用构建系统

idf.py

`idf.py` 命令行工具提供了一个前端，可以帮助您轻松管理项目的构建过程，它管理了以下工具：

- CMake，配置待构建的系统
- 命令行构建工具（Ninja 或 *GNU Make*）
- esptool.py，烧录 ESP32

入门指南 简要介绍了如何设置 `idf.py` 用于配置、构建并烧录项目。

`idf.py` 应运行在 ESP-IDF 的 项目目录下，即包含 `CMakeLists.txt` 文件的目录。仅包含 Makefile 的老式项目并不支持 `idf.py`。

运行 `idf.py --help` 查看完整的命令列表。下面总结了最常用的命令：

- `idf.py menuconfig` 会运行 `menuconfig` 工具来配置项目。
- `idf.py build` 会构建在当前目录下找到的项目，它包括以下步骤：
  - 根据需要创建 `build` 构建目录，它用于保存构建过程的输出文件，可以使用 `-B` 选项修改默认的构建目录。
  - 根据需要运行 CMake 配置命令，为主构建工具生成构建文件。
  - 运行主构建工具（Ninja 或 *GNU Make*）。默认情况下，构建工具会被自动检测，可以使用 `-G` 选项显式地指定构建工具。

  构建过程是增量式的，如果自上次构建以来源文件或项目配置没有发生改变，则不会执行任何操作。
- `idf.py clean` 会把构建输出的文件从构建目录中删除，从而清理整个项目。下次构建时会强制“重新完整构建”这个项目。清理时，不会删除 CMake 配置输出及其他文件。
- `idf.py fullclean` 会将整个 `build` 目录下的内容全部删除，包括所有 CMake 的配置输出文件。下次构建项目时，CMake 会从头开始配置项目。请注意，该命令会递归删除构建目录下的 *所有文件*，请谨慎使用。项目配置文件不会被删除。
- `idf.py flash` 会在必要时自动构建项目，并将生成的二进制程序烧录进 ESP32 设备中。`-p` 和 `-b` 选项可分别设置串口的设备名和烧录时的波特率。
- `idf.py monitor` 用于显示 ESP32 设备的串口输出。`-p` 选项可用于设置主机端串口的设备名，按下 `Ctrl-]` 可退出监视器。更多有关监视器的详情，请参阅*IDF 监视器*。

多个 `idf.py` 命令可合并成一个，例如，`idf.py -p COM4 clean flash monitor` 会依次清理源码树，构建项目，烧录进 ESP32 设备，最后运行串口监视器。

注解: 环境变量 ESPPORT 和 ESPBAUD 可分别用作 -p 和 -b 选项的默认值。在命令行中，重新为这两个选项赋值，会覆盖其默认值。

高级命令

- idf.py app，idf.py bootloader，idf.py partition_table 仅可用于从适用的项目中构建应用程序、引导程序或分区表。
- idf.py app-flash 等匹配命令，仅将项目的特定部分烧录至 ESP32。
- idf.py -p PORT erase_flash 会使用 esptool.py 擦除 ESP32 的整个 Flash。
- idf.py size 会打印应用程序相关的大小信息，idf.py size-components 和 idf.py size-files 这两个命令相似，分别用于打印每个组件或源文件的详细信息。
- idf.py reconfigure 命令会重新运行 CMake（即便无需重新运行）。正常使用时，并不需要运行此命令，但当源码树中添加/删除文件后或更改 CMake cache 变量时，此命令会非常有用，例如，idf.py -DNAME='VALUE' reconfigure 会将 CMake cache 中的变量 NAME 的值设置为 VALUE。

同时调用多个 idf.py 命令时，命令的输入顺序并不重要，它们会按照正确的顺序依次执行，并保证每一条命令都生效（即先构建后烧录，先擦除后烧录等）。

直接使用 CMake

为了方便，*idf.py* 已经封装了 CMake 命令，但是您愿意，也可以直接调用 CMake。

当 idf.py 在执行某些操作时，它会打印出其运行的每条命令以便参考。例如运行 idf.py build 命令与在 bash shell（或者 Windows Command Prompt）中运行以下命令是相同的:

```
mkdir -p build
cd build
cmake .. -G Ninja   # 或者 'Unix Makefiles'
ninja
```

在上面的命令列表中，cmake 命令对项目进行配置，并生成用于最终构建工具的构建文件。在这个例子中，最终构建工具是 Ninja: 运行 ninja 来构建项目。

没有必要多次运行 cmake。第一次构建后，往后每次只需运行 ninja 即可。如果项目需要重新配置，ninja 会自动重新调用 cmake。

若在 CMake 中使用 ninja 或 make，则多数 idf.py 子命令也会有其对应的目标，例如在构建目录下运行 make menuconfig 或 ninja menuconfig 与运行 idf.py menuconfig 是相同的。

注解： 如果您已经熟悉了 CMake，那么可能会发现 ESP-IDF 的 CMake 构建系统不同寻常，为了减少样板文件，该系统封装了 CMake 的许多功能。请参考*编写纯 CMake 组件* 以编写更多 `CMake` 风格的组件。

使用 Ninja/Make 来烧录

您可以直接使用 ninja 或 make 运行如下命令来构建项目并烧录:

```
ninja flash
```

或:

```
make app-flash
```

可用的目标还包括：`flash`、`app-flash` （仅用于 app）、`bootloader-flash` （仅用于 bootloader）。

以这种方式烧录时，可以通过设置 `ESPPORT` 和 `ESPBAUD` 环境变量来指定串口设备和波特率。您可以在操作系统或 IDE 项目中设置该环境变量，或者直接在命令行中进行设置:

```
ESPPORT=/dev/ttyUSB0 ninja flash
```

注解： 在命令的开头为环境变量赋值属于 Bash shell 的语法，可在 Linux 、macOS 和 Windows 的类 Bash shell 中运行，但在 Windows Command Prompt 中无法运行。

或:

```
make -j3 app-flash ESPPORT=COM4 ESPBAUD=2000000
```

注解： 在命令末尾为变量赋值属于 `make` 的语法，适用于所有平台的 `make`。

在 IDE 中使用 CMake

您还可以使用集成了 CMake 的 IDE，仅需将项目 `CMakeLists.txt` 文件的路径告诉 IDE 即可。集成 CMake 的 IDE 通常会有自己的构建工具（CMake 称之为 “生成器”），它是组成 IDE 的一部分，用来构建源文件。

向 IDE 中添加除 `build` 目标以外的自定义目标（如添加 “Flash” 目标到 IDE）时，建议调用 `idf.py` 命令来执行这些 “特殊” 的操作。

有关将 ESP-IDF 同 CMake 集成到 IDE 中的详细信息，请参阅*构建系统的元数据*。

设置 Python 解释器

目前，ESP-IDF 仅适用于 Python 2.7，如果系统中默认的 `python` 解释器是 Python 3.x，可能会出现问题。

如果使用了 `idf.py`，并以 `python2 $IDF_PATH/tools/idf.py ...` 的方式运行 `idf.py` 则会解决这个问题（`idf.py` 会通知其余 Python 进程使用相同的 Python 解释器）。你可以通过设置 shell 别名或其他脚本来简化该命令。

如果直接使用 CMake，运行 `cmake -D PYTHON=python2 ...`，CMake 会使用传入的值覆盖默认的 Python 解释器。

如果使用集成 CMake 的 IDE，可以在 IDE 的图形用户界面中给名为 `PYTHON` 的 CMake cache 变量设置新的值来覆盖默认的 Python 解释器。

如果想在命令行中更优雅地管理 Python 的各个版本，请查看 pyenv 或 virtualenv 工具，它们会帮助您更改默认的 python 版本。

5.2.3 示例项目

示例项目的目录树结构可能如下所示:

```
- myProject/
            - CMakeLists.txt
            - sdkconfig
            - components/ - component1/ - CMakeLists.txt
                                        - Kconfig
                                        - src1.c
                          - component2/ - CMakeLists.txt
                                        - Kconfig
                                        - src1.c
                                        - include/ - component2.h
            - main/       - src1.c
                          - src2.c
            - build/
```

该示例项目 `myproject` 包含以下组成部分:

- 顶层项目 CMakeLists.txt 文件，这是 CMake 用于学习如何构建项目的主要文件，可以在这个文件中设置项目全局的 CMake 变量。顶层项目 CMakeLists.txt 文件会导入 /tools/cmake/project.cmake 文件，由它负责实现构建系统的其余部分。该文件最后会设置项目的名称，并定义该项目。
- `sdkconfig` 项目配置文件，执行 `idf.py menuconfig` 时会创建或更新此文件，文件中保存了项目中所有组件（包括 ESP-IDF 本身）的配置信息。`sdkconfig` 文件可能会也可能不会被添加到项目的源码管理系统中。

- 可选的 `component` 目录中包含了项目的部分自定义组件，并不是每个项目都需要这种自定义组件，但它组件有助于构建可复用的代码或者导入第三方（不属于 ESP-IDF）的组件。
- `main` 目录是一个特殊的 *伪组件*，包含项目本身的源代码。`main` 是默认名称，CMake 变量 `COMPONENT_DIRS` 默认包含此组件，但您可以修改此变量。或者，您也可以在顶层 CMakeLists.txt 中设置 `EXTRA_COMPONENT_DIRS` 变量以查找其他指定位置处的组件。有关详细信息，请参阅*重命名 main 组件*。如果项目中源文件较多，建议将其归于组件中，而不是全部放在 `main` 中。
- `build` 目录是存放构建输出的地方，如果没有此目录，`idf.py` 会自动创建。CMake 会配置项目，并在此目录下生成临时的构建文件。随后，在主构建进程的运行期间，该目录还会保存临时目标文件、库文件以及最终输出的二进制文件。此目录通常不会添加到项目的源码管理系统中，也不会随项目源码一同发布。

每个组件目录都包含一个 `CMakeLists.txt` 文件，里面会定义一些变量以控制该组件的构建过程，以及其与整个项目的集成。更多详细信息请参阅*组件 CMakeLists 文件*。

每个组件还可以包含一个 `Kconfig` 文件，它用于定义 `menuconfig` 时展示的*组件配置* 选项。某些组件可能还会包含 `Kconfig.projbuild` 和 `project_include.cmake` 特殊文件，它们用于*覆盖项目的部分设置*。

5.2.4 项目 CMakeLists 文件

每个项目都有一个顶层 `CMakeLists.txt` 文件，包含整个项目的构建设置。默认情况下，项目 CMakeLists 文件会非常小。

最小 CMakeLists 文件示例

最小项目：

```
cmake_minimum_required(VERSION 3.5)
include($ENV{IDF_PATH}/tools/cmake/project.cmake)
project(myProject)
```

必要部分

每个项目都要按照上面显示的顺序添加上述三行代码：

- `cmake_minimum_required(VERSION 3.5)` 必须放在 CMakeLists.txt 文件的第一行，它会告诉 CMake 构建该项目所需要的最小版本号。ESP-IDF 支持 CMake 3.5 或更高的版本。
- `include($ENV{IDF_PATH}/tools/cmake/project.cmake)` 会导入 CMake 的其余功能来完成配置项目、检索组件等任务。
- `project(myProject)` 会创建项目本身，并指定项目名称。该名称会作为最终输出的二进制文件的名字，即 `myProject.elf` 和 `myProject.bin`。每个 CMakeLists 文件只能定义一个项目。

可选的项目变量

以下这些变量都有默认值，用户可以覆盖这些变量值以自定义构建行为。更多实现细节，请参阅 /tools/cmake/project.cmake 文件。

- COMPONENT_DIRS：组件的搜索目录，默认为 ${IDF_PATH}/components、${PROJECT_PATH}/components 和 EXTRA_COMPONENT_DIRS。如果您不想在这些位置搜索组件，请覆盖此变量。
- EXTRA_COMPONENT_DIRS：用于搜索组件的其它可选目录列表。路径可以是相对于项目目录的相对路径，也可以是绝对路径。
- COMPONENTS：要构建进项目中的组件名称列表，默认为 COMPONENT_DIRS 目录下检索到的所有组件。使用此变量可以“精简”项目以缩短构建时间。请注意，如果一个组件通过 COMPONENT_REQUIRES 指定了它依赖的另一个组件，则会自动将其添加到 COMPONENTS 中，所以 COMPONENTS 列表可能会非常短。
- COMPONENT_REQUIRES_COMMON：每个组件都需要的通用组件列表，这些通用组件会自动添加到每个组件的 COMPONENT_PRIV_REQUIRES 列表中以及项目的 COMPONENTS 列表中。默认情况下，此变量设置为 ESP-IDF 项目所需的最小核心“系统”组件集。通常您无需在项目中更改此变量。

以上变量中的路径可以是绝对路径，或者是相对于项目目录的相对路径。

请使用 cmake 中的 set 命令 来设置这些变量，即 set(VARIABLE "VALUE")。请注意，set() 命令需放在 include(...) 之前，cmake_minimum(...) 之后。

重命名 main 组件

构建系统会对 main 组件进行特殊处理。假如 main 组件位于预期的位置（即 *${PROJECT_PATH}/main*），那么它会被自动添加到构建系统中。其他组件也会作为其依赖项被添加到构建系统中，这使用户免于处理依赖关系，并提供即时可用的构建功能。重命名 main 组件会减轻上述这些幕后工作量，但要求用户指定重命名后的组件位置，并手动为其添加依赖项。重命名 main 组件的步骤如下：

1. 重命名 main 目录。
2. 在项目 CMakeLists.txt 文件中设置 EXTRA_COMPONENT_DIRS，并添加重命名后的 main 目录。
3. 在组件的 CMakeLists.txt 文件中设置 COMPONENT_REQUIRES 或 COMPONENT_PRIV_REQUIRES 以指定依赖项。

5.2.5 组件 CMakeLists 文件

每个项目都包含一个或多个组件，这些组件可以是 ESP-IDF 的一部分，可以是项目自身组件目录的一部分，也可以从自定义组件目录添加（见上文）。

组件是 COMPONENT_DIRS 列表中包含 CMakeLists.txt 文件的任何目录。

搜索组件

搜索 COMPONENT_DIRS 中的目录列表以查找项目的组件，此列表中的目录可以是组件自身（即包含 *CMakeLists.txt* 文件的目录），也可以是子目录为组件的顶级目录。

当 CMake 运行项目配置时，它会记录本次构建包含的组件列表，它可用于调试某些组件的添加/排除。

同名组件

ESP-IDF 在搜索所有待构建的组件时，会按照 COMPONENT_DIRS 指定的顺序依次进行，这意味着在默认情况下，首先搜索 ESP-IDF 内部组件，然后是项目组件，最后是 EXTRA_COMPONENT_DIRS 中的组件。如果这些目录中的两个或者多个包含具有相同名字的组件，则使用搜索到的最后一个位置的组件。这就允许将组件复制到项目目录中再修改以覆盖 ESP-IDF 组件，如果使用这种方式，ESP-IDF 目录本身可以保持不变。

最小的组件 CMakeLists 文件

最小组件 CMakeLists.txt 文件内容如下:

```
set(COMPONENT_SRCS "foo.c")
set(COMPONENT_ADD_INCLUDEDIRS "include")
register_component()
```

- COMPONENT_SRCS 是用空格分隔的源文件列表（*.c，*.cpp，*.cc，*.S），里面所有的源文件都将会编译进组件库中。
- COMPONENT_ADD_INCLUDEDIRS 是用空格分隔的目录列表，里面的路径会被添加到所有需要该组件的组件（包括 main 组件）全局 include 搜索路径中。
- register_component() 使用上述设置的变量将组件添加到构建系统中，构建生成与组件同名的库，并最终被链接到应用程序中。如果因为使用了 CMake 中的 if 命令 或类似命令而跳过了这一步，那么该组件将不会被添加到构建系统中。

上述目录通常设置为相对于 CMakeLists.txt 文件的相对路径，当然也可以设置为绝对路径。

有关更完整的 CMakeLists.txt 示例，请参阅组件 *CMakeLists* 示例。

预设的组件变量

以下专用于组件的变量可以在组件 CMakeLists 中使用，但不建议修改:

- COMPONENT_PATH: 组件目录，即包含 CMakeLists.txt 文件的绝对路径，它与 CMAKE_CURRENT_SOURCE_DIR 变量一样，路径中不能包含空格。
- COMPONENT_NAME: 组件名，与组件目录名相同。
- COMPONENT_TARGET: 库目标名，它由构建系统在内部为组件创建。

以下变量在项目级别中被设置，但可在组件 CMakeLists 中使用：

- PROJECT_NAME：项目名，在项目 CMakeLists.txt 文件中设置。
- PROJECT_PATH：项目目录（包含项目 CMakeLists 文件）的绝对路径，与 CMAKE_SOURCE_DIR 变量相同。
- COMPONENTS：此次构建中包含的所有组件的名称，具体格式为用分号隔开的 CMake 列表。
- CONFIG_*：项目配置中的每个值在 cmake 中都对应一个以 CONFIG_ 开头的变量。更多详细信息请参阅*Kconfig*。
- IDF_VER：ESP-IDF 的 git 版本号，由 git describe 命令生成。
- IDF_VERSION_MAJOR, IDF_VERSION_MINOR, IDF_VERSION_PATCH: ESP-IDF 的组件版本，可用于条件表达式。请注意这些信息的精确度不如 IDF_VER 变量，版本号 v4.0-dev-*，v4.0-beta1，v4.0-rc1 和 v4.0 对应的 IDF_VERSION_* 变量值是相同的，但是 IDF_VER 的值是不同的。
- IDF_TARGET：项目的硬件目标名称。
- PROJECT_VER：项目版本号。
  - 如果在项目 CMakeLists.txt 文件中设置了 PROJECT_VER 变量，则该变量值可以使用。
  - 或者，如果 ${PROJECT_PATH}/version.txt 文件存在，其内容会用作 PROJECT_VER 的值。
  - 或者，如果项目位于某个 Git 仓库中，则使用 git describe 命令的输出作为 PROJECT_VER 的值。
  - 否则，PROJECT_VER 的值为空。

如果您在组件的 CMakeLists.txt 中修改以上变量，并不会影响其他组件的构建，但可能会使该组件变得难以构建或调试。

- COMPONENT_ADD_INCLUDEDIRS：相对于组件目录的相对路径，为被添加到所有需要该组件的其他组件的全局 include 搜索路径中。如果某个 include 路径仅仅在编译当前组件时需要，请将其添加到 COMPONENT_PRIV_INCLUDEDIRS 中。
- COMPONENT_REQUIRES 是一个用空格分隔的组件列表，列出了当前组件依赖的其他组件。如果当前组件有一个头文件位于 COMPONENT_ADD_INCLUDEDIRS 目录下，且该头文件包含了另一个组件的头文件，那么这个被依赖的组件需要在 COMPONENT_REQUIRES 中指出。这种依赖关系可以是递归的。

  COMPONENT_REQUIRES 可以为空，因为所有的组件都需要一些常用的组件（如 newlib 组件提供的 libc 库、freertos 组件提供的 RTOS 功能），这些通用组件已经在项目级变量 COMPONENT_REQUIRES_COMMON 中被设置。

  如果一个组件仅需要额外组件的头文件来编译其源文件（而不是全局引入它们的头文件），则这些被依赖的组件需要在 COMPONENT_PRIV_REQUIRES 中指出。

  请参阅组件依赖，查看详细信息。

可选的组件特定变量

以下变量可在 CMakeLists.txt 中进行设置，用以控制该组件的构建行为：

- COMPONENT_PRIV_INCLUDEDIRS：相对于组件目录的相对路径，仅会被添加到该组件的 include 搜索路径中。
- COMPONENT_PRIV_REQUIRES：以空格分隔的组件列表，用于编译或链接当前组件的源文件。这些组件的头文件路径不会传递给其余需要它的组件，仅用于编译当前组件的源代码。更多详细信息请参阅组件依赖。
- COMPONENT_SRCS：要编译进当前组件的源文件的路径，推荐使用此方法向构建系统中添加源文件。
- COMPONENT_SRCDIRS：相对于组件目录的源文件目录路径，用于搜索源文件（*.cpp，*.c，*.S）。匹配成功的源文件会替代 COMPONENT_SRCS 中指定的源文件，进而被编译进组件。即设置 COMPONENT_SRCDIRS 会导致 COMPONENT_SRCS 会被忽略。此方法可以很容易地将源文件整体导入到组件中，但并不推荐使用（详情请参阅文件通配符 & 增量构建）。
- COMPONENT_SRCEXCLUDE：需要从组件中剔除的源文件路径。当某个目录中有大量的源文件需要被导入组件中，但同时又有个别文件不需要导入时，可以配合 COMPONENT_SRCDIRS 变量一起设置。路径可以是相对于组件目录的相对路径，也可以是绝对路径。
- COMPONENT_ADD_LDFRAGMENTS：组件使用的链接片段文件的路径，用于自动生成链接器脚本文件。详细信息请参阅链接脚本生成机制。

注解： 如果没有设置 COMPONENT_SRCDIRS 或 COMPONENT_SRCS，组件不会被编译成库文件，但仍可以被添加到 include 路径中，以便在编译其他组件时使用。

组件编译控制

在编译特定组件的源文件时，可以使用 target_compile_options 命令来传递编译器选项:

```
target_compile_options(${COMPONENT_LIB} PRIVATE -Wno-unused-variable)
```

这条命令封装了 CMake 的 target_compile_options 命令。

如果给单个源文件指定编译器标志，可以使用 CMake 的 set_source_files_properties 命令:

```
set_source_files_properties(mysrc.c
    PROPERTIES COMPILE_FLAGS
    -Wno-unused-variable
)
```

如果上游代码在编译的时候发出了警告，那这么做可能会很有效。

请注意，上述两条命令只能在组件 CMakeLists 文件的 register_component() 命令之后调用。

5.2.6 组件配置

每个组件都可以包含一个 Kconfig 文件，和 CMakeLists.txt 放在同一目录下。Kconfig 文件中包含要添加到该组件配置菜单中的一些配置设置信息。

运行 menuconfig 时，可以在 Component Settings 菜单栏下找到这些设置。

创建一个组件的 Kconfig 文件，最简单的方法就是使用 ESP-IDF 中现有的 Kconfig 文件作为模板，在这基础上进行修改。

有关示例请参阅添加条件配置。

5.2.7 预处理器定义

ESP-IDF 构建系统会在命令行中添加以下 C 预处理器定义：

- ESP_PLATFORM：可以用来检测在 ESP-IDF 内发生了构建行为。
- IDF_VER：定义 git 版本字符串，例如：v2.0 用于标记已发布的版本，v1.0-275-g0efaa4f 则用于标记任意某次的提交记录。
- PROJECT_VER：项目版本号，详细信息请参阅预设的组件变量。
- PROJECT_NAME：项目名称，定义在项目 CMakeLists.txt 文件中。

5.2.8 组件依赖

编译各个组件时，ESP-IDF 系统会递归评估其组件。

每个组件的源文件都会使用以下路径中的头文件进行编译：

- 当前组件的 COMPONENT_ADD_INCLUDEDIRS 和 COMPONENT_PRIV_INCLUDEDIRS。
- 当前组件的 COMPONENT_REQUIRES 和 COMPONENT_PRIV_REQUIRES 变量指定的其他组件（即当前组件的所有公共和私有依赖项）所设置的 COMPONENT_ADD_INCLUDEDIRS。
- 所有组件的 COMPONENT_REQUIRES 做递归操作，即该组件递归运算后的所有公共依赖项。

编写组件

- COMPONENT_REQUIRES 需要包含所有被当前组件的公共头文件 *#include* 的头文件所在的组件。
- COMPONENT_PRIV_REQUIRES 需要包含被当前组件的源文件 *#include* 的头文件所在的组件（除非已经被设置在了 COMPONENT_PRIV_REQUIRES 中）。或者是当前组件正常工作必须要链接的组件。
- COMPONENT_REQUIRES、COMPONENT_PRIV_REQUIRES 需要在调用 register_component() 之前设置。
- COMPONENT_REQUIRES 和 COMPONENT_PRIV_REQUIRES 的值不能依赖于任何配置选项（CONFIG_xxx），这是因为在配置加载之前，依赖关系就已经被展开。其它组件变量（比如 COMPONENT_SRCS 和 COMPONENT_ADD_INCLUDEDIRS）可以依赖配置选择。

- 如果当前组件除了 COMPONENT_REQUIRES_COMMON 中设置的通用组件（比如 RTOS、libc 等）外，并不依赖其它组件，那么上述两个 REQUIRES 变量可以为空。

如果组件仅支持某些硬件目标（即依赖于特定的 IDF_TARGET），则可以调用 require_idf_targets(NAMES...) CMake 函数来声明这个需求。在这种情况下，如果构建系统导入了不支持当前硬件目标的组件时就会报错。

创建项目

- 默认情况下，每个组件都会包含在构建系统中。
- 如果将 COMPONENTS 变量设置为项目直接使用的最小组件列表，那么构建系统会导入：
  - COMPONENTS 中明确提及的组件。
  - 这些组件的依赖项（以及递归运算后的组件）。
  - 每个组件都依赖的通用组件。
- 将 COMPONENTS 设置为所需组件的最小列表，可以显著减少项目的构建时间。

构建系统中依赖处理的实现细节

- 在 CMake 配置进程的早期阶段会运行 expand_requirements.cmake 脚本。该脚本会对所有组件的 CMakeLists.txt 文件进行局部的运算，得到一张组件依赖关系图（此图可能会有闭环）。此图用于在构建目录中生成 component_depends.cmake 文件。
- CMake 主进程会导入该文件，并以此来确定要包含到构建系统中的组件列表（内部使用的 BUILD_COMPONENTS 变量）。BUILD_COMPONENTS 变量已排好序，依赖组件会排在前面。由于组件依赖关系图中可能存在闭环，因此不能保证每个组件都满足该排序规则。如果给定相同的组件集和依赖关系，那么最终的排序结果应该是确定的。
- CMake 会将 BUILD_COMPONENTS 的值以 “Component names:” 的形式打印出来。
- 然后执行构建系统中包含的每个组件的配置。
- 每个组件都被正常包含在构建系统中，然后再次执行 CMakeLists.txt 文件，将组件库加入构建系统。

组件依赖顺序

BUILD_COMPONENTS 变量中组件的顺序决定了构建过程中的其它顺序，包括：

- 项目导入*project_include.cmake* 文件的顺序。
- 生成用于编译（通过 -I 参数）的头文件路径列表的顺序。请注意，对于给定组件的源文件，仅需将该组件的依赖组件的头文件路径告知编译器。

5.2.9 构建的内部过程

关于 CMake 以及 CMake 命令的详细信息，请参阅 CMake v3.5 官方文档 。

project.cmake 的内容

当项目 CMakeLists 文件导入 `project.cmake` 文件时，`project.cmake` 会定义一些实用的模块和全局变量。如果系统环境中没有设置 `IDF_PATH`，那么它还会自动设置 `IDF_PATH` 变量。

`project.cmake` 文件还重写了 CMake 内置的 `project` 函数，以添加所有 ESP-IDF 项目特有的功能。

project 函数

自定义的 `project()` 函数会执行以下步骤：

- 确定硬件目标（由 `IDF_TARGET` 环境变量设置），并将其保存在 CMake cache 中。如果环境变量中设置的硬件目标与 CMake cache 中的不匹配，则会报错并退出。
- 计算组件依赖，并构造 `BUILD_COMPONENTS` 变量，它是包含所有需要导入到构建系统中的组件列表（详情请见上文）。
- 查找项目中所有的组件（搜索 `COMPONENT_DIRS`，并按 `COMPONENTS` 进行过滤（前提是设置了该变量）。
- 从 `sdkconfig` 文件中加载项目配置信息，生成 `sdkconfig.cmake` 和 `sdkconfig.h` 文件，分别用在 CMake 和 C/C++ 中定义配置项。如果项目配置发生了更改，CMake 会自动重新运行，重新生成上述两个文件，接着重新配置项目。
- 根据硬件目标（`IDF_TARGET`）的值，将 CMAKE_TOOLCHAIN_FILE 变量设置为相应的工具链文件。
- 调用 CMake 的 project 函数 声明实际的 CMake-level 项目。
- 加载 git 版本号。如果在 git 中检出了新的版本，就会使用一些技巧重新运行 CMake。详情请参考文件通配符 & 增量构建。
- 从包含有*project_include.cmake* 文件的组件中导入该文件。
- 将每个组件都添加到构建系统中。每个组件的 CMakeLists 文件都会调用 `register_component` 函数，它会调用 CMake 的 add_library 函数来添加一个库，然后添加源文件、编译选项等。
- 将最终的应用程序可执行文件添加到构建系统中。
- 返回并为组件之间指定依赖关系（将每个组件的公共头文件目录添加到其他组件中）。

更多详细信息请参阅 /tools/cmake/project.cmake 文件和 /tools/cmake/idf_functions.cmake 文件。

CMake 调试

调试 ESP-IDF CMake 构建系统的一些技巧：

- CMake 运行时，会打印大量诊断信息，包括组件列表和组件路径。

- 运行 `cmake -DDEBUG=1`，IDF 构建系统会生成更详细的诊断输出。
- 运行 `cmake` 时指定 `--trace` 或 `--trace-expand` 选项会提供大量有关控制流信息。详情请参考 CMake 命令行文档。

警告未定义的变量

默认情况下，`idf.py` 在调用 CMake 时会给它传递 `--warn-uninitialized` 标志，如果在构建的过程中引用了未定义的变量，CMake 会打印警告。这对查找有错误的 CMake 文件非常有用。

如果您不想启用此功能，可以给 `idf.py` 传递 `--no-warnings` 标志。

覆盖项目的部分设置

project_include.cmake

如果组件的某些构建行为需要在组件 CMakeLists 文件之前被执行，您可以在组件目录下创建名为 `project_include.cmake` 的文件，`project.cmake` 在运行过程中会导入此 CMake 文件。

`project_include.cmake` 文件在 ESP-IDF 内部使用，以定义项目范围内的构建功能，比如 `esptool.py` 的命令行参数和 `bootloader` 这个特殊的应用程序。

与组件 `CMakeLists.txt` 文件有所不同，在导入 "project_include.cmake" 文件的时候，当前源文件目录（即 `CMAKE_CURRENT_SOURCE_DIR`）和工作目录为项目目录。如果想获得当前组件的绝对路径，可以使用 `COMPONENT_PATH` 变量。

请注意，`project_include.cmake` 对于大多数常见的组件并不是必需的。例如给项目添加 include 搜索目录，给最终的链接步骤添加 `LDFLAGS` 选项等等都可以通过 `CMakeLists.txt` 文件来自定义。详细信息请参考可选的项目变量。

`project_include.cmake` 文件会按照 `BUILD_COMPONENTS` 变量中组件的顺序（由 CMake 记录）依次导入。即只有在当前组件所有依赖组件的 `project_include.cmake` 文件都被导入后，当前组件的 `project_include.cmake` 文件才会被导入，除非两个组件在同一个依赖闭环中。如果某个 `project_include.cmake` 文件依赖于另一组件设置的变量，则要特别注意上述情况。更多详情请参阅构建系统中依赖处理的实现细节。

在 `project_include.cmake` 文件中设置变量或目标时要格外小心，这些值被包含在项目的顶层 CMake 文件中，因此他们会影响或破坏所有组件的功能。

KConfig.projbuild

与 `project_include.cmake` 类似，也可以为组件定义一个 KConfig 文件以实现全局的组件配置。如果要在 menuconfig 的顶层添加配置选项，而不是在 "Component Configuration" 子菜单中，则可以在 `CMakeLists.txt` 文件所在目录的 KConfig.projbuild 文件中定义这些选项。

在此文件中添加配置时要小心，因为这些配置会包含在整个项目配置中。在可能的情况下，请为组件配置 创建 KConfig 文件。

仅配置组件

仅配置组件是一类不包含源文件的特殊组件，仅包含 `Kconfig.projbuild`、`KConfig` 和 `CMakeLists.txt` 文件，该 `CMakeLists.txt` 文件仅有一行代码，调用了 `register_config_only_component()` 函数。此函数会将组件导入到项目构建中，但不会构建任何库，也不会将头文件添加到任何 include 搜索路径中。

如果 CMakeLists.txt 文件没有调用 `register_component()` 或 `register_config_only_component()`，那么该文件将会被排除在项目构建之外。根据项目的配置，有时可能需要这么做。

5.2.10 组件 CMakeLists 示例

因为构建环境试图设置大多数情况都能工作的合理默认值，所以组件 `CMakeLists.txt` 文件可能非常小，甚至是空的，请参考最小的组件 *CMakeLists* 文件。但有些功能往往需要覆盖预设的组件变量 才能实现。

以下是组件 CMakeLists 文件的更高级的示例。

添加条件配置

配置系统可用于根据项目配置中选择的选项有条件地编译某些文件。

`Kconfig`:

```
config FOO_ENABLE_BAR
    bool "Enable the BAR feature."
    help
        This enables the BAR feature of the FOO component.
```

`CMakeLists.txt`:

```
set(COMPONENT_SRCS "foo.c" "more_foo.c")

if(CONFIG_FOO_ENABLE_BAR)
    list(APPEND COMPONENT_SRCS "bar.c")
endif()
```

上述示例使用了 CMake 的 if 函数和 list APPEND 函数。

也可用于选择或删除某一实现，如下所示：

`Kconfig`:

```
config ENABLE_LCD_OUTPUT
    bool "Enable LCD output."
    help
        Select this if your board has a LCD.
```

(下页继续)

(续上页)

```
config ENABLE_LCD_CONSOLE
    bool "Output console text to LCD"
    depends on ENABLE_LCD_OUTPUT
    help
        Select this to output debugging output to the lcd

config ENABLE_LCD_PLOT
    bool "Output temperature plots to LCD"
    depends on ENABLE_LCD_OUTPUT
    help
        Select this to output temperature plots
```

CMakeLists.txt:

```
if(CONFIG_ENABLE_LCD_OUTPUT)
    set(COMPONENT_SRCS lcd-real.c lcd-spi.c)
else()
    set(COMPONENT_SRCS lcd-dummy.c)
endif()

# 如果启用了控制台或绘图功能，则需要加入字体
if(CONFIG_ENABLE_LCD_CONSOLE OR CONFIG_ENABLE_LCD_PLOT)
    list(APPEND COMPONENT_SRCS "font.c")
endif()
```

硬件目标的的条件判断

CMake 文件可以使用 `IDF_TARGET` 变量来获取当前的硬件目标。

此外，如果当前的硬件目标是 `xyz`` (即 ``IDF_TARGET=xyz`)，那么 Kconfig 变量 `CONFIG_IDF_TARGET_XYZ` 同样也会被设置。

请注意，组件可以依赖 `IDF_TARGET` 变量，但不能依赖这个 Kconfig 变量。同样也不可在 CMake 文件的 `include` 语句中使用 Kconfig 变量，在这种上下文中可以使用 `IDF_TARGET`。

生成源代码

有些组件的源文件可能并不是由组件本身提供，而必须从另外的文件生成。假设组件需要一个头文件，该文件由 BMP 文件转换后（使用 bmp2h 工具）的二进制数据组成，然后将头文件包含在名为 graphics_lib.c 的文件中:

```
add_custom_command(OUTPUT logo.h
    COMMAND bmp2h -i ${COMPONENT_DIR}/logo.bmp -o log.h
    DEPENDS ${COMPONENT_DIR}/logo.bmp
    VERBATIM)

add_custom_target(logo DEPENDS logo.h)
add_dependencies(${COMPONENT_LIB} logo)

set_property(DIRECTORY "${COMPONENT_DIR}" APPEND PROPERTY
    ADDITIONAL_MAKE_CLEAN_FILES logo.h)
```

这个示例改编自 CMake 的一则 FAQ，其中还包含了一些同样适用于 ESP-IDF 构建系统的示例。

这个示例会在当前目录（构建目录）中生成 logo.h 文件，而 logo.bmp 会随组件一起提供在组件目录中。因为 logo.h 是一个新生成的文件，一旦项目需要清理，该文件也应该要被清除。因此，要将该文件添加到 ADDITIONAL_MAKE_CLEAN_FILES 属性中。

注解： 如果需要生成文件作为项目 CMakeLists.txt 的一部分，而不是作为组件 CMakeLists.txt 的一部分，此时需要使用 `${PROJECT_PATH}` 替代 `${COMPONENT_DIR}`，使用 `${PROJECT_NAME}.elf` 替代 `${COMPONENT_LIB}`。

如果某个源文件是从其他组件中生成，且包含 `logo.h` 文件，则需要调用 `add_dependencies`，在这两个组件之间添加一个依赖项，以确保组件源文件按照正确顺序进行编译。

嵌入二进制数据

有时您的组件希望使用一个二进制文件或者文本文件，但是您又不希望将它们重新格式化为 C 源文件，这时，您可以在组件 CMakeLists 中添加 `COMPONENT_EMBED_FILES` 变量，指定要嵌入的文件名称（以空格分隔）：

```
set(COMPONENT_EMBED_FILES server_root_cert.der)
```

或者，如果文件是字符串，则可以设置 `COMPONENT_EMBED_TXTFILES` 变量，把文件的内容转成以 null 结尾的字符串嵌入:

```
set(COMPONENT_EMBED_TXTFILES server_root_cert.pem)
```

文件的内容会被添加到 Flash 的.rodata 段，用户可以通过符号名来访问，如下所示:

```
extern const uint8_t server_root_cert_pem_start[] asm("_binary_server_root_cert_pem_start
↪");
extern const uint8_t server_root_cert_pem_end[]   asm("_binary_server_root_cert_pem_end
↪");
```

符号名会根据文件全名生成，如 COMPONENT_EMBED_FILES 中所示，字符 /、. 等都会被下划线替代。符号名称中的 _binary 前缀由 objcopy 命令添加，对文本文件和二进制文件都是如此。

如果要将文件嵌入到项目中，而非组件中，可以调用 target_add_binary_data 函数:

```
target_add_binary_data(myproject.elf "main/data.bin" TEXT)
```

并这行代码放在项目 CMakeLists.txt 的 project() 命令之后，修改 myproject.elf 为你自己的项目名。如果最后一个参数是 TEXT，那么构建系统会嵌入以 null 结尾的字符串，如果最后一个参数被设置为 BINARY，则将文件内容按照原样嵌入。

有关使用此技术的示例，请参考 protocols/https_request，证书文件的内容会在编译时从.pem 文件中加载。

代码和数据的存放

ESP-IDF 还支持自动生成链接脚本，它允许组件通过链接片段文件定义其代码和数据在内存中的存放位置。构建系统会处理这些链接片段文件，并将处理后的结果扩充进链接脚本，从而指导应用程序二进制文件的链接过程。更多详细信息与快速上手指南，请参阅链接脚本生成机制。

完全覆盖组件的构建过程

当然，在有些情况下，上面提到的方法不一定够用。如果组件封装了另一个第三方组件，而这个第三方组件并不能直接在 ESP-IDF 的构建系统中工作，在这种情况下，就需要放弃 ESP-IDF 的构建系统，改为使用 CMake 的 ExternalProject 功能。组件 CMakeLists 示例如下:

```
# 用于 quirc 的外部构建过程，在源目录中运行并生成 libquirc.a
externalproject_add(quirc_build
    PREFIX ${COMPONENT_DIR}
    SOURCE_DIR ${COMPONENT_DIR}/quirc
    CONFIGURE_COMMAND ""
    BUILD_IN_SOURCE 1
    BUILD_COMMAND make CC=${CMAKE_C_COMPILER} libquirc.a
    INSTALL_COMMAND ""
    )

# 将 libquirc.a 添加到构建系统中
add_library(quirc STATIC IMPORTED GLOBAL)
```

(下页继续)

(续上页)

```
add_dependencies(quirc quirc_build)

set_target_properties(quirc PROPERTIES IMPORTED_LOCATION
    ${COMPONENT_DIR}/quirc/libquirc.a)
set_target_properties(quirc PROPERTIES INTERFACE_INCLUDE_DIRECTORIES
    ${COMPONENT_DIR}/quirc/lib)

set_directory_properties( PROPERTIES ADDITIONAL_MAKE_CLEAN_FILES
    "${COMPONENT_DIR}/quirc/libquirc.a")
```

(上述 CMakeLists.txt 可用于创建名为 `quirc` 的组件，该组件使用自己的 Makefile 构建 quirc 项目。)

- `externalproject_add` 定义了一个外部构建系统。
  - 设置 `SOURCE_DIR`、`CONFIGURE_COMMAND`、`BUILD_COMMAND` 和 `INSTALL_COMMAND`。如果外部构建系统没有配置这一步骤，可以将 `CONFIGURE_COMMAND` 设置为空字符串。在 ESP-IDF 的构建系统中，一般会将 `INSTALL_COMMAND` 变量设置为空。
  - 设置 `BUILD_IN_SOURCE`，即构建目录与源目录相同。否则，您也可以设置 `BUILD_DIR` 变量。
  - 有关 `externalproject_add()` 命令的详细信息，请参阅 ExternalProject。
- 第二组命令添加了一个目标库，指向外部构建系统生成的库文件。为了添加 include 目录，并告知 CMake 该文件的位置，需要再设置一些属性。
- 最后，生成的库被添加到 ADDITIONAL_MAKE_CLEAN_FILES 中。即执行 `make clean` 后会删除该库。请注意，构建系统中的其他目标文件不会被删除。

注解： 当外部构建系统使用 PSRAM 时，请记得将 `-mfix-esp32-psram-cache-issue` 添加到 C 编译器的参数中。关于该标志的更多详细信息，请参考*CONFIG_SPIRAM_CACHE_WORKAROUND*。

ExternalProject 的依赖与构建清理

对于外部项目的构建，CMake 会有一些不同寻常的行为：

- ADDITIONAL_MAKE_CLEAN_FILES 仅在使用 Make 构建系统时有效。如果使用 Ninja 或 IDE 自带的构建系统，执行项目清理时，这些文件不会被删除。
- ExternalProject 会在 clean 运行后自动重新运行配置和构建命令。
- 可以采用以下两种方法来配置外部构建命令：
  1. 将外部 `BUILD_COMMAND` 命令设置为对所有源代码完整的重新编译。如果传递给 `externalproject_add` 命令的 `DEPENDS` 的依赖项发生了改变，或者当前执行的是项目清理操作（即运行了 `idf.py clean`、`ninja clean` 或者 `make clean`），那么就会执行该命令。

2. 将外部 BUILD_COMMAND 命令设置为增量式构建命令，并给 externalproject_add 传递 BUILD_ALWAYS 1 参数。即不管实际的依赖情况，每次构建时，都会构建外部项目。这种方式仅当外部构建系统具备增量式构建的能力，且运行时间不会很长时才推荐。

构建外部项目的最佳方法取决于项目本身、其构建系统，以及是否需要频繁重新编译项目。

5.2.11 自定义 sdkconfig 的默认值

对于示例工程或者其他您不想指定完整 sdkconfig 配置的项目，但是您确实希望覆盖 ESP-IDF 默认值中的某些键值，则可以在项目中创建 sdkconfig.defaults 文件。重新创建新配置时将会用到此文件，另外在 sdkconfig 没有设置新配置值时，上述文件也会被用到。

如若需要覆盖此文件的名称，请设置 SDKCONFIG_DEFAULTS 环境变量。

依赖于硬件目标的 sdkconfig 默认值

除了 sdkconfig.defaults 之外，构建系统还将从 sdkconfig.defaults.TARGET_NAME 文件加载默认值，其中 IDF_TARGET 的值为 TARGET_NAME。例如，对于 ESP32 这个硬件目标，sdkconfig 的默认值会首先从 sdkconfig.defaults 获取，然后再从 sdkconfig.defaults.esp32 获取。

如果使用 SDKCONFIG_DEFAULTS 覆盖了 sdkconfig 默认文件的名称，则硬件目标的 sdkconfig 默认文件名也会从 SDKCONFIG_DEFAULTS 值中派生。

5.2.12 Flash 参数

有些情况下，我们希望在没有 IDF 时也能烧写目标板卡，为此，我们希望可以保存已构建的二进制文件、esptool.py 和 esptool write_flash 命令的参数。可以通过编写一段简单的脚本来保存二进制文件和 esptool.py。

运行项目构建之后，构建目录将包含项目二进制输出文件（.bin 文件），同时也包含以下烧录数据文件：

- flash_project_args 包含烧录整个项目的参数，包括应用程序 (app)、引导程序 (bootloader)、分区表，如果设置了 PHY 数据，也会包含此数据。
- flash_app_args 只包含烧录应用程序的参数。
- flash_bootloader_args 只包含烧录引导程序的参数。

您可以参照如下命令将任意烧录参数文件传递给 esptool.py:

```
python esptool.py --chip esp32 write_flash @build/flash_project_args
```

也可以手动复制参数文件中的数据到命令行中执行。

构建目录中还包含生成的 flasher_args.json 文件，此文件包含 JSON 格式的项目烧录信息，可用于 idf.py 和其它需要项目构建信息的工具。

5.2.13 构建 Bootloader

引导程序默认作为 `idf.py build` 的一部分被构建，也可以通过 `idf.py bootloader` 来单独构建。

引导程序是 /components/bootloader/subproject 内部独特的“子项目”，它有自己的项目 CMakeLists.txt 文件，能够构建独立于主项目的 `.ELF` 和 `.BIN` 文件，同时它又与主项目共享配置和构建目录。

子项目通过 /components/bootloader/project_include.cmake 文件作为外部项目插入到项目的顶层，主构建进程会运行子项目的 CMake，包括查找组件（主项目使用的组件的子集），生成引导程序专用的配置文件（从主 `sdkconfig` 文件中派生）。

5.2.14 选择硬件目标

当前 ESP-IDF 仅支持一个硬件目标，即 `esp32`，这也是构建系统默认的硬件目标。开发人员可以按照如下方法来添加对新硬件目标的支持:

```
rm sdkconfig
idf.py -DIDF_TARGET=new_target reconfigure
```

5.2.15 编写纯 CMake 组件

ESP-IDF 构建系统用“组件”的概念“封装”了 CMake，并提供了很多帮助函数来自动将这些组件集成到项目构建当中。

然而，“组件”概念的背后是一个完整的 CMake 构建系统，因此可以制作纯 CMake 组件。

下面是使用纯 CMake 语法为 `json` 组件编写的最小 CMakeLists 文件的示例:

```
add_library(json STATIC
cJSON/cJSON.c
cJSON/cJSON_Utils.c)

target_include_directories(json PUBLIC cJSON)
```

- 这实际上与 IDF 中的 json 组件 是等效的。
- 因为组件中的源文件不多，所以这个 CMakeLists 文件非常简单。对于具有大量源文件的组件而言，ESP-IDF 支持的组件通配符，可以简化组件 CMakeLists 的样式。
- 每当组件中新增一个与组件同名的库目标时，ESP-IDF 构建系统会自动将其添加到构建中，并公开公共的 include 目录。如果组件想要添加一个与组件不同名的库目标，就需要使用 CMake 命令手动添加依赖关系。

5.2.16 组件中使用第三方 CMake 项目

CMake 在许多开源的 C/C++ 项目中广泛使用，用户可以在自己的应用程序中使用开源代码。CMake 构建系统的一大好处就是可以导入这些第三方的项目，有时候甚至不用做任何改动。这就允许用户使用当前 ESP-IDF 组件尚未提供的功能，或者使用其它库来实现相同的功能。

假设 `main` 组件需要导入一个假想库 `foo`，相应的组件 CMakeLists 文件如下所示:

```
# 注册组件
register_component()

# 设置 `foo` 项目中的一些 CMake 变量，以控制 `foo` 的构建过程
set(FOO_BUILD_STATIC OFF)
set(FOO_BUILD_TESTS OFF)

# 创建并导入第三方库目标
add_subdirectory(foo)

# 将 IDF 全局的编译器设置、宏定义及其它选项传递给 `foo` 目标
target_include_directories(foo ${IDF_INCLUDE_DIRECTORIES})
target_compile_options(foo ${IDF_COMPILE_OPTIONS})
target_compile_definitions(foo ${IDF_COMPILE_DEFINITIONS})

# 将 `foo` 目标链接至 `main` 组件
target_link_libraries(main foo)
```

实际的案例请参考 build_system/cmake/import_lib。请注意，导入第三方库所需要做的工作可能会因库的不同而有所差异。建议仔细阅读第三方库的文档，了解如何将其导入到其它项目中。阅读第三方库的 CMakeLists.txt 文件以及构建结构也会有所帮助。

用这种方式还可以将第三方库封装成 ESP-IDF 的组件。例如 mbedtls 组件就是封装了 mbedtls 项目 得到的。详情请参考 mbedtls 组件的 CMakeLists.txt 文件 。

每当使用 ESP-IDF 构建系统时，CMake 变量 `ESP_PLATFORM` 都会被设置为 1。如果要在通用的 CMake 代码加入 IDF 特定的代码时，可以采用 `if (ESP_PLATFORM)` 的形式加以分隔。

5.2.17 在自定义 CMake 项目中使用 ESP-IDF

ESP-IDF 提供了一个模板 CMake 项目，可以基于此轻松创建应用程序。然而在有些情况下，用户可能已有一个现成的 CMake 项目，或者想自己创建一个 CMake 项目，此时就希望将 IDF 中的组件以库的形式链接到用户目标（库/可执行文件）。

使用 tools/cmake/idf_functions.cmake 中提供的 `idf_import_components` 和 `idf_link_components` 函数可以实现上述功能，例如:

```
cmake_minimum_required(VERSION 3.5)
project(my_custom_app C)

# 源文件 main.c 包含有 app_main() 函数的定义
add_executable(${CMAKE_PROJECT_NAME}.elf main.c)

# 提供 idf_import_components 及 idf_link_components 函数
include($ENV{IDF_PATH}/tools/cmake/idf_functions.cmake)

# 为 idf_import_components 做一些配置
# 使能创建构件（不是每个项目都必须）
set(IDF_BUILD_ARTIFACTS ON)
set(IDF_PROJECT_EXECUTABLE ${CMAKE_PROJECT_NAME}.elf)
set(IDF_BUILD_ARTIFACTS_DIR ${CMAKE_BINARY_DIR})

# idf_import_components 封装了 add_subdirectory()，为组件创建库目标，然后使用给定的变量接收
“返回”的库目标。
# 在本例中，返回的库目标被保存在“component”变量中。
idf_import_components(components $ENV{IDF_PATH} esp-idf)

# idf_link_components 封装了 target_link_libraries()，将被 idf_import_components 处理过的
组件链接到目标
idf_link_components(${CMAKE_PROJECT_NAME}.elf "${components}")
```

上述代码片段导入了 ESP-IDF 目录下的所有组件，并使用了 KConfig 中的默认值，同时还允许创建其它一些构件（比如分区表、包含项目信息的 json 文件、引导程序等）。除此以外，用户还可以设置其它的构建参数，其完整列表如下：

- IDF_BUILD_ARTIFACTS：构建工件，例如引导加载程序、分区表二进制文件、分区二进制数据、将二进制文件烧录到目标芯片时所需的包含项目信息的 json 文件等。同时需要设置 IDF_PROJECT_EXECUTABLE 和 IDF_BUILD_ARTIFACTS_DIR 变量。
- IDF_PROJECT_EXECUTABLE：最终可执行文件的名称。某些工件在创建的时候需要此参数。
- IDF_BUILD_ARTIFACTS_DIR：创建的构件被存放的位置。
- IDF_EXTRA_COMPONENTS_DIR：在 默认组件目录 之外的组件搜索路径。
- IDF_COMPONENTS：要导入的组件列表，设置此变量可以精简导入的组件，仅导入需要的组件，加快构建的速度。如果没有设置该变量，将会导入默认组件目录以及 IDF_EXTRA_COMPONENTS_DIR（如果设置了该变量）中找到的所有组件。请注意，该列表中组件的依赖组件（除了 IDF_COMPONENT_REQUIRES_COMMON 之外）也会被加入到构建之中。
- IDF_COMPONENT_REQUIRES_COMMON：通用组件依赖列表。无论 IDF_COMPONENTS 的值是什么，此列表中的组件及其依赖组件都会被导入到构建中。默认情况下，此变量被设置为核心“系统”组件的最小集

合。

- IDF_SDKCONFIG_DEFAULTS：配置文件的覆盖路径，如果未设置，组件将会使用默认的配置选项来构建。
- IDF_BUILD_TESTS：在构建中包含组件的测试。默认情况下，所有的组件测试都会被包含。组件测试可通过 IDF_TEST_COMPONENTS 和 IDF_TEST_EXCLUDE_COMPONENTS 进行过滤。
- IDF_TEST_COMPONENTS：如果设置了 IDF_BUILD_TESTS，构建中只会包含此列表中的组件测试。如果没有设置 IDF_BUILD_TESTS，请忽略此项。
- IDF_TEST_EXCLUDE_COMPONENTS：如果设置了 IDF_BUILD_TESTS，此列表中的组件测试将不会包含在构建中。如果没有设置 IDF_BUILD_TESTS，请忽略此项。该变量的优先级高于 IDF_TEST_COMPONENTS，这意味着，即使 IDF_TEST_COMPONENTS 中也存在此列表中的组件测试，它也不会被包含到构建之中。

build_system/cmake/idf_as_lib 中的示例演示了如何在自定义的 CMake 项目创建一个类似于 Hello World 的应用程序。

5.2.18 文件通配符 & 增量构建

在 ESP-IDF 组件中添加源文件的首选方法是在 COMPONENT_SRCS 中手动列出它们:

```
set(COMPONENT_SRCS library/a.c library/b.c platform/platform.c)
```

这是在 CMake 中手动列出源文件的 最佳实践。然而，当有许多源文件都需要添加到构建中时，这种方法就会很不方便。ESP-IDF 构建系统因此提供了另一种替代方法，即使用 COMPONENT_SRCDIRS 来指定源文件:

```
set(COMPONENT_SRCDIRS library platform)
```

后台会使用通配符在指定的目录中查找源文件。但是请注意，在使用这种方法的时候，如果组件中添加了一个新的源文件，CMake 并不知道重新运行配置，最终该文件也没有被加入构建中。

如果是自己添加的源文件，这种折衷还是可以接受的，因为用户可以触发一次干净的构建，或者运行 idf.py reconfigure 来手动重启 CMake。但是，如果你需要与其他使用 Git 等版本控制工具的开发人员共享项目时，问题就会变得更加困难，因为开发人员有可能会拉取新的版本。

ESP-IDF 中的组件使用了第三方的 Git CMake 集成模块(/tools/cmake/third_party/GetGitRevisionDescription.cmake)，任何时候源码仓库的提交记录发生了改变，该模块就会自动重新运行 CMake。即只要拉取了新的 ESP-IDF 版本，CMake 就会重新运行。

对于不属于 ESP-IDF 的项目组件，有以下几个选项供参考:

- 如果项目文件保存在 Git 中，ESP-IDF 会自动跟踪 Git 修订版本，并在它发生变化时重新运行 CMake。
- 如果一些组件保存在第三方 Git 仓库中（不在项目仓库或 ESP-IDF 仓库），则可以在组件 CMakeLists 文件中调用 git_describe 函数，以便在 Git 修订版本发生变化时自动重启 CMake。
- 如果没有使用 Git，请记住在源文件发生变化时手动运行 idf.py reconfigure。
- 使用 COMPONENT_SRCS 在项目组件中列出所有源文件，可以完全避免这一问题。

具体选择哪一方式，就要取决于项目本身，以及项目用户。

5.2.19 构建系统的元数据

为了将 ESP-IDF 集成到 IDE 或者其它构建系统中，CMake 在构建的过程中会在 `build/` 目录下生成大量元数据文件。运行 `cmake` 或 `idf.py reconfigure` （或任何其它 `idf.py` 构建命令），可以重新生成这些元数据文件。

- `compile_commands.json` 是标准格式的 JSON 文件，它描述了在项目中参与编译的每个源文件。CMake 其中的一个功能就是生成此文件，许多 IDE 都知道如何解析此文件。
- `project_description.json` 包含有关 ESP-IDF 项目、已配置路径等的一些常规信息。
- `flasher_args.json` 包含 esptool.py 工具用于烧录项目二进制文件的参数，此外还有 `flash_*_args` 文件，可直接与 esptool.py 一起使用。更多详细信息请参阅*Flash 参数*。
- `CMakeCache.txt` 是 CMake 的缓存文件，包含 CMake 进程、工具链等其它信息。
- `config/sdkconfig.json` 包含 JSON 格式的项目配置结果。
- `config/kconfig_menus.json` 是在 menuconfig 中显示菜单的 JSON 格式版本，用于外部 IDE 的 UI。

JSON 配置服务器

`confserver.py` 工具可以帮助 IDE 轻松地与配置系统的逻辑进行集成，它运行在后台，通过使用 stdin 和 stdout 读写 JSON 文件的方式与调用进程交互。

您可以通过 `idf.py confserver` 或 `ninja confserver` 从项目中运行 `confserver.py`，也可以使用不同的构建生成器来触发类似的目标。

配置服务器会向 stderr 输出方便阅读的错误和警告信息，向 stdout 输出 JSON 文件。启动时，配置服务器将以 JSON 字典的形式输出系统中每个配置项的完整值，以及范围受限的值的可用范围。`sdkconfig.json` 中包含有相同的信息:

```
{"version": 1, "values": { "ITEM": "value", "ITEM_2": 1024, "ITEM_3": false }, "ranges"␣
↪: { "ITEM_2" : [ 0, 32768 ] } }
```

配置服务器仅发送可见的配置项，其它不可见的或者被禁用的配置项可从 `kconfig_menus.json` 静态文件中解析得到，此文件还包含菜单结构和其它元数据（描述、类型、范围等）。

然后配置服务器将等待客户端的输入，客户端会发起请求，要求更改一个或多个配置项的值，内容的格式是个 JSON 对象，后面跟一个换行符:

```
{"version": "1", "set": {"SOME_NAME": false, "OTHER_NAME": true } }
```

配置服务器将解析此请求，更新 `sdkconfig` 文件，并返回完整的变更列表:

```
{"version": 1, "values": {"SOME_NAME": false, "OTHER_NAME": true , "DEPENDS_ON_SOME_NAME
↪": null}}
```

当前不可见或者禁用的配置项会返回 `null`，任何新的可见配置项则会返回其当前新的可见值。

如果配置项的取值范围因另一个值的变化发生了改变，那么配置服务器会发送:

```
{"version": 1, "values": {"OTHER_NAME": true }, "ranges" : { "HAS_RANGE" : [ 3, 4 ] } }
```

如果传递的数据无效，那么 JSON 对象中会有 `error` 字段:

```
{"version": 1, "values": {}, "error": ["The following config symbol(s) were not visible␣
↪so were not updated: NOT_VISIBLE_ITEM"]}
```

默认情况下，变更后的配置不会被写进 sdkconfig 文件。更改的内容在发出 “save” 命令之前会先储存在内存中:

```
{"version": 1, "save": null }
```

若要从已保存的文件中重新加载配置值，并丢弃内存中的任何更改，可以发送 “load” 命令:

```
{"version": 1, "load": null }
```

“load” 和 “save” 的值可以是新的路径名，也可以设置为 “null” 用以加载/保存之前使用的路径名。

配置服务器对 “load” 命令的响应始终是完整的配置值和取值范围的集合，这与服务器初始启动阶段的响应相同。

“load”、“set” 和 “save” 的任意组合可以在一条单独的命令中发送出去，服务器按照组合中的顺序执行命令。因此，可以使用一条命令实现从文件中加载配置，更新配置值，然后将其保存到文件中。

注解: 配置服务器不会自动加载外部对 `sdkconfig` 文件的任何更改。如果文件被外部编辑，则需要发送 “load” 命令或重启服务器。

注解: `sdkconfig` 文件更新后，配置服务器不会重新运行 CMake 来生成其它的构建文件和元数据文件。这些文件会在下一次运行 `CMake` 或 `idf.py` 时自动生成。

5.2.20 从 ESP-IDF GNU Make 构建系统迁移到 CMake 构建系统

ESP-IDF CMake 构建系统与旧版的 GNU Make 构建系统在某些方面非常相似，例如将 `component.mk` 文件改写 `CMakeLists.txt`，像 `COMPONENT_ADD_INCLUDEDIRS` 和 `COMPONENT_SRCDIRS` 等变量可以保持不变，只需将语法改为 CMake 语法即可。

自动转换工具

/tools/cmake/convert_to_cmake.py 中提供了一个项目自动转换工具。运行此命令时需要加上项目路径，如下所示:

```
$IDF_PATH/tools/cmake/convert_to_cmake.py /path/to/project_dir
```

项目目录必须包含 Makefile 文件，并确保主机已安装 GNU Make (`make`) 工具，并且被添加到了 PATH 环境变量中。

该工具会将项目 Makefile 文件和所有组件的 `component.mk` 文件转换为对应的 `CMakeLists.txt` 文件。

转换过程如下：该工具首先运行 `make` 来展开 ESP-IDF 构建系统设置的变量，然后创建相应的 CMakelists 文件来设置相同的变量。

转换工具并不能处理复杂的 Makefile 逻辑或异常的目标，这些需要手动转换。

CMake 中不可用的功能

有些功能已从 CMake 构建系统中移除，或者已经发生很大改变。GNU Make 构建系统中的以下变量已从 CMake 构建系统中删除：

- `COMPONENT_BUILD_DIR`：由 `CMAKE_CURRENT_BINARY_DIR` 替代。
- `COMPONENT_LIBRARY`：默认为 `$(COMPONENT_NAME).a` 但是库名可以被组件覆盖。在 CMake 构建系统中，组件库名称不可再被组件覆盖。
- `CC`、`LD`、`AR`、`OBJCOPY`：gcc xtensa 交叉工具链中每个工具的完整路径。CMake 使用 `CMAKE_C_COMPILER`、`CMAKE_C_LINK_EXECUTABLE` 和 `CMAKE_OBJCOPY` 进行替代。完整列表请参阅 CMake 语言变量。
- `HOSTCC`、`HOSTLD`、`HOSTAR`：宿主机本地工具链中每个工具的全名。CMake 系统不再提供此变量，外部项目需要手动检测所需的宿主机工具链。
- `COMPONENT_ADD_LDFLAGS`：用于覆盖链接标志。CMake 中使用 target_link_libraries 命令替代。
- `COMPONENT_ADD_LINKER_DEPS`：链接过程依赖的文件列表。target_link_libraries 通常会自动推断这些依赖。对于链接脚本，可以使用自定义的 CMake 函数 `target_linker_scripts`。
- `COMPONENT_SUBMODULES`：不再使用。CMake 会自动枚举 ESP-IDF 仓库中所有的子模块。
- `COMPONENT_EXTRA_INCLUDES`：曾是 `COMPONENT_PRIV_INCLUDEDIRS` 变量的替代版本，仅支持绝对路径。CMake 系统中统一使用 `COMPONENT_PRIV_INCLUDEDIRS`（可以是相对路径，也可以是绝对路径）。
- `COMPONENT_OBJS`：以前，可以以目标文件列表的方式指定组件源，现在，可以通过 `COMPONENT_SRCS` 以源文件列表的形式指定组件源。
- `COMPONENT_OBJEXCLUDE`：已被 `COMPONENT_SRCEXCLUDE` 替换。用于指定源文件（绝对路径或组件目录的相对路径）。
- `COMPONENT_EXTRA_CLEAN`：已被 `ADDITIONAL_MAKE_CLEAN_FILES` 属性取代，注意，*CMake 对此项功能有部分限制*。

- COMPONENT_OWNBUILDTARGET & COMPONENT_OWNCLEANTARGET：已被 CMake 外部项目 替代，详细内容请参阅完全覆盖组件的构建过程。
- COMPONENT_CONFIG_ONLY：已被 register_config_only_component() 函数替代，请参阅仅配置组件。
- CFLAGS、CPPFLAGS、CXXFLAGS：已被相应的 CMake 命令替代，请参阅组件编译控制。

无默认值的变量

以下变量不再具有默认值：

- COMPONENT_SRCDIRS
- COMPONENT_ADD_INCLUDEDIRS

不再需要的变量

如果设置了 COMPONENT_SRCS，就不需要再设置 COMPONENT_SRCDIRS。实际上，CMake 构建系统中如果设置了 COMPONENT_SRCDIRS，那么 COMPONENT_SRCS 就会被忽略。

从 Make 中烧录

仍然可以使用 make flash 或者类似的目标来构建和烧录，但是项目 sdkconfig 不能再用来指定串口和波特率。可以使用环境变量来覆盖串口和波特率的设置，详情请参阅使用 *Ninja/Make* 来烧录。

5.3 构建系统 (传统 GNU Make)

[English]

注解： ESP-IDF V4.0 及之后版本的默认构建系统为 CMake。本文档主要针对之前基于 GNU Make 的传统构建系统。请注意，未来，我们可能不会继续支持基于 GNU Make 的构建系统。

本文将介绍乐鑫物联网开发框架中的 构建系统和 组件的相关概念。

如果您想了解如何构建一个新的 ESP-IDF 项目，请阅读本文档。

我们建议您使用 ESP-IDF 模板工程 来开始您的新项目。

5.3.1 使用构建系统

ESP-IDF 的 README.md 文件对如何使用构建系统来构建项目作了简要的说明。

5.3.2 概述

一个 ESP-IDF 项目可以看作是许多不同组件的集合，例如对于一个展示当前湿度的网站服务器来说，它可能会包含如下一些组件：

- ESP32 基础库（libc，rom bindings 等）
- WiFi 驱动库
- TCP/IP 协议栈
- FreeRTOS 操作系统
- 网站服务器
- 湿度传感器的驱动
- 将上述组件组织在一起的主代码

ESP-IDF 可以显式地指定和配置每个组件。在构建项目的时候，构建系统会查找 ESP-IDF 目录、项目目录和用户自定义目录（可选）中所有的组件，然后使用基于文本的菜单系统让用户配置 ESP-IDF 项目中需要的每个组件。在配置结束后，构建系统开始编译整个项目。

概念

- *项目*特指一个目录，其中包含了构建可执行文件的所有源文件和配置，还有其他的支持型输出文件，比如分区表、数据/文件系统分区和引导程序。
- *项目配置*保存在项目根目录下名为 sdkconfig 的文件中，它可以通过 `make menuconfig` 进行修改，且一个项目只能包含一个项目配置。
- *应用程序*是由 ESP-IDF 构建得到的可执行文件。一个项目通常会构建两个应用程序：项目应用程序（主可执行文件，即用户自定义的固件）和引导程序（启动并初始化项目应用程序的引导程序）。
- *组件*是模块化的、独立的代码，它们被编译成静态库（.a 文件）后再链接成应用程序，有些组件是 ESP-IDF 官方提供的，有些则可能来自其它项目。

以下内容并不是项目的组成部分：

- `ESP-IDF` 并不是项目的一部分，相反，它是独立的，并通过 IDF_PATH 环境变量链接到项目中，这样做就可以使 IDF 框架与您的项目分离，其中 IDF_PATH 变量保存了 ESP-IDF 目录的路径。
- 交叉编译工具链并不是项目的组成部分，它应该被安装在系统 PATH 环境变量中，或者可以在项目配置中显式指定工具链的前缀为本地的安装路径。

示例项目

示例项目的目录树结构可能如下所示：

```
- myProject/
            - Makefile
            - sdkconfig
            - components/ - component1/ - component.mk
                                        - Kconfig
                                        - src1.c
                          - component2/ - component.mk
                                        - Kconfig
                                        - src1.c
                                        - include/ - component2.h
            - main/       - src1.c
                          - src2.c
                          - component.mk
            - build/
```

该示例项目 myProject 包含以下组成部分：

- 项目顶层 Makefile，该 Makefile 设置了 PROJECT_NAME 变量，还可以定义作用于整个项目的其它 make 变量（可选）。顶层 Makefile 会导入核心 Makefile 文件 $(IDF_PATH)/make/project.mk ，由它负责实现 ESP-IDF 构建系统的剩余部分。
- 项目配置文件 sdkconfig，执行 make menuconfig 后会创建或更新此文件，该文件中保存了项目中所有组件的配置信息（包括 ESP-IDF 本身）。sdkconfig 文件可能会也可能不会被添加到项目的源代码管理系统中。
- 可选的组件目录中包含了属于项目一部分的自定义组件，不是每一个项目都需要它，但它有助于构建可重用代码或者导入第三方组件。
- main 目录是一个特殊的 *伪组件*，它包含项目本身的源代码。main 是默认名称，Makefile 变量 COMPONENT_DIRS 默认会导入此组件，但您也可以修改此变量（或者设置 EXTRA_COMPONENT_DIRS ）以查找其他位置的组件。
- build 目录在项目构建的时候创建或者更新，里面包含有构建生成的临时目标文件和库以及最终输出的二进制文件。此目录通常不会被添加到项目的源代码管理系统中，也不会随着项目源代码被发布。

组件目录中会包含组件自己的 Makefile 文件 component.mk ，里面会定义一些变量来控制该组件的构建过程，以及它与整个项目的集成。更多详细信息请参考*组件 Makefiles*。

每个组件还可以包含一个 Kconfig 文件，它用于定义 menuconfig 时展示的组件配置信息的选项规则。某些组件还可能还会包含 Kconfig.projbuild 和 Makefile.projbuild 特殊文件，他们可以用来覆盖项目的部分配置。

项目 Makefiles

每个项目都有一个 Makefile ，它包含整个项目的构建设置。默认情况下，项目 Makefile 可以非常小。

最小 Makefile 示例

```
PROJECT_NAME := myProject

include $(IDF_PATH)/make/project.mk
```

必须设置的项目变量

- PROJECT_NAME: 项目名称，最终输出的二进制文件也使用该名称，即 myProject.bin，myProject.elf 。

可选的项目变量

以下这些变量都有默认值，用户可以重写这些变量以自定义构建行为。查看 make/project.mk 文件可以获得所有的实现细节。

- PROJECT_PATH：顶层项目目录，默认是包含 Makefile 文件的目录，许多其他的项目变量都基于此变量。注意，项目路径中不能包含有空格。
- BUILD_DIR_BASE：所有对象、库、二进制文件的输出目录，默认为 $(PROJECT_PATH)/build。
- COMPONENT_DIRS：组件的搜索目录，默认为 $(IDF_PATH)/components，$(PROJECT_PATH)/components，$(PROJECT_PATH)/main 和 EXTRA_COMPONENT_DIRS 。如果您不希望从这些目录中搜索组件，请重写此变量。
- EXTRA_COMPONENT_DIRS：组件额外的搜索路径，可选。
- COMPONENTS：要构建进项目中的组件列表，默认为 COMPONENT_DIRS 指定目录中所有的组件。
- EXCLUDE_COMPONENTS：在构建的过程中需要剔除的组件列表，可选。请注意这只会减少构建的时间，并不会减少最终二进制文件的大小。
- TEST_EXCLUDE_COMPONENTS：在构建单元测试的过程中需要剔除的组件列表，可选。

以上这些 Makefile 变量中的任何路径都要使用绝对路径，您可以使用 $(PROJECT_PATH)/xxx，$(IDF_PATH)/xxx，或者使用 Make 内置函数 $(abspath xxx) 将相对路径转换为绝对路径。

以上这些变量要在 Makefile 中 include $(IDF_PATH)/make/project.mk 的前面进行设置。

组件 Makefiles

每个项目都包含一个或者多个组件，这些组件可以是 ESP-IDF 的一部分，也可以从其他组件目录添加。

组件是包含 component.mk 文件的任何目录。

搜索组件

搜索 COMPONENT_DIRS 中指定的目录以查找项目会使用的组件，目录可以是组件本身（即他们包含 component.mk 文件），也可以是包含组件的上层目录。

运行 make list-components 命令可以查询这些变量的值，这有助于调试组件的搜索路径是否正确。

同名组件

ESP-IDF 搜索组件时，会按照 COMPONENT_DIRS 指定的顺序依次进行，这意味着在默认情况下，首先是 ESP-IDF 组件，然后是项目组件，最后是 EXTRA_COMPONENT_DIRS 中的组件。如果这些目录中的两个或者多个包含具有相同名字的组件，则使用搜索到的最后一个位置的组件。这就允许将组件复制到项目目录中再修改来覆盖 ESP-IDF 组件，如果使用这种方式，ESP-IDF 目录本身可以保持不变。

最小组件 Makefile

最简单的 component.mk 文件可以是一个空文件，如果文件为空，则组件的默认构建行为会被设置为：

- makefile 所在目录中的所有源文件（*.c，*.cpp，*.cc，*.S）将会被编译进组件库中。
- 子目录 include 将被添加到其他组件的全局头文件搜索路径中。
- 组件库将会被链接到项目的应用程序中。

更完整的组件 makefile 可以查看*组件 Makefile 示例*。

请注意，空的 component.mk 文件同没有 component.mk 文件之间存在本质差异，前者会调用默认的组件构建行为，后者不会发生默认的组件构建行为。一个组件中如果只包含影响项目配置或构建过程的文件，那么它可以没有 component.mk 文件。

预设的组件变量

以下特定于组件的变量可以在 component.mk 中使用，但不应该被修改。

- COMPONENT_PATH：组件的目录，即包含 component.mk 文件的绝对路径，路径中不能包含空格。
- COMPONENT_NAME：组件的名字，默认为组件目录的名称。
- COMPONENT_BUILD_DIR：组件的构建目录，即存放组件构建输出的绝对路径，它是 *$(BUILD_DIR_BASE)* 的子目录。该变量也是构建组件时的当前工作目录，所以 make 中的相对路径都以此目录为基础。
- COMPONENT_LIBRARY：组件构建后的静态库文件（相对于组件的构建目录）的名字，默认为 $(COMPONENT_NAME).a。

以下变量在项目顶层中设置，并被导出到组件中构建时使用：

- PROJECT_NAME：项目名称，在项目的 Makefile 中设置。

- PROJECT_PATH：包含项目 Makefile 的目录的绝对路径。
- COMPONENTS：此次构建中包含的所有组件的名字。
- CONFIG_*：项目配置中的每个值在 make 中都对应一个以 CONFIG_ 开头的变量。
- CC，LD，AR，OBJCOPY：gcc xtensa 交叉编译工具链中每个工具的完整路径。
- HOSTCC，HOSTLD，HOSTAR：主机本地工具链中每个工具的全名。
- IDF_VER：ESP-IDF 的版本号，可以通过检索 $(IDF_PATH)/version.txt 文件（假如存在的话）或者使用 git 命令 git describe 来获取。这里推荐的格式是在一行中指定主 IDF 的发布版本号，例如标记为 v2.0 的发布版本或者是标记任意一次提交记录的 v2.0-275-g0efaa4f。应用程序可以通过调用*esp_get_idf_version()* 函数来使用该变量。
- IDF_VERSION_MAJOR, IDF_VERSION_MINOR, IDF_VERSION_PATCH: ESP-IDF 的组件版本，可用于条件表达式。请注意这些信息的精确度不如 IDF_VER 变量，版本号 v4.0-dev-*，v4.0-beta1，v4.0-rc1 和 v4.0 对应的 IDF_VERSION_* 变量值是相同的，但是 IDF_VER 的值是不同的。

如果您在 component.mk 文件中修改这些变量，这并不会影响其它组件的构建，但可能会使您的组件变得难以构建或调试。

可选的项目通用组件变量

可以在 component.mk 中设置以下变量来控制整个项目的构建行为：

- COMPONENT_ADD_INCLUDEDIRS：相对于组件目录的路径，将被添加到项目中所有组件的头文件搜索路径中。如果该变量未被覆盖，则默认为 include 目录。如果一个头文件路径仅仅为当前组件所用，那么应该将该路径添加到 COMPONENT_PRIV_INCLUDEDIRS 中。
- COMPONENT_ADD_LDFLAGS：添加链接参数到全局 LDFLAGS 中用以指导链接最终的可执行文件，默认为 -l$(COMPONENT_NAME)。如果将预编译好的库添加到此目录，请使用它们为绝对路径，即 $(COMPONENT_PATH)/libwhatever.a。
- COMPONENT_DEPENDS：需要在当前组件之前构建的组件列表，这对于处理链接时的依赖不是必需的，因为所有组件的头文件目录始终可用。如果一个组件会生成一个头文件，然后另外一个组件需要使用它，此时该变量就有必要进行设置。大多数的组件不需要设置此变量。
- COMPONENT_ADD_LINKER_DEPS：保存一些文件的路径，当这些文件发生改变时，会触发 ELF 文件重新链接。该变量通常用于链接脚本文件和二进制文件，大多数的组件不需要设置此变量。

以下变量仅适用于属于 ESP-IDF 的组件：

- COMPONENT_SUBMODULES：组件使用的 git 子模块的路径列表（相对于 COMPONENT_PATH）。这些路径会在构建的过程中被检查（并在必要的时候初始化）。如果组件位于 IDF_PATH 目录之外，则忽略此变量。

可选的组件特定变量

以下变量可以在 component.mk 中进行设置，用以控制该组件的构建行为：

- COMPONENT_PRIV_INCLUDEDIRS：相对于组件目录的目录路径，该目录仅会被添加到该组件源文件的头文件搜索路径中。
- COMPONENT_EXTRA_INCLUDES：编译组件的源文件时需要指定的额外的头文件搜索路径，这些路径将以 -I 为前缀传递给编译器。这和 COMPONENT_PRIV_INCLUDEDIRS 变量的功能有些类似，但是这些路径不会相对于组件目录进行扩展。
- COMPONENT_SRCDIRS：相对于组件目录的目录路径，这些路径用于搜索源文件（*.cpp，*.c，*.S），默认为 .，即组件目录本身。重写该变量可以指定包含源文件的不同目录列表。
- COMPONENT_OBJS：要编译生成的目标文件，默认是 COMPONENT_SRCDIRS 中每个源文件的.o 文件。重写该变量将允许您剔除 COMPONENT_SRCDIRS 中的某些源文件，否则他们将会被编译。相关示例请参阅*指定需要编译的组件源文件*。
- COMPONENT_EXTRA_CLEAN：相对于组件构建目录的路径，指向 component.mk 文件中自定义 make 规则生成的任何文件，它们也是 make clean 命令需要删除的文件。相关示例请参阅*源代码生成*。
- COMPONENT_OWNBUILDTARGET & COMPONENT_OWNCLEANTARGET：这些目标允许您完全覆盖组件的默认编译行为。有关详细信息，请参阅*完全覆盖组件的 Makefile*。
- COMPONENT_CONFIG_ONLY：如果设置了此标志，则表示组件根本不会产生构建输出（即不会构建得到 COMPONENT_LIBRARY），并且会忽略大多数其它组件变量。此标志用于 IDF 内部组件，其中仅包含 KConfig.projbuild 和/或 Makefile.projbuild 文件来配置项目，但是没有源文件。
- CFLAGS：传递给 C 编译器的标志。根据项目设置已经定义一组默认的 CFLAGS，可以通过 CFLAGS += 来为组件添加特定的标志，也可以完全重写该变量（尽管不推荐这么做）。
- CPPFLAGS：传递给 C 预处理器的标志（用于 .c，.cpp 和 .S 文件）。根据项目设置已经定义一组默认的 CPPFLAGS ，可以通过 CPPFLAGS += 来为组件添加特定的标志，也可以完全重写该变量（尽管不推荐这么做）。
- CXXFLAGS：传递给 C++ 编译器的标志。根据项目设置已经定义一组默认的 CXXFLAGS ，可以通过 CXXFLAGS += 来为组件添加特定的标志，也可以完全重写该变量（尽管不推荐这么做）。

如果要将编译标志应用于单个源文件，您可以将该源文件的目标规则覆盖，例如：

```
apps/dhcpserver.o: CFLAGS += -Wno-unused-variable
```

如果上游代码在编译的时候发出了警告，那这么做可能会很有效。

配置组件

每个组件都可以包含一个 Kconfig 文件，和 component.mk 放在同一个目录下。Kconfig 中包含此组件在 make menuconfig 时要展示的配置规则的设置。

运行 menuconfig 时，可以在 Component Settings 菜单栏下找到这些设置。

创建一个组件的 Kconfig 文件，最简单的方法就是使用 ESP-IDF 中现有的 Kconfig 文件作为模板，在这基础上进行修改。

有关示例请参阅添加条件配置。

预处理器定义

ESP-IDF 构建系统会在命令行中添加以下 C 预处理定义：

- `ESP_PLATFORM` —可以用来检测在 ESP-IDF 内发生的构建行为。
- `IDF_VER` —ESP-IDF 的版本，请参阅预设的组件变量。

构建的内部过程

顶层：项目 Makefile

- `make` 始终从项目目录处运行，并且项目的 makefile 名字通常为 Makefile 。
- 项目的 makefile 文件会设置 `PROJECT_NAME` ，并且可以自定义其他可选的项目变量。
- 项目 makefile 文件会导入 `$(IDF_PATH)/make/project.mk` ，该文件中会导入项目级的 Make 逻辑。
- `project.mk` 填写默认的项目级 make 变量，并导入项目配置中的 make 变量。如果生成的包含项目配置的 makefile 文件已经过期，那么它将会被重新生成（通过 `project_config.mk` 中的目标规则），然后 make 进程从顶层重新开始。
- `project.mk` 根据默认组件目录或者可选项目变量中设置的自定义组件列表来编译组件。
- 每个组件都可以设置一些可选的项目通用组件变量 ，他们会通过 `component_project_vars.mk` 被导入 `project.mk` 文件中。如果这些文件有缺失或者过期，他们会被重新生成（通过对组件 makefile 的递归调用），然后 make 进程从顶层重新开始。
- 组件中的 Makefile.projbuild 文件被包含在了 make 的进程中，以添加额外的目标或者配置。
- 默认情况下，项目 makefile 还为每个组件生成顶层的编译和清理目标，并设置 app 和 clean 目标来调用所有这些子目标。
- 为了编译每个组件，对组件 makefile 执行递归构建。

为了更好地理解项目的构建过程，请通读 `project.mk` 文件。

第二层：组件 Makefile 文件

- 每次调用组件 makefile 文件都是通过 `$(IDF_PATH)/make/component_wrapper.mk` 这个包装器进行的。
- 此组件包装器包含了所有组件的 `Makefile.componentbuild` 文件，使这些文件中的任何配置，变量都可用于每个组件。
- 调用 `component_wrapper.mk` 时将当前目录设置为组件构建目录，并将 `COMPONENT_MAKEFILE` 变量设置为 `component.mk` 的绝对路径。

- component_wrapper.mk 为所有组件变量设置默认值，然后导入 component.mk 文件来覆盖或修改这些变量。
- 如果未定义 COMPONENT_OWNBUILDTARGET 和 COMPONENT_OWNCLEANTARGET 文件，则会为组件的源文件和必备组件 COMPONENT_LIBRARY 静态库文件创建默认构建和清理目标。
- component_project_vars.mk 文件在 component_wrapper.mk 中有自己的目标，如果由于组件的 makefile 或者项目配置的更改而需要重建此文件，则从 project.mk 文件中进行评估。

为了更好地理解组件制作过程，请阅读 component_wrapper.mk 文件和 ESP-IDF 中的 component.mk 文件。

以非交互的方式运行 Make

如果在运行 make 的时候不希望出现交互式提示（例如：在 IDE 或自动构建系统中），可以将 BATCH_BUILD=1 添加到 make 的参数中（或者将其设置为环境变量）。

设置 BATCH_BUILD 意味着：

- 详细输出（与 V=1 相同，见下文），如果不需要详细输出，就设置 V=0 。
- 如果项目配置缺少新配置项（来自新组件或者 ESP-IDF 更新），则项目使用默认值，而不是提示用户输入每个项目。
- 如果构建系统需要调用 menuconfig ，则会打印错误并且构建失败。

构建目标的进阶用法

- make app, make bootloader, make partition table 可以根据需要为项目单独构建生成应用程序文件、启动引导文件和分区表文件。
- make erase_flash 和 make erase_otadata 会调用 esptool.py 脚本分别擦除整块闪存芯片或者其中 OTA 分区的内容。
- make size 会打印应用程序的大小信息。make size-components 和 make size-files 两者功能相似，分别打印每个组件或者每个源文件大小的详细信息。

调试 Make 的过程

调试 ESP-IDF 构建系统的一些技巧：

- 将 V=1 添加到 make 的参数中（或将其设置为环境变量）将使 make 回显所有已经执行的命令，以及为子 make 输入的每个目录。
- 运行 make -w 将导致 make 在为子 make 输入时回显每个目录——与 V=1 相同但不回显所有命令。
- 运行 make --trace （可能除了上述参数之一）将打印出构建时的每个目标，以及导致它构建的依赖项）。
- 运行 make -p 会打印每个 makefile 中每个生成的目标的（非常详细的）摘要。

更多调试技巧和通用的构建信息，请参阅 GNU 构建手册。

警告未定义的变量

默认情况下，如果引用了未定义的变量（如 `$(DOES_NOT_EXIST)`），构建过程将会打印警告，这对于查找变量名称中的错误非常有用。

如果不想要此行为，可以在 menuconfig 顶层菜单下的 *SDK tool configuration* 中禁用它。

请注意，如果在 Makefile 中使用 `ifdef` 或者 `ifndef`，则此选项不会出发警告。

覆盖项目的部分内容

Makefile.projbuild

如果一个组件含有必须要在项目构建过程的顶层进行计算的变量，则可以在组件目录下创建名为 `Makefile.projbuild` 的文件，项目在执行 `project.mk` 的时候会导入此 makefile 。

例如，如果您的组件需要为整个项目添加 CFLAGS（不仅仅是为自身的源文件），那么可以在 `Makefile.projbuild` 中设置 `CFLAGS +=` 。

`Makefile.projbuild` 文件在 ESP-IDF 中大量使用，用于定义项目范围的构建功能，例如 `esptool.py` 命令行参数和 `bootloader` 这个特殊的程序。

请注意，`Makefile.projbuild` 对于最常见的组件不是必需的 - 例如向项目中添加 include 目录，或者将 LDFLAGS 添加到最终链接步骤，同样可以通过 `component.mk` 文件来自定义这些值。有关详细信息，请参阅可选的项目通用组件变量 。

> **警告：** 在此文件中设置变量或者目标时要小心，由于这些值包含在项目的顶层 makefile 中，因此他们可以影响或者破坏所有组件的功能！

KConfig.projbuild

这相当于 `Makefile.projbuild` 的组件配置 KConfig 文件，如果要在 menuconfig 的顶层添加配置选项，而不是在 组件配置子菜单中，则可以在 `component.mk` 文件所在目录中的 KConfig.projbuild 文件中定义这些选项。

在此文件中添加配置时要小心，因为他们将包含在整个项目配置中，在可能的情况下，通常最好为组件创建和配置 KConfig 文件。

Makefile.componentbuild

对于一些特殊的组件，比如它们会使用工具从其余文件中生成源文件，这时就有必要将配置、宏或者变量的定义添加到每个组件的构建过程中。这是通过在组件目录中包含 Makefile.componentbuild 文件来实现的。此文件在 component.mk 文件之前被导入 component_wrapper.mk 中。同 Makefile.projbuild 文件一样，请留意这些文件，因为他们包含在每个组件的构建中，所有只有在编译完全不同的组件时才会出现 Makefile.componentbuild 错误。

仅配置的组件

仅配置的组件是一类不包含源文件的特殊组件，只有 Kconfig.projbuild 和 Makefile.projbuild 文件，可以在 conponent.mk 文件中设置标志 COMPONENT_CONFIG_ONLY。如果设置了此标志，则忽略大多数其他组件变量，并且不会为组件执行构建操作。

组件 Makefile 示例

因为构建环境试图设置大多数情况都能工作的合理默认值，所以 component.mk 可能非常小，甚至是空的，请参考最小组件 *Makefile*。但是某些功能通常需要覆盖组件的变量。

以下是 component.mk 的一些更高级的示例：

增加源文件目录

默认情况下，将忽略子目录。如果您的项目在子目录中而不是在组件的根目录中有源文件，那么您可以通过设置 COMPONENT_SRCDIRS 将其告知构建系统：

```
COMPONENT_SRCDIRS := src1 src2
```

构建系统将会编译 src1/ 和 src2/ 子目录中的所有源文件。

指定源文件

标准 component.mk 逻辑将源目录中的所有.S 和.c 文件添加为无条件编译的源。通过将 COMPONENT_OBJS 变量手动设置为需要生成的对象的名称，可以绕过该逻辑并对要编译的对象进行硬编码。

```
COMPONENT_OBJS := file1.o file2.o thing/filea.o thing/fileb.o anotherthing/main.o
COMPONENT_SRCDIRS := . thing anotherthing
```

请注意，还需要另外设置 COMPONENT_SRCDIRS 。

添加条件配置

配置系统可用于有条件地编译某些文件，具体取决于 `make menuconfig` 中选择的选项。为此，ESP-IDF 具有 `compile_only_if` 和 `compile_only_if_not` 的宏：

Kconfig:

```
config FOO_ENABLE_BAR
    bool "Enable the BAR feature."
    help
        This enables the BAR feature of the FOO component.
```

component.mk:

```
$(call compile_only_if,$(CONFIG_FOO_ENABLE_BAR),bar.o)
```

从示例中可以看出，`compile_only_if` 宏将条件和目标文件列表作为参数。如果条件为真（在这种情况下：如果在 menuconfig 中启用了 BAR 功能），将始终编译目标文件（在本例中为 bar.o）。相反的情况也是如此，如果条件不成立，bar.o 将永远不会被编译。`compile_only_if_not` 执行相反的操作，如果条件为 false 则编译，如果条件为 true 则不编译。

这也可以用于选择或者删除实现，如下所示：

Kconfig:

```
config ENABLE_LCD_OUTPUT
    bool "Enable LCD output."
    help
        Select this if your board has a LCD.

config ENABLE_LCD_CONSOLE
    bool "Output console text to LCD"
    depends on ENABLE_LCD_OUTPUT
    help
        Select this to output debugging output to the lcd

config ENABLE_LCD_PLOT
    bool "Output temperature plots to LCD"
    depends on ENABLE_LCD_OUTPUT
    help
        Select this to output temperature plots
```

component.mk:

```
# If LCD is enabled, compile interface to it, otherwise compile dummy interface
$(call compile_only_if,$(CONFIG_ENABLE_LCD_OUTPUT),lcd-real.o lcd-spi.o)
$(call compile_only_if_not,$(CONFIG_ENABLE_LCD_OUTPUT),lcd-dummy.o)

#We need font if either console or plot is enabled
$(call compile_only_if,$(or $(CONFIG_ENABLE_LCD_CONSOLE),$(CONFIG_ENABLE_LCD_PLOT)),␣
↪font.o)
```

请注意使用 Make 或者函数来包含字体文件。其他的替换函数，比如 `and` 和 `if` 也适用于此处。也可以使用不在 menuconfig 中定义的变量，ESP-IDF 使用默认的 Make 策略，将一个空的或者只包含空格的变量视为 false ，而其中任何非空格的比那辆都为 true 。

(注意：本文档的历史版本建议将目标文件添加到 `COMPONENT_OBJS` 中，虽然这仍然可行，但是只有当组件中的所有目标文件都明确命名时才会起作用，并且在 `make clean` 后并不会清除 make 中取消选择的目标文件)。

源代码生成

某些组件会出现源文件未随组件本身提供，而必须从另外一个文件生成的情况。假设我们的组件有一个头文件，该文件由 BMP 文件转换后的二进制数据组成，假设使用 bmp2h 的工具进行转换，然后将头文件包含在名为 graphics_lib.c 的文件中：

```
COMPONENT_EXTRA_CLEAN := logo.h

graphics_lib.o: logo.h

logo.h: $(COMPONENT_PATH)/logo.bmp
    bmp2h -i $^ -o $@
```

这个示例会在当前目录（构建目录）中生成 graphics_lib.o 和 logo.h 文件，而 logo.bmp 随组件一起提供并位于组件路径下。因为 logo.h 是一个生成的文件，所以当调用 `make clean` 时需要清理它，这就是为什么要将它添加到 `COMPONENT_EXTRA_CLEAN` 变量中。

润色与改进

将 logo.h 添加作为 `graphics_lib.o` 的依赖项会导致在编译 `graphics_lib.c` 之前先生成它。

如果另一个组件中的源文件需要使用 logo.h，则必须将此组件的名称添加到另一个组件的 `COMPONENT_DEPENDS` 列表中，以确保组件按顺序编译。

嵌入二进制数据

有时您的组件希望使用一个二进制文件或者文本文件，但是您又不希望将它重新格式化为 C 源文件。

这时，您可以在 component.mk 文件中设置变量 COMPONENT_EMBED_FILES，以这种方式指定要嵌入的文件的名称：

```
COMPONENT_EMBED_FILES := server_root_cert.der
```

或者，如果文件是字符串，则可以使用变量 COMPONENT_EMBED_TXTFILES，这将把文本文件的内容当成以 null 结尾的字符串嵌入：

```
COMPONENT_EMBED_TXTFILES := server_root_cert.pem
```

文件的内容会被编译进 flash 中的.rodata 段，并通过符号名称来访问，如下所示：

```
extern const uint8_t server_root_cert_pem_start[] asm("_binary_server_root_cert_pem_start
↪");
extern const uint8_t server_root_cert_pem_end[]   asm("_binary_server_root_cert_pem_end
↪");
```

符号名称是根据文件的全名生成的，如 COMPONENT_EMBED_FILES 中的所示，字符 / ，. ，等都将会被下划线替代。符号名称中的 _binary 前缀由 objcopy 添加，对于文本和二进制文件都是相同的。

有关使用此技术的示例，请参考 protocols/https_request - 证书文件的内容会在编译时从.pem 文件中加载。

完全覆盖组件的 Makefile

显然，在某些情况下，所有这些配置都不足以满足某个组件，例如，当组件基本上是另一个第三方组件的包装器时，该第三方组件最初不打算在 ESP-IDF 构建系统下工作，在这种情况下，可以通过设置 COMPONENT_OWNBUILDTARGET 和可能的 COMPONENT_OWNCLEANTARGET，并在 component.mk 中定义名为 build 和 clean 的目标。构建目标可以执行任何操作，只要它为项目生成了 $(COMPONENT_LIBRARY) ，并最终被链接到应用程序二进制文件中即可。

(实际上，这并不是必须的 - 如果 COMPONENT_ADD_LDFLAGS 变量被覆盖，那么组件可以指示链接器链接其他二进制文件。)

自定义 sdkconfig 的默认值

对于示例工程或者其他您不想指定完整 sdkconfig 配置的项目，但是您确实希望覆盖 ESP-IDF 默认值中的某些键值，则可以在项目中创建文件 sdkconfig.defaults，运行 make defconfig 或从头创建新配置时将会使用此文件。

要想覆盖此文件的名称，请设置 SDKCONFIG_DEFAULTS 环境变量。

保存 flash 参数

在某些情况下，我们希望在没有 IDF 的情况下烧写目标板卡，对于这种情况，我们希望保存构建的二进制文件、esptool.py 和 esptool write_flash 命令的参数。可以简单编写一段脚本来保存二进制文件和 esptool.py，并且使用命令 `make print_flash_cmd` 来查看烧写 flash 时的参数。

```
--flash_mode dio --flash_freq 40m --flash_size detect 0x1000 bootloader/bootloader.bin␣
↪0x10000 example_app.bin 0x8000 partition_table_unit_test_app.bin
```

然后使用这段 flash 参数作为 esptool write_flash 命令的参数：

```
python esptool.py --chip esp32 --port /dev/ttyUSB0 --baud 921600 --before default_reset -
↪-after hard_reset write_flash -z --flash_mode dio --flash_freq 40m --flash_size detect␣
↪0x1000 bootloader/bootloader.bin 0x10000 example_app.bin 0x8000 partition_table_unit_
↪test_app.bin
```

5.3.3 构建 Bootloader

引导程序默认作为 `make all` 的一部分被构建，或者也可以通过 `make bootloader-clean` 来单独构建，此外还可以通过 `make bootloader-list-components` 来查看构建引导程序时包含的组件。

引导程序是一个特殊的组件，因为主项目中的二级引导程序拥有单独的.EFL 和.BIN 文件。但是它与主项目共享配置和构建目录。

这是通过在 components/bootloader/subproject 下添加子项目来完成的。这个子项目有自己的 Makefile，但它希望通过 components/bootloader/Makefile.projectbuild 文件中的一些配置使自己从主项目的 Makefile 被调用。有关详细信息，请参阅这些文件。

5.4 错误处理

[English]

5.4.1 概述

在应用程序开发中，及时发现并处理在运行时期的错误，对于保证应用程序的健壮性非常重要。常见的运行时错误有如下几种：

- 可恢复的错误：
  - 通过函数的返回值（错误码）表示的错误
  - 使用 `throw` 关键字抛出的 C++ 异常
- 不可恢复（严重）的错误：

- 断言失败（使用 assert 宏或者其它类似方法）或者直接调用 abort() 函数造成的错误
- CPU 异常：访问受保护的内存区域、非法指令等
- 系统级检查：看门狗超时、缓存访问错误、堆栈溢出、堆栈粉碎、堆栈损坏等

本文将介绍 ESP-IDF 中针对可恢复错误的错误处理机制，并提供一些常见错误的处理模式。

关于如何处理不可恢复的错误，请查阅不可恢复错误。

5.4.2 错误码

ESP-IDF 中大多数函数会返回*esp_err_t* 类型的错误码，*esp_err_t* 实质上是带符号的整型，ESP_OK 代表成功（没有错误），具体值定义为 0。

在 ESP-IDF 中，许多头文件都会使用预处理器，定义可能出现的错误代码。这些错误代码通常均以 ESP_ERR_ 前缀开头，一些常见错误（比如内存不足、超时、无效参数等）的错误代码则已经在 esp_err.h 文件中定义好了。此外，ESP-IDF 中的各种组件 (component) 也都可以针对具体情况，自行定义更多错误代码。

完整错误代码列表，请见错误代码参考 中查看完整的错误列表。

5.4.3 错误码到错误消息

错误代码并不直观，因此 ESP-IDF 还可以使用*esp_err_to_name()* 或者*esp_err_to_name_r()* 函数，将错误代码转换为具体的错误消息。例如，我们可以向*esp_err_to_name()* 函数传递错误代码 0x101，可以得到返回字符串 “ESP_ERR_NO_MEM”。这样一来，我们可以在日志中输出更加直观的错误消息，而不是简单的错误码，从而帮助研发人员更快理解发生了何种错误。

此外，如果出现找不到匹配的 ESP_ERR_ 值的情况，函数*esp_err_to_name_r()* 则会尝试将错误码作为一种标准 POSIX 错误代码 进行解释。具体过程为：POSIX 错误代码（例如 ENOENT，ENOMEM）定义在 errno.h 文件中，可以通过 errno 变量获得，进而调用 strerror_r 函数实现。在 ESP-IDF 中，errno 是一个基于线程的局部变量，即每个 FreeRTOS 任务都有自己的 errno 副本，通过函数修改 errno 也只会作用于当前任务中的 errno 变量值。

该功能（即在无法匹配 ESP_ERR_ 值时，尝试用标准 POSIX 解释错误码）默认启用。用户也可以禁用该功能，从而减小应用程序的二进制文件大小，详情可见*CONFIG_ESP_ERR_TO_NAME_LOOKUP*。注意，该功能对禁用并不影响*esp_err_to_name()* 和*esp_err_to_name_r()* 函数的定义，用户仍可调用这两个函数转化错误码。在这种情况下，*esp_err_to_name()* 函数在遇到无法匹配错误码的情况会返回 UNKNOWN ERROR，而*esp_err_to_name_r()* 函数会返回 Unknown error 0xXXXX(YYYYY)，其中 0xXXXX 和 YYYYY 分别代表错误代码的十六进制和十进制表示。

5.4.4 ESP_ERROR_CHECK 宏

宏 ESP_ERROR_CHECK() 的功能和 assert 类似，不同之处在于：这个宏会检查*esp_err_t* 的值，而非判断 bool 条件。如果传给 ESP_ERROR_CHECK() 的参数不等于*ESP_OK* ，则会在控制台上打印错误消息，然后调用 abort() 函数。

错误消息通常如下所示：

```
ESP_ERROR_CHECK failed: esp_err_t 0x107 (ESP_ERR_TIMEOUT) at 0x400d1fdf

file: "/Users/user/esp/example/main/main.c" line 20
func: app_main
expression: sdmmc_card_init(host, &card)

Backtrace: 0x40086e7c:0x3ffb4ff0 0x40087328:0x3ffb5010 0x400d1fdf:0x3ffb5030␣
↪0x400d0816:0x3ffb5050
```

- 第一行打印错误代码的十六进制表示，及该错误在源代码中的标识符。这个标识符取决于*CONFIG_ESP_ERR_TO_NAME_LOOKUP* 选项的设定。最后，第一行还会打印程序中该错误发生的具体位置。
- 下面几行显示了程序中调用 ESP_ERROR_CHECK() 宏的具体位置，以及传递给该宏的参数。
- 最后一行打印回溯结果。对于所有不可恢复错误，这里在应急处理程序中打印的内容都是一样的。更多有关回溯结果的详细信息，请参阅不可恢复错误 。

注解： 如果使用*IDF monitor*, 则最后一行回溯结果中的地址将会被替换为相应的文件名和行号。

5.4.5 错误处理模式

1. 尝试恢复。根据具体情况不同，我们具体可以：
   - 在一段时间后，重新调用该函数；
   - 尝试删除该驱动，然后重新进行“初始化”；
   - 采用其他带外机制，修改导致错误发生的条件（例如，对一直没有响应的外设进行复位等）。

   示例：

   ```
   esp_err_t err;
   do {
       err = sdio_slave_send_queue(addr, len, arg, timeout);
       // 如果发送队列已满就不断重试
   } while (err == ESP_ERR_TIMEOUT);
   if (err != ESP_OK) {
       // 处理其他错误
   }
   ```

2. 将错误传递回调用程序。在某些中间件组件中，采用此类处理模式代表函数必须以相同的错误码退出，这样才能确保所有分配的资源都能得到释放。

示例：

```
sdmmc_card_t* card = calloc(1, sizeof(sdmmc_card_t));
if (card == NULL) {
    return ESP_ERR_NO_MEM;
}
esp_err_t err = sdmmc_card_init(host, &card);
if (err != ESP_OK) {
    // 释放内存
    free(card);
    // 将错误码传递给上层（例如通知用户）
    // 或者，应用程序可以自定义错误代码并返回
    return err;
}
```

3. 转为不可恢复错误，比如使用 ESP_ERROR_CHECK。详情请见*ESP_ERROR_CHECK 宏* 章节。

   对于中间件组件而言，通常并不希望在发生错误时中止应用程序。不过，有时在应用程序级别，这种做法是可以接受的。在 ESP-IDF 的示例代码中，很多都会使用 ESP_ERROR_CHECK 来处理各种 API 引发的错误，虽然这不是应用程序的最佳做法，但可以让示例代码看起来更加简洁。

   示例：

```
ESP_ERROR_CHECK(spi_bus_initialize(host, bus_config, dma_chan));
```

5.4.6 C++ 异常

默认情况下，ESP-IDF 会禁用对 C++ 异常的支持，但是可以通过*CONFIG_COMPILER_CXX_EXCEPTIONS* 选项启用。

通常情况下，启用异常处理会让应用程序的二进制文件增加几 kB。此外，启用该功能时还应为异常事故池预留一定内存。当应用程序无法从堆中分配异常对象时，就可以使用这个池中的内存。该内存池的大小可以通过*CONFIG_COMPILER_CXX_EXCEPTIONS_EMG_POOL_SIZE* 来设定。

如果 C++ 程序抛出了异常，但是程序中并没有 catch 代码块来捕获该异常，那么程序的运行就会被 abort 函数中止，然后打印回溯信息。有关回溯的更多信息，请参阅不可恢复错误 。

5.5 严重错误

[English]

5.5.1 概述

在某些情况下，程序并不会按照我们的预期运行，在 ESP-IDF 中，这些情况包括：

- CPU 异常：非法指令，加载/存储时的内存对齐错误，加载/存储时的访问权限错误，双重异常。
- 系统级检查错误：
  - *中断看门狗* 超时
  - *任务看门狗* 超时（只有开启*CONFIG_ESP_TASK_WDT_PANIC* 后才会触发严重错误）
  - 高速缓存访问错误
  - 掉电检测事件
  - 堆栈溢出
  - Stack 粉碎保护检查
  - Heap 完整性检查
- 使用 `assert`、`configASSERT` 等类似的宏断言失败。

本指南会介绍 ESP-IDF 中这类错误的处理流程，并给出对应的解决建议。

5.5.2 紧急处理程序

概述 中列举的所有错误都会由 *紧急处理程序*（*Panic Handler*）负责处理。

紧急处理程序首先会将出错原因打印到控制台，例如 CPU 异常的错误信息通常会类似于：

```
Guru Meditation Error: Core 0 panic'ed (IllegalInstruction). Exception was unhandled.
```

对于一些系统级检查错误（如中断看门狗超时，高速缓存访问错误等），错误信息会类似于：

```
Guru Meditation Error: Core 0 panic'ed (Cache disabled but cached memory region accessed)
```

不管哪种情况，错误原因都会被打印在括号中。请参阅*Guru Meditation 错误* 以查看所有可能的出错原因。

紧急处理程序接下来的行为将取决于*CONFIG_ESP32_PANIC* 的设置，支持的选项包括：

- 打印 CPU 寄存器，然后重启（`CONFIG_ESP32_PANIC_PRINT_REBOOT`）- 默认选项

  打印系统发生异常时 CPU 寄存器的值，打印回溯，最后重启芯片。
- 打印 CPU 寄存器，然后暂停（`CONFIG_ESP32_PANIC_PRINT_HALT`）

  与上一个选项类似，但不会重启，而是选择暂停程序的运行。重启程序需要外部执行复位操作。
- 静默重启（`CONFIG_ESP32_PANIC_SILENT_REBOOT`）

  不打印 CPU 寄存器的值，也不打印回溯，立即重启芯片。

- 调用 GDB Stub（CONFIG_ESP32_PANIC_GDBSTUB）

  启动 GDB 服务器，通过控制台 UART 接口与 GDB 进行通信。详细信息请参阅*GDB Stub*。

紧急处理程序的行为还受到另外两个配置项的影响：

- 如果*CONFIG_ESP32_DEBUG_OCDAWARE* 被使能了（默认），紧急处理程序会检测 ESP32 是否已经连接 JTAG 调试器。如果检测成功，程序会暂停运行，并将控制权交给调试器。在这种情况下，寄存器和回溯不会被打印到控制台，并且也不会使用 GDB Stub 和 Core Dump 的功能。
- 如果使能了*Core Dump* 功能（CONFIG_ESP32_ENABLE_COREDUMP_TO_FLASH 或者 CONFIG_ESP32_ENABLE_COREDUMP_TO_UART 选项），系统状态（任务堆栈和寄存器）会被转储到 Flash 或者 UART 以供后续分析。

下图展示了紧急处理程序的行为：

5.5.3 寄存器转储与回溯

除非启用了 CONFIG_ESP32_PANIC_SILENT_REBOOT 否则紧急处理程序会将 CPU 寄存器和回溯打印到控制台：

```
Core 0 register dump:
PC      : 0x400e14ed  PS      : 0x00060030  A0      : 0x800d0805  A1      : 0x3ffb5030
A2      : 0x00000000  A3      : 0x00000001  A4      : 0x00000001  A5      : 0x3ffb50dc
A6      : 0x00000000  A7      : 0x00000001  A8      : 0x00000000  A9      : 0x3ffb5000
A10     : 0x00000000  A11     : 0x3ffb2bac  A12     : 0x40082d1c  A13     : 0x06ff1ff8
A14     : 0x3ffb7078  A15     : 0x00000000  SAR     : 0x00000014  EXCCAUSE: 0x0000001d
EXCVADDR: 0x00000000  LBEG    : 0x4000c46c  LEND    : 0x4000c477  LCOUNT  : 0xffffffff

Backtrace: 0x400e14ed:0x3ffb5030 0x400d0802:0x3ffb5050
```

仅会打印异常帧中 CPU 寄存器的值，即引发 CPU 异常或者其它严重错误时刻的值。

紧急处理程序如果是因 abort() 而调用，则不会打印寄存器转储。

在某些情况下，例如中断看门狗超时，紧急处理程序会额外打印 CPU 寄存器（EPC1-EPC4）的值，以及另一个 CPU 的寄存器值和代码回溯。

回溯行包含了当前任务中每个堆栈帧的 PC:SP 对（PC 是程序计数器，SP 是堆栈指针）。如果在 ISR 中发生了严重错误，回溯会同时包括被中断任务的 PC:SP 对，以及 ISR 中的 PC:SP 对。

如果使用了*IDF 监视器*，该工具会将程序计数器的值转换为对应的代码位置（函数名，文件名，行号），并加以注释：

```
Core 0 register dump:
PC      : 0x400e14ed  PS      : 0x00060030  A0      : 0x800d0805  A1      : 0x3ffb5030
0x400e14ed: app_main at /Users/user/esp/example/main/main.cpp:36
```

(下页继续)

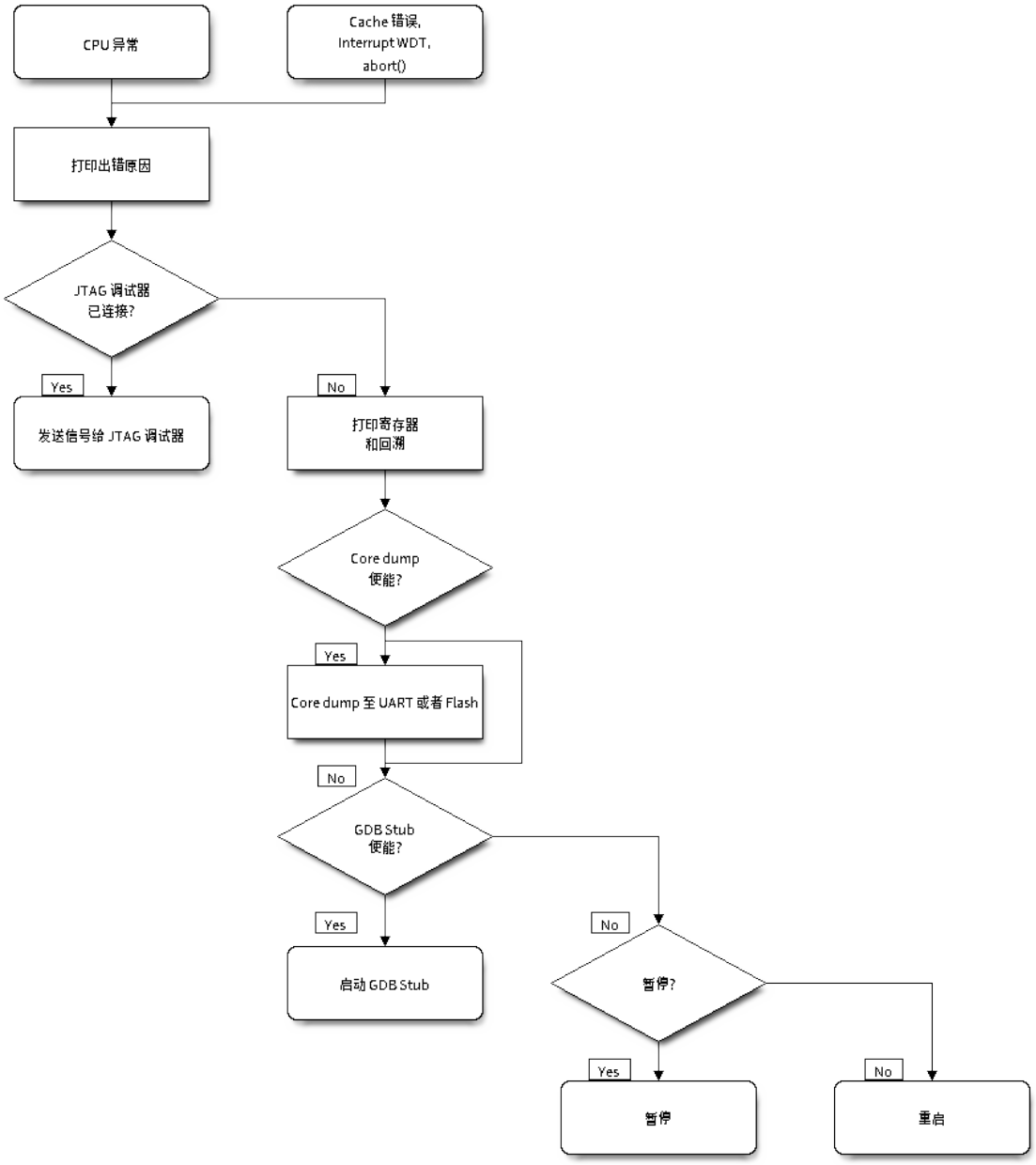


图 1: 紧急处理程序流程图（点击放大）

(续上页)

```
A2      : 0x00000000  A3      : 0x00000001  A4      : 0x00000001  A5      : 0x3ffb50dc
A6      : 0x00000000  A7      : 0x00000001  A8      : 0x00000000  A9      : 0x3ffb5000
A10     : 0x00000000  A11     : 0x3ffb2bac  A12     : 0x40082d1c  A13     : 0x06ff1ff8
0x40082d1c: _calloc_r at /Users/user/esp/esp-idf/components/newlib/syscalls.c:51

A14     : 0x3ffb7078  A15     : 0x00000000  SAR     : 0x00000014  EXCCAUSE: 0x0000001d
EXCVADDR: 0x00000000  LBEG    : 0x4000c46c  LEND    : 0x4000c477  LCOUNT  : 0xffffffff

Backtrace: 0x400e14ed:0x3ffb5030 0x400d0802:0x3ffb5050
0x400e14ed: app_main at /Users/user/esp/example/main/main.cpp:36

0x400d0802: main_task at /Users/user/esp/esp-idf/components/esp32/cpu_start.c:470
```

若要查找发生严重错误的代码位置，请查看 “Backtrace” 的后面几行，发生严重错误的代码显示在顶行，后续几行显示的是调用堆栈。

5.5.4 GDB Stub

如果启用了 CONFIG_ESP32_PANIC_GDBSTUB 选项，在发生严重错误时，紧急处理程序不会复位芯片，相反，它将启动 GDB 远程协议服务器，通常称为 GDB Stub。发生这种情况时，可以让主机上运行的 GDB 实例通过 UART 端口连接到 ESP32。

如果使用了*IDF 监视器*，该工具会在 UART 端口检测到 GDB Stub 提示符后自动启动 GDB，输出会类似于:

```
Entering gdb stub now.
$T0b#e6GNU gdb (crosstool-NG crosstool-ng-1.22.0-80-gff1f415) 7.10
Copyright (C) 2015 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=x86_64-build_apple-darwin16.3.0 --target=xtensa-esp32-
↪elf".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
```

(下页继续)

(续上页)

```
Type "apropos word" to search for commands related to "word"...
Reading symbols from /Users/user/esp/example/build/example.elf...done.
Remote debugging using /dev/cu.usbserial-31301
0x400e1b41 in app_main ()
    at /Users/user/esp/example/main/main.cpp:36
36      *((int*) 0) = 0;
(gdb)
```

在 GDB 会话中，我们可以检查 CPU 寄存器，本地和静态变量以及内存中任意位置的值。但是不支持设置断点，改变 PC 值或者恢复程序的运行。若要复位程序，请退出 GDB 会话，在 IDF 监视器中连续输入 Ctrl-T Ctrl-R，或者按下开发板上的复位按键也可以重新运行程序。

5.5.5 Guru Meditation 错误

本节将对打印在 `Guru Meditation Error: Core panic'ed` 后面括号中的致错原因进行逐一解释。

注解: 想要了解 “Guru Meditation” 的历史渊源，请参阅 维基百科 。

IllegalInstruction

此 CPU 异常表示当前执行的指令不是有效指令，引起此错误的常见原因包括：

- FreeRTOS 中的任务函数已返回。在 FreeRTOS 中，如果想终止任务函数，需要调用*vTaskDelete()* 函数释放当前任务的资源，而不是直接返回。
- 无法从 SPI Flash 中加载下一条指令，这通常发生在：
  - 应用程序将 SPI Flash 的引脚重新配置为其它功能（如 GPIO，UART 等等）。有关 SPI Flash 引脚的详细信息，请参阅硬件设计指南和芯片/模组的数据手册。
  - 某些外部设备意外连接到 SPI Flash 的引脚上，干扰了 ESP32 和 SPI Flash 之间的通信。

InstrFetchProhibited

此 CPU 异常表示 CPU 无法加载指令，因为指令的地址不在 IRAM 或者 IROM 中的有效区域中。

通常这意味着代码中调用了并不指向有效代码块的函数指针。这种情况下，可以查看 `PC`（程序计数器）寄存器的值并做进一步判断：若为 0 或者其它非法值（即只要不是 `0x4xxxxxxx` 的情况），则证实确实是该原因。

LoadProhibited, StoreProhibited

当应用程序尝试读取或写入无效的内存位置时，会发生此类 CPU 异常。此类无效内存地址可以在寄存器转储的 `EXCVADDR` 中找到。如果该地址为零，通常意味着应用程序正尝试解引用一个 NULL 指针。如果该地址接近于零，则通常意味着应用程序尝试访问某个结构体的成员，但是该结构体的指针为 NULL。如果该地址是其它非法值（不在 `0x3fxxxxxx` - `0x6xxxxxxx` 的范围内），则可能意味着用于访问数据的指针未初始化或者已经损坏。

IntegerDivideByZero

应用程序尝试将整数除以零。

LoadStoreAlignment

应用程序尝试读取/写入的内存位置不符合加载/存储指令对字节对齐大小的要求，例如，32 位加载指令只能访问 4 字节对齐的内存地址，而 16 位加载指令只能访问 2 字节对齐的内存地址。

LoadStoreError

应用程序尝试从仅支持 32 位加载/存储的内存区域执行 8 位或 16 位加载/存储操作，例如，解引用一个指向指令内存区域的 `char*` 指针就会导致这样的错误。

Unhandled debug exception

这后面通常会再跟一条消息:

```
Debug exception reason: Stack canary watchpoint triggered (task_name)
```

此错误表示应用程序写入的位置越过了 `task_name` 任务堆栈的末尾，请注意，并非每次堆栈溢出都会触发此错误。任务有可能会绕过堆栈金丝雀（stack canary）的位置访问堆栈，在这种情况下，监视点就不会被触发。

Interrupt wdt timeout on CPU0 / CPU1

这表示发生了中断看门狗超时，详细信息请查阅*看门狗* 文档。

Cache disabled but cached memory region accessed

在某些情况下，ESP-IDF 会暂时禁止通过高速缓存访问外部 SPI Flash 和 SPI RAM，例如在使用 spi_flash API 读取/写入/擦除/映射 SPI Flash 的时候。在这些情况下，任务会被挂起，并且未使用 `ESP_INTR_FLAG_IRAM` 注册的中断处理程序会被禁用。请确保任何使用此标志注册的中断处理程序所访问的代码和数据分别位于 IRAM 和 DRAM 中。更多详细信息请参阅*SPI Flash API 文档*。

5.5.6 其它严重错误

欠压

ESP32 内部集成掉电检测电路，并且会默认启用。如果电源电压低于安全值，掉电检测器可以触发系统复位。掉电检测器可以使用*CONFIG_ESP32_BROWNOUT_DET* 和*CONFIG_ESP32_BROWNOUT_DET_LVL_SEL* 这两个选项进行设置。当掉电检测器被触发时，会打印如下信息:

```
Brownout detector was triggered
```

芯片会在该打印信息结束后复位。

请注意，如果电源电压快速下降，则只能在控制台上看到部分打印信息。

Heap 不完整

ESP-IDF 堆的实现包含许多运行时的堆结构检查，可以在 menuconfig 中开启额外的检查（“Heap Poisoning”）。如果其中的某项检查失败，则会打印类似如下信息:

```
CORRUPT HEAP: Bad tail at 0x3ffe270a. Expected 0xbaad5678 got 0xbaac5678
assertion "head != NULL" failed: file "/Users/user/esp/esp-idf/components/heap/multi_
↪heap_poisoning.c", line 201, function: multi_heap_free
abort() was called at PC 0x400dca43 on core 0
```

更多详细信息，请查阅堆内存调试 文档。

Stack 粉碎

Stack 粉碎保护（基于 GCC `-fstack-protector*` 标志）可以通过 ESP-IDF 中的*CONFIG_COMPILER_STACK_CHECK_MODE* 选项来开启。如果检测到 Stack 粉碎，则会打印类似如下的信息:

```
Stack smashing protect failure!

abort() was called at PC 0x400d2138 on core 0

Backtrace: 0x4008e6c0:0x3ffc1780 0x4008e8b7:0x3ffc17a0 0x400d2138:0x3ffc17c0␣
↪0x400e79d5:0x3ffc17e0 0x400e79a7:0x3ffc1840 0x400e79df:0x3ffc18a0␣
↪0x400e2235:0x3ffc18c0 0x400e1916:0x3ffc18f0 0x400e19cd:0x3ffc1910␣
↪0x400e1a11:0x3ffc1930 0x400e1bb2:0x3ffc1950 0x400d2c44:0x3ffc1a80
0
```

回溯信息会指明发生 Stack 粉碎的函数，建议检查函数中是否有代码访问本地数组时发生了越界。

5.6 Event Handling

Several ESP-IDF components use *events* to inform application about state changes, such as connection or disconnection. This document gives an overview of these event mechanisms.

5.6.1 Wi-Fi, Ethernet, and IP Events

Before the introduction of *esp_event library*, events from Wi-Fi driver, Ethernet driver, and TCP/IP stack were dispatched using the so-called *legacy event loop*. The following sections explain each of the methods.

esp_event Library Event Loop

esp_event library is designed to supersede the legacy event loop for the purposes of event handling in ESP-IDF. In the legacy event loop, all possible event types and event data structures had to be defined in *system_event_id_t* enumeration and *system_event_info_t* union, which made it impossible to send custom events to the event loop, and use the event loop for other kinds of events (e.g. Mesh). Legacy event loop also supported only one event handler function, therefore application components could not handle some of Wi-Fi or IP events themselves, and required application to forward these events from its event handler function.

See *esp_event library API reference* for general information on using this library. Wi-Fi, Ethernet, and IP events are sent to the *default event loop* provided by this library.

Legacy Event Loop

This event loop implementation is started using *esp_event_loop_init()* function. Application typically supplies an *event handler*, a function with the following signature:

```
esp_err_t event_handler(void *ctx, system_event_t *event)
{
}
```

Both the pointer to event handler function, and an arbitrary context pointer are passed to *esp_event_loop_init()*.

When Wi-Fi, Ethernet, or IP stack generate an event, this event is sent to a high-priority `event` task via a queue. Application-provided event handler function is called in the context of this task. Event task stack size and event queue size can be adjusted using *CONFIG_ESP_SYSTEM_EVENT_TASK_STACK_SIZE* and *CONFIG_ESP_SYSTEM_EVENT_QUEUE_SIZE* options, respectively.

Event handler receives a pointer to the event structure (*system_event_t*) which describes current event. This structure follows a *tagged union* pattern: `event_id` member indicates the type of event, and

`event_info` member is a union of description structures. Application event handler will typically use `switch(event->event_id)` to handle different kinds of events.

If application event handler needs to relay the event to some other task, it is important to note that event pointer passed to the event handler is a pointer to temporary structure. To pass the event to another task, application has to make a copy of the entire structure.

Event IDs and Corresponding Data Structures

Event ID (legacy event ID)	Event data structure
Wi-Fi	
WIFI_EVENT_WIFI_READY (SYSTEM_EVENT_WIFI_READY)	n/a
WIFI_EVENT_SCAN_DONE (SYSTEM_EVENT_SCAN_DONE)	*wifi_event_sta_scan_done_t*
WIFI_EVENT_STA_START (SYSTEM_EVENT_STA_START)	n/a
WIFI_EVENT_STA_STOP (SYSTEM_EVENT_STA_STOP)	n/a
WIFI_EVENT_STA_CONNECTED (SYSTEM_EVENT_STA_CONNECTED)	*wifi_event_sta_connected_t*
WIFI_EVENT_STA_DISCONNECTED (SYSTEM_EVENT_STA_DISCONNECTED)	*wifi_event_sta_disconnected_t*
WIFI_EVENT_STA_AUTHMODE_CHANGE (SYSTEM_EVENT_STA_AUTHMODE_CHANGE)	*wifi_event_sta_authmode_change_t*
WIFI_EVENT_STA_WPS_ER_SUCCESS (SYSTEM_EVENT_STA_WPS_ER_SUCCESS)	n/a
WIFI_EVENT_STA_WPS_ER_FAILED (SYSTEM_EVENT_STA_WPS_ER_FAILED)	*wifi_event_sta_wps_fail_reason_t*
WIFI_EVENT_STA_WPS_ER_TIMEOUT (SYSTEM_EVENT_STA_WPS_ER_TIMEOUT)	n/a
WIFI_EVENT_STA_WPS_ER_PIN (SYSTEM_EVENT_STA_WPS_ER_PIN)	*wifi_event_sta_wps_er_pin_t*
WIFI_EVENT_AP_START (SYSTEM_EVENT_AP_START)	n/a
WIFI_EVENT_AP_STOP (SYSTEM_EVENT_AP_STOP)	n/a
WIFI_EVENT_AP_STACONNECTED (SYSTEM_EVENT_AP_STACONNECTED)	*wifi_event_ap_staconnected_t*
WIFI_EVENT_AP_STADISCONNECTED (SYSTEM_EVENT_AP_STADISCONNECTED)	*wifi_event_ap_stadisconnected_t*
WIFI_EVENT_AP_PROBEREQRECVED (SYSTEM_EVENT_AP_PROBEREQRECVED)	*wifi_event_ap_probe_req_rx_t*
Ethernet	
ETHERNET_EVENT_START (SYSTEM_EVENT_ETH_START)	n/a
ETHERNET_EVENT_STOP (SYSTEM_EVENT_ETH_STOP)	n/a
ETHERNET_EVENT_CONNECTED (SYSTEM_EVENT_ETH_CONNECTED)	n/a
ETHERNET_EVENT_DISCONNECTED (SYSTEM_EVENT_ETH_DISCONNECTED)	n/a
IP	
IP_EVENT_STA_GOT_IP (SYSTEM_EVENT_STA_GOT_IP)	`ip_event_got_ip_t`
IP_EVENT_STA_LOST_IP (SYSTEM_EVENT_STA_LOST_IP)	n/a
IP_EVENT_AP_STAIPASSIGNED (SYSTEM_EVENT_AP_STAIPASSIGNED)	n/a
IP_EVENT_GOT_IP6 (SYSTEM_EVENT_GOT_IP6)	`ip_event_got_ip6_t`
IP_EVENT_ETH_GOT_IP (SYSTEM_EVENT_ETH_GOT_IP)	`ip_event_got_ip_t`

5.6.2 Mesh Events

ESP-MESH uses a system similar to the *Legacy Event Loop* to deliver events to the application. See 系统事件 for details.

5.6.3 Bluetooth Events

Various modules of the Bluetooth stack deliver events to applications via dedicated callback functions. Callback functions receive the event type (enumerated value) and event data (union of structures for each event type). The following list gives the registration API name, event enumeration type, and event parameters type.

- BLE GAP: *esp_ble_gap_register_callback()*, *esp_gap_ble_cb_event_t*, *esp_ble_gap_cb_param_t*.
- BT GAP: *esp_bt_gap_register_callback()*, *esp_bt_gap_cb_event_t*, *esp_bt_gap_cb_param_t*.
- GATTC: *esp_ble_gattc_register_callback()*, esp_bt_gattc_cb_event_t, esp_bt_gattc_cb_param_t.
- GATTS: *esp_ble_gatts_register_callback()*, esp_bt_gatts_cb_event_t, esp_bt_gatts_cb_param_t.
- SPP: *esp_spp_register_callback()*, *esp_spp_cb_event_t*, *esp_spp_cb_param_t*.
- Blufi: *esp_blufi_register_callbacks()*, *esp_blufi_cb_event_t*, *esp_blufi_cb_param_t*.
- A2DP: *esp_a2d_register_callback()*, *esp_a2d_cb_event_t*, *esp_a2d_cb_param_t*.
- AVRC: *esp_avrc_ct_register_callback()*, *esp_avrc_ct_cb_event_t*, *esp_avrc_ct_cb_param_t*.
- HFP Client: *esp_hf_client_register_callback()*, *esp_hf_client_cb_event_t*, *esp_hf_client_cb_param_t*.
- HFP AG: esp_hf_ag_register_callback(), esp_hf_ag_cb_event_t, esp_hf_ag_cb_param_t.

5.7 Deep Sleep Wake Stubs

ESP32 supports running a "deep sleep wake stub" when coming out of deep sleep. This function runs immediately as soon as the chip wakes up - before any normal initialisation, bootloader, or ESP-IDF code has run. After the wake stub runs, the SoC can go back to sleep or continue to start ESP-IDF normally.

Deep sleep wake stub code is loaded into "RTC Fast Memory" and any data which it uses must also be loaded into RTC memory. RTC memory regions hold their contents during deep sleep.

5.7.1 Rules for Wake Stubs

Wake stub code must be carefully written:

- As the SoC has freshly woken from sleep, most of the peripherals are in reset states. The SPI flash is unmapped.
- The wake stub code can only call functions implemented in ROM or loaded into RTC Fast Memory (see below.)
- The wake stub code can only access data loaded in RTC memory. All other RAM will be unintiailised and have random contents. The wake stub can use other RAM for temporary storage, but the contents will be overwritten when the SoC goes back to sleep or starts ESP-IDF.
- RTC memory must include any read-only data (.rodata) used by the stub.
- Data in RTC memory is initialised whenever the SoC restarts, except when waking from deep sleep. When waking from deep sleep, the values which were present before going to sleep are kept.
- Wake stub code is a part of the main esp-idf app. During normal running of esp-idf, functions can call the wake stub functions or access RTC memory. It is as if these were regular parts of the app.

5.7.2 Implementing A Stub

The wake stub in esp-idf is called `esp_wake_deep_sleep()`. This function runs whenever the SoC wakes from deep sleep. There is a default version of this function provided in esp-idf, but the default function is weak-linked so if your app contains a function named `esp_wake_deep_sleep()` then this will override the default.

If supplying a custom wake stub, the first thing it does should be to call `esp_default_wake_deep_sleep()`.

It is not necessary to implement `esp_wake_deep_sleep()` in your app in order to use deep sleep. It is only necessary if you want to have special behaviour immediately on wake.

If you want to swap between different deep sleep stubs at runtime, it is also possible to do this by calling the `esp_set_deep_sleep_wake_stub()` function. This is not necessary if you only use the default `esp_wake_deep_sleep()` function.

All of these functions are declared in the `esp_deepsleep.h` header under components/esp32.

5.7.3 Loading Code Into RTC Memory

Wake stub code must be resident in RTC Fast Memory. This can be done in one of two ways.

The first way is to use the `RTC_IRAM_ATTR` attribute to place a function into RTC memory:

```
void RTC_IRAM_ATTR esp_wake_deep_sleep(void) {
    esp_default_wake_deep_sleep();
    // Add additional functionality here
}
```

The second way is to place the function into any source file whose name starts with `rtc_wake_stub`. Files names `rtc_wake_stub*` have their contents automatically put into RTC memory by the linker.

The first way is simpler for very short and simple code, or for source files where you want to mix "normal" and "RTC" code. The second way is simpler when you want to write longer pieces of code for RTC memory.

5.7.4 Loading Data Into RTC Memory

Data used by stub code must be resident in RTC memory. The data can be placed in RTC Fast memory or in RTC Slow memory which is also used by the ULP.

Specifying this data can be done in one of two ways:

The first way is to use the `RTC_DATA_ATTR` and `RTC_RODATA_ATTR` to specify any data (writeable or read-only, respectively) which should be loaded into RTC memory:

```
RTC_DATA_ATTR int wake_count;

void RTC_IRAM_ATTR esp_wake_deep_sleep(void) {
    esp_default_wake_deep_sleep();
    static RTC_RODATA_ATTR const char fmt_str[] = "Wake count %d\n";
    ets_printf(fmt_str, wake_count++);
}
```

The RTC memory area where this data will be placed can be configured via menuconfig option named `CONFIG_ESP32_RTCDATA_IN_FAST_MEM`. This option allows to keep slow memory area for ULP programs and once it is enabled the data marked with `RTC_DATA_ATTR` and `RTC_RODATA_ATTR` are placed in the RTC fast memory segment otherwise it goes to RTC slow memory (default option). This option depends on the `CONFIG_FREERTOS_UNICORE` because RTC fast memory can be accessed only by PRO_CPU.

The similar attributes `RTC_FAST_ATTR` and `RTC_SLOW_ATTR` can be used to specify data that will be force placed into RTC_FAST and RTC_SLOW memory respectively. Any access to data marked with `RTC_FAST_ATTR` is allowed by PRO_CPU only and it is responsibility of user to make sure about it.

Unfortunately, any string constants used in this way must be declared as arrays and marked with RTC_RODATA_ATTR, as shown in the example above.

The second way is to place the data into any source file whose name starts with `rtc_wake_stub`.

For example, the equivalent example in `rtc_wake_stub_counter.c`:

```
int wake_count;

void RTC_IRAM_ATTR esp_wake_deep_sleep(void) {
    esp_default_wake_deep_sleep();
    ets_printf("Wake count %d\n", wake_count++);
}
```

The second way is a better option if you need to use strings, or write other more complex code.

To reduce wake-up time use the *CONFIG_BOOTLOADER_SKIP_VALIDATE_IN_DEEP_SLEEP* Kconfig option, see more information in *Fast boot from Deep Sleep*.

5.8 ESP32 Core Dump

5.8.1 Overview

ESP-IDF provides support to generate core dumps on unrecoverable software errors. This useful technique allows post-mortem analysis of software state at the moment of failure. Upon the crash system enters panic state, prints some information and halts or reboots depending configuration. User can choose to generate core dump in order to analyse the reason of failure on PC later on. Core dump contains snapshots of all tasks in the system at the moment of failure. Snapshots include tasks control blocks (TCB) and stacks. So it is possible to find out what task, at what instruction (line of code) and what callstack of that task lead to the crash. ESP-IDF provides special script *espcoredump.py* to help users to retrieve and analyse core dumps. This tool provides two commands for core dumps analysis:

- info_corefile - prints crashed task’ s registers, callstack, list of available tasks in the system, memory regions and contents of memory stored in core dump (TCBs and stacks)
- dbg_corefile - creates core dump ELF file and runs GDB debug session with this file. User can examine memory, variables and tasks states manually. Note that since not all memory is saved in core dump only values of variables allocated on stack will be meaningfull

5.8.2 Configuration

There are a number of core dump related configuration options which user can choose in project configuration menu (*idf.py menuconfig*).

1. Core dump data destination (*Components -> Core dump -> Data destination*):

- Save core dump to Flash (Flash)
- Print core dump to UART (UART)
- Disable core dump generation (None)

2. Core dump data format (*Components -> Core dump -> Core dump data format*):

- ELF format (Executable and Linkable Format file for core dump)
- Binary format (Basic binary format for core dump)

The ELF format contains extended features and allow to save more information about broken tasks and crashed software but it requires more space in the flash memory. It also stores SHA256 of crashed application image. This format of core dump is recommended for new software designs and is flexible enough to extend saved information for future revisions. The Binary format is kept for compatibility standpoint, it uses less space in the memory to keep data and provides better performance.

3. Maximum number of tasks snapshots in core dump (*Components -> Core dump -> Maximum number of tasks*).

4. Delay before core dump is printed to UART (*Components -> Core dump -> Delay before print to UART*). Value is in ms.

5. Type of data integrity check for core dump (*Components -> Core dump -> Core dump data integrity check*).

- Use CRC32 for core dump integrity verification
- Use SHA256 for core dump integrity verification

The SHA256 hash algorithm provides greater probability of detecting corruption than a CRC32 with multiple bit errors. The CRC32 option provides better calculation performance and consumes less memory for storage.

5.8.3 Save core dump to flash

When this option is selected core dumps are saved to special partition on flash. When using default partition table files which are provided with ESP-IDF it automatically allocates necessary space on flash, But if user wants to use its own layout file together with core dump feature it should define separate partition for core dump as it is shown below:

```
# Name,   Type, SubType, Offset,  Size
# Note: if you have increased the bootloader size, make sure to update the offsets to␣
↪avoid overlap
nvs,      data, nvs,     0x9000,  0x6000
phy_init, data, phy,     0xf000,  0x1000
factory,  app,  factory, 0x10000, 1M
coredump, data, coredump,,        64K
```

There are no special requrements for partition name. It can be choosen according to the user application needs, but partition type should be ‘data’ and sub-type should be ‘coredump’. Also when choosing partition size note that core dump data structure introduces constant overhead of 20 bytes and per-task

overhead of 12 bytes. This overhead does not include size of TCB and stack for every task. So partirion size should be at least 20 + max tasks number x (12 + TCB size + max task stack size) bytes.

The example of generic command to analyze core dump from flash is: *espcoredump.py -p </path/to/serial/port> info_corefile </path/to/program/elf/file>* or *espcoredump.py -p </path/to/serial/port> dbg_corefile </path/to/program/elf/file>*

5.8.4 Print core dump to UART

When this option is selected base64-encoded core dumps are printed on UART upon system panic. In this case user should save core dump text body to some file manually and then run the following command: *espcoredump.py info_corefile -t b64 -c </path/to/saved/base64/text> </path/to/program/elf/file>* or *espcoredump.py dbg_corefile -t b64 -c </path/to/saved/base64/text> </path/to/program/elf/file>*

Base64-encoded body of core dump will be between the following header and footer:

```
================= CORE DUMP START =================
<body of base64-encoded core dump, save it to file on disk>
================= CORE DUMP END ===================
```

The *CORE DUMP START* and *CORE DUMP END* lines must not be included in core dump text file.

5.8.5 ROM Functions in Backtraces

It is possible situation that at the moment of crash some tasks or/and crashed task itself have one or more ROM functions in their callstacks. Since ROM is not part of the program ELF it will be impossible for GDB to parse such callstacks, because it tries to analyse functions' prologues to acomplish that. In that case callstack printing will be broken with error message at the first ROM function. To overcome this issue you can use ROM ELF provided by Espressif (https://dl.espressif.com/dl/esp32_rom.elf) and pass it to 'espcoredump.py' .

5.8.6 Running 'espcoredump.py'

Generic command syntax:

espcoredump.py [options] command [args]

Script Options

- –chip,-c {auto,esp32}. Target chip type. Supported values are *auto* and *esp32*.
- –port,-p PORT. Serial port device.
- –baud,-b BAUD. Serial port baud rate used when flashing/reading.

Commands

- info_corefile. Retrieve core dump and print useful info.
- dbg_corefile. Retrieve core dump and start GDB session with it.

Command Arguments

- –debug,-d DEBUG. Log level (0..3).
- –gdb,-g GDB. Path to gdb to use for data retrieval.
- –core,-c CORE. Path to core dump file to use (if skipped core dump will be read from flash).
- –core-format,-t CORE_FORMAT. Specifies that file passed with "-c" is an ELF ("elf"), dumped raw binary ("raw") or base64-encoded ("b64") format.
- –off,-o OFF. Offset of coredump partition in flash (type *idf.py partition_table* to see it).
- –save-core,-s SAVE_CORE. Save core to file. Othwerwise temporary core file will be deleted. Ignored with "-c" .
- –rom-elf,-r ROM_ELF. Path to ROM ELF file to use (if skipped "esp32_rom.elf" is used).
- –print-mem,-m Print memory dump. Used only with "info_corefile" .
- <prog> Path to program ELF file.

5.9 Flash 加密

[English]

本文档将介绍 ESP32 的 Flash 加密功能，并通过示例展示用户如何在开发及生产过程中使用此功能。本文档旨在引导用户快速入门如何测试及验证 Flash 加密的相关操作。有关 Flash 加密块的详细信息可参见 ESP32 技术参考手册。

5.9.1 概述

Flash 加密功能用于加密与 ESP32 搭载使用的 SPI Flash 中的内容。启用 Flash 加密功能后，物理读取 SPI Flash 便无法恢复大部分 Flash 内容。通过明文数据烧录 ESP32 可应用加密功能，（若已启用加密功能）引导加载程序会在首次启动时对数据进行加密。

启用 Flash 加密后，系统将默认加密下列类型的 Flash 数据：

- 引导加载程序
- 分区表
- 所有 "app" 类型的分区

其他类型的 Flash 数据将视情况进行加密：

- 安全启动引导加载程序摘要（如果已启用安全启动）
- 分区表中标有“加密”标记的分区

Flash 加密与*安全启动* 功能各自独立，您可以在关闭安全启动的状态下使用 Flash 加密。但是，为了安全的计算机环境，二者应同时使用。在关闭安全启动的状态下，需运行其他配置来确保 Flash 加密的有效性。详细信息可参见*使用无安全启动的 Flash 加密*。

重要： 启用 Flash 加密将限制后续 ESP32 更新。请务必阅读本文档（包括*Flash 加密的局限性*）了解启用 Flash 加密的影响。

5.9.2 Flash 加密过程中使用的 eFuse

Flash 加密操作由 ESP32 上的多个 eFuse 控制。以下是这些 eFuse 列表及其描述：

eFuse	描述	是否可锁定为默认值	读取/写入
编码方案	该 2 位宽 eFuse 控制 BLOCK1 中使用的实际位数，从而获得最终的 256 位 AES 密钥。编码方案值解码如下： 0: 256 bits 1: 192 bits 2: 128 bits 最终的 AES 密钥根据 FLASH_CRYPT_CONFIG 的值产生。	是	0
BLOCK1	存储 AES 密钥的 256 位宽 eFuse 块	是	x
FLASH_CRYPT_CONFIG	4 位宽 eFuse，控制 AES 加密进程	是	0xF
download_dis_encrypt	设置后，在 UART 下载模式运行时关闭 Flash 加密操作	是	0
download_dis_decrypt	设置后，在 UART 下载模式运行时关闭 Flash 解	是	0

（下页继续）

(续上页)

```
                        密操作

FLASH_CRYPT_CNT         7 位 eFuse，在启动时启用/关闭加密功能                  是      ␣
↪       0

                        偶数位（0，2，4，6）：
                        启动时加密 Flash
                        奇数位（1，3，5，7）：
                        启动时不加密 Flash
```

对上述位的读写访问由 `efuse_wr_disable` 和 `efuse_rd_disable` 寄存器中的相应位控制。有关 ESP32 eFuse 的详细信息可参见*eFuse 管理器*。

5.9.3 Flash 的加密过程

假设 eFuse 值处于默认状态，且第二阶段的引导加载程序编译为支持 Flash 加密，则 Flash 加密过程执行如下：

- 首次上电复位时，Flash 中的所有数据都是未加密形式（明文数据）。第一阶段加载器 (Rom) 将在 IRAM 中加载第二阶段加载器。
- 第二阶段引导加载程序将读取 flash_crypt_cnt (=00000000b) eFuse 值，因为该值为 0（偶数位），第二阶段引导加载程序将配置并启用 Flash 加密块，同时将 `FLASH_CRYPT_CFG` eFuse 的值编程为 0xF。
- Flash 加密块将生成 AES-256 位密钥，并将其储存于 BLOCK1 eFuse 中。该操作在硬件中执行，软件将无法访问此密钥。
- 接着，Flash 加密块将加密 Flash 的内容（根据分区表的标记值）。原地加密可能会有耗时（取决于大分区的耗时）。
- 随后，第二阶段引导加载程序将在 flash_crypt_cnt (=00000001b) 中设置第一个可用位，从而标记已加密的 Flash 内容（偶数位）。
- 在*释放模式* 下，第二阶段引导加载程序将把 `download_dis_encrypt`、`download_dis_decrypt` 和 `download_dis_cache` 的 eFuse 位改写为 1，防止 UART 引导加载程序解密 Flash 的内容。同时也将写保护 `FLASH_CRYPT_CNT` 的 eFuse 位。
- 在*开发模式* 下，第二阶段引导加载程序将仅改写 `download_dis_decrypt` 和 `download_dis_cache` 的 eFuse 位，从而允许 UART 引导加载程序重新烧录加密的二进制文件。同时将不会写保护 `FLASH_CRYPT_CNT` 的 eFuse 位。
- 然后，第二阶段引导加载程序重启设备并开始执行加密映像，同时将透明解密 Flash 的内容并将其加载至 IRAM。

在开发阶段常需编写不同的明文 Flash 映像，以及测试 Flash 的加密过程。这要求 UART 下载模式能够根据需求不断加载新的明文映像。但是，在量产和生产过程中，出于安全考虑，UART 下载模式不应有权限访问

Flash 内容。因此需要有两种不同的 ESP32 配置：一种用于开发，另一种用于生产。以下章节介绍了 Flash 加密的开发模式 和释放模式 及其使用指南。

重要： 顾名思义，开发模式应 **仅开发过程**使用，因为该模式可以修改和回读加密的 Flash 内容。

5.9.4 设置 Flash 加密的步骤

开发模式

可使用 ESP32 内部生成的密钥或外部主机生成的密钥在开发中运行 Flash 加密。

使用 ESP32 生成的 Flash 加密密钥

正如上文所说，开发模式 允许用户使用 UART 下载模式多次下载明文映像。需完成以下步骤测试 Flash 加密过程：

- 确保您的 ESP32 设备有*Flash 加密过程中使用的 eFuse* 中所示的 Flash 加密 eFuse 的默认设置。
- 可在 `$IDF_PATH/examples/security/flash_encryption` 文件夹中找到 Flash 加密的示例应用程序。该示例应用程序中有显示 Flash 加密的状态（启用或关闭）以及 `FLASH_CRYPT_CNT` eFuse 值。
- 在第二阶段引导加载程序中启用 Flash 加密支持。请前往*Project Configuration Menu*，选择 “Security Features”。
- 选择*Enable flash encryption on boot*。
- 默认设置模式为 **开发模式**。
- 在引导加载程序 config 下选择适当详细程度的日志。
- 启用 Flash 加密将增大引导加载程序，因而可能需更新分区表偏移。请参见*Bootloader Size*。
- 保存配置并退出。

构建并烧录完整的映像包括：引导加载程序、分区表和 app。这些分区最初以未加密形式写入 Flash。

```
idf.py flash monitor
```

一旦烧录完成，设备将重置，在下次启动时，第二阶段引导加载程序将加密 Flash 的 app 分区，然后重置该分区。现在，示例应用程序将在运行时解密并执行命令。以下是首次启用 Flash 加密后 ESP32 启动时的样例输出。

```
--- idf_monitor on /dev/cu.SLAB_USBtoUART 115200 ---
--- Quit: Ctrl+] | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---
ets Jun  8 2016 00:22:57
```

(下页继续)

(续上页)

```
rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0018,len:4
load:0x3fff001c,len:8452
load:0x40078000,len:13608
load:0x40080400,len:6664
entry 0x40080764
I (28) boot: ESP-IDF v4.0-dev-850-gc4447462d-dirty 2nd stage bootloader
I (29) boot: compile time 15:37:14
I (30) boot: Enabling RNG early entropy source...
I (35) boot: SPI Speed      : 40MHz
I (39) boot: SPI Mode       : DIO
I (43) boot: SPI Flash Size : 4MB
I (47) boot: Partition Table:
I (51) boot: ## Label            Usage          Type ST Offset   Length
I (58) boot:  0 nvs              WiFi data        01 02 0000a000 00006000
I (66) boot:  1 phy_init         RF data          01 01 00010000 00001000
I (73) boot:  2 factory          factory app      00 00 00020000 00100000
I (81) boot: End of partition table
I (85) esp_image: segment 0: paddr=0x00020020 vaddr=0x3f400020 size=0x0808c (␣
↪32908) map
I (105) esp_image: segment 1: paddr=0x000280b4 vaddr=0x3ffb0000 size=0x01ea4 (␣
↪ 7844) load
I (109) esp_image: segment 2: paddr=0x00029f60 vaddr=0x40080000 size=0x00400 (␣
↪ 1024) load
0x40080000: _WindowOverflow4 at esp-idf/esp-idf/components/freertos/xtensa_
↪vectors.S:1778

I (114) esp_image: segment 3: paddr=0x0002a368 vaddr=0x40080400 size=0x05ca8 (␣
↪23720) load
I (132) esp_image: segment 4: paddr=0x00030018 vaddr=0x400d0018 size=0x126a8 (␣
↪75432) map
0x400d0018: _flash_cache_start at ??:?

I (159) esp_image: segment 5: paddr=0x000426c8 vaddr=0x400860a8 size=0x01f4c (␣
↪ 8012) load
0x400860a8: prvAddNewTaskToReadyList at esp-idf/esp-idf/components/freertos/
↪tasks.c:4561
```

(下页继续)

(续上页)

```
I (168) boot: Loaded app from partition at offset 0x20000
I (168) boot: Checking flash encryption...
I (168) flash_encrypt: Generating new flash encryption key...
I (187) flash_encrypt: Read & write protecting new key...
I (187) flash_encrypt: Setting CRYPT_CONFIG efuse to 0xF
W (188) flash_encrypt: Not disabling UART bootloader encryption
I (195) flash_encrypt: Disable UART bootloader decryption...
I (201) flash_encrypt: Disable UART bootloader MMU cache...
I (208) flash_encrypt: Disable JTAG...
I (212) flash_encrypt: Disable ROM BASIC interpreter fallback...
I (219) esp_image: segment 0: paddr=0x00001020 vaddr=0x3fff0018 size=0x00004 (␣
↪    4)
I (227) esp_image: segment 1: paddr=0x0000102c vaddr=0x3fff001c size=0x02104 (␣
↪ 8452)
I (239) esp_image: segment 2: paddr=0x00003138 vaddr=0x40078000 size=0x03528 (␣
↪13608)
I (249) esp_image: segment 3: paddr=0x00006668 vaddr=0x40080400 size=0x01a08 (␣
↪ 6664)
I (657) esp_image: segment 0: paddr=0x00020020 vaddr=0x3f400020 size=0x0808c (␣
↪32908) map
I (669) esp_image: segment 1: paddr=0x000280b4 vaddr=0x3ffb0000 size=0x01ea4 (␣
↪ 7844)
I (672) esp_image: segment 2: paddr=0x00029f60 vaddr=0x40080000 size=0x00400 (␣
↪ 1024)
0x40080000: _WindowOverflow4 at esp-idf/esp-idf/components/freertos/xtensa_
↪vectors.S:1778

I (676) esp_image: segment 3: paddr=0x0002a368 vaddr=0x40080400 size=0x05ca8 (␣
↪23720)
I (692) esp_image: segment 4: paddr=0x00030018 vaddr=0x400d0018 size=0x126a8 (␣
↪75432) map
0x400d0018: _flash_cache_start at ??:?

I (719) esp_image: segment 5: paddr=0x000426c8 vaddr=0x400860a8 size=0x01f4c (␣
↪ 8012)
0x400860a8: prvAddNewTaskToReadyList at esp-idf/esp-idf/components/freertos/
↪tasks.c:4561

I (722) flash_encrypt: Encrypting partition 2 at offset 0x20000...
```

(下页继续)

(续上页)

```
I (13229) flash_encrypt: Flash encryption completed
I (13229) boot: Resetting with flash encryption enabled...
```

启用 Flash 加密后，在下次启动时输出将显示已启用 Flash 加密。

```
rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0018,len:4
load:0x3fff001c,len:8452
load:0x40078000,len:13652
ho 0 tail 12 room 4
load:0x40080400,len:6664
entry 0x40080764
I (30) boot: ESP-IDF v4.0-dev-850-gc4447462d-dirty 2nd stage bootloader
I (30) boot: compile time 16:32:53
I (31) boot: Enabling RNG early entropy source...
I (37) boot: SPI Speed      : 40MHz
I (41) boot: SPI Mode       : DIO
I (45) boot: SPI Flash Size : 4MB
I (49) boot: Partition Table:
I (52) boot: ## Label            Usage          Type ST Offset   Length
I (60) boot:  0 nvs              WiFi data        01 02 0000a000 00006000
I (67) boot:  1 phy_init         RF data          01 01 00010000 00001000
I (75) boot:  2 factory          factory app      00 00 00020000 00100000
I (82) boot: End of partition table
I (86) esp_image: segment 0: paddr=0x00020020 vaddr=0x3f400020 size=0x0808c (␣
↪32908) map
I (107) esp_image: segment 1: paddr=0x000280b4 vaddr=0x3ffb0000 size=0x01ea4 (␣
↪ 7844) load
I (111) esp_image: segment 2: paddr=0x00029f60 vaddr=0x40080000 size=0x00400 (␣
↪ 1024) load
0x40080000: _WindowOverflow4 at esp-idf/esp-idf/components/freertos/xtensa_
↪vectors.S:1778

I (116) esp_image: segment 3: paddr=0x0002a368 vaddr=0x40080400 size=0x05ca8 (␣
↪23720) load
I (134) esp_image: segment 4: paddr=0x00030018 vaddr=0x400d0018 size=0x126a8 (␣
↪75432) map
```

(下页继续)

(续上页)

```
0x400d0018: _flash_cache_start at ??:?

I (162) esp_image: segment 5: paddr=0x000426c8 vaddr=0x400860a8 size=0x01f4c (␣
↪ 8012) load
0x400860a8: prvAddNewTaskToReadyList at esp-idf/esp-idf/components/freertos/
↪tasks.c:4561

I (171) boot: Loaded app from partition at offset 0x20000
I (171) boot: Checking flash encryption...
I (171) flash_encrypt: flash encryption is enabled (3 plaintext flashes left)
I (178) boot: Disabling RNG early entropy source...
I (184) cpu_start: Pro cpu up.
I (188) cpu_start: Application information:
I (193) cpu_start: Project name:     flash-encryption
I (198) cpu_start: App version:      v4.0-dev-850-gc4447462d-dirty
I (205) cpu_start: Compile time:     Jun 17 2019 16:32:52
I (211) cpu_start: ELF file SHA256:  8770c886bdf561a7...
I (217) cpu_start: ESP-IDF:          v4.0-dev-850-gc4447462d-dirty
I (224) cpu_start: Starting app cpu, entry point is 0x40080e4c
0x40080e4c: call_start_cpu1 at esp-idf/esp-idf/components/esp32/cpu_start.c:265

I (0) cpu_start: App cpu up.
I (235) heap_init: Initializing. RAM available for dynamic allocation:
I (241) heap_init: At 3FFAE6E0 len 00001920 (6 KiB): DRAM
I (247) heap_init: At 3FFB2EC8 len 0002D138 (180 KiB): DRAM
I (254) heap_init: At 3FFE0440 len 00003AE0 (14 KiB): D/IRAM
I (260) heap_init: At 3FFE4350 len 0001BCB0 (111 KiB): D/IRAM
I (266) heap_init: At 40087FF4 len 0001800C (96 KiB): IRAM
I (273) cpu_start: Pro cpu start user code
I (291) cpu_start: Starting scheduler on PRO CPU.
I (0) cpu_start: Starting scheduler on APP CPU.

Sample program to check Flash Encryption
This is ESP32 chip with 2 CPU cores, WiFi/BT/BLE, silicon revision 1, 4MB␣
↪external flash
Flash encryption feature is enabled
Flash encryption mode is DEVELOPMENT
Flash in encrypted mode with flash_crypt_cnt = 1
Halting...
```

在此阶段，如果用户希望以加密格式将已修改的明文应用程序映像更新到 Flash 中，可使用以下命令：

```
idf.py encrypted-app-flash monitor
```

加密多重分区

如果所有分区都需以加密格式更新，则可使用以下命令：

```
idf.py encrypted-flash monitor
```

使用主机生成的 Flash 加密密钥

可在主机中预生成 Flash 加密密钥，并将其烧录到 ESP32 的 eFuse 密钥块中。这样，无需明文 Flash 更新便可以在主机上预加密数据并将其烧录到 ESP32 中。该功能允许在开发模式 和释放模式 modes 两模式下加密烧录。

- 确保您的 ESP32 设备有*Flash 加密过程中使用的 eFuse* 中所示 Flash 加密 eFuse 的默认设置。
- 使用 espsecure.py 随机生成一个密钥:

```
espsecure.py generate_flash_encryption_key my_flash_encryption_key.bin
```

- 将该密钥烧录到设备上（一次性）。**该步骤须在第一次加密启动前完成**，否则 ESP32 将随机生成一个软件无权限访问或修改的密钥:

```
espefuse.py --port PORT burn_key flash_encryption my_flash_encryption_key.bin
```

- 在第二阶段引导加载程序中启用 Flash 加密支持。请前往*Project Configuration Menu*，选择 “Security Features”。
- 选择*Enable flash encryption on boot*。
- 模式默认设置为 **开发模式**。
- 在引导加载程序 config 下选择适当详细程度的日志。
- 启用 Flash 加密将增大引导加载程序，因而可能需要更新分区表偏移。可参见 See *Bootloader Size*。
- 保存配置并退出。

构建并烧录完整的映像包括：引导加载程序、分区表和 app。这些分区最初以未加密形式写入 Flash

```
idf.py flash monitor
```

下次启动时，第二阶段引导加载程序将加密 Flash 的 app 分区并重置该分区。现在，示例应用程序将在运行时解密并执行命令。

在此阶段，如果用户希望将新的明文应用程序映像更新到 Flash，应调用以下命令

```
idf.py encrypted-app-flash monitor
```

如何以加密格式重新编程所有分区，可参考加密多重分区。

释放模式

在释放模式下，UART 引导加载程序无法执行 Flash 加密操作，**只能**使用 OTA 方案下载新的明文映像，该方案将在写入 Flash 前加密明文映像。

- 确保您的 ESP32 设备有*Flash 加密过程中使用的 eFuse* 中所示 Flash 加密 eFuse 的默认设置。
- 在第二阶段引导加载程序中启用 Flash 加密支持。请前往*Project Configuration Menu*，选择 “Security Features”。
- 选择*Enable flash encryption on boot*。
- 选择 **释放模式**，模式默认设置为 **开发模式**。请注意，**一旦选择了释放模式，“download_dis_encrypt“ 和 “download_dis_decrypt“ eFuse 位将被编程为禁止 UART 引导加载程序访问 Flash 的内容**。
- 在引导加载程序 config 下选择适当详细程度的日志。
- 启用 Flash 加密将增大引导加载程序，因而可能需要更新分区表偏移。可参见 See *Bootloader Size*。
- 保存配置并退出。

构建并烧录完整的映像包括：引导加载程序、分区表和 app。这些分区最初以未加密形式写入 Flash

```
idf.py flash monitor
```

下次启动时，第二阶段引导加载程序将加密 Flash app 分区并重置该分区。现在，示例应用程序应正确执行命令。

一旦在释放模式下启用 Flash 加密，引导加载程序将写保护 `FLASH_CRYPT_CNT` eFuse。

应使用 OTA 方案对字段中的明文进行后续更新。详情可参见*OTA*。

可能出现的错误

启用 Flash 加密后，如果 `FLASH_CRYPT_CNT` eFuse 值中有奇数位，则所有（标有加密标志的）分区都应包含加密密文。以下为 ESP32 加载明文数据会产生的三种典型错误情况：

1. 如果通过明文引导加载程序映像重新更新了引导加载程序分区，则 ROM 加载器将无法加载引导加载程序，并会显示以下错误类型：

```
rst:0x3 (SW_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
flash read err, 1000
ets_main.c 371
ets Jun  8 2016 00:22:57
```

(下页继续)

(续上页)

```
rst:0x7 (TG0WDT_SYS_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
flash read err, 1000
ets_main.c 371
ets Jun  8 2016 00:22:57

rst:0x7 (TG0WDT_SYS_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
flash read err, 1000
ets_main.c 371
ets Jun  8 2016 00:22:57

rst:0x7 (TG0WDT_SYS_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
flash read err, 1000
ets_main.c 371
ets Jun  8 2016 00:22:57

rst:0x7 (TG0WDT_SYS_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
flash read err, 1000
ets_main.c 371
ets Jun  8 2016 00:22:57
```

2. 如果引导加载程序已加密，但使用明文分区表映像重新更新了分区表，则引导加载程序将无法读取分区表，并会显示以下错误类型：

```
rst:0x3 (SW_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0018,len:4
load:0x3fff001c,len:10464
ho 0 tail 12 room 4
load:0x40078000,len:19168
load:0x40080400,len:6664
entry 0x40080764
I (60) boot: ESP-IDF v4.0-dev-763-g2c55fae6c-dirty 2nd stage bootloader
I (60) boot: compile time 19:15:54
I (62) boot: Enabling RNG early entropy source...
I (67) boot: SPI Speed      : 40MHz
I (72) boot: SPI Mode       : DIO
I (76) boot: SPI Flash Size : 4MB
```

(下页继续)

(续上页)

```
E (80) flash_parts: partition 0 invalid magic number 0x94f6
E (86) boot: Failed to verify partition table
E (91) boot: load partition table error!
```

3. 如果引导加载程序和分区表已加密，但使用明文应用程序映像重新更新了应用程序，则引导加载程序将无法加载新的应用程序，并会显示以下错误类型：

```
rst:0x3 (SW_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0018,len:4
load:0x3fff001c,len:8452
load:0x40078000,len:13616
load:0x40080400,len:6664
entry 0x40080764
I (56) boot: ESP-IDF v4.0-dev-850-gc4447462d-dirty 2nd stage bootloader
I (56) boot: compile time 15:37:14
I (58) boot: Enabling RNG early entropy source...
I (64) boot: SPI Speed      : 40MHz
I (68) boot: SPI Mode       : DIO
I (72) boot: SPI Flash Size : 4MB
I (76) boot: Partition Table:
I (79) boot: ## Label            Usage          Type ST Offset   Length
I (87) boot:  0 nvs              WiFi data        01 02 0000a000 00006000
I (94) boot:  1 phy_init         RF data          01 01 00010000 00001000
I (102) boot:  2 factory          factory app      00 00 00020000 00100000
I (109) boot: End of partition table
E (113) esp_image: image at 0x20000 has invalid magic byte
W (120) esp_image: image at 0x20000 has invalid SPI mode 108
W (126) esp_image: image at 0x20000 has invalid SPI size 11
E (132) boot: Factory app partition is not bootable
E (138) boot: No bootable app partitions in the partition table
```

5.9.5 Flash 加密的要点

- 使用 AES-256 加密 Flash 的内容。Flash 加密密钥存储于 eFuse 内部的芯片中，并（默认）受保护防止软件访问。
- *flash* 加密算法采用的是 AES-256，其中密钥随着 Flash 的每个 32 字节块的偏移地址 “调整”。这意味着，每个 32 字节块（2 个连续的 16 字节 AES 块）使用从 Flash 加密密钥中产生的一个特殊密钥进行

加密。

- 通过 ESP32 的 Flash 缓存映射功能，Flash 可支持透明访问——读取任何映射到地址空间的 Flash 区域时，都将透明解密该区域。

  为便于访问，某些数据分区最好保持未加密状态，或者也可使用对已加密数据无效的 Flash 友好型更新算法。由于 NVS 库无法与 Flash 加密直接兼容，因此无法加密非易失性存储器的 NVS 分区。详情可参见*NVS 加密*。

- 如果可能已启用 Flash 加密，则编写*使用加密 flash* 的代码时，编程人员须小心谨慎。
- 如果已启用安全启动，则重新烧录加密设备的引导加载程序则需要“可重新烧录”的安全启动摘要（可参见*Flash 加密与安全启动*）。

注解： 同时启用安全启动和 Flash 加密后，引导加载程序 app 二进制文件 `bootloader.bin` 可能会过大。参见*Bootloader Size*。

重要： 在首次启动加密过程中，请勿中断 ESP32 的电源。如果电源中断，Flash 的内容将受到破坏，并需要重新烧录未加密数据。而这类重新烧录将不计入烧录限制次数。

5.9.6 使用加密的 Flash

ESP32 app 代码可通过调用函数*esp_flash_encryption_enabled()* 来确认当前是否已启用 Flash 加密。同时，设备可通过调用函数*esp_get_flash_encryption_mode()* 来识别使用的 Flash 加密模式。

启用 Flash 加密后，使用代码访问 Flash 内容时需加注意。

Flash 加密的范围

只要 `FLASH_CRYPT_CNT` eFuse 设置为奇数位的值，所有通过 MMU 的 Flash 缓存访问的 Flash 内容都将被透明解密。包括：

- Flash 中可执行的应用程序代码 (IROM)。
- 所有存储于 Flash 中的只读数据 (DROM)。
- 通过函数 `spi_flash_mmap()` 访问的任意数据。
- ROM 引导加载程序读取的软件引导加载程序映像。

重要： MMU Flash 缓存将无条件解密所有数据。Flash 中未加密存储的数据将通过 Flash 缓存“被透明解密”，并在软件中存储为随机垃圾数据。

读取加密的 Flash

如在不使用 Flash 缓存 MMU 映射的情况下读取数据，推荐使用分区读取函数*esp_partition_read()*。使用该函数时，只有从加密分区读取的数据才会被解密。其他分区的数据将以未加密形式读取。这样，软件便能同样访问加密和未加密的 Flash。

通过其他 SPI 读取 API 读取的数据均未解密：

- 通过函数 `spi_flash_read()` 读取的数据均未解密。
- 通过 ROM 函数 `SPIRead()` 读取的数据均未解密（esp-idf app 不支持该函数）。
- 使用非易失性存储器 (NVS) API 存储的数据始终从 Flash 加密的角度进行存储和读取解密。如有需要，则由库提供加密功能。详情可参见*NVS 加密*。

写入加密的 Flash

在可能的情况下，推荐使用分区写入函数 `esp_partition_write`。使用该函数时，只有向加密分区写入的数据才会被加密。而写入其他分区的数据均未加密。这样，软件便可同样访问加密和未加密的 Flash。

当 write_encrypted 参数设置为 “是” 时，函数 `esp_spi_flash_write` 将写入数据。否则，数据将以未加密形式写入。

ROM 函数 `esp_rom_spiflash_write_encrypted` 将在 Flash 中写入加密数据，而 ROM 函数 `SPIWrite` 将在 Flash 中写入未加密数据（esp-idf app 不支持上述函数）。

由于数据均采用块加密方式，加密数据最小的写入大小为 16 字节（16 字节对齐）。

5.9.7 更新加密的 Flash

OTA 更新

只要使用了函数 `esp_partition_write`，则加密分区的 OTA 更新将自动以加密形式写入。

5.9.8 关闭 Flash 加密

若因某些原因意外启用了 Flash 加密，则接下来烧录明文数据时将使 ESP32 软砖（设备不断重启，并报错 `flash read err, 1000`）。

可通过写入 `FLASH_CRYPT_CNT` eFuse 再次关闭 Flash 加密（仅适用于开发模式下）：

- 首先，前往*Project Configuration Menu*，在 “安全性能” 目录下关闭*启用 Flash 加密启动*。
- 退出 menuconfig 并保存最新配置。
- 再次运行 `idf.py menuconfig` 并复核是否确认已关闭该选项！如果该选项仍处于已启用状态，则引导加载程序会在启动后立即重新启用加密。
- 在未启用 Flash 加密的状态下，运行 `idf.py flash` 构建并烧录新的引导加载程序与 app。

- 运行 `espefuse.py` （`components/esptool_py/esptool` 中）以关闭 FLASH_CRYPT_CNT:

```
espefuse.py burn_efuse FLASH_CRYPT_CNT
```

重置 ESP32，Flash 加密应处于关闭状态，引导加载程序将正常启动。

5.9.9 Flash 加密的局限性

Flash 加密可防止从加密 Flash 中读取明文，从而保护固件防止未经授权的读取与修改。了解 Flash 加密系统的局限之处亦十分重要：

- Flash 加密功能与密钥同样稳固。因而，推荐您首次启动设备时在设备上生成密钥（默认行为）。如果在设备外生成密钥，请确保遵循正确的后续步骤。
- 并非所有数据都是加密存储。因而在 Flash 上存储数据时，请检查您使用的存储方式（库、API 等）是否支持 Flash 加密。
- Flash 加密无法防止攻击者获取 Flash 的高层次布局信息。这是因为同一个 AES 密钥要用于每对相邻的 16 字节 AES 块。当这些相邻的 16 字节块中包含相同内容时（如空白或填充区域），这些字节块将加密以产生匹配的加密块对。这可能使得攻击者可在加密设备间进行高层次对比（例如，确认两设备是否可能在运行相同的固件版本）。
- 出于相同原因，攻击者始终可获知一对相邻的 16 字节块（32 字节对齐）何时包含相同内容。因此，在 Flash 上存储敏感数据时应牢记这点，并进行相关设置避免该情况发生（可使用计数器字节或每 16 字节设置不同的值即可）。
- 单独使用 Flash 加密可能无法防止攻击者修改本设备的固件。为防止设备上运行未经授权的固件，可搭配 Flash 加密使用*安全启动*。

5.9.10 Flash 加密与安全启动

推荐搭配使用 Flash 加密与安全启动。但是，如果已启用安全启动，则重新烧录设备时会受到其他限制：

- *OTA 更新* 不受限制（如果新的 app 已使用安全启动签名密钥进行正确签名）。
- 只有当选择*可再次烧录* 安全启动模式，且安全启动密钥已预生成并烧录至 ESP32（可参见*安全启动*），则*明文串行 flash 更新* 可实现。在该配置下，`idf.py bootloader` 将生成简化的引导加载程序和安全启动摘要文件，用于在偏移量 0x0 处进行烧录。当进行明文串行重新烧录步骤时，须在烧录其他明文数据前重新烧录此文件。
- 假设未重新烧录引导加载程序，*使用预生成的 Flash 加密密钥重新烧录* 仍可实现。重新烧录引导加载程序时，需在安全启动配置中启用相同的*可重新烧录* 选项。

5.9.11 使用无安全启动的 Flash 加密

尽管 Flash 加密与安全启动可独立使用，但强烈建议您将这二者 **搭配使用**以确保更高的安全性。

5.9.12 Flash 加密的高级功能

以下信息可帮助您使用 Flash 加密的高级功能：

加密分区标志

部分分区默认为已加密。除此之外，可将任意分区标记为需加密：

在分区表 文档对 CSV 文件的描述中有标志字段。

该字段通常保留为空白。如果在字段中写入”encrypted”，则这个分区将在分区表中标记为已加密，此处写入的数据也视为加密数据（app 分区同样适用）：

```
# Name,   Type, SubType, Offset,  Size, Flags
nvs,      data, nvs,     0x9000,  0x6000
phy_init, data, phy,     0xf000,  0x1000
factory,  app,  factory, 0x10000, 1M
secret_data, 0x40, 0x01, 0x20000, 256K, encrypted
```

- 默认分区表都不包含任何加密数据分区。
- “app” 分区一般都视为加密分区，因此无需将其标记为已加密。
- 如果未启用 Flash 加密，则” encrypted” 标记无效。
- 可将带有 phy_init 数据的可选 phy 分区标记为已加密，保护该数据防止物理访问读取或修改。
- nvs 分区无法标记为已加密。

启用 UART 引导加载程序加密/解密

默认情况下，首次启动 Flash 加密过程中将烧录 eFuse DISABLE_DL_ENCRYPT、DISABLE_DL_DECRYPT 和 DISABLE_DL_CACHE：

- DISABLE_DL_ENCRYPT 在 UART 引导加载程序启动模式下运行时，终止 Flash 加密操作。
- DISABLE_DL_DECRYPT 在 UART 引导加载程序模式下运行时，终止透明 Flash 解密（即使 FLASH_CRYPT_CNT 已设置为在正常操作中启用 Flash 透明解密）。
- DISABLE_DL_CACHE 在 UART 引导加载程序模式下运行时终止整个 MMU flash 缓存。

为了完整保存数据，可在首次启动前仅烧录部分 eFuse，并写保护其他部分（未设置值为 0）。例如：

```
espefuse.py --port PORT burn_efuse DISABLE_DL_DECRYPT
espefuse.py --port PORT write_protect_efuse DISABLE_DL_ENCRYPT
```

（请注意，一个写保护位即可关闭这 3 个 eFuse，因此，写保护一个 eFuse 将写保护上述所有 eFuse。所以，在写保护前须设置任意位）。

重要： 由于 esptool.py 不支持读取加密的 Flash，因此目前基本无法通过写保护这些 eFuse 来将其保持为未设状态。

重要： 如果保留 DISABLE_DL_DECRYPT 未设置（为 0），则实际上将使 Flash 加密无效，因为此时有物理访问权限的攻击者便可使用 UART 引导加载程序模式（使用自定义存根代码）读取 Flash 的内容。

设置 FLASH_CRYPT_CONFIG

FLASH_CRYPT_CONFIG eFuse 决定 Flash 加密密钥中随块偏移“调整”的位数。详情可参见*Flash 加密算法*。

首次启动引导加载程序时，该值始终设置为最大 *0xF*。

可手动写入这些 eFuse，并在首次启动前对其写保护，以便选择不同的调整值。但不推荐该操作。

当 FLASH_CRYPT_CONFIG 的值为 0 时，强烈建议始终不对其进行写保护。如果该 eFuse 设置为 0，则 Flash 加密密钥中无调整位，且 Flash 加密算法相当于 AES ECB 模式。

5.9.13 技术细节

下节将提供 Flash 加密操作的相关信息。

Flash 加密算法

- AES-256 在 16 字节的数据块上运行。Flash 加密引擎在 32 字节的数据块和 2 个串行 AES 块上加密或解密数据。
- Flash 加密的主密钥存储于 eFuse (BLOCK1) 中，默认受保护防止进一步写入或软件读取。
- AES-256 密钥大小为 256 位（32 字节），从 eFuse block 1 中读取。硬件 AES 引擎使用反字节序密钥于 eFuse 块中存储的字节序。
  - 如果 CODING_SCHEME eFuse 设置为 0（默认“无”编码方案），则 eFuse 密钥块为 256 位，且密钥按原方式存储（反字节序）。
  - 如果 CODING_SCHEME eFuse 设置为 1（3/4 编码），则 eFuse 密钥块为 192 位（反字节序），信息熵总量减少。硬件 Flash 加密仍在 256 字节密钥上运行，在读取后（字节序未反向），密钥扩展为 `key = key[0:255] + key[64:127]`。
- Flash 加密中使用了逆向 AES 算法，因此 Flash 加密的“加密”操作相当于 AES 解密，而其“解密”操作则相当于 AES 加密。这是为了优化性能，不会影响算法的有效性。
- 每个 32 字节块 (2 个相邻的 16 字节 AES 块) 都由一个特殊的密钥进行加密。该密钥由 eFuse 中 Flash 加密的主密钥产生，并随 Flash 中该字节块的偏移进行 XOR 运算（一次“密钥调整”）。

- 具体调整量取决于 FLASH_CRYPT_CONFIG eFuse 的设置。该 eFuse 共 4 位，每位可对特定范围的密钥位进行 XOR 运算：
  - Bit 1，对密钥的 0-66 位进行 XOR 运算。
  - Bit 2，对密钥的 67-131 位进行 XOR 运算。
  - Bit 3，对密钥的 132-194 位进行 XOR 运算。
  - Bit 4，对密钥的 195-256 位进行 XOR 运算。

  建议将 FLASH_CRYPT_CONFIG 的值始终保留为默认值 *0xF*，这样所有密钥位都随块偏移进行 XOR 运算。详情可参见设置 *FLASH_CRYPT_CONFIG*。
- 块偏移的 19 个高位（第 5-23 位）由 Flash 加密的主密钥进行 XOR 运算。选定该范围的原因为：Flash 的最大尺寸为 16MB（24 位），每个块大小为 32 字节，因而 5 个最低有效位始终为 0。
- 从 19 个块偏移位中每个位到 Flash 加密密钥的 256 位都有一个特殊的映射，以决定与哪个位进行 XOR 运算。有关完整映射可参见 espsecure.py 源代码中的变量 _FLASH_ENCRYPTION_TWEAK_PATTERN。
- 有关在 Python 中实现的完整 Flash 加密算法，可参见 espsecure.py 源代码中的函数 *_flash_encryption_operation()*。

5.10 ESP-IDF FreeRTOS SMP Changes

5.10.1 Overview

The vanilla FreeRTOS is designed to run on a single core. However the ESP32 is dual core containing a Protocol CPU (known as **CPU 0** or **PRO_CPU**) and an Application CPU (known as **CPU 1** or **APP_CPU**). The two cores are identical in practice and share the same memory. This allows the two cores to run tasks interchangeably between them.

The ESP-IDF FreeRTOS is a modified version of vanilla FreeRTOS which supports symmetric multiprocessing (SMP). ESP-IDF FreeRTOS is based on the Xtensa port of FreeRTOS v8.2.0. This guide outlines the major differences between vanilla FreeRTOS and ESP-IDF FreeRTOS. The API reference for vanilla FreeRTOS can be found via http://www.freertos.org/a00106.html

For information regarding features that are exclusive to ESP-IDF FreeRTOS, see *ESP-IDF FreeRTOS Additions*.

Backported Features: Although ESP-IDF FreeRTOS is based on the Xtensa port of FreeRTOS v8.2.0, a number of FreeRTOS v9.0.0 features have been backported to ESP-IDF.

Tasks and Task Creation: Use *xTaskCreatePinnedToCore()* or *xTaskCreateStaticPinnedToCore()* to create tasks in ESP-IDF FreeRTOS. The last parameter of the two functions is xCoreID. This parameter specifies which core the task is pinned to. Acceptable values are 0 for **PRO_CPU**, 1 for **APP_CPU**, or tskNO_AFFINITY which allows the task to run on both.

Round Robin Scheduling: The ESP-IDF FreeRTOS scheduler will skip tasks when implementing Round-Robin scheduling between multiple tasks in the Ready state that are of the same priority. To avoid this behavior, ensure that those tasks either enter a blocked state, or are distributed across a wider range of priorities.

Scheduler Suspension: Suspending the scheduler in ESP-IDF FreeRTOS will only affect the scheduler on the the calling core. In other words, calling *vTaskSuspendAll()* on **PRO_CPU** will not prevent **APP_CPU** from scheduling, and vice versa. Use critical sections or semaphores instead for simultaneous access protection.

Tick Interrupt Synchronicity: Tick interrupts of **PRO_CPU** and **APP_CPU** are not synchronized. Do not expect to use *vTaskDelay()* or *vTaskDelayUntil()* as an accurate method of synchronizing task execution between the two cores. Use a counting semaphore instead as their context switches are not tied to tick interrupts due to preemption.

Critical Sections & Disabling Interrupts: In ESP-IDF FreeRTOS, critical sections are implemented using mutexes. Entering critical sections involve taking a mutex, then disabling the scheduler and interrupts of the calling core. However the other core is left unaffected. If the other core attemps to take same mutex, it will spin until the calling core has released the mutex by exiting the critical section.

Floating Point Aritmetic: The ESP32 supports hardware acceleration of single precision floating point arithmetic (`float`). However the use of hardware acceleration leads to some behavioral restrictions in ESP-IDF FreeRTOS. Therefore, tasks that utilize `float` will automatically be pinned to a core if not done so already. Furthermore, `float` cannot be used in interrupt service routines.

Task Deletion: Task deletion behavior has been backported from FreeRTOS v9.0.0 and modified to be SMP compatible. Task memory will be freed immediately when *vTaskDelete()* is called to delete a task that is not currently running and not pinned to the other core. Otherwise, freeing of task memory will still be delegated to the Idle Task.

Thread Local Storage Pointers & Deletion Callbacks: ESP-IDF FreeRTOS has backported the Thread Local Storage Pointers (TLSP) feature. However the extra feature of Deletion Callbacks has been added. Deletion callbacks are called automatically during task deletion and are used to free memory pointed to by TLSP. Call *vTaskSetThreadLocalStoragePointerAndDelCallback()* to set TLSP and Deletion Callbacks.

Configuring ESP-IDF FreeRTOS: Several aspects of ESP-IDF FreeRTOS can be set in the project configuration (`idf.py menuconfig`) such as running ESP-IDF in Unicore (single core) Mode, or configuring the number of Thread Local Storage Pointers each task will have.

5.10.2 Backported Features

The following features have been backported from FreeRTOS v9.0.0 to ESP-IDF.

Static Alocation

This feature has been backported from FreeRTOS v9.0.0 to ESP-IDF. The *CONFIG_FREERTOS_SUPPORT_STATIC_ALLOCATION* option must be enabled in *menuconfig* in order for static allocation functions to be available. Once enabled, the following functions can be called···

- *xTaskCreateStatic()* (see *Backporting Notes* below)
- *xQueueCreateStatic*
- *xSemaphoreCreateBinaryStatic*
- *xSemaphoreCreateCountingStatic*
- *xSemaphoreCreateMutexStatic*
- *xSemaphoreCreateRecursiveMutexStatic*
- *xTimerCreateStatic()* (see *Backporting Notes* below)
- *xEventGroupCreateStatic()*

Other Features

- *vTaskSetThreadLocalStoragePointer()* (see *Backporting Notes* below)
- *pvTaskGetThreadLocalStoragePointer()* (see *Backporting Notes* below)
- *vTimerSetTimerID()*
- *xTimerGetPeriod()*
- *xTimerGetExpiryTime()*
- *pcQueueGetName()*
- *uxSemaphoreGetCount*

Backporting Notes

1) *xTaskCreateStatic()* has been made SMP compatible in a similar fashion to *xTaskCreate()* (see *Tasks and Task Creation*). Therefore *xTaskCreateStaticPinnedToCore()* can also be called.

2) Although vanilla FreeRTOS allows the Timer feature's daemon task to be statically allocated, the daemon task is always dynamically allocated in ESP-IDF. Therefore `vApplicationGetTimerTaskMemory` **does not** need to be defined when using statically allocated timers in ESP-IDF FreeRTOS.

3) The Thread Local Storage Pointer feature has been modified in ESP-IDF FreeRTOS to include Deletion Callbacks (see *Thread Local Storage Pointers & Deletion Callbacks*). Therefore the function *vTaskSetThreadLocalStoragePointerAndDelCallback()* can also be called.

5.10.3 Tasks and Task Creation

Tasks in ESP-IDF FreeRTOS are designed to run on a particular core, therefore two new task creation functions have been added to ESP-IDF FreeRTOS by appending `PinnedToCore` to the names of the task creation functions in vanilla FreeRTOS. The vanilla FreeRTOS functions of *`xTaskCreate()`* and *`xTaskCreateStatic()`* have led to the addition of *`xTaskCreatePinnedToCore()`* and *`xTaskCreateStaticPinnedToCore()`* in ESP-IDF FreeRTOS (see *Backported Features*).

For more details see freertos/task.c

The ESP-IDF FreeRTOS task creation functions are nearly identical to their vanilla counterparts with the exception of the extra parameter known as `xCoreID`. This parameter specifies the core on which the task should run on and can be one of the following values.

- `0` pins the task to **PRO_CPU**
- `1` pins the task to **APP_CPU**
- `tskNO_AFFINITY` allows the task to be run on both CPUs

For example `xTaskCreatePinnedToCore(tsk_callback, "APP_CPU Task", 1000, NULL, 10, NULL, 1)` creates a task of priority 10 that is pinned to **APP_CPU** with a stack size of 1000 bytes. It should be noted that the `uxStackDepth` parameter in vanilla FreeRTOS specifies a task's stack depth in terms of the number of words, whereas ESP-IDF FreeRTOS specifies the stack depth in terms of bytes.

Note that the vanilla FreeRTOS functions *`xTaskCreate()`* and *`xTaskCreateStatic()`* have been defined in ESP-IDF FreeRTOS as inline functions which call *`xTaskCreatePinnedToCore()`* and *`xTaskCreateStaticPinnedToCore()`* respectively with `tskNO_AFFINITY` as the `xCoreID` value.

Each Task Control Block (TCB) in ESP-IDF stores the `xCoreID` as a member. Hence when each core calls the scheduler to select a task to run, the `xCoreID` member will allow the scheduler to determine if a given task is permitted to run on the core that called it.

5.10.4 Scheduling

The vanilla FreeRTOS implements scheduling in the `vTaskSwitchContext()` function. This function is responsible for selecting the highest priority task to run from a list of tasks in the Ready state known as the Ready Tasks List (described in the next section). In ESP-IDF FreeRTOS, each core will call `vTaskSwitchContext()` independently to select a task to run from the Ready Tasks List which is shared between both cores. There are several differences in scheduling behavior between vanilla and ESP-IDF FreeRTOS such as differences in Round Robin scheduling, scheduler suspension, and tick interrupt synchronicity.

Round Robin Scheduling

Given multiple tasks in the Ready state and of the same priority, vanilla FreeRTOS implements Round Robin scheduling between each task. This will result in running those tasks in turn each time the scheduler is called (e.g. every tick interrupt). On the other hand, the ESP-IDF FreeRTOS scheduler may skip tasks when Round Robin scheduling multiple Ready state tasks of the same priority.

The issue of skipping tasks during Round Robin scheduling arises from the way the Ready Tasks List is implemented in FreeRTOS. In vanilla FreeRTOS, `pxReadyTasksList` is used to store a list of tasks that are in the Ready state. The list is implemented as an array of length `configMAX_PRIORITIES` where each element of the array is a linked list. Each linked list is of type `List_t` and contains TCBs of tasks of the same priority that are in the Ready state. The following diagram illustrates the `pxReadyTasksList` structure.

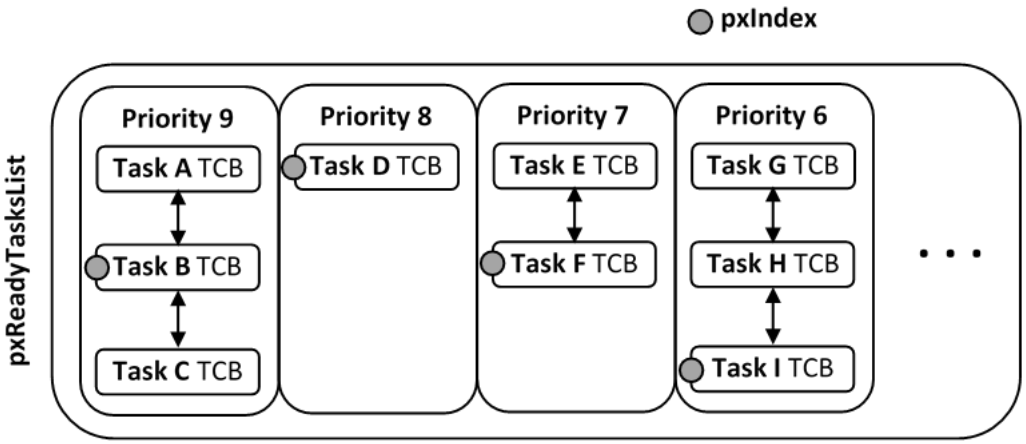


图 2: Illustration of FreeRTOS Ready Task List Data Structure

Each linked list also contains a `pxIndex` which points to the last TCB returned when the list was queried. This index allows the `vTaskSwitchContext()` to start traversing the list at the TCB immediately after `pxIndex` hence implementing Round Robin Scheduling between tasks of the same priority.

In ESP-IDF FreeRTOS, the Ready Tasks List is shared between cores hence `pxReadyTasksList` will contain tasks pinned to different cores. When a core calls the scheduler, it is able to look at the `xCoreID` member of each TCB in the list to determine if a task is allowed to run on calling the core. The ESP-IDF FreeRTOS `pxReadyTasksList` is illustrated below.

Therefore when **PRO_CPU** calls the scheduler, it will only consider the tasks in blue or purple. Whereas when **APP_CPU** calls the scheduler, it will only consider the tasks in orange or purple.

Although each TCB has an `xCoreID` in ESP-IDF FreeRTOS, the linked list of each priority only has a single `pxIndex`. Therefore when the scheduler is called from a particular core and traverses the linked list, it will skip all TCBs pinned to the other core and point the pxIndex at the selected task. If the other core then calls the scheduler, it will traverse the linked list starting at the TCB immediately after `pxIndex`. Therefore, TCBs skipped on the previous scheduler call from the other core would not be considered on the current scheduler call. This issue is demonstrated in the following illustration.

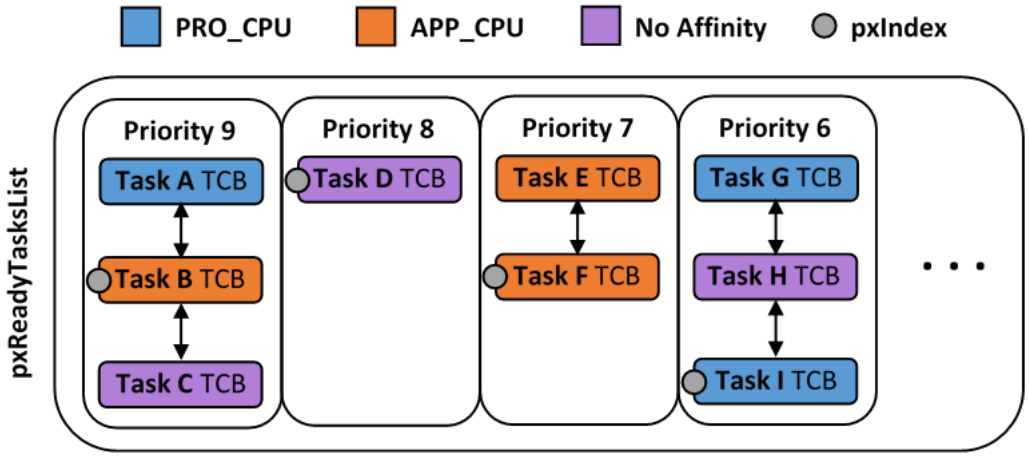


图 3: Illustration of FreeRTOS Ready Task List Data Structure in ESP-IDF

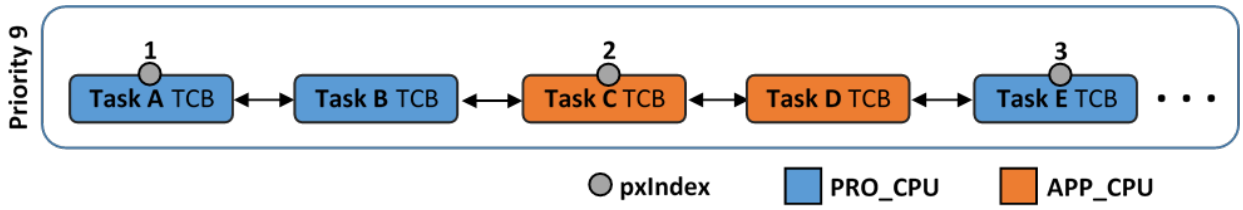


图 4: Illustration of pxIndex behavior in ESP-IDF FreeRTOS

Referring to the illustration above, assume that priority 9 is the highest priority, and none of the tasks in priority 9 will block hence will always be either in the running or Ready state.

1) **PRO_CPU** calls the scheduler and selects Task A to run, hence moves `pxIndex` to point to Task A

2) **APP_CPU** calls the scheduler and starts traversing from the task after `pxIndex` which is Task B. However Task B is not selected to run as it is not pinned to **APP_CPU** hence it is skipped and Task C is selected instead. `pxIndex` now points to Task C

3) **PRO_CPU** calls the scheduler and starts traversing from Task D. It skips Task D and selects Task E to run and points `pxIndex` to Task E. Notice that Task B isn't traversed because it was skipped the last time **APP_CPU** called the scheduler to traverse the list.

4) The same situation with Task D will occur if **APP_CPU** calls the scheduler again as `pxIndex` now points to Task E

One solution to the issue of task skipping is to ensure that every task will enter a blocked state so that they are removed from the Ready Task List. Another solution is to distribute tasks across multiple priorities such that a given priority will not be assigned multiple tasks that are pinned to different cores.

Scheduler Suspension

In vanilla FreeRTOS, suspending the scheduler via *vTaskSuspendAll()* will prevent calls of `vTaskSwitchContext` from context switching until the scheduler has been resumed with *xTaskResumeAll()*. However servicing ISRs are still permitted. Therefore any changes in task states as a result from the current running task or ISRSs will not be executed until the scheduler is resumed. Scheduler suspension in vanilla FreeRTOS is a common protection method against simultaneous access of data shared between tasks, whilst still allowing ISRs to be serviced.

In ESP-IDF FreeRTOS, `xTaskSuspendAll()` will only prevent calls of `vTaskSwitchContext()` from switching contexts on the core that called for the suspension. Hence if **PRO_CPU** calls *vTaskSuspendAll()*, **APP_CPU** will still be able to switch contexts. If data is shared between tasks that are pinned to different cores, scheduler suspension is **NOT** a valid method of protection against simultaneous access. Consider using critical sections (disables interrupts) or semaphores (does not disable interrupts) instead when protecting shared resources in ESP-IDF FreeRTOS.

In general, it's better to use other RTOS primitives like mutex semaphores to protect against data shared between tasks, rather than *vTaskSuspendAll()*.

Tick Interrupt Synchronicity

In ESP-IDF FreeRTOS, tasks on different cores that unblock on the same tick count might not run at exactly the same time due to the scheduler calls from each core being independent, and the tick interrupts to each core being unsynchronized.

In vanilla FreeRTOS the tick interrupt triggers a call to `xTaskIncrementTick()` which is responsible for incrementing the tick counter, checking if tasks which have called *vTaskDelay()* have fulfilled their delay period, and moving those tasks from the Delayed Task List to the Ready Task List. The tick interrupt will then call the scheduler if a context switch is necessary.

In ESP-IDF FreeRTOS, delayed tasks are unblocked with reference to the tick interrupt on PRO_CPU as PRO_CPU is responsible for incrementing the shared tick count. However tick interrupts to each core might not be synchronized (same frequency but out of phase) hence when PRO_CPU receives a tick interrupt, APP_CPU might not have received it yet. Therefore if multiple tasks of the same priority are unblocked on the same tick count, the task pinned to PRO_CPU will run immediately whereas the task pinned to APP_CPU must wait until APP_CPU receives its out of sync tick interrupt. Upon receiving the tick interrupt, APP_CPU will then call for a context switch and finally switches contexts to the newly unblocked task.

Therefore, task delays should **NOT** be used as a method of synchronization between tasks in ESP-IDF FreeRTOS. Instead, consider using a counting semaphore to unblock multiple tasks at the same time.

5.10.5 Critical Sections & Disabling Interrupts

Vanilla FreeRTOS implements critical sections in `vTaskEnterCritical` which disables the scheduler and calls `portDISABLE_INTERRUPTS`. This prevents context switches and servicing of ISRs during a critical section. Therefore, critical sections are used as a valid protection method against simultaneous access in vanilla FreeRTOS.

On the other hand, the ESP32 has no hardware method for cores to disable each other's interrupts. Calling `portDISABLE_INTERRUPTS()` will have no effect on the interrupts of the other core. Therefore, disabling interrupts is **NOT** a valid protection method against simultaneous access to shared data as it leaves the other core free to access the data even if the current core has disabled its own interrupts.

For this reason, ESP-IDF FreeRTOS implements critical sections using mutexes, and calls to enter or exit a critical must provide a mutex that is associated with a shared resource requiring access protection. When entering a critical section in ESP-IDF FreeRTOS, the calling core will disable its scheduler and interrupts similar to the vanilla FreeRTOS implementation. However, the calling core will also take the mutex whilst the other core is left unaffected during the critical section. If the other core attempts to take the same mutex, it will spin until the mutex is released. Therefore, the ESP-IDF FreeRTOS implementation of critical sections allows a core to have protected access to a shared resource without disabling the other core. The other core will only be affected if it tries to concurrently access the same resource.

The ESP-IDF FreeRTOS critical section functions have been modified as follows···

- `taskENTER_CRITICAL(mux)`, `taskENTER_CRITICAL_ISR(mux)`, `portENTER_CRITICAL(mux)`, `portENTER_CRITICAL_ISR(mux)` are all macro defined to call `vTaskEnterCritical()`
- `taskEXIT_CRITICAL(mux)`, `taskEXIT_CRITICAL_ISR(mux)`, `portEXIT_CRITICAL(mux)`, `portEXIT_CRITICAL_ISR(mux)` are all macro defined to call `vTaskExitCritical()`
- `portENTER_CRITICAL_SAFE(mux)`, `portEXIT_CRITICAL_SAFE(mux)` macro identifies the context of execution, i.e ISR or Non-ISR, and calls appropriate critical section functions (`port*_CRITICAL` in Non-ISR and `port*_CRITICAL_ISR` in ISR) in order to be in compliance with Vanilla FreeRTOS.

For more details see freertos/include/freertos/portmacro.h and freertos/task.c

It should be noted that when modifying vanilla FreeRTOS code to be ESP-IDF FreeRTOS compatible, it is trivial to modify the type of critical section called as they are all defined to call the same function. As long as the same mutex is provided upon entering and exiting, the type of call should not matter.

5.10.6 Floating Point Aritmetic

The ESP32 supports hardware acceleration of single precision floating point arithmetic (`float`) via Floating Point Units (FPU, also known as coprocessors) attached to each core. The use of the FPUs imposes some behavioral restrictions on ESP-IDF FreeRTOS.

ESP-IDF FreeRTOS implements Lazy Context Switching for FPUs. In other words, the state of a core's FPU registers are not immediately saved when a context switch occurs. Therefore, tasks that utilize `float`

must be pinned to a particular core upon creation. If not, ESP-IDF FreeRTOS will automatically pin the task in question to whichever core the task was running on upon the task' s first use of `float`. Likewise due to Lazy Context Switching, interrupt service routines must also not use `float`.

ESP32 does not support hardware acceleration for double precision floating point arithmetic (`double`). Instead `double` is implemented via software hence the behavioral restrictions with regards to `float` do not apply to `double`. Note that due to the lack of hardware acceleration, `double` operations may consume significantly larger amount of CPU time in comparison to `float`.

5.10.7 Task Deletion

FreeRTOS task deletion prior to v9.0.0 delegated the freeing of task memory entirely to the Idle Task. Currently, the freeing of task memory will occur immediately (within *vTaskDelete()*) if the task being deleted is not currently running or is not pinned to the other core (with respect to the core *vTaskDelete()* is called on). TLSP deletion callbacks will also run immediately if the same conditions are met.

However, calling *vTaskDelete()* to delete a task that is either currently running or pinned to the other core will still result in the freeing of memory being delegated to the Idle Task.

5.10.8 Thread Local Storage Pointers & Deletion Callbacks

Thread Local Storage Pointers (TLSP) are pointers stored directly in the TCB. TLSP allow each task to have its own unique set of pointers to data structures. However task deletion behavior in vanilla FreeRTOS does not automatically free the memory pointed to by TLSP. Therefore if the memory pointed to by TLSP is not explicitly freed by the user before task deletion, memory leak will occur.

ESP-IDF FreeRTOS provides the added feature of Deletion Callbacks. Deletion Callbacks are called automatically during task deletion to free memory pointed to by TLSP. Each TLSP can have its own Deletion Callback. Note that due to the to *Task Deletion* behavior, there can be instances where Deletion Callbacks are called in the context of the Idle Tasks. Therefore Deletion Callbacks **should never attempt to block** and critical sections should be kept as short as possible to minimize priority inversion.

Deletion callbacks are of type `void (*TlsDeleteCallbackFunction_t)( int, void * )` where the first parameter is the index number of the associated TLSP, and the second parameter is the TLSP itself.

Deletion callbacks are set alongside TLSP by calling *vTaskSetThreadLocalStoragePointerAndDelCallback()*. Calling the vanilla FreeRTOS function *vTaskSetThreadLocalStoragePointer()* will simply set the TLSP' s associated Deletion Callback to *NULL* meaning that no callback will be called for that TLSP during task deletion. If a deletion callback is *NULL*, users should manually free the memory pointed to by the associated TLSP before task deletion in order to avoid memory leak.

CONFIG_FREERTOS_THREAD_LOCAL_STORAGE_POINTERS in menuconfig can be used to configure the number TLSP and Deletion Callbacks a TCB will have.

For more details see *FreeRTOS API reference*.

5.10.9 Configuring ESP-IDF FreeRTOS

The ESP-IDF FreeRTOS can be configured in the project configuration menu (`idf.py menuconfig`) under `Component Config/FreeRTOS`. The following section highlights some of the ESP-IDF FreeRTOS configuration options. For a full list of ESP-IDF FreeRTOS configurations, see *FreeRTOS*

CONFIG_FREERTOS_UNICORE will run ESP-IDF FreeRTOS only on **PRO_CPU**. Note that this is **not equivalent to running vanilla FreeRTOS**. Behaviors of multiple components in ESP-IDF will be modified such as esp32/cpu_start.c. For more details regarding the effects of running ESP-IDF FreeRTOS on a single core, search for occurences of `CONFIG_FREERTOS_UNICORE` in the ESP-IDF components.

CONFIG_FREERTOS_THREAD_LOCAL_STORAGE_POINTERS will define the number of Thread Local Storage Pointers each task will have in ESP-IDF FreeRTOS.

CONFIG_FREERTOS_SUPPORT_STATIC_ALLOCATION will enable the backported functionality of *xTaskCreateStaticPinnedToCore()* in ESP-IDF FreeRTOS

CONFIG_FREERTOS_ASSERT_ON_UNTESTED_FUNCTION will trigger a halt in particular functions in ESP-IDF FreeRTOS which have not been fully tested in an SMP context.

CONFIG_FREERTOS_TASK_FUNCTION_WRAPPER will enclose all task functions within a wrapper function. In the case that a task function mistakenly returns (i.e. does not call *vTaskDelete()*), the call flow will return to the wrapper function. The wrapper function will then log an error and abort the application, as illustrated below:

```
E (25) FreeRTOS: FreeRTOS task should not return. Aborting now!
abort() was called at PC 0x40085c53 on core 0
```

5.11 Thread Local Storage

5.11.1 Overview

Thread-local storage (TLS) is a mechanism by which variables are allocated such that there is one instance of the variable per extant thread. ESP-IDF provides three ways to make use of such variables:

- *FreeRTOS Native API*: ESP-IDF FreeRTOS native API.
- *Pthread API*: ESP-IDF’ s pthread API.
- *C11 Standard*: C11 standard introduces special keyword to declare variables as thread local.

5.11.2 FreeRTOS Native API

The ESP-IDF FreeRTOS provides the following API to manage thread local variables:

- *vTaskSetThreadLocalStoragePointer()*

- *pvTaskGetThreadLocalStoragePointer()*
- *vTaskSetThreadLocalStoragePointerAndDelCallback()*

In this case maximum number of variables that can be allocated is limited by `configNUM_THREAD_LOCAL_STORAGE_POINTERS` macro. Variables are kept in the task control block (TCB) and accessed by their index. Note that index 0 is reserved for ESP-IDF internal uses. Using that API user can allocate thread local variables of an arbitrary size and assign them to any number of tasks. Different tasks can have different sets of TLS variables. If size of the variable is more then 4 bytes then user is responsible for allocating/deallocating memory for it. Variable' s deallocation is initiated by FreeRTOS when task is deleted, but user must provide function (callback) to do proper cleanup.

5.11.3 Pthread API

The ESP-IDF provides the following pthread API to manage thtread local variables:

- `pthread_key_create()`
- `pthread_key_delete()`
- `pthread_getspecific()`
- `pthread_setspecific()`

This API has all benefits of the one above, but eliminates some its limits. The number of variables is limited only by size of available memory on the heap. Due to the dynamic nature this API introduces additional performance overhead compared to the native one.

5.11.4 C11 Standard

The ESP-IDF FreeRTOS supports thread local variables according to C11 standard (ones specified with `__thread` keyword). For details on this GCC feature please see https://gcc.gnu.org/onlinedocs/gcc-5.5.0/gcc/Thread-Local.html#Thread-Local. Storage for that kind of variables is allocated on the task' s stack. Note that area for all such variables in the program will be allocated on the stack of every task in the system even if that task does not use such variables at all. For example ESP-IDF system tasks (like `ipc`, `timer` tasks etc.) will also have that extra stack space allocated. So this feature should be used with care. There is a tradeoff: C11 thread local variables are quite handy to use in programming and can be accessed using just a few Xtensa instructions, but this benefit goes with the cost of additional stack usage for all tasks in the system. Due to static nature of variables allocation all tasks in the system have the same sets of C11 thread local variables.

5.12 High-Level Interrupts

The Xtensa architecture has support for 32 interrupts, divided over 8 levels, plus an assortment of exceptions. On the ESP32, the interrupt mux allows most interrupt sources to be routed to these interrupts using the

interrupt allocator. Normally, interrupts will be written in C, but ESP-IDF allows high-level interrupts to be written in assembly as well, allowing for very low interrupt latencies.

5.12.1 Interrupt Levels

Level	Symbol	Remark
1	N/A	Exception and level 0 interrupts. Handled by ESP-IDF
2-3	N/A	Medium level interrupts. Handled by ESP-IDF
4	xt_highint4	Normally used by ESP-IDF debug logic
5	xt_highint5	Free to use
NMI	xt_nmi	Free to use
dbg	xt_debugexception	Debug exception. Called on e.g. a BREAK instruction.

Using these symbols is done by creating an assembly file (suffix .S) and defining the named symbols, like this:

```
    .section .iram1,"ax"
    .global     xt_highint5
    .type       xt_highint5,@function
    .align      4
xt_highint5:
    ... your code here
    rsr     a0, EXCSAVE_5
    rfi     5
```

For a real-life example, see the esp32/dport_panic_highint_hdl.S file; the panic handler interrupt is implemented there.

5.12.2 Notes

- Do not call C code from a high-level interrupt; because these interupts still run in critical sections, this can cause crashes. (The panic handler interrupt does call normal C code, but this is OK because there is no intention of returning to the normal code flow afterwards.)
- Make sure your assembly code gets linked in. If the interrupt handler symbol is the only symbol the rest of the code uses from this file, the linker will take the default ISR instead and not link the assembly file into the final project. To get around this, in the assembly file, define a symbol, like this:

```
    .global ld_include_my_isr_file
ld_include_my_isr_file:
```

(The symbol is called `ld_include_my_isr_file` here but can have any arbitrary name not defined anywhere else.) Then, in the component.mk, add this file as an unresolved symbol to the ld command line arguments:

```
COMPONENT_ADD_LDFLAGS := -u ld_include_my_isr_file
```

This should cause the linker to always include a file defining `ld_include_my_isr_file`, causing the ISR to always be linked in.

- High-level interrupts can be routed and handled using esp_intr_alloc and associated functions. The handler and handler arguments to esp_intr_alloc must be NULL, however.
- In theory, medium priority interrupts could also be handled in this way. For now, ESP-IDF does not support this.

5.13 JTAG 调试

[English]

本文将指导安装 ESP32 的 OpenOCD 调试环境，并介绍如何使用 GDB 来调试 ESP32 的应用程序。本文的组织结构如下：

引言 介绍本指南主旨。

工作原理 介绍 ESP32，JTAG（Joint Test Action Group）接口，OpenOCD 和 GDB 是如何相互连接从而实现 ESP32 的调试功能。

选择 *JTAG* 适配器 介绍有关 JTAG 硬件适配器的选择及参照标准。

安装 *OpenOCD* 介绍如何安装官方预编译好的 OpenOCD 软件包并验证是否安装成功。

配置 *ESP32* 目标板 介绍如何设置 OpenOCD 软件并安装 JTAG 硬件适配器，这两者共同组成最终的调试目标。

启动调试器 介绍如何从*Eclipse 集成开发环境* 和*命令行终端* 启动 GDB 调试会话。

调试范例 如果你对 GDB 不太熟悉，本小节会分别针对*Eclipse 集成开发环境* 和*命令行终端* 来讲解调试的范例。

从源码构建 *OpenOCD* 介绍如何在*Windows*，*Linux* 和*MacOS* 操作系统上从源码构建 OpenOCD。

注意事项和补充内容 介绍使用 OpenOCD 和 GDB 通过 JTAG 接口调试 ESP32 时的注意事项和补充内容。

5.13.1 引言

ESP32 具有两个强大的 Xtensa 内核，支持多种程序架构。ESP-IDF 自带的 FreeRTOS 操作系统具有多核抢占式多线程的功能，它允许用户以更加直观的方式编写软件。

与此相对地，简便的编程方式会给程序的调试带来困难（如果没有合适的工具），比如找出由两个线程引起的错误，并且这两个线程在单独的 CPU 核上同时运行，仅凭 `printf` 语句会花费很长的时间来定位到该错误。在大多数情况下，调试此类问题更快的方法是使用调试器，连接到处理器的调试端口。

乐鑫已经为 ESP32 处理器和多核 FreeRTOS 架构移植好了 OpenOCD，它将成为大多数 ESP32 应用程序的基础。此外，乐鑫还提供了一些 OpenOCD 本身并不支持的工具来进一步丰富调试的功能。

本文将指导如何在 Linux，Windows 和 MacOS 环境下为 ESP32 安装 OpenOCD，并使用 GDB 进行软件调试。除了个别操作系统的安装过程有所差别以外，软件用户界面和使用流程都是一样的。

注解： 本文使用的图片素材来自于 Ubuntu 16.04 LTE 上 Eclipse Neon 3 软件的截图，不同的操作系统（Windows，MacOS 或者 Linux）和 Eclipse 软件版本在用户界面上可能会有细微的差别。

5.13.2 工作原理

通过 JTAG（Joint Test Action Group）接口使用 OpenOCD 调试 ESP32 时所需要的一些关键的软件和硬件包括 **xtensa-esp32-elf-gdb 调试器**，**OpenOCD 片上调试器**和连接到 **ESP32** 目标的 **JTAG 适配器**。

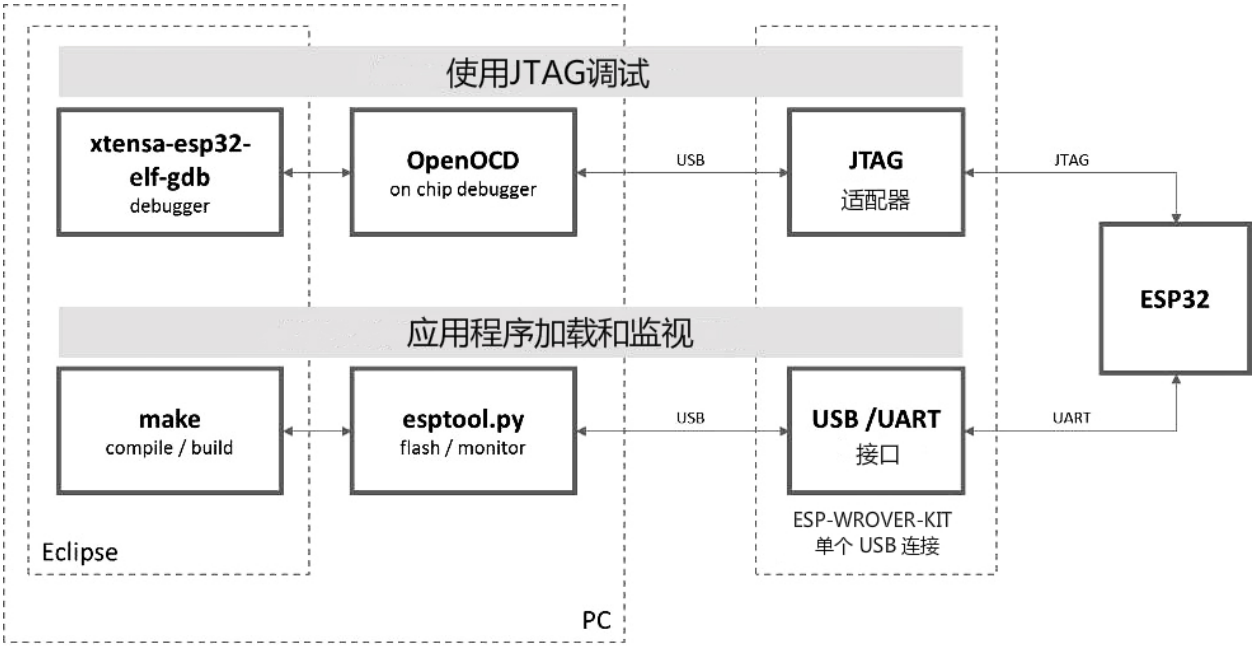


图 5: JTAG 调试 - 概述图

在“Application Loading and Monitoring”下还有另外一组软件和硬件，它们用来编译、构建和烧写应用程序到 ESP32 上，以及监视来自 ESP32 的运行诊断信息。

Eclipse 环境集成了 JTAG 调试和应用程序加载、监视的功能，它使得软件从编写、编译、加载到调试的迭代过程变得更加快速而简单。所有的软件均适用于 Windows，Linux 和 MacOS 平台。

如果你使用的是*ESP-WROVER-KIT 开发板*，得益于板载的 FT232H 芯片，PC 和 ESP32 的连接仅仅需要一根 USB 线即可完成。FT232H 提供了两路 USB 通道，一路连接到 JTAG，另一路连接到 UART。

根据用户的喜好，除了使用 Eclipse 集成开发环境，还可以直接在命令行终端运行 *debugger* 和 *idf.py build*。

5.13.3 选择 JTAG 适配器

上手 JTAG 最快速便捷的方式是使用*ESP-WROVER-KIT 开发板*，因为它板载了 JTAG 调试接口，无需使用外部的 JTAG 硬件适配器和额外的线缆来连接 JTAG 与 ESP32。ESP-WROVER-KIT 采用 FT2232H 提供的 JTAG 接口，可以稳定运行在 20 MHz 的时钟频率，外接的适配器很难达到这个速度。

如果你想使用单独的 JTAG 适配器，请确保其与 ESP32 的电平电压和 OpenOCD 软件都兼容。ESP32 使用的是业界标准的 JTAG 接口，它省略了（实际上也并不需要）TRST 信号脚。JTAG 使用的 IO 引脚由 VDD_3P3_RTC 电源引脚供电（通常连接到外部 3.3 V 的电源轨），因此 JTAG 硬件适配器的引脚需要能够在该电压范围内正常工作。

在软件方面，OpenOCD 支持相当多数量的 JTAG 适配器，可以参阅 OpenOCD 支持的适配器列表 （尽管上面显示的器件不太完整），这个页面还列出了兼容 SWD 接口的适配器，但是请注意，ESP32 目前并不支持 SWD。此外那些被硬编码为只支持特定产品线的 JTAG 适配器也不能在 ESP32 上工作，比如用于 STM32 产品家族的 ST-LINK 适配器。

JTAG 正常工作至少需要连接的信号线有：TDI，TDO，TCK，TMS 和 GND。某些 JTAG 适配器还需要 ESP32 提供一路电源到适配器的某个引脚上（比如 Vtar）用以设置适配器的工作电压。SRST 信号线是可选的，它可以连接到 ESP32 的 CH_PD 引脚上，尽管目前 OpenOCD 对该信号线的支持还非常有限。

5.13.4 安装 OpenOCD

如果你已经按照快速入门 一文中的介绍安装好了 ESP-IDF 及其 CMake 构建系统，那么 OpenOCD 已经被默认安装到了你的开发系统中。在设置开发环境 结束后，你应该能够在终端中运行如下 OpenOCD 命令:

```
openocd --version
```

终端会输出以下信息（实际版本号可能会比这里列出的更新）：

```
Open On-Chip Debugger  v0.10.0-esp32-20190708 (2019-07-08-11:04)
Licensed under GNU GPL v2
For bug reports, read
    http://openocd.org/doc/doxygen/bugs.html
```

你还可以检查 `OPENOCD_SCRIPTS` 环境变量的值来确认 OpenOCD 配置文件的路径，Linux 和 macOS 用户可以在终端输入 `echo $OPENOCD_SCRIPTS`，Windows 用户需要输入 `echo %OPENOCD_SCRIPTS%`。如果终端打印了有效的路径，则表明 OpenOCD 已经被正确安装。

如果上述步骤没有成功执行，请返回快速入门手册，参考其中设置安装工具 章节的说明。

注解: 另外，我们还可以从源代码编译 OpenOCD 工具，相关详细信息请参阅*从源码构建 OpenOCD* 章节。

5.13.5 配置 ESP32 目标板

安装好 OpenOCD 之后就可以配置 ESP32 目标（即带 JTAG 接口的 ESP32 板），具体可以通过以下三个步骤进行：

- 配置并连接 JTAG 接口
- 运行 OpenOCD
- 上传待调试的应用程序

配置并连接 JTAG 接口

此步骤取决于您使用的 JTAG 和 ESP32 板，请参考以下两种情况。

配置 WROVER 上的 JTAG 接口

[English]

所有版本的 ESP-WROVER-KIT 板子都内置了 JTAG 调试功能，要使其正常工作，还需要设置相关跳帽来启用 JTAG 功能，设置 SPI 闪存电压和配置 USB 驱动程序。具体步骤请参考以下说明。

配置硬件

1. 根据*ESP-WROVER-KIT V4.1 入门指南* 文档中设置选项 章节所描述的信息，设置 JP8 便可以启用 JTAG 功能。
2. 检查 ESP32 上用于 JTAG 通信的引脚是否被接到了其它硬件上，这可能会影响 JTAG 的工作。

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

配置 USB 驱动

安装和配置 USB 驱动，这样 OpenOCD 才能够与 ESP-WROVER-KIT 板上的 JTAG 接口通信，并且使用 UART 接口上传待烧写的镜像文件。请根据你的操作系统按照以下步骤进行安装配置。

注解： ESP-WROVER-KIT 使用了 FT2232 芯片实现了 JTAG 适配器，所以以下说明同样适用于其他基于 FT2232 的 JTAG 适配器。

Windows

1. 使用标准 USB A / micro USB B 线将 ESP-WROVER-KIT 与计算机相连接，并打开板子的电源。
2. 等待 Windows 识别出 ESP-WROVER-KIT 并且为其安装驱动。如果驱动没有被自动安装，请前往 官网 下载并手动安装。
3. 从 Zadig 官网 下载 Zadig 工具（Zadig_X.X.exe）并运行。
4. 在 Zadig 工具中，进入 “Options” 菜单中选中 “List All Devices”。
5. 检查设备列表，其中应该包含两条与 ESP-WROVER-KIT 相关的条目：“Dual RS232-HS (Interface 0)” 和 “Dual RS232-HS (Interface 1)”。驱动的名字应该是 “FTDIBUS (vxxxx)” 并且 USB ID 为：0403 6010。

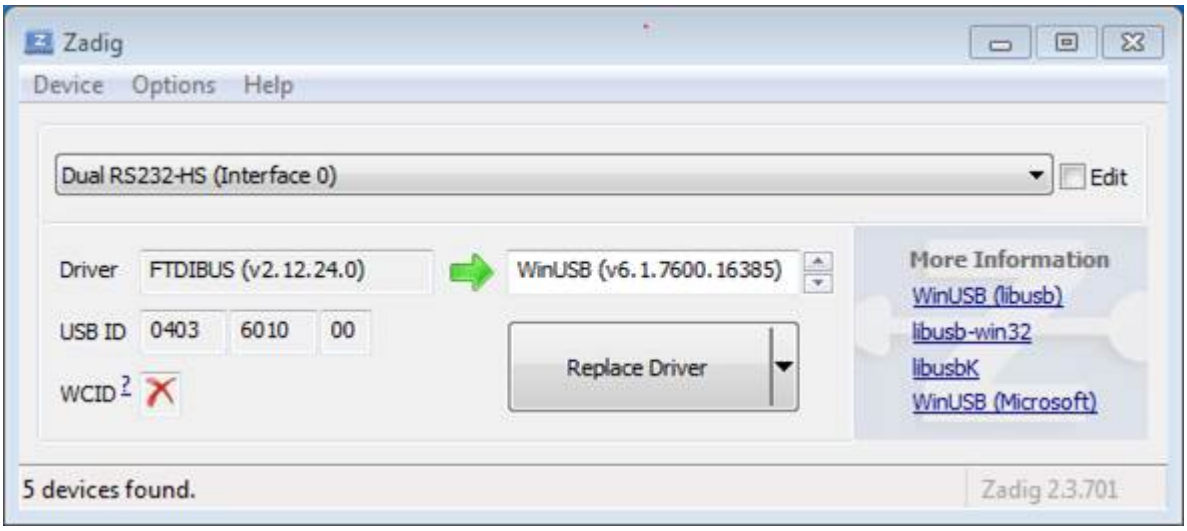


图 6: 在 Zadig 工具中配置 JTAG USB 驱动

6. 第一个设备 “Dual RS232-HS (Interface 0)” 连接到了 ESP32 的 JTAG 端口，此设备原来的 “FTDIBUS (vxxxx)” 驱动需要替换成 “WinUSB (v6xxxxx)”。为此，请选择 “Dual RS232-HS (Interface 0)” 并将驱动重新安装为 “WinUSB (v6xxxxx)”，具体可以参考上图。

注解： 请勿更改第二个设备 “Dual RS232-HS（Interface 1）” 的驱动，它被连接到 ESP32 的串口（UART），用于上传应用程序映像给 ESP32 进行烧写。

现在，ESP-WROVER-KIT 的 JTAG 接口应该可以被 OpenOCD 使用了，想要进一步设置调试环境，请前往*运行 OpenOCD* 章节。

Linux

1. 使用标准 USB A / micro USB B 线将 ESP-WROVER-KIT 与计算机相连接，并打开板子的电源。
2. 打开终端，输入 `ls -l /dev/ttyUSB*` 命令检查操作系统是否能够识别板子的 USB 端口。类似识别结果如下：

```
user-name@computer-name:~/esp$ ls -l /dev/ttyUSB*
crw-rw---- 1 root dialout 188, 0 Jul 10 19:04 /dev/ttyUSB0
crw-rw---- 1 root dialout 188, 1 Jul 10 19:04 /dev/ttyUSB1
```

3. 根据 OpenOCD README 文档 中 “Permissions delegation” 小节的介绍，设置这两个 USB 端口的访问权限。
4. 注销并重新登录 Linux 系统，然后重新插拔板子的电源使之前的改动生效。在终端再次输入 `ls -l /dev/ttyUSB*` 命令进行验证，查看这两个设备的组所有者是否已经从 `dialout` 更改为 `plugdev`:

```
user-name@computer-name:~/esp$ ls -l /dev/ttyUSB*
crw-rw-r-- 1 root plugdev 188, 0 Jul 10 19:07 /dev/ttyUSB0
crw-rw-r-- 1 root plugdev 188, 1 Jul 10 19:07 /dev/ttyUSB1
```

   如果看到类似的输出结果，并且你也是 `plugdev` 组的成员，那么设置工作就完成了。

   具有较低编号的 `/dev/ttyUSBn` 接口用于 JTAG 通信，另一路接口被连接到 ESP32 的串口（UART），用于上传应用程序映像给 ESP32 进行烧写。

现在，ESP-WROVER-KIT 的 JTAG 接口应该可以被 OpenOCD 使用了，想要进一步设置调试环境，请前往*运行 OpenOCD* 章节。

MacOS

在 macOS 上，同时使用 FT2232 的 JTAG 接口和串口还需另外进行其它操作。当操作系统加载 FTDI 串口驱动的时候，它会对 FT2232 芯片的两个通道做相同的操作。但是，这两个通道中只有一个是被用作串口，而另一个用于 JTAG，如果操作系统已经为用于 JTAG 的通道加载了 FTDI 串口驱动的话，OpenOCD 将无法连接到芯片。有两个方法可以解决这个问题：

1. 在启动 OpenOCD 之前手动卸载 FTDI 串口驱动程序，然后启动 OpenOCD，再加载串口驱动程序。

2. 修改 FTDI 驱动程序的配置，使其不会为 FT2232 芯片的通道 B 进行自我加载，该通道用于 ESP-WROVER-KIT 板上的 JTAG 通道。

手动卸载驱动程序

1. 从 FTDI 官网 安装驱动。
2. 使用 USB 线连接 ESP-WROVER-KIT。
3. 卸载串口驱动

```
sudo kextunload -b com.FTDI.driver.FTDIUSBSerialDriver
```

有时，您可能还需要卸载苹果的 FTDI 驱动:

```
sudo kextunload -b com.apple.driver.AppleUSBFTDI
```

4. 运行 OpenOCD:

```
openocd -f board/esp32-wrover-kit-3.3v.cfg
```

5. 在另一个终端窗口，再一次加载 FTDI 串口驱动:

```
sudo kextload -b com.FTDI.driver.FTDIUSBSerialDriver
```

注解： 如果你需要重启 OpenOCD，则无需再次卸载 FTDI 驱动程序，只需停止 OpenOCD 并再次启动它。只有在重新连接 ESP-WROVER-KIT 或者切换了电源的情况下才需要再次卸载驱动。

你也可以根据自身需求，将此过程包装进 shell 脚本中。

修改 FTDI 驱动

简而言之，这种方法需要修改 FTDI 驱动程序的配置文件，这样可以防止为 FT2232H 的通道 B 自动加载串口驱动。

注解： 其他板子可能将通道 A 用于 JTAG，因此请谨慎使用此选项。

警告： 此方法还需要操作系统禁止对驱动进行签名验证，因此可能无法被所有的用户所接受。

1. 使用文本编辑器打开 FTDI 驱动的配置文件（注意 `sudo`）：

```
sudo nano /Library/Extensions/FTDIUSBSerialDriver.kext/Contents/Info.plist
```

2. 找到并删除以下几行:

```
<key>FT2232H_B</key>
<dict>
    <key>CFBundleIdentifier</key>
    <string>com.FTDI.driver.FTDIUSBSerialDriver</string>
    <key>IOClass</key>
    <string>FTDIUSBSerialDriver</string>
    <key>IOProviderClass</key>
    <string>IOUSBInterface</string>
    <key>bConfigurationValue</key>
    <integer>1</integer>
    <key>bInterfaceNumber</key>
    <integer>1</integer>
    <key>bcdDevice</key>
    <integer>1792</integer>
    <key>idProduct</key>
    <integer>24592</integer>
    <key>idVendor</key>
    <integer>1027</integer>
</dict>
```

3. 保存并关闭文件
4. 禁用驱动的签名认证:
    1. 点击苹果的 logo，选择“Restart…”
    2. 重启后当听到响铃时，立即按下键盘上的 CMD+R 组合键
    3. 进入恢复模式后，打开终端
    4. 运行命令:

    ```
    csrutil enable --without kext
    ```

    5. 再一次重启系统

完成这些步骤后，可以同时使用串口和 JTAG 接口了。

想要进一步设置调试环境，请前往运行 *OpenOCD* 章节。

配置其它 JTAG 接口

[English]

关于适配 OpenOCD 和 ESP32 的 JTAG 接口选择问题，请参考选择 *JTAG* 适配器 章节，确保 JTAG 适配器能够与 OpenOCD 和 ESP32 一同工作。然后按照以下三个步骤进行设置，使其正常工作。

配置硬件

1. 找到 JTAG 接口和 ESP32 板上需要相互连接并建立通信的引脚/信号。

	ESP32 引脚	JTAG 信号
1	CHIP_PU	TRST_N
2	MTDO / GPIO15	TDO
3	MTDI / GPIO12	TDI
4	MTCK / GPIO13	TCK
5	MTMS / GPIO14	TMS
6	GND	GND

2. 检查 ESP32 上用于 JTAG 通信的的引脚是否被连接到了其它硬件上，这可能会影响 JTAG 的工作。
3. 连接 ESP32 和 JTAG 接口上的引脚/信号。

配置驱动

你可能还需要安装软件驱动，才能使 JTAG 在计算机上正常工作，请参阅你所使用的 JTAG 适配器的有关文档，获取相关详细信息。

连接

将 JTAG 接口连接到计算机，打开 ESP32 和 JTAG 接口板上的电源，然后检查计算机是否可以识别到 JTAG 接口。

要继续设置调试环境，请前往运行 *OpenOCD* 章节。

运行 OpenOCD

配置完目标并将其连接到电脑后，即可启动 OpenOCD。

打开终端，按照快速入门中的指南设置好开发环境，然后运行如下命令，启动 OpenOCD（该命令在 Windows，Linux，和 macOS 中通用）：

```
openocd -f board/esp32-wrover-kit-3.3v.cfg
```

注解: 上述命令中 `-f` 选项后跟的配置文件专用于板载*ESP-WROOM-32* 模组的 ESP-WROVER-KIT 开发板。您可能需要根据具体使用的硬件而选择或修改不同的配置文件，相关指导请参阅*针对特定目标的 OpenOCD 配置*。

现在应该可以看到如下输入（此日志来自 ESP-WROVER-KIT）：

```
user-name@computer-name:~/esp/esp-idf$ openocd -f board/esp32-wrover-kit-3.3v.cfg
Open On-Chip Debugger  v0.10.0-esp32-20190708 (2019-07-08-11:04)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
none separate
adapter speed: 20000 kHz
force hard breakpoints
Info : ftdi: if you experience problems at higher adapter clocks, try the command "ftdi_
↪tdo_sample_edge falling"
Info : clock speed 20000 kHz
Info : JTAG tap: esp32.cpu0 tap/device found: 0x120034e5 (mfg: 0x272 (Tensilica), part:␣
↪0x2003, ver: 0x1)
Info : JTAG tap: esp32.cpu1 tap/device found: 0x120034e5 (mfg: 0x272 (Tensilica), part:␣
↪0x2003, ver: 0x1)
Info : esp32: Debug controller was reset (pwrstat=0x5F, after clear 0x0F).
Info : esp32: Core was reset (pwrstat=0x5F, after clear 0x0F).
```

- 如果出现指示权限问题的错误，请参阅 `~/esp/openocd-esp32` 目录下 OpenOCD README 文件中关于 “Permissions delegation” 的说明。
- 如果发现配置文件有错误，例如 `Can't find board/esp32-wrover-kit-3.3v.cfg`，请检查 `-s` 后面的路径，OpenOCD 会根据此路径来查找 `-f` 指定的文件。此外，还需要检查配置文件是否确实位于该路径下。
- 如果看到 JTAG 错误（输出全是 1 或者全是 0），请检查硬件连接，除了 ESP32 的引脚之外是否还有其他信号连接到了 JTAG，并查看是否所有器件都已经上电。

上传待调试的应用程序

您可以像往常一样构建并上传 ESP32 应用程序，具体请参阅*第八步：编译工程* 章节。

除此以外，还支持使用 OpenOCD 通过 JTAG 接口将应用程序镜像烧写到闪存中，命令如下:

```
openocd -f board/esp32-wrover-kit-3.3v.cfg -c "program_esp filename.bin 0x10000 verify␣
↪exit"
```

其中 OpenOCD 的烧写命令 `program_esp` 具有以下格式：

`program_esp <image_file> <offset> [verify] [reset] [exit]`

- `image_file` - 程序镜像文件存放的路径
- `offset` - 镜像烧写到闪存中的偏移地址
- `verify` - 烧写完成后校验闪存中的内容（可选）
- `reset` - 烧写完成后重启目标（可选）
- `exit` - 烧写完成后退出 OpenOCD（可选）

现在可以进行应用程序的调试了，请按照以下章节中讲解的步骤进行操作。

5.13.6 启动调试器

ESP32 的工具链中带有 GNU 调试器（简称 GDB）`xtensa-esp32-elf-gdb`，它和其它工具链软件存放在同一个 bin 目录下。除了直接在命令行终端中调用并操作 GDB 外，还可以在 IDE (例如 Eclipse, Visual Studio Code 等）中调用它，在图形用户界面的帮助下间接操作 GDB，无需在终端中输入任何命令。

关于以上两种调试器的使用方法，详见以下链接。

- *在 Eclipse 中使用 GDB*
- *在命令行中使用 GDB*

建议首先检查调试器是否能在命令行终端 下正常工作，然后再转到使用 Eclipse 等集成开发环境 下进行调试工作。

5.13.7 调试范例

本节适用于不熟悉 GDB 的用户，将使用 get-started/blink 下简单的应用程序来演示调试会话的工作流程，同时会介绍以下常用的调试操作：

1. 浏览代码，查看堆栈和线程
2. 设置和清除断点
3. 手动暂停目标
4. 单步执行代码
5. 查看并设置内存
6. 观察和设置程序变量
7. 设置条件断点

此外还会提供在命令行终端进行调试 的案例。

在演示之前，请设置好 ESP32 目标板并加载 get-started/blink 至 ESP32 中。

5.13.8 从源码构建 OpenOCD

请参阅以下文档，它们分别介绍了在各大操作系统平台上从源码构建 OpenOCD 的流程。

Windows 环境下从源码编译 OpenOCD

[English]

除了从 Espressif 官方 直接下载 OpenOCD 可执行文件，你还可以选择从源码编译得到 OpenOCD。如果想要快速设置 OpenOCD 而不是自行编译，请备份好当前文件，前往安装 *OpenOCD* 章节查阅。

下载 OpenOCD 源码

支持 ESP32 的 OpenOCD 源代码可以从乐鑫官方的 GitHub 获得，网址为 https://github.com/espressif/openocd-esp32。请使用以下命令来下载源代码:

```
cd ~/esp
git clone --recursive https://github.com/espressif/openocd-esp32.git
```

克隆后的源代码被保存在 `~/esp/openocd-esp32` 目录中。

安装依赖的软件包

安装编译 OpenOCD 所需的软件包。

注解: 依次安装以下软件包，检查安装是否成功，然后继续下一个软件包的安装。在进行下一步操作之前，要先解决当前报告的问题。

```
pacman -S libtool
pacman -S autoconf
pacman -S automake
pacman -S texinfo
pacman -S mingw-w64-i686-libusb-compat-git
pacman -S pkg-config
```

注解: 安装 pkg-config 会破坏 esp-idf 的工具链，因而在 OpenOCD 构建完成后，应将其卸载。详见文末进一步说明。如果想要再次构建 OpenOCD，你需要再次运行 pacman -S pkg-config。此步骤安装的其他软件包（在 pkg-config 之前）并不会出现这一问题。

构建 OpenOCD

配置和构建 OpenOCD 的流程如下:

```
cd ~/esp/openocd-esp32
./bootstrap
./configure
make
```

你可以选择最后再执行 sudo make install ，如果你已经安装过别的开发平台的 OpenOCD，请跳过这个步骤，因为它可能会覆盖掉原来的 OpenOCD。

注解:

- 如果发生错误，请解决后再次尝试编译，直到 make 成功为止。
- 如果 OpenOCD 存在子模块问题，请 cd 到 openocd-esp32 目录，并输入 git submodule update --init 命令。
- 如果 ./configure 成功运行，JTAG 被使能的信息会被打印在 OpenOCD configuration summary 下面。
- 如果您的设备信息未显示在日志中，请根据 ../openocd-esp32/doc/INSTALL.txt 文中的描述使用 ./configure 启用它。
- 有关编译 OpenOCD 的详细信息，请参阅 openocd-esp32/README.Windows。

一旦 make 过程成功完成，OpenOCD 的可执行文件会被保存到 ~/esp/openocd-esp32/src/openocd 目录中。

如安装依赖步骤所述，最后还需要移除 pkg-config 软件包:

```
pacman -Rs pkg-config
```

下一步

想要进一步配置调试环境，请前往*配置 ESP32 目标板* 章节。

Linux 环境下从源码编译 OpenOCD

[English]

除了从 Espressif 官方 直接下载 OpenOCD 可执行文件，你还可以选择从源码编译得到 OpenOCD。如果想要快速设置 OpenOCD 而不是自行编译，请备份好当前文件，前往*安装 OpenOCD* 章节查阅。

下载 OpenOCD 源码

支持 ESP32 的 OpenOCD 源代码可以从乐鑫官方的 GitHub 获得，网址为 https://github.com/espressif/openocd-esp32。请使用以下命令来下载源代码:

```
cd ~/esp
git clone --recursive https://github.com/espressif/openocd-esp32.git
```

克隆后的源代码被保存在 `~/esp/openocd-esp32` 目录中。

安装依赖的软件包

安装编译 OpenOCD 所需的软件包。

注解: 依次安装以下软件包，检查安装是否成功，然后继续下一个软件包的安装。在进行下一步操作之前，要先解决当前报告的问题。

```
sudo apt-get install make
sudo apt-get install libtool
sudo apt-get install pkg-config
sudo apt-get install autoconf
sudo apt-get install automake
sudo apt-get install texinfo
sudo apt-get install libusb-1.0
```

注解:

- pkg-config 应为 0.2.3 或以上的版本。
- autoconf 应为 2.6.4 或以上的版本。
- automake 应为 1.9 或以上的版本。
- 当使用 USB-Blaster，ASIX Presto，OpenJTAG 和 FT2232 作为适配器时，需要下载安装 libFTDI 和 FTD2XX 的驱动。

- 当使用 CMSIS-DAP 时，需要安装 HIDAPI。

构建 OpenOCD

配置和构建 OpenOCD 的流程如下:

```
cd ~/esp/openocd-esp32
./bootstrap
./configure
make
```

你可以选择最后再执行 `sudo make install` ，如果你已经安装过别的开发平台的 OpenOCD，请跳过这个步骤，因为它可能会覆盖掉原来的 OpenOCD。

注解:

- 如果发生错误，请解决后再次尝试编译，直到 `make` 成功为止。
- 如果 OpenOCD 存在子模块问题，请 `cd` 到 `openocd-esp32` 目录，并输入 `git submodule update --init` 命令。
- 如果 `./configure` 成功运行，JTAG 被使能的信息会被打印在 `OpenOCD configuration summary` 下面。
- 如果您的设备信息未显示在日志中，请根据 `../openocd-esp32/doc/INSTALL.txt` 文中的描述使用 `./configure` 启用它。
- 有关编译 OpenOCD 的详细信息，请参阅 `openocd-esp32/README`。

一旦 `make` 过程成功结束，OpenOCD 的可执行文件会被保存到 `~/openocd-esp32/bin` 目录中。

下一步

想要进一步配置调试环境，请前往*配置 ESP32 目标板* 章节。

MacOS 环境下从源码编译 OpenOCD

[English]

除了从 Espressif 官方 直接下载 OpenOCD 可执行文件，你还可以选择从源码编译得到 OpenOCD。如果想要快速设置 OpenOCD 而不是自行编译，请备份好当前文件，前往*安装 OpenOCD* 章节查阅。

下载 OpenOCD 源码

支持 ESP32 的 OpenOCD 源代码可以从乐鑫官方的 GitHub 获得，网址为 https://github.com/espressif/openocd-esp32。请使用以下命令来下载源代码:

```
cd ~/esp
git clone --recursive https://github.com/espressif/openocd-esp32.git
```

克隆后的源代码被保存在 ~/esp/openocd-esp32 目录中。

安装依赖的软件包

使用 Homebrew 安装编译 OpenOCD 所需的软件包:

```
brew install automake libtool libusb wget gcc@4.9 pkg-config
```

构建 OpenOCD

配置和构建 OpenOCD 的流程如下:

```
cd ~/esp/openocd-esp32
./bootstrap
./configure
make
```

你可以选择最后再执行 sudo make install ，如果你已经安装过别的开发平台的 OpenOCD，请跳过这个步骤，因为它可能会覆盖掉原来的 OpenOCD。

注解:

- 如果发生错误，请解决后再次尝试编译，直到 make 成功为止。
- 如果 OpenOCD 存在子模块问题，请 cd 到 openocd-esp32 目录，并输入 git submodule update --init 命令。
- 如果 ./configure 成功运行，JTAG 被使能的信息会被打印在 OpenOCD configuration summary 下面。
- 如果您的设备信息未显示在日志中，请根据 ../openocd-esp32/doc/INSTALL.txt 文中的描述使用 ./configure 启用它。
- 有关编译 OpenOCD 的详细信息，请参阅 openocd-esp32/README.OSX。

一旦 make 过程成功结束，OpenOCD 的可执行文件会被保存到 ~/esp/openocd-esp32/src/openocd 目录中。

下一步

想要进一步配置调试环境，请前往配置 *ESP32* 目标板 章节。

本文档演示所使用的 OpenOCD 是安装 *OpenOCD* 章节中介绍的预编译好的二进制发行版。

如果要使用本地从源代码编译的 OpenOCD 程序，需要将相应可执行文件的路径修改为 src/openocd，并设置 OPENOCD_SCRIPTS 环境变量，这样 OpenOCD 才能找到配置文件。Linux 和 macOS 用户可以执行:

```
cd ~/esp/openocd-esp32
export OPENOCD_SCRIPTS=$PWD/tcl
```

Windows 用户可以执行:

```
cd %USERPROFILE%\esp\openocd-esp32
set "OPENOCD_SCRIPTS=%CD%\tcl"
```

运行本地编译的 OpenOCD 的示例如下（Linux 和 macOS 用户）：

```
src/openocd -f board/esp32-wrover-kit-3.3v.cfg
```

Windows 用户:

```
src\openocd -f board\esp32-wrover-kit-3.3v.cfg
```

5.13.9 注意事项和补充内容

本节列出了本指南中提到的所有注意事项和补充内容的链接。

- 可用的断点和观察点
- 关于断点的补充知识
- 闪存映射 *vs* 软件闪存断点
- “*next*” 命令无法跳过子程序的原因
- *OpenOCD* 支持的编译时的选项
- 支持 *FreeRTOS*
- 在 *OpenOCD* 的配置文件中设置 *SPI* 闪存的工作电压
- 优化 *JTAG* 的速度
- 调试器的启动命令的含义

- 针对特定目标的 *OpenOCD* 配置
- 复位 *ESP32*
- 不要将 *JTAG* 引脚用于其他功能
- 报告 *OpenOCD / GDB* 的问题

5.13.10 相关文档

使用调试器

[English]

本节会在*Eclipse* 和命令行 中分别介绍配置和运行调试器的方法。我们建议你首先通过命令行 检查调试器是否正常工作，然后再转到使用*Eclipse* 平台。

在 Eclipse 中使用 GDB

标准的 Eclipse 安装流程默认安装调试功能，另外我们还可以使用插件来调试，比如 “GDB Hardware Debugging”。这个插件用起来非常方便，本指南会详细介绍该插件的使用方法。

首先，通过打开 Eclipse 并转到 “Help” > “Install New Software” 来安装 “GDB Hardware Debugging” 插件。

安装完成后，按照以下步骤配置调试会话。请注意，一些配置参数是通用的，有些则针对特定项目。我们会通过配置 “blink” 示例项目的调试环境来进行展示，请先按照使用 *Eclipse IDE* 编译和烧写 文章介绍的方法将该示例项目添加到 Eclipse 的工作空间。示例项目 get-started/blink 的源代码可以在 ESP-IDF 仓库的 examples 目录下找到。

1. 在 Eclipse 中，进入 *Run* > *Debug Configuration*，会出现一个新的窗口。在窗口的左侧窗格中，双击 “GDB Hardware Debugging”（或者选择 “GDB Hardware Debugging” 然后按下 “New” 按钮）来新建一个配置。
2. 在右边显示的表单中，“Name:” 一栏中输入配置的名称，例如：“Blink checking”。
3. 在下面的 “Main” 选项卡中，点击 “Project:” 边上的 “Browse” 按钮，然后选择当前的 “blink” 项目。
4. 在下一行的 “C/C++ Application:” 中，点击 “Browse” 按钮，选择 “blink.elf” 文件。如果 “blink.elf” 文件不存在，那么很有可能该项目还没有编译，请参考使用 *Eclipse IDE* 编辑和烧写 指南中的介绍。
5. 最后，在 “Build (if required) before launching” 下面点击 “Disable auto build”。

   上述步骤 1 - 5 的示例输入如下图所示。
6. 点击 “Debugger” 选项卡，在 “GDB Command” 栏中输入 `xtensa-esp32-elf-gdb` 来调用调试器。
7. 更改 “Remote host” 的默认配置，在 “Port number” 下面输入 `3333`。

   上述步骤 6 - 7 的示例输入如下图所示。

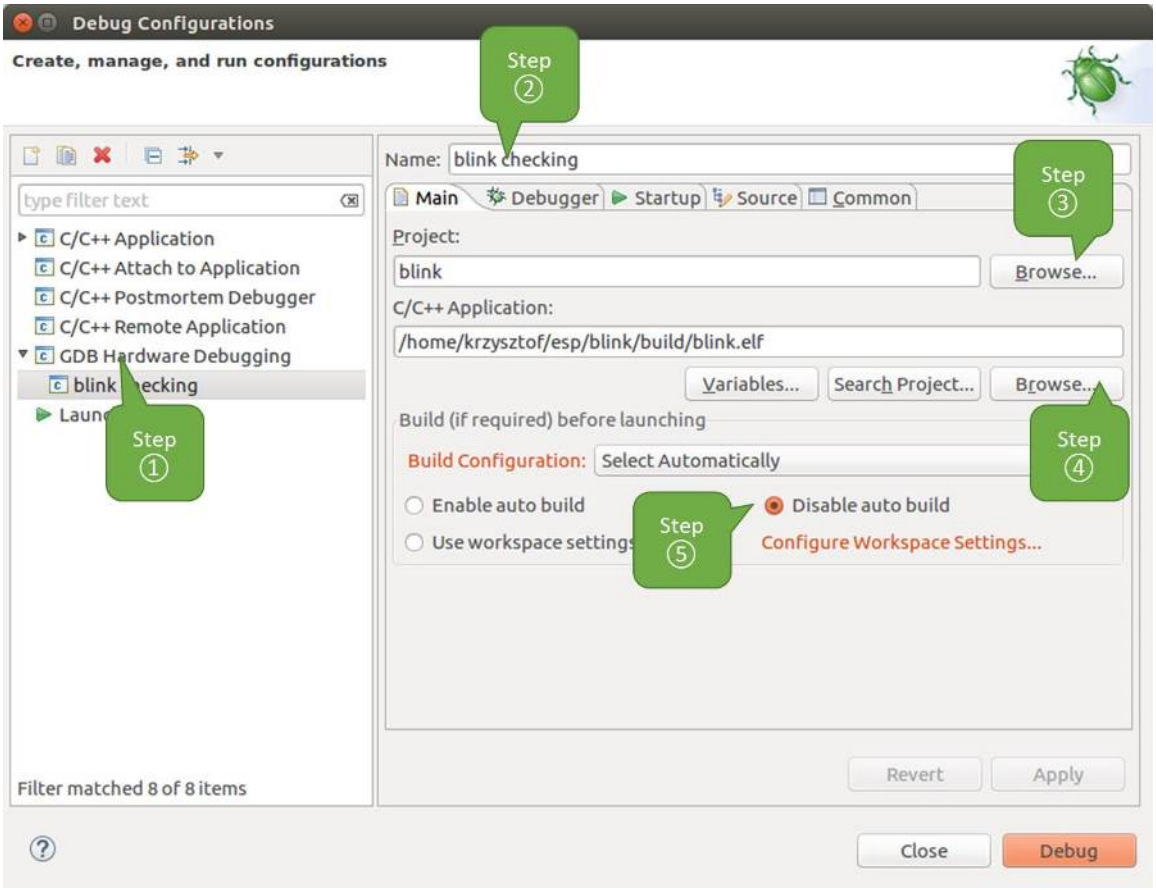


图 7: GDB 硬件调试的配置 - Main 选项卡

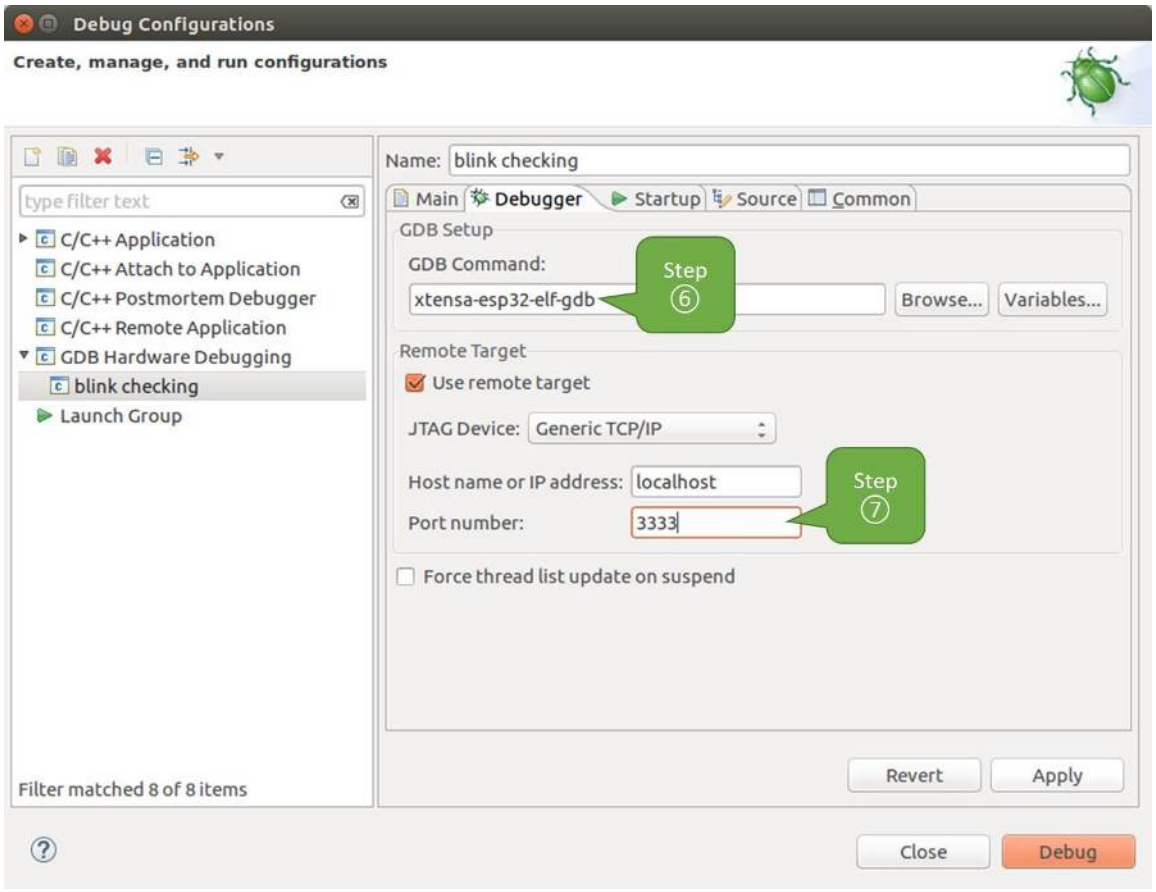


图 8: GDB 硬件调试的配置 - Debugger 选项卡

8. 最后一个需要更改默认配置的选项卡是 “Startup” 选项卡。在 “Initialization Commands” 下，取消选中 “Reset and Delay (seconds)” 和 “Halt”，然后在下面一栏中输入以下命令：

```
mon reset halt
flushregs
set remote hardware-watchpoint-limit 2
```

注解： 如果你想在启动新的调试会话之前自动更新闪存中的镜像，请在 “Initialization Commands” 文本框的开头添加以下命令行:

```
mon reset halt
mon program_esp ${workspace_loc:blink/build/blink.bin} 0x10000 verify
```

有关 `program_esp` 命令的说明请参考*上传待调试的应用程序* 章节。

9. 在 “Load Image and Symbols” 下，取消选中 “Load image” 选项。
10. 在同一个选项卡中继续往下浏览，建立一个初始断点用来在调试器复位后暂停 CPU。插件会根据 “Set break point at:” 一栏中输入的函数名，在该函数的开头设置断点。选中这一选项，并在相应的字段中输入 `app_main`。
11. 选中 “Resume” 选项，这会使得程序在每次调用步骤 8 中的 `mon reset halt` 之后恢复，然后在 `app_main` 的断点处停止。

    上述步骤 8 - 11 的示例输入如下图所示。

    上面的启动序列看起来有些复杂，如果你对其中的初始化命令不太熟悉，请查阅*调试器的启动命令的含义* 章节获取更多说明。
12. 如果你前面已经完成*配置 ESP32 目标板* 中介绍的步骤，那么目标正在运行并准备与调试器进行对话。按下 “Debug” 按钮就可以直接调试。否则请按下 “Apply” 按钮保存配置，返回*配置 ESP32 目标板* 章节进行配置，最后再回到这里开始调试。

一旦所有 1 - 12 的配置步骤都已经完成，Eclipse 就会打开 “Debug” 视图，如下图所示。

如果你不太了解 GDB 的常用方法，请查阅*使用 Eclipse 的调试示例* 文章中的调试示例章节*调试范例*。

在命令行中使用 GDB

1. 为了能够启动调试会话，需要先启动并运行目标，如果还没有完成，请按照*配置 ESP32 目标板* 中的介绍进行操作。
2. 打开一个新的终端会话并前往待调试的项目目录，比如：

```
cd ~/esp/blink
```

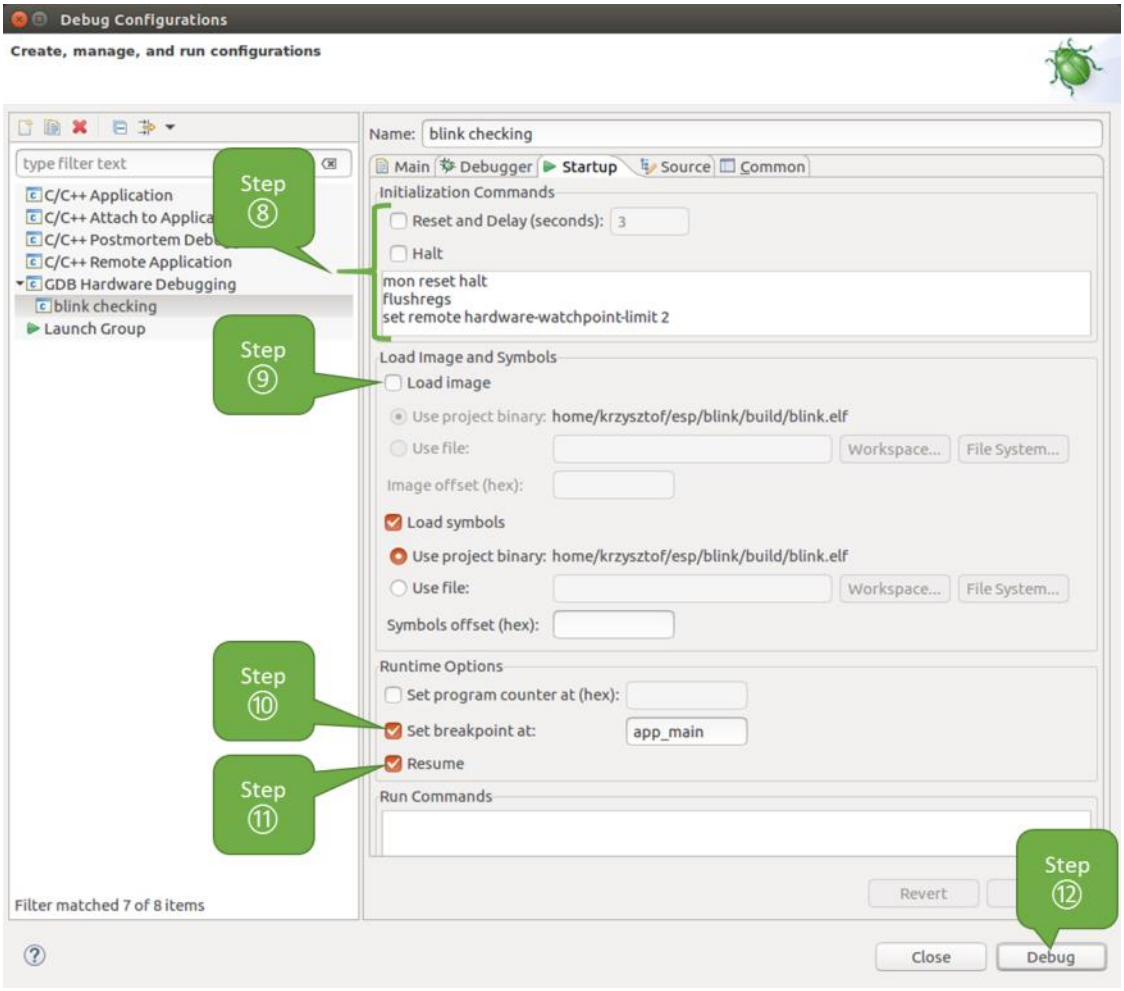


图 9: GDB 硬件调试的配置 - Startup 选项卡

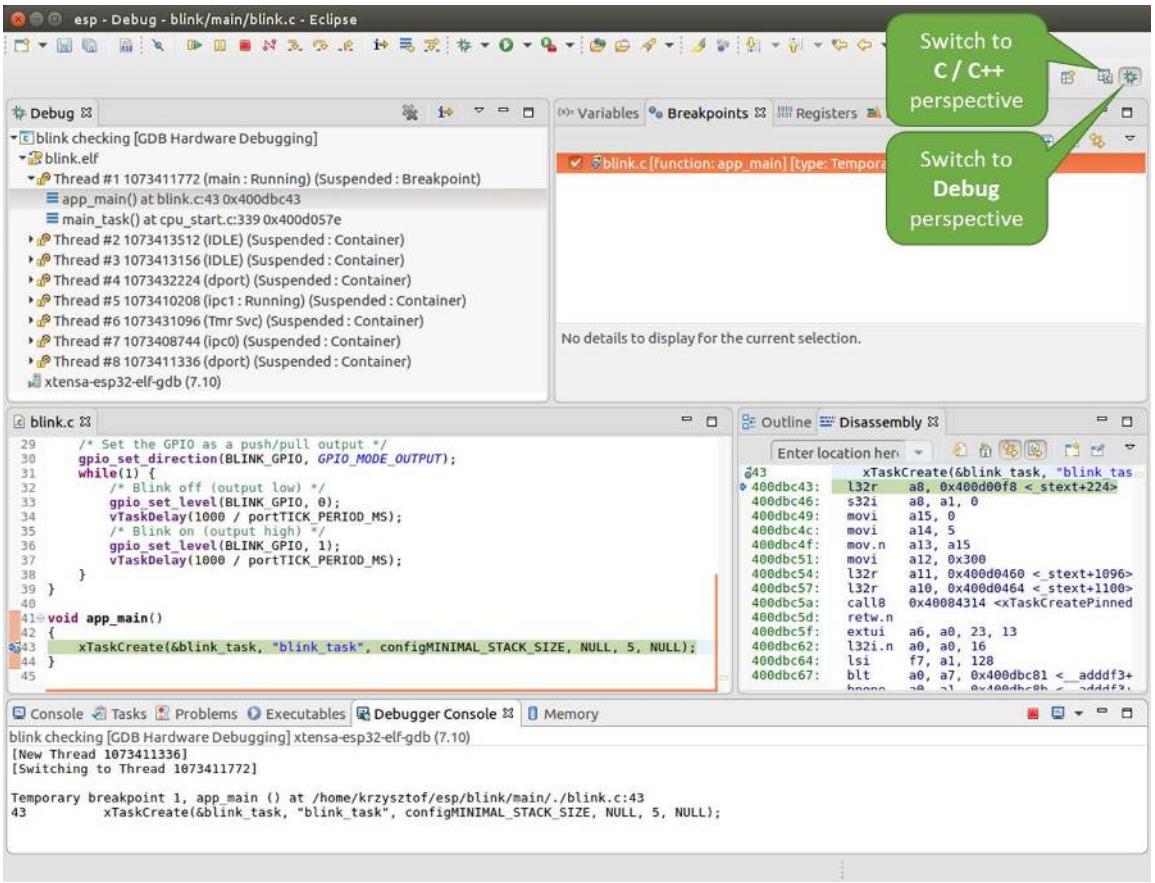


图 10: Eclipse 中的调试视图

3. 当启动调试器时，通常需要提供几个配置参数和命令，为了避免每次都在命令行中逐行输入这些命令，我们可以新建一个配置文件，并将其命名为 gdbinit:

```
target remote :3333
set remote hardware-watchpoint-limit 2
mon reset halt
flushregs
thb app_main
c
```

将此文件保存在当前目录中。

有关 gdbinit 文件内部的更多详细信息，请参阅调试器的启动命令的含义 章节。

4. 准备好启动 GDB，请在终端中输入以下内容:

```
xtensa-esp32-elf-gdb -x gdbinit build/blink.elf
```

5. 如果前面的步骤已经正确完成，你会看到如下所示的输出日志，在日志的最后会出现 (gdb) 提示符:

```
user-name@computer-name:~/esp/blink$ xtensa-esp32-elf-gdb -x gdbinit build/blink.elf
GNU gdb (crosstool-NG crosstool-ng-1.22.0-61-gab8375a) 7.10
Copyright (C) 2015 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=x86_64-build_pc-linux-gnu --target=xtensa-esp32-
↪elf".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from build/blink.elf...done.
0x400d10d8 in esp_vApplicationIdleHook () at /home/user-name/esp/esp-idf/components/
↪esp32/./freertos_hooks.c:52
52          asm("waiti 0");
JTAG tap: esp32.cpu0 tap/device found: 0x120034e5 (mfg: 0x272 (Tensilica), part:␣
↪0x2003, ver: 0x1)
JTAG tap: esp32.slave tap/device found: 0x120034e5 (mfg: 0x272 (Tensilica), part:␣
↪0x2003, ver: 0x1)
```

(下页继续)

(续上页)

```
esp32: Debug controller was reset (pwrstat=0x5F, after clear 0x0F).
esp32: Core was reset (pwrstat=0x5F, after clear 0x0F).
Target halted. PRO_CPU: PC=0x5000004B (active)    APP_CPU: PC=0x00000000
esp32: target state: halted
esp32: Core was reset (pwrstat=0x1F, after clear 0x0F).
Target halted. PRO_CPU: PC=0x40000400 (active)    APP_CPU: PC=0x40000400
esp32: target state: halted
Hardware assisted breakpoint 1 at 0x400db717: file /home/user-name/esp/blink/main/./
↪blink.c, line 43.
0x0:    0x00000000
Target halted. PRO_CPU: PC=0x400DB717 (active)    APP_CPU: PC=0x400D10D8
[New Thread 1073428656]
[New Thread 1073413708]
[New Thread 1073431316]
[New Thread 1073410672]
[New Thread 1073408876]
[New Thread 1073432196]
[New Thread 1073411552]
[Switching to Thread 1073411996]

Temporary breakpoint 1, app_main () at /home/user-name/esp/blink/main/./blink.c:43
43      xTaskCreate(&blink_task, "blink_task", 512, NULL, 5, NULL);
(gdb)
```

注意上面日志的倒数第三行显示了调试器已经在 `app_main()` 函数的断点处停止，该断点在 `gdbinit` 文件中设定。由于处理器已经暂停运行，LED 也不会闪烁。如果这也是你看到的现象，你可以开始调试了。

如果你不太了解 GDB 的常用方法，请查阅使用命令行的调试示例 文章中的调试示例章节调试范例。

调试示例

[English]

本节将介绍如何在*Eclipse* 和命令行 中使用 GDB 进行调试的示例。

使用 Eclipse 的调试示例

请检查目标板是否已经准备好，并加载了 get-started/blink 示例代码，然后按照*在 Eclipse 中使用 GDB* 中介绍的步骤配置和启动调试器，最后选择让应用程序在 `app_main()` 建立的断点处停止。

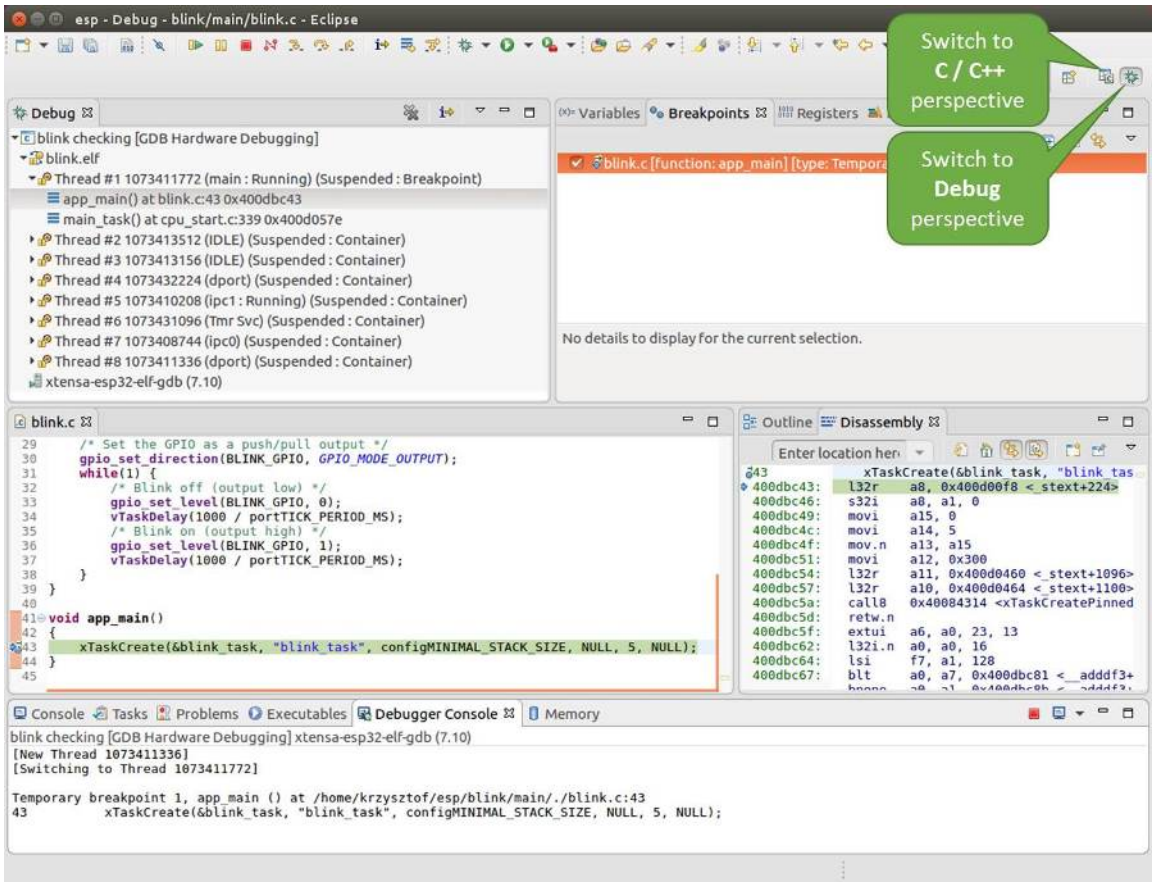


图 11: Eclipse 中的 Debug 视图

本小节的示例

1. 浏览代码，查看堆栈和线程
2. 设置和清除断点
3. 手动暂停目标
4. 单步执行代码
5. 查看并设置内存
6. 观察和设置程序变量
7. 设置条件断点

浏览代码，查看堆栈和线程

当目标暂停时，调试器会在“Debug”窗口中显示线程的列表，程序暂停的代码行在下面的另一个窗口中被高亮显示，如下图所示。此时板子上的 LED 停止了闪烁。

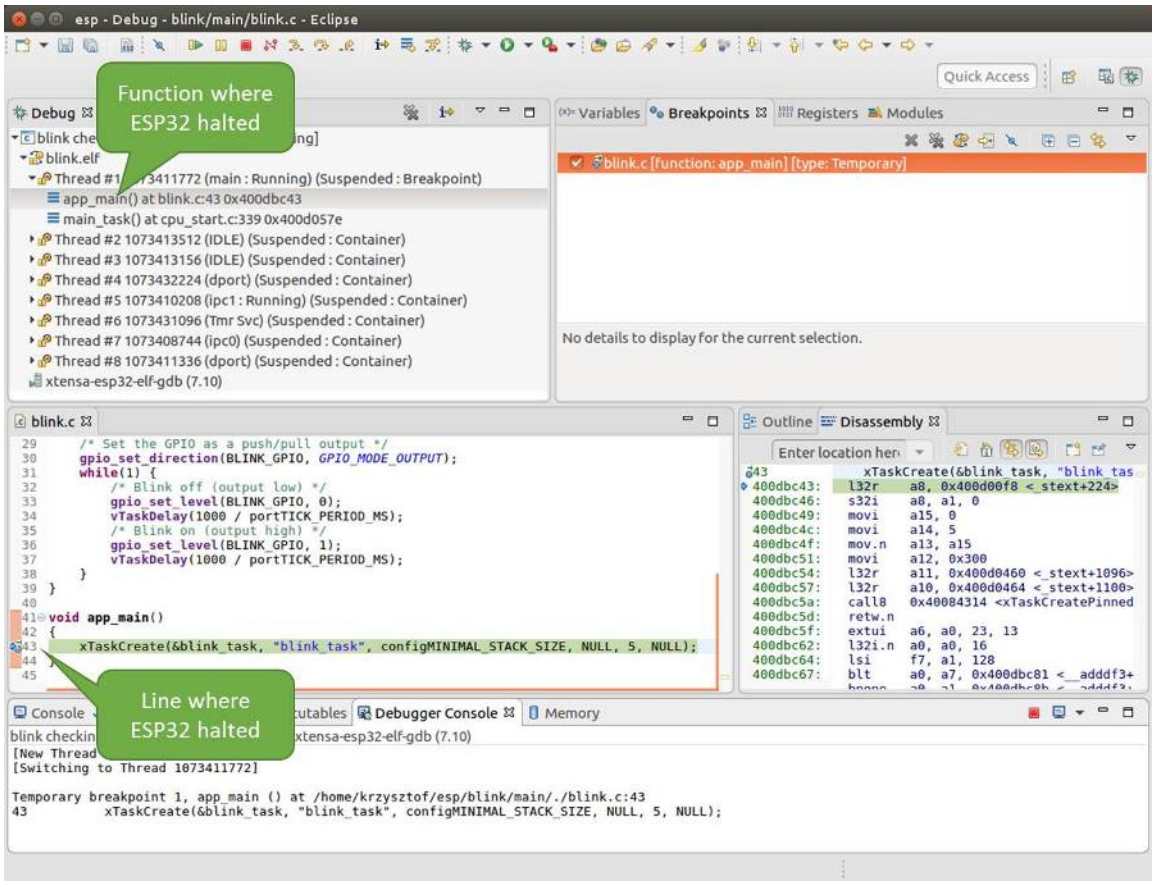


图 12: 调试时目标停止

暂停的程序所在线程也会被展开，显示函数调用的堆栈，它表示直到目标暂停所在代码行（下图高亮处）为止的相关函数的调用关系。1 号线程下函数调用堆栈的第一行包含了最后一个调用的函数 `app_main()`，根据下一行显示，它又是在函数 `main_task()` 中被调用的。堆栈的每一行还包含调用函数的文件名和行号。通过单击每个堆栈的条目，在下面的窗口中，你将看到此文件的内容。

通过展开线程，你可以浏览整个应用程序。展开 5 号线程，它包含了更长的函数调用堆栈，你可以看到函数调用旁边的数字，比如 `0x4000000c`，它们代表未以源码形式提供的二进制代码所在的内存地址。

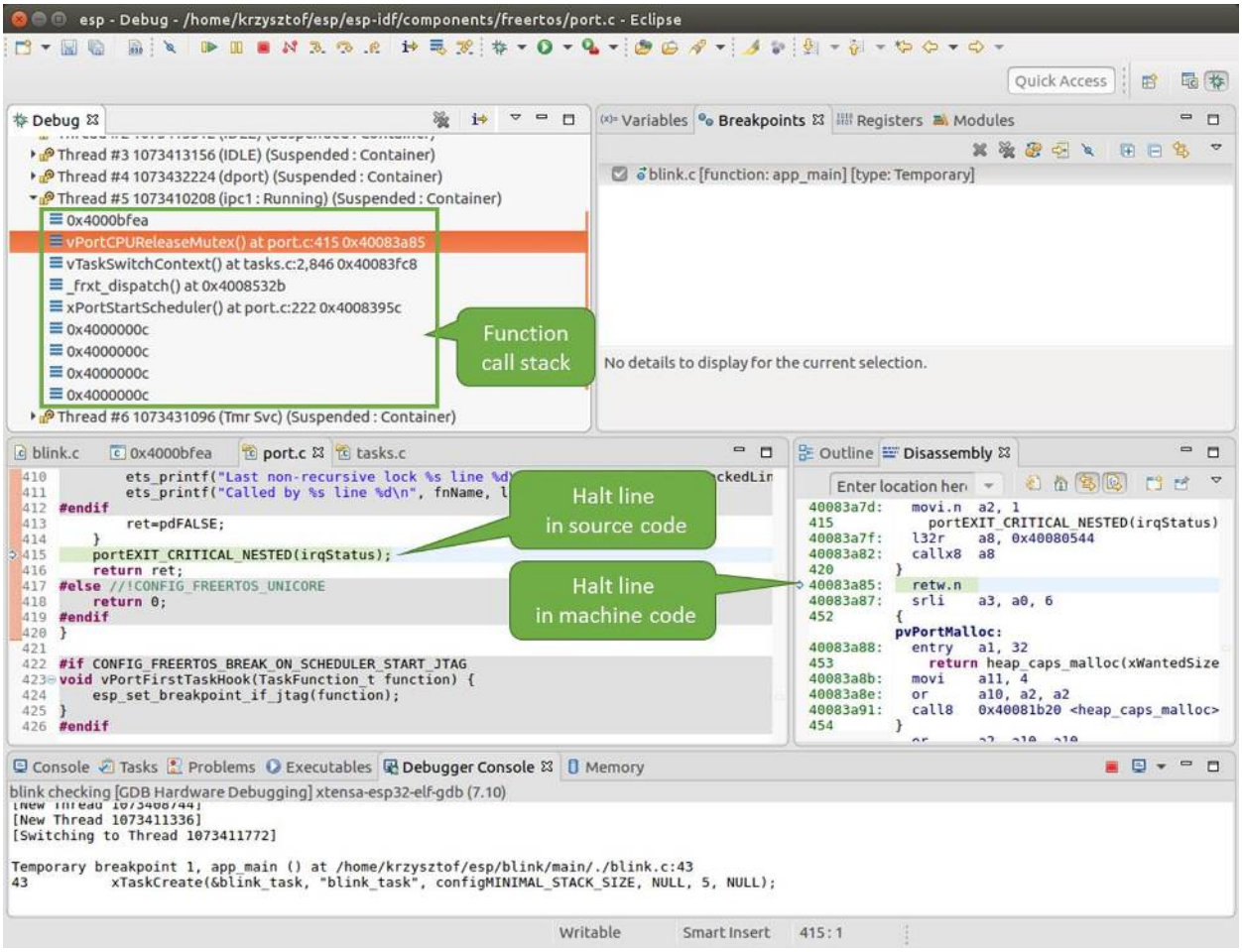


图 13: 浏览函数调用堆栈

无论项目是以源代码还是仅以二进制形式提供，在右边一个窗口中，都可以看到反汇编后的机器代码。

回到 1 号线程中的 `app_main()` 函数所在的 `blink.c` 源码文件，下面的示例将会以该文件为例介绍调试的常用功能。调试器可以轻松浏览整个应用程序的代码，这给单步调试代码和设置断点带来了很大的便利，下面将一一展开讨论。

设置和清除断点

在调试时，我们希望能够在关键的代码行停止应用程序，然后检查特定的变量、内存、寄存器和外设的状态。为此我们需要使用断点，以便在特定某行代码处快速访问和停止应用程序。

我们在控制 LED 状态发生变化的两处代码行分别设置一个断点。基于以上代码列表，这两处分别为第 33 和 36 代码行。按住键盘上的“Control”键，双击 blink.c 文件中的行号 33，并在弹出的对话框中点击“OK”按钮进行确定。如果你不想看到此对话框，双击行号即可。执行同样操作，在第 36 行设置另外一个断点。

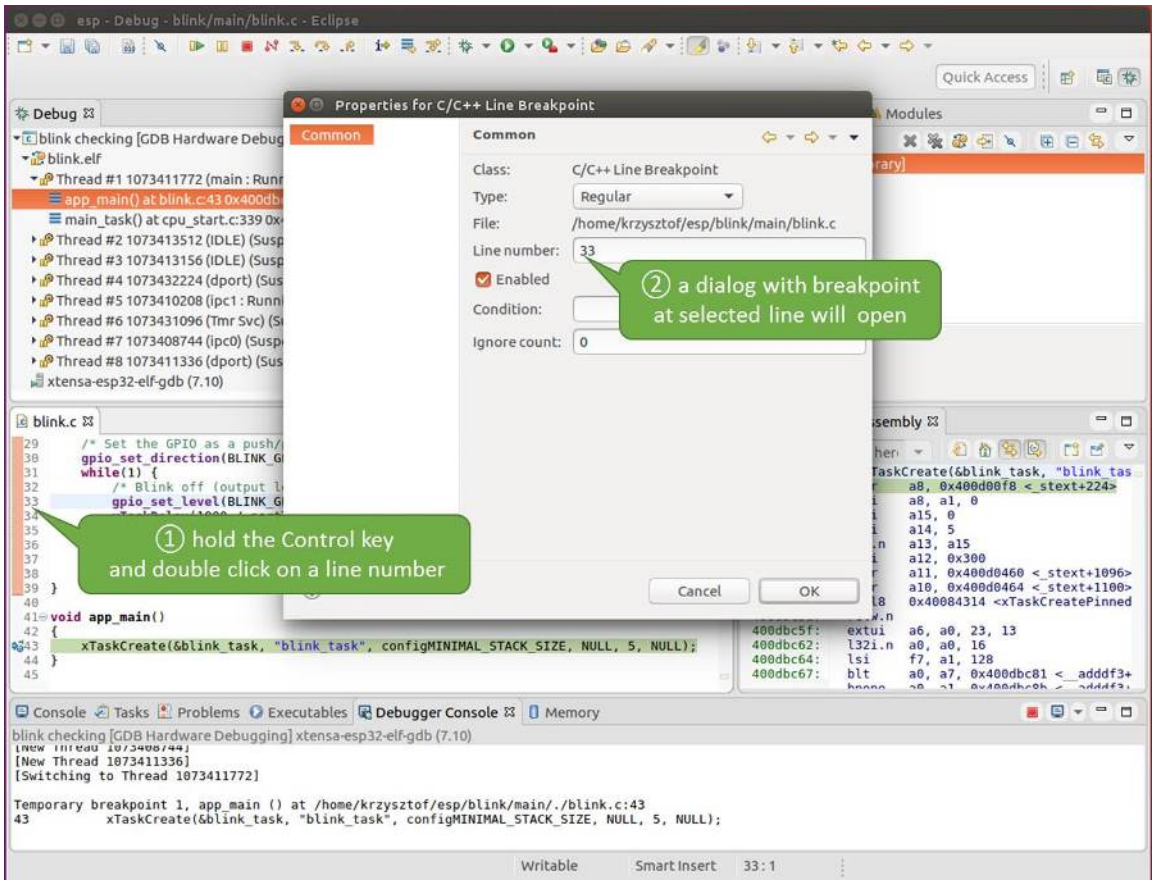


图 14: 设置断点

断点的数量和位置信息会显示在右上角的“断点”窗口中。单击“Show Breakpoints Supported by Selected Target”图标可以刷新此列表。除了刚才设置的两个断点外，列表中可能还包含在调试器启动时设置在 `app_main()` 函数处的临时断点。由于最多只允许设置两个断点（详细信息请参阅可用的断点和观察点），你需要将其删除，否则调试会失败。

单击“Resume”（如果“Resume”按钮是灰色的，请先单击 8 号线程的 `blink_task()` 函数）后处理器将开始继续运行，并在断点处停止。再一次单击“Resume”按钮，使程序再次运行，然后停在第二个断点处，依次类推。

每次单击“Resume”按钮恢复程序运行后，都会看到 LED 切换状态。

更多关于断点的信息，请参阅可用的断点和观察点 和关于断点的补充知识。

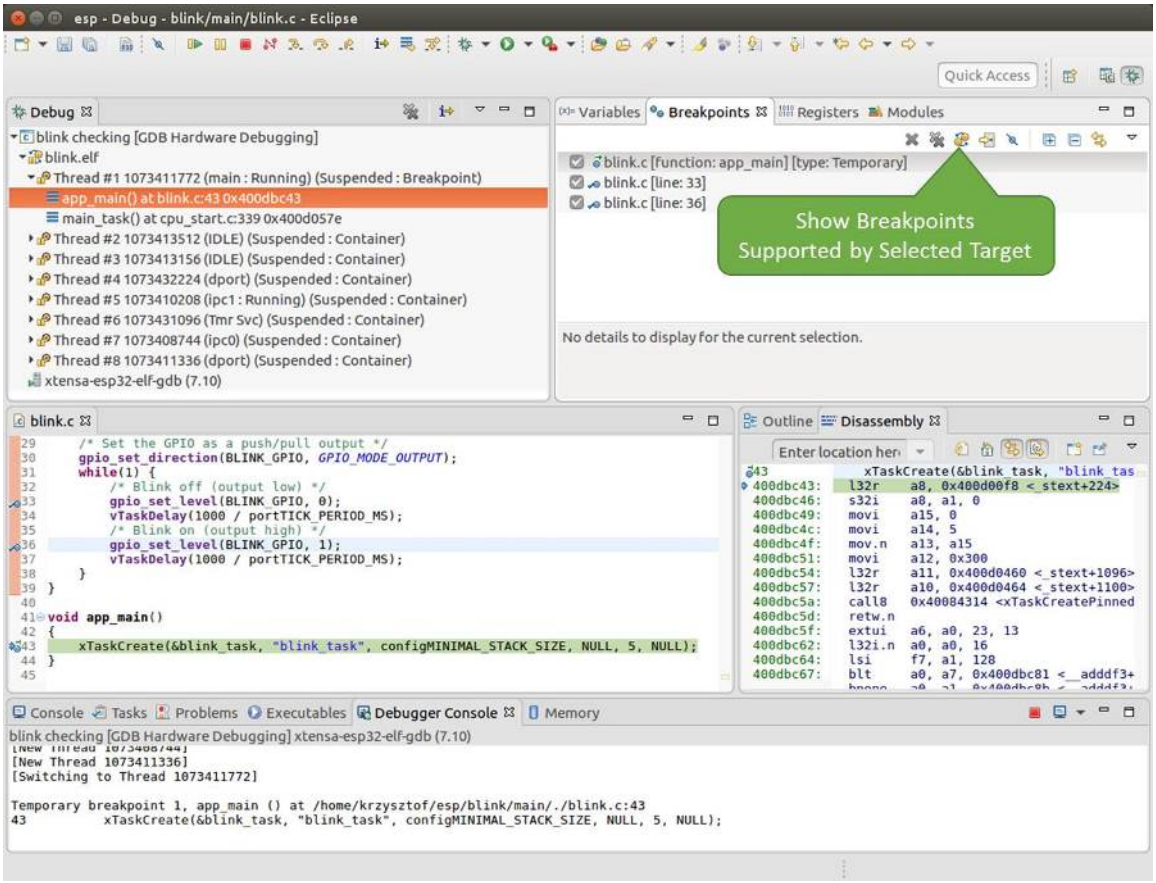


图 15: 设置了三个断点 / 最多允许两个断点

手动暂停目标

在调试时，你可以恢复程序运行并输入代码等待某个事件发生或者保持无限循环而不设置任何断点。后者，如果想要返回调试模式，可以通过单击“Suspend”按钮来手动中断程序的运行。

在此之前，请删除所有的断点，然后单击“Resume”按钮。接着单击“Suspend”按钮，应用程序会停止在某个随机的位置，此时 LED 也将停止闪烁。调试器将展开线程并高亮显示停止的代码行。

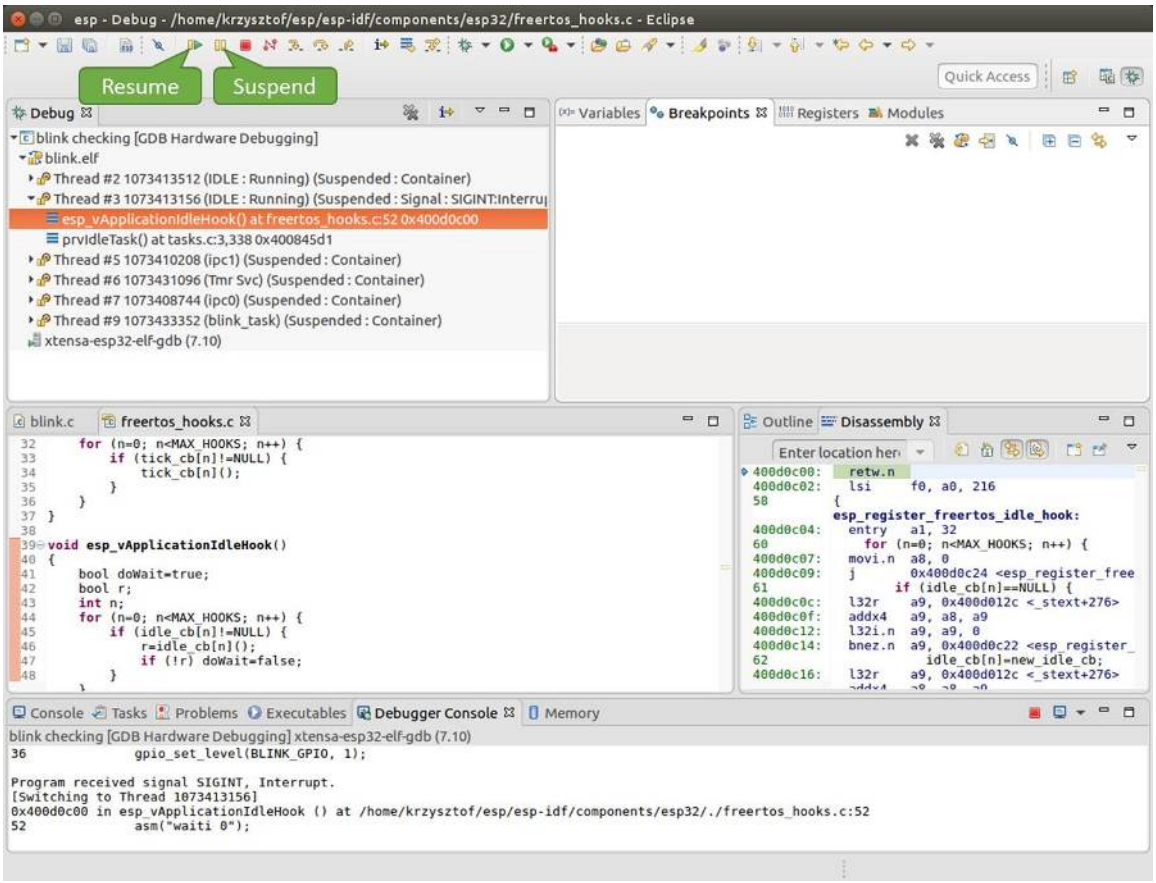


图 16: 手动暂停目标

在上图所示的情况中，应用程序已经在 `freertos_hooks.c` 文件的第 52 行暂停运行，现在你可以通过单击“Resume”按钮再次将其恢复运行或者进行下面要介绍的调试工作。

单步执行代码

我们还可以使用“Step Into (F5)”和“Step Over (F6)”命令单步执行代码，这两者之间的区别是执行“Step Into (F5)”命令会进入调用的子程序，而执行“Step Over (F6)”命令则会直接将子程序看成单个源码行，单步就能将其运行结束。

在继续演示此功能之前，请参照上文所述确保目前只在 `blink.c` 文件的第 36 行设置了一个断点。

按下 F8 键让程序继续运行然后在断点处停止运行，多次按下 “Step Over (F6)” 按钮，观察调试器是如何单步执行一行代码的。

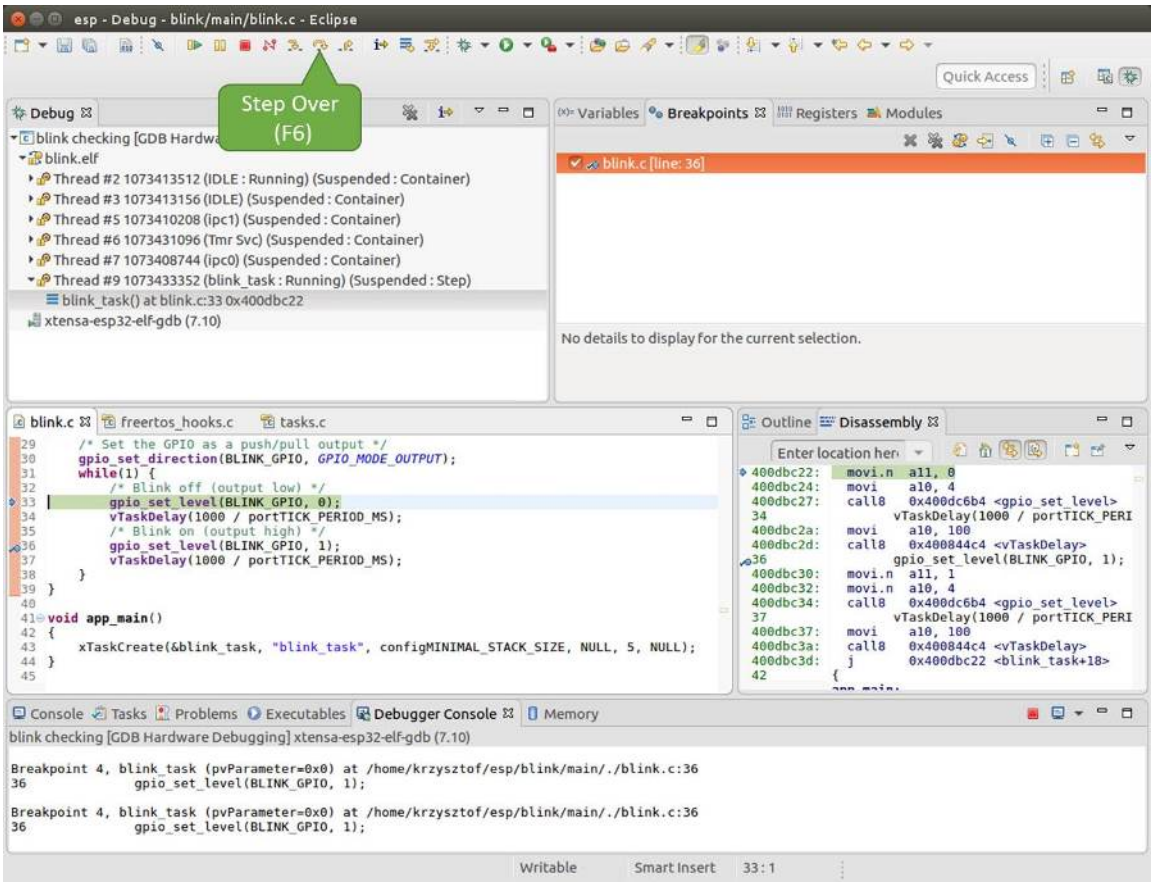


图 17: 使用 “Step Over (F6)” 单步执行代码

如果你改用 “Step Into (F5)”，那么调试器将会进入调用的子程序内部。

在上述例子中，调试器进入 `gpio_set_level(BLINK_GPIO, 0)` 代码内部，同时代码窗口快速切换到 `gpio.c` 驱动文件。

请参阅 *“next” 命令无法跳过子程序的原因* 文档以了解 `next` 命令的潜在局限。

查看并设置内存

要显示或者设置内存的内容，请使用 “调试” 视图中位于底部的 “Memory” 选项卡。

在 “Memory” 选项卡下，我们将在内存地址 `0x3FF44004` 处读取和写入内容。该地址也是 `GPIO_OUT_REG` 寄存器的地址，可以用来控制（设置或者清除）某个 GPIO 的电平。关于该寄存器的更多详细信息，请参阅 ESP32 技术参考手册 中的 IO_MUX 和 GPIO Matrix 章节。

同样在 `blink.c` 项目文件中，在两个 `gpio_set_level` 语句的后面各设置一个断点，单击 “Memory” 选项卡，然后单击 “Add Memory Monitor” 按钮，在弹出的对话框中输入 `0x3FF44004`。

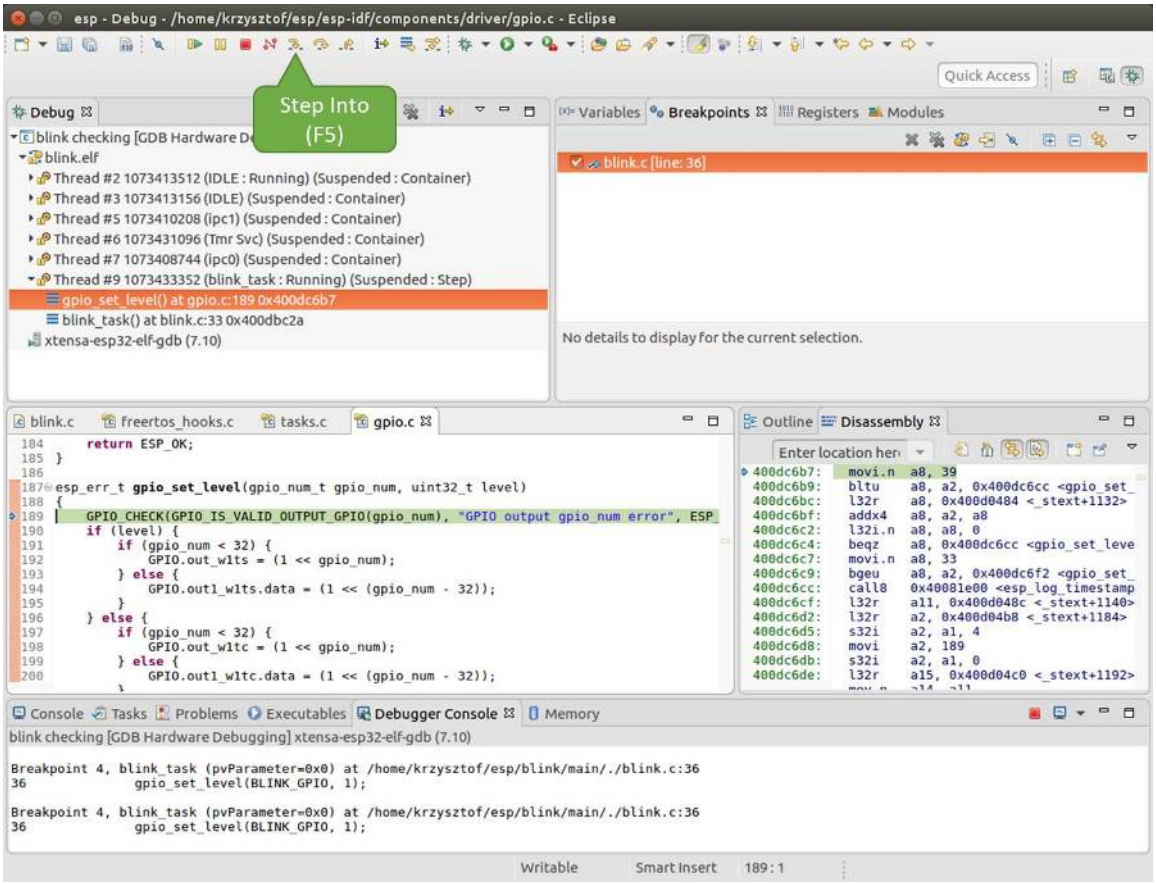


图 18: 使用“Step Into (F5)”单步执行代码

按下 F8 按键恢复程序运行，并观察“Monitor”选项卡。

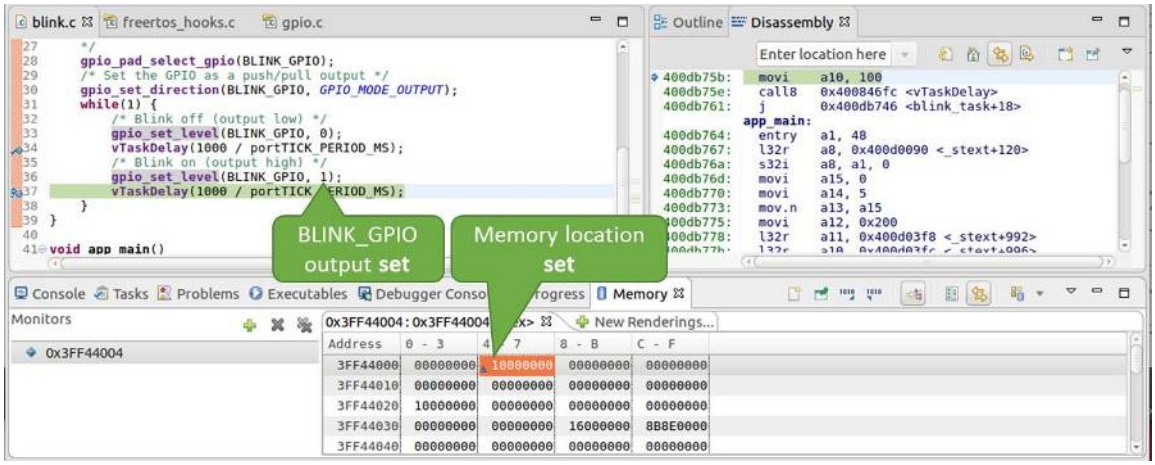


图 19: 观察内存地址 0x3FF44004 处的某个比特被置高

每按一下 F8，你就会看到在内存 `0x3FF44004` 地址处的一个比特位被翻转（并且 LED 会改变状态）。

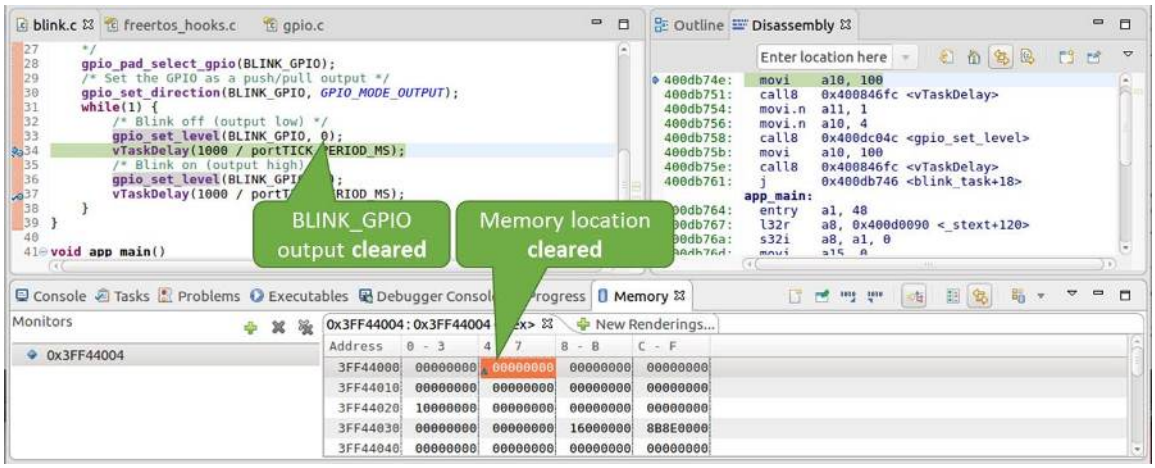


图 20: 观察内存地址 0x3FF44004 处的某个比特被置低

要修改内存的数值，请在“Monitor”选项卡中找到待修改的内存地址，如前面观察的结果一样，输入特定比特翻转后的值。当按下回车键后，将立即看到 LED 的状态发生了改变。

观察和设置程序变量

常见的调试任务是在程序运行期间检查程序中某个变量的值，为了演示这个功能，更新 `blink.c` 文件，在 `blink_task` 函数的上面添加一个全局变量的声明 `int i`，然后在 `while(1)` 里添加 `i++`，这样每次 LED 改变状态的时候，变量 `i` 都会增加 1。

退出调试器，这样就不会与新代码混淆，然后重新构建并烧写代码到 ESP32 中，接着重启调试器。注意，这里不需要我们重启 OpenOCD。

一旦程序停止运行，在代码 `i++` 处添加一个断点。

下一步，在 “Breakpoints” 所在的窗口中，选择 “Expressions” 选项卡。如果该选项卡不存在，请在顶部菜单栏的 Window > Show View > Expressions 中添加这一选项卡。然后在该选项卡中单击 “Add new expression”，并输入 `i`。

按下 F8 继续运行程序，每次程序停止时，都会看到变量 `i` 的值在递增。

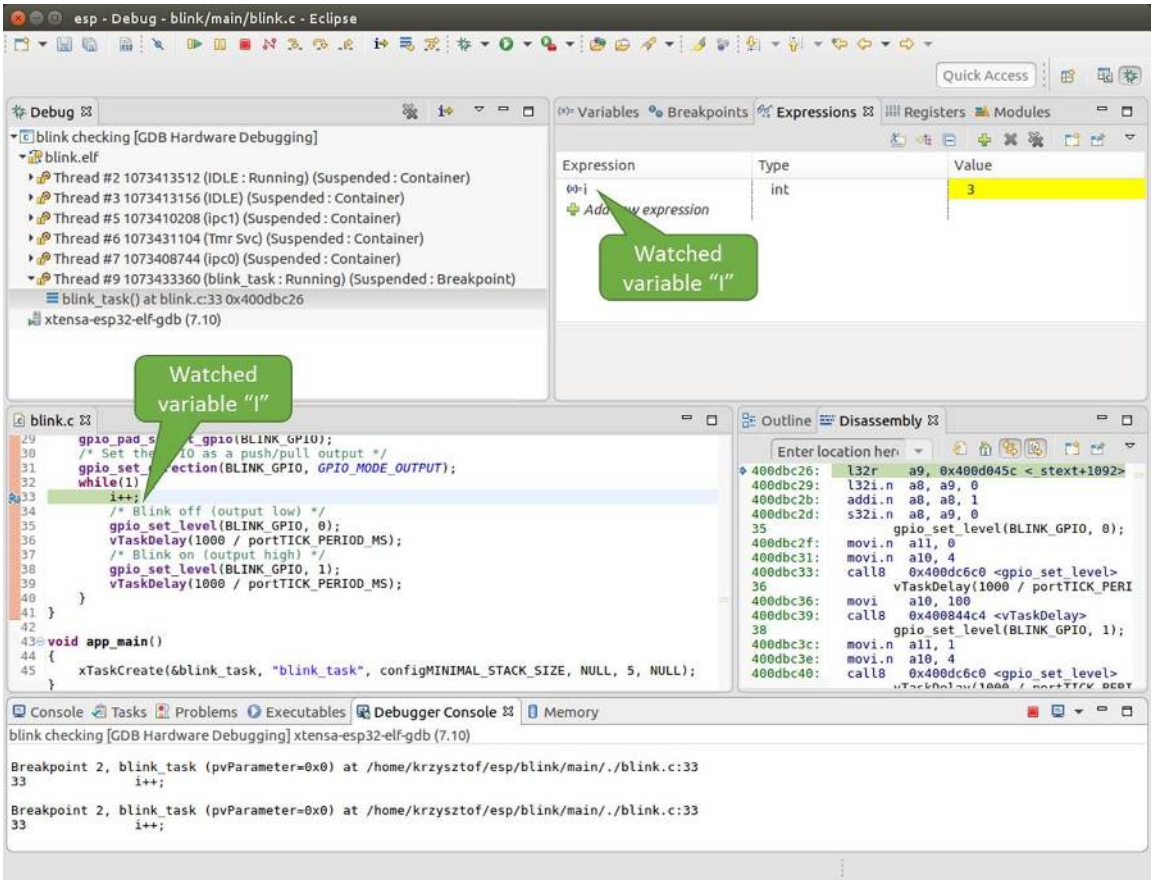


图 21: 观察程序变量 “i”

如想更改 `i` 的值，可以在 “Value” 一栏中输入新的数值。按下 “Resume (F8)” 后，程序将从新输入的数字开始递增 `i`。

设置条件断点

接下来的内容更为有趣，你可能想在一定条件满足的情况下设置断点，然后让程序停止运行。右击断点打开上下文菜单，选择 “Breakpoint Properties”，将 “Type:” 改选为 “Hardware” 然后在 “Condition:” 一栏中输入条件表达式，例如 `i == 2`。

如果当前 `i` 的值小于 2（如果有需要也可以更改这个阈值）并且程序被恢复运行，那么 LED 就会循环闪烁，直到 `i == 2` 条件成立，最后程序停止在该处。

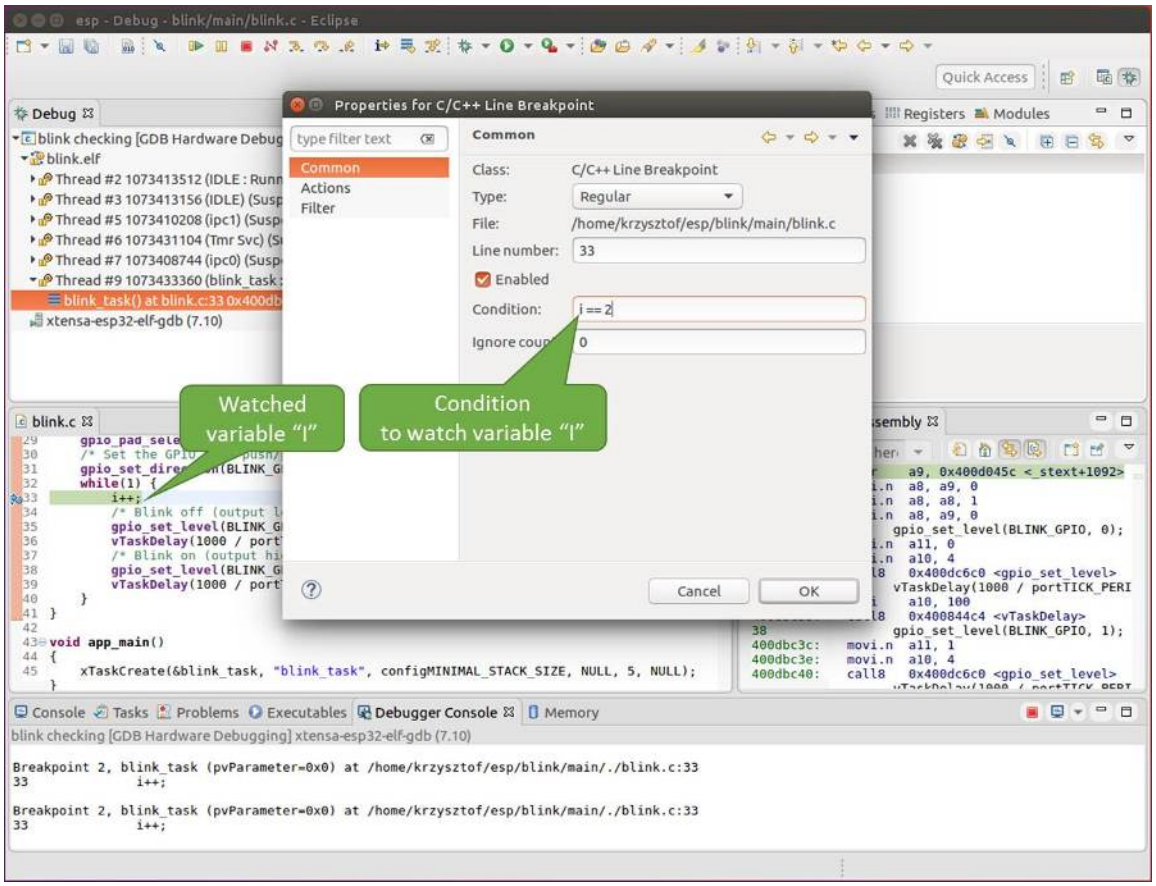

esp - Debug - blink/main/blink.c - Eclipse
Properties for C/C++ Line Breakpoint
Common
Actions
Filter
Class: C/C++ Line Breakpoint
Type: Regular
File: /home/krzysztof/esp/blink/main/blink.c
Line number: 33
Enabled
Condition: i == 2
Cancel
OK
Watched variable "I"
Condition to watch variable "I"
Debugger Console
Breakpoint 2, blink_task (pvParameter=0x0) at /home/krzysztof/esp/blink/main/./blink.c:33
33 i++;

图 22: 设置条件断点

使用命令行的调试示例

请检查您的目标板是否已经准备好，并加载了 get-started/blink 示例代码，然后按照*在命令行中使用 GDB* 中介绍的步骤配置和启动调试器，最后选择让应用程序在 app_main() 建立的断点处停止运行

```
Temporary breakpoint 1, app_main () at /home/user-name/esp/blink/main/./blink.c:43
43          xTaskCreate(&blink_task, "blink_task", configMINIMAL_STACK_SIZE, NULL, 5,␣
↪NULL);
(gdb)
```

本小节的示例

1. 浏览代码，查看堆栈和线程
2. 设置和清除断点
3. 暂停和恢复应用程序的运行
4. 单步执行代码
5. 查看并设置内存
6. 观察和设置程序变量
7. 设置条件断点

浏览代码，查看堆栈和线程

当看到 (gdb) 提示符的时候，应用程序已停止运行，LED 也停止闪烁。

要找到代码暂停的位置，输入 l 或者 list 命令，调试器会打印出停止点（blink.c 代码文件的第 43 行）附近的几行代码

```
(gdb) l
38          }
39      }
40
41      void app_main()
42      {
43          xTaskCreate(&blink_task, "blink_task", configMINIMAL_STACK_SIZE, NULL, 5,␣
↪NULL);
44      }
(gdb)
```

也可以通过输入 l 30，40 等命令来查看特定行号范围内的代码。

使用 bt 或者 backtrace 来查看哪些函数最终导致了此代码被调用:

```
(gdb) bt
#0  app_main () at /home/user-name/esp/blink/main/./blink.c:43
#1  0x400d057e in main_task (args=0x0) at /home/user-name/esp/esp-idf/components/esp32/./
↪cpu_start.c:339
(gdb)
```

输出的第 0 行表示应用程序暂停之前调用的最后一个函数，即我们之前列出的 app_main ()。app_main () 又被位于 cpu_start.c 文件第 339 行的 main_task 函数调用。

想查看 cpu_start.c 文件中 main_task 函数的上下文，需要输入 frame N，其中 N = 1，因为根据前面的输出，main_task 位于 #1 下:

```
(gdb) frame 1
#1  0x400d057e in main_task (args=0x0) at /home/user-name/esp/esp-idf/components/esp32/./
↪cpu_start.c:339
339         app_main();
(gdb)
```

输入 l 将显示一段名为 app_main() 的代码（在第 339 行）:

```
(gdb) l
334             ;
335         }
336     #endif
337         //Enable allocation in region where the startup stacks were located.
338         heap_caps_enable_nonos_stack_heaps();
339         app_main();
340         vTaskDelete(NULL);
341     }
342
(gdb)
```

通过打印前面的一些行，你会看到我们一直在寻找的 main_task 函数:

```
(gdb) l 326, 341
326     static void main_task(void* args)
327     {
328         // Now that the application is about to start, disable boot watchdogs
329         REG_CLR_BIT(TIMG_WDTCONFIG0_REG(0), TIMG_WDT_FLASHBOOT_MOD_EN_S);
330         REG_CLR_BIT(RTC_CNTL_WDTCONFIG0_REG, RTC_CNTL_WDT_FLASHBOOT_MOD_EN);
331     #if !CONFIG_FREERTOS_UNICORE
```

(下页继续)

(续上页)

```
332         // Wait for FreeRTOS initialization to finish on APP CPU, before replacing␣
↪its startup stack
333         while (port_xSchedulerRunning[1] == 0) {
334             ;
335         }
336     #endif
337         //Enable allocation in region where the startup stacks were located.
338         heap_caps_enable_nonos_stack_heaps();
339         app_main();
340         vTaskDelete(NULL);
341     }
(gdb)
```

如果要查看其他代码，可以输入 `i threads` 命令，则会输出目标板上运行的线程列表:

```
(gdb) i threads
  Id   Target Id         Frame
  8    Thread 1073411336 (dport) 0x400d0848 in dport_access_init_core (arg=<optimized␣
↪out>)
    at /home/user-name/esp/esp-idf/components/esp32/./dport_access.c:170
  7    Thread 1073408744 (ipc0) xQueueGenericReceive (xQueue=0x3ffae694, pvBuffer=0x0,␣
↪xTicksToWait=1644638200,
    xJustPeeking=0) at /home/user-name/esp/esp-idf/components/freertos/./queue.c:1452
  6    Thread 1073431096 (Tmr Svc) prvTimerTask (pvParameters=0x0)
    at /home/user-name/esp/esp-idf/components/freertos/./timers.c:445
  5    Thread 1073410208 (ipc1 : Running) 0x4000bfea in ?? ()
  4    Thread 1073432224 (dport) dport_access_init_core (arg=0x0)
    at /home/user-name/esp/esp-idf/components/esp32/./dport_access.c:150
  3    Thread 1073413156 (IDLE) prvIdleTask (pvParameters=0x0)
    at /home/user-name/esp/esp-idf/components/freertos/./tasks.c:3282
  2    Thread 1073413512 (IDLE) prvIdleTask (pvParameters=0x0)
    at /home/user-name/esp/esp-idf/components/freertos/./tasks.c:3282
* 1    Thread 1073411772 (main : Running) app_main () at /home/user-name/esp/blink/main/.
↪/blink.c:43
(gdb)
```

线程列表显示了每个线程最后一个被调用的函数以及所在的 C 源文件名（如果存在的话）。

您可以通过输入 `thread N` 进入特定的线程，其中 `N` 是线程 ID。我们进入 5 号线程来看一下它是如何工作的:

```
(gdb) thread 5
[Switching to thread 5 (Thread 1073410208)]
#0  0x4000bfea in ?? ()
(gdb)
```

然后查看回溯:

```
(gdb) bt
#0  0x4000bfea in ?? ()
#1  0x40083a85 in vPortCPUReleaseMutex (mux=<optimized out>) at /home/user-name/esp/esp-
↪idf/components/freertos/./port.c:415
#2  0x40083fc8 in vTaskSwitchContext () at /home/user-name/esp/esp-idf/components/
↪freertos/./tasks.c:2846
#3  0x4008532b in _frxt_dispatch ()
#4  0x4008395c in xPortStartScheduler () at /home/user-name/esp/esp-idf/components/
↪freertos/./port.c:222
#5  0x4000000c in ?? ()
#6  0x4000000c in ?? ()
#7  0x4000000c in ?? ()
#8  0x4000000c in ?? ()
(gdb)
```

如上所示，回溯可能会包含多个条目，方便查看直至目标停止运行的函数调用顺序。如果找不到某个函数的源码文件，将会使用问号 `??` 替代，这表示该函数是以二进制格式提供的。像 `0x4000bfea` 这样的值是被调用函数所在的内存地址。

使用诸如 `bt`，`i threads`，`thread N` 和 `list` 命令可以浏览整个应用程序的代码。这给单步调试代码和设置断点带来很大的便利，下面将一一展开来讨论。

设置和清除断点

在调试时，我们希望能够在关键的代码行停止应用程序，然后检查特定的变量、内存、寄存器和外设的状态。为此我们需要使用断点，以便在特定某行代码处快速访问和停止应用程序。

我们在控制 LED 状态发生变化的两处代码行分别设置一个断点。基于以上代码列表，这两处分别为第 33 和 36 代码行。使用命令 `break M` 设置断点，其中 M 是具体的代码行:

```
(gdb) break 33
Breakpoint 2 at 0x400db6f6: file /home/user-name/esp/blink/main/./blink.c, line 33.
(gdb) break 36
Breakpoint 3 at 0x400db704: file /home/user-name/esp/blink/main/./blink.c, line 36.
```

输入命令 c，处理器将运行并在断点处停止。再次输入 c 将使其再次运行，并在第二个断点处停止，依此类推:

```
(gdb) c
Continuing.
Target halted. PRO_CPU: PC=0x400DB6F6 (active)    APP_CPU: PC=0x400D10D8

Breakpoint 2, blink_task (pvParameter=0x0) at /home/user-name/esp/blink/main/./blink.c:33
33          gpio_set_level(BLINK_GPIO, 0);
(gdb) c
Continuing.
Target halted. PRO_CPU: PC=0x400DB6F8 (active)    APP_CPU: PC=0x400D10D8
Target halted. PRO_CPU: PC=0x400DB704 (active)    APP_CPU: PC=0x400D10D8

Breakpoint 3, blink_task (pvParameter=0x0) at /home/user-name/esp/blink/main/./blink.c:36
36          gpio_set_level(BLINK_GPIO, 1);
(gdb)
```

只有在输入命令 c 恢复程序运行后才能看到 LED 改变状态。

查看已设置断点的数量和位置，请使用命令 info break:

```
(gdb) info break
Num     Type           Disp Enb Address    What
2       breakpoint     keep y   0x400db6f6 in blink_task at /home/user-name/esp/blink/
↪main/./blink.c:33
    breakpoint already hit 1 time
3       breakpoint     keep y   0x400db704 in blink_task at /home/user-name/esp/blink/
↪main/./blink.c:36
    breakpoint already hit 1 time
(gdb)
```

请注意，断点序号（在 Num 栏列出）从 2 开始，这是因为在调试器启动时执行 thb app_main 命令已经在 app_main() 函数处建立了第一个断点。由于它是一个临时断点，已经被自动删除，所以没有被列出。

要删除一个断点，请输入 delete N 命令（或者简写成 d N），其中 N 代表断点序号:

```
(gdb) delete 1
No breakpoint number 1.
(gdb) delete 2
(gdb)
```

更多关于断点的信息，请参阅可用的断点和观察点 和关于断点的补充知识。

暂停和恢复应用程序的运行

在调试时，可以恢复程序运行并输入代码等待某个事件发生或者保持无限循环而不设置任何断点。对于后者，想要返回调试模式，可以通过输入 Ctrl+C 手动中断程序的运行。

在此之前，请删除所有的断点，然后输入 `c` 恢复程序运行。接着输入 Ctrl+C，应用程序会停止在某个随机的位置，此时 LED 也将停止闪烁。调试器会打印如下信息:

```
(gdb) c
Continuing.
^CTarget halted. PRO_CPU: PC=0x400D0C00             APP_CPU: PC=0x400D0C00 (active)
[New Thread 1073433352]

Program received signal SIGINT, Interrupt.
[Switching to Thread 1073413512]
0x400d0c00 in esp_vApplicationIdleHook () at /home/user-name/esp/esp-idf/components/
↪esp32/./freertos_hooks.c:52
52              asm("waiti 0");
(gdb)
```

在上图所示的情况下，应用程序已经在 `freertos_hooks.c` 文件的第 52 行暂停运行，现在您可以通过输入 `c` 再次将其恢复运行或者进行如下所述的一些调试工作。

注解: 在 MSYS2 的 shell 中输入 Ctrl+C 并不会暂停目标的运行，而是会退出调试器。解决这个问题的方法可以通过使用 *Eclipse* 来调试 或者参考 http://www.mingw.org/wiki/Workaround_for_GDB_Ctrl_C_Interrupt 里的解决方案。

单步执行代码

我们还可以使用 `step` 和 `next` 命令（可以简写成 `s` 和 `n`）单步执行代码，这两者之间的区别是执行 “step” 命令会进入调用的子程序内部，而执行 “next” 命令则会直接将子程序看成单个源码行，单步就能将其运行结束。

在继续演示此功能之前，请使用前面介绍的 `break` 和 `delete` 命令，确保目前只在 `blink.c` 文件的第 36 行设置了一个断点:

```
(gdb) info break
Num     Type           Disp Enb Address    What
3       breakpoint     keep y   0x400db704 in blink_task at /home/user-name/esp/blink/
↪main/./blink.c:36
    breakpoint already hit 1 time
```

(下页继续)

(续上页)

```
(gdb)
```

输入 c 恢复程序运行然后等它在断点处停止运行:

```
(gdb) c
Continuing.
Target halted. PRO_CPU: PC=0x400DB754 (active)    APP_CPU: PC=0x400D1128

Breakpoint 3, blink_task (pvParameter=0x0) at /home/user-name/esp/blink/main/./blink.c:36
36          gpio_set_level(BLINK_GPIO, 1);
(gdb)
```

然后输入 n 多次，观察调试器是如何单步执行一行代码的:

```
(gdb) n
Target halted. PRO_CPU: PC=0x400DB756 (active)    APP_CPU: PC=0x400D1128
Target halted. PRO_CPU: PC=0x400DB758 (active)    APP_CPU: PC=0x400D1128
Target halted. PRO_CPU: PC=0x400DC04C (active)    APP_CPU: PC=0x400D1128
Target halted. PRO_CPU: PC=0x400DB75B (active)    APP_CPU: PC=0x400D1128
37          vTaskDelay(1000 / portTICK_PERIOD_MS);
(gdb) n
Target halted. PRO_CPU: PC=0x400DB75E (active)    APP_CPU: PC=0x400D1128
Target halted. PRO_CPU: PC=0x400846FC (active)    APP_CPU: PC=0x400D1128
Target halted. PRO_CPU: PC=0x400DB761 (active)    APP_CPU: PC=0x400D1128
Target halted. PRO_CPU: PC=0x400DB746 (active)    APP_CPU: PC=0x400D1128
33          gpio_set_level(BLINK_GPIO, 0);
(gdb)
```

如果你输入 s，那么调试器将进入子程序:

```
(gdb) s
Target halted. PRO_CPU: PC=0x400DB748 (active)    APP_CPU: PC=0x400D1128
Target halted. PRO_CPU: PC=0x400DB74B (active)    APP_CPU: PC=0x400D1128
Target halted. PRO_CPU: PC=0x400DC04C (active)    APP_CPU: PC=0x400D1128
Target halted. PRO_CPU: PC=0x400DC04F (active)    APP_CPU: PC=0x400D1128
gpio_set_level (gpio_num=GPIO_NUM_4, level=0) at /home/user-name/esp/esp-idf/components/
↪driver/./gpio.c:183
183     GPIO_CHECK(GPIO_IS_VALID_OUTPUT_GPIO(gpio_num), "GPIO output gpio_num error",␣
↪ESP_ERR_INVALID_ARG);
(gdb)
```

上述例子中，调试器进入 gpio_set_level(BLINK_GPIO, 0) 代码内部，同时代码窗口快速切换到 gpio.c

驱动文件。

请参阅 “*next* 命令无法跳过子程序的原因 文档以了解 `next` 命令的潜在局限。

查看并设置内存

使用命令 `x` 可以显示内存的内容，配合其余参数还可以调整所显示内存位置的格式和数量。运行 `help x` 可以查看更多相关细节。与 `x` 命令配合使用的命令是 `set`，它允许你将值写入内存。

为了演示 `x` 和 `set` 的使用，我们将在内存地址 `0x3FF44004` 处读取和写入内容。该地址也是 `GPIO_OUT_REG` 寄存器的地址，可以用来控制（设置或者清除）某个 GPIO 的电平。关于该寄存器的更多详细信息，请参阅 ESP32 技术参考手册 中的 IO_MUX 和 GPIO Matrix 章节。

同样在 `blink.c` 项目文件中，在两个 `gpio_set_level` 语句的后面各设置一个断点。输入两次 `c` 命令后停止在断点处，然后输入 `x /1wx 0x3FF44004` 来显示 `GPIO_OUT_REG` 寄存器的值:

```
(gdb) c
Continuing.
Target halted. PRO_CPU: PC=0x400DB75E (active)    APP_CPU: PC=0x400D1128
Target halted. PRO_CPU: PC=0x400DB74E (active)    APP_CPU: PC=0x400D1128

Breakpoint 2, blink_task (pvParameter=0x0) at /home/user-name/esp/blink/main/./blink.c:34
34          vTaskDelay(1000 / portTICK_PERIOD_MS);
(gdb) x /1wx 0x3FF44004
0x3ff44004: 0x00000000
(gdb) c
Continuing.
Target halted. PRO_CPU: PC=0x400DB751 (active)    APP_CPU: PC=0x400D1128
Target halted. PRO_CPU: PC=0x400DB75B (active)    APP_CPU: PC=0x400D1128

Breakpoint 3, blink_task (pvParameter=0x0) at /home/user-name/esp/blink/main/./blink.c:37
37          vTaskDelay(1000 / portTICK_PERIOD_MS);
(gdb) x /1wx 0x3FF44004
0x3ff44004: 0x00000010
(gdb)
```

如果闪烁的 LED 连接到了 GPIO4，那么每次 LED 改变状态时你会看到第 4 比特被翻转:

```
0x3ff44004: 0x00000000
...
0x3ff44004: 0x00000010
```

现在，当 LED 熄灭时，与之对应地会显示 `0x3ff44004: 0x00000000`，尝试使用 `set` 命令向相同的内存地址写入 `0x00000010` 来将该比特置高:

```
(gdb) x /1wx 0x3FF44004
0x3ff44004: 0x00000000
(gdb) set {unsigned int}0x3FF44004=0x000010
```

在输入 `set {unsigned int}0x3FF44004=0x000010` 命令后，你会立即看到 LED 亮起。

观察和设置程序变量

常见的调试任务是在程序运行期间检查程序中某个变量的值，为了能够演示这个功能，更新 `blink.c` 文件，在 `blink_task` 函数的上面添加一个全局变量的声明 `int i`，然后在 `while(1)` 里添加 `i++`，这样每次 LED 改变状态的时候，变量 `i` 都会增加 1。

退出调试器，这样就不会与新代码混淆，然后重新构建并烧写代码到 ESP32 中，接着重启调试器。注意，这里不需要我们重启 OpenOCD。

一旦程序停止运行，输入命令 `watch i`:

```
(gdb) watch i
Hardware watchpoint 2: i
(gdb)
```

这会在所有变量 `i` 发生改变的代码处插入所谓的 “观察点”。现在输入 `continue` 命令来恢复应用程序的运行并观察它停止:

```
(gdb) c
Continuing.
Target halted. PRO_CPU: PC=0x400DB751 (active)    APP_CPU: PC=0x400D0811
[New Thread 1073432196]

Program received signal SIGTRAP, Trace/breakpoint trap.
[Switching to Thread 1073432196]
0x400db751 in blink_task (pvParameter=0x0) at /home/user-name/esp/blink/main/./blink.c:33
33          i++;
(gdb)
```

多次恢复程序运行后，变量 `i` 的值会增加，现在你可以输入 `print i` （简写 `p i`）来查看当前 `i` 的值:

```
(gdb) p i
$1 = 3
(gdb)
```

要修改 `i` 的值，请使用 `set` 命令，如下所示（可以将其打印输出来查看是否确已修改）:

```
(gdb) set var i = 0
(gdb) p i
$3 = 0
(gdb)
```

最多可以使用两个观察点，详细信息请参阅可用的断点和观察点。

设置条件断点

接下来的内容更为有趣，你可能想在一定条件满足的情况下设置断点。请先删除已有的断点，然后尝试如下命令:

```
(gdb) break blink.c:34 if (i == 2)
Breakpoint 3 at 0x400db753: file /home/user-name/esp/blink/main/./blink.c, line 34.
(gdb)
```

以上命令在 `blink.c` 文件的 34 处设置了一个条件断点，当 `i==2` 条件满足时，程序会停止运行。

如果当前 `i` 的值小于 2 并且程序被恢复运行，那么 LED 就会循环闪烁，直到 `i == 2` 条件成立，最后程序停止在该处:

```
(gdb) set var i = 0
(gdb) c
Continuing.
Target halted. PRO_CPU: PC=0x400DB755 (active)    APP_CPU: PC=0x400D112C
Target halted. PRO_CPU: PC=0x400DB753 (active)    APP_CPU: PC=0x400D112C
Target halted. PRO_CPU: PC=0x400DB755 (active)    APP_CPU: PC=0x400D112C
Target halted. PRO_CPU: PC=0x400DB753 (active)    APP_CPU: PC=0x400D112C

Breakpoint 3, blink_task (pvParameter=0x0) at /home/user-name/esp/blink/main/./blink.c:34
34          gpio_set_level(BLINK_GPIO, 0);
(gdb)
```

获得命令的帮助信息

目前所介绍的都是些非常基础的命令，目的在于让您快速上手 JTAG 调试。如果想获得特定命令的语法和功能相关的信息，请在 `(gdb)` 提示符下输入 `help` 和命令名:

```
(gdb) help next
Step program, proceeding through subroutine calls.
Usage: next [N]
```

(下页继续)

(续上页)

```
Unlike "step", if the current source line calls a subroutine,
this command does not enter the subroutine, but instead steps over
the call, in effect treating it as a single source line.
(gdb)
```

只需输入 `help` 命令，即可获得高级命令列表，帮助你了解更多详细信息。此外，还可以参考一些 GDB 命令速查表，比如 http://darkdust.net/files/GDB%20Cheat%20Sheet.pdf。虽然不是所有命令都适用于嵌入式环境，但还是会有所裨益。

结束调试会话

输入命令 `q` 可以退出调试器:

```
(gdb) q
A debugging session is active.

    Inferior 1 [Remote target] will be detached.

Quit anyway? (y or n) y
Detaching from program: /home/user-name/esp/blink/build/blink.elf, Remote target
Ending remote debugging.
user-name@computer-name:~/esp/blink$
```

注意事项和补充内容

[English]

本节提供了本指南中各部分提到的一些注意事项和补充内容。

可用的断点和观察点

ESP32 调试器支持 2 个硬件断点和 64 个软件断点。硬件断点是由 ESP32 芯片内部的逻辑电路实现的，能够设置在代码的任何位置：闪存或者 IRAM 的代码区域。除此以外，OpenOCD 实现了两种软件断点：闪存断点（最多 32 个）和 IRAM 断点（最多 32 个）。目前 GDB 无法在闪存中设置软件断点，因此除非解决此限制，否则这些断点只能由 OpenOCD 模拟为硬件断点。(详细信息可以参阅下面)。ESP32 还支持 2 个观察点，所以可以观察两个变量的变化或者通过 GDB 命令 `watch myVariable` 来读取变量的值。请注意 menuconfig 中的*CONFIG_FREERTOS_WATCHPOINT_END_OF_STACK* 选项会使用第二个观察点，如果你想在 OpenOCD 或者 GDB 中再次尝试使用这个观察点，可能不会得到预期的结果。详情请查看 menuconfig 中的帮助文档。

关于断点的补充知识

使用软件闪存模拟部分硬件断点的意思就是当使用 GDB 命令 `hb myFunction` 给某个函数设置硬件断点时，如果该函数位于闪存中，并且此时还有可用的硬件断点，那调试器就会使用硬件断点，否则就使用 32 个软件闪存断点中的一个来模拟。这个规则同样适用于 `b myFunction` 之类的命令，在这种情况下，GDB 会自己决定该使用哪种类型的断点。如果 `myFunction` 位于可写区域（IRAM），那就会使用软件 IRAM 断点，否则就会像处理 `hb` 命令一样使用硬件断点或者软件闪存断点。

闪存映射 vs 软件闪存断点

为了在闪存中设置或者清除软件断点，OpenOCD 需要知道它们在闪存中的地址。为了完成从 ESP32 的地址空间到闪存地址的转换，OpenOCD 使用闪存中程序代码区域的映射。这些映射被保存在程序映像的头部，位于二进制数据（代码段和数据段）之前，并且特定于写入闪存的每一个应用程序的映像。因此，为了支持软件闪存断点，OpenOCD 需要知道待调试的应用程序映像在闪存中的位置。默认情况下，OpenOCD 会在 0x8000 处读取分区表并使用第一个找到的应用程序映像的映射，但是也可能会存在无法工作的情况，比如分区表不在标准的闪存位置，甚至可能有多个映像：一个出厂映像和两个 OTA 映像，你可能想要调试其中的任意一个。为了涵盖所有可能的调试情况，OpenOCD 支持特殊的命令，用于指定待调试的应用程序映像在闪存中的具体位置。该命令具有以下格式：

```
esp32 appimage_offset <offset>
```

偏移量应为十六进制格式，如果要恢复默认行为，可以将偏移地址设置为 `-1` 。

注解： 由于 GDB 在连接 OpenOCD 时仅仅请求一次内存映射，所以可以在 TCL 配置文件中指定该命令，或者通过命令行传递给 OpenOCD。对于后者，命令行示例如下：

```
openocd -f board/esp32-wrover-kit-3.3v.cfg -c "init; halt; esp32 appimage_offset
0x210000"
```

另外还可以通过 OpenOCD 的 telnet 会话执行该命令，然后再连接 GDB，不过这种方式似乎没有那么便捷。

“next” 命令无法跳过子程序的原因

当使用 `next` 命令单步执行代码时，GDB 会在子程序的前面设置一个断点（两个中可用的一个），这样就可以跳过进入子程序内部的细节。如果这两个断点已经用在代码的其它位置，那么 `next` 命令将不起作用。在这种情况下，请删掉一个断点以使其中一个变得可用。当两个断点都已经被使用时，`next` 命令会像 `step` 命令一样工作，调试器就会进入子程序内部。

OpenOCD 支持的编译时的选项

ESP-IDF 有一些针对 OpenOCD 调试功能的选项可以在编译时进行设置：

- *CONFIG_ESP32_DEBUG_OCDAWARE* 默认会被使能。如果程序抛出了不可修复或者未处理的异常，并且此时已经连接上了 JTAG 调试器（即 OpenOCD 正在运行），那么 ESP-IDF 将会进入调试器工作模式。
- *CONFIG_FREERTOS_WATCHPOINT_END_OF_STACK* 默认没有使能。在所有任务堆栈的末尾设置观察点，从 1 号开始索引。这是调试任务堆栈溢出的最准确的方式。

更多有关设置编译时的选项的信息，请参阅项目配置菜单。

支持 FreeRTOS

OpenOCD 完全支持 ESP-IDF 自带的 FreeRTOS 操作系统，GDB 会将 FreeRTOS 中的任务当做线程。使用 GDB 命令 `i threads` 可以查看所有的线程，使用命令 `thread n` 可以切换到某个具体任务的堆栈，其中 `n` 是线程的编号。检测 FreeRTOS 的功能可以在配置目标时被禁用。更多详细信息，请参阅针对特定目标的 *OpenOCD* 配置.

在 OpenOCD 的配置文件中设置 SPI 闪存的工作电压

ESP32 的 MTDI 引脚是用于 JTAG 通信的四个引脚之一，同时也是 ESP32 的 bootstrapping 引脚。上电时，ESP32 会在 MTDI 引脚上采样二进制电平，据此来设置内部的稳压器，用于给外部的 SPI 闪存芯片供电。如果上电时 MTDI 引脚上的二进制电平为低电平，则稳压器会被设置为 3.3 V；如果 MTDI 引脚为高电平，则稳压器会被设置为 1.8 V。MTDI 引脚通常需要一个上拉电阻或者直接使能内部的弱下拉电阻（详见 ESP32 系列芯片技术规格书 ），具体取决于所使用的 SPI 芯片的类型。但是一旦连接上 JTAG 后，原来用于实现 bootstrapping 功能的上拉或者下拉电阻都会被覆盖掉。

为了解决这个问题，OpenOCD 的板级配置文件（例如 ESP32-WROOM-32 模组的 `boards\esp-wroom-32.cfg`）提供了 `ESP32_FLASH_VOLTAGE` 参数来设置 `TDO` 信号线在空闲状态下的二进制电平，这样就可以减少由于闪存电压不正确而导致的应用程序启动不良的几率。

查看 JTAG 连接的 ESP32 模组的规格书，检查其 SPI 闪存芯片的供电电压值，然后再相应的设置 `ESP32_FLASH_VOLTAGE`。大多数的 WROOM 模组使用 3.3 V 的闪存芯片，但是 WROVER 模组使用 1.8 V 的闪存芯片。

优化 JTAG 的速度

为了实现更高的数据通信速率同时最小化丢包数，建议优化 JTAG 时钟频率的设置，使其达到 JTAG 能稳定运行的最大值。为此，请参考以下建议。

1. 如果 CPU 以 80 MHz 运行，则 JTAG 时钟频率的上限为 20 MHz；如果 CPU 以 160 MHz 或者 240 MHz 运行，则上限为 26 MHz。
2. 根据特定的 JTAG 适配器和连接线缆的长度，你可能需要将 JTAG 的工作频率降低至 20 / 26 MHz 以下。

3. 在某些特殊情况下，如果你看到 DSR/DIR 错误（并且它并不是由 OpenOCD 试图从一个没有物理存储器映射的地址空间读取数据而导致的），请降低 JTAG 的工作频率。
4. ESP-WROVER-KIT 能够稳定运行在 20 / 26 MHz 频率下。

调试器的启动命令的含义

在启动时，调试器发出一系列命令来复位芯片并使其在特定的代码行停止运行。这个命令序列（如下所示）支持自定义，用户可以选择在最方便合适的代码行开始调试工作。

- `set remote hardware-watchpoint-limit 2` —限制 GDB 仅使用 ESP32 支持的两个硬件观察点。更多详细信息，请查阅 GDB 配置远程目标 。
- `mon reset halt` —复位芯片并使 CPU 停止运行。
- `flushregs` —monitor (`mon`) 命令无法通知 GDB 目标状态已经更改，GDB 会假设在 `mon reset halt` 之前所有的任务堆栈仍然有效。实际上，复位后目标状态将发生变化。执行 `flushregs` 是一种强制 GDB 从目标获取最新状态的方法。
- `thb app_main` —在 `app_main` 处插入一个临时的硬件断点，如果有需要，可以将其替换为其他函数名。
- `c` —恢复程序运行，它将会在 `app_main` 的断点处停止运行。

针对特定目标的 OpenOCD 配置

OpenOCD 需要知道当前使用的 JTAG 适配器的类型，以及其连接的目标板和处理器的类型。为此，请使用位于 OpenOCD 安装目录下 `share/openocd/scripts/interface` 和 `share/openocd/scripts/board` 文件夹中现有的配置文件。

例如，如果使用板载 ESP-WROOM-32 模组的 ESP-WROVER-KIT 开发板（详见*ESP-WROVER-KIT V1 / ESP32 DevKitJ V1*），请使用以下配置文件：

- `board/esp32-wrover-kit-3.3v.cfg`

当然也可以使用自定义的配置文件，建议在已有配置文件的基础上进行修改，以匹配你的硬件。下面列举一些常用的板级配置参数。

适配器的时钟速度

```
adapter_khz 20000
```

请参阅*优化 JTAG 的速度* 以获取有关如何设置此值的指导。

单核调试

```
set ESP32_ONLYCPU 1
```

如果是双核调试，请注释掉这一行。

禁用 RTOS 支持

```
set ESP32_RTOS none
```

如果要支持 RTOS，请注释掉这一行。

ESP32 的 SPI 闪存芯片的电源电压

```
set ESP32_FLASH_VOLTAGE 1.8
```

如果 SPI 闪存芯片的电源电压为 3.3 V，请注释掉这一行，更多信息请参阅：*在 OpenOCD 的配置文件中设置 SPI 闪存的工作电压*。

ESP32 的目标配置文件

```
source [find target/esp32.cfg]
```

注解： 除非你熟悉 OpenOCD 内部的工作原理，否则请不要更改 `source [find target/esp32.cfg]` 这一行。

目前 `target/esp32.cfg` 仍然是 ESP32 目标（esp108 和 esp32）的唯一配置文件。支持的配置矩阵如下所示：

Dual/single	RTOS	Target used
dual	FreeRTOS	esp32
single	FreeRTOS	esp108 (*)
dual	none	esp108
single	none	esp108

(*) —我们计划修复此问题，并在后续提交中添加对 esp32 目标的单核调试的支持。

更多信息，请查看 `board/esp-wroom-32.cfg` 配置文件的注释部分。

复位 ESP32

通过在 GDB 中输入 `mon reset` 或者 `mon reset halt` 来复位板子。

不要将 JTAG 引脚用于其他功能

如果除了 ESP32 模组和 JTAG 适配器之外的其他硬件也连接到了 JTAG 引脚，那么 JTAG 的操作可能会受到干扰。ESP32 JTAG 使用以下引脚：

	ESP32 JTAG Pin	JTAG Signal
1	MTDO / GPIO15	TDO
2	MTDI / GPIO12	TDI
3	MTCK / GPIO13	TCK
4	MTMS / GPIO14	TMS

如果用户应用程序更改了 JTAG 引脚的配置，JTAG 通信可能会失败。如果 OpenOCD 正确初始化（检测到两个 Tensilica 内核），但在程序运行期间失去了同步并报出大量 DTR/DIR 错误，则应用程序可能将 JTAG 引脚重新配置为其他功能或者用户忘记将 Vtar 连接到 JTAG 适配器。

下面是 GDB 在应用程序进入重新配置 MTDO/GPIO15 作为输入代码后报告的一系列错误摘录：

```
cpu0: xtensa_resume (line 431): DSR (FFFFFFFF) indicates target still busy!
cpu0: xtensa_resume (line 431): DSR (FFFFFFFF) indicates DIR instruction generated an␣
↪exception!
cpu0: xtensa_resume (line 431): DSR (FFFFFFFF) indicates DIR instruction generated an␣
↪overrun!
cpu1: xtensa_resume (line 431): DSR (FFFFFFFF) indicates target still busy!
cpu1: xtensa_resume (line 431): DSR (FFFFFFFF) indicates DIR instruction generated an␣
↪exception!
cpu1: xtensa_resume (line 431): DSR (FFFFFFFF) indicates DIR instruction generated an␣
↪overrun!
```

JTAG with Flash Encryption or Secure Boot

By default, enabling Flash Encryption and/or Secure Boot will disable JTAG debugging. On first boot, the bootloader will burn an eFuse bit to permanently disable JTAG at the same time it enables the other features.

The project configuration option *CONFIG_SECURE_BOOT_ALLOW_JTAG* will keep JTAG enabled at this time, removing all physical security but allowing debugging. (Although the name suggests Secure Boot, this option can be applied even when only Flash Encryption is enabled).

However, OpenOCD may attempt to automatically read and write the flash in order to set *software breakpoints*. This has two problems:

- Software breakpoints are incompatible with Flash Encryption, OpenOCD currently has no support for encrypting or decrypting flash contents.
- If Secure Boot is enabled, setting a software breakpoint will change the digest of a signed app and make the signature invalid. This means if a software breakpoint is set and then a reset occurs, the signature verification will fail on boot.

To disable software breakpoints while using JTAG, add an extra argument `-c 'set ESP_FLASH_SIZE 0'` to the start of the OpenOCD command line. For example:

```
openocd -c 'set ESP_FLASH_SIZE 0' -f board/esp32-wrover-kit-3.3v.cfg
```

注解: For the same reason, the ESP-IDF app may fail bootloader verification of app signatures, when this option is enabled and a software breakpoint is set.

报告 OpenOCD / GDB 的问题

如果你遇到 OpenOCD 或者 GDB 程序本身的问题，并且在网上没有找到可用的解决方案，请前往 https://github.com/espressif/openocd-esp32/issues 新建一个议题。

1. 请在问题报告中提供你使用的配置的详细信息：
   a. JTAG 适配器类型。
   b. 用于编译和加载正在调试的应用程序的 ESP-IDF 版本号。
   c. 用于调试的操作系统的详细信息。
   d. 操作系统是在本地计算机运行还是在虚拟机上运行?
2. 创建一个能够演示问题的简单示例工程，描述复现该问题的步骤。且这个调试示例不能受到 Wi-Fi 协议栈引入的非确定性行为的影响，因而再次遇到同样问题时，更容易复现。
3. 在启动命令中添加额外的参数来输出调试日志。

   OpenOCD 端：

   ```
   openocd -l openocd_log.txt -d 3 -f board/esp32-wrover-kit-3.3v.cfg
   ```

   这种方式会将日志输出到文件，但是它会阻止调试信息打印在终端上。当有大量信息需要输出的时候（比如调试等级提高到 `-d 3`）这是个不错的选择。如果你仍然希望在屏幕上看到调试日志，请改用以下命令：

```
openocd -d 3 -f board/esp32-wrover-kit-3.3v.cfg 2>&1 | tee openocd.log
```

Debugger 端：

```
xtensa-esp32-elf-gdb -ex "set remotelogfile gdb_log.txt" <all other options>
```

也可以将命令 `remotelogfile gdb_log.txt` 添加到 `gdbinit` 文件中。

4. 请将 `openocd_log.txt` 和 `gdb_log.txt` 文件附在你的问题报告中。

应用层跟踪库

[English]

概述

为了分析应用程序的行为，IDF 提供了一个有用的功能：应用层跟踪。这个功能以库的形式提供，可以通过 menuconfig 开启。此功能使得用户可以在程序运行开销很小的前提下，通过 JTAG 接口在主机和 ESP32 之间传输任意数据。

开发人员可以使用这个功能库将应用程序的运行状态发送给主机，在运行时接收来自主机的命令或者其他类型的信息。该库的主要使用场景有：

1. 收集应用程序特定的数据，具体请参阅*特定应用程序的跟踪*
2. 轻量级的日志记录，具体请参阅*记录日志到主机*
3. 系统行为分析，具体请参阅*基于 SEGGER SystemView 的系统行为分析*

使用 JTAG 接口的跟踪组件工作示意图：

运行模式

该库支持两种操作模式：

后验模式：这是默认的模式，该模式不需要和主机进行交互。在这种模式下，跟踪模块不会检查主机是否已经从 *HW UP BUFFER* 缓冲区读走所有数据，而是直接使用新数据覆盖旧数据。该模式在用户仅对最新的跟踪数据感兴趣时会很有用，例如分析程序在崩溃之前的行为。主机可以稍后根据用户的请求来读取数据，例如通过特殊的 OpenOCD 命令（假如使用了 JTAG 接口）。

流模式：当主机连接到 ESP32 时，跟踪模块会进入此模式。在这种模式下，跟踪模块在新数据写入 *HW UP BUFFER* 之前会检查其中是否有足够的空间，并在必要的时候等待主机读取数据并释放足够的内存。用户会将最长的等待时间作为超时时间参数传递给相应的 API 函数，如果超时时间是个有限值，那么应用程序有可能会因为超时而将待写的数据丢弃。尤其需要注意，如果在讲究时效的代码中（如中断处理函数，操作系统调度等）指定了无限的超时时间，那么系统会产生故障。为了避免丢失此类关键数据，开发人员可以通过在 menuconfig 中开启*CONFIG_APPTRACE_PENDING_DATA_SIZE_MAX* 选项来启用额外的数据

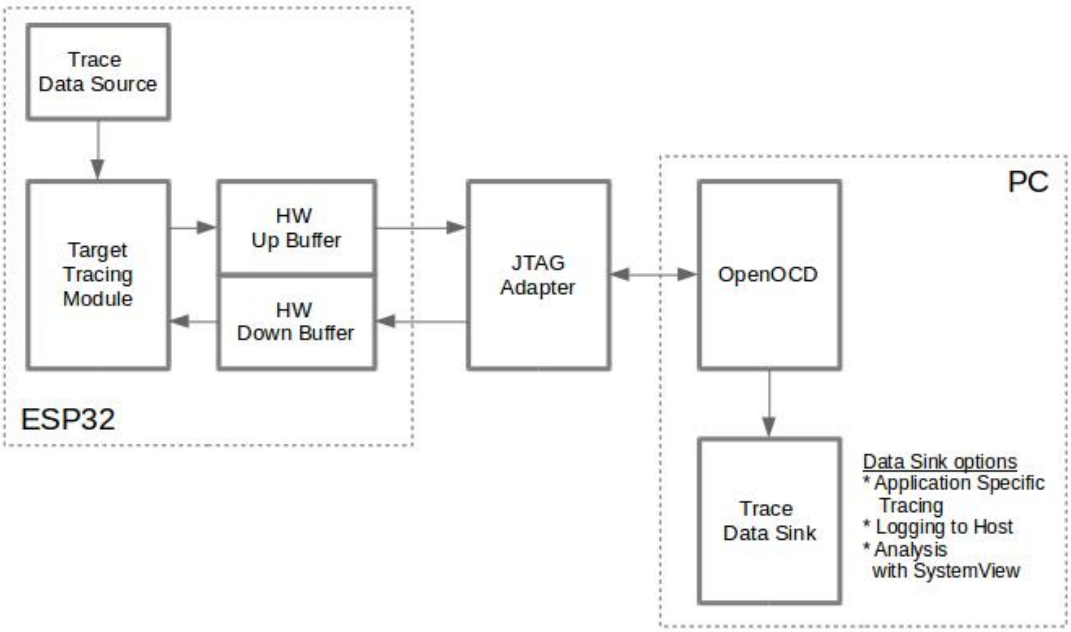


图 23: 使用 JTAG 接口的跟踪组件

缓冲区。此宏还指定了在上述条件下可以缓冲的数据大小，它有助于缓解由于 USB 总线拥塞等原因导致的向主机传输数据间歇性减缓的状况。但是，当跟踪数据流的平均比特率超过硬件接口的能力时，它也无能为力。

配置选项与依赖项

使用此功能需要在主机端和目标端做相应的配置：

1. **主机端：** 应用程序跟踪是通过 JTAG 来完成的，因此需要在主机上安装并运行 OpenOCD。相关详细信息请参阅*JTAG Debugging*。
2. **目标端：** 在 menuconfig 中开启应用程序跟踪功能。*Component config* > *Application Level Tracing* 菜单允许选择跟踪数据的传输目标（具体用于传输的硬件接口），选择任一非 None 的目标都会自动开启 `CONFIG_APPTRACE_ENABLE` 这个选项。

注解： 为了实现更高的数据速率并降低丢包率，建议优化 JTAG 的时钟频率，使其达到能够稳定运行的最大值。详细信息请参阅*优化 JTAG 的速度*。

以下为前述未提及的另外两个 menuconfig 选项：

1. *Threshold for flushing last trace data to host on panic* (*CONFIG_APPTRACE_POSTMORTEM_FLUSH_THRESH*) 。由于在 JTAG 上工作的性质，此选项是必选项。在该模式下，跟踪数据以 16 KB 数据块的形式曝露给主机。在后验模式中，当一个块被填充时，它会曝露给主机，而之前的块会变得不可用。换句话说，跟踪数据以 16 KB 的粒度进行覆盖。在发生 panic 的时候，当前输入块的最新数据将会被曝露给主机，主机可以读取它们以进行后续分析。如果系统发生 panic 的时候仍有少量数据还没来得及曝光给主机，那么之前收集的 16 KB 的数据将丢失，主机只能看到非常少的最新的跟踪部分，它可能不足以用来诊断问题所在。此 menuconfig 选项允许避免此类情况，它可以控制在发生 panic 时刷新数据的阈值，例如用户可以确定它需要不少于 512 字节的最新跟踪数据，所以如果在发生 panic 时待处理的数据少于 512 字节，它们不会被刷新，也不会覆盖之前的 16 KB。该选项仅在后验模式和 JTAG 工作时有意义。
2. *Timeout for flushing last trace data to host on panic* (*CONFIG_APPTRACE_ONPANIC_HOST_FLUSH_TMO*) 。该选项仅在流模式下才起作用，它控制跟踪模块在发生 panic 时等待主机读取最新数据的最长时间。

如何使用这个库

该库提供了用于在主机和 ESP32 之间传输任意数据的 API。当在 menuconfig 中启用时，目标应用程序的跟踪模块会在系统启动时自动初始化，因此用户需要做的就是调用相应的 API 来发送、接收或者刷新数据。

特定应用程序的跟踪

通常，用户需要决定在每个方向上待传输数据的类型以及如何解析（处理）这些数据。要想在目标和主机之间传输数据，用户必须要执行以下几个步骤。

1. 在目标端，用户需要实现将跟踪数据写入主机的算法，下面的代码片段展示了如何执行此操作。

```
#include "esp_app_trace.h"
...
char buf[] = "Hello World!";
esp_err_t res = esp_apptrace_write(ESP_APPTRACE_DEST_TRAX, buf, strlen(buf), ESP_
↪APPTRACE_TMO_INFINITE);
if (res != ESP_OK) {
    ESP_LOGE(TAG, "Failed to write data to host!");
    return res;
}
```

`esp_apptrace_write()` 函数使用 memcpy 把用户数据复制到内部缓存中。在某些情况下，使用 `esp_apptrace_buffer_get()` 和 `esp_apptrace_buffer_put()` 函数会更加理想，它们允许开发人员自行分配缓冲区并填充。下面的代码片段展示了如何执行此操作。

```
#include "esp_app_trace.h"
...
int number = 10;
char *ptr = (char *)esp_apptrace_buffer_get(ESP_APPTRACE_DEST_TRAX, 32, 100/*tmo in␣
↪us*/);
if (ptr == NULL) {
    ESP_LOGE(TAG, "Failed to get buffer!");
    return ESP_FAIL;
}
sprintf(ptr, "Here is the number %d", number);
esp_err_t res = esp_apptrace_buffer_put(ESP_APPTRACE_DEST_TRAX, ptr, 100/*tmo in␣
↪us*/);
if (res != ESP_OK) {
    /* in case of error host tracing tool (e.g. OpenOCD) will report incomplete␣
↪user buffer */
    ESP_LOGE(TAG, "Failed to put buffer!");
    return res;
}
```

另外，根据实际项目的需要，用户可能希望从主机接收数据。下面的代码片段展示了如何执行此操作。

```
#include "esp_app_trace.h"
...
char buf[32];
char down_buf[32];
size_t sz = sizeof(buf);

/* config down buffer */
esp_apptrace_down_buffer_config(down_buf, sizeof(down_buf));
/* check for incoming data and read them if any */
esp_err_t res = esp_apptrace_read(ESP_APPTRACE_DEST_TRAX, buf, &sz, 0/*do not wait*/
↪);
if (res != ESP_OK) {
    ESP_LOGE(TAG, "Failed to read data from host!");
    return res;
}
if (sz > 0) {
    /* we have data, process them */
    ...
}
```

esp_apptrace_read() 函数使用 memcpy 来把主机端的数据复制到用户缓存区。在某些情况下，使用

esp_apptrace_down_buffer_get() 和 esp_apptrace_down_buffer_put() 函数可能更为理想。它们允许开发人员占用一块读缓冲区并就地进行有关处理操作。下面的代码片段展示了如何执行此操作。

```
#include "esp_app_trace.h"
...
char down_buf[32];
uint32_t *number;
size_t sz = 32;

/* config down buffer */
esp_apptrace_down_buffer_config(down_buf, sizeof(down_buf));
char *ptr = (char *)esp_apptrace_down_buffer_get(ESP_APPTRACE_DEST_TRAX, &sz, 100/
↪*tmo in us*/);
if (ptr == NULL) {
    ESP_LOGE(TAG, "Failed to get buffer!");
    return ESP_FAIL;
}
if (sz > 4) {
    number = (uint32_t *)ptr;
    printf("Here is the number %d", *number);
} else {
    printf("No data");
}
esp_err_t res = esp_apptrace_down_buffer_put(ESP_APPTRACE_DEST_TRAX, ptr, 100/*tmo␣
↪in us*/);
if (res != ESP_OK) {
    /* in case of error host tracing tool (e.g. OpenOCD) will report incomplete␣
↪user buffer */
    ESP_LOGE(TAG, "Failed to put buffer!");
    return res;
}
```

2. 下一步是编译应用程序的镜像并将其下载到目标板上，这一步可以参考文档构建并烧写。
3. 运行 OpenOCD（参见*JTAG 调试*）。
4. 连接到 OpenOCD 的 telnet 服务器，在终端执行如下命令 `telnet <oocd_host> 4444`。如果在运行 OpenOCD 的同一台机器上打开 telnet 会话，您可以使用 `localhost` 替换上面命令中的 `<oocd_host>`。
5. 使用特殊的 OpenOCD 命令开始收集待跟踪的命令，此命令将传输跟踪数据并将其重定向到指定的文件或套接字（当前仅支持文件作为跟踪数据目标）。相关命令的说明请参阅启动调试器 。
6. 最后一步是处理接收到的数据，由于数据格式由用户定义，因此处理阶段超出了本文档的范围。数据处理的范例可以参考位于 `$IDF_PATH/tools/esp_app_trace` 下的 Python 脚本 `apptrace_proc.py`（用

于功能测试）和 `logtrace_proc.py` （请参阅记录日志到主机 章节中的详细信息）。

OpenOCD 应用程序跟踪命令

HW UP BUFFER 在用户数据块之间共享，并且会替 API 的调用者（在任务或者中断上下文中）填充分配到的内存。在多线程环境中，正在填充缓冲区的任务/中断可能会被另一个高优先级的任务/中断抢占，有可能发生主机读取还未准备好的用户数据的情况。为了处理这样的情况，跟踪模块在所有用户数据块之前添加一个数据头，其中包含有分配的用户缓冲区的大小（2 字节）和实际写入的数据长度（2 字节），也就是说数据头总共长 4 字节。负责读取跟踪数据的 OpenOCD 命令在读取到不完整的用户数据块时会报错，但是无论如何它都会将整个用户数据块（包括还未填充的区域）的内容放到输出文件中。

下面是 OpenOCD 应用程序跟踪命令的使用说明。

注解: 目前，OpenOCD 还不支持将任意用户数据发送到目标的命令。

命令用法：

```
esp32 apptrace [start <options>] | [stop] | [status] | [dump <cores_num> <outfile>]
```

子命令：

start 开始跟踪（连续流模式）。

stop 停止跟踪。

status 获取跟踪状态。

dump 转储所有后验模式的数据。

Start 子命令的语法：

```
start <outfile> [poll_period [trace_size [stop_tmo [wait4halt [skip_size]]]]
```

outfile 用于保存来自两个 CPU 的数据文件的路径，该参数需要具有以下格式：`file://path/to/file`。

poll_period 轮询跟踪数据的周期（单位：毫秒），如果大于 0 则以非阻塞模式运行。默认为 1 毫秒。

trace_size 最多要收集的数据量（单位：字节），接收到指定数量的数据后将会停止跟踪。默认情况下是 -1（禁用跟踪大小停止触发器）。

stop_tmo 空闲超时（单位：秒），如果指定的时间段内都没有数据就会停止跟踪。默认为 -1（禁用跟踪超时停止触发器）。还可以将其设置为比目标跟踪命令之间的最长暂停值更长的值（可选）。

wait4halt 如果设置为 0 则立即开始跟踪，否则命令等待目标停止（复位，打断点等），然后自动恢复它并开始跟踪。默认值为 0。

skip_size 开始时要跳过的字节数，默认为 0。

注解: 如果 `poll_period` 为 0，则在跟踪停止之前，OpenOCD 的 telnet 命令将不可用。必须通过复位电路板或者在 OpenOCD 的窗口中（不是 telnet 会话窗口）按下 Ctrl+C。另一种选择是设置 `trace_size` 并等待，当收集到指定数据量时，跟踪会自动停止。

命令使用示例：

1. 将 2048 个字节的跟踪数据收集到 “trace.log” 文件中，该文件将保存在 “openocd-esp32” 目录中。

```
esp32 apptrace start file://trace.log 1 2048 5 0 0
```

   跟踪数据会被检索并以非阻塞的模式保存到文件中，如果收集满 2048 字节的数据或者在 5 秒内都没有新的数据，那么该过程就会停止。

   注解: 在将数据提供给 OpenOCD 之前，会对其进行缓冲。如果看到 “Data timeout!” 的消息，则目标可能在超时之前没有发送足够的数据给 OpenOCD 来清空缓冲区。增加超时时间或者使用函数 `esp_apptrace_flush()` 以特定间隔刷新数据都可以解决这个问题。

2. 在非阻塞模式下无限地检索跟踪数据。

```
esp32 apptrace start file://trace.log 1 -1 -1 0 0
```

   对收集数据的大小没有限制，并且没有设置任何超时时间。可以通过在 OpenOCD 的 telnet 会话窗口中发送 `esp32 apptrace stop` 命令，或者在 OpenOCD 窗口中使用快捷键 Ctrl+C 来停止此过程。

3. 检索跟踪数据并无限期保存。

```
esp32 apptrace start file://trace.log 0 -1 -1 0 0
```

   在跟踪停止之前，OpenOCD 的 telnet 会话窗口将不可用。要停止跟踪，请在 OpenOCD 的窗口中使用快捷键 Ctrl+C。

4. 等待目标停止，然后恢复目标的操作并开始检索数据。当收集满 2048 字节的数据后就停止：

```
esp32 apptrace start file://trace.log 0 2048 -1 1 0
```

   想要复位后立即开始跟踪，请使用 OpenOCD 的 `reset halt` 命令。

记录日志到主机

记录日志到主机是 IDF 的一个非常实用的功能：通过应用层跟踪库将日志保存到主机端。某种程度上这也算是一种半主机（semihosting）机制，相较于调用 `ESP_LOGx` 将待打印的字符串发送到 UART 的日志记录方式，这个功能的优势在于它减少了本地的工作量，而将大部分工作转移到了主机端。

IDF 的日志库会默认使用类 vprintf 的函数将格式化的字符串输出到专用的 UART。一般来说，它涉及到以下几个步骤：

1. 解析格式字符串以获取每个参数的类型。
2. 根据其类型，将每个参数都转换为字符串。
3. 格式字符串与转换后的参数一起发送到 UART。

虽然可以将类 vprintf 函数优化到一定程度，但是上述步骤在任何情况下都是必须要执行的，并且每个步骤都会消耗一定的时间（尤其是步骤 3）。所以经常会发生以下这种情况：向程序中添加额外的打印信息以诊断问题，却改变了应用程序的行为，使得问题无法复现。在最差的情况下，程序会无法正常工作，最终导致报错甚至挂起。

解决此类问题的可能方法是使用更高的波特率或者其他更快的接口，并将字符串格式化的工作转移到主机端。

通过应用层跟踪库的 `esp_apptrace_vprintf` 函数，可以将日志信息发送到主机，该函数不执行格式字符串和参数的完全解析，而仅仅计算传递的参数的数量，并将它们与格式字符串地址一起发送给主机。主机端会通过一个特殊的 Python 脚本来处理并打印接收到的日志数据。

局限

目前通过 JTAG 实现记录日志还存在以下几点局限：

1. 不支持使用 `ESP_EARLY_LOGx` 宏进行跟踪。
2. 不支持大小超过 4 字节的 printf 参数（例如 `double` 和 `uint64_t`）。
3. 仅支持.rodata 段中的格式字符串和参数。
4. printf 参数最多 256 个。

如何使用

为了使用跟踪模块来记录日志，用户需要执行以下步骤：

1. 在目标端，需要安装特殊的类 vprintf 函数，正如前面提到过的，这个函数是 `esp_apptrace_vprintf`，它会负责将日志数据发送给主机。示例代码参见 system/app_trace_to_host 。
2. 按照特定应用程序的跟踪 章节中第 2-5 步骤中的说明进行操作。
3. 打印接收到的日志记录，请在终端运行以下命令：`$IDF_PATH/tools/esp_app_trace/logtrace_proc.py /path/to/trace/file /path/to/program/elf/file`。

Log Trace Processor 命令选项

命令用法：

```
logtrace_proc.py [-h] [--no-errors] <trace_file> <elf_file>
```

位置参数（必要）：

trace_file 日志跟踪文件的路径

elf_file 程序 ELF 文件的路径

可选参数：

-h, --help 显示此帮助信息并退出

--no-errors, -n 不打印错误信息

基于 SEGGER SystemView 的系统行为分析

IDF 中另一个基于应用层跟踪库的实用功能是系统级跟踪，它会生成与 SEGGER SystemView 工具 相兼容的跟踪信息。SEGGER SystemView 是一种实时记录和可视化工具，用来分析应用程序运行时的行为。

注解： 目前，基于 IDF 的应用程序能够以文件的形式生成与 SystemView 格式兼容的跟踪信息，并可以使用 SystemView 工具软件打开。但是还无法使用该工具控制跟踪的过程。

如何使用

若需使用这个功能，需要在 menuconfig 中开启*CONFIG_SYSVIEW_ENABLE* 选项，具体路径为：*Component config* > *Application Level Tracing* > *FreeRTOS SystemView Tracing* 。在同一个菜单栏下还开启了其他几个选项：

1. *ESP32 timer to use as SystemView timestamp source* （*CONFIG_SYSVIEW_TS_SOURCE*）选择 SystemView 事件使用的时间戳来源。在单核模式下，使用 ESP32 内部的循环计数器生成时间戳，其最大的工作频率是 240 MHz（时间戳粒度大约为 4 ns）。在双核模式下，使用工作在 40 MHz 的外部定时器，因此时间戳粒度为 25 ns。
2. 可以单独启用或禁用的 SystemView 事件集合（`CONFIG_SYSVIEW_EVT_XXX`）：
   - Trace Buffer Overflow Event
   - ISR Enter Event
   - ISR Exit Event
   - ISR Exit to Scheduler Event
   - Task Start Execution Event
   - Task Stop Execution Event
   - Task Start Ready State Event

- Task Stop Ready State Event
- Task Create Event
- Task Terminate Event
- System Idle Event
- Timer Enter Event
- Timer Exit Event

IDF 中已经包含了所有用于生成兼容 SystemView 跟踪信息的代码，用户只需配置必要的项目选项（如上所示），然后构建、烧写映像到目标板，接着参照前面的介绍，使用 OpenOCD 收集数据。

OpenOCD SystemView 跟踪命令选项

命令用法：

```
esp32 sysview [start <options>] | [stop] | [status]
```

自命令：

start 开启跟踪（连续流模式）。

stop 停止跟踪。

status 获取跟踪状态。

Start 子命令语法：

```
start <outfile1> [outfile2] [poll_period [trace_size [stop_tmo]]]
```

outfile1 保存 PRO CPU 数据的文件路径，此参数需要具有如下格式：`file://path/to/file`。

outfile2 保存 APP CPU 数据的文件路径，此参数需要具有如下格式：`file://path/to/file`。

poll_period 跟踪数据的轮询周期（单位：毫秒）。如果该值大于 0，则命令以非阻塞的模式运行。默认为 1 毫秒。

trace_size 最多要收集的数据量（单位：字节）。当收到指定数量的数据后，将停止跟踪。默认值是 -1（禁用跟踪大小停止触发器）。

stop_tmo 空闲超时（单位：秒）。如果指定的时间内没有数据，将停止跟踪。默认值是 -1（禁用跟踪超时停止触发器）。

注解： 如果 `poll_period` 为 0，则在跟踪停止之前，OpenOCD 的 telnet 命令行将不可用。你需要通过复位板卡或者在 OpenOCD 的窗口（不是 telnet 会话窗口）输入 Ctrl+C 命令来手动停止它。另一个办法是设置 `trace_size` 然后等到收集满指定数量的数据后自动停止跟踪。

命令使用示例：

1. 将 SystemView 跟踪数据收集到文件 “pro-cpu.SVDat” 和 “pro-cpu.SVDat” 中。这些文件会被保存在 “openocd-esp32” 目录中。

```
esp32 sysview start file://pro-cpu.SVDat file://app-cpu.SVDat
```

   跟踪数据被检索并以非阻塞的方式保存，要停止此过程，需要在 OpenOCD 的 telnet 会话窗口输入 `esp32 apptrace stop` 命令，或者也可以在 OpenOCD 窗口中按下 Ctrl+C。

2. 检索跟踪数据并无限保存。

```
esp32 sysview start file://pro-cpu.SVDat file://app-cpu.SVDat 0 -1 -1
```

   OpenOCD 的 telnet 命令行在跟踪停止前会无法使用，要停止跟踪，请在 OpenOCD 窗口按下 Ctrl+C。

数据可视化

收集到跟踪数据后，用户可以使用特殊的工具来可视化结果并分析程序的行为。遗憾的是，SystemView 不支持从多个核心进行跟踪。所以当追踪双核模式下的 ESP32 时会生成两个文件：一个用于 PRO CPU，另一个用于 APP CPU。用户可以将每个文件加载到工具中单独分析。

在工具中单独分析每个核的跟踪数据是比较棘手的，幸运的是，Eclipse 中有一款 *Impulse* 的插件可以加载多个跟踪文件，并且可以在同一个视图中检查来自两个内核的事件。此外，与免费版的 SystemView 相比，此插件没有 1,000,000 个事件的限制。

关于如何安装、配置 Impulse 并使用它可视化来自单个核心的跟踪数据，请参阅 官方教程 。

注解： IDF 使用自己的 SystemView FreeRTOS 事件 ID 映射，因此用户需要将 `$SYSVIEW_INSTALL_DIR/Description/SYSVIEW_FreeRTOS.txt` 替换成 `$IDF_PATH/docs/api-guides/SYSVIEW_FreeRTOS.txt`。在使用上述链接配置 SystemView 序列化程序时，也应该使用该 IDF 特定文件的内容。

配置 Impulse 实现双核跟踪

在安装好 Impulse 插件后，先确保它能够在单独的选项卡中成功加载每个核心的跟踪文件，然后用户可以添加特殊的 Multi Adapter 端口并将这两个文件加载到一个视图中。为此，用户需要在 Eclipse 中执行以下操作：

1. 打开 “Signal Ports” 视图，前往 Windows->Show View->Other 菜单，在 Impulse 文件夹中找到 “Signal Ports” 视图，然后双击它。
2. 在 “Signal Ports” 视图中，右键单击 “Ports” 并选择 “Add …”，然后选择 New Multi Adapter Port。
3. 在打开的对话框中按下 “Add” 按钮，选择 “New Pipe/File”。
4. 在打开的对话框中选择 “SystemView Serializer” 并设置 PRO CPU 跟踪文件的路径，按下确定保存设置。

5. 对 APP CPU 的跟踪文件重复步骤 3 和 4。
6. 双击创建的端口，会打开此端口的视图。
7. 单击 Start/Stop Streaming 按钮，数据将会被加载。
8. 使用 “Zoom Out”，“Zoom In” 和 “Zoom Fit” 按钮来查看数据。
9. 有关设置测量光标和其他的功能，请参阅 Impulse 官方文档 。

注解： 如果您在可视化方面遇到了问题（未显示数据或者缩放操作很奇怪），您可以尝试删除当前的信号层次结构，再双击必要的文件或端口。Eclipse 会请求您创建新的信号层次结构。

5.14 引导加载程序（Bootloader）

[English]

引导加载程序（Bootloader）主要执行以下任务：

1. 内部模块的最小化初始配置；
2. 根据分区表和 ota_data（如果存在）选择需要引导的应用程序（app）分区；
3. 将此应用程序映像加载到 RAM（IRAM 和 DRAM）中，最后把控制权转交给应用程序。

引导加载程序位于 Flash 的 *0x1000* 偏移地址处。

5.14.1 恢复出厂设置

用户可以编写一个基本的工作固件，然后将其加载到工厂分区（factory）中。

接下来，通过 OTA（空中升级）更新固件，更新后的固件会被保存到某个 OTA app 分区中，OTA 数据分区也会做相应更新以指示从该分区引导应用程序。

如果你希望回滚到出厂固件并清除设置，则需要设置*CONFIG_BOOTLOADER_FACTORY_RESET*。

出厂重置机制允许将设备重置为出厂模式：

- 清除一个或多个数据分区。
- 从工厂分区启动。

CONFIG_BOOTLOADER_DATA_FACTORY_RESET - 允许用户选择在恢复出厂设置时需要删除的数据分区。可以通过逗号来分隔多个分区的名字，并适当增加空格以便阅读（例如 “nvs, phy_init, nvs_custom, …”）。请确保此处指定的名称和分区表中的名称相同，且不含有 “app” 类型的分区。

CONFIG_BOOTLOADER_OTA_DATA_ERASE - 恢复出厂模式后，设备会从工厂分区启动，OTA 数据分区会被清除。

CONFIG_BOOTLOADER_NUM_PIN_FACTORY_RESET- 设置用于触发出厂重置的 GPIO 编号，必须在芯片复位时将此 GPIO 拉低才能触发出厂重置事件。

CONFIG_BOOTLOADER_HOLD_TIME_GPIO- 设置进入重置或测试模式所需要的保持时间（默认为 5 秒）。设备复位后，GPIO 必须在这段时间内持续保持低电平，然后才会执行出厂重置或引导测试分区。

示例分区表如下:

```
# Name,   Type, SubType, Offset,   Size, Flags
# 注意，如果你增大了引导加载程序的大小，请确保更新偏移量，避免和其它分区发生重叠
nvs,      data, nvs,     0x9000,   0x4000
otadata,  data, ota,     0xd000,   0x2000
phy_init, data, phy,     0xf000,   0x1000
factory,  0,    0,       0x10000,  1M
test,     0,    test,    ,         512K
ota_0,    0,    ota_0,   ,         512K
ota_1,    0,    ota_1,   ,         512K
```

5.14.2 从测试固件启动

用户可以编写在生产环境中测试用的特殊固件，然后在需要的时候运行。此时需要在分区表中专门申请一块分区用于保存该测试固件（详情请参阅分区表） 。如果想要触发测试固件，还需要设置*CONFIG_BOOTLOADER_APP_TEST*。

CONFIG_BOOTLOADER_NUM_PIN_APP_TEST - 设置引导测试分区的 GPIO 管脚编号，该 GPIO 会被配置为输入模式，并且会使能内部上拉电阻。若想触发测试固件，该 GPIO 必须在芯片复位时拉低。设备重启时如果该 GPIO 没有被激活（即处于高电平状态），那么会加载常规配置的应用程序（可能位于工厂分区或者 OTA 分区）。

CONFIG_BOOTLOADER_HOLD_TIME_GPIO - 设置进入重置或测试模式所需要的保持时间（默认为 5 秒）。设备复位后，GPIO 必须在这段时间内持续保持低电平，然后才会执行出厂重置或引导测试分区。

5.14.3 自定义引导程序

用户可以重写当前的引导加载程序，具体做法是，复制 */esp-idf/components/bootloader* 文件夹到项目目录中，然后编辑 */your_project/components/bootloader/subproject/ain/bootloader_main.c* 文件。

在引导加载程序的代码中，用户不可以使用驱动和其他组件提供的函数，如果确实需要，请将该功能的实现部分放在 bootloader 目录中（注意，这会增加引导程序的大小）。监视生成的引导程序的大小是有必要的，因为它可能会与内存中的分区表发生重叠而损坏固件。目前，引导程序被限制在了分区表之前的区域（分区表位于 *0x8000* 地址处）。

5.15 分区表

[English]

5.15.1 概述

每片 ESP32 的 flash 可以包含多个应用程序，以及多种不同类型的数据（例如校准数据、文件系统数据、参数存储器数据等）。因此，我们需要引入分区表的概念。

具体来说，ESP32 在 flash 的默认偏移地址 0x8000 处烧写一张分区表。该分区表的长度为 0xC00 字节（最多可以保存 95 条分区表条目）。分区表数据后还保存着该表的 MD5 校验和，用于验证分区表的完整性。此外，如果芯片使能了安全启动 功能，则该分区表后还会保存签名信息。

分区表中的每个条目都包括以下几个部分：Name（标签）、Type（app、data 等）、SubType 以及在 flash 中的偏移量（分区的加载地址）。

在使用分区表时，最简单的方法就是打开项目配置菜单（`idf.py menuconfig`），并在*CONFIG_PARTITION_TABLE_TYPE* 下选择一张预定义的分区表：

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

在以上两种选项中，出厂应用程序均将被烧录至 flash 的 0x10000 偏移地址处。这时，运行 *idf.py partition_table*，即可以打印当前使用分区表的信息摘要。

5.15.2 内置分区表

以下是 “Single factory app, no OTA” 选项的分区表信息摘要：

> # Espressif ESP32 Partition Table # Name, Type, SubType, Offset, Size, Flags nvs, data, nvs, 0x9000, 0x6000, phy_init, data, phy, 0xf000, 0x1000, factory, app, factory, 0x10000, 1M,

- flash 的 0x10000 (64KB) 偏移地址处存放一个标记为 “factory” 的二进制应用程序，且 Bootloader 将默认加载这个应用程序。
- 分区表中还定义了两个数据区域，分别用于存储 NVS 库专用分区和 PHY 初始化数据。

以下是 “Factory app, two OTA definitions” 选项的分区表信息摘要：

> # Espressif ESP32 Partition Table # Name, Type, SubType, Offset, Size, Flags nvs, data, nvs, 0x9000, 0x4000, otadata, data, ota, 0xd000, 0x2000, phy_init, data, phy, 0xf000, 0x1000, factory, app, factory, 0x10000, 1M, ota_0, app, ota_0, 0x110000, 1M, ota_1, app, ota_1, 0x210000, 1M,

- 分区表中定义了三个应用程序分区，这三个分区的类型都被设置为 “app”，但具体 app 类型不同。其中，位于 0x10000 偏移地址处的为出厂应用程序（factory），其余两个为 OTA 应用程序（ota_0，ota_1）。

- 新增了一个名为“otadata”的数据分区，用于保存 OTA 升级时候需要的数据。Bootloader 会查询该分区的数据，以判断该从哪个 OTA 应用程序分区加载程序。如果“otadata”分区为空，则会执行出厂程序。

5.15.3 创建自定义分区表

如果在 `menuconfig` 中选择了“Custom partition table CSV”，则还需要输入该分区表的 CSV 文件在项目中的路径。CSV 文件可以根据需要，描述任意数量的分区信息。

CSV 文件的格式与上面摘要中打印的格式相同，但是在 CSV 文件中并非所有字段都是必需的。例如下面是一个自定义的 OTA 分区表的 CSV 文件：

```
# Name, Type, SubType, Offset, Size, Flags nvs, data, nvs, 0x9000, 0x4000 otadata, data, ota,
0xd000, 0x2000 phy_init, data, phy, 0xf000, 0x1000 factory, app, factory, 0x10000, 1M ota_0,
app, ota_0, , 1M ota_1, app, ota_1, , 1M nvs_key, data, nvs_keys, , 0x1000
```

- 字段之间的空格会被忽略，任何以 `#` 开头的行（注释）也会被忽略。
- CSV 文件中的每个非注释行均为一个分区定义。
- 每个分区的 `Offset` 字段可以为空，`gen_esp32part.py` 工具会从分区表位置的后面开始自动计算并填充该分区的偏移地址，同时确保每个分区的偏移地址正确对齐。

Name 字段

Name 字段可以是任何有意义的名称，但不能超过 16 个字符（之后的内容将被截断）。该字段对 ESP32 并不是特别重要。

Type 字段

Type 字段可以指定为 app (0) 或者 data (1)，也可以直接使用数字 0-254（或者十六进制 0x00-0xFE）。注意，0x00-0x3F 不得使用（预留给 esp-idf 的核心功能）。

如果您的应用程序需要保存数据，请在 0x40-0xFE 内添加一个自定义分区类型。

注意，bootloader 将忽略 app (0) 和 data (1) 以外的其他分区类型。

SubType 字段

SubType 字段长度为 8 bit，内容与具体 Type 有关。目前，esp-idf 仅仅规定了“app”和“data”两种子类型。

- 当 Type 定义为 `app` 时，SubType 字段可以指定为 factory (0)，ota_0 (0x10) ···ota_15 (0x1F) 或者 test (0x20)。

  - factory (0) 是默认的 app 分区。Bootloader 将默认加在该应用程序。但如果存在类型为 data/ota 分区，则 Bootloader 将加载 data/ota 分区中的数据，进而判断启动哪个 OTA 镜像文件。- OTA 升级永远都不会更新 factory 分区中的内容。- 如果您希望在 OTA 项目中预留更多 flash，可以删除 factory 分区，转而使用 ota_0 分区。
  - ota_0 (0x10) ···ota_15 (0x1F) 为 OTA 应用程序分区，Bootloader 将根据 OTA 数据分区中的数据来决定加载哪个 OTA 应用程序分区中的程序。在使用 OTA 功能时，应用程序应至少拥有 2 个 OTA 应用程序分区（ota_0 和 ota_1）。更多详细信息，请参考*OTA 文档* 。
  - test (0x2) 为预留 app 子类型，用于工厂测试过程。注意，目前，esp-idf 并不支持这种子类型。

- 当 Type 定义为 `data` 时，SubType 字段可以指定为 ota (0)，phy (1)，nvs (2) 或者 nvs_keys (4)。
  - ota (0) 即*OTA 数据分区* ，用于存储当前所选的 OTA 应用程序的信息。这个分区的大小需要设定为 0x2000。更多详细信息，请参考*OTA 文档* 。
  - phy (1) 分区用于存放 PHY 初始化数据，从而保证可以为每个设备单独配置 PHY，而非必须采用固件中的统一 PHY 初始化数据。
    * 默认配置下，phy 分区并不启用，而是直接将 phy 初始化数据编译至应用程序中，从而节省分区表空间（直接将此分区删掉）。
    * 如果需要从此分区加载 phy 初始化数据，请打开项目配置菜单（`idf.py menuconfig`），并且使能*CONFIG_ESP32_PHY_INIT_DATA_IN_PARTITION* 选项。此时，您还需要手动将 phy 初始化数据烧至设备 flash（esp-idf 编译系统并不会自动完成该操作）。
  - nvs (2) 是专门给*非易失性存储 (NVS) API* 使用的分区。
    * 用于存储每台设备的 PHY 校准数据（注意，并不是 PHY 初始化数据）。
    * 用于存储 Wi-Fi 数据（如果使用了*esp_wifi_set_storage(WIFI_STORAGE_FLASH)* 初始化函数）。
    * NVS API 还可以用于其他应用程序数据。
    * 强烈建议您应为 NVS 分区分配至少 0x3000 字节空间。
    * 如果使用 NVS API 存储大量数据，请增加 NVS 分区的大小（默认是 0x6000 字节）。
  - nvs_keys (4) 是 NVS 秘钥分区。详细信息，请参考*非易失性存储 (NVS) API* 文档。
    * 用于存储加密密钥（如果启用了 *NVS 加密*功能）。
    * 此分区应至少设定为 4096 字节。

其它数据子类型已预留给 esp-idf 的未来使用。

Offset 和 Size 字段

分区若为指定偏移地址，则会紧跟着前一个分区之后开始。若此分区为首个分区，则将紧跟着分区表开始。

app 分区的偏移地址必须要与 0x10000 (64K) 对齐，如果将偏移字段留空，`gen_esp32part.py` 工具会自动计算得到一个满足对齐要求的偏移地址。如果 app 分区的偏移地址没有与 0x10000 (64K) 对齐，则该工具会报错。

app 分区的大小和偏移地址可以采用十进制数、以 0x 为前缀的十六进制数，且支持 K 或 M 的倍数单位（分别代表 1024 和 1024*1024 字节）。

如果您希望允许分区表中的分区采用任意起始偏移量 (*CONFIG_PARTITION_TABLE_OFFSET*)，请将分区表（CSV 文件）中所有分区的偏移字段都留空。注意，此时，如果您更改了分区表中任意分区的偏移地址，则其他分区的偏移地址也会跟着改变。这种情况下，如果您之前还曾设定某个分区采用固定偏移地址，则可能造成分区表冲突，从而导致报错。

Flags 字段

当前仅支持 `encrypted` 标记。如果 Flags 字段设置为 `encrypted`，且已启用*Flash Encryption* 功能，则该分区将会被加密。

注解： app 分区始终会被加密，不管 Flags 字段是否设置。

5.15.4 生成二进制分区表

烧写到 ESP32 中的分区表采用二进制格式，而不是 CSV 文件本身。此时，partition_table/gen_esp32part.py 工具可以实现 CSV 和二进制文件之间的转换。

如果您在项目配置菜单（`idf.py menuconfig`）中设置了分区表 CSV 文件的名称，然后构建项目获执行 `idf.py partition_table`。这时，转换将在编译过程中自动完成。

手动将 CSV 文件转换为二进制文件:

```
python gen_esp32part.py input_partitions.csv binary_partitions.bin
```

手动将二进制文件转换为 CSV 文件:

```
python gen_esp32part.py binary_partitions.bin input_partitions.csv
```

在标准输出（stdout）上，打印二进制分区表的内容（在运行 `idf.py partition_table` 时，我们正是这样打印上文展示的信息摘要的）：

```
python gen_esp32part.py binary_partitions.bin
```

MD5 校验和

二进制格式的分区表中含有一个 MD5 校验和。这个 MD5 校验和是根据分区表内容计算的，可在设备启动阶段，用于验证分区表的完整性。

注意，一些版本较老的 bootloader 无法支持 MD5 校验，如果发现 MD5 校验和则将报错 `invalid magic number 0xebeb`。此时，用户可通过 `gen_esp32part.py` 的 `--disable-md5sum` 选项或者 `menuconfig` 的*CONFIG_PARTITION_TABLE_MD5* 选项关闭 MD5 校验。

5.15.5 烧写分区表

- `idf.py partition_table-flash`： 使用 esptool.py 工具烧写分区表。
- `idf.py flash`： 会烧写所有内容，包括分区表。

在执行 `idf.py partition_table` 命令时，手动烧写分区表的命令也将打印在终端上。

注解： 分区表的更新并不会擦除根据之前分区表存储的数据。此时，您可以使用 `idf.py erase_flash` 命令或者 `esptool.py erase_flash` 命令来擦除 flash 中的所有内容。

5.16 Secure Boot

重要： All references in this document are related to Secure Boot V1 (The AES based Secure Boot Scheme). ESP32 Revision 3 onwards, the preferred secure boot scheme is *Secure Boot V2*. Please refer to Secure Boot V2 document for ESP32 Revision 3.

Secure Boot is a feature for ensuring only your code can run on the chip. Data loaded from flash is verified on each reset.

Secure Boot is separate from the *Flash Encryption* feature, and you can use secure boot without encrypting the flash contents. However, for a secure environment both should be used simultaneously. See *Secure Boot & Flash Encryption* for more details.

重要： Enabling secure boot limits your options for further updates of your ESP32. Make sure to read this document throughly and understand the implications of enabling secure boot.

5.16.1 Background

- Most data is stored in flash. Flash access does not need to be protected from physical access in order for secure boot to function, because critical data is stored (non-software-accessible) in Efuses internal to the chip.

- Efuses are used to store the secure bootloader key (in efuse BLOCK2), and also a single Efuse bit (ABS_DONE_0) is burned (written to 1) to permanently enable secure boot on the chip. For more details about efuse, see Chapter 11 "eFuse Controller" in the Technical Reference Manual.
- To understand the secure boot process, first familiarise yourself with the standard *ESP-IDF boot process*.
- Both stages of the boot process (initial software bootloader load, and subsequent partition & app loading) are verified by the secure boot process, in a "chain of trust" relationship.

5.16.2 Secure Boot Process Overview

This is a high level overview of the secure boot process. Step by step instructions are supplied under *How To Enable Secure Boot*. Further in-depth details are supplied under *Technical Details*:

1. The options to enable secure boot are provided in the *Project Configuration Menu*, under "Secure Boot Configuration" .
2. Secure Boot defaults to signing images and partition table data during the build process. The "Secure boot private signing key" config item is a file path to a ECDSA public/private key pair in a PEM format file.
3. The software bootloader image is built by esp-idf with secure boot support enabled and the public key (signature verification) portion of the secure boot signing key compiled in. This software bootloader image is flashed at offset 0x1000.
4. On first boot, the software bootloader follows the following process to enable secure boot:
   - Hardware secure boot support generates a device secure bootloader key (generated via hardware RNG, then stored read/write protected in efuse), and a secure digest. The digest is derived from the key, an IV, and the bootloader image contents.
   - The secure digest is flashed at offset 0x0 in the flash.
   - Depending on Secure Boot Configuration, efuses are burned to disable JTAG and the ROM BASIC interpreter (it is strongly recommended these options are turned on.)
   - Bootloader permanently enables secure boot by burning the ABS_DONE_0 efuse. The software bootloader then becomes protected (the chip will only boot a bootloader image if the digest matches.)
5. On subsequent boots the ROM bootloader sees that the secure boot efuse is burned, reads the saved digest at 0x0 and uses hardware secure boot support to compare it with a newly calculated digest. If the digest does not match then booting will not continue. The digest and comparison are performed entirely by hardware, and the calculated digest is not readable by software. For technical details see *Secure Boot Hardware Support*.
6. When running in secure boot mode, the software bootloader uses the secure boot signing key (the public key of which is embedded in the bootloader itself, and therefore validated as part of the bootloader)

to verify the signature appended to all subsequent partition tables and app images before they are booted.

5.16.3 Keys

The following keys are used by the secure boot process:

- "secure bootloader key" is a 256-bit AES key that is stored in Efuse block 2. The bootloader can generate this key itself from the internal hardware random number generator, the user does not need to supply it (it is optionally possible to supply this key, see *Re-Flashable Software Bootloader*). The Efuse holding this key is read & write protected (preventing software access) before secure boot is enabled.
  - By default, the Efuse Block 2 Coding Scheme is "None" and a 256 bit key is stored in this block. On some ESP32s, the Coding Scheme is set to 3/4 Encoding (CODING_SCHEME efuse has value 1) and a 192 bit key must be stored in this block. See ESP32 Technical Reference Manual section 20.3.1.3 *System Parameter coding_scheme* for more details. The algorithm operates on a 256 bit key in all cases, 192 bit keys are extended by repeating some bits (*details*).
- "secure boot signing key" is a standard ECDSA public/private key pair (see *Image Signing Algorithm*) in PEM format.
  - The public key from this key pair (for signature verification but not signature creation) is compiled into the software bootloader and used to verify the second stage of booting (partition table, app image) before booting continues. The public key can be freely distributed, it does not need to be kept secret.
  - The private key from this key pair *must be securely kept private*, as anyone who has this key can authenticate to any bootloader that is configured with secure boot and the matching public key.

5.16.4 Bootloader Size

When secure boot is enabled the bootloader app binary `bootloader.bin` may exceed the default bootloader size limit. This is especially likely if flash encryption is enabled as well. The default size limit is 0x7000 (28672) bytes (partition table offset 0x8000 - bootloader offset 0x1000).

If the bootloader becomes too large, the ESP32 will fail to boot - errors will be logged about either invalid partition table or invalid bootloader checksum.

Options to work around this are:

- Reduce *bootloader log level*. Setting log level to Warning, Error or None all significantly reduce the final binary size (but may make it harder to debug).
- Set *partition table offset* to a higher value than 0x8000, to place the partition table later in the flash. This increases the space available for the bootloader. If the *partition table* CSV file contains explicit partition offsets, they will need changing so no partition has an offset lower than

`CONFIG_PARTITION_TABLE_OFFSET + 0x1000`. (This includes the default partition CSV files supplied with ESP-IDF.)

5.16.5 How To Enable Secure Boot

1. Open the *Project Configuration Menu*, navigate to "Secure Boot Configuration" and select the option "One-time Flash". (To understand the alternative "Reflashable" choice, see *Re-Flashable Software Bootloader*.)
2. Select a name for the secure boot signing key. This option will appear after secure boot is enabled. The file can be anywhere on your system. A relative path will be evaluated from the project directory. The file does not need to exist yet.
3. Set other menuconfig options (as desired). Pay particular attention to the "Bootloader Config" options, as you can only flash the bootloader once. Then exit menuconfig and save your configuration
4. The first time you run `make`, if the signing key is not found then an error message will be printed with a command to generate a signing key via `espsecure.py generate_signing_key`.

重要: A signing key generated this way will use the best random number source available to the OS and its Python installation (*/dev/urandom* on OSX/Linux and *CryptGenRandom()* on Windows). If this random number source is weak, then the private key will be weak.

重要: For production environments, we recommend generating the keypair using openssl or another industry standard encryption program. See *Generating Secure Boot Signing Key* for more details.

---

5. Run `idf.py bootloader` to build a secure boot enabled bootloader. The build output will include a prompt for a flashing command, using `esptool.py write_flash`.
6. When you're ready to flash the bootloader, run the specified command (you have to enter it yourself, this step is not performed by make) and then wait for flashing to complete. **Remember this is a one time flash, you can't change the bootloader after this!**.
7. Run `idf.py flash` to build and flash the partition table and the just-built app image. The app image will be signed using the signing key you generated in step 4.

注解: `idf.py flash` doesn't flash the bootloader if secure boot is enabled.

---

8. Reset the ESP32 and it will boot the software bootloader you flashed. The software bootloader will enable secure boot on the chip, and then it verifies the app image signature and boots the app. You should watch the serial console output from the ESP32 to verify that secure boot is enabled and no errors have occurred due to the build configuration.

注解: Secure boot won't be enabled until after a valid partition table and app image have been flashed. This is to prevent accidents before the system is fully configured.

注解: If the ESP32 is reset or powered down during the first boot, it will start the process again on the next boot.

9. On subsequent boots, the secure boot hardware will verify the software bootloader has not changed (using the secure bootloader key) and then the software bootloader will verify the signed partition table and app image (using the public key portion of the secure boot signing key).

5.16.6 Re-Flashable Software Bootloader

Configuration "Secure Boot: One-Time Flash" is the recommended configuration for production devices. In this mode, each device gets a unique key that is never stored outside the device.

However, an alternative mode *Secure Boot: Reflashable* is also available. This mode allows you to supply a binary key file that is used for the secure bootloader key. As you have the key file, you can generate new bootloader images and secure boot digests for them.

In the esp-idf build process, this 256-bit key file is derived from the ECDSA app signing key generated by the user (see the *Generating Secure Boot Signing Key* step below). This private key's SHA-256 digest is used as the secure bootloader key in efuse (as-is for Coding Scheme None, or truncate to 192 bytes for 3/4 Encoding). This is a convenience so you only need to generate/protect a single private key.

注解: Although it's possible, we strongly recommend not generating one secure boot key and flashing it to every device in a production environment. The "One-Time Flash" option is recommended for production environments.

To enable a reflashable bootloader:

1. In the *Project Configuration Menu*, select "Bootloader Config" -> *CONFIG_SECURE_BOOT* -> CONFIG_SECURE_BOOT_V1_ENABLED -> *CONFIG_SECURE_BOOTLOADER_MODE* -> Reflashable.
2. If necessary, set the *CONFIG_SECURE_BOOTLOADER_KEY_ENCODING* based on the coding scheme used by the device. The coding scheme is shown in the `Features` line when `esptool.py` connects to the chip, or in the `espefuse.py summary` output.
2. Follow the steps shown above to choose a signing key file, and generate the key file.
3. Run `idf.py bootloader`. A binary key file will be created, derived from the private key that is used for signing. Two sets of flashing steps will be printed - the first set of steps includes an `espefuse.py`

`burn_key` command which is used to write the bootloader key to efuse. (Flashing this key is a one-time-only process.) The second set of steps can be used to reflash the bootloader with a pre-calculated digest (generated during the build process).

4. Resume from *Step 6 of the one-time flashing process*, to flash the bootloader and enable secure boot. Watch the console log output closely to ensure there were no errors in the secure boot configuration.

5.16.7 Generating Secure Boot Signing Key

The build system will prompt you with a command to generate a new signing key via `espsecure.py generate_signing_key`. This uses the python-ecdsa library, which in turn uses Python's os.urandom() as a random number source.

The strength of the signing key is proportional to (a) the random number source of the system, and (b) the correctness of the algorithm used. For production devices, we recommend generating signing keys from a system with a quality entropy source, and using the best available EC key generation utilities.

For example, to generate a signing key using the openssl command line:

` openssl ecparam -name prime256v1 -genkey -noout -out my_secure_boot_signing_key.pem `

Remember that the strength of the secure boot system depends on keeping the signing key private.

5.16.8 Remote Signing of Images

For production builds, it can be good practice to use a remote signing server rather than have the signing key on the build machine (which is the default esp-idf secure boot configuration). The espsecure.py command line program can be used to sign app images & partition table data for secure boot, on a remote system.

To use remote signing, disable the option "Sign binaries during build". The private signing key does not need to be present on the build system. However, the public (signature verification) key is required because it is compiled into the bootloader (and can be used to verify image signatures during OTA updates.

To extract the public key from the private key:

```
espsecure.py extract_public_key --keyfile PRIVATE_SIGNING_KEY PUBLIC_VERIFICATION_KEY
```

The path to the public signature verification key needs to be specified in the menuconfig under "Secure boot public signature verification key" in order to build the secure bootloader.

After the app image and partition table are built, the build system will print signing steps using espsecure.py:

```
espsecure.py sign_data --keyfile PRIVATE_SIGNING_KEY BINARY_FILE
```

The above command appends the image signature to the existing binary. You can use the *–output* argument to write the signed binary to a separate file:

```
espsecure.py sign_data --keyfile PRIVATE_SIGNING_KEY --output SIGNED_BINARY_FILE BINARY_
↪FILE
```

5.16.9 Secure Boot Best Practices

- Generate the signing key on a system with a quality source of entropy.
- Keep the signing key private at all times. A leak of this key will compromise the secure boot system.
- Do not allow any third party to observe any aspects of the key generation or signing process using espsecure.py. Both processes are vulnerable to timing or other side-channel attacks.
- Enable all secure boot options in the Secure Boot Configuration. These include flash encryption, disabling of JTAG, disabling BASIC ROM interpeter, and disabling the UART bootloader encrypted flash access.
- Use secure boot in combination with *flash encryption* to prevent local readout of the flash contents.

5.16.10 Technical Details

The following sections contain low-level reference descriptions of various secure boot elements:

Secure Boot Hardware Support

The first stage of secure boot verification (checking the software bootloader) is done via hardware. The ESP32' s Secure Boot support hardware can perform three basic operations:

1. Generate a random sequence of bytes from a hardware random number generator.
2. Generate a digest from data (usually the bootloader image from flash) using a key stored in Efuse block 2. The key in Efuse can (& should) be read/write protected, which prevents software access. For full details of this algorithm see *Secure Bootloader Digest Algorithm*. The digest can only be read back by software if Efuse ABS_DONE_0 is *not* burned (ie still 0).
3. Generate a digest from data (usually the bootloader image from flash) using the same algorithm as step 2 and compare it to a pre-calculated digest supplied in a buffer (usually read from flash offset 0x0). The hardware returns a true/false comparison without making the digest available to software. This function is available even when Efuse ABS_DONE_0 is burned.

Secure Bootloader Digest Algorithm

Starting with an "image" of binary data as input, this algorithm generates a digest as output. The digest is sometimes referred to as an "abstract" in hardware documentation.

For a Python version of this algorithm, see the `espsecure.py` tool in the components/esptool_py directory (specifically, the `digest_secure_bootloader` command).

Items marked with (^) are to fulfill hardware restrictions, as opposed to cryptographic restrictions.

1. Read the AES key from efuse block 2, in reversed byte order. If Coding Scheme is set to 3/4 Encoding, extend the 192 bit key to 256 bits using the same algorithm described in *Flash 加密算法*.
2. Prefix the image with a 128 byte randomly generated IV.
3. If the image length is not modulo 128, pad the image to a 128 byte boundary with 0xFF. (^)
4. For each 16 byte plaintext block of the input image: - Reverse the byte order of the plaintext input block (^) - Apply AES256 in ECB mode to the plaintext block. - Reverse the byte order of the ciphertext output block. (^) - Append to the overall ciphertext output.
5. Byte-swap each 4 byte word of the ciphertext (^)
6. Calculate SHA-512 of the ciphertext.

Output digest is 192 bytes of data: The 128 byte IV, followed by the 64 byte SHA-512 digest.

Image Signing Algorithm

Deterministic ECDSA as specified by RFC 6979.

- Curve is NIST256p (openssl calls this curve "prime256v1", it is also sometimes called secp256r1).
- Hash function is SHA256.
- Key format used for storage is PEM.
  - In the bootloader, the public key (for signature verification) is flashed as 64 raw bytes.
- Image signature is 68 bytes - a 4 byte version word (currently zero), followed by a 64 bytes of signature data. These 68 bytes are appended to an app image or partition table data.

Manual Commands

Secure boot is integrated into the esp-idf build system, so `make` will automatically sign an app image if secure boot is enabled. `idf.py bootloader` will produce a bootloader digest if menuconfig is configured for it.

However, it is possible to use the `espsecure.py` tool to make standalone signatures and digests.

To sign a binary image:

```
espsecure.py sign_data --keyfile ./my_signing_key.pem --output ./image_signed.bin image-
↪unsigned.bin
```

Keyfile is the PEM file containing an ECDSA private signing key.

To generate a bootloader digest:

```
espsecure.py digest_secure_bootloader --keyfile ./securebootkey.bin --output ./
↪bootloader-digest.bin build/bootloader/bootloader.bin
```

Keyfile is the 32 byte raw secure boot key for the device.

The output of the `espsecure.py digest_secure_bootloader` command is a single file which contains both the digest and the bootloader appended to it. To flash the combined digest plus bootloader to the device:

```
esptool.py write_flash 0x0 bootloader-digest.bin
```

5.16.11 Secure Boot & Flash Encryption

If secure boot is used without *Flash Encryption*, it is possible to launch "time-of-check to time-of-use" attack, where flash contents are swapped after the image is verified and running. Therefore, it is recommended to use both the features together.

5.16.12 Signed App Verification Without Hardware Secure Boot

The integrity of apps can be checked even without enabling the hardware secure boot option. This option uses the same app signature scheme as hardware secure boot, but unlike hardware secure boot it does not prevent the bootloader from being physically updated. This means that the device can be secured against remote network access, but not physical access. Compared to using hardware Secure Boot this option is much simpler to implement. See *How To Enable Signed App Verification* for step by step instructions.

An app can be verified on update and, optionally, be verified on boot.

- Verification on update: When enabled, the signature is automatically checked whenever the esp_ota_ops.h APIs are used for OTA updates. If hardware secure boot is enabled, this option is always enabled and cannot be disabled. If hardware secure boot is not enabled, this option still adds significant security against network-based attackers by preventing spoofing of OTA updates.
- Verification on boot: When enabled, the bootloader will be compiled with code to verify that an app is signed before booting it. If hardware secure boot is enabled, this option is always enabled and cannot be disabled. If hardware secure boot is not enabled, this option doesn't add significant security by itself so most users will want to leave it disabled.

How To Enable Signed App Verification

1. Open *Project Configuration Menu* -> Security features -> Enable *CONFIG_SECURE_SIGNED_APPS_NO_SECURE_BOOT*
2. "Bootloader verifies app signatures" can be enabled, which verifies app on boot.

3. By default, "Sign binaries during build" will be enabled on selecting "Require signed app images" option, which will sign binary files as a part of build process. The file named in "Secure boot private signing key" will be used to sign the image.

4. If you disable "Sign binaries during build" option then you' ll have to enter path of a public key file used to verify signed images in "Secure boot public signature verification key". In this case, private signing key should be generated by following instructions in *Generating Secure Boot Signing Key*; public verification key and signed image should be generated by following instructions in *Remote Signing of Images*.

5.16.13 Advanced Features

JTAG Debugging

By default, when Secure Boot is enabled then JTAG debugging is disabled via eFuse. The bootloader does this on first boot, at the same time it enables Secure Boot.

See *JTAG with Flash Encryption or Secure Boot* for more information about using JTAG Debugging with either Secure Boot or signed app verification enabled.

5.17 Secure Boot V2

重要: The references in this document are related to Secure Boot v2, the preferred scheme from ESP32-ECO3 onwards. (Refer to *Secure Boot* for ESP32)

Secure Boot V2 uses RSA based app and bootloader verification. This document can also be referred for signing apps with the RSA scheme without signing the bootloader.

5.17.1 Background

Secure Boot protects a device from running unsigned code (verification at time of load). A new RSA based secure boot verification scheme (Secure Boot V2) has been introduced for ESP32 ECO3 onwards.

- The software bootloader' s RSA-PSS signature is verified by the Mask ROM and it is executed post successful verification.
- The verified software bootloader verifies the RSA-PSS signature of the application image before it is executed.

5.17.2 Advantages

- The RSA public key is stored on the device. The corresponding RSA private key is kept secret on a server and is never accessed by the device.
- Same image format & signature verification is applied for applications & software bootloader.
- No secrets are stored on the device. Therefore immune to passive side-channel attacks (timing or power analysis, etc.)

5.17.3 Secure Boot V2 Process

This is an overview of the Secure Boot V2 Process, Step by step instructions are supplied under *How To Enable Secure Boot V2*.

1. Secure Boot V2 verifies the signature blocks appended to the bootloader and application binaries. The signature block contains the image binary signed by a RSA-3072 private key and its corresponding public key. More details on the *Signature Block Format*.
2. On startup, ROM code checks the secure boot v2 bit in eFuse.
3. If secure boot is enabled, ROM checks the SHA-256 of the public key in the signature block in the eFuse.
4. The ROM code validates the public key embedded in the software bootloader' s signature block by matching the SHA-256 of its public key to the SHA-256 in eFuse as per the earlier step. Boot process will be aborted if a valid hash of the public key isn' t found in the eFuse.
5. The ROM code verifies the signature of the bootloader with the pre-validated public key with the RSA-PSS Scheme. In depth information on *Verifying the signature Block*.
6. Software bootloader, reads the app partition and performs similar verification on the application. The application is verified on every boot up and OTA update. If selected OTA app partition fails verification, bootloader will fall back and look for another correctly signed partition.

5.17.4 Signature Block Format

The bootloader and application images are padded to the next 4096 byte boundary, thus the signature has a flash sector of its own. The signature is calculated over all bytes in the image including the padding bytes.

Each signature block contains the following:

- **Offset 0 (1 byte):** Magic byte (0xe7)
- **Offset 1 (1 byte):** Version number byte (currently 0x02), 0x01 is for Secure Boot V1.
- **Offset 2 (2 bytes):** Padding bytes, Reserved. Should be zero.
- **Offset 4 (32 bytes):** SHA-256 hash of only the image content, not including the signature block.

- **Offset 36 (384 bytes):** RSA Public Modulus used for signature verification. (value ‘n’ in RFC8017).
- **Offset 420 (4 bytes):** RSA Public Exponent used for signature verification (value ‘e’ in RFC8017).
- **Offset 424 (384 bytes):** Precalculated R, derived from ‘n’ .
- **Offset 808 (4 bytes):** Precalculated M’ , derived from ‘n’
- **Offset 812 (384 bytes):** RSA-PSS Signature result (section 8.1.1 of RFC8017) of image content, computed using following PSS parameters: SHA256 hash, MFG1 function, 0 length salt, default trailer field (0xBC).
- **Offset 1196:** CRC32 of the preceding 1095 bytes.
- **Offset 1200 (16 bytes):** Zero padding to length 1216 bytes.

注解: R and M’ are used for hardware-assisted Montgomery Multiplication.

The remainder of the signature sector is erased flash (0xFF) which allows writing other signature blocks after previous signature block.

5.17.5 Verifying the signature Block

A signature block is “valid” if the first byte is 0xe7 and a valid CRC32 is stored at offset 1196.

An image is “verified” if the public key stored in any signature block is valid for this device, and if the stored signature is valid for the image data read from flash.

1. The magic byte, signature block CRC is validated.
2. Public key digests are generated per signature block and compared with the digests from eFuse. If none of the digests match, the verification process is aborted.
3. The application image digest is generated and matched with the image digest in the signature blocks. The verification process is aborted is the digests don’ t match.
4. The public key is used to verify the signature of the bootloader image, using RSA-PSS (section 8.1.2 of RFC8017) with the image digest calculated in step (3) for comparison.

- The application signing scheme is set to RSA for secure boot V2 and to ECDSA for secure boot V1.

重要: It is recommended to use secure boot V2 on the chip versions supporting them.

5.17.6 Bootloader Size

{IDF_TARGET_MAX_BOOTLOADER_SIZE:default = "64KB (0x10000 bytes)", esp32 = "48KB (0xC000 bytes)"} {IDF_TARGET_MAX_PARTITION_TABLE_OFFSET:default = "0x12000", esp32 = "0xE000"} .. Above is calculated as 0x1000 at start of flash + IDF_TARGET_MAX_BOOTLOADER_SIZE + 0x1000 signature sector

When secure boot is enabled the bootloader app binary `bootloader.bin` may exceed the default bootloader size limit. This is especially likely if flash encryption is enabled as well. The default size limit is 0x7000 (28672) bytes (partition table offset 0x8000 - bootloader offset 0x1000).

If the bootloader becomes too large, the {IDF_TARGET_NAME} will fail to boot - errors will be logged about either invalid partition table or invalid bootloader checksum.

When Secure Boot V2 is enabled, there is also an absolute binary size limit of {IDF_TARGET_MAX_BOOTLOADER_SIZE} (excluding the 4KB signature), because the bootloader is first loaded into a fixed size buffer for verification.

Options to work around this are:

- Set *bootloader compiler optimization* back to "Size" if it has been changed from this default value.
- Reduce *bootloader log level*. Setting log level to Warning, Error or None all significantly reduce the final binary size (but may make it harder to debug).
- Set *partition table offset* to a higher value than 0x8000, to place the partition table later in the flash. This increases the space available for the bootloader. If the *partition table* CSV file contains explicit partition offsets, they will need changing so no partition has an offset lower than `CONFIG_PARTITION_TABLE_OFFSET + 0x1000`. (This includes the default partition CSV files supplied with ESP-IDF.)

  Note that because of the absolute binary size limit, there is no benefit to moving the partition table any higher than offset {IDF_TARGET_MAX_PARTITION_TABLE_OFFSET}.

5.17.7 eFuse usage

ESP32-ECO3:

- ABS_DONE_1 - Enables secure boot protection on boot.
- BLK2 - Stores the SHA-256 digest of the public key. SHA-256 hash of public key modulus, exponent, precalculated R & M' values (represented as 776 bytes – offsets 36 to 812 - as per the *Signature Block Format*) is written to an eFuse key block.

5.17.8 How To Enable Secure Boot V2

1. Open the *Project Configuration Menu*, in "Security Features" set "Enable hardware Secure Boot in bootloader" to enable Secure Boot. The chip revision should be changed to revision 3(ESP32- ECO3) to view the Secure Boot V2 option.
2. To change the chip revision, set "Minimum Supported ESP32 Revision" to Rev 3 in "Component Config" -> "ESP32- Specific" , the Secure Boot V2 option can be enabled under "Enable hardware Secure Boot in bootloader" -> "Secure Boot Version" . Secure Boot V2 is available for ESP32 ECO3 onwards.
3. Specify the secure boot signing key path. The file can be anywhere on your system. A relative path will be evaluated from the project directory. The file does not need to exist yet.
4. Set other menuconfig options (as desired). Pay particular attention to the "Bootloader Config" options, as you can only flash the bootloader once. Then exit menuconfig and save your configuration
5. The first time you run `make` or `idf.py build`, if the signing key is not found then an error message will be printed with a command to generate a signing key via `espsecure.py generate_signing_key`.

重要: A signing key generated this way will use the best random number source available to the OS and its Python installation (*/dev/urandom* on OSX/Linux and *CryptGenRandom()* on Windows). If this random number source is weak, then the private key will be weak.

重要: For production environments, we recommend generating the keypair using openssl or another industry standard encryption program. See *Generating Secure Boot Signing Key* for more details.

---

6. Run `idf.py bootloader` to build a secure boot enabled bootloader. The build output will include a prompt for a flashing command, using `esptool.py write_flash`.
7. When you' re ready to flash the bootloader, run the specified command (you have to enter it yourself, this step is not performed by the build system) and then wait for flashing to complete.
8. Run `idf.py flash` to build and flash the partition table and the just-built app image. The app image will be signed using the signing key you generated in step 4.

注解: `idf.py flash` doesn' t flash the bootloader if secure boot is enabled.

---

9. Reset the ESP32 and it will boot the software bootloader you flashed. The software bootloader will enable secure boot on the chip, and then it verifies the app image signature and boots the app. You should watch the serial console output from the ESP32 to verify that secure boot is enabled and no errors have occurred due to the build configuration.

注解: Secure boot won’ t be enabled until after a valid partition table and app image have been flashed. This is to prevent accidents before the system is fully configured.

注解: If the ESP32 is reset or powered down during the first boot, it will start the process again on the next boot.

10. On subsequent boots, the secure boot hardware will verify the software bootloader has not changed and the software bootloader will verify the signed app image (using the validated public key portion of its appended signature block).

5.17.9 Restrictions after Secure Boot is enabled

- Any updated bootloader or app will need to be signed with a key matching the digest already stored in efuse.
- After Secure Boot is enabled, no further efuses can be read protected. (If *Flash 加密* is enabled then the bootloader will ensure that any flash encryption key generated on first boot will already be read protected.) If *CONFIG_SECURE_BOOT_INSECURE* is enabled then this behaviour can be disabled, but this is not recommended.

5.17.10 Generating Secure Boot Signing Key

The build system will prompt you with a command to generate a new signing key via `espsecure.py generate_signing_key`. The –version 2 parameter will generate the RSA 3072 private key for Secure Boot V2.

The strength of the signing key is proportional to (a) the random number source of the system, and (b) the correctness of the algorithm used. For production devices, we recommend generating signing keys from a system with a quality entropy source, and using the best available RSA key generation utilities.

For example, to generate a signing key using the openssl command line:

```
` openssl genrsa -out my_secure_boot_signing_key.pem 3072 `
```

Remember that the strength of the secure boot system depends on keeping the signing key private.

5.17.11 Remote Signing of Images

For production builds, it can be good practice to use a remote signing server rather than have the signing key on the build machine (which is the default esp-idf secure boot configuration). The espsecure.py command line program can be used to sign app images & partition table data for secure boot, on a remote system.

To use remote signing, disable the option "Sign binaries during build". The private signing key does not need to be present on the build system.

After the app image and partition table are built, the build system will print signing steps using espsecure.py:

```
espsecure.py sign_data --version 2 --keyfile PRIVATE_SIGNING_KEY BINARY_FILE
```

The above command appends the image signature to the existing binary. You can use the *–output* argument to write the signed binary to a separate file:

```
espsecure.py sign_data --version 2 --keyfile PRIVATE_SIGNING_KEY --output SIGNED_BINARY_
↪FILE BINARY_FILE
```

5.17.12 Secure Boot Best Practices

- Generate the signing key on a system with a quality source of entropy.
- Keep the signing key private at all times. A leak of this key will compromise the secure boot system.
- Do not allow any third party to observe any aspects of the key generation or signing process using espsecure.py. Both processes are vulnerable to timing or other side-channel attacks.
- Enable all secure boot options in the Secure Boot Configuration. These include flash encryption, disabling of JTAG, disabling BASIC ROM interpeter, and disabling the UART bootloader encrypted flash access.
- Use secure boot in combination with *flash encryption* to prevent local readout of the flash contents.

5.17.13 Technical Details

The following sections contain low-level reference descriptions of various secure boot elements:

Manual Commands

Secure boot is integrated into the esp-idf build system, so `make` or `idf.py build` will sign an app image and `idf.py bootloader` will produce a signed bootloader if secure signed binaries on build is enabled.

However, it is possible to use the `espsecure.py` tool to make standalone signatures and digests.

To sign a binary image:

```
espsecure.py sign_data --version 2 --keyfile ./my_signing_key.pem --output ./image_
↪signed.bin image-unsigned.bin
```

Keyfile is the PEM file containing an RSA-3072 private signing key.

5.17.14 Secure Boot & Flash Encryption

If secure boot is used without *Flash Encryption*, it is possible to launch "time-of-check to time-of-use" attack, where flash contents are swapped after the image is verified and running. Therefore, it is recommended to use both the features together.

5.17.15 Advanced Features

JTAG Debugging

By default, when Secure Boot is enabled then JTAG debugging is disabled via eFuse. The bootloader does this on first boot, at the same time it enables Secure Boot.

See *JTAG with Flash Encryption or Secure Boot* for more information about using JTAG Debugging with either Secure Boot or signed app verification enabled.

5.18 ULP 协处理器编程

[English]

5.18.1 ESP32 ULP coprocessor instruction set

This document provides details about the instructions used by ESP32 ULP coprocessor assembler.

ULP coprocessor has 4 16-bit general purpose registers, labeled R0, R1, R2, R3. It also has an 8-bit counter register (stage_cnt) which can be used to implement loops. Stage count regiter is accessed using special instructions.

ULP coprocessor can access 8k bytes of RTC_SLOW_MEM memory region. Memory is addressed in 32-bit word units. It can also access peripheral registers in RTC_CNTL, RTC_IO, and SENS peripherals.

All instructions are 32-bit. Jump instructions, ALU instructions, peripheral register and memory access instructions are executed in 1 cycle. Instructions which work with peripherals (TSENS, ADC, I2C) take variable number of cycles, depending on peripheral operation.

The instruction syntax is case insensitive. Upper and lower case letters can be used and intermixed arbitrarily. This is true both for register names and instruction names.

Note about addressing

ESP32 ULP coprocessor' s JUMP, ST, LD instructions which take register as an argument (jump address, store/load base address) expect the argument to be expressed in 32-bit words.

Consider the following example program:

```
entry:
        NOP
        NOP
        NOP
        NOP
loop:
        MOVE R1, loop
        JUMP R1
```

When this program is assembled and linked, address of label `loop` will be equal to 16 (expressed in bytes). However *JUMP* instruction expects the address stored in register to be expressed in 32-bit words. To account for this common use case, assembler will convert the address of label *loop* from bytes to words, when generating `MOVE` instruction, so the code generated code will be equivalent to:

```
0000    NOP
0004    NOP
0008    NOP
000c    NOP
0010    MOVE R1, 4
0014    JUMP R1
```

The other case is when the argument of `MOVE` instruction is not a label but a constant. In this case assembler will use the value as is, without any conversion:

```
.set        val, 0x10
MOVE        R1, val
```

In this case, value loaded into R1 will be `0x10`.

Similar considerations apply to `LD` and `ST` instructions. Consider the following code:

```
        .global array
array:  .long 0
        .long 0
        .long 0
        .long 0

        MOVE R1, array
        MOVE R2, 0x1234
        ST R2, R1, 0      // write value of R2 into the first array element,
                          // i.e. array[0]
```

(下页继续)

(续上页)

```
        ST R2, R1, 4      // write value of R2 into the second array element
                          // (4 byte offset), i.e. array[1]

        ADD R1, R1, 2     // this increments address by 2 words (8 bytes)
        ST R2, R1, 0      // write value of R2 into the third array element,
                          // i.e. array[2]
```

Note about instruction execution time

ULP coprocessor is clocked from RTC_FAST_CLK, which is normally derived from the internal 8MHz oscillator. Applications which need to know exact ULP clock frequency can calibrate it against the main XTAL clock:

```
#include "soc/rtc.h"

// calibrate 8M/256 clock against XTAL, get 8M/256 clock period
uint32_t rtc_8md256_period = rtc_clk_cal(RTC_CAL_8MD256, 100);
uint32_t rtc_fast_freq_hz = 1000000ULL * (1 << RTC_CLK_CAL_FRACT) * 256 / rtc_8md256_
↪period;
```

ULP coprocessor needs certain number of clock cycles to fetch each instuction, plus certain number of cycles to execute it, depending on the instruction. See description of each instruction below for details on the execution time.

Instruction fetch time is:

- 2 clock cycles —for instructions following ALU and branch instructions.
- 4 clock cycles —in other cases.

Note that when accessing RTC memories and RTC registers, ULP coprocessor has lower priority than the main CPUs. This means that ULP coprocessor execution may be suspended while the main CPUs access same memory region as the ULP.

NOP - no operation

Syntax **NOP**

Operands None

Cycles 2 cycle to execute, 4 cycles to fetch next instruction

Description No operation is performed. Only the PC is incremented.

Example:

```
1:    NOP
```

ADD - Add to register

Syntax **ADD** *Rdst, Rsrc1, Rsrc2*

ADD *Rdst, Rsrc1, imm*

Operands

- *Rdst* - Register R[0..3]
- *Rsrc1* - Register R[0..3]
- *Rsrc2* - Register R[0..3]
- *Imm* - 16-bit signed value

Cycles 2 cycles to execute, 4 cycles to fetch next instruction

Description The instruction adds source register to another source register or to a 16-bit signed value and stores result to the destination register.

Examples:

```
1:    ADD R1, R2, R3        //R1 = R2 + R3

2:    Add R1, R2, 0x1234    //R1 = R2 + 0x1234

3:    .set value1, 0x03     //constant value1=0x03
      Add R1, R2, value1    //R1 = R2 + value1


4:    .global label         //declaration of variable label
      Add R1, R2, label     //R1 = R2 + label
        ...
      label: nop            //definition of variable label
```

SUB - Subtract from register

Syntax **SUB** *Rdst, Rsrc1, Rsrc2*

SUB *Rdst, Rsrc1, imm*

Operands

- *Rdst* - Register R[0..3]

- *Rsrc1* - Register R[0..3]
- *Rsrc2* - Register R[0..3]
- *Imm* - 16-bit signed value

Cycles 2 cycles to execute, 4 cycles to fetch next instruction

Description The instruction subtracts the source register from another source register or subtracts 16-bit signed value from a source register, and stores result to the destination register.

Examples:

```
1:          SUB R1, R2, R3              //R1 = R2 - R3

2:          sub R1, R2, 0x1234          //R1 = R2 - 0x1234

3:          .set value1, 0x03           //constant value1=0x03
            SUB R1, R2, value1          //R1 = R2 - value1
4:          .global label               //declaration of variable label
            SUB R1, R2, label           //R1 = R2 - label
              ....
  label:    nop                         //definition of variable label
```

AND - Logical AND of two operands

Syntax **AND** *Rdst, Rsrc1, Rsrc2*

AND *Rdst, Rsrc1, imm*

Operands

- *Rdst* - Register R[0..3]
- *Rsrc1* - Register R[0..3]
- *Rsrc2* - Register R[0..3]
- *Imm* - 16-bit signed value

Cycles 2 cycles to execute, 4 cycles to fetch next instruction

Description The instruction does logical AND of a source register and another source register or 16-bit signed value and stores result to the destination register.

Examples:

```
1:          AND R1, R2, R3          //R1 = R2 & R3
```

(下页继续)

(续上页)

```
2:        AND R1, R2, 0x1234      //R1 = R2 & 0x1234

3:        .set value1, 0x03       //constant value1=0x03
          AND R1, R2, value1      //R1 = R2 & value1

4:        .global label           //declaration of variable label
          AND R1, R2, label       //R1 = R2 & label
              ...
  label:  nop                     //definition of variable label
```

OR - Logical OR of two operands

Syntax **OR** *Rdst, Rsrc1, Rsrc2*

OR *Rdst, Rsrc1, imm*

Operands

- *Rdst* - Register R[0..3]
- *Rsrc1* - Register R[0..3]
- *Rsrc2* - Register R[0..3]
- *Imm* - 16-bit signed value

Cycles 2 cycles to execute, 4 cycles to fetch next instruction

Description The instruction does logical OR of a source register and another source register or 16-bit signed value and stores result to the destination register.

Examples:

```
1:        OR R1, R2, R3           //R1 = R2 \| R3

2:        OR R1, R2, 0x1234       //R1 = R2 \| 0x1234

3:        .set value1, 0x03       //constant value1=0x03
          OR R1, R2, value1       //R1 = R2 \| value1

4:        .global label           //declaration of variable label
          OR R1, R2, label        //R1 = R2 \|label
          ...
  label: nop                      //definition of variable label
```

LSH - Logical Shift Left

Syntax **LSH** *Rdst, Rsrc1, Rsrc2*

LSH *Rdst, Rsrc1, imm*

Operands

- *Rdst* - Register R[0..3]
- *Rsrc1* - Register R[0..3]
- *Rsrc2* - Register R[0..3]
- *Imm* - 16-bit signed value

Cycles 2 cycles to execute, 4 cycles to fetch next instruction

Description The instruction does logical shift to left of source register to number of bits from another source register or 16-bit signed value and store result to the destination register.

Examples:

```
1:       LSH R1, R2, R3            //R1 = R2 << R3

2:       LSH R1, R2, 0x03          //R1 = R2 << 0x03

3:       .set value1, 0x03         //constant value1=0x03
         LSH R1, R2, value1        //R1 = R2 << value1

4:       .global label             //declaration of variable label
         LSH R1, R2, label         //R1 = R2 << label
         ...
  label:  nop                       //definition of variable label
```

RSH - Logical Shift Right

Syntax **RSH** *Rdst, Rsrc1, Rsrc2*

RSH *Rdst, Rsrc1, imm*

Operands *Rdst* - Register R[0..3] *Rsrc1* - Register R[0..3] *Rsrc2* - Register R[0..3] *Imm* - 16-bit signed value

Cycles 2 cycles to execute, 4 cycles to fetch next instruction

Description The instruction does logical shift to right of source register to number of bits from another source register or 16-bit signed value and store result to the destination register.

Examples:

```
1:        RSH R1, R2, R3              //R1 = R2 >> R3

2:        RSH R1, R2, 0x03            //R1 = R2 >> 0x03

3:        .set value1, 0x03           //constant value1=0x03
          RSH R1, R2, value1          //R1 = R2 >> value1

4:        .global label               //declaration of variable label
          RSH R1, R2, label           //R1 = R2 >> label
  label:  nop                         //definition of variable label
```

MOVE – Move to register

Syntax **MOVE** *Rdst, Rsrc*

MOVE *Rdst, imm*

Operands

- *Rdst* – Register R[0..3]
- *Rsrc* – Register R[0..3]
- *Imm* – 16-bit signed value

Cycles 2 cycles to execute, 4 cycles to fetch next instruction

Description The instruction move to destination register value from source register or 16-bit signed value.

Note that when a label is used as an immediate, the address of the label will be converted from bytes to words. This is because LD, ST, and JUMP instructions expect the address register value to be expressed in words rather than bytes. To avoid using an extra instruction

Examples:

```
1:        MOVE       R1, R2           //R1 = R2

2:        MOVE       R1, 0x03         //R1 = 0x03

3:        .set       value1, 0x03     //constant value1=0x03
          MOVE       R1, value1       //R1 = value1

4:        .global     label           //declaration of label
          MOVE        R1, label       //R1 = address_of(label) / 4
          ...
  label:  nop                         //definition of label
```

ST – Store data to the memory

Syntax ST *Rsrc, Rdst, offset*

Operands

- *Rsrc* – Register R[0..3], holds the 16-bit value to store
- *Rdst* – Register R[0..3], address of the destination, in 32-bit words
- *Offset* – 10-bit signed value, offset in bytes

Cycles 4 cycles to execute, 4 cycles to fetch next instruction

Description The instruction stores the 16-bit value of Rsrc to the lower half-word of memory with address Rdst+offset. The upper half-word is written with the current program counter (PC), expressed in words, shifted left by 5 bits:

```
Mem[Rdst + offset / 4]{31:0} = {PC[10:0], 5'b0, Rsrc[15:0]}
```

The application can use higher 16 bits to determine which instruction in the ULP program has written any particular word into memory.

Examples:

```
1:        ST  R1, R2, 0x12        //MEM[R2+0x12] = R1

2:        .data                   //Data section definition
  Addr1:  .word     123           // Define label Addr1 16 bit
          .set      offs, 0x00    // Define constant offs
          .text                   //Text section definition
          MOVE      R1, 1         // R1 = 1
          MOVE      R2, Addr1     // R2 = Addr1
          ST        R1, R2, offs  // MEM[R2 +  0] = R1
                                  // MEM[Addr1 + 0] will be 32'h600001
```

LD – Load data from the memory

Syntax LD *Rdst, Rsrc, offset*

Operands *Rdst* – Register R[0..3], destination

Rsrc – Register R[0..3], holds address of destination, in 32-bit words

Offset – 10-bit signed value, offset in bytes

Cycles 4 cycles to execute, 4 cycles to fetch next instruction

Description The instruction loads lower 16-bit half-word from memory with address Rsrc+offset into the destination register Rdst:

```
Rdst[15:0] = Mem[Rsrc + offset / 4][15:0]
```

Examples:

```
1:        LD  R1, R2, 0x12            //R1 = MEM[R2+0x12]

2:        .data                       //Data section definition
  Addr1:  .word     123               // Define label Addr1 16 bit
          .set      offs, 0x00        // Define constant offs
          .text                       //Text section definition
          MOVE      R1, 1             // R1 = 1
          MOVE      R2, Addr1         // R2 = Addr1 / 4 (address of label is converted␣
↪into words)
          LD        R1, R2, offs      // R1 = MEM[R2 +  0]
                                      // R1 will be 123
```

JUMP – Jump to an absolute address

Syntax **JUMP** *Rdst*

JUMP *ImmAddr*

JUMP *Rdst, Condition*

JUMP *ImmAddr, Condition*

Operands

- *Rdst* – Register R[0..3] containing address to jump to (expressed in 32-bit words)
- *ImmAddr* – 13 bits address (expressed in bytes), aligned to 4 bytes
- ***Condition*:**
  - EQ – jump if last ALU operation result was zero
  - OV – jump if last ALU has set overflow flag

Cycles 2 cycles to execute, 2 cycles to fetch next instruction

Description The instruction makes jump to the specified address. Jump can be either unconditional or based on an ALU flag.

Examples:

```
1:        JUMP       R1             // Jump to address in R1 (address in R1 is in 32-bit␣
↪words)

2:        JUMP       0x120, EQ      // Jump to address 0x120 (in bytes) if ALU result is␣
↪zero

3:        JUMP       label          // Jump to label
          ...
  label:  nop                       // Definition of label

4:        .global    label          // Declaration of global label

          MOVE       R1, label      // R1 = label (value loaded into R1 is in words)
          JUMP       R1             // Jump to label
          ...
  label:  nop                       // Definition of label
```

JUMPR – Jump to a relative offset (condition based on R0)

Syntax JUMPR *Step, Threshold, Condition*

Operands

- *Step* – relative shift from current position, in bytes
- *Threshold* – threshold value for branch condition
- ***Condition*:**
  - *EQ* (equal) – jump if value in R0 == threshold
  - *LT* (less than) – jump if value in R0 < threshold
  - *LE* (less or equal) – jump if value in R0 <= threshold
  - *GT* (greater than) – jump if value in R0 > threshold
  - *GE* (greater or equal) – jump if value in R0 >= threshold

Cycles Conditions *LT*, *GE*, *LE* and *GT*: 2 cycles to execute, 2 cycles to fetch next instruction

Conditions *LE* and *GT* are implemented in the assembler using one **JUMPR** instructions:

```
// JUMPR target, threshold, GT is implemented as:

        JUMPR target, threshold+1, GE
```

(下页继续)

(续上页)

```
// JUMPR target, threshold, LE is implemented as:

        JUMPR target, threshold + 1, LT
```

Conditions *EQ* is implemented in the assembler using two **JUMPR** instructions:

```
// JUMPR target, threshold, EQ is implemented as:

        JUMPR next, threshold + 1, GE
        JUMPR target, threshold, GE
next:
```

Therefore the execution time will depend on the branches taken: either 2 cycles to execute + 2 cycles to fetch, or 4 cycles to execute + 4 cycles to fetch.

Description The instruction makes a jump to a relative address if condition is true. Condition is the result of comparison of R0 register value and the threshold value.

Examples:

```
1:pos:    JUMPR       16, 20, GE   // Jump to address (position + 16 bytes) if value in␣
↪R0 >= 20

2:        // Down counting loop using R0 register
          MOVE        R0, 16       // load 16 into R0
  label:  SUB         R0, R0, 1    // R0--
          NOP                      // do something
          JUMPR       label, 1, GE // jump to label if R0 >= 1
```

JUMPS – Jump to a relative address (condition based on stage count)

Syntax JUMPS *Step, Threshold, Condition*

Operands

- *Step* – relative shift from current position, in bytes
- *Threshold* – threshold value for branch condition
- ***Condition*:**
  - *EQ* (equal) – jump if value in stage_cnt == threshold
  - *LT* (less than) – jump if value in stage_cnt < threshold
  - *LE* (less or equal) - jump if value in stage_cnt <= threshold

- *GT* (greater than) – jump if value in stage_cnt > threshold
- *GE* (greater or equal) —jump if value in stage_cnt >= threshold

Cycles Conditions *LE*, *LT*, *GE*: 2 cycles to execute, 2 cycles to fetch next instruction

Conditions *EQ*, *GT* are implemented in the assembler using two **JUMPS** instructions:

```
// JUMPS target, threshold, EQ is implemented as:

        JUMPS next, threshold, LT
        JUMPS target, threshold, LE
next:

// JUMPS target, threshold, GT is implemented as:

        JUMPS next, threshold, LE
        JUMPS target, threshold, GE
next:
```

Therefore the execution time will depend on the branches taken: either 2 cycles to execute + 2 cycles to fetch, or 4 cycles to execute + 4 cycles to fetch.

Description The instruction makes a jump to a relative address if condition is true. Condition is the result of comparison of count register value and threshold value.

Examples:

```
1:pos:    JUMPS     16, 20, EQ     // Jump to (position + 16 bytes) if stage_cnt == 20

2:        // Up counting loop using stage count register
          STAGE_RST                  // set stage_cnt to 0
  label:  STAGE_INC  1               // stage_cnt++
          NOP                        // do something
          JUMPS       label, 16, LT  // jump to label if stage_cnt < 16
```

STAGE_RST – Reset stage count register

Syntax STAGE_RST

Operands No operands

Description The instruction sets the stage count register to 0

Cycles 2 cycles to execute, 4 cycles to fetch next instruction

Examples:

```
1:       STAGE_RST      // Reset stage count register
```

STAGE_INC – Increment stage count register

Syntax STAGE_INC *Value*

Operands

- *Value* – 8 bits value

Cycles 2 cycles to execute, 4 cycles to fetch next instruction

Description The instruction increments stage count register by given value.

Examples:

```
1:        STAGE_INC      10          // stage_cnt += 10

2:        // Up counting loop example:
          STAGE_RST                  // set stage_cnt to 0
  label:  STAGE_INC  1               // stage_cnt++
          NOP                        // do something
          JUMPS      label, 16, LT   // jump to label if stage_cnt < 16
```

STAGE_DEC – Decrement stage count register

Syntax STAGE_DEC *Value*

Operands

- *Value* – 8 bits value

Cycles 2 cycles to execute, 4 cycles to fetch next instruction

Description The instruction decrements stage count register by given value.

Examples:

```
1:        STAGE_DEC      10        // stage_cnt -= 10;

2:        // Down counting loop exaple
          STAGE_RST                // set stage_cnt to 0
          STAGE_INC  16            // increment stage_cnt to 16
  label:  STAGE_DEC  1             // stage_cnt--;
          NOP                      // do something
          JUMPS      label, 0, GT  // jump to label if stage_cnt > 0
```

HALT – End the program

Syntax HALT

Operands No operands

Cycles 2 cycles to execute

Description The instruction halts the ULP coprocessor and restarts ULP wakeup timer, if it is enabled.

Examples:

```
1:      HALT      // Halt the coprocessor
```

WAKE – Wake up the chip

Syntax WAKE

Operands No operands

Cycles 2 cycles to execute, 4 cycles to fetch next instruction

Description The instruction sends an interrupt from ULP to RTC controller.

- If the SoC is in deep sleep mode, and ULP wakeup is enabled, this causes the SoC to wake up.
- If the SoC is not in deep sleep mode, and ULP interrupt bit (RTC_CNTL_ULP_CP_INT_ENA) is set in RTC_CNTL_INT_ENA_REG register, RTC interrupt will be triggered.

Note that before using WAKE instruction, ULP program may needs to wait until RTC controller is ready to wake up the main CPU. This is indicated using RTC_CNTL_RDY_FOR_WAKEUP bit of RTC_CNTL_LOW_POWER_ST_REG register. If WAKE instruction is executed while RTC_CNTL_RDY_FOR_WAKEUP is zero, it has no effect (wake up does not occur).

Examples:

```
1: is_rdy_for_wakeup:                   // Read RTC_CNTL_RDY_FOR_WAKEUP bit
          READ_RTC_FIELD(RTC_CNTL_LOW_POWER_ST_REG, RTC_CNTL_RDY_FOR_WAKEUP)
          AND r0, r0, 1
          JUMP is_rdy_for_wakeup, eq    // Retry until the bit is set
          WAKE                          // Trigger wake up
          REG_WR 0x006, 24, 24, 0       // Stop ULP timer (clear RTC_CNTL_ULP_CP_SLP_
↪TIMER_EN)
          HALT                          // Stop the ULP program
          // After these instructions, SoC will wake up,
          // and ULP will not run again until started by the main program.
```

SLEEP – set ULP wakeup timer period

Syntax **SLEEP** *sleep_reg*

Operands

- *sleep_reg* – 0..4, selects one of `SENS_ULP_CP_SLEEP_CYCx_REG` registers.

Cycles 2 cycles to execute, 4 cycles to fetch next instruction

Description The instruction selects which of the `SENS_ULP_CP_SLEEP_CYCx_REG` (x = 0..4) register values is to be used by the ULP wakeup timer as wakeup period. By default, the value from `SENS_ULP_CP_SLEEP_CYC0_REG` is used.

Examples:

```
1:        SLEEP     1          // Use period set in SENS_ULP_CP_SLEEP_CYC1_REG

2:        .set sleep_reg, 4    // Set constant
          SLEEP  sleep_reg     // Use period set in SENS_ULP_CP_SLEEP_CYC4_REG
```

WAIT – wait some number of cycles

Syntax **WAIT** *Cycles*

Operands

- *Cycles* – number of cycles for wait

Cycles 2 + *Cycles* cycles to execute, 4 cycles to fetch next instruction

Description The instruction delays for given number of cycles.

Examples:

```
1:        WAIT     10          // Do nothing for 10 cycles

2:        .set  wait_cnt, 10   // Set a constant
          WAIT  wait_cnt       // wait for 10 cycles
```

TSENS – do measurement with temperature sensor

Syntax

- **TSENS** *Rdst, Wait_Delay*

Operands

- *Rdst* – Destination Register R[0..3], result will be stored to this register

- *Wait_Delay* – number of cycles used to perform the measurement

Cycles 2 + *Wait_Delay* + 3 * TSENS_CLK to execute, 4 cycles to fetch next instruction

Description The instruction performs measurement using TSENS and stores the result into a general purpose register.

Examples:

```
1:        TSENS     R1, 1000      // Measure temperature sensor for 1000 cycles,
                                   // and store result to R1
```

ADC – do measurement with ADC

Syntax

- **ADC** *Rdst, Sar_sel, Mux*
- **ADC** *Rdst, Sar_sel, Mux, 0* —deprecated form

Operands

- *Rdst* – Destination Register R[0..3], result will be stored to this register
- *Sar_sel* – Select ADC: 0 = SARADC1, 1 = SARADC2
- *Mux* - selected PAD, SARADC Pad[Mux+1] is enabled

Cycles `23 + max(1, SAR_AMP_WAIT1) + max(1, SAR_AMP_WAIT2) + max(1, SAR_AMP_WAIT3) + SARx_SAMPLE_CYCLE + SARx_SAMPLE_BIT` cycles to execute, 4 cycles to fetch next instruction

Description The instruction makes measurements from ADC.

Examples:

```
1:        ADC       R1, 0, 1      // Measure value using ADC1 pad 2 and store result into␣
↪R1
```

I2C_RD - read single byte from I2C slave

Syntax

- **I2C_RD** *Sub_addr, High, Low, Slave_sel*

Operands

- *Sub_addr* – Address within the I2C slave to read.
- *High*, *Low* —Define range of bits to read. Bits outside of [High, Low] range are masked.
- *Slave_sel* - Index of I2C slave address to use.

Cycles Execution time mostly depends on I2C communication time. 4 cycles to fetch next instruction.

Description `I2C_RD` instruction reads one byte from I2C slave with index `Slave_sel`. Slave address (in 7-bit format) has to be set in advance into *SENS_I2C_SLAVE_ADDRx* register field, where `x == Slave_sel`. 8 bits of read result is stored into *R0* register.

Examples:

```
1:        I2C_RD      0x10, 7, 0, 0      // Read byte from sub-address 0x10 of slave␣
↪with address set in SENS_I2C_SLAVE_ADDR0
```

I2C_WR - write single byte to I2C slave

Syntax

- **I2C_WR** *Sub_addr, Value, High, Low, Slave_sel*

Operands

- *Sub_addr* – Address within the I2C slave to write.
- *Value* – 8-bit value to be written.
- *High*, *Low* —Define range of bits to write. Bits outside of [High, Low] range are masked.
- *Slave_sel* - Index of I2C slave address to use.

Cycles Execution time mostly depends on I2C communication time. 4 cycles to fetch next instruction.

Description `I2C_WR` instruction writes one byte to I2C slave with index `Slave_sel`. Slave address (in 7-bit format) has to be set in advance into *SENS_I2C_SLAVE_ADDRx* register field, where `x == Slave_sel`.

Examples:

```
1:        I2C_WR      0x20, 0x33, 7, 0, 1      // Write byte 0x33 to sub-address 0x20 of␣
↪slave with address set in SENS_I2C_SLAVE_ADDR1.
```

REG_RD – read from peripheral register

Syntax **REG_RD** *Addr, High, Low*

Operands

- *Addr* – Register address, in 32-bit words
- *High* – Register end bit number
- *Low* – Register start bit number

Cycles 4 cycles to execute, 4 cycles to fetch next instruction

Description The instruction reads up to 16 bits from a peripheral register into a general purpose register: `R0 = REG[Addr][High:Low]`.

This instruction can access registers in RTC_CNTL, RTC_IO, SENS, and RTC_I2C peripherals. Address of the the register, as seen from the ULP, can be calculated from the address of the same register on the DPORT bus as follows:

```
addr_ulp = (addr_dport - DR_REG_RTCCNTL_BASE) / 4
```

Examples:

```
1:        REG_RD      0x120, 7, 4     // load 4 bits: R0 = {12'b0, REG[0x120][7:4]}
```

REG_WR – write to peripheral register

Syntax **REG_WR** *Addr, High, Low, Data*

Operands

- *Addr* – Register address, in 32-bit words.
- *High* – Register end bit number
- *Low* – Register start bit number
- *Data* – Value to write, 8 bits

Cycles 8 cycles to execute, 4 cycles to fetch next instruction

Description The instruction writes up to 8 bits from an immediate data value into a peripheral register: `REG[Addr][High:Low] = data`.

This instruction can access registers in RTC_CNTL, RTC_IO, SENS, and RTC_I2C peripherals. Address of the the register, as seen from the ULP, can be calculated from the address of the same register on the DPORT bus as follows:

```
addr_ulp = (addr_dport - DR_REG_RTCCNTL_BASE) / 4
```

Examples:

```
1:        REG_WR      0x120, 7, 0, 0x10   // set 8 bits: REG[0x120][7:0] = 0x10
```

Convenience macros for peripheral registers access

ULP source files are passed through C preprocessor before the assembler. This allows certain macros to be used to facilitate access to peripheral registers.

Some existing macros are defined in `soc/soc_ulp.h` header file. These macros allow access to the fields of peripheral registers by their names. Peripheral registers names which can be used with these macros are the ones defined in `soc/rtc_cntl_reg.h`, `soc/rtc_io_reg.h`, `soc/sens_reg.h`, and `soc/rtc_i2c_reg.h`.

READ_RTC_REG(rtc_reg, low_bit, bit_width) Read up to 16 bits from rtc_reg[low_bit + bit_width - 1 : low_bit] into R0. For example:

```
#include "soc/soc_ulp.h"
#include "soc/rtc_cntl_reg.h"

/* Read 16 lower bits of RTC_CNTL_TIME0_REG into R0 */
READ_RTC_REG(RTC_CNTL_TIME0_REG, 0, 16)
```

READ_RTC_FIELD(rtc_reg, field) Read from a field in rtc_reg into R0, up to 16 bits. For example:

```
#include "soc/soc_ulp.h"
#include "soc/sens_reg.h"

/* Read 8-bit SENS_TSENS_OUT field of SENS_SAR_SLAVE_ADDR3_REG into R0 */
READ_RTC_FIELD(SENS_SAR_SLAVE_ADDR3_REG, SENS_TSENS_OUT)
```

WRITE_RTC_REG(rtc_reg, low_bit, bit_width, value) Write immediate value into rtc_reg[low_bit + bit_width - 1 : low_bit], bit_width <= 8. For example:

```
#include "soc/soc_ulp.h"
#include "soc/rtc_io_reg.h"

/* Set BIT(2) of RTC_GPIO_OUT_DATA_W1TS field in RTC_GPIO_OUT_W1TS_REG */
WRITE_RTC_REG(RTC_GPIO_OUT_W1TS_REG, RTC_GPIO_OUT_DATA_W1TS_S + 2, 1, 1)
```

WRITE_RTC_FIELD(rtc_reg, field, value) Write immediate value into a field in rtc_reg, up to 8 bits. For example:

```
#include "soc/soc_ulp.h"
#include "soc/rtc_cntl_reg.h"

/* Set RTC_CNTL_ULP_CP_SLP_TIMER_EN field of RTC_CNTL_STATE0_REG to 0 */
WRITE_RTC_FIELD(RTC_CNTL_STATE0_REG, RTC_CNTL_ULP_CP_SLP_TIMER_EN, 0)
```

5.18.2 ESP32-S2 ULP coprocessor instruction set

This document provides details about the instructions used by ESP32-S2 ULP coprocessor assembler.

ULP coprocessor has 4 16-bit general purpose registers, labeled R0, R1, R2, R3. It also has an 8-bit counter register (stage_cnt) which can be used to implement loops. Stage count regiter is accessed using special instructions.

ULP coprocessor can access 8k bytes of RTC_SLOW_MEM memory region. Memory is addressed in 32-bit word units. It can also access peripheral registers in RTC_CNTL, RTC_IO, and SENS peripherals.

All instructions are 32-bit. Jump instructions, ALU instructions, peripheral register and memory access instructions are executed in 1 cycle. Instructions which work with peripherals (TSENS, ADC, I2C) take variable number of cycles, depending on peripheral operation.

The instruction syntax is case insensitive. Upper and lower case letters can be used and intermixed arbitrarily. This is true both for register names and instruction names.

Note about addressing

ESP32-S2 ULP coprocessor' s JUMP, ST, LD instructions which take register as an argument (jump address, store/load base address) expect the argument to be expressed in 32-bit words.

Consider the following example program:

```
entry:
        NOP
        NOP
        NOP
        NOP
loop:
        MOVE R1, loop
        JUMP R1
```

When this program is assembled and linked, address of label `loop` will be equal to 16 (expressed in bytes). However *JUMP* instruction expects the address stored in register to be expressed in 32-bit words. To account for this common use case, assembler will convert the address of label *loop* from bytes to words, when generating `MOVE` instruction, so the code generated code will be equivalent to:

```
0000    NOP
0004    NOP
0008    NOP
000c    NOP
0010    MOVE R1, 4
0014    JUMP R1
```

The other case is when the argument of `MOVE` instruction is not a label but a constant. In this case assembler will use the value as is, without any conversion:

```
.set        val, 0x10
MOVE        R1, val
```

In this case, value loaded into R1 will be `0x10`.

Similar considerations apply to `LD` and `ST` instructions. Consider the following code:

```
        .global array
array:  .long 0
        .long 0
        .long 0
        .long 0

        MOVE R1, array
        MOVE R2, 0x1234
        ST R2, R1, 0      // write value of R2 into the first array element,
                          // i.e. array[0]

        ST R2, R1, 4      // write value of R2 into the second array element
                          // (4 byte offset), i.e. array[1]

        ADD R1, R1, 2     // this increments address by 2 words (8 bytes)
        ST R2, R1, 0      // write value of R2 into the third array element,
                          // i.e. array[2]
```

Note about instruction execution time

ULP coprocessor is clocked from RTC_FAST_CLK, which is normally derived from the internal 8MHz oscillator. Applications which need to know exact ULP clock frequency can calibrate it against the main XTAL clock:

```
#include "soc/rtc.h"

// calibrate 8M/256 clock against XTAL, get 8M/256 clock period
uint32_t rtc_8md256_period = rtc_clk_cal(RTC_CAL_8MD256, 100);
uint32_t rtc_fast_freq_hz = 1000000ULL * (1 << RTC_CLK_CAL_FRACT) * 256 / rtc_8md256_
↪period;
```

ULP coprocessor needs certain number of clock cycles to fetch each instruction, plus certain number of cycles to execute it, depending on the instruction. See description of each instruction below for details on the execution time.

Instruction fetch time is:

- 2 clock cycles —for instructions following ALU and branch instructions.
- 4 clock cycles —in other cases.

Note that when accessing RTC memories and RTC registers, ULP coprocessor has lower priority than the main CPUs. This means that ULP coprocessor execution may be suspended while the main CPUs access same memory region as the ULP.

Difference between ESP32 ULP and ESP32-S2 ULP Instruction sets

Compare to the ESP32 ULP coprocessor, the ESP-S2 ULP coprocessor has extended instruction set. The ESP32-S2 ULP is not binary compatible with ESP32 ULP, but the assembled program that was written for the ESP32 ULP will also work on the ESP32-S2 ULP after rebuild. The list of the new instructions that was added to the ESP32-S2 ULP is: LDL, LDH, STO, ST32, STI32. The detailed description of these commands please see below.

NOP - no operation

Syntax **NOP**

Operands None

Cycles 2 cycle to execute, 4 cycles to fetch next instruction

Description No operation is performed. Only the PC is incremented.

Example:

```
1:    NOP
```

ADD - Add to register

Syntax **ADD** *Rdst, Rsrc1, Rsrc2*

ADD *Rdst, Rsrc1, imm*

Operands

- *Rdst* - Register R[0..3]
- *Rsrc1* - Register R[0..3]
- *Rsrc2* - Register R[0..3]
- *Imm* - 16-bit signed value

Cycles 2 cycles to execute, 4 cycles to fetch next instruction

Description The instruction adds source register to another source register or to a 16-bit signed value and stores result to the destination register.

Examples:

```
1:    ADD R1, R2, R3        //R1 = R2 + R3

2:    Add R1, R2, 0x1234    //R1 = R2 + 0x1234

3:    .set value1, 0x03     //constant value1=0x03
      Add R1, R2, value1    //R1 = R2 + value1


4:    .global label         //declaration of variable label
      Add R1, R2, label     //R1 = R2 + label
        ...
      label: nop            //definition of variable label
```

SUB - Subtract from register

Syntax SUB *Rdst, Rsrc1, Rsrc2*

SUB *Rdst, Rsrc1, imm*

Operands

- *Rdst* - Register R[0..3]
- *Rsrc1* - Register R[0..3]
- *Rsrc2* - Register R[0..3]
- *Imm* - 16-bit signed value

Cycles 2 cycles to execute, 4 cycles to fetch next instruction

Description The instruction subtracts the source register from another source register or subtracts 16-bit signed value from a source register, and stores result to the destination register.

Examples:

```
1:          SUB R1, R2, R3            //R1 = R2 - R3

2:          sub R1, R2, 0x1234        //R1 = R2 - 0x1234

3:          .set value1, 0x03         //constant value1=0x03
            SUB R1, R2, value1        //R1 = R2 - value1
4:          .global label             //declaration of variable label
            SUB R1, R2, label         //R1 = R2 - label
```

(下页继续)

(续上页)

```
            ....
 label:   nop                          //definition of variable label
```

AND - Logical AND of two operands

Syntax **AND** *Rdst, Rsrc1, Rsrc2*

AND *Rdst, Rsrc1, imm*

Operands

- *Rdst* - Register R[0..3]
- *Rsrc1* - Register R[0..3]
- *Rsrc2* - Register R[0..3]
- *Imm* - 16-bit signed value

Cycles 2 cycles to execute, 4 cycles to fetch next instruction

Description The instruction does logical AND of a source register and another source register or 16-bit signed value and stores result to the destination register.

Examples:

```
1:        AND R1, R2, R3          //R1 = R2 & R3

2:        AND R1, R2, 0x1234      //R1 = R2 & 0x1234

3:        .set value1, 0x03       //constant value1=0x03
          AND R1, R2, value1      //R1 = R2 & value1

4:        .global label           //declaration of variable label
          AND R1, R2, label       //R1 = R2 & label
              ...
 label:   nop                     //definition of variable label
```

OR - Logical OR of two operands

Syntax **OR** *Rdst, Rsrc1, Rsrc2*

OR *Rdst, Rsrc1, imm*

Operands

- *Rdst* - Register R[0..3]

- *Rsrc1* - Register R[0..3]
- *Rsrc2* - Register R[0..3]
- *Imm* - 16-bit signed value

Cycles 2 cycles to execute, 4 cycles to fetch next instruction

Description The instruction does logical OR of a source register and another source register or 16-bit signed value and stores result to the destination register.

Examples:

```
1:       OR R1, R2, R3           //R1 = R2 \| R3

2:       OR R1, R2, 0x1234       //R1 = R2 \| 0x1234

3:       .set value1, 0x03       //constant value1=0x03
         OR R1, R2, value1       //R1 = R2 \| value1

4:       .global label           //declaration of variable label
         OR R1, R2, label        //R1 = R2 \|label
         ...
  label: nop                     //definition of variable label
```

LSH - Logical Shift Left

Syntax **LSH** *Rdst, Rsrc1, Rsrc2*

LSH *Rdst, Rsrc1, imm*

Operands

- *Rdst* - Register R[0..3]
- *Rsrc1* - Register R[0..3]
- *Rsrc2* - Register R[0..3]
- *Imm* - 16-bit signed value

Cycles 2 cycles to execute, 4 cycles to fetch next instruction

Description The instruction does logical shift to left of source register to number of bits from another source register or 16-bit signed value and store result to the destination register.

Examples:

```
1:       LSH R1, R2, R3            //R1 = R2 << R3

2:       LSH R1, R2, 0x03          //R1 = R2 << 0x03

3:       .set value1, 0x03         //constant value1=0x03
         LSH R1, R2, value1        //R1 = R2 << value1

4:       .global label             //declaration of variable label
         LSH R1, R2, label         //R1 = R2 << label
         ...
  label:  nop                       //definition of variable label
```

RSH - Logical Shift Right

Syntax **RSH** *Rdst, Rsrc1, Rsrc2*

RSH *Rdst, Rsrc1, imm*

Operands *Rdst* - Register R[0..3] *Rsrc1* - Register R[0..3] *Rsrc2* - Register R[0..3] *Imm* - 16-bit signed value

Cycles 2 cycles to execute, 4 cycles to fetch next instruction

Description The instruction does logical shift to right of source register to number of bits from another source register or 16-bit signed value and store result to the destination register.

Examples:

```
1:        RSH R1, R2, R3              //R1 = R2 >> R3

2:        RSH R1, R2, 0x03            //R1 = R2 >> 0x03

3:        .set value1, 0x03           //constant value1=0x03
          RSH R1, R2, value1          //R1 = R2 >> value1

4:        .global label               //declaration of variable label
          RSH R1, R2, label           //R1 = R2 >> label
  label:  nop                         //definition of variable label
```

MOVE – Move to register

Syntax **MOVE** *Rdst, Rsrc*

MOVE *Rdst, imm*

Operands

- *Rdst* – Register R[0..3]
- *Rsrc* – Register R[0..3]
- *Imm* – 16-bit signed value

Cycles 2 cycles to execute, 4 cycles to fetch next instruction

Description The instruction move to destination register value from source register or 16-bit signed value.

Note that when a label is used as an immediate, the address of the label will be converted from bytes to words. This is because LD, ST, and JUMP instructions expect the address register value to be expressed in words rather than bytes. To avoid using an extra instruction

Examples:

```
1:        MOVE       R1, R2            //R1 = R2

2:        MOVE       R1, 0x03          //R1 = 0x03

3:        .set       value1, 0x03      //constant value1=0x03
          MOVE       R1, value1        //R1 = value1

4:        .global     label            //declaration of label
          MOVE        R1, label        //R1 = address_of(label) / 4
          ...
  label:  nop                          //definition of label
```

STL/ST – Store data to the low 16 bits of 32-bits memory

Syntax **ST** *Rsrc, Rdst, offset, Label* **STL** *Rsrc, Rdst, offset, Label*

Operands

- *Rsrc* – Register R[0..3], holds the 16-bit value to store
- *Rdst* – Register R[0..3], address of the destination, in 32-bit words
- *Offset* – 11-bit signed value, offset in bytes
- *Label* – 2-bit user defined unsigned value

Cycles 4 cycles to execute, 4 cycles to fetch next instruction

Description The instruction stores the 16-bit value of Rsrc to the lower half-word of memory with address Rdst+offset:

```
Mem[Rdst + offset / 4]{15:0} = {Rsrc[15:0]}
Mem[Rdst + offset / 4]{15:0} = {Label[1:0],Rsrc[13:0]}
```

The ST command introduced to make compatibility with previous versions of UPL core. The application can use higher 16 bits to determine which instruction in the ULP program has written any particular word into memory.

Examples:

```
1:        STL  R1, R2, 0x12        //MEM[R2+0x12] = R1

2:        .data                   //Data section definition
  Addr1:  .word     123           // Define label Addr1 16 bit
          .set      offs, 0x00    // Define constant offs
          .text                   //Text section definition
          MOVE      R1, 1         // R1 = 1
          MOVE      R2, Addr1     // R2 = Addr1
          STL       R1, R2, offs  // MEM[R2 +  0] = R1
                                  // MEM[Addr1 + 0] will be 32'hxxxx0001
3:
          MOVE      R1, 1             // R1 = 1
          STL       R1, R2, 0x12,1    // MEM[R2+0x12] 0xxxxx4001
```

STH – Store data to the high 16 bits of 32-bits memory

Syntax STH *Rsrc, Rdst, offset, Label*

Operands

- *Rsrc* – Register R[0..3], holds the 16-bit value to store
- *Rdst* – Register R[0..3], address of the destination, in 32-bit words
- *Offset* – 11-bit signed value, offset in bytes
- *Label* – 2-bit user defined unsigned value

Cycles 4 cycles to execute, 4 cycles to fetch next instruction

Description The instruction stores the 16-bit value of Rsrc to the high half-word of memory with address Rdst+offset:

```
Mem[Rdst + offset / 4]{31:16} = {Rsrc[15:0]}
Mem[Rdst + offset / 4]{31:16} = {Label[1:0],Rsrc[13:0]}
```

Examples:

```
1:        STH  R1, R2, 0x12       //MEM[R2+0x12][31:16] = R1
```

(下页继续)

(续上页)

```
2:        .data                  //Data section definition
  Addr1:  .word     123          // Define label Addr1 16 bit
          .set      offs, 0x00   // Define constant offs
          .text                  //Text section definition
          MOVE      R1, 1        // R1 = 1
          MOVE      R2, Addr1    // R2 = Addr1
          STH       R1, R2, offs // MEM[R2 +  0] = R1
                                 // MEM[Addr1 + 0] will be 32'h0001xxxx
3:
          MOVE      R1, 1              // R1 = 1
          STH       R1, R2, 0x12, 1    //MEM[R2+0x12] 0x4001xxxx
```

STO – Set offset value for auto increment operation

Syntax STO *offset*

Operands

- *Offset* – 11-bit signed value, offset in bytes

Cycles 4 cycles to execute, 4 cycles to fetch next instruction

Description The instruction set 16-bit value to the offset register:

```
offset = value/ 4
```

Examples:

```
1:        STO  0x12              // Offset = 0x12/4

2:        .data                  //Data section definition
  Addr1:  .word     123          // Define label Addr1 16 bit
          .set      offs, 0x00   // Define constant offs
          .text                  //Text section definition
          STO       offs         // Offset = 0x00
```

STI – Store data to the 32-bits memory with auto increment of predefined offset address

Syntax STI *Rsrc, Rdst, Label*

Operands

- *Rsrc* – Register R[0..3], holds the 16-bit value to store

- *Rdst* – Register R[0..3], address of the destination, in 32-bit words
- *Label* – 2-bit user defined unsigned value

Cycles 4 cycles to execute, 4 cycles to fetch next instruction

Description The instruction stores the 16-bit value of Rsrc to the low and high half-word of memory with address Rdst+offset with auto increment of offset:

```
Mem[Rdst + offset / 4]{15:0/31:16} = {Rsrc[15:0]}
Mem[Rdst + offset / 4]{15:0/31:16} = {Label[1:0],Rsrc[13:0]}
```

Examples:

```
1:        STO  0                  // Set offset to 0
          STI  R1, R2, 0x12       //MEM[R2+0x12][15:0] = R1
          STI  R1, R2, 0x12       //MEM[R2+0x12][31:16] = R1

2:        .data                   //Data section definition
  Addr1:  .word     123           // Define label Addr1 16 bit
          .set      offs, 0x00    // Define constant offs
          .text                   //Text section definition
          STO       0             // Set offset to 0
          MOVE      R1, 1         // R1 = 1
          MOVE      R2, Addr1     // R2 = Addr1
          STI       R1, R2        // MEM[R2 +  0] = R1
                                  // MEM[Addr1 + 0] will be 32'hxxxx0001
          STIx       R1, R2        // MEM[R2 +  0] = R1
                                  // MEM[Addr1 + 0] will be 32'h00010001
3:
          STO       0             // Set offset to 0
          MOVE      R1, 1         // R1 = 1
          STI       R1, R2, 1     //MEM[R2+0x12] 0xxxxx4001
          STI       R1, R2, 1     //MEM[R2+0x12] 0x40014001
```

ST32 – Store 32-bits data to the 32-bits memory

Syntax ST32 *Rsrc, Rdst, offset, Label*

Operands

- *Rsrc* – Register R[0..3], holds the 16-bit value to store
- *Rdst* – Register R[0..3], address of the destination, in 32-bit words
- *Offset* – 11-bit signed value, offset in bytes

- *Label* – 2-bit user defined unsigned value

Cycles 4 cycles to execute, 4 cycles to fetch next instruction

Description The instruction stores 11 bits of the PC value, label value and the 16-bit value of Rsrc to the 32-bits memory with address Rdst+offset:

```
Mem[Rdst + offset / 4]{31:0} = {PC[10:0],0[2:0],Label[1:0],Rsrc[15:0]}
```

Examples:

```
1:        ST32  R1, R2, 0x12, 0   //MEM[R2+0x12][31:0] = {PC[10:0],0[2:0],Label[1:0],
↪Rsrc[15:0]}

2:        .data                   //Data section definition
  Addr1:  .word     123           // Define label Addr1 16 bit
          .set      offs, 0x00    // Define constant offs
          .text                   //Text section definition
          MOVE      R1, 1         // R1 = 1
          MOVE      R2, Addr1     // R2 = Addr1
          ST32      R1, R2, offs,1// MEM[R2 +  0] = {PC[10:0],0[2:0],Label[1:0],
↪Rsrc[15:0]}
                                  // MEM[Addr1 + 0] will be 32'h00010001
```

STI32 – Store 32-bits data to the 32-bits memory with auto increment of adress offset

Syntax STI32 *Rsrc, Rdst, Label*

Operands

- *Rsrc* – Register R[0..3], holds the 16-bit value to store
- *Rdst* – Register R[0..3], address of the destination, in 32-bit words
- *Label* – 2-bit user defined unsigned value

Cycles 4 cycles to execute, 4 cycles to fetch next instruction

Description The instruction stores 11 bits of the PC value, label value and the 16-bit value of Rsrc to the 32-bits memory with address Rdst+offset:

```
Mem[Rdst + offset / 4]{31:0} = {PC[10:0],0[2:0],Label[1:0],Rsrc[15:0]}
```

Where offset value set by STO instruction

Examples:

```
1:        STO    0x12
          STI32  R1, R2, 0   //MEM[R2+0x12][31:0] = {PC[10:0],0[2:0],Label[1:0],
↪Rsrc[15:0]}
          STI32  R1, R2, 0   //MEM[R2+0x13][31:0] = {PC[10:0],0[2:0],Label[1:0],
↪Rsrc[15:0]}

2:        .data                  //Data section definition
  Addr1:  .word     123          // Define label Addr1 16 bit
          .set      offs, 0x00   // Define constant offs
          .text                  //Text section definition
          MOVE      R1, 1        // R1 = 1
          MOVE      R2, Addr1    // R2 = Addr1
          STO       offs
          STI32     R1, R2, 1// MEM[R2 +  0] = {PC[10:0],0[2:0],Label[1:0],Rsrc[15:0]}
                                 // MEM[Addr1 + 0] will be 32'h00010001
          ST32      R1, R2, 1// MEM[R2 +  1] = {PC[10:0],0[2:0],Label[1:0],Rsrc[15:0]}
                                 // MEM[Addr1 + 1] will be 32'h00010001
```

LDL/LD – Load data from low part of the 32-bits memory

Syntax **LD** *Rdst, Rsrc, offset* **LDL** *Rdst, Rsrc, offset*

Operands *Rdst* – Register R[0..3], destination

Rsrc – Register R[0..3], holds address of destination, in 32-bit words

Offset – 10-bit signed value, offset in bytes

Cycles 4 cycles to execute, 4 cycles to fetch next instruction

Description The instruction loads lower 16-bit half-word from memory with address Rsrc+offset into the destination register Rdst:

```
Rdst[15:0] = Mem[Rsrc + offset / 4][15:0]
```

The LD command do the same as LDL, and included for compatibility with previous versions of ULP core.

Examples:

```
1:        LDL  R1, R2, 0x12          //R1 = MEM[R2+0x12]

2:        .data                     //Data section definition
  Addr1:  .word     123             // Define label Addr1 16 bit
```

(下页继续)

(续上页)

```
          .set      offs, 0x00          // Define constant offs
          .text                         //Text section definition
          MOVE      R1, 1               // R1 = 1
          MOVE      R2, Addr1           // R2 = Addr1 / 4 (address of label is converted␣
↪into words)
          LDL       R1, R2, offs        // R1 = MEM[R2 +  0]
                                        // R1 will be 123
```

LDH – Load data from high part of the 32-bits memory

Syntax LDH *Rdst, Rsrc, offset*

Operands *Rdst* – Register R[0..3], destination

Rsrc – Register R[0..3], holds address of destination, in 32-bit words

Offset – 10-bit signed value, offset in bytes

Cycles 4 cycles to execute, 4 cycles to fetch next instruction

Description The instruction loads higher 16-bit half-word from memory with address Rsrc+offset into the destination register Rdst:

```
Rdst[15:0] = Mem[Rsrc + offset / 4][15:0]
```

The LD command do the same as LDL, and included for compatibility with previous versions of ULP core.

Examples:

```
1:        LDH  R1, R2, 0x12            //R1 = MEM[R2+0x12]

2:        .data                       //Data section definition
  Addr1:  .word     0x12345678              // Define label Addr1 16 bit
          .set      offs, 0x00          // Define constant offs
          .text                         //Text section definition
          MOVE      R1, 1               // R1 = 1
          MOVE      R2, Addr1           // R2 = Addr1 / 4 (address of label is converted␣
↪into words)
          LDH       R1, R2, offs        // R1 = MEM[R2 +  0]
                                        // R1 will be 0x1234
```

JUMP – Jump to an absolute address

Syntax **JUMP** *Rdst*

JUMP *ImmAddr*

JUMP *Rdst, Condition*

JUMP *ImmAddr, Condition*

Operands

- *Rdst* – Register R[0..3] containing address to jump to (expressed in 32-bit words)
- *ImmAddr* – 13 bits address (expressed in bytes), aligned to 4 bytes
- ***Condition*:**
  - EQ – jump if last ALU operation result was zero
  - OV – jump if last ALU has set overflow flag

Cycles 2 cycles to execute, 2 cycles to fetch next instruction

Description The instruction makes jump to the specified address. Jump can be either unconditional or based on an ALU flag.

Examples:

```
1:        JUMP       R1            // Jump to address in R1 (address in R1 is in 32-bit␣
↪words)

2:        JUMP       0x120, EQ     // Jump to address 0x120 (in bytes) if ALU result is␣
↪zero

3:        JUMP       label         // Jump to label
          ...
  label:  nop                      // Definition of label

4:        .global    label         // Declaration of global label

          MOVE       R1, label     // R1 = label (value loaded into R1 is in words)
          JUMP       R1            // Jump to label
          ...
  label:  nop                      // Definition of label
```

JUMPR – Jump to a relative offset (condition based on R0)

Syntax **JUMPR** *Step, Threshold, Condition*

Operands

- *Step* – relative shift from current position, in bytes
- *Threshold* – threshold value for branch condition
- ***Condition*:**
  - *EQ* (equal) – jump if value in R0 == threshold
  - *LT* (less than) – jump if value in R0 < threshold
  - *LE* (less or equal) – jump if value in R0 <= threshold
  - *GT* (greater than) – jump if value in R0 > threshold
  - *GE* (greater or equal) – jump if value in R0 >= threshold

Cycles Conditions *EQ*, *GT* and *LT*: 2 cycles to execute, 2 cycles to fetch next instruction

Conditions *LE* and *GE* are implemented in the assembler using two **JUMPR** instructions:

```
// JUMPR target, threshold, LE is implemented as:

        JUMPR target, threshold, EQ
        JUMPR target, threshold, LT

// JUMPR target, threshold, GE is implemented as:

        JUMPR target, threshold, EQ
        JUMPR target, threshold, GT
```

Therefore the execution time will depend on the branches taken: either 2 cycles to execute + 2 cycles to fetch, or 4 cycles to execute + 4 cycles to fetch.

Description The instruction makes a jump to a relative address if condition is true. Condition is the result of comparison of R0 register value and the threshold value.

Examples:

```
1:pos:    JUMPR       16, 20, GE   // Jump to address (position + 16 bytes) if value in␣
↪R0 >= 20

2:        // Down counting loop using R0 register
          MOVE        R0, 16       // load 16 into R0
  label:  SUB         R0, R0, 1    // R0--
          NOP                      // do something
          JUMPR       label, 1, GE // jump to label if R0 >= 1
```

JUMPS – Jump to a relative address (condition based on stage count)

Syntax **JUMPS** *Step, Threshold, Condition*

Operands

- *Step* – relative shift from current position, in bytes
- *Threshold* – threshold value for branch condition
- ***Condition*:**
  - *EQ* (equal) – jump if value in stage_cnt == threshold
  - *LT* (less than) – jump if value in stage_cnt < threshold
  - *LE* (less or equal) - jump if value in stage_cnt <= threshold
  - *GT* (greater than) – jump if value in stage_cnt > threshold
  - *GE* (greater or equal) —jump if value in stage_cnt >= threshold

Cycles 2 cycles to execute, 2 cycles to fetch next instruction:

```
// JUMPS target, threshold, EQ is implemented as:

        JUMPS next, threshold, LT
        JUMPS target, threshold, LE
next:

// JUMPS target, threshold, GT is implemented as:

        JUMPS next, threshold, LE
        JUMPS target, threshold, GE
next:
```

Therefore the execution time will depend on the branches taken: either 2 cycles to execute + 2 cycles to fetch, or 4 cycles to execute + 4 cycles to fetch.

Description The instruction makes a jump to a relative address if condition is true. Condition is the result of comparison of count register value and threshold value.

Examples:

```
1:pos:    JUMPS     16, 20, EQ     // Jump to (position + 16 bytes) if stage_cnt == 20

2:        // Up counting loop using stage count register
          STAGE_RST                 // set stage_cnt to 0
  label:  STAGE_INC  1              // stage_cnt++
```

(下页继续)

(续上页)

```
          NOP                         // do something
          JUMPS       label, 16, LT   // jump to label if stage_cnt < 16
```

STAGE_RST – Reset stage count register

Syntax STAGE_RST

Operands No operands

Description The instruction sets the stage count register to 0

Cycles 2 cycles to execute, 4 cycles to fetch next instruction

Examples:

```
1:        STAGE_RST      // Reset stage count register
```

STAGE_INC – Increment stage count register

Syntax STAGE_INC *Value*

Operands

- *Value* – 8 bits value

Cycles 2 cycles to execute, 4 cycles to fetch next instruction

Description The instruction increments stage count register by given value.

Examples:

```
1:        STAGE_INC      10           // stage_cnt += 10

2:        // Up counting loop example:
          STAGE_RST                   // set stage_cnt to 0
  label:  STAGE_INC  1                // stage_cnt++
          NOP                         // do something
          JUMPS      label, 16, LT    // jump to label if stage_cnt < 16
```

STAGE_DEC – Decrement stage count register

Syntax STAGE_DEC *Value*

Operands

- *Value* – 8 bits value

Cycles 2 cycles to execute, 4 cycles to fetch next instruction

Description The instruction decrements stage count register by given value.

Examples:

```
1:        STAGE_DEC      10        // stage_cnt -= 10;

2:        // Down counting loop exaple
          STAGE_RST                // set stage_cnt to 0
          STAGE_INC  16            // increment stage_cnt to 16
  label:  STAGE_DEC  1             // stage_cnt--;
          NOP                      // do something
          JUMPS      label, 0, GT  // jump to label if stage_cnt > 0
```

HALT – End the program

Syntax HALT

Operands No operands

Cycles 2 cycles to execute

Description The instruction halts the ULP coprocessor and restarts ULP wakeup timer, if it is enabled.

Examples:

```
1:       HALT      // Halt the coprocessor
```

WAKE – Wake up the chip

Syntax WAKE

Operands No operands

Cycles 2 cycles to execute, 4 cycles to fetch next instruction

Description The instruction sends an interrupt from ULP to RTC controller.

- If the SoC is in deep sleep mode, and ULP wakeup is enabled, this causes the SoC to wake up.
- If the SoC is not in deep sleep mode, and ULP interrupt bit (RTC_CNTL_ULP_CP_INT_ENA) is set in RTC_CNTL_INT_ENA_REG register, RTC interrupt will be triggered.

Note that before using WAKE instruction, ULP program may needs to wait until RTC controller is ready to wake up the main CPU. This is indicated using RTC_CNTL_RDY_FOR_WAKEUP bit of RTC_CNTL_LOW_POWER_ST_REG register. If WAKE instruction is executed while RTC_CNTL_RDY_FOR_WAKEUP is zero, it has no effect (wake up does not occur).

Examples:

```
1: is_rdy_for_wakeup:                   // Read RTC_CNTL_RDY_FOR_WAKEUP bit
          READ_RTC_FIELD(RTC_CNTL_LOW_POWER_ST_REG, RTC_CNTL_RDY_FOR_WAKEUP)
          AND r0, r0, 1
          JUMP is_rdy_for_wakeup, eq    // Retry until the bit is set
          WAKE                          // Trigger wake up
          REG_WR 0x006, 24, 24, 0       // Stop ULP timer (clear RTC_CNTL_ULP_CP_SLP_
↪TIMER_EN)
          HALT                          // Stop the ULP program
          // After these instructions, SoC will wake up,
          // and ULP will not run again until started by the main program.
```

SLEEP – set ULP wakeup timer period

Syntax **SLEEP** *sleep_reg*

Operands

- *sleep_reg* – 0..4, selects one of `SENS_ULP_CP_SLEEP_CYCx_REG` registers.

Cycles 2 cycles to execute, 4 cycles to fetch next instruction

Description The instruction selects which of the `SENS_ULP_CP_SLEEP_CYCx_REG` (x = 0..4) register values is to be used by the ULP wakeup timer as wakeup period. By default, the value from `SENS_ULP_CP_SLEEP_CYC0_REG` is used.

Examples:

```
1:        SLEEP     1          // Use period set in SENS_ULP_CP_SLEEP_CYC1_REG

2:        .set sleep_reg, 4    // Set constant
          SLEEP  sleep_reg     // Use period set in SENS_ULP_CP_SLEEP_CYC4_REG
```

WAIT – wait some number of cycles

Syntax **WAIT** *Cycles*

Operands

- *Cycles* – number of cycles for wait

Cycles 2 + *Cycles* cycles to execute, 4 cycles to fetch next instruction

Description The instruction delays for given number of cycles.

Examples:

```
1:        WAIT     10          // Do nothing for 10 cycles

2:        .set  wait_cnt, 10   // Set a constant
          WAIT  wait_cnt       // wait for 10 cycles
```

TSENS – do measurement with temperature sensor

Syntax

- **TSENS** *Rdst, Wait_Delay*

Operands

- *Rdst* – Destination Register R[0..3], result will be stored to this register
- *Wait_Delay* – number of cycles used to perform the measurement

Cycles 2 + *Wait_Delay* + 3 * TSENS_CLK to execute, 4 cycles to fetch next instruction

Description The instruction performs measurement using TSENS and stores the result into a general purpose register.

Examples:

```
1:        TSENS     R1, 1000     // Measure temperature sensor for 1000 cycles,
                                 // and store result to R1
```

ADC – do measurement with ADC

Syntax

- **ADC** *Rdst, Sar_sel, Mux*
- **ADC** *Rdst, Sar_sel, Mux, 0* —deprecated form

Operands

- *Rdst* – Destination Register R[0..3], result will be stored to this register
- *Sar_sel* – Select ADC: 0 = SARADC1, 1 = SARADC2
- *Mux* - selected PAD, SARADC Pad[Mux+1] is enabled

Cycles `23 + max(1, SAR_AMP_WAIT1) + max(1, SAR_AMP_WAIT2) + max(1, SAR_AMP_WAIT3) + SARx_SAMPLE_CYCLE + SARx_SAMPLE_BIT` cycles to execute, 4 cycles to fetch next instruction

Description The instruction makes measurements from ADC.

Examples:

```
1:        ADC      R1, 0, 1      // Measure value using ADC1 pad 2 and store result into␣
↪R1
```

I2C_RD - read single byte from I2C slave

Syntax

- **I2C_RD** *Sub_addr, High, Low, Slave_sel*

Operands

- *Sub_addr* – Address within the I2C slave to read.
- *High*, *Low* —Define range of bits to read. Bits outside of [High, Low] range are masked.
- *Slave_sel* - Index of I2C slave address to use.

Cycles Execution time mostly depends on I2C communication time. 4 cycles to fetch next instruction.

Description `I2C_RD` instruction reads one byte from I2C slave with index `Slave_sel`. Slave address (in 7-bit format) has to be set in advance into *SENS_I2C_SLAVE_ADDRx* register field, where `x == Slave_sel`. 8 bits of read result is stored into *R0* register.

Examples:

```
1:        I2C_RD      0x10, 7, 0, 0      // Read byte from sub-address 0x10 of slave␣
↪with address set in SENS_I2C_SLAVE_ADDR0
```

I2C_WR - write single byte to I2C slave

Syntax

- **I2C_WR** *Sub_addr, Value, High, Low, Slave_sel*

Operands

- *Sub_addr* – Address within the I2C slave to write.
- *Value* – 8-bit value to be written.
- *High*, *Low* —Define range of bits to write. Bits outside of [High, Low] range are masked.
- *Slave_sel* - Index of I2C slave address to use.

Cycles Execution time mostly depends on I2C communication time. 4 cycles to fetch next instruction.

Description `I2C_WR` instruction writes one byte to I2C slave with index `Slave_sel`. Slave address (in 7-bit format) has to be set in advance into *SENS_I2C_SLAVE_ADDRx* register field, where `x == Slave_sel`.

Examples:

```
1:        I2C_WR      0x20, 0x33, 7, 0, 1      // Write byte 0x33 to sub-address 0x20 of␣
↪slave with address set in SENS_I2C_SLAVE_ADDR1.
```

REG_RD – read from peripheral register

Syntax **REG_RD** *Addr, High, Low*

Operands

- *Addr* – register address, in 32-bit words
- *High* – High part of R0
- *Low* – Low part of R0

Cycles 4 cycles to execute, 4 cycles to fetch next instruction

Description The instruction reads up to 16 bits from a peripheral register into a general purpose register: `R0 = REG[Addr][High:Low]`.

This instruction can access registers in RTC_CNTL, RTC_IO, SENS, and RTC_I2C peripherals. Address of the the register, as seen from the ULP, can be calculated from the address of the same register on the DPORT bus as follows:

```
addr_ulp = (addr_dport - DR_REG_RTCCNTL_BASE) / 4
```

Examples:

```
1:        REG_RD      0x120, 2, 0     // load 4 bits: R0 = {12'b0, REG[0x120][7:4]}
```

REG_WR – write to peripheral register

Syntax **REG_WR** *Addr, High, Low, Data*

Operands

- *Addr* – register address, in 32-bit words.
- *High* – High part of R0
- *Low* – Low part of R0
- *Data* – value to write, 8 bits

Cycles 8 cycles to execute, 4 cycles to fetch next instruction

Description The instruction writes up to 8 bits from a general purpose register into a peripheral register. `REG[Addr][High:Low] = data`

This instruction can access registers in RTC_CNTL, RTC_IO, SENS, and RTC_I2C peripherals. Address of the the register, as seen from the ULP, can be calculated from the address of the same register on the DPORT bus as follows:

```
addr_ulp = (addr_dport - DR_REG_RTCCNTL_BASE) / 4
```

Examples:

```
1:        REG_WR      0x120, 7, 0, 0x10   // set 8 bits: REG[0x120][7:0] = 0x10
```

Convenience macros for peripheral registers access

ULP source files are passed through C preprocessor before the assembler. This allows certain macros to be used to facilitate access to peripheral registers.

Some existing macros are defined in `soc/soc_ulp.h` header file. These macros allow access to the fields of peripheral registers by their names. Peripheral registers names which can be used with these macros are the ones defined in `soc/rtc_cntl_reg.h`, `soc/rtc_io_reg.h`, `soc/sens_reg.h`, and `soc/rtc_i2c_reg.h`.

READ_RTC_REG(rtc_reg, low_bit, bit_width) Read up to 16 bits from rtc_reg[low_bit + bit_width - 1 : low_bit] into R0. For example:

```
#include "soc/soc_ulp.h"
#include "soc/rtc_cntl_reg.h"

/* Read 16 lower bits of RTC_CNTL_TIME0_REG into R0 */
READ_RTC_REG(RTC_CNTL_TIME0_REG, 0, 16)
```

READ_RTC_FIELD(rtc_reg, field) Read from a field in rtc_reg into R0, up to 16 bits. For example:

```
#include "soc/soc_ulp.h"
#include "soc/sens_reg.h"

/* Read 8-bit SENS_TSENS_OUT field of SENS_SAR_SLAVE_ADDR3_REG into R0 */
READ_RTC_FIELD(SENS_SAR_SLAVE_ADDR3_REG, SENS_TSENS_OUT)
```

WRITE_RTC_REG(rtc_reg, low_bit, bit_width, value) Write immediate value into rtc_reg[low_bit + bit_width - 1 : low_bit], bit_width <= 8. For example:

```
#include "soc/soc_ulp.h"
#include "soc/rtc_io_reg.h"

/* Set BIT(2) of RTC_GPIO_OUT_DATA_W1TS field in RTC_GPIO_OUT_W1TS_REG */
WRITE_RTC_REG(RTC_GPIO_OUT_W1TS_REG, RTC_GPIO_OUT_DATA_W1TS_S + 2, 1, 1)
```

WRITE_RTC_FIELD(rtc_reg, field, value) Write immediate value into a field in rtc_reg, up to 8 bits. For example:

```
#include "soc/soc_ulp.h"
#include "soc/rtc_cntl_reg.h"

/* Set RTC_CNTL_ULP_CP_SLP_TIMER_EN field of RTC_CNTL_STATE0_REG to 0 */
WRITE_RTC_FIELD(RTC_CNTL_STATE0_REG, RTC_CNTL_ULP_CP_SLP_TIMER_EN, 0)
```

5.18.3 Programming ULP coprocessor using C macros

In addition to the existing binutils port for the ESP32 ULP coprocessor, it is possible to generate programs for the ULP by embedding assembly-like macros into an ESP32 application. Here is an example how this can be done:

```
const ulp_insn_t program[] = {
    I_MOVI(R3, 16),          // R3 <- 16
    I_LD(R0, R3, 0),         // R0 <- RTC_SLOW_MEM[R3 + 0]
    I_LD(R1, R3, 1),         // R1 <- RTC_SLOW_MEM[R3 + 1]
    I_ADDR(R2, R0, R1),      // R2 <- R0 + R1
    I_ST(R2, R3, 2),         // R2 -> RTC_SLOW_MEM[R2 + 2]
    I_HALT()
};
size_t load_addr = 0;
size_t size = sizeof(program)/sizeof(ulp_insn_t);
ulp_process_macros_and_load(load_addr, program, &size);
ulp_run(load_addr);
```

The `program` array is an array of `ulp_insn_t`, i.e. ULP coprocessor instructions. Each `I_XXX` preprocessor define translates into a single 32-bit instruction. Arguments of these preprocessor defines can be register numbers (`R0` —`R3`) and literal constants. See *ULP coprocessor instruction defines* section for descriptions of instructions and arguments they take.

注解: Because some of the instruction macros expand to inline function calls, defining such array in global scope will cause the compiler to produce an "initializer element is not constant" error. To fix this error, move the definition of instructions array into local scope.

Load and store instructions use addresses expressed in 32-bit words. Address 0 corresponds to the first word of `RTC_SLOW_MEM` (which is address 0x50000000 as seen by the main CPUs).

To generate branch instructions, special `M_` preprocessor defines are used. `M_LABEL` define can be used to

define a branch target. Label identifier is a 16-bit integer. `M_Bxxx` defines can be used to generate branch instructions with target set to a particular label.

Implementation note: these `M_` preprocessor defines will be translated into two `ulp_insn_t` values: one is a token value which contains label number, and the other is the actual instruction. `ulp_process_macros_and_load` function resolves the label number to the address, modifies the branch instruction to use the correct address, and removes the the extra `ulp_insn_t` token which contains the label numer.

Here is an example of using labels and branches:

```
const ulp_insn_t program[] = {
    I_MOVI(R0, 34),         // R0 <- 34
    M_LABEL(1),             // label_1
    I_MOVI(R1, 32),         // R1 <- 32
    I_LD(R1, R1, 0),        // R1 <- RTC_SLOW_MEM[R1]
    I_MOVI(R2, 33),         // R2 <- 33
    I_LD(R2, R2, 0),        // R2 <- RTC_SLOW_MEM[R2]
    I_SUBR(R3, R1, R2),     // R3 <- R1 - R2
    I_ST(R3, R0, 0),        // R3 -> RTC_SLOW_MEM[R0 + 0]
    I_ADDI(R0, R0, 1),      // R0++
    M_BL(1, 64),            // if (R0 < 64) goto label_1
    I_HALT(),
};
RTC_SLOW_MEM[32] = 42;
RTC_SLOW_MEM[33] = 18;
size_t load_addr = 0;
size_t size = sizeof(program)/sizeof(ulp_insn_t);
ulp_process_macros_and_load(load_addr, program, &size);
ulp_run(load_addr);
```

Functions

esp_err_t **ulp_process_macros_and_load**(uint32_t *load_addr*, **const** ulp_insn_t **program*, size_t **psize*)

Resolve all macro references in a program and load it into RTC memory.

Return

- ESP_OK on success
- ESP_ERR_NO_MEM if auxiliary temporary structure can not be allocated
- one of ESP_ERR_ULP_xxx if program is not valid or can not be loaded

Parameters

- `load_addr`: address where the program should be loaded, expressed in 32-bit words
- `program`: ulp_insn_t array with the program
- `psize`: size of the program, expressed in 32-bit words

esp_err_t **ulp_run**(uint32_t *entry_point*)

Run the program loaded into RTC memory.

Return ESP_OK on success

Parameters

- `entry_point`: entry point, expressed in 32-bit words

Error codes

`ESP_ERR_ULP_BASE`

Offset for ULP-related error codes

`ESP_ERR_ULP_SIZE_TOO_BIG`

Program doesn't fit into RTC memory reserved for the ULP

`ESP_ERR_ULP_INVALID_LOAD_ADDR`

Load address is outside of RTC memory reserved for the ULP

`ESP_ERR_ULP_DUPLICATE_LABEL`

More than one label with the same number was defined

`ESP_ERR_ULP_UNDEFINED_LABEL`

Branch instructions references an undefined label

`ESP_ERR_ULP_BRANCH_OUT_OF_RANGE`

Branch target is out of range of B instruction (try replacing with BX)

ULP coprocessor registers

ULP co-processor has 4 16-bit general purpose registers. All registers have same functionality, with one exception. R0 register is used by some of the compare-and-branch instructions as a source register.

These definitions can be used for all instructions which require a register.

`R0`

general purpose register 0

`R1`

general purpose register 1

`R2`

general purpose register 2

R3

general purpose register 3

ULP coprocessor instruction defines

I_DELAY(cycles_)

Delay (nop) for a given number of cycles

I_HALT()

Halt the coprocessor.

This instruction halts the coprocessor, but keeps ULP timer active. As such, ULP program will be restarted again by timer. To stop the program and prevent the timer from restarting the program, use I_END(0) instruction.

I_END()

Stop ULP program timer.

This is a convenience macro which disables the ULP program timer. Once this instruction is used, ULP program will not be restarted anymore until ulp_run function is called.

ULP program will continue running after this instruction. To stop the currently running program, use I_HALT().

I_ST(reg_val, reg_addr, offset_)

Store value from register reg_val into RTC memory.

The value is written to an offset calculated by adding value of reg_addr register and offset_ field (this offset is expressed in 32-bit words). 32 bits written to RTC memory are built as follows:

- bits [31:21] hold the PC of current instruction, expressed in 32-bit words
- bits [20:16] = 5’ b1
- bits [15:0] are assigned the contents of reg_val

RTC_SLOW_MEM[addr + offset_] = { 5’ b0, insn_PC[10:0], val[15:0] }

I_LD(reg_dest, reg_addr, offset_)

Load value from RTC memory into reg_dest register.

Loads 16 LSBs from RTC memory word given by the sum of value in reg_addr and value of offset_.

I_WR_REG(reg, low_bit, high_bit, val)

Write literal value to a peripheral register

reg[high_bit : low_bit] = val This instruction can access RTC_CNTL_, RTC_IO_, SENS_, and RTC_I2C peripheral registers.

I_RD_REG(reg, low_bit, high_bit)

Read from peripheral register into R0

R0 = reg[high_bit : low_bit] This instruction can access RTC_CNTL_, RTC_IO_, SENS_, and RTC_I2C peripheral registers.

`I_BL`(pc_offset, imm_value)

Branch relative if R0 less than immediate value.

pc_offset is expressed in words, and can be from -127 to 127 imm_value is a 16-bit value to compare R0 against

`I_BGE`(pc_offset, imm_value)

Branch relative if R0 greater or equal than immediate value.

pc_offset is expressed in words, and can be from -127 to 127 imm_value is a 16-bit value to compare R0 against

`I_BXR`(reg_pc)

Unconditional branch to absolute PC, address in register.

reg_pc is the register which contains address to jump to. Address is expressed in 32-bit words.

`I_BXI`(imm_pc)

Unconditional branch to absolute PC, immediate address.

Address imm_pc is expressed in 32-bit words.

`I_BXZR`(reg_pc)

Branch to absolute PC if ALU result is zero, address in register.

reg_pc is the register which contains address to jump to. Address is expressed in 32-bit words.

`I_BXZI`(imm_pc)

Branch to absolute PC if ALU result is zero, immediate address.

Address imm_pc is expressed in 32-bit words.

`I_BXFR`(reg_pc)

Branch to absolute PC if ALU overflow, address in register

reg_pc is the register which contains address to jump to. Address is expressed in 32-bit words.

`I_BXFI`(imm_pc)

Branch to absolute PC if ALU overflow, immediate address

Address imm_pc is expressed in 32-bit words.

`I_ADDR`(reg_dest, reg_src1, reg_src2)

Addition: dest = src1 + src2

`I_SUBR`(reg_dest, reg_src1, reg_src2)

Subtraction: dest = src1 - src2

`I_ANDR`(reg_dest, reg_src1, reg_src2)

Logical AND: dest = src1 & src2

I_ORR(reg_dest, reg_src1, reg_src2)
Logical OR: dest = src1 | src2

I_MOVR(reg_dest, reg_src)
Copy: dest = src

I_LSHR(reg_dest, reg_src, reg_shift)
Logical shift left: dest = src << shift

I_RSHR(reg_dest, reg_src, reg_shift)
Logical shift right: dest = src >> shift

I_ADDI(reg_dest, reg_src, imm_)
Add register and an immediate value: dest = src1 + imm

I_SUBI(reg_dest, reg_src, imm_)
Subtract register and an immediate value: dest = src - imm

I_ANDI(reg_dest, reg_src, imm_)
Logical AND register and an immediate value: dest = src & imm

I_ORI(reg_dest, reg_src, imm_)
Logical OR register and an immediate value: dest = src | imm

I_MOVI(reg_dest, imm_)
Copy an immediate value into register: dest = imm

I_LSHI(reg_dest, reg_src, imm_)
Logical shift left register value by an immediate: dest = src << imm

I_RSHI(reg_dest, reg_src, imm_)
Logical shift right register value by an immediate: dest = val >> imm

M_LABEL(label_num)
Define a label with number label_num.

This is a macro which doesn' t generate a real instruction. The token generated by this macro is removed by ulp_process_macros_and_load function. Label defined using this macro can be used in branch macros defined below.

M_BL(label_num, imm_value)
Macro: branch to label label_num if R0 is less than immediate value.

This macro generates two ulp_insn_t values separated by a comma, and should be used when defining contents of ulp_insn_t arrays. First value is not a real instruction; it is a token which is removed by ulp_process_macros_and_load function.

M_BGE(label_num, imm_value)
Macro: branch to label label_num if R0 is greater or equal than immediate value

This macro generates two ulp_insn_t values separated by a comma, and should be used when defining

contents of ulp_insn_t arrays. First value is not a real instruction; it is a token which is removed by ulp_process_macros_and_load function.

`M_BX`(label_num)
Macro: unconditional branch to label

This macro generates two ulp_insn_t values separated by a comma, and should be used when defining contents of ulp_insn_t arrays. First value is not a real instruction; it is a token which is removed by ulp_process_macros_and_load function.

`M_BXZ`(label_num)
Macro: branch to label if ALU result is zero

This macro generates two ulp_insn_t values separated by a comma, and should be used when defining contents of ulp_insn_t arrays. First value is not a real instruction; it is a token which is removed by ulp_process_macros_and_load function.

`M_BXF`(label_num)
Macro: branch to label if ALU overflow

This macro generates two ulp_insn_t values separated by a comma, and should be used when defining contents of ulp_insn_t arrays. First value is not a real instruction; it is a token which is removed by ulp_process_macros_and_load function.

Defines

`RTC_SLOW_MEM`
RTC slow memory, 8k size

ULP（Ultra Low Power 超低功耗）协处理器是一种简单的有限状态机（FSM），可以在主处理器处于深度睡眠模式时，使用 ADC、温度传感器和外部 I2C 传感器执行测量操作。ULP 协处理器可以访问 RTC_SLOW_MEM 内存区域及 RTC_CNTL、RTC_IO、SARADC 等外设寄存器。ULP 协处理器使用 32 位固定宽度的指令，32 位内存寻址，配备 4 个 16 位通用寄存器。

5.18.4 安装工具链

ULP 协处理器代码是用汇编语言编写的，并使用 *binutils-esp32ulp* 工具链进行编译。

如果你已经按照快速入门指南 中的介绍安装好了 ESP-IDF 及其 CMake 构建系统，那么 ULP 工具链已经被默认安装到了你的开发环境中。

如果你的 ESP-IDF 仍在使用传统的基于 GNU Make 的构建系统，请参考*ULP 协处理器 (传统的 GNU Make)* 一文中的说明，完成工具链的安装。

5.18.5 编译 ULP 代码

若需要将 ULP 代码编译为某组件的一部分，则必须执行以下步骤：

1. 用汇编语言编写的 ULP 代码必须导入到一个或多个 *.S* 扩展文件中，且这些文件必须放在组件目录中一个独立的目录中，例如 *ulp/*。

2. 注册后从组件 CMakeLists.txt 中调用 `ulp_embed_binary` 示例如下:

```
...
idf_component_register()

set(ulp_app_name ulp_${COMPONENT_NAME})
set(ulp_s_sources ulp/ulp_assembly_source_file.S)
set(ulp_exp_dep_srcs "ulp_c_source_file.c")

ulp_embed_binary(${ulp_app_name} "${ulp_s_sources}" "${ulp_exp_dep_srcs}")
```

   上述第一个参数到 `ulp_embed_binary` 为 ULP 二进制文件命名。此名称也用于生成的其他文件，如：ELF 文件、.map 文件、头文件和链接器导出文件。第二个参数设置 ULP 程序集源文件。最后，第三个参数设置组件源文件列表，其中包括被生成的头文件。此列表用以建立正确的依赖项，并确保在构建过程会先生成再编译包含头文件的源文件。请参考下文，查看为 ULP 应用程序生成的头文件等相关概念。

3. 使用常规方法（例如 *idf.py app*）编译应用程序

   在内部，构建系统将按照以下步骤编译 ULP 程序：

   1. **通过 C 预处理器运行每个程序集文件 (foo.S)**。此步骤在组件编译目录中生成预处理的程序集文件 (foo.ulp.S)，同时生成依赖文件 (foo.ulp.d)。
   2. **通过汇编器运行预处理过的汇编源码**。此步骤会生成目标文件 (foo.ulp.o) 和清单 (foo.ulp.lst)。清单文件仅用于调试，不用于编译进程的后续步骤。
   3. **通过 C 预处理器运行链接器脚本模板**。模板位于 `components/ulp/ld` 目录中。
   4. **将目标文件链接到 ELF 输出文件** (`ulp_app_name.elf`)。此步骤生成的.map 文件 (`ulp_app_name.map`) 默认用于调试。
   5. **将 ELF 文件中的内容转储为二进制文件** (`ulp_app_name.bin`)，以便嵌入到应用程序中。
   6. 使用 `esp32ulp-elf-nm` 在 ELF 文件中 **生成全局符号列表** (`ulp_app_name.sym`)。
   7. **创建 LD 导出脚本和头文件** (`ulp_app_name.ld` 和 `ulp_app_name.h`)，包含来自 `ulp_app_name.sym` 的符号。此步骤可借助 `esp32ulp_mapgen.py` 工具来完成。
   8. **将生成的二进制文件添加到要嵌入应用程序的二进制文件列表中**。

5.18.6 访问 ULP 程序变量

在 ULP 程序中定义的全局符号也可以在主程序中使用。

例如，ULP 程序可以定义 measurement_count 变量，此变量可以定义程序从深度睡眠中唤醒芯片之前需要进行的 ADC 测量的次数:

```
                        .global measurement_count
measurement_count:      .long 0

                        /* later, use measurement_count */
                        move r3, measurement_count
                        ld r3, r3, 0
```

主程序需要在启动 ULP 程序之前初始化 measurement_count 变量，构建系统生成定义 ULP 编程中全局符号的 ${ULP_APP_NAME}.h 和 ${ULP_APP_NAME}.ld 文件，可以实现上述操作，这些文件包含在 ULP 程序中定义的所有全局符号，文件以 ulp_ 开头。

头文件包含对此类符号的声明:

```
extern uint32_t ulp_measurement_count;
```

注意，所有符号（包括变量、数组、函数）均被声明为 uint32_t。对于函数和数组，先获取符号地址，然后转换为适当的类型。

生成的链接器脚本文件定义了 RTC_SLOW_MEM 中的符号位置:

```
PROVIDE ( ulp_measurement_count = 0x50000060 );
```

如果要从主程序访问 ULP 程序变量，应先使用 include 语句包含生成的头文件，这样，就可以像访问常规变量一样访问 ulp 程序变量。操作如下:

```
#include "ulp_app_name.h"

// later
void init_ulp_vars() {
    ulp_measurement_count = 64;
}
```

注意，ULP 程序在 RTC 内存中只能使用 32 位字的低 16 位，因为寄存器是 16 位的，并且不具备从字的高位加载的指令。

同样，ULP 储存指令将寄存器值写入 32 位字的低 16 位中。高 16 位写入的值取决于储存指令的地址，因此在读取 ULP 写的变量时，主应用程序需要屏蔽高 16 位，例如:

```
printf("Last measurement value: %d\n", ulp_last_measurement & UINT16_MAX);
```

5.18.7 启动 ULP 程序

要运行 ULP 程序，主应用程序需要调用 `ulp_load_binary` 函数将 ULP 程序加载到 RTC 内存中，然后调用 `ulp_run` 函数，启动 ULP 程序。

注意，在 menuconfig 中必须启用 “Enable Ultra Low Power (ULP) Coprocessor” 选项，以便为 ULP 预留内存。” RTC slow memory reserved for coprocessor” 选项设置的值必须足够储存 ULP 代码和数据。如果应用程序组件包含多个 ULP 程序，则 RTC 内存必须足以容纳最大的程序。

每个 ULP 程序均以二进制 BLOB 的形式嵌入到 ESP-IDF 应用程序中。应用程序可以引用此 BLOB，并以下面的方式加载此 BLOB （假设 ULP_APP_NAME 已被定义为 `ulp_app_name`）：

```
extern const uint8_t bin_start[] asm("_binary_ulp_app_name_bin_start");
extern const uint8_t bin_end[]   asm("_binary_ulp_app_name_bin_end");

void start_ulp_program() {
    ESP_ERROR_CHECK( ulp_load_binary(
        0 /* load address, set to 0 when using default linker scripts */,
        bin_start,
        (bin_end - bin_start) / sizeof(uint32_t)) );
}
```

esp_err_t **ulp_load_binary**(uint32_t *load_addr*, **const** uint8_t **program_binary*, size_t *program_size*)

Load ULP program binary into RTC memory.

ULP program binary should have the following format (all values little-endian):

1. MAGIC, (value 0x00706c75, 4 bytes)
2. TEXT_OFFSET, offset of .text section from binary start (2 bytes)
3. TEXT_SIZE, size of .text section (2 bytes)
4. DATA_SIZE, size of .data section (2 bytes)
5. BSS_SIZE, size of .bss section (2 bytes)
6. (TEXT_OFFSET - 12) bytes of arbitrary data (will not be loaded into RTC memory)
7. .text section
8. .data section

Linker script in components/ulp/ld/esp32.ulp.ld produces ELF files which correspond to this format. This linker script produces binaries with load_addr == 0.

Return

- ESP_OK on success

- ESP_ERR_INVALID_ARG if load_addr is out of range
- ESP_ERR_INVALID_SIZE if program_size doesn't match (TEXT_OFFSET + TEXT_SIZE + DATA_SIZE)
- ESP_ERR_NOT_SUPPORTED if the magic number is incorrect

Parameters

- `load_addr`: address where the program should be loaded, expressed in 32-bit words
- `program_binary`: pointer to program binary
- `program_size`: size of the program binary

一旦上述程序加载到 RTC 内存后，应用程序即可启动此程序，并将入口点的地址传递给 `ulp_run` 函数:

```
ESP_ERROR_CHECK( ulp_run(&ulp_entry - RTC_SLOW_MEM) );
```

esp_err_t **ulp_run**(uint32_t *entry_point*)

Run the program loaded into RTC memory.

Return ESP_OK on success

Parameters

- `entry_point`: entry point, expressed in 32-bit words

上述生成的头文件 `${ULP_APP_NAME}.h` 声明了入口点符号。在 ULP 应用程序的汇编源代码中，此符号必须标记为 `.global`:

```
        .global entry
entry:
        /* code starts here */
```

5.18.8 ULP 程序流

ULP 协处理器由定时器启动，而调用 `ulp_run` 则可启动此定时器。定时器为 RTC_SLOW_CLK 的 Tick 事件计数（默认情况下，Tick 由内部 150 KHz 晶振器生成）。使用 `SENS_ULP_CP_SLEEP_CYCx_REG` 寄存器 (x = 0..4) 设置 Tick 数值。第一次启动 ULP 时，使用 `SENS_ULP_CP_SLEEP_CYC0_REG` 设置定时器 Tick 数值，之后，ULP 程序可以使用 `sleep` 指令来另外选择 `SENS_ULP_CP_SLEEP_CYCx_REG` 寄存器。

此应用程序可以调用 `ulp_set_wakeup_period` 函数来设置 ULP 定时器周期值 (SENS_ULP_CP_SLEEP_CYCx_REG, x = 0..4)。

esp_err_t **ulp_set_wakeup_period**(size_t *period_index*, uint32_t *period_us*)

Set one of ULP wakeup period values.

ULP coprocessor starts running the program when the wakeup timer counts up to a given value (called period). There are 5 period values which can be programmed into SENS_ULP_CP_SLEEP_CYCx_REG registers, x = 0..4. By default, wakeup timer will use the period set into SENS_ULP_CP_SLEEP_CYC0_REG, i.e. period number 0. ULP program code can use SLEEP instruction to select which of the SENS_ULP_CP_SLEEP_CYCx_REG should be used for subsequent wakeups.

However, please note that SLEEP instruction issued (from ULP program) while the system is in deep sleep mode does not have effect, and sleep cycle count 0 is used.

Note The ULP FSM requires two clock cycles to wakeup before being able to run the program. Then additional 16 cycles are reserved after wakeup waiting until the 8M clock is stable. The FSM also requires two more clock cycles to go to sleep after the program execution is halted. The minimum wakeup period that may be set up for the ULP is equal to the total number of cycles spent on the above internal tasks. For a default configuration of the ULP running at 150kHz it makes about 133us.

Return

- ESP_OK on success
- ESP_ERR_INVALID_ARG if period_index is out of range

Parameters

- `period_index`: wakeup period setting number (0 - 4)
- `period_us`: wakeup period, us

一旦定时器为所选的 `SENS_ULP_CP_SLEEP_CYCx_REG` 寄存器的 Tick 事件计数，ULP 协处理器就会启动，并调用 `ulp_run` 的入口点开始运行程序。

程序保持运行，直到遇到 `halt` 指令或非法指令。一旦程序停止，ULP 协处理器电源关闭，定时器再次启动。

如果想禁用定时器（有效防止 ULP 程序再次运行），请清除 `RTC_CNTL_STATE0_REG` 寄存器中的 `RTC_CNTL_ULP_CP_SLP_TIMER_EN` 位，可在 ULP 代码或主程序中进行以上操作。

5.19 ULP 协处理器 (传统的 GNU Make)

[English]

注解： ESP-IDF V4.0 及之后版本的默认构建系统为 CMake。本文档主要针对之前基于 GNU Make 的传统构建系统。请注意，未来，我们可能不会继续支持基于 GNU Make 的构建系统。

ULP（Ultra Low Power 超低功耗）协处理器是一种简单的有限状态机（FSM），可以在主处理器处于深度睡眠模式时，使用 ADC、温度传感器和外部 I2C 传感器执行测量操作。ULP 协处理器可以访问

RTC_SLOW_MEM 内存区域及 RTC_CNTL、RTC_IO、SARADC 等外设寄存器。ULP 协处理器使用 32 位固定宽度的指令，32 位内存寻址，配备 4 个 16 位通用寄存器。

5.19.1 安装工具链

ULP 协处理器代码是用汇编语言编写的，并使用 *binutils-esp32ulp toolchain* 进行编译。

1. 从 https://github.com/espressif/binutils-esp32ulp/releases 网站下载预编译的最新版本工具链二进制文件。
2. 将工具链提取至目录中，向工具链的 `bin/` 目录中添加路径至 `PATH` 环境变量中。

5.19.2 编译 ULP 代码

若需要将 ULP 代码编译为某组件的一部分，则必须执行以下步骤：

1. 用汇编语言编写的 ULP 代码必须导入到一个或多个 *.S* 扩展文件中，且这些文件必须放在组件目录中一个独立的目录中，例如 *ulp/*。
2. 修改组件 makefile，添加下列参数：

```
ULP_APP_NAME ?= ulp_$(COMPONENT_NAME)
ULP_S_SOURCES = $(COMPONENT_PATH)/ulp/ulp_source_file.S
ULP_EXP_DEP_OBJECTS := main.o
include $(IDF_PATH)/components/ulp/component_ulp_common.mk
```

   代码解释如下：

   ULP_APP_NAME 为生成的 ULP 应用程序命名，不带扩展名。此名称用于 ULP 应用程序的构建输出：ELF 文件、map 文件、二进制文件、生成的头文件和链接器导出文件。

   ULP_S_SOURCES 设置要传递给 ULP 汇编器的程序集文件列表。列表中的路径须是绝对路径，即以 `$(COMPONENT_PATH)` 开头的路径。如果列表中需列出多个文件，可使用函数 `$(addprefix)`。因为路径与组件构建目录相关，因而不需前置。

   ULP_EXP_DEP_OBJECTS 设置组件中目标文件名称的列表，其中包含生成的头文件。此列表用以建立正确的依赖项，并确保在此类文件被编译前生成头文件。ULP 应用程序头文件的定义请见下文。

   include $(IDF_PATH)/components/ulp/component_ulp_common.mk 包含常见的 ULP 编译步骤。为 ULP 目标文件、ELF 文件、二进制文件等定义编译目标。

3. 使用常规方法（例如 `idf.py build` 或 `idf.py app`）编译应用程序

   在编译过程中，构建系统将按照以下步骤编译 ULP 程序：

   1. **通过 C 预处理器运行每个程序集文件 (foo.S)**。此步骤在组件编译目录中生成预处理的程序集文件 (foo.ulp.pS)，同时生成依赖文件 (foo.ulp.d)。

2. **通过汇编器运行预处理过的汇编源码**。此步骤会生成目标文件（foo.ulp.o）和清单文件（foo.ulp.lst）。生成的清单文件用于调试，不用于构建过程的后续步骤。
3. **通过 C 预处理器运行链接器脚本模板**。模版位于 components/ulp/ld 目录中。
4. **将目标文件链接到 ELF 输出文件**（ulp_app_name.elf）。此步骤生成的.map 文件（ulp_app_name.map）可能会用于调试。
5. **将 ELF 文件中的内容转储为二进制文件**（ulp_app_name.bin），以便嵌入到应用程序中。
6. **使用 esp32ulp-elf-nm 在 ELF 文件中生成全局符号列表**（ulp_app_name.sym）。
7. **创建 LD 导出脚本和头文件**（ulp_app_name.ld 和 ulp_app_name.h），包含来自 ulp_app_name.sym 的符号。此步骤通过 esp32ulp_mapgen.py 工具来完成。
8. **将生成的二进制文件添加到要嵌入应用程序的二进制文件列表中**。

5.19.3 访问 ULP 程序变量

在 ULP 程序中定义的全局符号也可以在主程序中使用。

例如，ULP 程序可以定义 `measurement_count` 变量，此变量规定了芯片从深度睡眠模式唤醒之前，程序需要进行 ADC 测量的次数:

```
                        .global measurement_count
measurement_count:      .long 0

                        /* later, use measurement_count */
                        move r3, measurement_count
                        ld r3, r3, 0
```

主程序需要在启动 ULP 程序之前初始化 `measurement_count` 变量，构建系统通过生成定义 ULP 程序中全局符号的 `$(ULP_APP_NAME).h` 和 `$(ULP_APP_NAME).ld` 文件，可以实现上述操作；这些文件包含了 ULP 程序中所有全局符号，文件名以 `ulp_` 开头。

头文件包含对此类符号的声明:

```
extern uint32_t ulp_measurement_count;
```

注意，所有符号（包括变量、数组、函数）均被声明为 `uint32_t`。对于函数和数组，先获取符号地址，然后转换为适当的类型。

生成的链接器脚本文件定义了符号在 RTC_SLOW_MEM 中的位置:

```
PROVIDE ( ulp_measurement_count = 0x50000060 );
```

如果要从主程序访问 ULP 程序变量，应先使用 `include` 语句包含生成的头文件，这样，就可以像访问常规变量一样访问 ulp 程序变量。操作如下:

```
#include "ulp_app_name.h"

// later
void init_ulp_vars() {
    ulp_measurement_count = 64;
}
```

注意，ULP 程序只能使用 RTC 内存中 32 位字的低 16 位，因为寄存器是 16 位的，并且不存在从字的高位加载的指令。

同样，ULP 储存指令将寄存器值写入 32 位字的低 16 位中。写入高 16 位的值取决于存储指令的地址，因此在读取 ULP 写的变量时，主应用程序需要屏蔽高 16 位，例如:

```
printf("Last measurement value: %d\n", ulp_last_measurement & UINT16_MAX);
```

5.19.4 启动 ULP 程序

要运行 ULP 程序，主应用程序需要调用 `ulp_load_binary` 函数将 ULP 程序加载到 RTC 内存中，然后调用 `ulp_run` 函数，启动 ULP 程序。

注意，在 menuconfig 中必须启用 “Enable Ultra Low Power (ULP) Coprocessor” 选项，以便为 ULP 预留内存。” RTC slow memory reserved for coprocessor” 选项设置的值必须足够储存 ULP 代码和数据。如果应用程序组件包含多个 ULP 程序，则 RTC 内存必须足以容纳最大的程序。

每个 ULP 程序均以二进制 BLOB 的形式嵌入到 ESP-IDF 应用程序中。应用程序可以引用此 BLOB，并以下面的方式加载此 BLOB （假设 ULP_APP_NAME 已被定义为 `ulp_app_name`）：

```
extern const uint8_t bin_start[] asm("_binary_ulp_app_name_bin_start");
extern const uint8_t bin_end[]   asm("_binary_ulp_app_name_bin_end");

void start_ulp_program() {
    ESP_ERROR_CHECK( ulp_load_binary(
        0 /* load address, set to 0 when using default linker scripts */,
        bin_start,
        (bin_end - bin_start) / sizeof(uint32_t)) );
}
```

esp_err_t **ulp_load_binary**(uint32_t *load_addr*, **const** uint8_t **program_binary*, size_t *program_size*)

Load ULP program binary into RTC memory.

ULP program binary should have the following format (all values little-endian):

1. MAGIC, (value 0x00706c75, 4 bytes)

2. TEXT_OFFSET, offset of .text section from binary start (2 bytes)
3. TEXT_SIZE, size of .text section (2 bytes)
4. DATA_SIZE, size of .data section (2 bytes)
5. BSS_SIZE, size of .bss section (2 bytes)
6. (TEXT_OFFSET - 12) bytes of arbitrary data (will not be loaded into RTC memory)
7. .text section
8. .data section

Linker script in components/ulp/ld/esp32.ulp.ld produces ELF files which correspond to this format. This linker script produces binaries with load_addr == 0.

Return

- ESP_OK on success
- ESP_ERR_INVALID_ARG if load_addr is out of range
- ESP_ERR_INVALID_SIZE if program_size doesn't match (TEXT_OFFSET + TEXT_SIZE + DATA_SIZE)
- ESP_ERR_NOT_SUPPORTED if the magic number is incorrect

Parameters

- `load_addr`: address where the program should be loaded, expressed in 32-bit words
- `program_binary`: pointer to program binary
- `program_size`: size of the program binary

一旦上述程序加载到 RTC 内存后，应用程序即可启动此程序，并将入口点的地址传递给 `ulp_run` 函数:

```
ESP_ERROR_CHECK( ulp_run(&ulp_entry - RTC_SLOW_MEM) );
```

esp_err_t **ulp_run**(uint32_t *entry_point*)

Run the program loaded into RTC memory.

Return ESP_OK on success

Parameters

- `entry_point`: entry point, expressed in 32-bit words

上述生成的头文件 `$(ULP_APP_NAME).h` 声明了入口点符号。在 ULP 应用程序的汇编源代码中，此符号必须标记为 `.global`:

```
        .global entry
entry:
        /* code starts here */
```

5.19.5 ULP 程序流

ULP 协处理器由定时器启动，而调用 `ulp_run` 则可启动此定时器。定时器为 RTC_SLOW_CLK 的 Tick 事件计数（默认情况下，Tick 由内部 150 kHz 晶振器生成）。使用 `SENS_ULP_CP_SLEEP_CYCx_REG` 寄存器 (x = 0..4) 设置 Tick 数值。第一次启动 ULP 时，使用 `SENS_ULP_CP_SLEEP_CYC0_REG` 设置定时器 Tick 数值，之后，ULP 程序可以使用 `sleep` 指令来另外选择 `SENS_ULP_CP_SLEEP_CYCx_REG` 寄存器。

此应用程序可以调用 `ulp_set_wakeup_period` 函数来设置 ULP 定时器周期值 (SENS_ULP_CP_SLEEP_CYCx_REG, x = 0..4)。

esp_err_t **ulp_set_wakeup_period**(size_t *period_index*, uint32_t *period_us*)

Set one of ULP wakeup period values.

ULP coprocessor starts running the program when the wakeup timer counts up to a given value (called period). There are 5 period values which can be programmed into SENS_ULP_CP_SLEEP_CYCx_REG registers, x = 0..4. By default, wakeup timer will use the period set into SENS_ULP_CP_SLEEP_CYC0_REG, i.e. period number 0. ULP program code can use SLEEP instruction to select which of the SENS_ULP_CP_SLEEP_CYCx_REG should be used for subsequent wakeups.

However, please note that SLEEP instruction issued (from ULP program) while the system is in deep sleep mode does not have effect, and sleep cycle count 0 is used.

Note The ULP FSM requires two clock cycles to wakeup before being able to run the program. Then additional 16 cycles are reserved after wakeup waiting until the 8M clock is stable. The FSM also requires two more clock cycles to go to sleep after the program execution is halted. The minimum wakeup period that may be set up for the ULP is equal to the total number of cycles spent on the above internal tasks. For a default configuration of the ULP running at 150kHz it makes about 133us.

Return

- ESP_OK on success
- ESP_ERR_INVALID_ARG if period_index is out of range

Parameters

- `period_index`: wakeup period setting number (0 - 4)
- `period_us`: wakeup period, us

一旦定时器数值达到在所选的 SENS_ULP_CP_SLEEP_CYCx_REG 寄存器中设置的数值，ULP 协处理器就会启动，并调用 ulp_run 的入口点，开始运行程序。

程序保持运行，直到遇到 halt 指令或非法指令才停止。一旦程序停止，ULP 协处理器电源关闭，定时器再次启动。

如果想禁用定时器（禁用定时器可有效防止 ULP 程序再次运行），请清除 RTC_CNTL_STATE0_REG 寄存器中的 RTC_CNTL_ULP_CP_SLP_TIMER_EN 位，可在 ULP 代码或主程序中进行以上操作。

5.20 ESP32 中的单元测试

[English]

ESP-IDF 中附带了一个基于 Unity 的单元测试应用程序框架，且所有的单元测试用例分别保存在 ESP-IDF 仓库中每个组件的 test 子目录中。

5.20.1 添加常规测试用例

单元测试被添加在相应组件的 test 子目录中，测试用例写在 C 文件中，一个 C 文件可以包含多个测试用例。测试文件的名字要以“test”开头。

测试文件需要包含 unity.h 头文件，此外还需要包含待测试 C 模块需要的头文件。

测试用例需要通过 C 文件中特定的函数来添加，如下所示：

```
TEST_CASE("test name", "[module name]"
{
        // 在这里添加测试用例
}
```

- 第一个参数是字符串，用来描述当前测试。
- 第二个参数是字符串，用方括号中的标识符来表示，标识符用来对相关测试或具有特定属性的测试进行分组。

注解: 没有必要在每个测试用例中使用 UNITY_BEGIN() 和 UNITY_END() 来声明主函数的区域，unity_platform.c 会自动调用 UNITY_BEGIN()，然后运行测试用例，最后调用 UNITY_END()。

---

test 子目录应包含：ref：组件 *CMakeLists.txt <component-directories>*，因为他们本身就是一种组件。ESP-IDF 使用了 unity 测试框架，需要将其指定为组件的依赖项。通常，组件：ref：需要手动指定待编译的源文件 *<cmake-file-globbing>*；但是，对于测试组件来说，这个要求被放宽为仅建议将参数 SRC_DIRS 用于 idf_component_register。

总的来说，test 子目录下最小的 CMakeLists.txt 文件可能如下所示：

```
idf_component_register(SRC_DIRS "."
                       INCLUDE_DIRS "."
                       REQUIRES unity)
```

更多关于如何在 Unity 下编写测试用例的信息，请查阅 http://www.throwtheswitch.org/unity 。

5.20.2 添加多设备测试用例

常规测试用例会在一个 DUT（Device Under Test，在试设备）上执行. 但是，那些需要互相通信的组件（比如 GPIO、SPI）需要与其通信的其他设备，因此不能使用常规测试用例进行测试。多设备测试用例包括写入多个测试函数，并在多个 DUT 进行运行测试。

以下是一个多设备测试用例：

```
void gpio_master_test()
{
    gpio_config_t slave_config = {
            .pin_bit_mask = 1 << MASTER_GPIO_PIN,
            .mode = GPIO_MODE_INPUT,
    };
    gpio_config(&slave_config);
    unity_wait_for_signal("output high level");
    TEST_ASSERT(gpio_get_level(MASTER_GPIO_PIN) == 1);
}

void gpio_slave_test()
{
    gpio_config_t master_config = {
            .pin_bit_mask = 1 << SLAVE_GPIO_PIN,
            .mode = GPIO_MODE_OUTPUT,
    };
    gpio_config(&master_config);
    gpio_set_level(SLAVE_GPIO_PIN, 1);
    unity_send_signal("output high level");
}

TEST_CASE_MULTIPLE_DEVICES("gpio multiple devices test example", "[driver]", gpio_master_
↪test, gpio_slave_test);
```

宏 TEST_CASE_MULTIPLE_DEVICES 用来声明多设备测试用例，

- 第一个参数指定测试用例的名字。

- 第二个参数是测试用例的描述。
- 从第三个参数开始，可以指定最多 5 个测试函数，每个函数都是单独运行在一个 DUT 上的测试入口点。

在不同的 DUT 上运行的测试用例，通常会要求它们之间进行同步。我们提供 `unity_wait_for_signal` 和 `unity_send_signal` 这两个函数来使用 UART 去支持同步操作。如上例中的场景，slave 应该在在 master 设置好 GPIO 电平后再去读取 GPIO 电平，DUT 的 UART 终端会打印提示信息，并要求用户进行交互。

DUT1（master）终端：

```
Waiting for signal: [output high level]!
Please press "Enter" key once any board send this signal.
```

DUT2（slave）终端：

```
Send signal: [output high level]!
```

一旦 DUT2 发送了该信号，您需要在 DUT1 的终端按回车键，然后 DUT1 会从 `unity_wait_for_signal` 函数中解除阻塞，并开始更改 GPIO 的电平。

5.20.3 添加多阶段测试用例

常规的测试用例无需重启就会结束（或者仅需要检查是否发生了重启），可有些时候我们想在某些特定类型的重启事件后运行指定的测试代码，例如，我们想在深度睡眠唤醒后检查复位的原因是否正确。首先我们需要出发深度睡眠复位事件，然后检查复位的原因。为了实现这一点，我们可以定义多阶段测试用例来将这些测试函数组合在一起。

```
static void trigger_deepsleep(void)
{
    esp_sleep_enable_timer_wakeup(2000);
    esp_deep_sleep_start();
}

void check_deepsleep_reset_reason()
{
    RESET_REASON reason = rtc_get_reset_reason(0);
    TEST_ASSERT(reason == DEEPSLEEP_RESET);
}

TEST_CASE_MULTIPLE_STAGES("reset reason check for deepsleep", "[esp32]", trigger_
↪deepsleep, check_deepsleep_reset_reason);
```

多阶段测试用例向用户呈现了一组测试函数，它需要用户进行交互（选择用例并选择不同的阶段）来运行。

5.20.4 编译单元测试程序

按照 esp-idf 顶层目录的 README 文件中的说明进行操作，请确保 `IDF_PATH` 环境变量已经被设置指向了 esp-idf 的顶层目录。

切换到 `tools/unit-test-app` 目录下进行配置和编译：

- `idf.py menuconfig` - 配置单元测试程序。
- `idf.py -T all build` - 编译单元测试程序，测试每个组件 `test` 子目录下的用例。
- `idf.py -T xxx build` - 编译单元测试程序，测试指定的组件。
- `idf.py -T all -E xxxbuild` - 编译单元测试程序，测试所有（除开指定）的组件。（例如 `idf.py -T all -E ulp mbedtls build` - 编译所有的单元测试，不包括 `ulp` 和 `mbedtls` 组件。）

当编译完成时，它会打印出烧写芯片的指令。您只需要运行 `idf.py flash` 即可烧写所有编译输出的文件。

您还可以运行 `idf.py -T all flash` 或者 `idf.py -T xxx flash` 来编译并烧写，所有需要的文件都会在烧写之前自动重新编译。

使用 `menuconfig` 可以设置烧写测试程序所使用的串口。

5.20.5 运行单元测试

烧写完成后重启 ESP32，它将启动单元测试程序。

当单元测试应用程序空闲时，输入回车键，它会打印出测试菜单，其中包含所有的测试项目。

```
Here's the test menu, pick your combo:
(1)     "esp_ota_begin() verifies arguments" [ota]
(2)     "esp_ota_get_next_update_partition logic" [ota]
(3)     "Verify bootloader image in flash" [bootloader_support]
(4)     "Verify unit test app image" [bootloader_support]
(5)     "can use new and delete" [cxx]
(6)     "can call virtual functions" [cxx]
(7)     "can use static initializers for non-POD types" [cxx]
(8)     "can use std::vector" [cxx]
(9)     "static initialization guards work as expected" [cxx]
(10)    "global initializers run in the correct order" [cxx]
(11)    "before scheduler has started, static initializers work correctly" [cxx]
(12)    "adc2 work with wifi" [adc]
(13)    "gpio master/slave test example" [ignore][misc][test_env=UT_T2_1][multi_device]
        (1)     "gpio_master_test"
        (2)     "gpio_slave_test"
(14)    "SPI Master clockdiv calculation routines" [spi]
```

(下页继续)

(续上页)

```
(15)    "SPI Master test" [spi][ignore]
(16)    "SPI Master test, interaction of multiple devs" [spi][ignore]
(17)    "SPI Master no response when switch from host1 (HSPI) to host2 (VSPI)" [spi]
(18)    "SPI Master DMA test, TX and RX in different regions" [spi]
(19)    "SPI Master DMA test: length, start, not aligned" [spi]
(20)    "reset reason check for deepsleep" [esp32][test_env=UT_T2_1][multi_stage]
        (1)     "trigger_deepsleep"
        (2)     "check_deepsleep_reset_reason"
```

常规测试用例会打印用例名字和描述，主从测试用例还会打印子菜单（已注册的测试函数的名字）。

可以输入以下任意一项来运行测试用例：

- 引号中的测试用例的名字，运行单个测试用例。
- 测试用例的序号，运行单个测试用例。
- 方括号中的模块名字，运行指定模块所有的测试用例。
- 星号，运行所有测试用例。

[multi_device] 和 [multi_stage] 标签告诉测试运行者该用例是多设备测试还是多阶段测试。这些标签由 TEST_CASE_MULTIPLE_STAGES 和 TEST_CASE_MULTIPLE_DEVICES 宏自动生成。

一旦选择了多设备测试用例，它会打印一个子菜单：

```
Running gpio master/slave test example...
gpio master/slave test example
        (1)     "gpio_master_test"
        (2)     "gpio_slave_test"
```

您需要输入数字以选择在 DUT 上运行的测试。

与多设备测试用例相似，多阶段测试用例也会打印子菜单：

```
Running reset reason check for deepsleep...
reset reason check for deepsleep
        (1)     "trigger_deepsleep"
        (2)     "check_deepsleep_reset_reason"
```

第一次执行此用例时，输入 1 来运行第一阶段（触发深度睡眠）。在重启 DUT 并再次选择运行此用例后，输入 2 来运行第二阶段。只有在最后一个阶段通过并且之前所有的阶段都成功触发了复位的情况下，该测试才算通过。

5.21 单元测试（传统的 GNU Make）

[English]

注解： ESP-IDF V4.0 及之后版本的默认构建系统为 CMake。本文档主要针对之前基于 GNU Make 的传统构建系统。请注意，未来，我们可能不会继续支持基于 GNU Make 的构建系统。

ESP-IDF 中附带了一个基于 Unity 的单元测试应用程序框架，且所有的单元测试用例分别保存在 ESP-IDF 仓库中每个组件的 test 子目录中。

5.21.1 添加常规测试用例

单元测试被添加在相应组件的 test 子目录中，测试用例写在 C 文件中，一个 C 文件可以包含多个测试用例。测试文件的名字要以“test”开头。

测试文件需要包含 unity.h 头文件，此外还需要包含待测试 C 模块需要的头文件。

测试用例需要通过 C 文件中特定的函数来添加，如下所示：

```
TEST_CASE("test name", "[module name]"
{
        // 在这里添加测试用例
}
```

第一个参数是字符串，用来描述当前测试。第二个参数是字符串，用方括号中的标识符来表示，标识符用来对相关测试或具有特定属性的测试进行分组。

注解： 没有必要在每个测试用例中使用 UNITY_BEGIN() 和 UNITY_END() 来声明主函数的区域，unity_platform.c 会自动调用 UNITY_BEGIN()，然后运行测试用例，最后调用 UNITY_END()。

每一个 测试子目录下都需包含 component.mk 文件，并且其中至少要包含如下的一行代码：

```
COMPONENT_ADD_LDFLAGS = -Wl,--whole-archive -l$(COMPONENT_NAME) -Wl,--no-whole-archive
```

更多关于如何在 Unity 下编写测试用例的信息，请查阅 http://www.throwtheswitch.org/unity 。

5.21.2 添加多设备测试用例

常规测试用例会在一个 DUT（Device Under Test，在试设备）上执行. 但是，那些需要互相通信的组件（比如 GPIO、SPI）需要与其通信的其他设备，因此不能使用常规测试用例进行测试。多设备测试用例包括写入多个测试函数，并在多个 DUT 进行运行测试。

以下是一个多设备测试用例：

```
void gpio_master_test()
{
    gpio_config_t slave_config = {
            .pin_bit_mask = 1 << MASTER_GPIO_PIN,
            .mode = GPIO_MODE_INPUT,
    };
    gpio_config(&slave_config);
    unity_wait_for_signal("output high level");
    TEST_ASSERT(gpio_get_level(MASTER_GPIO_PIN) == 1);
}

void gpio_slave_test()
{
    gpio_config_t master_config = {
            .pin_bit_mask = 1 << SLAVE_GPIO_PIN,
            .mode = GPIO_MODE_OUTPUT,
    };
    gpio_config(&master_config);
    gpio_set_level(SLAVE_GPIO_PIN, 1);
    unity_send_signal("output high level");
}

TEST_CASE_MULTIPLE_DEVICES("gpio multiple devices test example", "[driver]", gpio_master_
↪test, gpio_slave_test);
```

宏 TEST_CASE_MULTIPLE_DEVICES 用来声明多设备测试用例，

- 第一个参数指定测试用例的名字。
- 第二个参数是测试用例的描述。
- 从第三个参数开始，可以指定最多 5 个测试函数，每个函数都是单独运行在一个 DUT 上的测试入口点。

在不同的 DUT 上运行的测试用例，通常会要求它们之间进行同步。我们提供 unity_wait_for_signal 和 unity_send_signal 这两个函数来使用 UART 去支持同步操作。如上例中的场景，slave 应该在在 master 设置好 GPIO 电平后再去读取 GPIO 电平，DUT 的 UART 终端会打印提示信息，并要求用户进行交互。

DUT1（master）终端：

```
Waiting for signal: [output high level]!
Please press "Enter" key once any board send this signal.
```

DUT2（slave）终端：

```
Send signal: [output high level]!
```

一旦 DUT2 发送了该信号，您需要在 DUT1 按回车键，然后 DUT1 会从 unity_wait_for_signal 函数中解除阻塞，并开始更改 GPIO 的电平。

信号也可以用来在不同设备之间传递参数。例如，DUT1 希望得到 DUT2 的 MAC 地址来进行蓝牙连接，这时，可使用 unity_wait_for_signal_param 以及 unity_send_signal_param。

DUT1 终端:

```
Waiting for signal: [dut2 mac address]!
Please input parameter value from any board send this signal and press "Enter" key.
```

DUT2 终端:

```
Send signal: [dut2 mac address][10:20:30:40:50:60]!
```

一旦 DUT2 发送信号，您需要在 DUT1 输入 10:20:30:40:50:60 并按回车键，然后 DUT1 会获取 DUT2 的 MAC 地址字符串，并从 unity_wait_for_signal_param 函数中解除阻塞，然后蓝牙连接 DUT2。

5.21.3 添加多阶段测试用例

常规的测试用例无需重启就会结束（或者仅需要检查是否发生了重启），可有些时候我们想在某些特定类型的重启事件后运行指定的测试代码，例如，我们想在深度睡眠唤醒后检查复位的原因是否正确。首先我们需要出发深度睡眠复位事件，然后检查复位的原因。为了实现这一点，我们可以定义多阶段测试用例来将这些测试函数组合在一起。

```
static void trigger_deepsleep(void)
{
    esp_sleep_enable_timer_wakeup(2000);
    esp_deep_sleep_start();
}

void check_deepsleep_reset_reason()
{
    RESET_REASON reason = rtc_get_reset_reason(0);
    TEST_ASSERT(reason == DEEPSLEEP_RESET);
}

TEST_CASE_MULTIPLE_STAGES("reset reason check for deepsleep", "[esp32]", trigger_
↪deepsleep, check_deepsleep_reset_reason);
```

多阶段测试用例向用户呈现了一组测试函数，它需要用户进行交互（选择用例并选择不同的阶段）来运行。

5.21.4 编译单元测试程序

按照 esp-idf 顶层目录的 README 文件中的说明进行操作，请确保 IDF_PATH 环境变量已经被设置指向了 esp-idf 的顶层目录。

切换到 tools/unit-test-app 目录下进行配置和编译：

- make menuconfig - 配置单元测试程序。
- make TESTS_ALL=1 - 编译单元测试程序，测试每个组件 test 子目录下的用例。
- make TEST_COMPONENTS='xxx' - 编译单元测试程序，测试指定的组件。
- make TESTS_ALL=1 TEST_EXCLUDE_COMPONENTS='xxx' - 编译单元测试程序，测试所有（除开指定）的组件。（例如 make TESTS_ALL=1 TEST_EXCLUDE_COMPONENTS='ulp mbedtls' - 编译所有的单元测试，不包括 ulp 和 mbedtls 组件。）

当编译完成时，它会打印出烧写芯片的指令。您只需要运行 make flash 即可烧写所有编译输出的文件。

您还可以运行 make flash TESTS_ALL=1 或者 make TEST_COMPONENTS='xxx' 来编译并烧写，所有需要的文件都会在烧写之前自动重新编译。

使用 menuconfig 可以设置烧写测试程序所使用的串口。

5.21.5 运行单元测试

烧写完成后重启 ESP32，它将启动单元测试程序。

当单元测试应用程序空闲时，输入回车键，它会打印出测试菜单，其中包含所有的测试项目。

```
Here's the test menu, pick your combo:
(1)     "esp_ota_begin() verifies arguments" [ota]
(2)     "esp_ota_get_next_update_partition logic" [ota]
(3)     "Verify bootloader image in flash" [bootloader_support]
(4)     "Verify unit test app image" [bootloader_support]
(5)     "can use new and delete" [cxx]
(6)     "can call virtual functions" [cxx]
(7)     "can use static initializers for non-POD types" [cxx]
(8)     "can use std::vector" [cxx]
(9)     "static initialization guards work as expected" [cxx]
(10)    "global initializers run in the correct order" [cxx]
(11)    "before scheduler has started, static initializers work correctly" [cxx]
(12)    "adc2 work with wifi" [adc]
(13)    "gpio master/slave test example" [ignore][misc][test_env=UT_T2_1][multi_device]
        (1)     "gpio_master_test"
        (2)     "gpio_slave_test"
```

（下页继续）

(续上页)

```
(14)    "SPI Master clockdiv calculation routines" [spi]
(15)    "SPI Master test" [spi][ignore]
(16)    "SPI Master test, interaction of multiple devs" [spi][ignore]
(17)    "SPI Master no response when switch from host1 (HSPI) to host2 (VSPI)" [spi]
(18)    "SPI Master DMA test, TX and RX in different regions" [spi]
(19)    "SPI Master DMA test: length, start, not aligned" [spi]
(20)    "reset reason check for deepsleep" [esp32][test_env=UT_T2_1][multi_stage]
        (1)     "trigger_deepsleep"
        (2)     "check_deepsleep_reset_reason"
```

常规测试用例会打印用例名字和描述，主从测试用例还会打印子菜单（已注册的测试函数的名字）。

可以输入以下任意一项来运行测试用例：

- 引号中的测试用例的名字（例如 "esp_ota_begin() verifies arguments"），运行单个测试用例。
- 测试用例的序号（例如 1），运行单个测试用例。
- 方括号中的模组名称（例如 [cxx]），运行指定模组的所有的测试用例。
- 星号 (*)，运行所有测试用例。

[multi_device] 和 [multi_stage] 标签告诉测试运行者该用例是多设备测试还是多阶段测试。这些标签由 TEST_CASE_MULTIPLE_STAGES 和 TEST_CASE_MULTIPLE_DEVICES 宏自动生成。

一旦选择了多设备测试用例，它会打印一个子菜单：

```
Running gpio master/slave test example...
gpio master/slave test example
        (1)     "gpio_master_test"
        (2)     "gpio_slave_test"
```

您需要输入数字以选择在 DUT 上运行的测试。

与多设备测试用例相似，多阶段测试用例也会打印子菜单：

```
Running reset reason check for deepsleep...
reset reason check for deepsleep
        (1)     "trigger_deepsleep"
        (2)     "check_deepsleep_reset_reason"
```

第一次执行此用例时，输入 1 来运行第一阶段（触发深度睡眠）。在重启 DUT 并再次选择运行此用例后，输入 2 来运行第二阶段。只有在最后一个阶段通过并且之前所有的阶段都成功触发了复位的情况下，该测试才算通过。

5.22 控制台终端

[English]

ESP-IDF 提供了 `console` 组件，它包含了开发基于串口的交互式控制终端所需要的所有模块，主要支持以下功能：

- 行编辑，由 linenoise 库具体实现，它支持处理退格键和方向键，支持回看命令的历史记录，支持命令的自动补全和参数提示。
- 将命令行拆分为参数列表。
- 参数解析，由 argtable3 库具体实现，它支持解析 GNU 样式的命令行参数。
- 用于注册和调度命令的函数。

这些功能模块可以一起使用也可以独立使用，例如仅使用行编辑和命令注册的功能，然后使用 `getopt` 函数或者自定义的函数来实现参数解析，而不是直接使用 argtable3 库。同样地，还可以使用更简单的命令输入方法（比如 `fgets` 函数）和其他用于命令分割和参数解析的方法。

5.22.1 行编辑

行编辑功能允许用户通过按键输入来编辑命令，使用退格键删除符号，使用左/右键在命令中移动光标，使用上/下键导航到之前输入的命令，使用制表键（“Tab”）来自动补全命令。

注解： 此功能依赖于终端应用程序对 ANSI 转移符的支持，显示原始 UART 数据的串口监视器不能与行编辑库一同使用。如果运行 get_started/console 示例程序的时候观察到的输出结果是 `[6n` 或者类似的转义字符而不是命令行提示符 `[esp32]>` 时，就表明当前的串口监视器不支持 ANSI 转移字符。已知可用的串口监视程序有 GNU screen，minicom 和 idf_monitor.py（可以通过在项目目录下执行 `idf.py monitor` 来调用）。

前往这里可以查看 linenoise 库提供的所有函数的描述。

配置

Linenoise 库不需要显式地初始化，但是在调用行编辑函数之前，可能需要对某些配置的默认值稍作修改。

`linenoiseClearScreen`

使用转移字符清除终端屏幕，并将光标定位在左上角。

`linenoiseSetMultiLine`

在单行和多行编辑模式之间进行切换。单行模式下，如果命令的长度超过终端的宽度，会在行内滚动命令文本以显示文本的结尾，在这种情况下，文本的开头部分会被隐藏。单行模式在每次按下按键时发送给屏幕刷新的数据比较少，与多行模式相比更不容易发生故障。另一方面，在单行模式下编辑命令和复制命令将变得更加困难。默认情况下开启的是单行模式。

主循环

`linenoise`

在大多数情况下，控制台应用程序都会具有相同的工作形式——在某个循环中不断读取输入的内容，然后解析再处理。`linenoise` 是专门用来获取用户按键输入的函数，当回车键被按下后会便返回完整的一行内容。因此可以用它来完成前面循环中的“读取”任务。

`linenoiseFree`

必须调用此函数才能释放从 `linenoise` 函数获取的命令行缓冲。

提示和补全

`linenoiseSetCompletionCallback`

当用户按下制表键时，linenoise 会调用 **补全回调函数**，该回调函数会检查当前已经输入的内容，然后调用 `linenoiseAddCompletion` 函数来提供所有可能的补全后的命令列表。启用补全功能，需要事先调用 `linenoiseSetCompletionCallback` 函数来注册补全回调函数。

`console` 组件提供了一个现成的函数来为注册的命令提供补全功能 `esp_console_get_completion`（见后文）。

`linenoiseAddCompletion`

补全回调函数会通过调用此函数来通知 linenoise 库当前键入命令所有可能的补全结果。

`linenoiseSetHintsCallback`

每当用户的输入改变时，linenoise 就会调用此回调函数，检查到目前为止输入的命令行内容，然后提供带有提示信息的字符串（例如命令参数列表），然后会在同一行上用不同的颜色显示出该文本。

`linenoiseSetFreeHintsCallback`

如果 **提示回调函数**返回的提示字符串是动态分配的或者需要以其它方式回收，就需要使用 `linenoiseSetFreeHintsCallback` 注册具体的清理函数。

历史记录

`linenoiseHistorySetMaxLen`

该函数设置要保留在内存中的最近输入的命令的数量。用户通过使用向上/向下箭头来导航历史记录。

`linenoiseHistoryAdd`

Linenoise 不会自动向历史记录中添加命令，应用程序需要调用此函数来将命令字符串添加到历史记录中。

`linenoiseHistorySave`

该函数将命令的历史记录从 RAM 中保存为文本文件，例如保存到 SD 卡或者 Flash 的文件系统中。

`linenoiseHistoryLoad`

与 `linenoiseHistorySave` 相对应，从文件中加载历史记录。

`linenoiseHistoryFree`

释放用于存储命令历史记录的内存。当使用完 linenoise 库后需要调用此函数。

5.22.2 将命令行拆分成参数列表

`console` 组件提供 `esp_console_split_argv` 函数来将命令行字符串拆分为参数列表。该函数会返回参数的数量（`argc`）和一个指针数组，该指针数组可以作为 `argv` 参数传递给任何接受 `argc`，`argv` 格式参数的函数。

根据以下规则来将命令行拆分成参数列表：

- 参数由空格分隔
- 如果参数本身需要使用空格，可以使用 `\`（反斜杠）对它们进行转义
- 其它能被识别的转义字符有 `\\`（显示反斜杠本身）和 `\"`（显示双引号）
- 可以使用双引号来引用参数，引号只可能出现在参数的开头和结尾。参数中的引号必须如上所述进行转移。参数周围的引号会被 `esp_console_split_argv` 函数删除

示例：

- `abc def 1 20 .3` [`abc`, `def`, `1`, `20`, `.3`]
- `abc "123 456" def` [`abc`, `123 456`, `def`]
- `` `a\ b\\c\" `` [`a b\c"`]

5.22.3 参数解析

对于参数解析，`console` 组件使用 argtable3 库。有关 argtable3 的介绍请查看 教程 或者 Github 仓库中的 示例代码 。

5.22.4 命令的注册与调度

`console` 组件包含了一些工具函数，用来注册命令，将用户输入的命令和已经注册的命令进行匹配，使用命令行输入的参数调用命令。

应用程序首先调用 `esp_console_init` 来初始化命令注册模块，然后调用 `esp_console_cmd_register` 函数注册命令处理程序。

对于每个命令，应用程序需要提供以下信息（需要以 `esp_console_cmd_t` 结构体的形式给出）：

- 命令名字（不含空格的字符串）
- 帮助文档，解释该命令的用途
- 可选的提示文本，列出命令的参数。如果应用程序使用 `Argtable3` 库来解析参数，则可以通过提供指向 argtable 参数定义结构体的指针来自动生成提示文本
- 命令处理函数

命令注册模块还提供了其它函数：

`esp_console_run`

> 该函数接受命令行字符串，使用 `esp_console_split_argv` 函数将其拆分为 argc/argv 形式的参数列表，在已经注册的组件列表中查找命令，如果找到，则执行其对应的处理程序。

`esp_console_register_help_command`

> 将 `help` 命令添加到已注册命令列表中，此命令将会以列表的方式打印所有注册的命令及其参数和帮助文本。

`esp_console_get_completion`

> 与 linenoise 库中的 `linenoiseSetCompletionCallback` 一同使用的回调函数，根据已经注册的命令列表为 linenoise 提供补全功能。

`esp_console_get_hint`

> 与 linenoise 库中 `linenoiseSetHintsCallback` 一同使用的回调函数，为 linenoise 提供已经注册的命令的参数提示功能。

5.22.5 示例

`examples/system/console` 目录下提供了 `console` 组件的示例应用程序，展示了具体的使用方法。该示例介绍了如何初始化 UART 和 VFS 的功能，设置 linenoise 库，从 UART 中读取命令并加以处理，然后将历史命令存储到 Flash 中。更多信息，请参阅示例代码目录中的 README.md 文件。

5.23 ESP32 ROM console

When an ESP32 is unable to boot from flash ROM (and the fuse disabling it hasn’ t been blown), it boots into a rom console. The console is based on TinyBasic, and statements entered should be in the form of BASIC statements. As is common in the BASIC language, without a preceeding line number, commands entered are executed immediately; lines with a prefixed line number are stored as part of a program.

5.23.1 Full list of supported statements and functions

System

- BYE - *exits Basic, reboots ESP32, retries booting from flash*
- END - *stops execution from the program, also "STOP"*
- MEM - *displays memory usage statistics*
- NEW - *clears the current program*
- RUN - *executes the current program*

IO, Documentation

- PEEK(address) - *get a 32-bit value from a memory address*
- POKE - *write a 32-bit value to memory*
- USR(addr, arg1, ..) - *Execute a machine language function*
- PRINT expression - *print out the expression, also "?"*
- PHEX expression - *print expression as a hex number*
- REM stuff - *remark/comment, also "'"*

Expressions, Math

- A=V, LET A=V - *assign value to a variable*
- +, -, *, / - *Math*
- <,<=,=,<>,!=,>=,> - *Comparisons*
- ABS(expression) - *returns the absolute value of the expression*
- RSEED(v) - *sets the random seed to v*
- RND(m) - *returns a random number from 0 to m*
- A=1234 - * Assign a decimal value*
- A=&h1A2 - * Assign a hex value*
- A=&b1001 - *Assign a binary value*

Control

- IF expression statement - *perform statement if expression is true*
- FOR variable = start TO end - *start for block*
- FOR variable = start TO end STEP value - *start for block with step*

- NEXT - *end of for block*
- GOTO linenumber - *continue execution at this line number*
- GOSUB linenumber - *call a subroutine at this line number*
- RETURN - *return from a subroutine*
- DELAY - *Delay a given number of milliseconds*

Pin IO

- IODIR - *Set a GPIO-pin as an output (1) or input (0)*
- IOSET - *Set a GPIO-pin, configured as output, to high (1) or low (0)*
- IOGET - *Get the value of a GPIO-pin*

5.23.2 Example programs

Here are a few example commands and programs to get you started···

Read UART_DATE register of uart0

```
> PHEX PEEK(&h3FF40078)
15122500
```

Set GPIO2 using memory writes to GPIO_OUT_REG

Note: you can do this easier with the IOSET command

```
> POKE &h3FF44004,PEEK(&h3FF44004) OR &b100
```

Get value of GPIO0

```
> IODIR 0,0
> PRINT IOGET(0)
0
```

Blink LED

Hook up an LED between GPIO2 and ground. When running the program, the LED should blink 10 times.

```
10 IODIR 2,1
20 FOR A=1 TO 10
30 IOSET 2,1
40 DELAY 250
50 IOSET 2,0
60 DELAY 250
70 NEXT A
RUN
```

5.23.3 Credits

The ROM console is based on "TinyBasicPlus" by Mike Field and Scott Lawrence, which is based on "68000 TinyBasic" by Gordon Brandly

5.24 RF calibration

ESP32 supports three RF calibration methods during RF initialization:

1. Partial calibration
2. Full calibration
3. No calibration

5.24.1 Partial calibration

During RF initialization, the partial calibration method is used by default for RF calibration. It is done based on the full calibration data which is stored in the NVS. To use this method, please go to `menuconfig` and enable *CONFIG_ESP32_PHY_CALIBRATION_AND_DATA_STORAGE*.

5.24.2 Full calibration

Full calibration is triggered in the follwing conditions:

1. NVS does not exist.
2. The NVS partition to store calibration data is erased.
3. Hardware MAC address is changed.
4. PHY library version is changed.
5. The RF calibration data loaded from the NVS partition is broken.

It takes about 100ms more than partial calibration. If boot duration is not critical, it is suggested to use the full calibration method. To switch to the full calibration method, go to `menuconfig` and disable *CONFIG_ESP32_PHY_CALIBRATION_AND_DATA_STORAGE*. If you use the default method of RF calibration, there are two ways to add the function of triggering full calibration as a last-resort remedy.

1. Erase the NVS partition if you don' t mind all of the data stored in the NVS partition is erased. That is indeed the easiest way.
2. Call API `esp_phy_erase_cal_data_in_nvs()` before initializing WiFi and BT/BLE based on some conditions (e.g. an option provided in some diagnostic mode). In this case, only phy namespace of the NVS partition is erased.

5.24.3 No calibration

No calibration method is only used when ESP32 wakes up from deep sleep.

5.24.4 PHY initialization data

The PHY initialization data is used for RF calibration. There are two ways to get the PHY initialization data.

One is the default initialization data which is located in the header file components/esp_wifi/esp32/include/phy_init_data.h. It is embedded into the application binary after compiling and then stored into read-only memory (DROM). To use the default initialization data, please go to `menuconfig` and disable *CONFIG_ESP32_PHY_INIT_DATA_IN_PARTITION*.

Another is the initialization data which is stored in a partition. When using a custom partition table, make sure that PHY data partition is included (type: *data*, subtype: *phy*). With default partition table, this is done automatically. If initialization data is stored in a partition, it has to be flashed there, otherwise runtime error will occur. To switch to the initialization data stored in a partition, go to `menuconfig` and enable *CONFIG_ESP32_PHY_INIT_DATA_IN_PARTITION*.

5.25 Wi-Fi Driver

5.25.1 ESP32 Wi-Fi Feature List

- Support Station-only mode, AP-only mode, Station/AP-coexistence mode
- Support IEEE-802.11B, IEEE-802.11G, IEEE802.11N and APIs to configure the protocol mode
- Support WPA/WPA2/WPA2-Enterprise and WPS
- Support AMPDU, HT40, QoS and other key features
- Support Modem-sleep

- Support an Espressif-specific protocol which, in turn, supports up to **1 km** of data traffic
- Up to 20 MBit/sec TCP throughput and 30 MBit/sec UDP throughput over the air
- Support Sniffer
- Support set fast_crypto algorithm and normal algorithm switch which used in wifi connect
- Support both fast scan and all channel scan feature
- Support multiple antennas
- Support channel state information

5.25.2 How To Write a Wi-Fi Application

Preparation

Generally, the most effective way to begin your own Wi-Fi application is to select an example which is similar to your own application, and port the useful part into your project. It is not a MUST but it is strongly recommended that you take some time to read this article first, especially if you want to program a robust Wi-Fi application. This article is supplementary to the Wi-Fi APIs/Examples. It describes the principles of using the Wi-Fi APIs, the limitations of the current Wi-Fi API implementation, and the most common pitfalls in using Wi-Fi. This article also reveals some design details of the Wi-Fi driver. We recommend that you become familiar at least with the following sections: <*ESP32 Wi-Fi API Error Code*>, <*ESP32 Wi-Fi Programming Model*>, and <*ESP32 Wi-Fi Event Description*>.

Setting Wi-Fi Compile-time Options

Refer to <*Wi-Fi Menuconfig*>

Init Wi-Fi

Refer to <*ESP32 Wi-Fi Station General Scenario*>, <*ESP32 Wi-Fi AP General Scenario*>.

Start/Connect Wi-Fi

Refer to <*ESP32 Wi-Fi Station General Scenario*>, <*ESP32 Wi-Fi AP General Scenario*>.

Event-Handling

Generally, it is easy to write code in "sunny-day" scenarios, such as <*WIFI_EVENT_STA_START*>, <*WIFI_EVENT_STA_CONNECTED*> etc. The hard part is to write routines in "rainy-day" scenarios, such as <*WIFI_EVENT_STA_DISCONNECTED*> etc. Good handling of "rainy-day" scenarios is fundamental to robust Wi-Fi applications. Refer to <*ESP32 Wi-Fi Event Description*>, <*ESP32 Wi-Fi*

Station General Scenario>, <*ESP32 Wi-Fi AP General Scenario*>. See also *an overview of event handling in ESP-IDF*.

Write Error-Recovery Routines Correctly at All Times

Just like the handling of "rainy-day" scenarios, a good error-recovery routine is also fundamental to robust Wi-Fi applications. Refer to <*ESP32 Wi-Fi API Error Code*>

5.25.3 ESP32 Wi-Fi API Error Code

All of the ESP32 Wi-Fi APIs have well-defined return values, namely, the error code. The error code can b

- No errors, e.g. ESP_OK means that the API returns successfully
- Recoverable errors, such as ESP_ERR_NO_MEM, etc.
- Non-recoverable, non-critical errors
- Non-recoverable, critical errors

Whether the error is critical or not depends on the API and the application scenario, and it is defined by the API user.

The primary principle to write a robust application with Wi-Fi API is to always check the error code and write the error-handling code. Generally, the error-handling code can be used:

- for recoverable errors, in which case you can write a recoverable-error code. For example, when esp_wifi_start returns ESP_ERR_NO_MEM, the recoverable-error code vTaskDelay can be called, in order to get a microseconds' delay for another try.
- for non-recoverable, yet non-critical, errors, in which case printing the error code is a good method for error handling.
- for non-recoverable, critical errors, in which case "assert" may be a good method for error handling. For example, if esp_wifi_set_mode returns ESP_ERR_WIFI_NOT_INIT, it means that the Wi-Fi driver is not initialized by esp_wifi_init successfully. You can detect this kind of error very quickly in the application development phase.

In esp_err.h, ESP_ERROR_CHECK checks the return values. It is a rather commonplace error-handling code and can be used as the default error-handling code in the application development phase. However, we strongly recommend that API users write their own error-handling code.

5.25.4 ESP32 Wi-Fi API Parameter Initialization

When initializing struct parameters for the API, one of two approaches should be followed: - explicitly set all fields of the parameter or - use get API to get current configuration first, then set application specific

fields

Initializing or getting the entire structure is very important because most of the time the value 0 indicates the default value is used. More fields may be added to the struct in the future and initializing these to zero ensures the application will still work correctly after IDF is updated to a new release.

5.25.5 ESP32 Wi-Fi Programming Model

The ESP32 Wi-Fi programming model is depicted as follows:

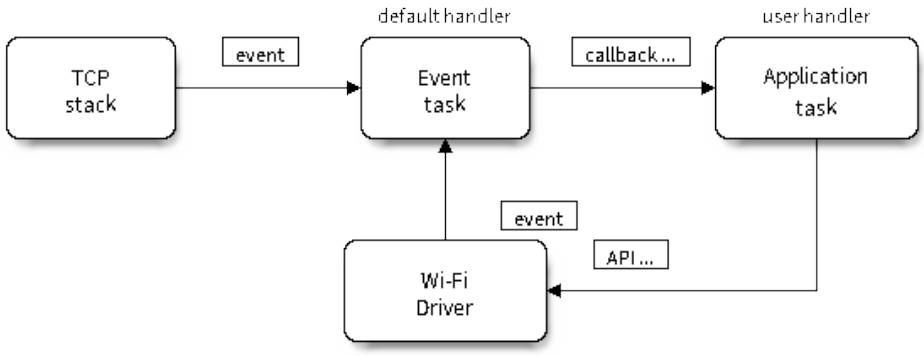


图 24: Wi-Fi Programming Model

The Wi-Fi driver can be considered a black box that knows nothing about high-layer code, such as the TCP/IP stack, application task, event task, etc. The application task (code) generally calls *Wi-Fi driver APIs* to initialize Wi-Fi and handles Wi-Fi events when necessary. Wi-Fi driver receives API calls, handles them, and post events to the application.

Wi-Fi event handling is based on the *esp_event library*. Events are sent by the Wi-Fi driver to the *default event loop*. Application may handle these events in callbacks registered using `esp_event_handler_register()`. Wi-Fi events are also handled by *esp_netif component* to provide a set of default behaviors. For example, when Wi-Fi station connects to an AP, esp_netif will automatically start the DHCP client (by default).

5.25.6 ESP32 Wi-Fi Event Description

WIFI_EVENT_WIFI_READY

The Wi-Fi driver will never generate this event, which, as a result, can be ignored by the application event callback. This event may be removed in future releases.

WIFI_EVENT_SCAN_DONE

The scan-done event is triggered by esp_wifi_scan_start() and will arise in the following scenarios:

- The scan is completed, e.g., the target AP is found successfully, or all channels have been scanned.
- The scan is stopped by esp_wifi_scan_stop().
- The esp_wifi_scan_start() is called before the scan is completed. A new scan will override the current scan and a scan-done event will be generated.

The scan-done event will not arise in the following scenarios:

- It is a blocked scan.
- The scan is caused by esp_wifi_connect().

Upon receiving this event, the event task does nothing. The application event callback needs to call esp_wifi_scan_get_ap_num() and esp_wifi_scan_get_ap_records() to fetch the scanned AP list and trigger the Wi-Fi driver to free the internal memory which is allocated during the scan **(do not forget to do this)**! Refer to 'ESP32 Wi-Fi Scan' for a more detailed description.

WIFI_EVENT_STA_START

If esp_wifi_start() returns ESP_OK and the current Wi-Fi mode is Station or AP+Station, then this event will arise. Upon receiving this event, the event task will initialize the LwIP network interface (netif). Generally, the application event callback needs to call esp_wifi_connect() to connect to the configured AP.

WIFI_EVENT_STA_STOP

If esp_wifi_stop() returns ESP_OK and the current Wi-Fi mode is Station or AP+Station, then this event will arise. Upon receiving this event, the event task will release the station's IP address, stop the DHCP client, remove TCP/UDP-related connections and clear the LwIP station netif, etc. The application event callback generally does not need to do anything.

WIFI_EVENT_STA_CONNECTED

If esp_wifi_connect() returns ESP_OK and the station successfully connects to the target AP, the connection event will arise. Upon receiving this event, the event task starts the DHCP client and begins the DHCP process of getting the IP address. Then, the Wi-Fi driver is ready for sending and receiving data. This moment is good for beginning the application work, provided that the application does not depend on LwIP, namely the IP address. However, if the application is LwIP-based, then you need to wait until the *got ip* event comes in.

WIFI_EVENT_STA_DISCONNECTED

This event can be generated in the following scenarios:

- When esp_wifi_disconnect(), or esp_wifi_stop(), or esp_wifi_deinit(), or esp_wifi_restart() is called and the station is already connected to the AP.
- When esp_wifi_connect() is called, but the Wi-Fi driver fails to set up a connection with the AP due to certain reasons, e.g. the scan fails to find the target AP, authentication times out, etc. If there are more than one AP with the same SSID, the disconnected event is raised after the station fails to connect all of the found APs.
- When the Wi-Fi connection is disrupted because of specific reasons, e.g., the station continuously loses N beacons, the AP kicks off the station, the AP' s authentication mode is changed, etc.

Upon receiving this event, the default behavior of the event task is: - Shuts down the station' s LwIP netif. - Notifies the LwIP task to clear the UDP/TCP connections which cause the wrong status to all sockets. For socket-based applications, the application callback can choose to close all sockets and re-create them, if necessary, upon receiving this event.

The most common event handle code for this event in application is to call esp_wifi_connect() to reconnect the Wi-Fi. However, if the event is raised because esp_wifi_disconnect() is called, the application should not call esp_wifi_connect() to reconnect. It' s application' s responsibility to distinguish whether the event is caused by esp_wifi_disconnect() or other reasons. Sometimes a better reconnect strategy is required, refer to <*Wi-Fi Reconnect*> and <*Scan When Wi-Fi Is Connecting*>.

Another thing deserves our attention is that the default behavior of LwIP is to abort all TCP socket connections on receiving the disconnect. Most of time it is not a problem. However, for some special application, this may not be what they want, consider following scenarios:

- The application creates a TCP connection to maintain the application-level keep-alive data that is sent out every 60 seconds.
- Due to certain reasons, the Wi-Fi connection is cut off, and the <*WIFI_EVENT_STA_DISCONNECTED*> is raised. According to the current implementation, all TCP connections will be removed and the keep-alive socket will be in a wrong status. However, since the application designer believes that the network layer should NOT care about this error at the Wi-Fi layer, the application does not close the socket.
- Five seconds later, the Wi-Fi connection is restored because esp_wifi_connect() is called in the application event callback function. **Moreover, the station connects to the same AP and gets the same IPV4 address as before**.
- Sixty seconds later, when the application sends out data with the keep-alive socket, the socket returns an error and the application closes the socket and re-creates it when necessary.

In above scenario, ideally, the application sockets and the network layer should not be affected, since the Wi-Fi connection only fails temporarily and recovers very quickly. The application can enable "Keep TCP

connections when IP changed" via LwIP menuconfig.

IP_EVENT_STA_GOT_IP

This event arises when the DHCP client successfully gets the IPV4 address from the DHCP server, or when the IPV4 address is changed. The event means that everything is ready and the application can begin its tasks (e.g., creating sockets).

The IPV4 may be changed because of the following reasons:

- The DHCP client fails to renew/rebind the IPV4 address, and the station' s IPV4 is reset to 0.
- The DHCP client rebinds to a different address.
- The static-configured IPV4 address is changed.

Whether the IPV4 address is changed or NOT is indicated by field `ip_change` of `ip_event_got_ip_t`.

The socket is based on the IPV4 address, which means that, if the IPV4 changes, all sockets relating to this IPV4 will become abnormal. Upon receiving this event, the application needs to close all sockets and recreate the application when the IPV4 changes to a valid one.

IP_EVENT_GOT_IP6

This event arises when the IPV6 SLAAC support auto-configures an address for the ESP32, or when this address changes. The event means that everything is ready and the application can begin its tasks (e.g., creating sockets).

IP_STA_LOST_IP

This event arises when the IPV4 address become invalid.

IP_STA_LOST_IP doesn' t arise immediately after the WiFi disconnects, instead it starts an IPV4 address lost timer, if the IPV4 address is got before ip lost timer expires, IP_EVENT_STA_LOST_IP doesn' t happen. Otherwise, the event arises when IPV4 address lost timer expires.

Generally the application don' t need to care about this event, it is just a debug event to let the application know that the IPV4 address is lost.

WIFI_EVENT_AP_START

Similar to <*WIFI_EVENT_STA_START*>.

WIFI_EVENT_AP_STOP

Similar to <*WIFI_EVENT_STA_STOP*>.

WIFI_EVENT_AP_STACONNECTED

Every time a station is connected to ESP32 AP, the <*WIFI_EVENT_AP_STACONNECTED*> will arise. Upon receiving this event, the event task will do nothing, and the application callback can also ignore it. However, you may want to do something, for example, to get the info of the connected STA, etc.

WIFI_EVENT_AP_STADISCONNECTED

This event can happen in the following scenarios:

- The application calls esp_wifi_disconnect(), or esp_wifi_deauth_sta(), to manually disconnect the station.
- The Wi-Fi driver kicks off the station, e.g. because the AP has not received any packets in the past five minutes, etc.
- The station kicks off the AP.

When this event happens, the event task will do nothing, but the application event callback needs to do something, e.g., close the socket which is related to this station, etc.

WIFI_EVENT_AP_PROBEREQRECVED

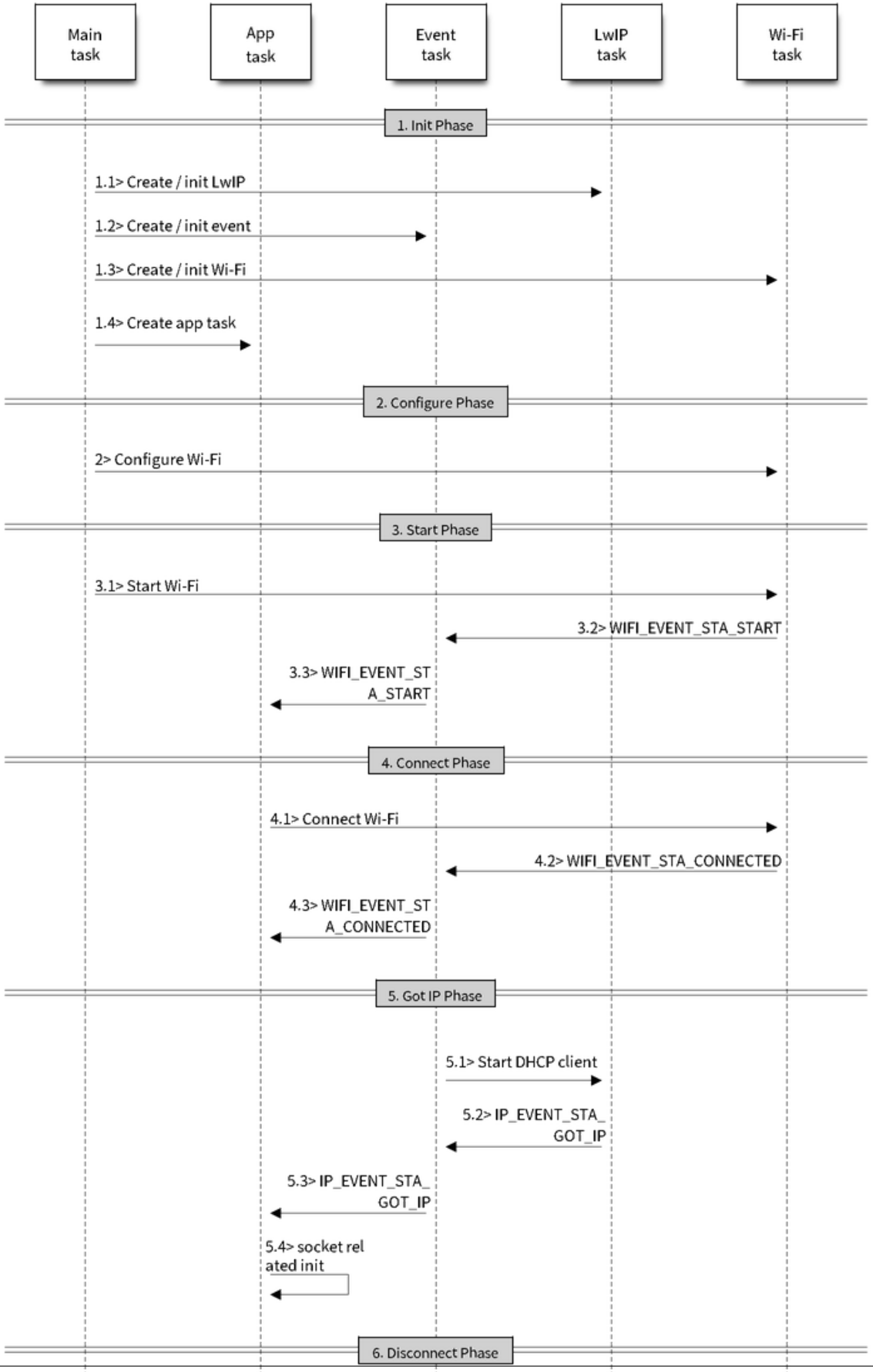
This event is disabled by default. The application can enable it via API esp_wifi_set_event_mask(). When this event is enabled, it will be raised each time the AP receives a probe request.

5.25.7 ESP32 Wi-Fi Station General Scenario

Below is a "big scenario" which describes some small scenarios in Station mode:

1. Wi-Fi/LwIP Init Phase

- s1.1: The main task calls esp_netif_init() to create an LwIP core task and initialize LwIP-related work.
- s1.2: The main task calls esp_event_loop_init() to create a system Event task and initialize an application event's callback function. In the scenario above, the application event's callback function does nothing but relaying the event to the application task.
- s1.3: The main task calls esp_netif_create_default_wifi_ap() or esp_netif_create_default_wifi_sta() to create default network interface instance binding station or AP with TCP/IP stack.
- s1.4: The main task calls esp_wifi_init() to create the Wi-Fi driver task and initialize the Wi-Fi driver.
- s1.5: The main task calls OS API to create the application task.


Main task
App task
Event task
LwIP task
Wi-Fi task
1. Init Phase
1.1> Create / init LwIP
1.2> Create / init event
1.3> Create / init Wi-Fi
1.4> Create app task
2. Configure Phase
2> Configure Wi-Fi
3. Start Phase
3.1> Start Wi-Fi
3.2> WIFI_EVENT_STA_START
3.3> WIFI_EVENT_STA_START
4. Connect Phase
4.1> Connect Wi-Fi
4.2> WIFI_EVENT_STA_CONNECTED
4.3> WIFI_EVENT_STA_CONNECTED
5. Got IP Phase
5.1> Start DHCP client
5.2> IP_EVENT_STA_GOT_IP
5.3> IP_EVENT_STA_GOT_IP
5.4> socket related init
6. Disconnect Phase
6.1> WIFI_EVENT_STA_DISCONNECTED
6.2> WIFI_EVENT_ST

Step 1.1~1.5 is a recommended sequence that initializes a Wi-Fi-/LwIP-based application. However, it is **NOT** a must-follow sequence, which means that you can create the application task in step 1.1 and put all other initializations in the application task. Moreover, you may not want to create the application task in the initialization phase if the application task depends on the sockets. Rather, you can defer the task creation until the IP is obtained.

2. Wi-Fi Configuration Phase

Once the Wi-Fi driver is initialized, you can start configuring the Wi-Fi driver. In this scenario, the mode is Station, so you may need to call esp_wifi_set_mode(WIFI_MODE_STA) to configure the Wi-Fi mode as Station. You can call other esp_wifi_set_xxx APIs to configure more settings, such as the protocol mode, country code, bandwidth, etc. Refer to <*ESP32 Wi-Fi Configuration*>.

Generally, we configure the Wi-Fi driver before setting up the Wi-Fi connection, but this is **NOT** mandatory, which means that you can configure the Wi-Fi connection anytime, provided that the Wi-Fi driver is initialized successfully. However, if the configuration does not need to change after the Wi-Fi connection is set up, you should configure the Wi-Fi driver at this stage, because the configuration APIs (such as esp_wifi_set_protocol) will cause the Wi-Fi to reconnect, which may not be desirable.

If the Wi-Fi NVS flash is enabled by menuconfig, all Wi-Fi configuration in this phase, or later phases, will be stored into flash. When the board powers on/reboots, you do not need to configure the Wi-Fi driver from scratch. You only need to call esp_wifi_get_xxx APIs to fetch the configuration stored in flash previously. You can also configure the Wi-Fi driver if the previous configuration is not what you want.

3. Wi-Fi Start Phase

- s3.1: Call esp_wifi_start to start the Wi-Fi driver.
- s3.2: The Wi-Fi driver posts <*WIFI_EVENT_STA_START*> to the event task; then, the event task will do some common things and will call the application event callback function.
- s3.3: The application event callback function relays the <*WIFI_EVENT_STA_START*> to the application task. We recommend that you call esp_wifi_connect(). However, you can also call esp_wifi_connect() in other phrases after the <*WIFI_EVENT_STA_START*> arises.

4. Wi-Fi Connect Phase

- s4.1: Once esp_wifi_connect() is called, the Wi-Fi driver will start the internal scan/connection process.
- s4.2: If the internal scan/connection process is successful, the <*WIFI_EVENT_STA_CONNECTED*> will be generated. In the event task, it starts the DHCP client, which will finally trigger the DHCP process.

- s4.3: In the above-mentioned scenario, the application event callback will relay the event to the application task. Generally, the application needs to do nothing, and you can do whatever you want, e.g., print a log, etc.

In step 4.2, the Wi-Fi connection may fail because, for example, the password is wrong, the AP is not found, etc. In a case like this, <*WIFI_EVENT_STA_DISCONNECTED*> will arise and the reason for such a failure will be provided. For handling events that disrupt Wi-Fi connection, please refer to phase 6.

5. Wi-Fi 'Got IP' Phase

- s5.1: Once the DHCP client is initialized in step 4.2, the *got IP* phase will begin.
- s5.2: If the IP address is successfully received from the DHCP server, then <*IP_EVENT_STA_GOT_IP*> will arise and the event task will perform common handling.
- s5.3: In the application event callback, <*IP_EVENT_STA_GOT_IP*> is relayed to the application task. For LwIP-based applications, this event is very special and means that everything is ready for the application to begin its tasks, e.g. creating the TCP/UDP socket, etc. A very common mistake is to initialize the socket before <*IP_EVENT_STA_GOT_IP*> is received. **DO NOT start the socket-related work before the IP is received.**

6. Wi-Fi Disconnect Phase

- s6.1: When the Wi-Fi connection is disrupted, e.g. because the AP is powered off, the RSSI is poor, etc., <*WIFI_EVENT_STA_DISCONNECTED*> will arise. This event may also arise in phase 3. Here, the event task will notify the LwIP task to clear/remove all UDP/TCP connections. Then, all application sockets will be in a wrong status. In other words, no socket can work properly when this event happens.
- s6.2: In the scenario described above, the application event callback function relays <*WIFI_EVENT_STA_DISCONNECTED*> to the application task. We recommend that esp_wifi_connect() be called to reconnect the Wi-Fi, close all sockets and re-create them if necessary. Refer to <*WIFI_EVENT_STA_DISCONNECTED*>.

7. Wi-Fi IP Change Phase

- s7.1: If the IP address is changed, the <*IP_EVENT_STA_GOT_IP*> will arise with "ip_change" set to true.
- s7.2: **This event is important to the application. When it occurs, the timing is good for closing all created sockets and recreating them.**

8. Wi-Fi Deinit Phase

- s8.1: Call esp_wifi_disconnect() to disconnect the Wi-Fi connectivity.

- s8.2: Call esp_wifi_stop() to stop the Wi-Fi driver.
- s8.3: Call esp_wifi_deinit() to unload the Wi-Fi driver.

5.25.8 ESP32 Wi-Fi AP General Scenario

Below is a "big scenario" which describes some small scenarios in AP mode:

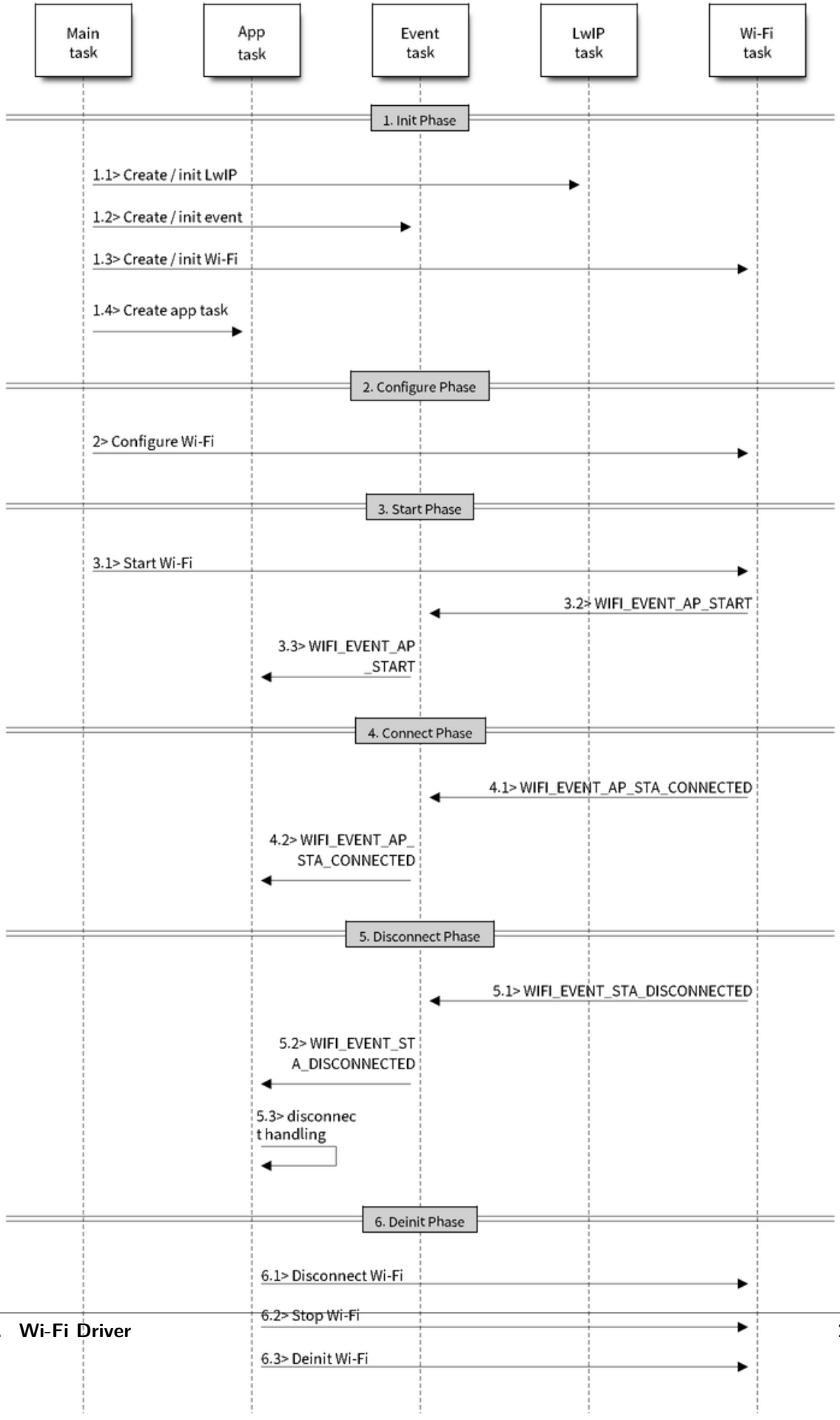
5.25.9 ESP32 Wi-Fi Scan

Currently, the esp_wifi_scan_start() API is supported only in Station or Station+AP mode.

Scan Type

Mode	Description
Active Scan	Scan by sending a probe request. The default scan is an active scan.
Passive Scan	No probe request is sent out. Just switch to the specific channel and wait for a beacon. Application can enable it via the scan_type field of wifi_scan_config_t.
Foreground Scan	This scan is applicable when there is no Wi-Fi connection in Station mode. Foreground or background scanning is controlled by the Wi-Fi driver and cannot be configured by the application.
Background Scan	This scan is applicable when there is a Wi-Fi connection in Station mode or in Station+AP mode. Whether it is a foreground scan or background scan depends on the Wi-Fi driver and cannot be configured by the application.
All-Channel Scan	It scans all of the channels. If the channel field of wifi_scan_config_t is set to 0, it is an all-channel scan.
Specific Channel Scan	It scans specific channels only. If the channel field of wifi_scan_config_t set to 1, it is a specific-channel scan.

The scan modes in above table can be combined arbitrarily, so we totally have 8 different scans:


Main task
App task
Event task
LwIP task
Wi-Fi task
1. Init Phase
1.1> Create / init LwIP
1.2> Create / init event
1.3> Create / init Wi-Fi
1.4> Create app task
2. Configure Phase
2> Configure Wi-Fi
3. Start Phase
3.1> Start Wi-Fi
3.2> WIFI_EVENT_AP_START
3.3> WIFI_EVENT_AP_START
4. Connect Phase
4.1> WIFI_EVENT_AP_STA_CONNECTED
4.2> WIFI_EVENT_AP_STA_CONNECTED
5. Disconnect Phase
5.1> WIFI_EVENT_STA_DISCONNECTED
5.2> WIFI_EVENT_STA_DISCONNECTED
5.3> disconnect handling
6. Deinit Phase
6.1> Disconnect Wi-Fi
6.2> Stop Wi-Fi
6.3> Deinit Wi-Fi

- All-Channel Background Active Scan
- All-Channel Background Passive Scan
- All-Channel Foreground Active Scan
- All-Channel Foreground Passive Scan
- Specific-Channel Background Active Scan
- Specific-Channel Background Passive Scan
- Specific-Channel Foreground Active Scan
- Specific-Channel Foreground Passive Scan

Scan Configuration

The scan type and other per-scan attributes are configured by esp_wifi_scan_start. The table below provides a detailed description of wifi_scan_config_t.

Field	Description
ssid	If the SSID is not NULL, it is only the AP with the same SSID that can be scanned.
bssid	If the BSSID is not NULL, it is only the AP with the same BSSID that can be scanned.
channel	If "channel" is 0, there will be an all-channel scan; otherwise, there will be a specific-channel scan.
show_hidden	If "show_hidden" is 0, the scan ignores the AP with a hidden SSID; otherwise, the scan considers the hidden AP a normal one.
scan_type	If "scan_type" is WIFI_SCAN_TYPE_ACTIVE, the scan is "active" ; otherwise, it is a "passive" one.
scan_time	This field is used to control how long the scan dwells on each channel. For passive scans, scan_time.passive designates the dwell time for each channel. For active scans, dwell times for each channel are listed in the table below. Here, min is short for scan time.active.min and max is short for scan_time.active.max. • min=0, max=0: scan dwells on each channel for 120 ms. • min>0, max=0: scan dwells on each channel for 120 ms. • min=0, max>0: scan dwells on each channel for `max` ms. • min>0, max>0: the minimum time the scan dwells on each channel is `min` ms. If no AP is found during this time frame, the scan switches to the next channel. Otherwise, the scan dwells on the channel for `max` ms. If you want to improve the performance of the the scan, you can try to modify these two parameters.

There also some global scan attributes which is configured by API esp_wifi_set_config, refer to *Station Basic Configuration*

Scan All APs In All Channels(foreground)

Scenario:

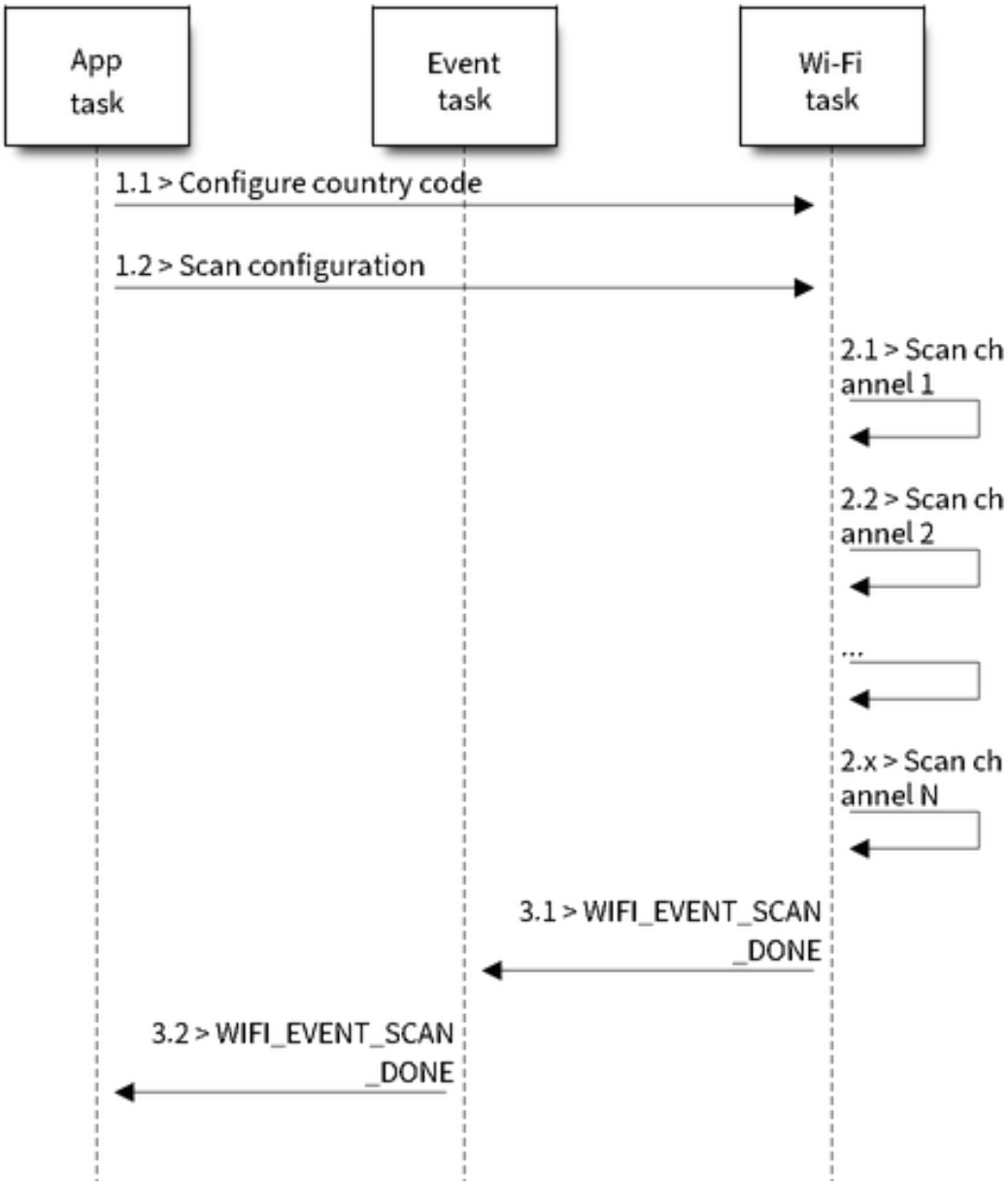


图 27: Foreground Scan of all Wi-Fi Channels

The scenario above describes an all-channel, foreground scan. The foreground scan can only occur in Station mode where the station does not connect to any AP. Whether it is a foreground or background scan is totally determined by the Wi-Fi driver, and cannot be configured by the application.

Detailed scenario description:

Scan Configuration Phase

- s1.1: Call esp_wifi_set_country() to set the country info if the default country info is not what you want, refer to <*Wi-Fi Country Code*>.
- s1.2: Call esp_wifi_scan_start() to configure the scan. To do so, you can refer to <*Scan Configuration*>. Since this is an all-channel scan, just set the SSID/BSSID/channel to 0.

Wi-Fi Driver' s Internal Scan Phase

- s2.1: The Wi-Fi driver switches to channel 1, in case the scan type is WIFI_SCAN_TYPE_ACTIVE, and broadcasts a probe request. Otherwise, the Wi-Fi will wait for a beacon from the APs. The Wi-Fi driver will stay in channel 1 for some time. The dwell time is configured in min/max time, with default value being 120 ms.
- s2.2: The Wi-Fi driver switches to channel 2 and performs the same operation as in step 2.1.
- s2.3: The Wi-Fi driver scans the last channel N, where N is determined by the country code which is configured in step 1.1.

Scan-Done Event Handling Phase

- s3.1: When all channels are scanned, <*WIFI_EVENT_SCAN_DONE*> will arise.
- s3.2: The application' s event callback function notifies the application task that <*WIFI_EVENT_SCAN_DONE*> is received. esp_wifi_scan_get_ap_num() is called to get the number of APs that have been found in this scan. Then, it allocates enough entries and calls esp_wifi_scan_get_ap_records() to get the AP records. Please note that the AP records in the Wi-Fi driver will be freed, once esp_wifi_scan_get_ap_records() is called. Do not call esp_wifi_scan_get_ap_records() twice for a single scan-done event. If esp_wifi_scan_get_ap_records() is not called when the scan-done event occurs, the AP records allocated by the Wi-Fi driver will not be freed. So, make sure you call esp_wifi_scan_get_ap_records(), yet only once.

Scan All APs on All Channels(background)

Scenario:

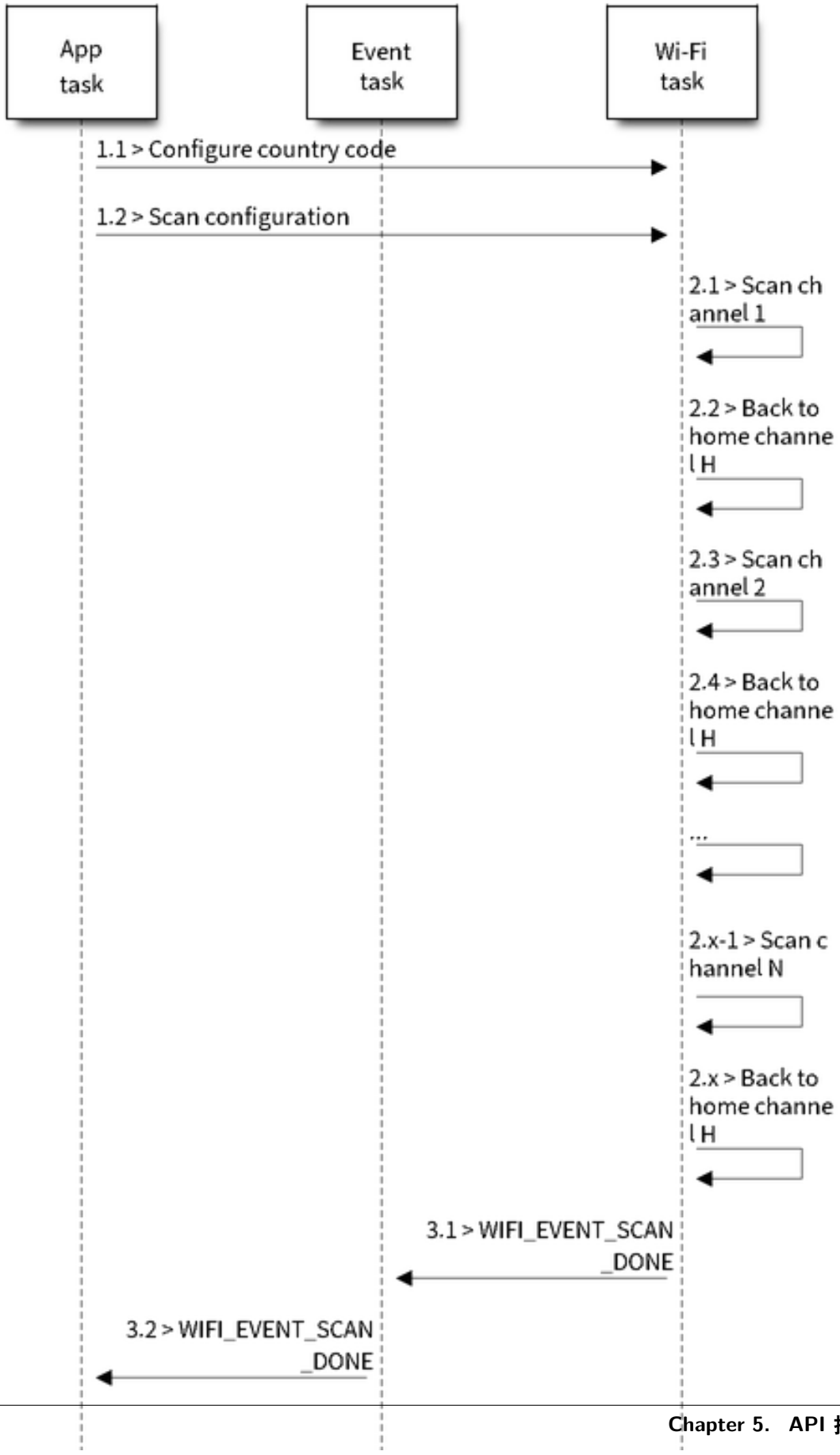


图 28: Background Scan of all Wi-Fi Channels

The scenario above is an all-channel background scan. Compared to *Scan All APs In All Channels(foreground)* , the difference in the all-channel background scan is that the Wi-Fi driver will scan the back-to-home channel for 30 ms before it switches to the next channel to give the Wi-Fi connection a chance to transmit/receive data.

Scan for a Specific AP in All Channels

Scenario:

This scan is similar to *Scan All APs In All Channels(foreground)*. The differences are:

- s1.1: In step 1.2, the target AP will be configured to SSID/BSSID.
- s2.1~s2.N: Each time the Wi-Fi driver scans an AP, it will check whether it is a target AP or not. If the scan is WIFI_FAST_SCAN scan and the target AP is found, then the scan-done event will arise and scanning will end; otherwise, the scan will continue. Please note that the first scanned channel may not be channel 1, because the Wi-Fi driver optimizes the scanning sequence.

If there are multiple APs which match the target AP info, for example, if we happen to scan two APs whose SSID is "ap" . If the scan is WIFI_FAST_SCAN, then only the first scanned "ap" will be found, if the scan is WIFI_ALL_CHANNEL_SCAN, both "ap" will be found and the station will connect the "ap" according to the configured strategy, refer to *Station Basic Configuration*.

You can scan a specific AP, or all of them, in any given channel. These two scenarios are very similar.

Scan in Wi-Fi Connect

When esp_wifi_connect() is called, then the Wi-Fi driver will try to scan the configured AP first. The scan in "Wi-Fi Connect" is the same as *Scan for a Specific AP In All Channels*, except that no scan-done event will be generated when the scan is completed. If the target AP is found, then the Wi-Fi driver will start the Wi-Fi connection; otherwise, <*WIFI_EVENT_STA_DISCONNECTED*> will be generated. Refer to *Scan for a Specific AP in All Channels*

Scan In Blocked Mode

If the block parameter of esp_wifi_scan_start() is true, then the scan is a blocked one, and the application task will be blocked until the scan is done. The blocked scan is similar to an unblocked one, except that no scan-done event will arise when the blocked scan is completed.

Parallel Scan

Two application tasks may call esp_wifi_scan_start() at the same time, or the same application task calls esp_wifi_scan_start() before it gets a scan-done event. Both scenarios can happen. **However, the Wi-Fi driver does not support multiple concurrent scans adequately. As a result, concurrent scans**

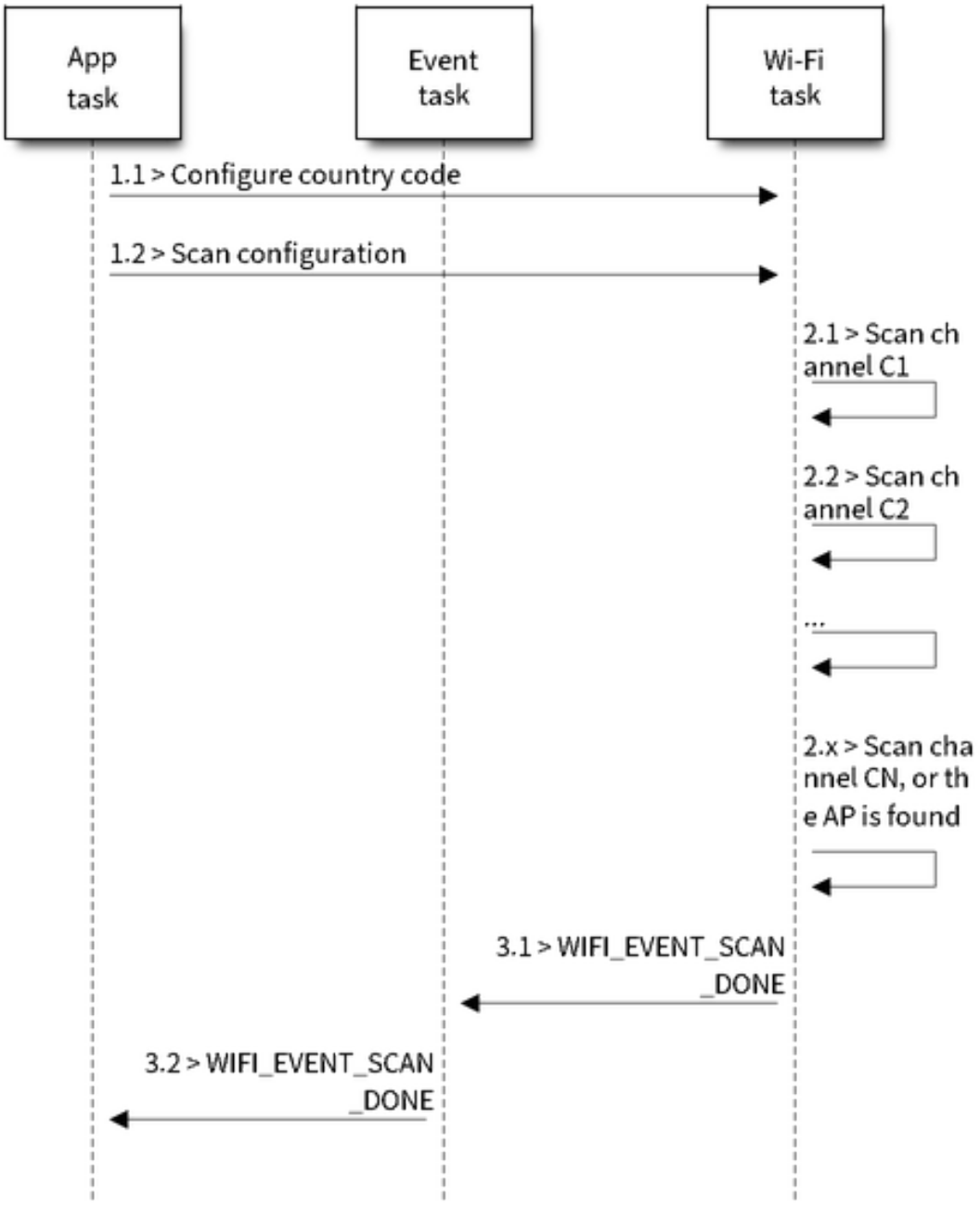


图 29: Scan of specific Wi-Fi Channels

should be avoided. Support for concurrent scan will be enhanced in future releases, as the ESP32' s Wi-Fi functionality improves continuously.

Scan When Wi-Fi Is Connecting

The esp_wifi_scan_start() fails immediately if the Wi-Fi is in connecting process because the connecting has higher priority than the scan. If scan fails because of connecting, the recommended strategy is to delay sometime and retry scan again, the scan will succeed once the connecting is completed.

However, the retry/delay strategy may not work all the time. Considering following scenario: - The station is connecting a non-existed AP or if the station connects the existed AP with a wrong password, it always raises the event <*WIFI_EVENT_STA_DISCONNECTED*>. - The application call esp_wifi_connect() to do reconnection on receiving the disconnect event. - Another application task, e.g. the console task, call esp_wifi_scan_start() to do scan, the scan always fails immediately because the station is keeping connecting. - When scan fails, the application simply delay sometime and retry the scan.

In above scenario the scan will never succeed because the connecting is in process. So if the application supports similar scenario, it needs to implement a better reconnect strategy. E.g. - The application can choose to define a maximum continuous reconnect counter, stop reconnect once the reconnect reaches the max counter. - The application can choose to do reconnect immediately in the first N continous reconnect, then give a delay sometime and reconnect again.

The application can define its own reconnect strategy to avoid the scan starve to death. Refer to <*Wi-Fi Reconnect*>.

5.25.10 ESP32 Wi-Fi Station Connecting Scenario

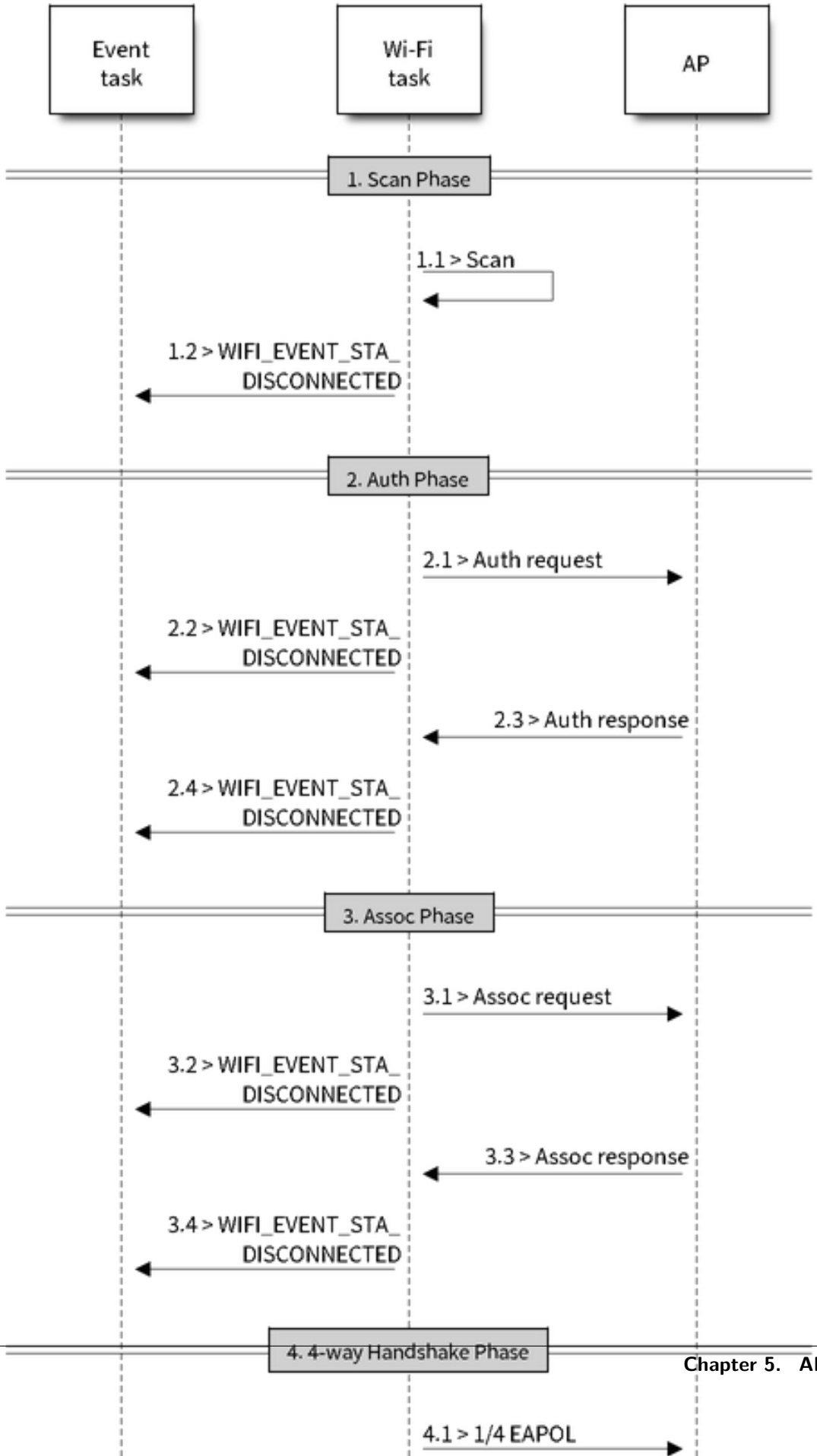
This scenario only depicts the case when there is only one target AP are found in scan phase, for the scenario that more than one AP with the same SSID are found, refer to <*ESP32 Wi-Fi Station Connecting When Multiple APs Are Found*>.

Generally, the application does not need to care about the connecting process. Below is a brief introduction to the process for those who are really interested.

Scenario:

Scan Phase

- s1.1, The Wi-Fi driver begins scanning in "Wi-Fi Connect" . Refer to <*Scan in Wi-Fi Connect*> for more details.
- s1.2, If the scan fails to find the target AP, <*WIFI_EVENT_STA_DISCONNECTED*> will arise and the reason-code will be WIFI_REASON_NO_AP_FOUND. Refer to <*Wi-Fi Reason Code*>.


Event task
Wi-Fi task
AP
1. Scan Phase
1.1 > Scan
1.2 > WIFI_EVENT_STA_ DISCONNECTED
2. Auth Phase
2.1 > Auth request
2.2 > WIFI_EVENT_STA_ DISCONNECTED
2.3 > Auth response
2.4 > WIFI_EVENT_STA_ DISCONNECTED
3. Assoc Phase
3.1 > Assoc request
3.2 > WIFI_EVENT_STA_ DISCONNECTED
3.3 > Assoc response
3.4 > WIFI_EVENT_STA_ DISCONNECTED
4. 4-way Handshake Phase
4.1 > 1/4 EAPOL

Auth Phase

- s2.1, The authentication request packet is sent and the auth timer is enabled.
- s2.2, If the authentication response packet is not received before the authentication timer times out, <*WIFI_EVENT_STA_DISCONNECTED*> will arise and the reason-code will be WIFI_REASON_AUTH_EXPIRE. Refer to <*Wi-Fi Reason Code*>.
- s2.3, The auth-response packet is received and the auth-timer is stopped.
- s2.4, The AP rejects authentication in the response and <*WIFI_EVENT_STA_DISCONNECTED*> arises, while the reason-code is WIFI_REASON_AUTH_FAIL or the reasons specified by the AP. Refer to <*Wi-Fi Reason Code*>.

Association Phase

- s3.1, The association request is sent and the association timer is enabled.
- s3.2, If the association response is not received before the association timer times out, <*WIFI_EVENT_STA_DISCONNECTED*> will arise and the reason-code will be WIFI_REASON_ASSOC_EXPIRE. Refer to <*Wi-Fi Reason Code*>.
- s3.3, The association response is received and the association timer is stopped.
- s3.4, The AP rejects the association in the response and <*WIFI_EVENT_STA_DISCONNECTED*> arises, while the reason-code is the one specified in the association response. Refer to <*Wi-Fi Reason Code*>.

Four-way Handshake Phase

- s4.1, The four-way handshake is sent out and the association timer is enabled.
- s4.2, If the association response is not received before the association timer times out, <*WIFI_EVENT_STA_DISCONNECTED*> will arise and the reason-code will be WIFI_REASON_ASSOC_EXPIRE. Refer to <*Wi-Fi Reason Code*>.
- s4.3, The association response is received and the association timer is stopped.
- s4.4, The AP rejects the association in the response and <*WIFI_EVENT_STA_DISCONNECTED*> arises and the reason-code will be the one specified in the association response. Refer to <*Wi-Fi Reason Code*>.

Wi-Fi Reason Code

The table below shows the reason-code defined in ESP32. The first column is the macro name defined in esp_wifi_types.h. The common prefix *WIFI_REASON* is removed, which means that *UNSPECIFIED* actually stands for *WIFI_REASON_UNSPECIFIED* and so on. The second column is the value of the

reason. The third column is the standard value to which this reason is mapped in section 8.4.1.7 of ieee802.11-2012. (For more information, refer to the standard mentioned above.) The last column is a description of the reason.

Reason code	Value	Mapped To	Description
UNSPECIFIED	1	1	Generally, it means an internal failure, e.g., the memory runs out, the internal TX fails, or the reason is received from the remote side, etc.
AUTH_EXPIRE	2	2	The previous authentication is no longer valid. For the ESP32 Station, this reason is reported when: • auth is timed out • the reason is received from the AP. For the ESP32 AP, this reason is reported when: • the AP has not received any packets from the station in the past five minutes. • the AP is stopped by calling esp_wifi_stop(). • the station is deauthed by calling esp_wifi_deauth_sta()
AUTH_LEAVE	3	3	De-authenticated, because the sending STA is leaving (or has left). For the ESP32 Station, this reason is reported when: • it is received from the AP.
ASSOC_EXPIRE	4	4	Disassociated due to inactivity. For the ESP32 Station, this reason is reported when: • it is received from the AP.

5.25.11 ESP32 Wi-Fi Station Connecting When Multiple APs Are Found

This scenario is similar as <*ESP32 Wi-Fi Station Connecting Scenario*>, the difference is the station will not raise the event <*WIFI_EVENT_STA_DISCONNECTED*> unless it fails to connect all of the found APs.

5.25.12 Wi-Fi Reconnect

The station may disconnect due to many reasons, e.g. the connected AP is restarted etc. It's the application's responsibility to do the reconnect. The recommended reconnect strategy is to call esp_wifi_connect() on receiving event <*WIFI_EVENT_STA_DISCONNECTED*>.

Sometimes the application needs more complex reconnect strategy: - If the disconnect event is raised because the esp_wifi_disconnect() is called, the application may not want to do reconnect. - If the esp_wifi_scan_start() may be called at anytime, a better reconnect strategy is necessary, refer to <*Scan When Wi-Fi Is Connecting*>.

Another thing we need to consider is the reconnect may not connect the same AP if there are more than one APs with the same SSID. The reconnect always select current best APs to connect.

5.25.13 Wi-Fi Beacon Timeout

The beacon timeout mechanism is used by ESP32 station to detect whether the AP is alive or not. If the station continuously loses 60 beacons of the connected AP, the beacon timeout happens.

After the beacon timeout happens, the station sends 5 probe requests to AP, it disconnects the AP and raises the event <*WIFI_EVENT_STA_DISCONNECTED*> if still no probe response or beacon is received from AP.

5.25.14 ESP32 Wi-Fi Configuration

All configurations will be stored into flash when the Wi-Fi NVS is enabled; otherwise, refer to <*Wi-Fi NVS Flash*>.

Wi-Fi Mode

Call esp_wifi_set_mode() to set the Wi-Fi mode.

Mode	Description
WIFI_MODE_NULL	Null mode: in this mode, the internal data struct is not allocated to the station and the AP, while both the station and AP interfaces are not initialized for RX/TX Wi-Fi data. Generally, this mode is used for Sniffer, or when you only want to stop both the STA and the AP without calling esp_wifi_deinit() to unload the whole Wi-Fi driver.
WIFI_MODE_STA	Station mode: in this mode, esp_wifi_start() will init the internal station data, while the station's interface is ready for the RX and TX Wi-Fi data. After esp_wifi_connect() is called, the STA will connect to the target AP.
WIFI_MODE_AP	AP mode: in this mode, esp_wifi_start() will init the internal AP data, while the AP's interface is ready for RX/TX Wi-Fi data. Then, the Wi-Fi driver starts broad- casting beacons, and the AP is ready to get connected to other stations.
WIFI_MODE_APSTA	Station-AP coexistence mode: in this mode, esp_wifi_start() will simultaneously init both the station and the AP. This is done in station mode and AP mode. Please note that the channel of the external AP, which the ESP32 Station is connected to, has higher priority over the ESP32 AP channel.

Station Basic Configuration

API esp_wifi_set_config() can be used to configure the station. The table below describes the fields in detail.

Field	Description
ssid	This is the SSID of the target AP, to which the station wants to connect to.
password	Password of the target AP
scan_method	For WIFI_FAST_SCAN scan, the scan ends when the first matched AP is found, for WIFI_ALL_CHANNEL_SCAN, the scan finds all matched APs in all channels. The default scan is WIFI_FAST_SCAN.
bssid_set	If bssid_set is 0, the station connects to the AP whose SSID is the same as the field "ssid", while the field "bssid" is ignored. In all other cases, the station connects to the AP whose SSID is the same as the "ssid" field, while its BSSID is the same the "bssid" field .
bssid	This is valid only when bssid_set is 1; see field "bssid_set".
channel	If the channel is 0, the station scans the channel 1~N to search for the target AP; otherwise, the station starts by scanning the channel whose value is the same as that of the "channel" field, and then scans others to find the target AP. If you do not know which channel the target AP is running on, set it to 0.
sort_method	This field is only for WIFI_ALL_CHANNEL_SCAN If the sort_method is WIFI_CONNECT_AP_BY_SIGNAL, all matched APs are sorted by signal, for AP with best signal will be connected firstly. E.g. if the station want to connect AP whose ssid is "apxx", the scan finds two AP whose ssid equals to "apxx", the first AP's signal is -90dBm, the second AP's signal is -30dBm, the station connects the second AP firstly, it doesn't connect the first one unless it fails to connect the second one. If the sort_method is WIFI_CONNECT_AP_BY_SECURITY, all matched APs are sorted by security. E.g. if the station wants to connect AP whose ssid is "apxx", the scan finds two AP whose ssid is "apxx", the security of the first found AP is open while the second one is WPA2, the stations connects to the second AP firstly, it doesn't connect the second one unless it fails to connect the first one.
threshold	The threshold is used to filter the found AP, if the RSSI or security mode is less than the configured threshold, the AP will be discard. If the RSSI set to 0, it means default threshold, the default RSSI threshold is -127dBm. If the authmode threshold is set to 0, it means default threshold, the default authmode threshold is open.

注意: WEP/WPA security modes are deprecated in IEEE802.11-2016 specifications and are recommended not to be used. These modes can be rejected using authmode threshold by setting threshold as WPA2 by threshold.authmode as WIFI_AUTH_WPA2_PSK.

AP Basic Configuration

API esp_wifi_set_config() can be used to configure the AP. The table below describes the fields in detail.

Field	Description
ssid	SSID of AP; if the ssid[0] is 0xFF and ssid[1] is 0xFF, the AP defaults the SSID to ESP_aabbcc, where "aabbcc" is the last three bytes of the AP MAC.
password	Password of AP; if the auth mode is WIFI_AUTH_OPEN, this field will be ignored.
ssid_len	Length of SSID; if ssid_len is 0, check the SSID until there is a termination character. If ssid_len > 32, change it to 32; otherwise, set the SSID length according to ssid_len.
channel	Channel of AP; if the channel is out of range, the Wi-Fi driver defaults the channel to channel 1. So, please make sure the channel is within the required range. For more details, refer to <*Wi-Fi Country Code*>.
authmode	Auth mode of ESP32 AP; currently, ESP32 Wi-Fi does not support AUTH_WEP. If the authmode is an invalid value, AP defaults the value to WIFI_AUTH_OPEN.
ssid_hidden	If ssid_hidden is 1, AP does not broadcast the SSID; otherwise, it does broadcast the SSID.
max_connection	Currently, ESP32 Wi-Fi supports up to 10 Wi-Fi connections. If max_connection > 10, AP defaults the value to 10.
beacon_interval	Beacon interval; the value is 100 ~ 60000 ms, with default value being 100 ms. If the value is out of range, AP defaults it to 100 ms.

Wi-Fi Protocol Mode

Currently, the IDF supports the following protocol modes:

Protocol Mode	Description
802.11B	Call esp_wifi_set_protocol(ifx, WIFI_PROTOCOL_11B) to set the station/AP to 802.11B-only mode.
802.11BG	Call esp_wifi_set_protocol(ifx, WIFI_PROTOCOL_11B|WIFI_ PROTOCOL_11G) to set the station/AP to 802.11BG mode.
802.11BGN	Call esp_wifi_set_protocol(ifx, WIFI_PROTOCOL_11B| WIFI_PROTOCOL_11G|WIFI_PROTOCOL_11N) to set the station/ AP to BGN mode.
802.11BGNLR	Call esp_wifi_set_protocol(ifx, WIFI_PROTOCOL_11B| WIFI_PROTOCOL_11G|WIFI_PROTOCOL_11N|W to set the station/AP to BGN and the Espressif-specific mode.
802.11LR	Call esp_wifi_set_protocol(ifx, WIFI_PROTOCOL_LR) to set the station/AP only to the Espressif-specific mode. **This mode is an Espressif-patented mode which can achieve a one-kilometer line of sight range. Please, make sure both the station and the AP are connected to an ESP32 device**

Long Range (LR)

Long Range (LR) mode is an Espressif-patented Wi-Fi mode which can achieve a one-kilometer line of sight range. It has better reception sensitivity, stronger anti-interference ability and longer transmission distance than the traditional 802.11B mode.

LR Compitability

Since LR is Espressif unique Wi-Fi mode, only ESP32 devices can transmit and receive the LR data. In other words, the ESP32 device should NOT transmit the data in LR data rate if the connected device doesn't support LR. The application can achieve this by configuring suitable Wi-Fi mode. If the negotiated mode supports LR, the ESP32 may transmit data in LR rate, otherwise, ESP32 will transmit all data in traditional Wi-Fi data rate.

Following table depicts the Wi-Fi mode negotiation:

APSTA	BGN	BG	B	BGNLR	BGLR	BLR	LR
BGN	BGN	BG	B	BGN	BG	B	•
BG	BG	BG	B	BG	BG	B	•
B	B	B	B	B	B	B	•
BGNLR	•	•	•	BGNLR	BGLR	BLR	LR
BGLR	•	•	•	BGLR	BGLR	BLR	LR
BLR	•	•	•	BLR	BLR	BLR	LR
LR	•	•	•	LR	LR	LR	LR

In above table, the row is the Wi-Fi mode of AP and the column is the Wi-Fi mode of station. The "-" indicates Wi-Fi mode of the AP and station are not compatible.

According to the table, we can conclude that:

- For LR enabled in ESP32 AP, it's incompatible with traditional 802.11 mode because the beacon is sent in LR mode.
- For LR enabled in ESP32 station and the mode is NOT LR only mode, it's compatible with traditional 802.11 mode.

- If both station and AP are ESP32 devices and both of them enable LR mode, the negotiated mode supports LR.

If the negotiated Wi-Fi mode supports both traditional 802.11 mode and LR mode, it' s the WiFi driver' s responsibility to automatically select the best data rate in different Wi-Fi mode and the application don' t need to care about it.

LR Impacts to Traditional Wi-Fi device

The data transmission in LR rate has no impacts on the traditional Wi-Fi device because:

- The CCA and backoff process in LR mode are consistent with 802.11 specification.
- The traditional Wi-Fi device can detect the LR signal via CCA and do backoff.

In other words, the impact transmission in LR mode is similar as the impact in 802.11B mode.

LR Transmission Distance

The reception sensitivity of LR has about 4 dB gain than the traditional 802.11 B mode, theoretically the transmission distance is about 2 to 2.5 times the distance of 11B.

LR Throughput

The LR rate has very limited throughput because the raw PHY data rate LR is 1/2 Mbits and 1/4 Mbits.

When to Use LR

The general conditions for using LR are:

- Both the AP and station are ESP32 devices.
- Long distance WiFi connection and data transmission is required.
- Data throughput requirements are very small, such as remote device control, etc.

Wi-Fi Country Code

Call esp_wifi_set_country() to set the country info. The table below describes the fields in detail, please consult local 2.4GHz RF operating regulations before configuring these fields.

Field	Description
cc[3]	Country code string, this attributes identify the country or noncountry entity in which the station/AP is operating. If it' s a country, the first two octets of this string is the two character country info as described in document ISO/IEC3166-1. The third octect is one of the following: • an ASCII space character, if the regulations under which the station/AP is operating encompass all environments for the current frequency band in the country • an ASCII 'O' character if the regulations under which the station/AP is operating are for an outdoor environment only, or • an ASCII 'I' character if the regulations under which the station/AP is operating are for an indoor environment only. • an ASCII 'X' character if the station/AP is operating under a noncountry entity. The first two octets of the noncountry entity is two ASSCII 'XX' characters. • the binary representation of the Operating Class table number currently in use. Refer 802.11-2012 Annex E.
schan	Start channel, it' s the minimum channel number of the regulations under which the station/AP can operate.
snum	Total channel number of the regulations, e.g. if the schan=1, nchan=13, it means the station/AP can send data from channel 1 to 13.
policy	Country policy, this field control which country info will be used if the configured country info is conflict with the connected AP' s. More description about policy is provided in following section.

The default country info is {.cc=" CN" , .schan=1, .nchan=13, policy=WIFI_COUNTRY_POLICY_AUTO}, if the WiFi Mode is station/AP coexist mode, they share the same configured country info. Sometimes, the country info of AP, to which the station is connected, is different from the country info of configured. For example, the configured station has country info {.cc=" JP" , .schan=1, .nchan=14, policy=WIFI_COUNTRY_POLICY_AUTO}, but the connected AP has

country info {.cc=" CN" , .schan=1, .nchan=13}, then country info of connected AP' s is used. Following table depicts which country info is used in different WiFi Mode and different country policy, also describe the impact to active scan.

WiFi Mode	Policy	Description
Station	WIFI_COUNTRY_POLICY_AUTO	If the connected AP has country IE in its beacon, the country info equals to the country info in beacon, otherwise, use default country info. For scan: • before the station connects to the AP, scans channel "schan" to "min(11, schan+nchan-1)" with active scan and channel min(12, schan+nchan)" to 14 with passive scan. E.g. if the used country info is {.cc=" CN" , .schan=1, .nchan=6} then 1 to 6 is active scan and 7 to 14 is passive scan If the used country info is {.cc=" CN" , .schan=1, .nchan=12} then 1 to 11 is active scan and 12 to 14 is passive scan • after the station connects to the AP, scans channel "schan" to "schan+nchan-1" with active scan and channel "schan+nchan" to 14 with passive scan Always keep in mind that if if a AP with with hidden SSID is set to a passive scan channel, the passive scan will not find it. In other words, if the application hopes to find the AP with hidden SSID in every channel, the policy of country info should be configured to WIFI_COUNTRY_POLICY_MANUAL.
Station	WIFI_COUNTRY_POLICY_MANUAL	Always use the configured country info For scan, scans channel "schan" to "schan+nchan-1" with active scan
AP	WIFI_COUNTRY_POLICY_AUTO	Always use the configured country info

Home Channel

In AP mode, the home channel is defined as that of the AP channel. In Station mode, the home channel is defined as the channel of the AP to which the station is connected. In Station+AP mode, the home channel of AP and station must be the same. If the home channels of Station and AP are different, the station' s home channel is always in priority. Take the following as an example: at the beginning, the AP is on channel 6, then the station connects to an AP whose channel is 9. Since the station' s home channel has a higher priority, the AP needs to switch its channel from 6 to 9 to make sure that both station and AP have the same home channel. While switching channel, the ESP32 in SoftAP mode will notify the connected stations about the channel migration using a Channel Switch Announcement (CSA). Stations that support channel switching will transition smoothly whereas stations who do not will disconnect and reconnect to the SoftAP.

Wi-Fi Vendor IE Configuration

By default, all Wi-Fi management frames are processed by the Wi-Fi driver, and the application does not need to care about them. Some applications, however, may have to handle the beacon, probe request, probe response and other management frames. For example, if you insert some vendor-specific IE into the management frames, it is only the management frames which contain this vendor-specific IE that will be processed. In ESP32, esp_wifi_set_vendor_ie() and esp_wifi_set_vendor_ie_cb() are responsible for this kind of tasks.

5.25.15 Wi-Fi Security

In addition to traditional security methods (WEP/WPA-TKIP/WPA2-CCMP), ESP32 Wi-Fi now supports state-of-the-art security protocols, namely Protected Management Frames based on 802.11w standard and Wi-Fi Protected Access 3 (WPA3-Personal). Together, PMF and WPA3 provide better privacy and robustness against known attacks in traditional modes.

Protected Management Frames (PMF)

In Wi-Fi, management frames such as beacons, probes, (de)authentication, (dis)association are used by non-AP stations to scan and connect to an AP. Unlike data frames, these frames are sent unencrypted. An attacker can use eavesdropping and packet injection to send spoofed (de)authentication/(dis)association frames at the right time, leading to following attacks in case of unprotected management frame exchanges.

- DOS attack on one or all clients in the range of the attacker.
- Tearing down existing association on AP side by sending association request.
- Forcing a client to perform 4-way handshake again in case PSK is compromised in order to get PTK.
- Getting SSID of hidden network from association request.

- Launching man-in-the-middle attack by forcing clients to deauth from legitimate AP and associating to a rogue one.

PMF provides protection against these attacks by encrypting unicast management frames and providing integrity checks for broadcast management frames. These include deauthentication, disassociation and robust management frames. It also provides Secure Association (SA) teardown mechanism to prevent spoofed association/authentication frames from disconnecting already connected clients.

ESP32 supports the following three modes of operation with respect to PMF.

- PMF not supported: In this mode, ESP32 indicates to AP that it is not capable of supporting management protection during association. In effect, security in this mode will be equivalent to that in traditional mode.
- PMF capable, but not required: In this mode, ESP32 indicates to AP that it is capable of supporting PMF. The management protection will be used if AP mandates PMF or is at least capable of supporting PMF.
- PMF capable and required: In this mode, ESP32 will only connect to AP, if AP supports PMF. If not, ESP32 will refuse to connect to the AP.

esp_wifi_set_config() can be used to configure PMF mode by setting appropriate flags in *pmf_cfg* parameter. Currently, PMF is supported only in Station mode.

WPA3-Personal

Wi-Fi Protected Access-3 (WPA3) is a set of enhancements to Wi-Fi access security intended to replace the current WPA2 standard. In order to provide more robust authentication, WPA3 uses Simultaneous Authentication of Equals (SAE), which is password-authenticated key agreement method based on Diffie-Hellman key exchange. Unlike WPA2, the technology is resistant to offline-dictionary attack, where the attacker attempts to determine shared password based on captured 4-way handshake without any further network interaction. WPA3 also provides forward secrecy, which means the captured data cannot be decrypted even if password is compromised after data transmission. Please refer to Security section of Wi-Fi Alliance' s official website for further details.

In order to enable WPA3-Personal, "Enable WPA3-Personal" should be selected in menuconfig. If enabled, ESP32 uses SAE for authentication if supported by the AP. Since PMF is a mandatory requirement for WPA3, PMF capability should be at least set to "PMF capable, but not required" for ESP32 to use WPA3 mode. Application developers need not worry about the underlying security mode as highest available is chosen from security standpoint. Note that Wi-Fi stack size requirement will increase approximately by 3k when WPA3 is used. Currently, WPA3 is supported only in Station mode.

5.25.16 ESP32 Wi-Fi Power-saving Mode

Station Sleep

Currently, ESP32 Wi-Fi supports the Modem-sleep mode which refers to the legacy power-saving mode in the IEEE 802.11 protocol. Modem-sleep mode works in Station-only mode and the station must connect to the AP first. If the Modem-sleep mode is enabled, station will switch between active and sleep state periodically. In sleep state, RF, PHY and BB are turned off in order to reduce power consumption. Station can keep connection with AP in modem-sleep mode.

Modem-sleep mode includes minimum and maximum power save modes. In minimum power save mode, station wakes up every DTIM to receive beacon. Broadcast data will not be lost because it is transmitted after DTIM. However, it can not save much more power if DTIM is short for DTIM is determined by AP.

In maximum power save mode, station wakes up every listen interval to receive beacon. This listen interval can be set longer than the AP DTIM period. Broadcast data may be lost because station may be in sleep state at DTIM time. If listen interval is longer, more power is saved but broadcast data is more easy to lose. Listen interval can be configured by calling API *esp_wifi_set_config()* before connecting to AP.

Call `esp_wifi_set_ps(WIFI_PS_MIN_MODEM)` to enable Modem-sleep minimum power save mode or `esp_wifi_set_ps(WIFI_PS_MAX_MODEM)` to enable Modem-sleep maximum power save mode after calling *esp_wifi_init()*. When station connects to AP, Modem-sleep will start. When station disconnects from AP, Modem-sleep will stop.

Call `esp_wifi_set_ps(WIFI_PS_NONE)` to disable modem sleep entirely. This has much higher power consumption, but provides minimum latency for receiving Wi-Fi data in real time. When modem sleep is enabled, received Wi-Fi data can be delayed for as long as the DTIM period (minimum power save mode) or the listen interval (maximum power save mode).

The default Modem-sleep mode is WIFI_PS_MIN_MODEM.

AP Sleep

Currently ESP32 AP doesn't support all of the power save feature defined in Wi-Fi specification. To be specific, the AP only caches unicast data for the stations connect to this AP, but doesn't cache the multicast data for the stations. If stations connected to the ESP32 AP are power save enabled, they may experience multicast packet loss.

In future, all power save features will be supported on ESP32 AP.

5.25.17 ESP32 Wi-Fi Connect Crypto

Now ESP32 have two group crypto functions can be used when do wifi connect, one is the original functions, the other is optimized by ESP hardware: 1. Original functions which is the source code used in the folder components/wpa_supplicant/src/crypto function; 2. The optimized functions is in the folder components/wpa_supplicant/src/fast_crypto, these function used the hardware crypto to make it faster than origin one, the type of function's name add *fast_* to distinguish with the original one. For example, the

API aes_wrap() is used to encrypt frame information when do 4 way handshake, the fast_aes_wrap() has the same result but can be faster.

Two groups of crypto function can be used when register in the wpa_crypto_funcs_t, wpa2_crypto_funcs_t and wps_crypto_funcs_t structure, also we have given the recommend functions to register in the fast_crypto_ops.c, you can register the function as the way you need, however what should make action is that the crypto_hash_xxx function and crypto_cipher_xxx function need to register with the same function to operation. For example, if you register crypto_hash_init() function to initialize the esp_crypto_hash structure, you need use the crypto_hash_update() and crypto_hash_finish() function to finish the operation, rather than fast_crypto_hash_update() or fast_crypto_hash_finish().

5.25.18 ESP32 Wi-Fi Throughput

The table below shows the best throughput results we got in Espressif' s lab and in a shield box.

Type/Throughput	Air In Lab	Shield-box	Test Tool	IDF Version (commit ID)
Raw 802.11 Packet RX	N/A	**130 MBit/sec**	Internal tool	NA
Raw 802.11 Packet TX	N/A	**130 MBit/sec**	Internal tool	NA
UDP RX	30 MBit/sec	90 MBit/sec	iperf example	05838641
UDP TX	30 MBit/sec	60 MBit/sec	iperf example	05838641
TCP RX	20 MBit/sec	50 MBit/sec	iperf example	05838641
TCP TX	20 MBit/sec	50 MBit/sec	iperf example	05838641

When the throughput is tested by iperf example, the sdkconfig is examples/wifi/iperf/sdkconfig.defaults.99

5.25.19 Wi-Fi 80211 Packet Send

Important notes: The API esp_wifi_80211_tx is not available in IDF 2.1, but will be so in the upcoming release.

The esp_wifi_80211_tx API can be used to:

- Send the beacon, probe request, probe response, action frame.
- Send the non-QoS data frame.

It cannot be used for sending encrypted or QoS frames.

Preconditions of Using esp_wifi_80211_tx

- The Wi-Fi mode is Station, or AP, or Station+AP.
- Either esp_wifi_set_promiscuous(true), or esp_wifi_start(), or both of these APIs return ESP_OK. This is because we need to make sure that Wi-Fi hardware is initialized before esp_wifi_80211_tx()

is called. In ESP32, both esp_wifi_set_promiscuous(true) and esp_wifi_start() can trigger the initialization of Wi-Fi hardware.

- The parameters of esp_wifi_80211_tx are hereby correctly provided.

Data rate

- If there is no WiFi connection, the data rate is 1Mbps.
- If there is WiFi connection and the packet is from station to AP or from AP to station, the data rate is same as the WiFi connection. Otherwise the data rate is 1Mbps.

Side-Effects to Avoid in Different Scenarios

Theoretically, if we do not consider the side-effects the API imposes on the Wi-Fi driver or other stations/APs, we can send a raw 802.11 packet over the air, with any destination MAC, any source MAC, any BSSID, or any other type of packet. However,robust/useful applications should avoid such side-effects. The table below provides some tips/recommendations on how to avoid the side-effects of esp_wifi_80211_tx in different scenarios.

Scenario	Description
No WiFi connection	In this scenario, no Wi-Fi connection is set up, so there are no side-effects on the Wi-Fi driver. If en_sys_seq==true, the Wi-Fi driver is responsible for the sequence control. If en_sys_seq==false, the application needs to ensure that the buffer has the correct sequence. Theoretically, the MAC address can be any address. However, this may impact other stations/APs with the same MAC/BSSID. Side-effect example#1 The application calls esp_wifi_80211_tx to send a beacon with BSSID == mac_x in AP mode, but the mac_x is not the MAC of the AP interface. Moreover, there is another AP, say "other-AP" , whose bssid is mac_x. If this happens, an "unexpected behavior" may occur, because the stations which connect to the "other-AP" cannot figure out whether the beacon is from the "other-AP" or the esp_wifi_80211_tx. To avoid the above-mentioned side-effects, we recommend that: • If esp_wifi_80211_tx is called in Station mode, the first MAC should be a multicast MAC or the exact target-device' s MAC, while the second MAC should be that of the station interface. • If esp_wifi_80211_tx is called in AP mode, the first MAC should be a multicast MAC or the exact target-device' s MAC, while the second MAC should be that of the AP interface. The recommendations above are only for avoiding side-effects and can be ignored when there are good reasons for doing this.
Have WiFi connection	When the Wi-Fi connection is already set up, and the sequence is controlled by the application, the latter may impact the sequence control of the Wi-Fi connection, as a whole. So, the en_sys_seq need to be true, otherwise ESP_ERR_WIFI_ARG is returned. The MAC-address recommendations in the "No WiFi connection" scenario also apply to this scenario. If the WiFi mode is station mode and the MAC address1 is the MAC of AP to which the station is connected, the MAC address2 is the MAC of station

5.25.20 Wi-Fi Sniffer Mode

The Wi-Fi sniffer mode can be enabled by esp_wifi_set_promiscuous(). If the sniffer mode is enabled, the following packets **can** be dumped to the application:

- 802.11 Management frame
- 802.11 Data frame, including MPDU, AMPDU, AMSDU, etc.
- 802.11 MIMO frame, for MIMO frame, the sniffer only dumps the length of the frame.
- 802.11 Control frame

The following packets will **NOT** be dumped to the application:

- 802.11 error frame, such as the frame with a CRC error, etc.

For frames that the sniffer **can** dump, the application can additionally decide which specific type of packets can be filtered to the application by using esp_wifi_set_promiscuous_filter() and esp_wifi_set_promiscuous_ctrl_filter(). By default, it will filter all 802.11 data and management frames to the application.

The Wi-Fi sniffer mode can be enabled in the Wi-Fi mode of WIFI_MODE_NULL, or WIFI_MODE_STA, or WIFI_MODE_AP, or WIFI_MODE_APSTA. In other words, the sniffer mode is active when the station is connected to the AP, or when the AP has a Wi-Fi connection. Please note that the sniffer has a **great impact** on the throughput of the station or AP Wi-Fi connection. Generally, we should **NOT** enable the sniffer, when the station/AP Wi-Fi connection experiences heavy traffic unless we have special reasons.

Another noteworthy issue about the sniffer is the callback wifi_promiscuous_cb_t. The callback will be called directly in the Wi-Fi driver task, so if the application has a lot of work to do for each filtered packet, the recommendation is to post an event to the application task in the callback and defer the real work to the application task.

5.25.21 Wi-Fi Multiple Antennas

The Wi-Fi multiple antennas selecting can be depicted as following picture:

```
                ----------
               |Enabled   |
            ___|Antenna 0 |\\                                              ----------
               |__________| \\        GPIO[0] <----> antenna_select[0] ---|          | ---
↪ antenna 0
RX/TX ___                    \\____\  GPIO[1] <----> antenna_select[1] ---| Antenna  | ---
↪ antenna 1
         \      __________   //    /  GPIO[2] <----> antenna_select[2] ---| Switch   | ...
↪  ...
```

(下页继续)

(续上页)

```
         \ ___|Enabled   | //        GPIO[3] <----> antenna_select[3] ---|_________| ---
↪ antenna 15
          \   |Antenna 1 |//
              |__________|
```

ESP32 supports up to sixteen antennas through external antenna switch. The antenna switch can be controlled by up to four address pins - antenna_select[0:3]. Different input value of antenna_select[0:3] means selecting different antenna. E.g. the value '0b1011' means the antenna 11 is selected. The default value of antenna_select[3:0] is '0b0000', it means the antenna 0 is selected by default.

Up to four GPIOs are connected to the four active high antenna_select pins. ESP32 can select the antenna by control the GPIO[0:3]. The API *esp_wifi_set_ant_gpio()* is used to configure which GPIOs are connected to antenna_selects. If GPIO[x] is connected to antenna_select[x], then gpio_config->gpio_cfg[x].gpio_select should be set to 1 and gpio_config->gpio_cfg[x].gpio_num should be provided.

Although up to sixteen anteenas are supported, only one or two antennas can be simultaneously enabled for RX/TX. The API *esp_wifi_set_ant()* is used to configure which antennas are enabled.

The enabled antennas selecting algorithm is also configured by *esp_wifi_set_ant()*. The RX/TX antenna mode can be WIFI_ANT_MODE_ANT0, WIFI_ANT_MODE_ANT1 or WIFI_ANT_MODE_AUTO. If the antenna mode is WIFI_ANT_MODE_ANT0, the enabled antenna 0 is selected for RX/TX data. If the antenna mode is WIFI_ANT_MODE_ANT1, the enabled antenna 1 is selected for RX/TX data. Otherwise, WiFi automatically selects the antenna that has better signal from the enabled antennas.

If the RX antenna mode is WIFI_ANT_MODE_AUTO, the default antenna mode also needs to be set. Because the RX antenna switching only happens when some conditions are met, e.g. the RX antenna starts to switch if the RSSI is lower than -65dBm and if another antenna has better signal etc, RX uses the default antenna if the conditions are not met. If the default antenna mode is WIFI_ANT_MODE_ANT1, the enabled antenna 1 is used as the default RX antenna, otherwise the enabled antenna 0 is used as the default RX antenna.

Some limitations need to be considered:

- The TX antenna can be set to WIFI_ANT_MODE_AUTO only if the RX antenna mode is WIFI_ANT_MODE_AUTO because TX antenna selecting algorithm is based on RX antenna in WIFI_ANT_MODE_AUTO type.
- Currently BT doesn't support the multiple antennas feature, please don't use multiple antennas related APIs.

Following is the recommended scenarios to use the multiple antennas:

- In Wi-Fi mode WIFI_MODE_STA, both RX/TX antenna modes are configured to WIFI_ANT_MODE_AUTO. The WiFi driver selects the better RX/TX antenna automatically.
- The RX antenna mode is configured to WIFI_ANT_MODE_AUTO. The TX antenna mode is configured to WIFI_ANT_MODE_ANT0 or WIFI_ANT_MODE_ANT1. The applications

can choose to always select a specified antenna for TX, or implement their own TX antenna selecting algorithm, e.g. selecting the TX antenna mode based on the channel switch information etc.

- Both RX/TX antenna modes are configured to WIFI_ANT_MODE_ANT0 or WIFI_ANT_MODE_ANT1.

Wi-Fi Multiple Antennas Configuration

Generally, following steps can be taken to configure the multiple antennas:

- Configure which GPIOs are connected to the antenna_selects, for example, if four antennas are supported and GPIO20/GPIO21 are connected to antenna_select[0]/antenna_select[1], the configurations look like:

```
wifi_ant_gpio_config_t config = {
    { .gpio_select = 1, .gpio_num = 20 },
    { .gpio_select = 1, .gpio_num = 21 }
};
```

- Configure which antennas are enabled and how RX/TX use the enabled antennas, for example, if antenna1 and antenna3 are enabled, the RX needs to select the better antenna automatically and uses antenna1 as its default antenna, the TX always selects the antenna3. The configuration looks like:

```
wifi_ant_config_t config = {
    .rx_ant_mode = WIFI_ANT_MODE_AUTO,
    .rx_ant_default = WIFI_ANT_ANT0,
    .tx_ant_mode = WIFI_ANT_MODE_ANT1,
    .enabled_ant0 = 1,
    .enabled_ant1 = 3
};
```

5.25.22 Wi-Fi Channel State Information

Channel state information (CSI) refers to the channel information of a Wi-Fi connection. In ESP32, this information consists of channel frequency responses of sub-carriers and is estimated when packets are received from the transmitter. Each channel frequency response of sub-carrier is recorded by two bytes of signed characters. The first one is imaginary part and the second one is real part. There are up to three fields of channel frequency responses according to the type of received packet. They are legacy long training field (LLTF), high throughput LTF (HT-LTF) and space time block code HT-LTF (STBC-HT-LTF). For different types of packets which are received on channels with different state, the sub-carrier index and total bytes of signed characters of CSI is shown in the following table.

channel	secondary channel	none			below					above				
packet information	signal mode	non HT	HT		non HT	HT				non HT	HT			
	channel bandwidth	20MHz	20MHz		20MHz	20MHz		40MHz		20MHz	20MHz		40MHz	
	STBC	non STBC	non STBC	STBC	non STBC	non STBC	STBC	non STBC	STBC	non STBC	non STBC	STBC	non STBC	STBC
sub-carrier index	LLTF	0~31,-32~-1	0~31,-32~-1	0~31,-32~-1	0~63	0~63	0~63	0~63	0~63	-64~-1	-64~-1	-64~-1	-64~-1	-64~-1
	HT-LTF	•	0~31,-32~-1	0~31,-32~-1	•	0~63	0~62	0~63,-64~-1	0~60,-60~-1	•	-64~-1	-62~-1	0~63,-64~-1	0~60,-60~-1
	STBC-HT-LTF	•	•	0~31,-32~-1	•	•	0~62	•	0~60,-60~-1	•	•	-62~-1	•	0~60,-60~-1
total bytes		128	256	384	128	256	380	384	612	128	256	376	384	612

All of the information in the table can be found in the structure wifi_csi_info_t.

- Secondary channel refers to secondary_channel field of rx_ctrl field.
- Signal mode of packet refers to sig_mode field of rx_ctrl field.
- Channel bandwidth refers to cwb field of rx_ctrl field.
- STBC refers to stbc field of rx_ctrl field.
- Total bytes refers to len field.
- The CSI data corresponding to each Long Training Field(LTF) type is stored in a buffer starting from the buf field. Each item is stored as two bytes: imaginary part followed by real part. The order of each item is the same as the sub-carrier in the table. The order of LTF is: LLTF, HT-LTF, STBC-HT-LTF. However all 3 LTFs may not be present, depending on the channel and packet information (see above).
- If first_word_invalid field of wifi_csi_info_t is true, it means that the first four bytes of CSI data is invalid due to a hardware limitation in ESP32.
- More information like RSSI, noise floor of RF, receiving time and antenna is in the rx_ctrl field.

注解:

- For STBC packet, CSI is provided for every space-time stream without CSD (cyclic shift delay). As each cyclic shift on the additional chains shall be -200ns, only the CSD angle of first space-time stream is recorded in sub-carrier 0 of HT-LTF and STBC-HT-LTF for there is no channel frequency response in sub-carrier 0. CSD[10:0] is 11 bits, ranging from -pi to pi.
- If LLTF, HT-LTF or STBC-HT-LTF is not enabled by calling API *esp_wifi_set_csi_config()*, the total bytes of CSI data will be fewer than that in the table. For example, if LLTF and HT-LTF is not enabled and STBC-HT-LTF is enabled, when a packet is received with the condition above/HT/40MHz/STBC, the total bytes of CSI data is 244 ((61 + 60) * 2 + 2 = 244, the result is aligned to four bytes and the last two bytes is invalid).

5.25.23 Wi-Fi Channel State Information Configure

To use Wi-Fi CSI, the following steps need to be done.

- Select Wi-Fi CSI in menuconfig. It is "Menuconfig –> Components config –> Wi-Fi –> WiFi CSI(Channel State Information)" .
- Set CSI receiving callback function by calling API *esp_wifi_set_csi_rx_cb()*.
- Configure CSI by calling API *esp_wifi_set_csi_config()*.
- Enable CSI by calling API *esp_wifi_set_csi()*.

The CSI receiving callback function runs from Wi-Fi task. So, do not do lengthy operations in the callback function. Instead, post necessary data to a queue and handle it from a lower priority task. Because station does not receive any packet when it is disconnected and only receives packets from AP when it is connected, it is suggested to enable sniffer mode to receive more CSI data by calling *esp_wifi_set_promiscuous()*.

5.25.24 Wi-Fi HT20/40

ESP32 supports Wi-Fi bandwidth HT20 or HT40, it doesn't support HT20/40 coexist. *esp_wifi_set_bandwidth* can be used to change the default bandwidth of station or AP. The default bandwidth for ESP32 station and AP is HT40.

In station mode, the actual bandwidth is firstly negotiated during the Wi-Fi connection. It is HT40 only if both the station and the connected AP support HT40, otherwise it's HT20. If the bandwidth of connected AP is changes, the actual bandwidth is negotiated again without Wi-Fi disconnecting.

Similarly, in AP mode, the actual bandwidth is negotiated between AP and the stations that connect to the AP. It's HT40 if the AP and one of the stations support HT40, otherwise it's HT20.

In station/AP coexist mode, the station/AP can configure HT20/40 seperately. If both station and AP are negotiated to HT40, the HT40 channel should be the channel of station because the station always has higher priority than AP in ESP32. E.g. the configured bandwidth of AP is HT40, the configured primary channel is 6 and the configured secondary channel is 10. The station is connected to an router whose primary channel

is 6 and secondary channel is 2, then the actual channel of AP is changed to primary 6 and secondary 2 automatically.

Theoretically the HT40 can gain better throughput because the maximum raw physicial (PHY) data rate for HT40 is 150Mbps while it's 72Mbps for HT20. However, if the device is used in some special environment, e.g. there are too many other Wi-Fi devices around the ESP32 device, the performance of HT40 may be degraded. So if the applications need to support same or similar scenarios, it's recommended that the bandwidth is always configured to HT20.

5.25.25 Wi-Fi QoS

ESP32 supports all the mandatory features required in WFA Wi-Fi QoS Certification.

Four ACs(Access Category) are defined in Wi-Fi specification, each AC has a its own priority to access the Wi-Fi channel. Moreover a map rule is defined to map the QoS priority of other protocol, such as 802.11D or TCP/IP precedence to Wi-Fi AC.

Below is a table describes how the IP Precedences are mapped to Wi-Fi ACs in ESP32, it also indicates whether the AMPDU is supported for this AC. The table is sorted with priority descending order, namely, the AC_VO has highest priority.

IP Precedence	Wi-Fi AC	Support AMPDU?
6, 7	AC_VO (Voice)	No
4, 5	AC_VI (Video)	Yes
3, 0	AC_BE (Best Effort)	Yes
1, 2	AC_BK (Background)	Yes

The application can make use of the QoS feature by configuring the IP precedence via socket option IP_TOS. Here is an example to make the socket to use VI queue:

```
const int ip_precedence_vi = 4;
const int ip_precedence_offset = 5;
int priority = (ip_precedence_vi << ip_precedence_offset);
setsockopt(socket_id, IPPROTO_IP, IP_TOS, &priority, sizeof(priority));
```

Theoretically the higher priority AC has better performance than the low priority AC, however, it's not a

- For some really important application traffic, can put it into AC_VO queue. Avoid sending big traffic via AC_VO queue. On one hand, the AC_VO queue doesn't support AMPDU and it can't get better performance than other queue if the traffic is big, on the other hand, it may impact the the management frames that also use AC_VO queue.
- Avoid using more than two different AMPDU supported precedences, e.g. socket A uses precedence 0, socket B uses precedence 1, socket C uses precedence 2, this is a bad design because it may

need much more memory. To be detailed, the Wi-Fi driver may generate a Block Ack session for each precedence and it needs more memory if the Block Ack session is setup.

5.25.26 Wi-Fi AMSDU

ESP32 supports receiving AMSDU but doesn't support transmitting AMSDU. The transmitting AMSDU is not necessary since ESP32 has transmitting AMPDU.

5.25.27 Wi-Fi Fragment

ESP32 supports Wi-Fi receiving fragment, but doesn't support Wi-Fi transmitting fragment. The Wi-Fi transmitting fragment will be supported in future release.

5.25.28 WPS Enrolle

ESP32 supports WPS enrollee feature in Wi-Fi mode WIFI_MODE_STA or WIFI_MODE_APSTA. Currently ESP32 supports WPS enrollee type PBC and PIN.

5.25.29 Wi-Fi Buffer Usage

This section is only about the dynamic buffer configuration.

Why Buffer Configuration Is Important

In order to get a robust, high-performance system, we need to consider the memory usage/configuration ve

- the available memory in ESP32 is limited.
- currently, the default type of buffer in LwIP and Wi-Fi drivers is "dynamic", **which means that both the LwIP and Wi-Fi share memory with the application**. Programmers should always keep this in mind; otherwise, they will face a memory issue, such as "running out of heap memory".
- it is very dangerous to run out of heap memory, as this will cause ESP32 an "undefined behavior". Thus, enough heap memory should be reserved for the application, so that it never runs out of it.
- the Wi-Fi throughput heavily depends on memory-related configurations, such as the TCP window size, Wi-Fi RX/TX dynamic buffer number, etc.
- the peak heap memory that the ESP32 LwIP/Wi-Fi may consume depends on a number of factors, such as the maximum TCP/UDP connections that the application may have, etc.

- the total memory that the application requires is also an important factor when considering memory configuration.

Due to these reasons, there is not a good-for-all application configuration. Rather, we have to consider memory configurations separately for every different application.

Dynamic vs. Static Buffer

The default type of buffer in Wi-Fi drivers is "dynamic". Most of the time the dynamic buffer can significantly save memory. However, it makes the application programming a little more difficult, because in this case the application needs to consider memory usage in Wi-Fi.

lwIP also allocates buffers at the TCP/IP layer, and this buffer allocation is also dynamic. See *lwIP documentation section about memory use and performance*.

Peak Wi-Fi Dynamic Buffer

The Wi-Fi driver supports several types of buffer (refer to *Wi-Fi Buffer Configure*). However, this section is about the usage of the dynamic Wi-Fi buffer only. The peak heap memory that Wi-Fi consumes is the **theoretically-maximum memory** that the Wi-Fi driver consumes. Generally, the peak memory depends on:

- the number of dynamic rx buffers that are configured: wifi_rx_dynamic_buf_num
- the number of dynamic tx buffers that are configured: wifi_tx_dynamic_buf_num
- the maximum packet size that the Wi-Fi driver can receive: wifi_rx_pkt_size_max
- the maximum packet size that the Wi-Fi driver can send: wifi_tx_pkt_size_max

So, the peak memory that the Wi-Fi driver consumes can be calculated with the following formula:

wifi_dynamic_peek_memory = (wifi_rx_dynamic_buf_num * wifi_rx_pkt_size_max) + (wifi_tx_dynamic_buf_num * wifi_tx_pkt_size_max)

Generally, we do not need to care about the dynamic tx long buffers and dynamic tx long long buffers, because they are management frames which only have a small impact on the system.

5.25.30 Wi-Fi Menuconfig

Wi-Fi Buffer Configure

If you are going to modify the default number or type of buffer, it would be helpful to also have an overview of how the buffer is allocated/freed in the data path. The following diagram shows this process in the TX direction:

Description:

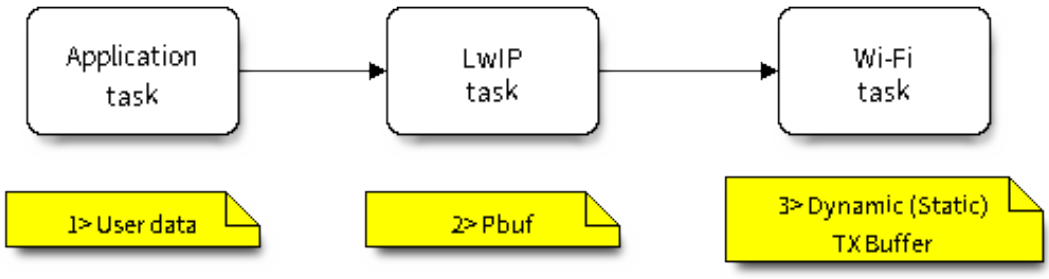


图 31: TX Buffer Allocation

- The application allocates the data which needs to be sent out.
- The application calls TCPIP-/Socket-related APIs to send the user data. These APIs will allocate a PBUF used in LwIP, and make a copy of the user data.
- When LwIP calls a Wi-Fi API to send the PBUF, the Wi-Fi API will allocate a "Dynamic Tx Buffer" or "Static Tx Buffer", make a copy of the LwIP PBUF, and finally send the data.

The following diagram shows how buffer is allocated/freed in the RX direction:

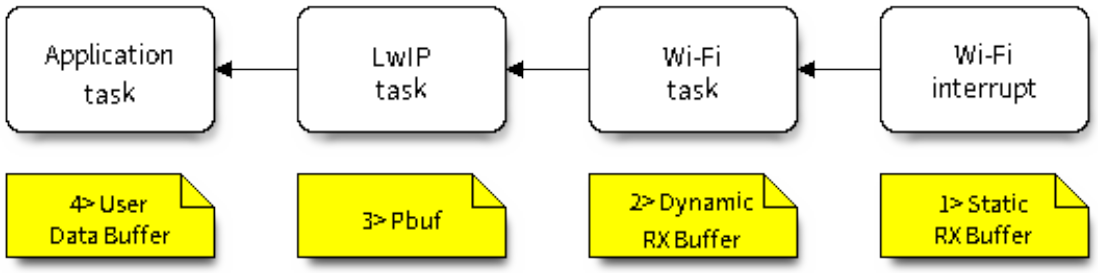


图 32: RX Buffer Allocation

Description:

- The Wi-Fi hardware receives a packet over the air and puts the packet content to the "Static Rx Buffer", which is also called "RX DMA Buffer".
- The Wi-Fi driver allocates a "Dynamic Rx Buffer", makes a copy of the "Static Rx Buffer", and returns the "Static Rx Buffer" to hardware.
- The Wi-Fi driver delivers the packet to the upper-layer (LwIP), and allocates a PBUF for holding the "Dynamic Rx Buffer".
- The application receives data from LwIP.

The diagram shows the configuration of the Wi-Fi internal buffer.

Buffer Type	Alloc Type	Default	Configurable	Description
Static RX Buffer (Hardware RX Buffer)	Static	10 * 1600 Bytes	Yes	This is a kind of DMA memory. It is initialized in esp_wifi_init() and freed in esp_wifi_deinit(). The 'Static Rx Buffer' forms the hardware receiving list. Upon receiving a frame over the air, hardware writes the frame into the buffer and raises an interrupt to the CPU. Then, the Wi-Fi driver reads the content from the buffer and returns the buffer back to the list. If the application want to reduce the the memory statically allocated by Wi-Fi, they can reduce this value from 10 to 6 to save 6400 Bytes memory. It' s not recommended to reduce the configuration to a value less than 6 unless the AMPDU feature is disabled.
Dynamic RX Buffer	Dynamic	32	Yes	The buffer length is variable and it depends on the received frames' length. When the Wi-Fi driver receives

Wi-Fi NVS Flash

If the Wi-Fi NVS flash is enabled, all Wi-Fi configurations set via the Wi-Fi APIs will be stored into flash, and the Wi-Fi driver will start up with these configurations next time it powers on/reboots. However, the application can choose to disable the Wi-Fi NVS flash if it does not need to store the configurations into persistent memory, or has its own persistent storage, or simply due to debugging reasons, etc.

Wi-Fi AMPDU

ESP32 supports both receiving and transmitting AMPDU, the AMPDU can greatly improve the Wi-Fi throughput.

Generally, the AMPDU should be enabled. Disabling AMPDU is usually for debugging purposes.

5.25.31 Troubleshooting

Please refer to a separate document with *乐鑫 Wireshark 使用指南*.

乐鑫 Wireshark 使用指南

[English]

1. 概述

1.1 什么是 Wireshark ?

Wireshark（原称 Ethereal）是一个网络封包分析软件。网络封包分析软件的功能是撷取网络封包，并尽可能显示出最为详细的网络封包资料。Wireshark 使用 WinPCAP 作为接口，直接与网卡进行数据报文交换。

网络封包分析软件的功能可想像成“电工技师使用电表来量测电流、电压、电阻”的工作，只是将场景移植到网络上，并将电线替换成网线。

在过去，网络封包分析软件是非常昂贵，或是专门属于营利用的软件。Wireshark 的出现改变了这一切。

在 GNU GPL 通用许可证的保障范围下，使用者可以以免费的代价取得软件与其源代码，并拥有针对其源代码修改及客制化的权利。

Wireshark 是目前全世界最广泛的网络封包分析软件之一。

1.2 Wireshark 的主要应用

下面是 Wireshark 一些应用的举例：

- 网络管理员用来解决网络问题

- 网络安全工程师用来检测安全隐患
- 开发人员用来测试协议执行情况
- 用来学习网络协议

除了上面提到的，Wireshark 还可以用在其它许多场合。

1.3 Wireshark 的特性

- 支持 UNIX 和 Windows 平台
- 在接口实时捕捉包
- 能详细显示包的详细协议信息
- 可以打开/保存捕捉的包
- 可以导入导出其他捕捉程序支持的包数据格式
- 可以通过多种方式过滤包
- 多种方式查找包
- 通过过滤以多种色彩显示包
- 创建多种统计分析
- 等等

1.4 Wireshark 的“能”与“不能”?

- **捕捉多种网络接口**

  Wireshark 可以捕捉多种网络接口类型的包，哪怕是无线局域网接口。

- **支持多种其它程序捕捉的文件**

  Wireshark 可以打开多种网络分析软件捕捉的包。

- **支持多格式输出**

  Wireshark 可以将捕捉文件输出为多种其他捕捉软件支持的格式。

- **对多种协议解码提供支持**

  Wireshark 可以支持许多协议的解码。

- **Wireshark 不是入侵检测系统**

  如果您的网络中存在任何可疑活动，Wireshark 并不会主动发出警告。不过，当您希望对这些可疑活动一探究竟时，Wireshark 可以发挥作用。

- **Wireshark 不会处理网络事务，它仅仅是“测量”（监视）网络**

  Wireshark 不会发送网络包或做其它交互性的事情（名称解析除外，但您也可以禁止解析）。

2. 如何获取 Wireshark

官网链接：https://www.wireshark.org/download.html

Wireshark 支持多种操作系统，请在下载安装文件时，注意选择与您所用操作系统匹配的安装文件。

3. 使用步骤

本文档仅以 Linux 系统下的 Wireshark（版本号：2.2.6）为例。

1) 启动 Wireshark

Linux 下，可编写一个 Shell 脚本，运行该文件即可启动 Wireshark 配置抓包网卡和信道。Shell 脚本如下：

```
ifconfig $1 down
iwconfig $1 mode monitor
iwconfig $1 channel $2
ifconfig $1 up
Wireshark&
```

脚本中有两个参数：`$1` 和 `$2`，分别表示网卡和信道，例如，`./xxx.sh wlan0 6`（此处，`wlan0` 即为抓包使用的网卡，后面的数字 `6` 即为 AP 或 soft-AP 所在的 channel）。

2) 运行 Shell 脚本打开 Wireshark，会出现 Wireshark 抓包开始界面

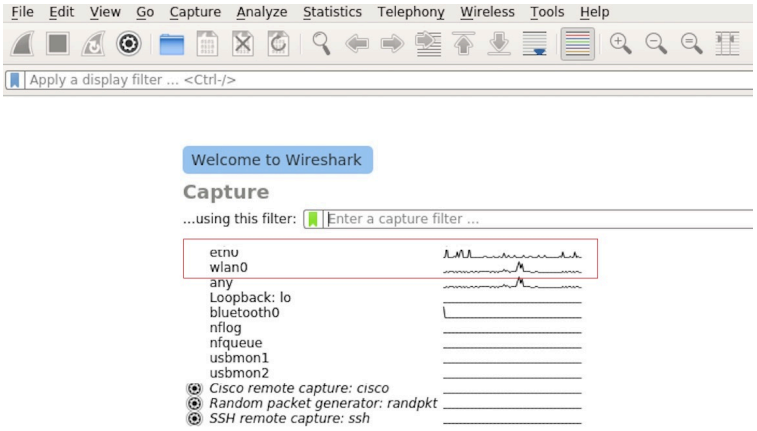


图 33: Wireshark 抓包界面

3) 选择接口，开始抓包

从上图红色框中可以看到有多个接口，第一个为本地网卡，第二个为无线网络。

可根据自己的需求选取相应的网卡，本文是以利用无线网卡抓取空中包为例进行简单说明。

双击 *wlan0* 即可开始抓包。

4) 设置过滤条件

抓包过程中会抓取到同信道所有的空中包，但其实很多都是我们不需要的，因此很多时候我们会设置抓包的过滤条件从而得到我们想要的包。

下图中红色框内即为设置 filter 的位置。

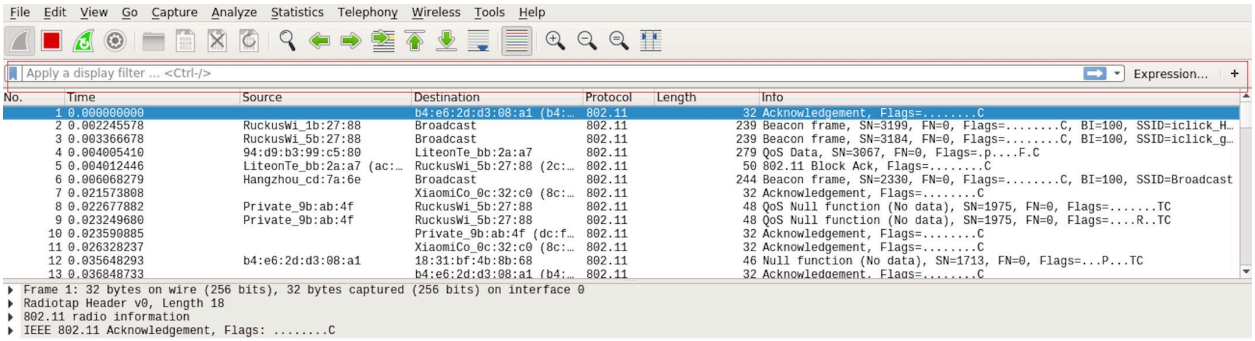


图 34: 设置 Wireshark 过滤条件

点击 *Filter* 按钮（下图的左上角蓝色按钮）会弹出 *display filter* 对话框。

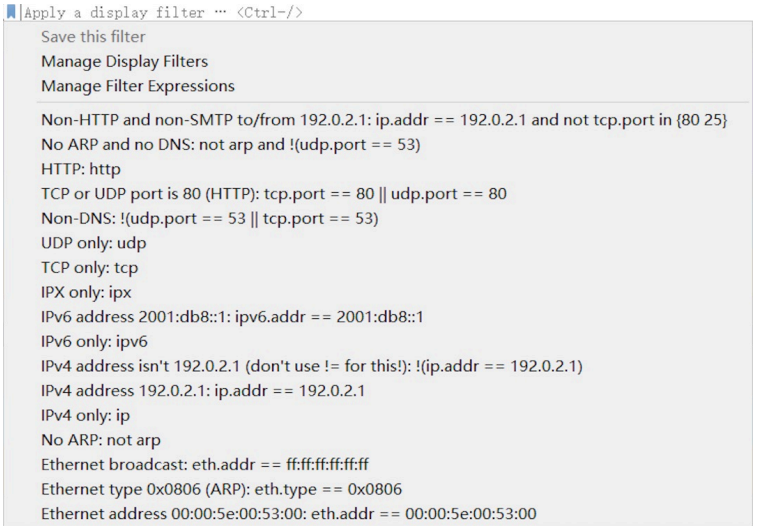


图 35: *Display Filter* 对话框

点击 *Expression* 按钮，会出现 *Filter Expression* 对话框，在此你可以根据需求进行 filter 的设置。

最直接的方法：直接在工具栏上输入过滤条件。

点击在此区域输入或修改显示的过滤字符，在输入过程中会进行语法检查。如果您输入的格式不正确，或者未输入完成，则背景显示为红色。直到您输入合法的表达式，背景会变为绿色。你可以点击下拉列表选择您

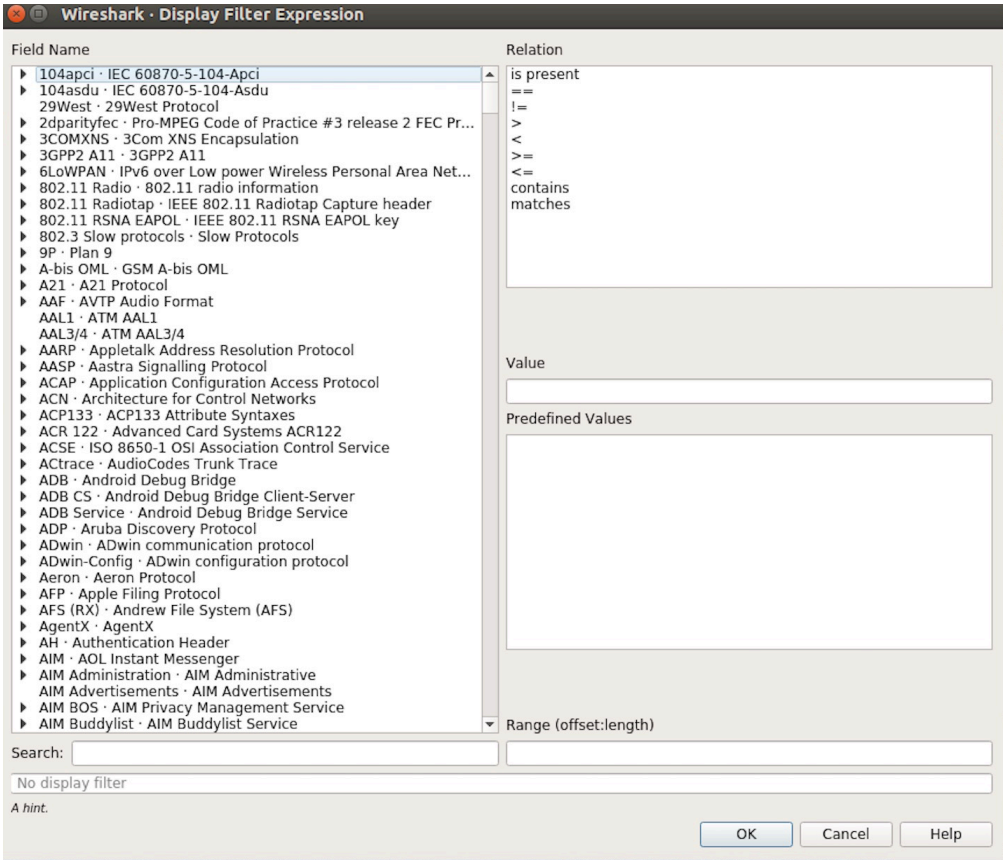


图 36: *Filter Expression* 对话框

图 37: 过滤条件工具栏

先前键入的过滤字符。列表会一直保留，即使您重新启动程序。

例如：下图所示，直接输入 2 个 MAC 作为过滤条件，点击 *Apply* （即图中的蓝色箭头），则表示只抓取 2 个此 MAC 地址之间的交互的包。

图 38: 在过滤条件工具栏中运用 MAC 地址过滤示例

5) 封包列表

若想查看包的具体的信息只需要选中要查看的包，在界面的下方会显示出包的具体的格式和包的内容。

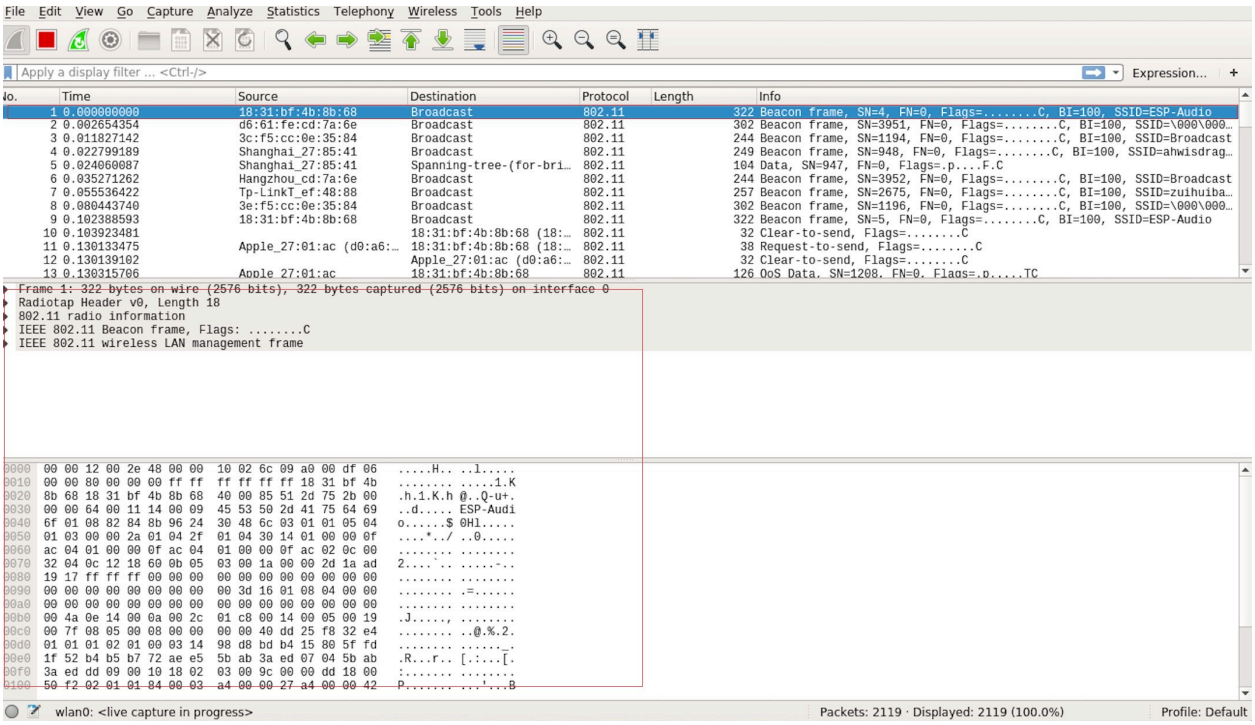


图 39: 封包列表具体信息示例

如上图所示，我要查看第 1 个包，选中此包，图中红色框中即为包的具体内容。

6) 停止/开始包的捕捉

若要停止当前抓包，点击下图的红色按钮即可。

若要重新开始抓包，点击下图左上角的蓝色按钮即可。

7) 保存当前捕捉包

Linux 下，可以通过依次点击 “File” -> “Export Packet Dissections” -> “As Plain Text File” 进行保存。

上图中，需要注意的是，选择 *All packets*、*Displayed* 以及 *All expanded* 三项。

Wireshark 捕捉的包可以保存为其原生格式文件（libpcap），也可以保存为其他格式（如.txt 文件）供其他工具进行读取分析。

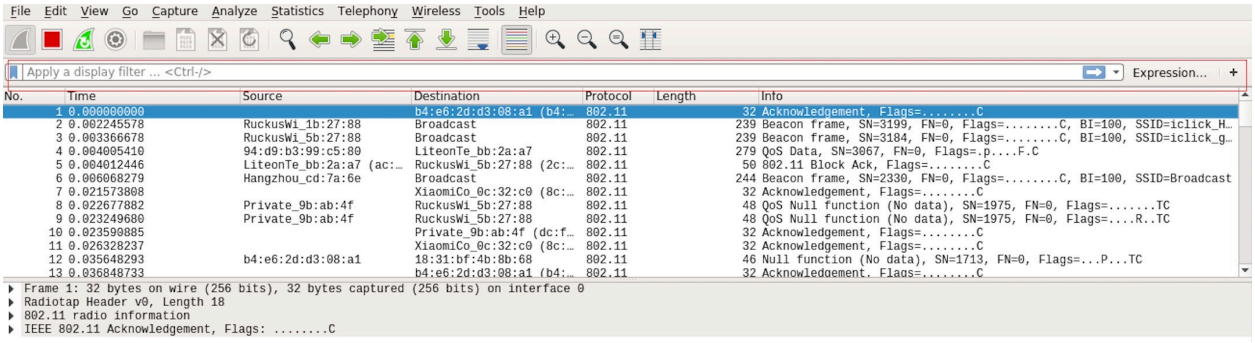


图 40: 停止包的捕捉

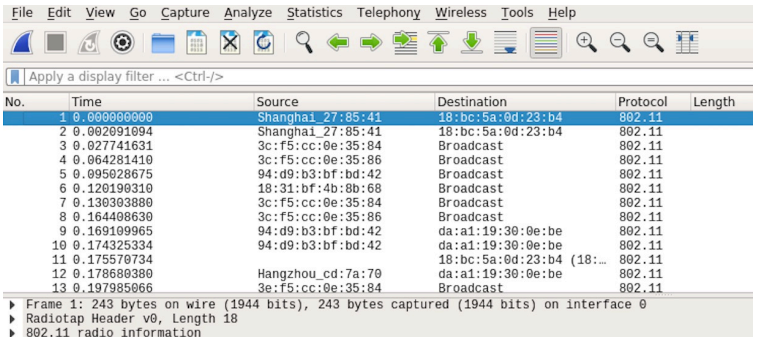


图 41: 开始或继续包的捕捉

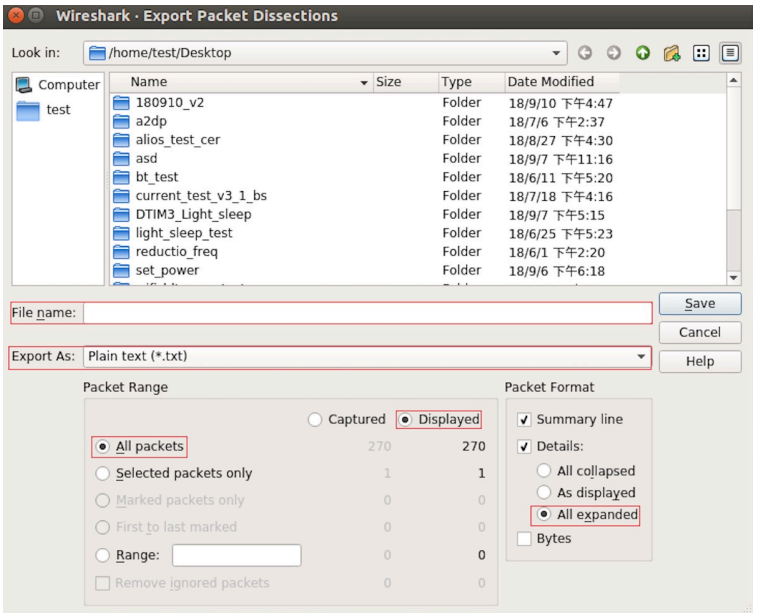


图 42: 保存捕捉包

5.26 ESP-BLE-MESH

[English]

5.26.1 概述

蓝牙 mesh 网络实现了无线设备的“多对多”通讯，其可用于建立包含大量设备的网络。

设备能将数据中继至不在初始设备无线电覆盖范围内的其他设备。这样，mesh 网络就能够覆盖非常大的物理区域，并且囊括大量设备。Mesh 网络非常适用于楼宇自动化、传感器网络和其他物联网解决方案，这些情景下数以十计、百计、千计的设备需要与其他设备进行安全可靠的通信。

蓝牙 mesh 并非无线通信技术，而是一种网络技术。该技术基于一种无线通讯协议栈，即低功耗蓝牙。

ESP-BLE-MESH 基于 Zephyr 蓝牙 Mesh 协议栈的顶端，其实现支持设备配网和节点控制，同时也实现了代理、中继、低功耗和朋友等节点功能。

有关 ESP-BLE-MESH 架构实现的信息，请参见*ESP-BLE-MESH 架构*；有关各自 API 的信息，请参见*ESP-BLE-MESH API Reference*。

ESP-BLE-MESH 的实现和认证基于最新的 Mesh Profile v1.0.1 。有关 ESP-BLE-MESH 认证的细节，请参考 此处 。

注解： 如果您在寻找 ESP32 基于 Wi-Fi 的 mesh 方案，请查阅乐鑫的另一款产品 ESP-MESH。更多相关信息及文档，请参见:doc:*ESP-MESH <../../api-reference/network/esp_mesh>*。

5.26.2 ESP-BLE-MESH 快速入门

该章节旨在帮助您基于乐鑫的 ESP32 开发板搭建 ESP-BLE-MESH 网络。

我们将会展示如何搭建并运行一个包含 3 个节点的小型 ESP-BLE-MESH 网络，其中包含设备配网、节点配置，以及向特定节点上的 Generic OnOff Server Model 发送开关灯命令。

如果您是第一次接触 ESP-IDF，请参见 esp-idf :doc:../../get-started/index 来设置开发环境，编译、烧写和运行示例应用程序。

硬件及软件准备

硬件：

- 3 块 ESP32 开发板，请参见*options*。
- 连接开发板的 USB 线。
- ESP-IDF 开发环境。

- 运行 Android 或 iOS 的手机或平板。

软件：

- 下载至 ESP32 开发板的示例应用 bluetooth/esp_ble_mesh/ble_mesh_node。
- 手机 App: **nRF Mesh** Android 或 iOS 版本。除 nRF Mesh 的 App，以下 App 也支持 ESP-BLE-MESH：
  - EspBleMesh Android App
  - Silicon Labs Android 或 iOS App

安装

以下详细步骤可指导您完成安装过程。

步骤 1. 检查硬件

ESP32-DevKitC 和 ESP-WROVER-KIT 开发板均支持 ESP-BLE-MESH。您可以通过 menuconfig: `idf.py menuconfig` > `Example Configuration` > `Board selection for ESP-BLE-MESH` 选择特定的开发板。

注解： 如果您打算使用 ESP32-DevKitC 开发板，请将 RGB 灯焊接至 GPIO 管脚 25、26 和 27。

步骤 2. 配置软件

进入 bluetooth/esp_ble_mesh/ble_mesh_node 示例文件夹，运行 `idf.py menuconfig` 选择所使用的开发板，然后运行 `idf.py build` 编译示例。

步骤 3. 下载应用

> bluetooth/esp_ble_mesh/ble_mesh_node 示例编译成功后，用户可以运行 `idf.py flash` 将编译生成的二进制文件下载至 3 块开发板中。

当开发板上电后，RGB 灯会变为 **绿色**。

步骤 4. 设备配网

在该章节中，我们将使用 **nRF Mesh Android** App 演示如何配网设备。用户也可以从 App Store 下载其 iOS 版本。

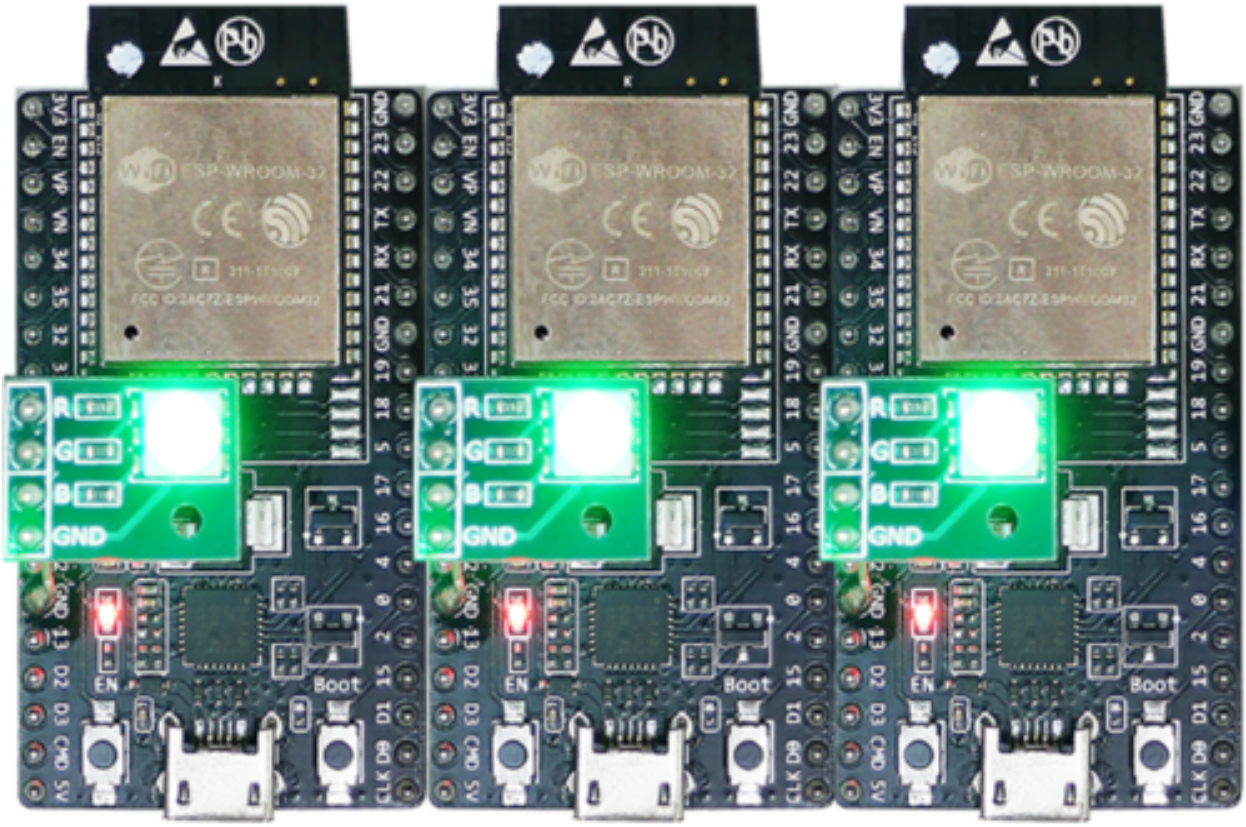


图 43: ESP-BLE-MESH 设备上电

4.1 扫描 (scanner)

扫描 (Scanner) 是 nRF Mesh App 搜索蓝牙通信范围内未配网设备的功能。打开 App，点击底部的扫描按钮 **Scanner**。App 就会开始扫描设备，很快，我们便可在屏幕上看到 3 个未配网设备。

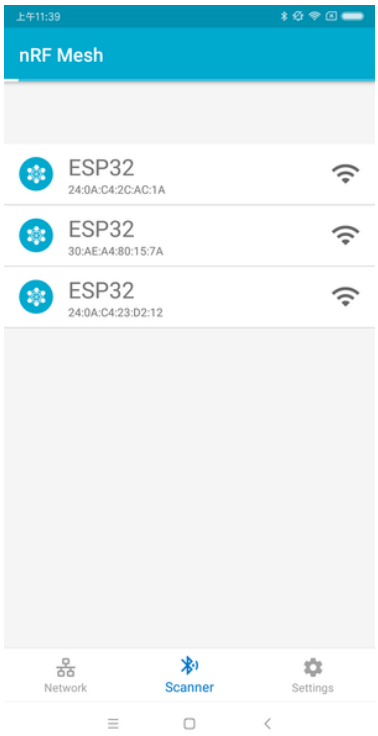


图 44: nRF Mesh - 扫描

4.2 识别

用户可以选择任何一个未配网设备，此时 App 会尝试和该设备建立连接。连接成功（有时可能需要尝试多次），且发现相应的 ESP-BLE-MESH GATT 服务后，用户可以在屏幕中看到识别按钮 **IDENTIFY**。IDENTIFY 操作告诉用户哪个设备将被配网。

注解: IDENTIFY 需要设备侧的支持，然后才能用来识别当前正在配网的设备。当前如果点击识别按钮 **IDENTIFY**，除了串口输出的 log，在当前的 example 中设备侧不会有其他现象。

点击识别按钮 **IDENTIFY** 后，用户可以看到配网按钮 **PROVISION**。

4.3 配网

点击配网按钮 **PROVISION**，App 会开始配网设备。当设备配网成功后，开发板上的 RGB 灯会熄灭，此时 App 会执行以下几个步骤：

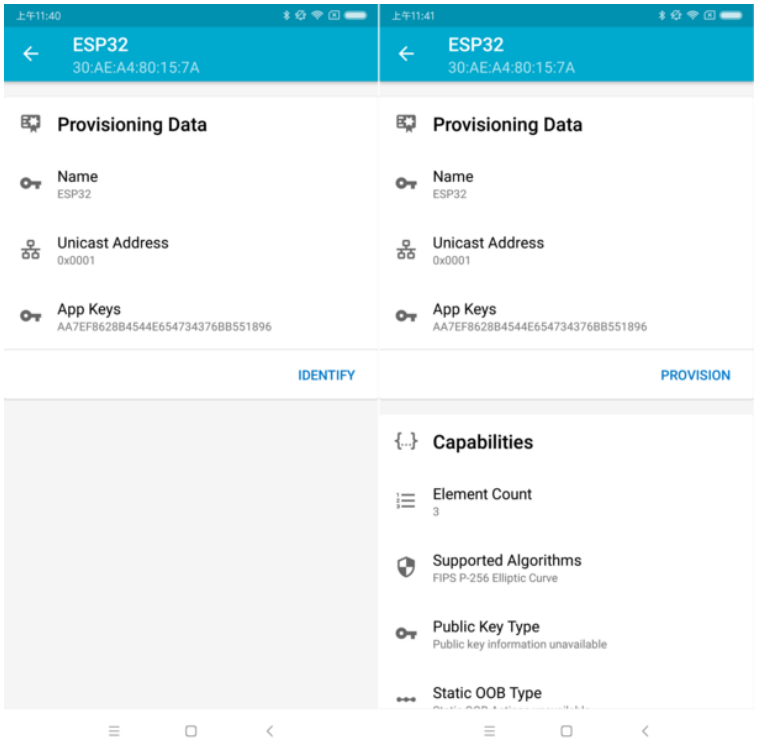


图 45: nRF Mesh - 识别 - 配网

1. 和该节点（设备配网后成为节点）断开连接
2. 尝试和该节点重新建立连接
3. 连接成功并且发现了相应的 ESP-BLE-MESH GATT 服务
4. 获取节点的 Composition Data 并且给该节点添加 AppKey

当以上所有的步骤完成后，节点初始配置完成. 此时点击 **OK**，用户可以看见节点的单播地址分配成功，并且其 Composition Data 也被成功解析.

有时在上述步骤 2 中，App 可能与节点连接失败。这种情况下，用户点击 **OK** 后可以看到，节点只有单播地址被成功分配，Composition data 并没有被成功获取。此时用户需要点击右上角的连接按钮 **CONNECT**，屏幕上会显示原先配网的节点，用户需要选择该节点并与其建立连接。

连接成功后，App 会显示获取 Composition Data 以及添加 AppKey 的按钮。

如果该设备是 App 配网的第二个或第三个节点，此时点击连接按钮 **CONNECT** 后，用户可以在屏幕中看到 2 个或 3 个节点。这种情况下，用户可以选择其中的任何一个节点建立连接，连接成功后可以返回至主界面选择需要配置的节点。

这里给出了一个 3 个节点的示例。

- 左侧图片表示第三个设备成功配网，但是 App 没有和其成功建立连接。当 App 尝试去重新连接第三个节点时，界面上会出现 3 个节点。
- 右侧图片表示和节点成功建立连接后，App 显示这 3 个节点的信息。用户可以看到 App 已经获取了

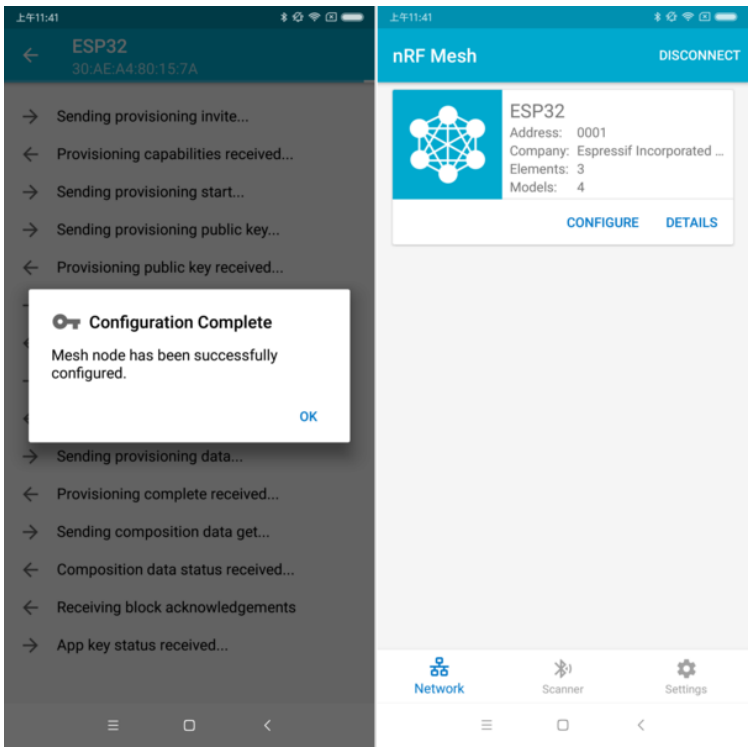


图 46: nRF Mesh - 配置完成

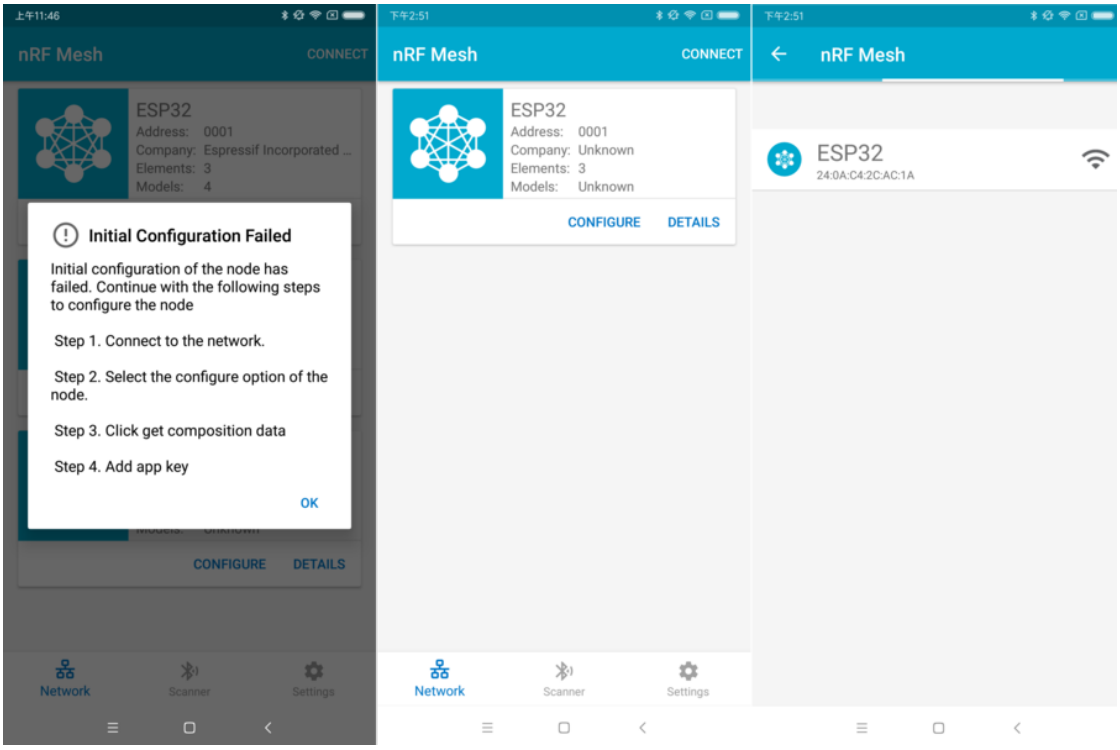


图 47: nRF Mesh - 初始配置失败

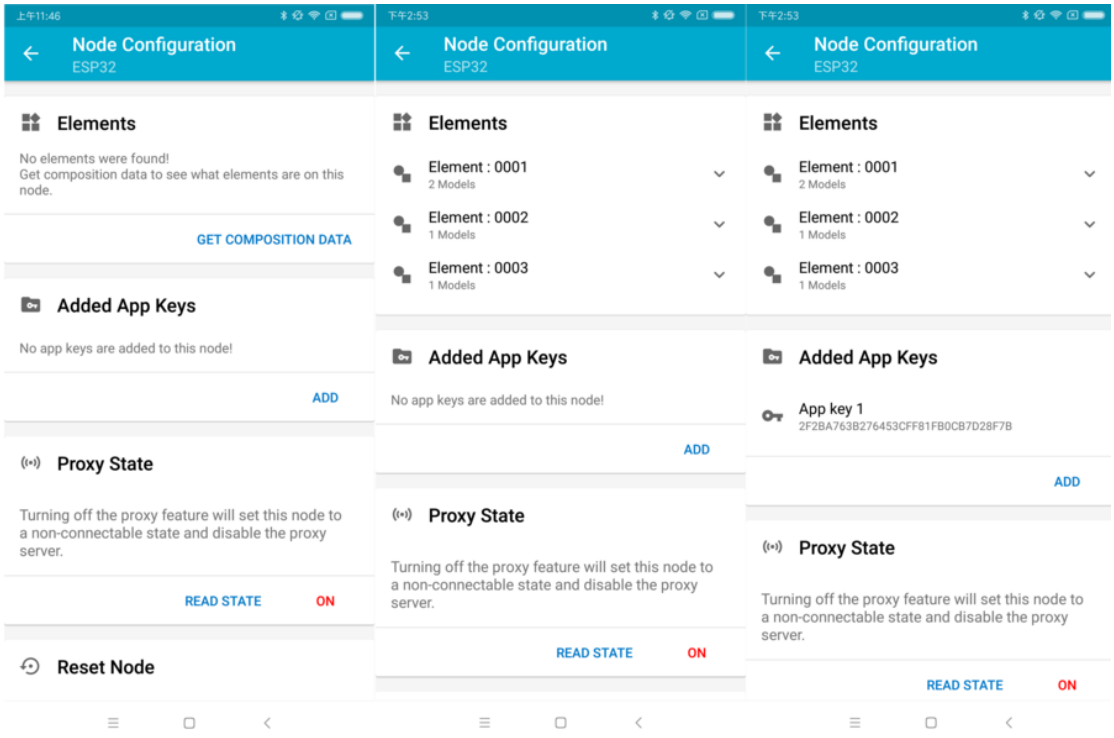


图 48: nRF Mesh - 重连 - 初始配置

第一个和第二个节点的 Composition Data，但是对于第三个节点，只有单播地址被成功分配而节点的 Composition Data 未知。

4.4 配置

当成功配网和初始配置完成后，用户可以配置节点的其余信息，例如将 AppKey 绑定至每个元素 (element) 的每个模型 (model) 中、设置模型的发布信息等。

下图展示了如何将 AppKey 绑定至 Primary Element 中的 Generic OnOff Server Model 上。

注解: 用户不需要将 AppKey 绑定至 Configuration Server Model，因为该模型使用 DevKey 在 Upper Transport Layer 中对消息进行加密。

Step 5. 运行网络

当 3 个元素中的 Generic OnOff Server Models 均成功绑定 AppKey 后，用户可以使用 App 开关 RBG 灯。

在 bluetooth/esp_ble_mesh/ble_mesh_node 示例中，第一个 Generic OnOff Server Model 用来控制 **红色**，第二个用来控制 **绿色**，同时第三个用来控制 **蓝色**.

下图展示了打开了不同色灯的开发板。

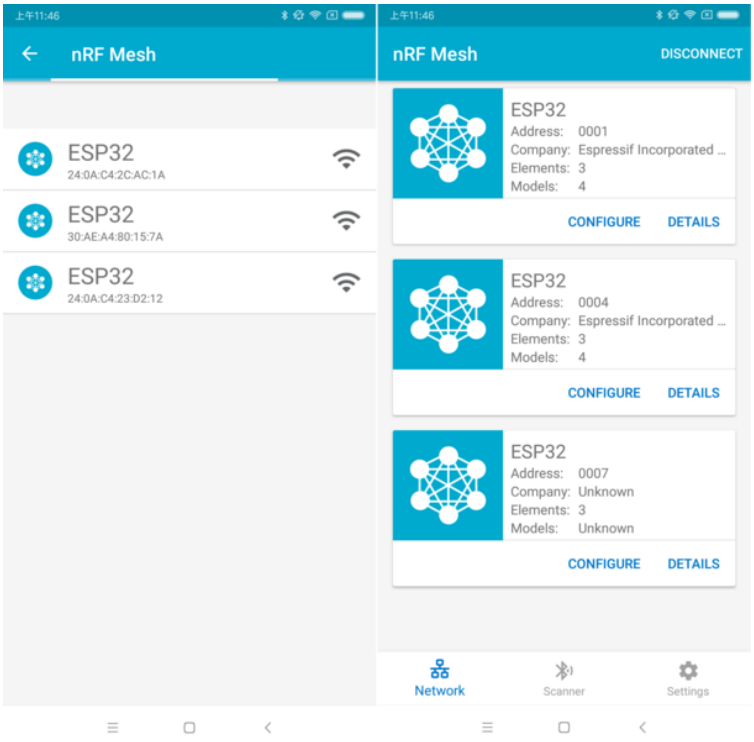


图 49: nRF Mesh - 重连 - 3 个节点

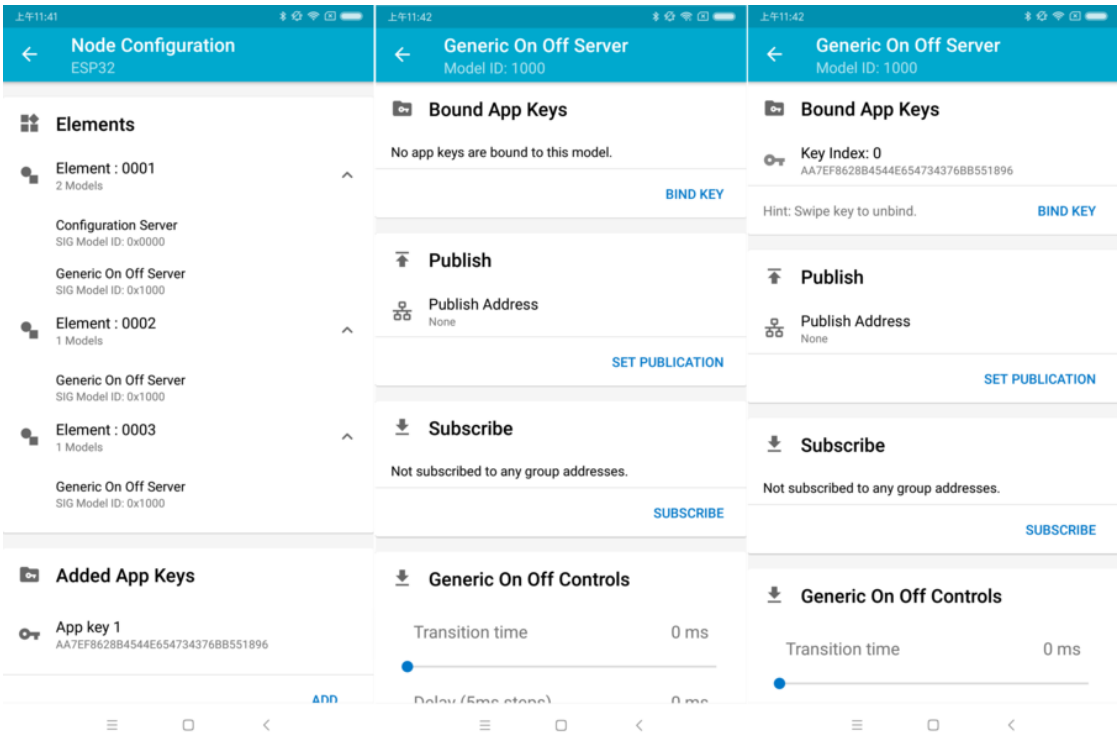


图 50: nRF Mesh - Model Bind AppKey

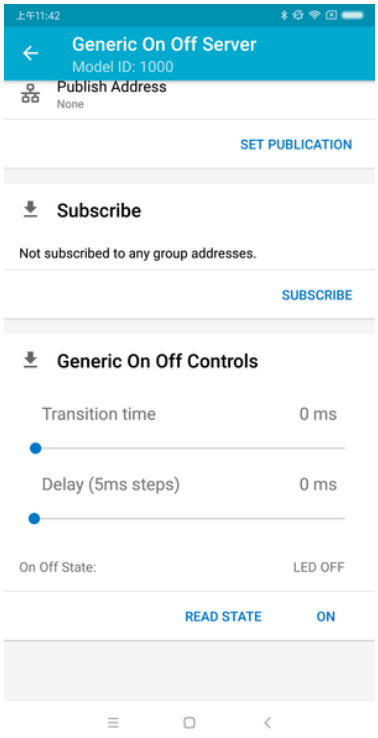


图 51: nRF Mesh - 通用开关控制

注解: 对于 **nRF Mesh** iOS App [version 1.0.4]，当节点包含超过一个元素时，App 表现不正确。如果用户尝试打开或者关闭第 2 个或第 3 个 Generic OnOff Server Model，App 会将相应的消息发送至第 1 个 Generic OnOff Server Model。

5.26.3 ESP-BLE-MESH 示例

- ESP-BLE-MESH 节点 - 展示了将 ESP-BLE-MESH 作为拥有 Configuration Server model 和 Generic OnOff Server model 的节点设备的用法。然后，ESP-BLE-MESH Provisioner 可以配网设备，控制表示开/关状态的 RGB 灯，示例请见 example code 。
- ESP-BLE-MESH 客户端模型 - 展示了 Generic OnOff Client model 如何在节点内工作。节点拥有 Configuration Server model、Generic OnOff Server model 和 Generic OnOff Client model，示例请见: example code 。
- ESP-BLE-MESH Provisioner - 展示了设备如何充当 ESP-BLE-MESH Provisioner 以配网设备。Provisioner 拥有 Configuration Server model、Configuration Client model 和 Generic OnOff Client model，示例请见 example code 。
- ESP-BLE-MESH 快速配网 - Client 和:example_file‘Server <bluetooth/esp_ble_mesh/ble_mesh_fast_provision/fast_
  - 该示例用于演示快速配网。配网 100 个设备费时不超过 60 秒，示例请见: example client code 和

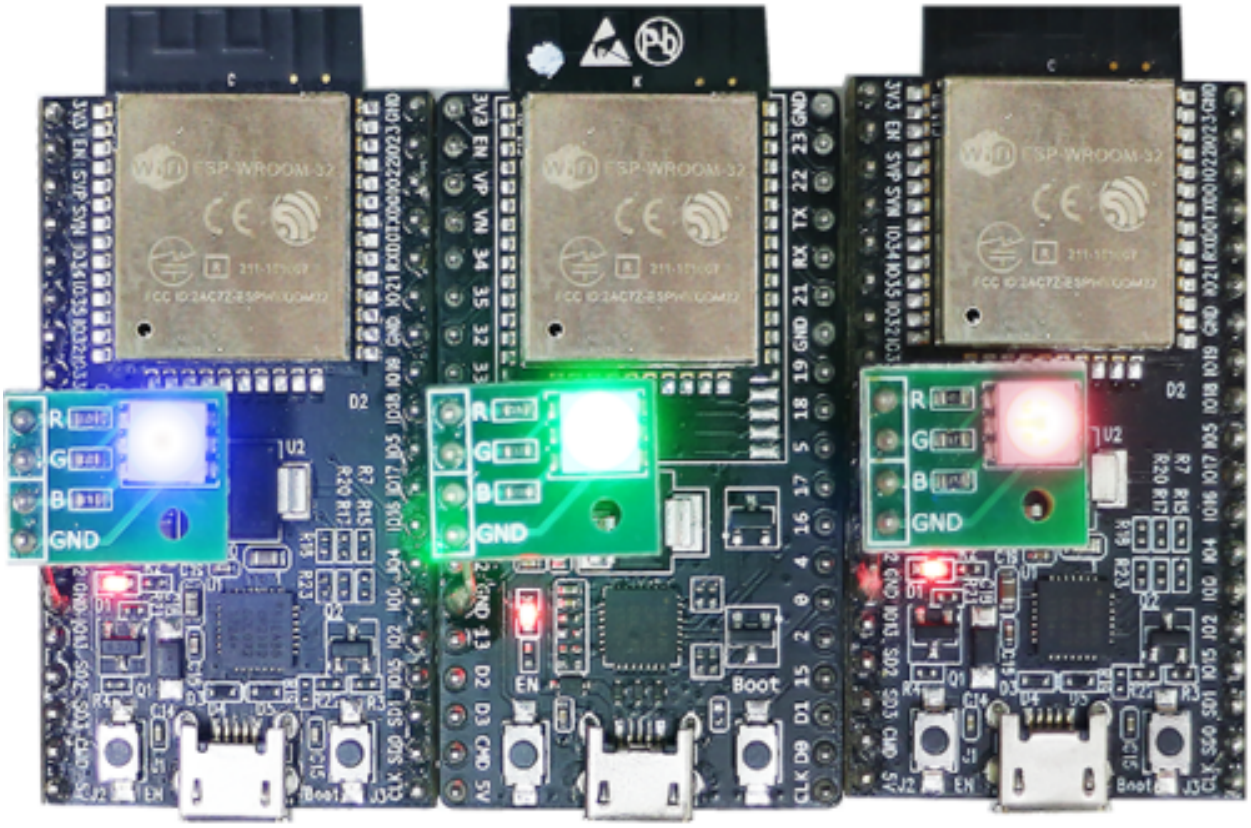


图 52: 3 个上电的 ESP-BLE-MESH 节点

example server code 。

- Wi-Fi 和 ESP-BLE-MESH 共存 - 该示例用于演示 Wi-Fi 和 ESP-BLE-MESH 共存的功能。简而言之，用户可在运行 ESP-BLE-MESH 时使用 Wi-Fi，示例请见 example code 。
- ESP-BLE-MESH 节点控制台 - 该演示实现 ESP-BLE-MESH 节点的基本功能。在演示中，Provisioner 可以扫描、验证节点，节点可以回复 Provisioner 的获取/设置消息，示例请见：example node code 和 example provisioner code 。

5.26.4 ESP-BLE-MESH 演示视频

- Provisioning of ESP-BLE-MESH nodes using Smartphone App
- Espressif Fast Provisioning using ESP-BLE-MESH App
- Espressif ESP-BLE-MESH and Wi-Fi Coexistence

5.26.5 ESP-BLE-MESH 常见问题手册

- *1. Provisioner 开发*
- *2. 节点开发*
- *3. ESP-BLE-MESH 和 Wi-Fi 共存*
- *4. 快速配网*
- *5. Log 帮助*
- *6. 示例帮助*
- *7. 其他*

5.26.6 相关文档

ESP-BLE-MESH Feature List

Supported Features

Mesh Core

- **Provisioning: Node Role**
  - PB-ADV and PB-GATT
  - OOB Authentication
- **Provisioning: Provisioner Role**
  - PB-ADV and PB-GATT

  - OOB Authentication

- **Networking**
  - Relay
  - Segmentation and Reassembly
  - Key Refresh Procedure
  - IV Update Procedure
  - Friend
  - Low Power
  - Proxy Server
  - Proxy Client
- **Multiple Client Models Run Simultaneously**
  - Support multiple client models send packets to different nodes simultaneously
  - No blocking between client model and server model
- **NVS Storing**
  - Store provisioning and configuration information of ESP-BLE-MESH Node

Mesh Models

- **Foundation models**
  - Configuration Server model
  - Configuration Client model
  - Health Server model
  - Health Client model
- **Generic client models**
  - Generic OnOff Client
  - Generic Level Client
  - Generic Default Transition Time Client
  - Generic Power OnOff Client
  - Generic Power Level Client
  - Generic Battery Client
  - Generic Location Client

  - Generic Property Client

- **Sensor client models**
  - Sensor Client
- **Time and Scenes client models**
  - Time Client
  - Scene Client
  - Scheduler Client
- **Lighting client models**
  - Light Lightness Client
  - Light CTL Client
  - Light HSL Client
  - Light xyL Client
  - Light LC Client
- **Generic server models**
  - Generic OnOff Server
  - Generic Level Server
  - Generic Default Transition Time Server
  - Generic Power OnOff Server
  - Generic Power OnOff Setup Server
  - Generic Power Level Server
  - Generic Power Level Setup Server
  - Generic Battery Server
  - Generic Location Server
  - Generic Location Setup Server
  - Generic User Property Server
  - Generic Admin Property Server
  - Generic Manufacturer Property Server
  - Generic Client Property Server
- **Sensor server models**
  - Sensor Server

  - Sensor Setup Server

- **Time and Scenes server models**
  - Time Server
  - Time Setup Server
  - Scene Server
  - Scene Setup Server
  - Scheduler Server
  - Scheduler Setup Server
- **Lighting server models**
  - Light Lightness Server
  - Light Lightness Setup Server
  - Light CTL Server
  - Light CTL Temperature Server
  - Light CTL Setup Server
  - Light HSL Server
  - Light HSL Hue Server
  - Light HSL Saturation Server
  - Light HSL Setup Server
  - Light xyL Server
  - Light xyL Setup Server
  - Light LC Server
  - Light LC Setup Server

Mesh Applications

- **ESP-BLE-MESH Node**
  - Tutorial
  - Tutorial
  - Example
- **ESP-BLE-MESH Provisioner**
  - Tutorial

  - Example

- **ESP-BLE-MESH Fast Provisioning**
  - Fast Provisioning Client Model Tutorial
  - Fast Provisioning Server Model Tutorial
  - Example
  - Demo Video
- **ESP-BLE-MESH and Wi-Fi Coexistence**
  - Tutorial
  - Example
  - Demo Video
- **ESP-BLE-MESH Console Commands**
  - Example

Future Release Features

Mesh Core

- Provisioner NVS Storage

Mesh Applications

- Fast OTA
- Friendship

ESP-BLE-MESH 架构

[English]

本文档将介绍 ESP-BLE-MESH 的架构概览、架构实现和辅助程序。

- ESP-BLE-MESH 架构概览
  - 描述了 ESP-BLE-MESH 架构的 5 大部分及每个部分的功能。
- ESP-BLE-MESH 架构实现
  - 描述了 ESP-BLE-MESH 文件的基本功能、文件与 ESP-BLE-MESH 架构的对应关系及文件间调用的接口。
- ESP-BLE-MESH 辅助程序

  - 描述了 ESP-BLE-MESH 的辅助程序，比如 Mesh 网络管理，Mesh 特性等。

1. ESP-BLE-MESH 架构概览

目前，ESP-BLE-MESH 已经实现了 Mesh Profile 的大多数功能及 Mesh Model 规范中定义的所有 Client Model。未支持的功能/模型尚在开发中，会尽快提供。ESP-BLE-MESH 已通过 Bluetooth SIG 蓝牙技术联盟的 认证。

ESP-BLE-MESH 架构主要由以下 5 大部分组成:

- `Mesh` 协议栈
  - `Mesh Networking` 负责 BLE Mesh 设备的网络消息处理等。
  - `Mesh Provisioning` 负责 BLE Mesh 设备的启动配置流程。
  - `Mesh Models` 负责实现 SIG 定义的模型。
- 网络管理
  - 负责实现网络管理程序，包括节点删除程序、网络索引 (IV Index) 恢复程序等。
- 特性
  - 包括 BLE Mesh 特性，如低功耗特性、好友特性、中继特性等。
- `Mesh` 承载层
  - 包括 广播承载层和 `GATT` 承载层。承载层对于 ESP-BLE-MESH 协议栈至关重要，因为协议栈基于蓝牙低功耗技术构建而成，其必须利用承载层通过 BLE 广播通道和连接通道进行数据传输。
- 应用程序
  - 基于 ESP-BLE-MESH 协议栈和 `Mesh Models`。
  - 通过调用 API 和处理事件，`Applications` 实现了与 ESP-BLE-MESH 协议栈中的 `Mesh Networking` 和 `Mesh Provisioning` 的交互，也实现了与 `Mesh Models` 中一系列模型的交互。

1.1 Mesh 协议栈

1.1.1 Mesh Networking

协议栈架构中的 `Mesh Networking` 实现了如下功能:

- Mesh 网络中节点间的通讯。
- Mesh 网络中消息的加解密。
- Mesh 网络资源的管理，如网络秘钥 (NetKey)、网络索引等。
- Mesh 网络消息的分包与重组。
- 消息在不同模型间的模型映射。

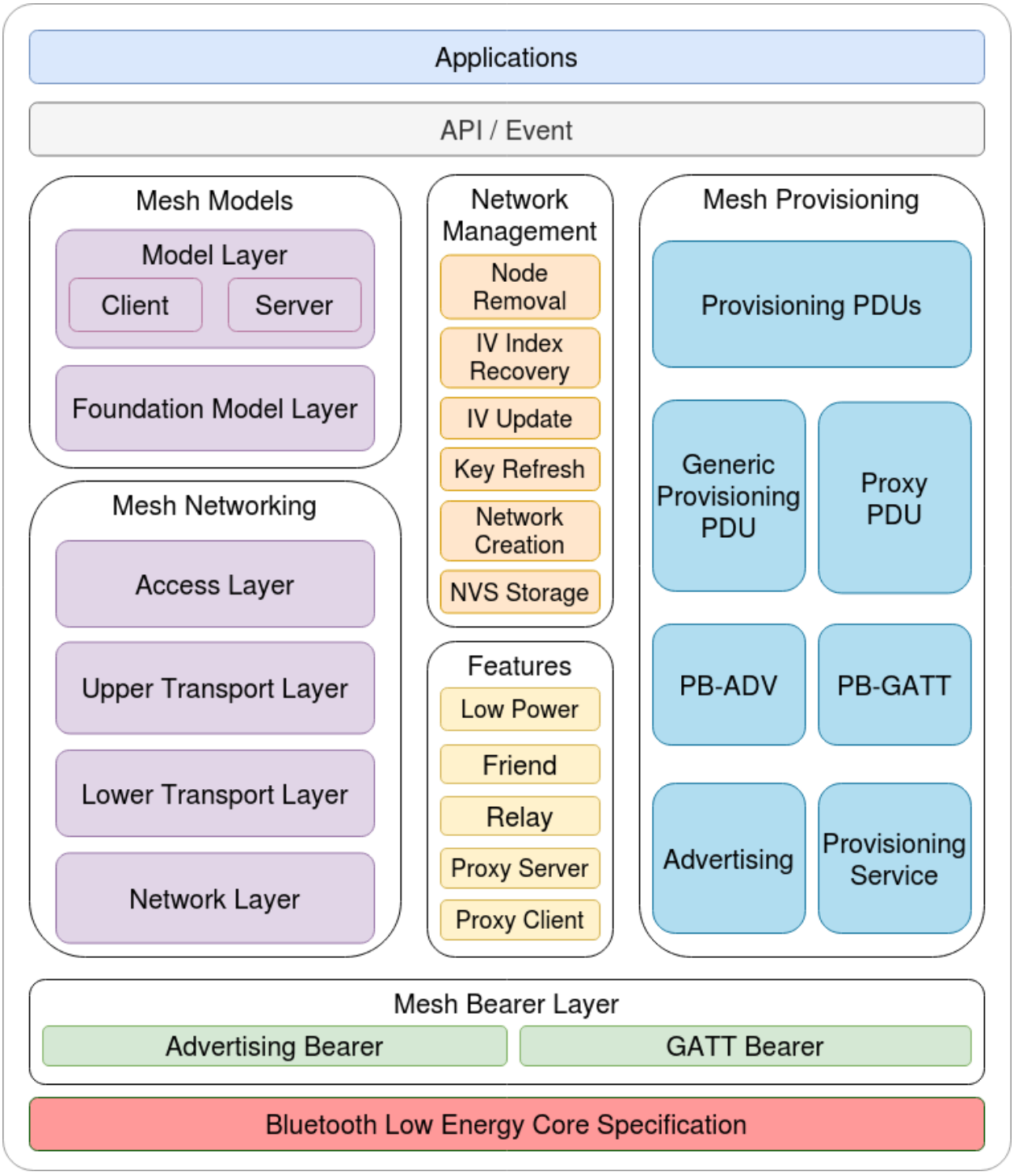


图 53: 图 1.1 ESP-BLE-MESH 架构图

- 更多功能，请参见*ESP-BLE-MESH Feature List*。

`Mesh Networking` 功能的实现是基于层级结构的。每一层的功能如表 1.1 所示：

表 1: 表 1.1 Mesh Networking 框架描述

层级	功能
接入层	接入层定义应用程序数据的格式，还对上层传输层对数据包的加密和解密进行定义和控制。
上层传输层	上层传输层对接入层进出的应用数据进行加密、解密和认证，同时也处理被称为“传输控制消息”的特殊消息，这种消息包括了与“友谊”和心跳包相关的消息。
底层传输层	底层传输层处理 PDU 的分包和重组。
网络层	网络层定义网络消息的地址类型和格式，实现设备的中继功能。

1.1.2 Mesh Provisioning

协议栈架构中的 `Mesh Provisioning` 实现了如下功能：

- 对未配网设备的配网。
- Mesh 网络资源的分配 (单播地址、网络索引和网络秘钥)。
- 配网期间对 4 种验证方法的支持。
- 更多功能，请参见*ESP-BLE-MESH Feature List*。

`Mesh Provisioning` 功能的实现是基于层级结构的。每一层的功能如表 1.2 所示：

表 2: 表 1.2 Mesh Provisioning 框架描述

层级	功能
Provisioning PDUs	通过配网协议处理不同层级的 Provisioning PDUs。
Generic Provisioning PDU/Proxy PDU	使用 Generic Provisioning 层或代理协议层将 Provisioning PDU 传输到未配网的设备。
PB-ADV/PB-GATT	这些层级定义了 Provisioning PDUs 作为可分包和重组的消息进行传输的方式。
Advertising/Provisioning Service	Provisioning bearer 定义了会话建立的方式，该方式用来将 Generic Provisioning 层的传输包传送到设备。

1.1.3 Mesh Models

协议栈架构中的 `Mesh Models` 实现了如下功能：

- Configuration Client/Server Models
- Health Client/Server Models

- Generic Client/Server Models
- Sensor Client/Server Models
- Time and Scenes Client/Server Models
- Lighting Client/Server Models

每一层的功能如表 1.3 所示：

表 3: 表 1.3 Mesh Models 框架描述

层级	功能
模型层	模型层实现用于标准化典型用户场景操作的模型，包括 Generic Client/Server Models、Sensor Client/Server Models、Time and Scenes Client/Server Models、Lighting Client/Server Models 和若干自定义模型。
基础模型层	基础模型层实现与 ESP-BLE-MESH 网络配置、管理和自我诊断等相关的模型。

1.2 Mesh 网络管理

网络管理实现了如下功能：

- 节点移除程序：用于将节点从网络中移除。
- 网络索引恢复程序：用于恢复节点的网络索引。
- 网络索引更新程序：用于更新节点的网络索引。
- 秘钥更新程序：用于更新节点的网络秘钥、应用秘钥 (AppKey) 等。
- 网络创建程序：用于创建 mesh 网络。
- NVS 存储器：用于存储节点的网络信息。

1.3 Mesh 特性

特性包括以下几项：

- 低功耗特性：用于降低节点的能耗。
- 好友特性：用于为低功耗节点存储消息。
- 中继特性：用于中继/转发节点通过广播承载层收到的网络 PDU.
- Proxy Server/Client 是代理协议中的两个节点角色，其使节点可以通过面向连接的承载层收发 Network PDUs、mesh beacons、代理配置消息和 Provisioning PDU。

1.4 Mesh 承载层

协议栈框架中的 承载层负责 ESP-BLE-MESH 协议栈和低功耗蓝牙核心协议间的数据传输。

承载层可视为是基于蓝牙低功耗核心协议的载体层，其实现了 ESP-BLE-MESH 协议栈数据的接收和传输。

表 4: 表 1.3 Mesh 承载层描述

层级	功能
GATT 承载层	GATT 承载层使用代理协议通过 GATT 连接在两个设备之间发送和接收 `Proxy PDUs`。
广播承载层	使用广播承载层时，必须使用低功耗蓝牙广播通道来发送 mesh 数据包, 数据包中的 AD Type 需要设置为 mesh 数据包的类型。

1.5 Mesh 应用层

协议栈框架图中的 应用层通过调用 ESP-BLE-MESH 协议栈提供的 API 并处理协议栈上报的事件来实现相应的功能，有一些常见应用，比如网关、照明等。

应用层和 `API / 事件`之间的交互

- 应用层调用 API
  - 调用配网相关的 API 进行配网。
  - 调用模型相关的 API 发送消息。
  - 调用设备属性相关的 API 获取设备的本地信息。
- 应用层处理事件

  应用层的设计基于事件设计，事件将参数传输给应用层。事件主要分为两大类。
  - **调用 API 完成的事件**。
    - 比如接收消息的节点。
  - **协议栈主动上报给应用层的事件**。
    - 协议栈主动上报的事件。
    - 模型主动上报的事件。
- 事件通过应用层注册的回调函数进行上报，同时回调函数中也会包含对事件的相应处理。

`API /事件`与 ESP-BLE-MESH 协议栈的交互

- 用户使用的 API 主要调用 ‘‘Mesh Networking‘‘、`Mesh Provisioning` 和 `Mesh Models` 提供的函数。
- `API /事件`和协议栈的交互不会跨越协议栈的层级进行操作。比如 API 不会调用 `Network Layer` 相关的函数。

2. ESP-BLE-MESH 架构实现

ESP-BLE-MESH 架构的设计和实现是基于层级和模块的。具体而言，第 2.1 节（Mesh 网络的实现），第 2.2 节（Mesh 配网实现）和第 2.3 节（Mesh 层级实现）基于层级思想，第 2.4 节（网格模型的实现）基于模块思想。

- **层级思想**: 基于层级思想，网络架构根据 Mesh Profile Specification 中指定的层级设计而成。每层都有独特的文件，文件包括该层的 API 等。具体设计如图 2.1 所示。
- **模块思想**: 每个文件实现一个独立的功能，供其它程序调用。

ESP-BLE-MESH 架构采用分层的方式进行设计，数据包的处理所经过的层级顺序是固定的，也就是数据包的处理过程会形成一个 消息流。因此，我们可以从图 2.1 的协议栈接口图中看到消息流。

2.1 Mesh 协议栈的实现

2.1.1 Mesh Networking 实现

`Mesh Networking` 中的文件列表和每个文件实现的功能如表 2.1 所示：

表 5: 表 2.1 Mesh Networking 文件描述

文件	功能
access.c	BLE Mesh 接入层
transport.c	BLE Mesh 底层/上层传输层
net.c	BLE Mesh 网络层
adv.c	用于发送 BLE Mesh 广播包的任务，一个用于处理收到的广播包的回调以及用于分配 adv 缓冲区的 API

2.1.2 Mesh Provisioning 实现

由于 Node/Provisioner 共存的限制，Mesh Provisioning 的实现分为两大模块。

实现 Node 启动配置的特定文件如表 2.2 所示：

表 6: 表 2.2 Mesh Provisioning（节点）文件描述

文件	功能
prov.c	BLE Mesh 节点配网 (PB-ADV & PB-GATT)
proxy_server.c	BLE Mesh 节点代理服务器相关功能
beacon.c	用于处理 BLE Mesh Beacon 的 API

实现 Provisioner 配置功能的特定文件如表 2.3 所示：

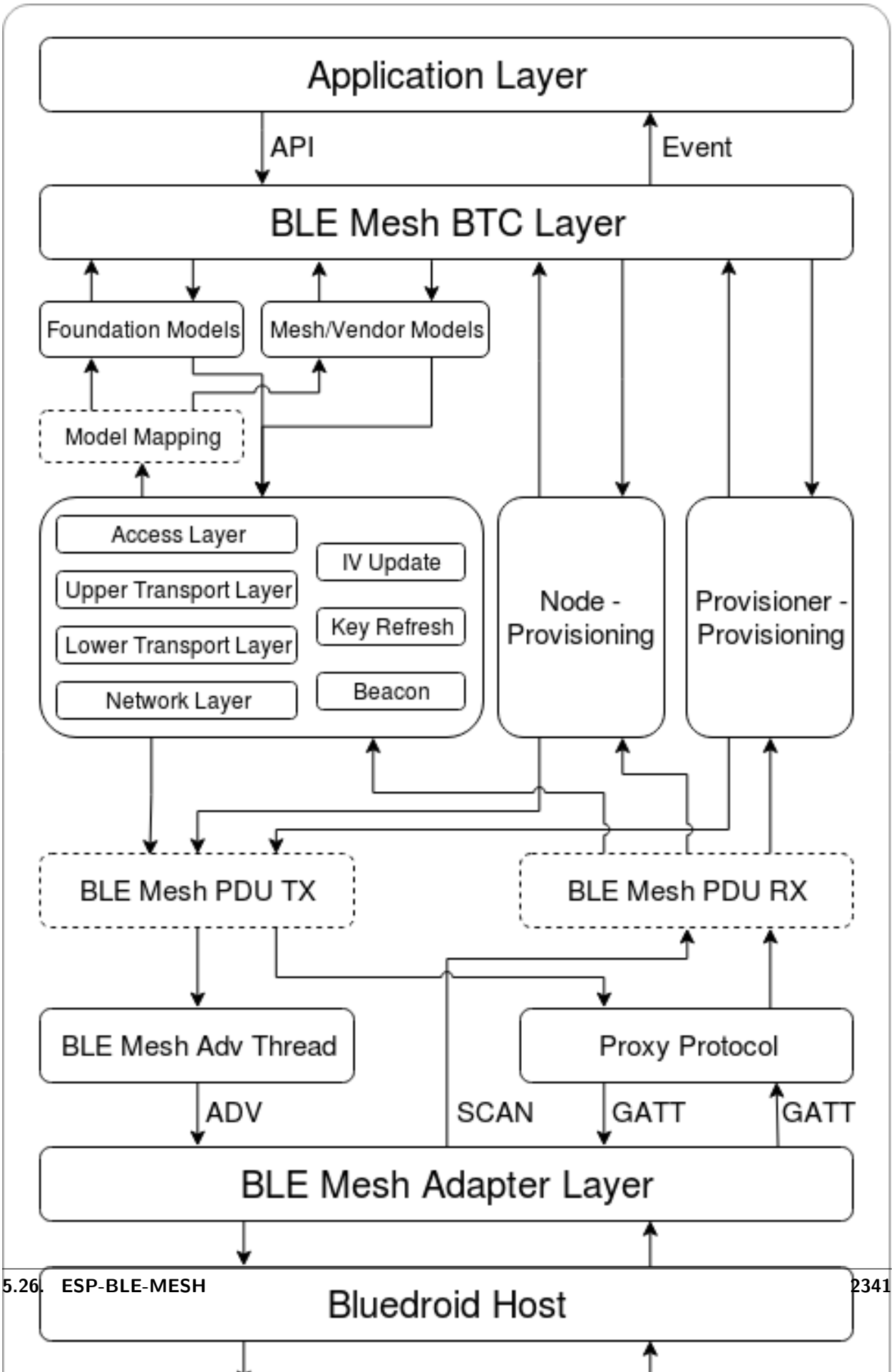

Application Layer
API
Event
BLE Mesh BTC Layer
Foundation Models
Mesh/Vendor Models
Model Mapping
Access Layer
Upper Transport Layer
Lower Transport Layer
Network Layer
IV Update
Key Refresh
Beacon
Node - Provisioning
Provisioner - Provisioning
BLE Mesh PDU TX
BLE Mesh PDU RX
BLE Mesh Adv Thread
Proxy Protocol
ADV
SCAN
GATT
GATT
BLE Mesh Adapter Layer
Bluedroid Host

表 7: 表 2.3 Mesh Provisioning (Provisioner) 文件描述

文件	功能
provisioner_prov.c	BLE Mesh Provisioner 配置入网 (PB-ADV & PB-GATT)
proxy_client.c	BLE Mesh 代理客户端相关功能
provisioner_main.c	BLE Mesh Provisioner 网络相关功能

2.1.3 Mesh Models 实现

Mesh Models 用于实现节点中所包含的模型的具体功能。服务器模型主要用于维护节点状态。客户端模型主要用于获取和修改节点状态。

表 8: Table 2.4 Mesh Models 文件描述

文件	功能
cfg_cli.c	发送 Configuration Client 消息，接收相应应答消息
cfg_srv.c	接收 Configuration Client 消息，发送适当应答消息
health_cli.c	发送 Health Client 消息，接收相应应答消息
health_srv.c	接收 Health Client 消息，发送适当应答消息
client_common.c	BLE Mesh 模型相关操作
generic_client.c	发送 BLE Mesh Generic Client 消息，接收相应应答消息
lighting_client.c	发送 BLE Mesh Lighting Client 消息，接收相应应答消息
sensor_client.c	发送 BLE Mesh Sensor Client 消息，接收相应应答消息
time_scene_client.c	发送 BLE Mesh Time Scene Client 消息，接收相应应答消息

2.2 Mesh Bearers 实现

Mesh Bearers 在实现时充分考虑了可移植性。当 ESP-BLE-MESH 协议栈需要移植到其它平台时，用户只需要修改 mesh_bearer_adapt.c 就能移植成功。

表 9: 表 2.5 Mesh Bearers 文件描述

文件	功能
mesh_bearer_adapt.c	BLE Mesh 承载层适配文件。此文件提供用于接收和发送 BLE Mesh ADV 和 GATT 相关数据包的接口。

注解: mesh_bearer_adapt.c 是对 Mesh 网络框架中 `Advertising Bearer` 和 `GATT Bearer` 的实现。

2.3 Mesh Applications 实现

我们提供了一系列用于客户开发的应用示例，用户可以基于*ESP-BLE-MESH 示例* 开发产品。

3. ESP-BLE-MESH 辅助程序

辅助程序指的是 ESP-BLE-MESH 协议栈中可选的功能。辅助程序的设计通常通过*CONFIG_BLE_MESH* 来实现代码的裁剪。

3.1 特性

- 低功耗
- 好友
- 中继
- 代理客户端/代理服务器

3.2 网络管理

- 节点移除程序
- 网络索引恢复程序
- 网络索引更新程序
- 秘钥更新程序
- 网络创建程序
- NVS 存储器

3.3 辅助程序实现

采用独立模块的设计主要考虑到两个因素：

- 该模块不具备分层实现的条件，其实现可以完全独立，不需要依赖其它模块。
- 模块中的函数会被反复使用到，因此最好设计成独立模块。独立模块如表 3.1 所示：

表 10: 表 3.1 模块文件描述

文件	功能
lpn.c	BLE Mesh 低功耗功能
friend.c	BLE Mesh 好友功能
net.c	BLE Mesh 中继功能、网络创建、网络索引更新程序、网络索引恢复程序、秘钥更新程序相关功能
proxy_server.c	BLE Mesh 代理服务器相关功能
proxy_client.c	BLE Mesh 代理客户端相关功能
settings.c	BLE Mesh NVS 存储器功能
main.c	BLE Mesh 协议栈初始化，协议栈使能，节点移除相关功能

ESP-BLE-MESH 常见问题手册

[English]

本文汇总了 ESP-BLE-MESH 协议栈开发的常见问题及解答，全文分为 7 个章节。

- *1. Provisioner 开发*
- *2. 节点开发*
- *3. ESP-BLE-MESH 和 Wi-Fi 共存*
- *4. 快速配网*
- *5. Log 帮助*
- *6. 示例帮助*
- *7. 其他*

用户可以参考这些章节，快速找到问题的答案。该文档会根据各种渠道收集的反馈进行更新。

1. Provisioner 开发

通常而言，Provisioner 用于配网未配网设备并形成 mesh 网络。组网后，设备的角色变成节点。

1.1 未配网设备加入 ESP-BLE-MESH 网络的流程是什么？

设备通过 Provisioner 加入 ESP-BLE-MESH 网络分为两个阶段，配网阶段和配置阶段。

- 配网阶段：为设备分配单播地址、添加网络密钥 (NetKey) 等。通过配网，设备加入 ESP-BLE-MESH 网络，身份从未配网设备变为节点。

- 配置阶段：为节点添加应用密钥 (AppKey)，并将应用密钥绑定到相应模型。配置期间，有些选项是可选的，比如为节点添加订阅地址、设置发布地址等。通过配置，该节点实际上可以向 Provisioner 发送消息，也可以接收来自 Provisioner 的消息。

1.2 如果 Provisioner 想要改变节点状态，其需满足什么条件？

- 需要有和节点的服务器模型相对应的客户端模型。
- 需要和节点有相同的、可用于加密消息的网络密钥和应用密钥。
- 需要知道节点的地址，可以是单播地址，也可以是订阅地址。

1.3 如何使用网络密钥和应用密钥？

- 网络密钥用于加密网络层的消息。具有相同网络密钥的节点视作在同一网络中，具有不同网络密钥的节点相互之间不能进行通信。
- 应用密钥用于加密上层传输层中的消息。如果服务器模型和客户端模型绑定的应用密钥不同，则无法实现相互通信。

1.4 如何生成网络密钥或应用密钥？是否可以采用固定的网络密钥或应用密钥？

- API *esp_ble_mesh_provisioner_add_local_net_key()* 可以用来添加包含固定值或随机值的网络密钥。
- API *esp_ble_mesh_provisioner_add_local_app_key()* 可以用来添加包含固定值或随机值的应用密钥。

1.5 Provisioner 的单播地址是不是固定的？

esp_ble_mesh_prov_t 中 `prov_unicast_addr` 的值用于设置 Provisioner 的单播地址，只能在初始化期间设置一次，此后不能更改。

1.6 Provisioner 的地址是否可以作为节点上报状态消息的目的地址？

Provisioner 的单播地址只能在初始化期间设置一次，此后不能更改。理论而言，只要节点知道 Provisioner 的单播地址，此地址便可用作节点上报状态消息的目的地址。节点在网络配置的过程中可以知道 Provisioner 的单播地址，因为 Provisioner 往节点发送消息时，消息的源地址就是 Provisioner 的单播地址。

订阅地址也可使用。Provisioner 订阅组地址或者虚拟地址，节点向该订阅地址发送消息。

1.7 被 Provisioner 配网到 ESP-BLE-MESH 网络中的第一个节点的单播地址是不是固定的？

esp_ble_mesh_prov_t 中 prov_start_address 的值用于设置 Provisioner 配网未配网设备的起始地址，即其首先配网的节点的单播地址。单播地址只能在初始化期间设置一次，此后不能修改。

1.8 手机 App 首先配置的节点的单播地址是不是固定的？

该 App 将确定单播地址，目前大多数单播地址是固定的。

1.9 如何知道当前 Provisioner 正在配网哪个未配网设备？

esp_ble_mesh_prov_t 中 prov_attention 的值由 Provisioner 在配网过程中设置给未配网设备。该值只能在初始化期间设置一次，此后不能修改。未配网设备加入 mesh 网络后可以用特定的方式来显示自己正在配网，比如灯光闪烁，以告知 Provisioner 其正在配网。

1.10 配网过程中，认证设备共有多少种方法？提供的范例中 provided examples 使用了什么方法？

共有四种设备认证方法，即 No OOB、Static OOB、Output OOB 和 Input OOB。提供的范例使用了 No OOB 的方式。

1.11 配置入网前，未配网设备的广播包可以携带哪些信息？

- Device UUID
- OOB Info
- URL Hash (可选的)

1.12 这些信息可以用于设备识别吗？

是的。每个设备都有独一无二的 Device UUID, 用户可以通过 Device UUID 识别设备。

1.13 当 Provisioner 配网的节点包含多个元素时，单播地址是如何分配的？

- Provisioner 会给设备的主元素分配一个单播地址，其余元素的单播地址在此基础上递增。
- 比如：如果一个未配网设备有三个元素，即主要元素、第二元素和第三元素。配网完成后，节点主元素的单播地址为 0x0002，节点第二元素的单播地址为 0x0003，节点第三元素的单播地址为 0x0004。

1.14 Provisioner 如何通过 Configuration Client Model 获取并且解析节点的构成数据？

- Provisioner 可以调用 *Configuration Client Model* API *esp_ble_mesh_config_client_set_state()* 设置参数，调用 *esp_ble_mesh_cfg_client_get_state_t* 中的 comp_data_get 获取节点的构成数据。
- 用户可以参考以下代码解析 Composition Data:

```
#include <stdio.h>
#include <string.h>
#include <stdint.h>

//test date: 0C001A0001000800030000010501000000800100001003103F002A00
//0C00 1A00 0100 0800 0300 0001 05 01 0000 0080 0100 0010 0310 3F002A00

// CID is 0x000C
// PID is 0x001A
// VID is 0x0001
// CRPL is 0x0008
// Features is 0x0003 - Relay and Friend features.
// Loc is "front" - 0x0100
// NumS is 5
// NumV is 1
// The Bluetooth SIG Models supported are: 0x0000, 0x8000, 0x0001, 0x1000,␣
↪0x1003
// The Vendor Models supported are: Company Identifier 0x003F and Model␣
↪Identifier 0x002A

typedef struct {
    int16_t cid;
    int16_t pid;
    int16_t vid;
    int16_t crpl;
    int16_t features;
    int16_t all_models;
    uint8_t sig_models;
    uint8_t vnd_models;
} esp_ble_mesh_composition_head;

typedef struct {
    uint16_t model_id;
```

(下页继续)

(续上页)

```
    uint16_t vendor_id;
} tsModel;

typedef struct {
    // reserve space for up to 20 SIG models
    uint16_t SIG_models[20];
    uint8_t numSIGModels;

    // reserve space for up to 4 vendor models
    tsModel Vendor_models[4];
    uint8_t numVendorModels;
} esp_ble_mesh_composition_decode;

int decode_comp_data(esp_ble_mesh_composition_head *head, esp_ble_mesh_
↪composition_decode *data, uint8_t *mystr, int size)
{
    int pos_sig_base;
    int pos_vnd_base;
    int i;

    memcpy(head, mystr, sizeof(*head));

    if(size < sizeof(*head) + head->sig_models * 2 + head->vnd_models * 4) {
        return -1;
    }

    pos_sig_base = sizeof(*head) - 1;

    for(i = 1; i < head->sig_models * 2; i = i + 2) {
        data->SIG_models[i/2] = mystr[i + pos_sig_base] | (mystr[i + pos_sig_
↪base + 1] << 8);
        printf("%d: %4.4x\n", i/2, data->SIG_models[i/2]);
    }

    pos_vnd_base = head->sig_models * 2 + pos_sig_base;

    for(i = 1; i < head->vnd_models * 2; i = i + 2) {
        data->Vendor_models[i/2].model_id = mystr[i + pos_vnd_base] | (mystr[i␣
↪+ pos_vnd_base + 1] << 8);
        printf("%d: %4.4x\n", i/2, data->Vendor_models[i/2].model_id);
```

(下页继续)

(续上页)

```
        data->Vendor_models[i/2].vendor_id = mystr[i + pos_vnd_base + 2] |␣
↪(mystr[i + pos_vnd_base + 3] << 8);
        printf("%d: %4.4x\n", i/2, data->Vendor_models[i/2].vendor_id);
    }

    return 0;
}

void app_main(void)
{
    esp_ble_mesh_composition_head head = {0};
    esp_ble_mesh_composition_decode data = {0};
    uint8_t mystr[] = { 0x0C, 0x00, 0x1A, 0x00,
                        0x01, 0x00, 0x08, 0x00,
                        0x03, 0x00, 0x00, 0x01,
                        0x05, 0x01, 0x00, 0x00,
                        0x00, 0x80, 0x01, 0x00,
                        0x00, 0x10, 0x03, 0x10,
                        0x3F, 0x00, 0x2A, 0x00};
    int ret;

    ret = decode_comp_data(&head, &data, mystr, sizeof(mystr));
    if (ret == -1) {
        printf("decode_comp_data error");
    }
}
```

1.15 Provisioner 如何通过获取的 Composition Data 进一步配置节点？

Provisioner 通过调用 *Configuration Client Model* API *esp_ble_mesh_config_client_set_state()* 来进行如下配置。

- 正确设置参数 *esp_ble_mesh_cfg_client_set_state_t* 中的 app_key_add，将应用密钥添加到节点中。
- 正确设置参数 *esp_ble_mesh_cfg_client_set_state_t* 中的 model_sub_add，将订阅地址添加到节点的模型中。
- 正确设置参数 *esp_ble_mesh_cfg_client_set_state_t* 中的 model_pub_set，将发布地址添加到节点的模型中。

1.16 节点可以自己添加相应的配置吗？

本法可用于特殊情况，如测试阶段。

- 此示例展示了节点如何为自己的模型添加新的组地址。

```
esp_err_t example_add_fast_prov_group_address(uint16_t model_id, uint16_t␣
↪group_addr)
{
    const esp_ble_mesh_comp_t *comp = NULL;
    esp_ble_mesh_elem_t *element = NULL;
    esp_ble_mesh_model_t *model = NULL;
    int i, j;

    if (!ESP_BLE_MESH_ADDR_IS_GROUP(group_addr)) {
        return ESP_ERR_INVALID_ARG;
    }

    comp = esp_ble_mesh_get_composition_data();
    if (!comp) {
        return ESP_FAIL;
    }

    for (i = 0; i < comp->element_count; i++) {
        element = &comp->elements[i];
        model = esp_ble_mesh_find_sig_model(element, model_id);
        if (!model) {
            continue;
        }
        for (j = 0; j < ARRAY_SIZE(model->groups); j++) {
            if (model->groups[j] == group_addr) {
                break;
            }
        }
        if (j != ARRAY_SIZE(model->groups)) {
            ESP_LOGW(TAG, "%s: Group address already exists, element index: %d
↪", __func__, i);
            continue;
        }
        for (j = 0; j < ARRAY_SIZE(model->groups); j++) {
            if (model->groups[j] == ESP_BLE_MESH_ADDR_UNASSIGNED) {
                model->groups[j] = group_addr;
```

(下页继续)

(续上页)

```
                break;
            }
        }
        if (j == ARRAY_SIZE(model->groups)) {
            ESP_LOGE(TAG, "%s: Model is full of group addresses, element␣
↪index: %d", __func__, i);
        }
    }

    return ESP_OK;
}
```

注解： 使能了节点的 NVS 存储器后，通过该方式添加的组地址以及绑定的应用密钥在设备掉电的情况下不能保存。这些配置信息只有通过 Configuration Client Model 配置时才会保存。

1.17 Provisioner 如何通过分组的方式控制节点？

通常而言，在 ESP-BLE-MESH 网络中实现组控制有两种方法，即组地址方法和虚拟地址方法。假设有 10 个设备，即 5 个带蓝灯的设备和 5 个带红灯的设备。

- 方案一：5 个蓝灯设备订阅一个组地址，5 个红灯设备订阅另一个组地址。Provisioner 往不同的组地址发送消息，即可实现分组控制设备。
- 方案二：5 个蓝灯设备订阅一个虚拟地址，5 个红灯设备订阅另一个虚拟地址，Provisioner 往不同的虚拟地址发送消息，即可实现分组控制设备。

1.18 Provisioner 如何将节点添加至多个子网？

节点配置期间，Provisioner 可以为节点添加多个网络密钥，拥有相同网络密钥的节点属于同一子网。Provisioner 可以通过不同的网络密钥与不同子网内的节点进行通信。

1.19 Provisioner 如何知道网络中的某个设备是否离线？

节点离线通常定义为：电源故障或其他原因导致的节点无法与 mesh 网络中的其他节点正常通信的情况。

ESP-BLE-MESH 网络中的节点间彼此不连接，它们通过广播通道进行通信。

此示例展示了如何通过 Provisioner 检测节点是否离线。

- 节点定期给 Provisioner 发送心跳包。如果 Provisioner 超过一定的时间未接收到心跳包，则视该节点离线。

注解： 心跳包的设计应该采用单包（字节数小于 11 个字节）的方式，这样收发效率会更高。

1.20 Provisioner 删除网络中的节点时，需要进行哪些操作？

通常而言，Provisioner 从网络中移除节点主要涉及三个步骤：

- 首先，Provisioner 将需要移除的节点添加至“黑名单”。
- 其次，Provisioner 启动密钥更新程序。
- 最后，节点执行节点重置程序，切换自身身份为未配网设备。

1.21 在密钥更新的过程中，Provisioner 如何更新节点的网络密钥？

- 通过正确设置参数 *esp_ble_mesh_cfg_client_set_state_t* 中的 `net_key_update`，使用 *Configuration Client Model* API *esp_ble_mesh_config_client_set_state()*，Provisioner 更新节点的网络密钥。
- 通过正确设置参数 *esp_ble_mesh_cfg_client_set_state_t* 中的 `app_key_update`，使用 *Configuration Client Model* API *esp_ble_mesh_config_client_set_state()*，Provisioner 更新节点的应用密钥。

1.22 Provisioner 如何管理 mesh 网络中的节点？

ESP-BLE-MESH 在示例中实现了一些基本的节点管理功能，比如 `esp_ble_mesh_store_node_info()`。ESP-BLE-MESH 还提供可用于设置节点本地名称的 API *esp_ble_mesh_provisioner_set_node_name()* 和可用于获取节点本地名称的 API *esp_ble_mesh_provisioner_get_node_name()*。

1.23 Provisioner 想要控制节点的服务器模型时需要什么？

Provisioner 在控制节点的服务器模型前，必须包括相应的客户端模型。

Provisioner 应当添加本地的网络密钥和应用密钥。

- Provisioner 调用 API *esp_ble_mesh_provisioner_add_local_net_key()* 以添加网络密钥。
- Provisioner 调用 API *esp_ble_mesh_provisioner_add_local_app_key()* 以添加应用密钥。

Provisioner 应当配置自己的客户端模型。

- Provisioner 调用 API *esp_ble_mesh_provisioner_bind_app_key_to_local_model()* 以绑定应用密钥至自己的客户端模型。

1.24 Provisoner 如何控制节点的服务器模型？

ESP-BLE-MESH 支持所有 SIG 定义的客户端模型。Provisioner 可以使用这些客户端模型控制节点的服务器模型。客户端模型分为 6 类，每类有相应的功能。

- Configuration Client Model
  - API *esp_ble_mesh_config_client_get_state()* 可用于获取 Configuration Server Model 的*esp_ble_mesh_cfg_client_get_state_t* 值。
  - API *esp_ble_mesh_config_client_set_state()* 可用于获取 Configuration Server Model 的*esp_ble_mesh_cfg_client_set_state_t* 值。
- Health Client Model
  - API *esp_ble_mesh_health_client_get_state()* 可用于获取 Health Server Model 的*esp_ble_mesh_health_client_get_state_t* 值。
  - API *esp_ble_mesh_health_client_set_state()* 可用于获取 Health Server Model 的*esp_ble_mesh_health_client_set_state_t* 值。
- Generic Client Models
  - API *esp_ble_mesh_generic_client_get_state()* 可用于获取 Generic Server Model 的*esp_ble_mesh_generic_client_get_state_t* 值。
  - API *esp_ble_mesh_generic_client_set_state()* 可用于获取 Generic Server Model 的*esp_ble_mesh_generic_client_set_state_t* 值。
- Lighting Client Models
  - API *esp_ble_mesh_light_client_get_state()* 可用于获取 Lighting Server Model 的*esp_ble_mesh_light_client_get_state_t* 值。
  - API *esp_ble_mesh_light_client_set_state()* 可用于获取 Lighting Server Model 的*esp_ble_mesh_light_client_set_state_t* 值。
- Sensor Client Models
  - API *esp_ble_mesh_sensor_client_get_state()* 可用于获取 Sensor Server Model 的*esp_ble_mesh_sensor_client_get_state_t* 值。
  - API *esp_ble_mesh_sensor_client_set_state()* 可用于获取 Sensor Server Model 的*esp_ble_mesh_sensor_client_set_state_t* 值。
- **Time and Scenes Client Models**

  - API *esp_ble_mesh_time_scene_client_get_state()* 可用于获取 Time and Scenes Server Model 的*esp_ble_mesh_time_scene_client_get_state_t* 值。
  - API *esp_ble_mesh_time_scene_client_set_state()* 可用于获取 Time and Scenes Server Model 的*esp_ble_mesh_time_scene_client_set_state_t* 值。

2. 节点开发

2.1 节点包含什么样的模型？

- ESP-BLE-MESH 中，节点由一系列的模型组成，每个模型实现节点的某些功能。
- 模型分为两种，客户端模型和服务器模型。客户端模型可以获取并设置服务器模型的状态。
- 模型也可以分为 SIG 模型和自定义模型。SIG 模型的所有行为都由官方定义，而自定义模型的行为均由用户定义。

2.2 每个模型对应的消息格式是不是固定的？

- 消息由 opcode 和 payload 组成，通过 opcode 进行区分。
- 与模型对应的消息的类型和格式都是固定的，这意味着模型之间传输的消息是固定的。

2.3 节点的模型可以使用哪些函数发送消息？

- 对于客户端模型，用户可以调用 API *esp_ble_mesh_client_model_send_msg()* 发送消息。
- 对于服务器模型，用户可以调用 API *esp_ble_mesh_server_model_send_msg()* 发送消息。
- 对于发布，用户可以调用 API *esp_ble_mesh_model_publish()* 发布消息。

2.4 如何实现消息传输不丢包？

如果用户要实现消息传输不丢包，则需有应答的消息。等待应答的默认时间在*CONFIG_BLE_MESH_CLIENT_MSG_TIMEOUT* 中设置。如果发送端等待应答超时，就会触发对应的超时事件。

注解： API *esp_ble_mesh_client_model_send_msg()* 中可以设置应答的超时时间。如果参数 `msg_timeout` 设为 **0**，那么超时时间便会采用默认值（4 秒）。

2.5 如何发送无应答的消息？

对于客户端模型，用户可以调用 API *esp_ble_mesh_client_model_send_msg()* with the parameter `need_rsp` set to `false` 发送无应答消息。

对于服务器模型，调用 API *esp_ble_mesh_server_model_send_msg()* 发送的消息总是无应答的消息。

2.6 如何为模型添加订阅地址？

通过 Configuration Client Model 添加订阅地址。

2.7 模型发送的消息和发布的消息有何不同？

调用 API *esp_ble_mesh_client_model_send_msg()* 或 *esp_ble_mesh_server_model_send_msg()* 发送的消息会在 Network Transmit 状态规定的期限内发送。

调用 API *esp_ble_mesh_model_publish()* 发布的消息将由模型发布状态决定是否发布。消息的发布一般是周期性的，或者有固定次数。发布周期和发布次数由模型发布状态控制，并且可以通过 Configuration Client Model 进行配置。

2.8 发送不分包消息时，最多可携带多少有效字节？

不分包消息的总有效载荷长度（可由用户设置）为 11 个八位位组，因此，如果消息的 opcode 为 2 个八位位组，则该消息可以携带 9 个八位位组的有效信息。对于 vendor 消息，由于 opcode 是 3 个八位位组，剩余的有效负载长度为 8 个八位位组。

2.9 什么时候应该使能节点的 Relay 功能？

如果 mesh 网络中检测到的节点很稀疏，用户可以使能节点的 Relay 功能。

如果 mesh 网络中检测到的节点很密集，用户可以选择仅使能一些节点的 Relay 功能。

如果 mesh 网络大小未知，用户可以默认使能 Relay 功能。

2.10 什么时候应该使能节点的 Proxy 功能？

如果未配网设备将由电话配网，则未配网设备应该使能 Proxy 功能，因为当前几乎所有电话都不支持通过广播承载层发送 ESP-BLE-MESH 数据包。并且，未配网设备成功配网成为 Proxy 节点后，其会通过 GATT 承载层和广播承载层与 mesh 网络中的其他节点通信。

2.11 如何使用代理过滤器?

代理过滤器用于减少 Proxy Client（如手机）和 Proxy Server（如节点）之间交换的 Network PDU 的数量。另外，通过代理过滤器，Proxy Client 可以明确请求仅接收来自 Proxy Server 的某些目标地址的 mesh 消息。

2.12 Relay 节点什么时候可以中继消息？

如果要中继消息，消息需满足以下要求。

- 消息存在于 mesh 网络中。
- 消息的目的地址不是节点的单播地址。
- 消息的 TTL 值需大于 1。

2.13 如果一条消息分成几段，那么其他 Relay 节点是接收到一段消息就中继还是等接收到完整的数据包才中继？

Relay 节点收到其中一段消息时就中继，而非一直等到接收所有的消息。

2.14 使用 Low Power 功能降低功耗的原理是什么？

- 开启无线电进行收听时，设备消耗能量。使能节点的低功耗功能后，它将在大多数时间内关闭无线电功能。
- 低功耗节点和好友节点需要合作，因此低功耗节点可以以适当或较低的频率接收消息，而无需一直收听。
- 当低功耗节点有一些新消息时，好友节点将为其存储消息。低功耗节点可以间隔固定时间轮询好友节点，以查看是否有新的消息。

2.15 设备断电后上电，如何能继续在网络中进行通讯？

在 *menuconfig* 中启用配置 `Store BLE Mesh Node configuration persistently` 。

2.16 如何实现将节点自检的信息发送出来？

推荐节点通过 Health Server Model 定期发布其自检结果。

2.17 节点间如何传输消息？

节点间传输信息的可能应用场景是，一旦烟雾警报检测到高浓度的烟雾，就会触发喷淋设备。有两种实现方法。

- 方法 1：喷淋设备订阅组地址。当烟雾警报器检测到高浓度的烟雾时，它会发布一条消息，该消息的目标地址是喷淋设备已订阅的组地址。
- 方法 2：Provisioner 可以配置喷淋设备的单播地址为烟雾报警器的地址。当检测到高浓度的烟雾时，烟雾警报器以喷淋设备的单播地址为目标地址，将消息发送到喷淋设备。

2.18 设备通信必须要网关吗？

- 情况 1：节点仅在 mesh 网络内通信。这种情况下，不需要网关。ESP-BLE-MESH 网络是一个泛洪的网络，网络中的消息没有固定的路径，节点与节点之间可以随意通信.
- 情况 2：如果用户想要远程控制网络，比如在到家之前打开某些节点，则需要网关。

2.19 何时使用 IV Update 更新程序？

一旦节点的底层检测到发送的消息的序列号达到临界值，IV Update 更新程序便会启用。

2.20 如何启用 IV Update 更新程序？

节点可以使用带有 Secure Network Beacon 的 IV Update 更新程序。

3. ESP-BLE-MESH 和 Wi-Fi 共存

3.1 Wi-Fi 和 ESP-BLE-MESH 共存时，支持哪些模式？

目前，只有 Wi-Fi STA 模式支持共存。

3.2 Wi-Fi 和 ESP-BLE-MESH 共存时，为什么 Wi-Fi 吞吐量很低？

未搭载 PSRAM 的 ESP32-DevKitC 开发板，Wi-Fi 和 ESP-BLE-MESH 共存可以正常运行，但是吞吐率较低。当 Wi-Fi 和 ESP-BLE-MESH 共存时，搭载 PSRAM 的 ESP32-DevKitC 速率可以稳定在 1 Mbps 以上。

应使能 menuconfig 中的一些配置来支持 PSRAM。

- `ESP32-specific --> Support for external,SPI-connected RAM --> Try to allocate memories of Wi-Fi and LWIP...`

- `Bluetooth --> Bluedriod Enable --> BT/BLE will first malloc the memory from the PSRAM`
- `Bluetooth --> Bluedriod Enable --> Use dynamic memory allocation in BT/BLE stack.`
- `Bluetooth --> Blutooth controller --> BLE full scan feature supported.`
- `Wi-Fi --> Software controls Wi-Fi/Bluetooth coexistence --> Wi-Fi`

4. 快速配网

4.1 为什么需要快速配网？

通常而言，存在少量未配网设备时，用户可以逐个配置。但是如果有大量未配网设备（比如 100 个）时，逐个配置会耗费大量时间。通过快速配网，用户可以在约 50 秒内配网 100 个未配网设备。

4.2 为什么会出现 EspBleMesh App 在快速配网期间长时间等待的情况？

快速配网期间，代理节点在配置完一个节点后会断开与 APP 的连接，待所有节点配网完成后再与 APP 重新建立连接。

4.3 为什么 APP 中显示的节点地址的数量比现有的节点地址更多？

每完成一次快速配网后、开始新一次快速配网前，APP 会存有上次配网的数据，因此 APP 中显示的节点地址的数量比现有的节点地址更多。

4.4 在 EspBleMesh App 中输入的 ** count ** 值有什么用途？

此 **count** 值提供给 App 配置的代理节点，以决定何时提前开始 Proxy 广播信息。

4.5 运行以下示例 fast_prov_server 的节点的 Configuration Client Model 何时开始工作？

使能了 Temporary Provisioner 功能后，Configuration Client Model 会开始工作。

4.6 Temporary Provisioner 功能会一直处于使能的状态吗？

节点收到打开/关闭电灯的消息后，所有节点会禁用其 Temporary Provisioner 功能并且转化为一般节点。

5. Log 帮助

当 ESP-BLE-MESH 协议栈底层出现错误或者警告时，您可以在这儿找到这些错误和警告的含义。

5.1 `ran out of retransmit attempts` 代表什么？

节点发送分段消息时，由于某些原因，接收端未收到完整的消息。节点会重传消息。当重传次数达到最大重传数时，会出现该警告，当前最大重传数为 4。

5.2 `Duplicate found in Network Message Cache` 代表什么？

当节点收到一条消息时，它会把该消息与网络缓存中存储的消息进行比较。如果在缓存中找到相同的消息，这意味着之前已接受过该消息，则该消息会被丢弃。

5.3 `Incomplete timer expired` 代表什么？

当节点在一定时间段（比如 10 秒）内未收到分段消息的所有段时，则 Incomplete 计时器到时，并且出现该警告。

5.4 `No matching TX context for ack` 代表什么？

当节点收到一个分段 ack 且不能找到任何自己发送的与该 ack 相关的消息时，会出现该警告。

5.5 `No free slots for new incoming segmented messages` 代表什么？

当节点没有空间来接收新的分段消息时，会出现该警告。用户可以通过配置*CONFIG_BLE_MESH_RX_SEG_MSG_COUNT* 扩大空间。

5.6 `Model not bound to AppKey 0x0000` 代表什么？

当节点发送带有模型的消息且该模型尚未绑定到索引为 0x000 的应用密钥时，会出现该报错。

5.7 `Busy sending message to DST xxxx` 代表什么？

该错误表示节点的客户端模型已将消息发送给目标节点，并且正在等待响应，用户无法将消息发送到单播地址相同的同一节点。接收到相应的响应或计时器到时后，可以发送另一条消息。

6. 示例帮助

6.1 ESP-BLE-MESH 回调函数如何分类？

- API *esp_ble_mesh_register_prov_callback()* 用于注册处理配网和入网相关事件的回调函数。
- API *esp_ble_mesh_register_config_client_callback()* 用于注册处理 Configuration Client Model 相关事件的回调函数。
- API *esp_ble_mesh_register_config_server_callback()* 用于注册处理 Configuration Server Model 相关事件的回调函数。
- API *esp_ble_mesh_register_health_client_callback()* 用于注册处理 Health Client Model 相关事件的回调函数。
- API *esp_ble_mesh_register_health_server_callback()* 用于注册处理 Health Server Model 相关事件的回调函数。
- API *esp_ble_mesh_register_generic_client_callback()* 用于注册处理 Generic Client Models 相关事件的回调函数。
- API *esp_ble_mesh_register_light_client_callback()* 用于注册处理 Lighting Client Models 相关事件的回调函数。
- API *esp_ble_mesh_register_sensor_client_callback()* 用于注册处理 Sensor Client Model 相关事件的回调函数。
- API *esp_ble_mesh_register_time_scene_client_callback()* 用于注册处理 Time and Scenes Client Models 相关事件的回调函数。
- API *esp_ble_mesh_register_custom_model_callback()* 用于注册处理自定义模型和未实现服务器模型的相关事件的回调函数。

7. 其他

7.1 如何打印数据包？

示例使用如下函数 `ESP_LOG_BUFFER_HEX()` 打印信息语境，而 ESP-BLE-MESH 协议栈使用 `bt_hex()` 打印。

7.2 重启 ESP32 应使用哪个 API？

API *esp_restart()*.

7.3 如何监测任务栈的剩余空间？

API *vTaskList()* 可以用于定期打印任务栈的剩余空间。

7.4 如何在不更改 menuconfig 输出级别的情况下改变 log 级别？

无需使用 menuconfig，可以通过 API *esp_log_level_set()* 修改 log 的输出级别。

ESP-BLE-MESH Terminology

[English]

表 11: 表 1 ESP-BLE-MESH 术语 - 身份

术语	官方定义	详细说明
未配网设备	“A device that is not a member of a mesh network is known as an unprovisioned device.”	示例：照明装置、温控设备、制造设备和电动门等。
节点	“A node is a provisioned device.”	经配网加入 ESP-BLE-MESH 网络后，未配网设备的身份转变成节点。节点（如照明装置、温控设备、制造设备和电动门）是指能在蓝牙 ESP-BLE-MESH 网络中发送、接收或中继消息的设备，且节点可以选择性地支持一个或多个子网。
中继节点	“A node that supports the Relay feature and has the Relay feature enabled is known as a Relay node.”	中继节点接收并中继 ESP-BLE-MESH 消息，因此消息可以传输得更远。用户可以根据节点的状态来决定是否使能节点的中继功能。消息可以中继多次，每次中继为“一跳”，消息最多可有 126 跳，足以让消息在广阔的区域内传输。
代理节点	“A node that supports the Proxy feature and has the Proxy feature enabled is known as a Proxy node.”	代理节点从一个承载层（通常包括广播承载层和 GATT 承载层）接收消息，并通过另一个承载层重新发送消息。其目的是将只支持 GATT 承载层的通讯设备接入到 ESP-BLE-MESH 网络中。通常而言，手机 App 需要一个代理节点才能接入 Mesh 网络。没有代理节点，手机 App 无法与 Mesh 网络中成员通信。
好友节点	“A node that supports the Friend feature, has the Friend feature enabled, and has a friendship with a node that supports the Low Power feature is known as a Friend node.”	好友节点相当于低功耗节点（LPN）的备份，可存储发往低功耗节点的消息及安全更新信息；当低功耗节点需要这些存储的信息时，这些信息便会被传输至低功耗节点。低功耗节点必须与支持好友特性的另一节点建立“友谊”，以减少其接收器的占空比，从而降低低功耗节点的功耗。低功耗节点需要找到好友节点，与其建立友谊关系，其中涉及的过程称为“友谊建立”。低功耗节点与好友节点的搭配可让低功耗节点规划对无线电的使用，从而以适当或较低的频率接收消息，无需保持收听状态。低功耗节点会轮询好友节点以查看是否有新的消息。
低功耗节点	“A node that supports the Low Power feature and has a friendship with a node that supports the Friend feature is known as a Low Power node.”	低功耗节点通过轮询从好友节点获取信息，比如消息、安全更新等。
启动配置设备（以下称为 Provisioner）	“A node that is capable of adding a device to a mesh network.”	能够配网未配网设备的设备称为启动配置设备。这一流程通常需要通过产品制造商的提供的 App 来实现，并可在网关、智能手机、平板电脑和其他载体上使用。

表 12: 表 2 ESP-BLE-MESH 术语 - 节点构成

术语	官方定义	详细说明
状态	“A value representing a condition of an element that is exposed by an element of a node.”	ESP-BLE-MESH 网络中的每台设备都具有一组独立的状态值，表示设备的某些状态，比如照明设备的亮度、颜色等状态。更改状态值会修改设备本身的物理状态，比如更改设备的开关状态值实际是在打开或关闭设备。
模型	“A model defines the basic functionality of a node.”	一个节点可能包含多个模型，而每个模型定义了节点的基本功能，比如节点所需要的状态、控制状态的消息以及处理消息所产生的动作等。节点功能的实现是基于模型的，模型可分为 SIG 模型和自定义模型，前者由 SIG 定义，而后者由用户定义。
元素	“An addressable entity within a device.”	一个节点可以包含一个或多个元素，每个元素都有一个单播地址和一个或多个模型，并且同一元素所包含的模型不可以出现重复。
节点构成状态	“The Composition Data state contains information about a node, the elements it includes, and the supported models.”	通过读取节点构成状态的值，用户可以了解节点的基本信息，比如元素的数量及每个元素中的模型。Provisioner 通过获取这个消息对设备进一步配置，比如配置节点的订阅地址与发布地址。

表 13: 表 3 ESP-BLE-MESH 术语 - 特性

术语	官方定义	详细说明
低功耗特性	“The ability to operate within a mesh network at significantly reduced receiver duty cycles only in conjunction with a node supporting the Friend feature.”	低功耗功能可降低节点的功耗。当低功耗节点寻找好友节点、且附近有多个好友节点时，它会通过算法选择最适合的好友节点。
好友特性	“The ability to help a node supporting the Low Power feature to operate by storing messages destined for those nodes.”	通过使能好友特性，节点可以帮助存储低功耗节点的信息。使能了好友特性的节点可能会产生更大的功耗和内存消耗。
中继特性	“The ability to receive and retransmit mesh messages over the advertising bearer to enable larger networks.”	中继特性能让 ESP-BLE-MESH 的消息在节点之间实现多次跳跃，传输距离可超过两个节点之间直接进行无线电传输的范围，从而覆盖整个网络。使能了中继特性的节点中继消息时，只中继其所在子网的消息，不中继其它子网的消息。使能了中继特性的节点中继分段消息时不考虑数据的完整性。节点每收到一条分段消息便直接中继，不会等待收到完整的消息。
代理特性	“The ability to receive and retransmit mesh messages between GATT and advertising bearers.”	代理特性的目的是允许不具备广播承载层的节点访问 ESP-BLE-MESH 网络。代理特性通常为需要和手机 App 连接的节点所用。

表 14: 表 4 ESP-BLE-MESH 术语 - 配置入网

术语	官方定义	详细说明
PB-ADV	“PB-ADV is a provisioning bearer used to provision a device using Generic Provisioning PDUs over the advertising channels.”	PB-ADV 通过广播通道传输配网过程中产生的数据包。只有 Provisioner 和未配网设备都支持 PB-ADV 时才可使用这种方法进行配网。
PB-GATT	“PB-GATT is a provisioning bearer used to provision a device using Proxy PDUs to encapsulate Provisioning PDUs within the Mesh Provisioning Service.”	PB-GATT 通过连接通道传输配网过程中产生的数据包。如果未配网设备想使用此方式进行配网，其需要实现相关的 Mesh Provisioning Service。未实现此服务的未配网设备不能通过 PB-GATT 承载层配网接入 mesh 网络。
配置入网	“Provisioning is a process of adding an unprovisioned device to a mesh network, managed by a Provisioner.”	经过配网，“未配网设备” 的身份转变为 “节点”，成为 ESP-BLE-MESH 网络中的一员。
认证方式	“Authentication is a step during the provisioning of nodes.”	未配网设备有四种认证方法：输入带外 (Input OOB)、输出带外 (Output OOB)、静态带外 (Static OOB) 和无带外 (No OOB)。
输入带外 (Input OOB)	Input Out-of-Band	比如，Provisioner 生成并显示随机数，然后提示用户采取适当操作将随机数输入未配网的设备中。以照明开关为例，用户可以在一定时间内数次按下按钮，以这种形式输入 Provisioner 端显示的随机数。输入带外认证方法与输出带外的认证方法类似，但设备的角色相反。
输出带外 (Output OOB)	Output Out-of-Band	比如，未配网设备会选择一个随机数，并通过与其功能兼容的方式输出该数字。如果未配网设备是一个灯泡，则其能够闪烁指定的次数。如果未配网设备有 LCD 屏幕，则可以将随机数显示为多位数值。启动 Provisioner 的用户需要输入观察到的数字，来认证未配网的设备。
静态带外 (Static OOB)	Static Out-of-Band	静态 OOB 的认证方法：使用静态 OOB 信息。如果需要使用无 OOB 信息，请将静态 OOB 字段赋值为 0。如果需要使用 OOB 信息，请使用静态 OOB 信息认证正在配网的设备。
无带外 (No OOB)	No Out-of-Band	无 OOB 的认证方法：将 “静态 OOB” 字段赋值为 0。采用这种方式相当于不认证未配网的设备。

表 15: 表 5 ESP-BLE-MESH 术语 - 地址

术语	官方定义	详细说明
未分配地址	“This is a special address type, with a value of 0x0000. Its use indicates that an Element has not yet been configured or had a Unicast Address assigned to it.”	未配置的元素地址或未分配的元素地址都称为未分配地址。鉴于这些元素没有固定的地址，它们不会用于消息的传输。建议在设置用户代码的地址之前，将该地址的值设为未分配地址。
单播地址	“A unicast address is a unique address allocated to each element.”	在配网期间，Provisioner 会给网络中处于生命周期内节点的每个元素分配一个单播地址。单播地址可能会出现在消息的源/目标地址字段中。发送到单播地址的消息只能由拥有该单播地址的元素进行处理。
虚拟地址	“A virtual address represents a set of destination addresses. Each virtual address logically represents a Label UUID, which is a 128-bit value that does not have to be managed centrally.”	虚拟地址与特定的 UUID 标签相关联，可以用作模型的发布地址或订阅地址。UUID 标签是与一个或多个节点的元素相关联的 128 位值。虚拟地址的第 15 位和第 14 位分别设置为 1 和 0。从第 13 位到第 0 位设置为散列值（提供 16384 个散列值）。散列是 UUID 标签的派生。使用订阅元素检查完整的 128 位 UUID 十分低效，而散列值提供了一种更有效的方法来确定最终将哪些消息发送到哪些元素。
群组地址	“A group address is an address that is programmed into zero or more elements.”	群组地址是 ESP-BLE-MESH 网络中的另一种多播地址，通常用于将节点进行分组。发送到 all-proxies 地址的信息应由启用了代理功能的所有节点的主要元素处理。发送到 all-friends 地址的消息应由启用了好友功能的所有节点的主要元素处理。发送到 all-relays 地址的消息应由启用了中继功能的所有节点的主要元素处理。发送到 all-nodes 地址的消息应由所有节点的主要元素处理。

表 16: 表 6 ESP-BLE-MESH 术语 - 安全

术语	官方定义	详细说明
设备密钥 (DevKey)	"There is also a device key, which is a special application key that is unique to each node, is known only to the node and a Configuration Client, and is used to secure communications between the node and a Configuration Client."	设备密钥让您能够配网未配网设备、配置节点。设备密钥用来加密配置信息，即配置设备时 Provisioner 和节点之间传输的消息。
应用密钥 (AppKey)	"Application keys are used to secure communications at the upper transport layer."	应用密钥用于应用数据传递至应用层过程中对应用数据的解密，和应用层下发过程中对数据的加密。网络中的一些节点有特定的用途，并且可以根据应用程序的需求对一些潜在敏感数据的访问进行限制。通过特定的应用密钥，这些节点与特定应用程序相关联。通常而言，使用不同应用密钥的领域有安全（楼宇门禁、机房门禁和 CEO 办公室门禁）、照明（工厂、外部楼宇和人行道）和 HVAC 系统。应用密钥绑定在网络密钥上，这意味着应用密钥仅在绑定网络密钥的情况下使用。每一个应用密钥仅可绑定到一个网络密钥。
主安全资料	"The master security material is derived from the network key (NetKey) and can be used by other nodes in the same network. Messages encrypted with master security material can be decoded by any node in the same network. "	使用好友安全材料加密的相应友谊消息有：1. 好友轮询 (Friend Poll)，2. 好友更新 (Friend Update)，3. 好友订阅列表 (Friend Subscription List)，添加/删除/确认，4. 好友节点发送到低功耗节点的 "已存储消息"，使用主安全材料加密的相应友谊消息有：1. 好友清除 (Friend Clear)，2. 好友清除确认 (Friend Clear Confirm)。根据应用程序的设置，从低功耗节点发送到好友节点的消息会使用友谊安全材料或主安全材料进行加密，前者用于低功耗节点与好友节点之间的消息传输，而后者用于其他网络消息。

表 17: 表 7 ESP-BLE-MESH 术语 - 消息

术语	官方定义	详细说明
重组 / 分包	“Segmentation and reassembly (SAR) is a method of communication network, which is divided into small units before transmitting packets and reassembled in a proper order at the communication receiving end.”	底层传输层会自动分包过大的消息。接收端会回复一条应答消息，根据应答消息，发送端会重新向接收端发送其未接收到的数据包。这些都是底层传输层自动完成的。未分包的消息最多携带 15 个字节，其中 4 个字节是 transMIC，所以剩余 11 个字节；在分包的情况下，前面的包中每包有 12 个有效字节，最后一个包中有 8 个有效字节。特殊情况：一个较短的包需要底层传输端强制分包，这种情况下有 8 个有效字节。
无应答 / 有应答	“There are two types of messages: Unacknowledged or Acknowledged.”	根据接收端是否需要发送应答消息，发送的消息可分为两种。发送端需要设置最大重传次数。

表 18: 表 8 ESP-BLE-MESH 术语 - 基础模型

术语	官方定义	详细说明
Configuration Server Model	“This model is used to represent a mesh network configuration of a device.”	节点必须包含 Configuration Server Model，其负责维护配置相关的状态。Configuration Server Model 维护的状态包含：网络密钥名单 (NetKey List)、应用密钥名单 (AppKey List)、模型绑定的应用密钥名单 (Model to AppKey List)、节点身份 (Node Identity)、密钥更新阶段 (Key Refresh Phase)、心跳消息发布 (Heartbeat Publish)、心跳消息订阅 (Heartbeat Subscription)、网络传输 (Network Transmit) 和中继重传 (Relay Retransmit) 等。
Configuration Client Model	“The model is used to represent an element that can control and monitor the configuration of a node.”	Configuration Client Model 通过消息控制 Configuration Server Model 维护的状态。Provisioner 必须包含 Configuration Client Model，有了该模型才可发送 “Configuration Composition Data Get” 等配置消息。
Health Server Model	“This model is used to represent a mesh network diagnostics of a device.”	Health Server Model 主要用于设备检查自身状态，查看自身是否发生错误。Health Server model 维护的状态包含：当前故障 (Current Fault)、已登记故障 (Registered Fault)、健康周期 (Health Period) 和关注计时器 (Attention Timer)。
Health Client Model	“The model is used to represent an element that can control and monitor the health of a node.”	Health Client Model 通过消息控制 Health Server Model 维护的状态。该模型可通过消息 “Health Fault Get” 获取其他节点的自检信息。

表 19: 表 9 ESP-BLE-MESH 术语 - 网络管理

术语	官方定义	详细说明
密钥更新程序	“This procedure is used when the security of one or more network keys and/or one or more of the application keys has been compromised or could be compromised.”	密钥更新程序用于更新 ESP-BLE-MESH 网络的网络密钥和应用密钥。当一个或多个网络密钥和/或一个或多个应用密钥的安全受到威胁或可能受到威胁时，会启动密钥更新程序。通常而言，在网络中某些节点移除后可以进行密钥更新。
IV 更新程序	“A node can also use an IV Update procedure to signal to peer nodes that it is updating the IV Index.”	IV 更新程序用于更新 ESP-BLE-MESH 网络的 IV Index 的值，这个值和消息加密时所需的随机数相关。为了保证随机数的值不重复，所以将这个值定期增加。IV Index 是一个 32 位的值，是一种共享网络资源，比如一个 mesh 网中的所有节点共享一个 IV Index 值。IV Index 从 0x00000000 开始，在 IV 更新过程中递增，并由特定的进程维护，以保证整个 Mesh 网内共享一个 IV Index。当节点认为它有耗尽其序列号的风险，或它确定另一个节点即将耗尽其序列号时，可以启动该程序。注意：每次的更新时间不得低于 96 小时。节点接收到 secure network beacon 或者确定自己的序列号大于特定值时，会触发 IV 更新程序。

官方定义摘自 ESP-BLE-MESH Glossary of Terms. 查看更多术语，也请参照上述网址。

蓝牙 SIG 文档

- BLE Mesh Profile Specification
- BLE Mesh Model Specification
- An Intro to Bluetooth Mesh Part 1 / Part 2
- The Fundamental Concepts of Bluetooth Mesh Networking, Part 1 / Part 2
- Bluetooth Mesh Networking: Friendship
- Management of Devices in a Bluetooth Mesh Network
- Bluetooth Mesh Security Overview
- Provisioning a Bluetooth Mesh Network Part 1 / Part 2

5.27 ESP-MESH

This guide provides information regarding the ESP-MESH protocol. Please see the *MESH API Reference* for more information about API usage.

5.27.1 Overview

ESP-MESH is a networking protocol built atop the Wi-Fi protocol. ESP-MESH allows numerous devices (henceforth referred to as nodes) spread over a large physical area (both indoors and outdoors) to be interconnected under a single WLAN (Wireless Local-Area Network). ESP-MESH is self-organizing and self-healing meaning the network can be built and maintained autonomously.

The ESP-MESH guide is split into the following sections:

1. *Introduction*
2. *ESP-MESH Concepts*
3. *Building a Network*
4. *Managing a Network*
5. *Data Transmission*
6. *Channel Switching*
7. *Performance*
8. *Further Notes*

5.27.2 Introduction

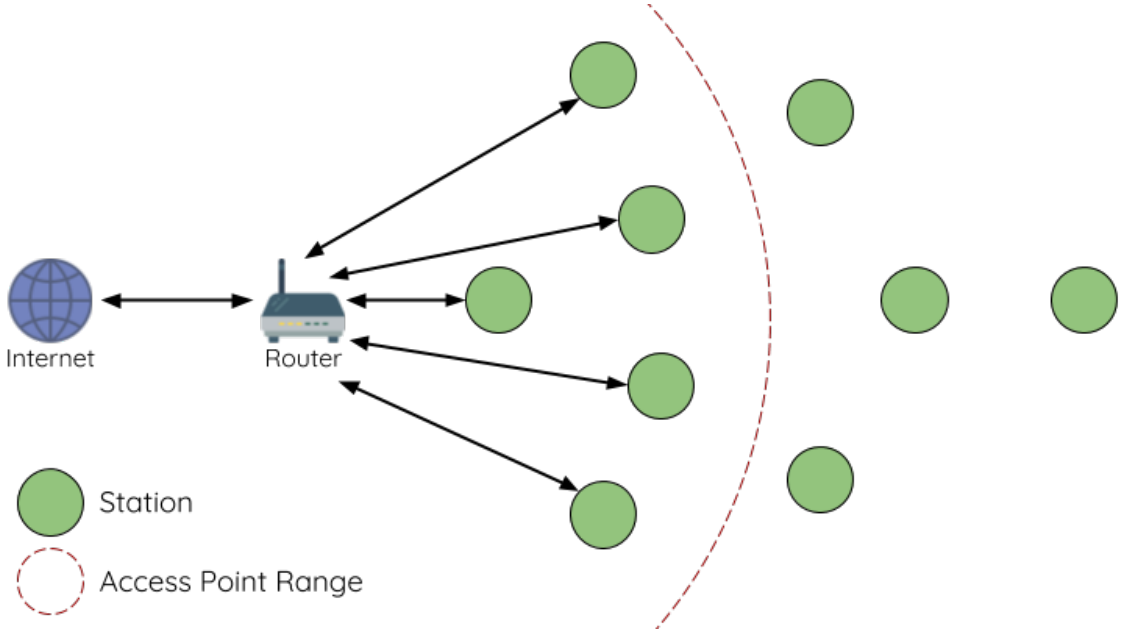


图 55: Traditional Wi-Fi Network Architectures

A traditional infrastructure Wi-Fi network is a point-to-multipoint network where a single central node known as the access point (AP) is directly connected to all other nodes known as stations. The AP is responsible for

arbitrating and forwarding transmissions between the stations. Some APs also relay transmissions to/from an external IP network via a router. Traditional infrastructure Wi-Fi networks suffer the disadvantage of limited coverage area due to the requirement that every station must be in range to directly connect with the AP. Furthermore, traditional Wi-Fi networks are susceptible to overloading as the maximum number of stations permitted in the network is limited by the capacity of the AP.

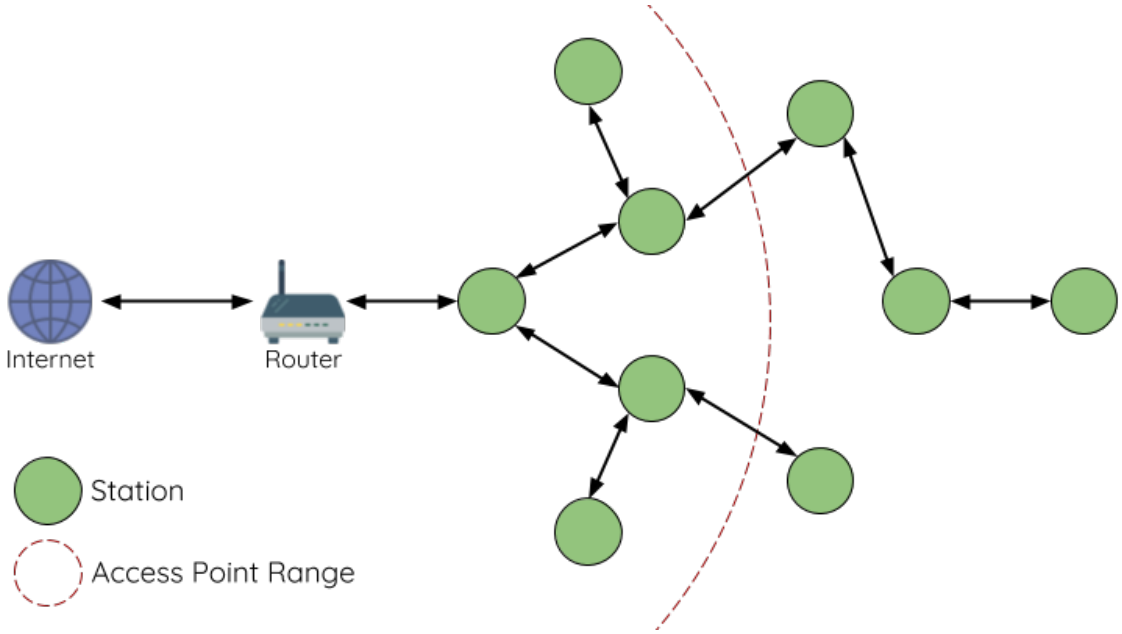


图 56: ESP-MESH Network Architecture

ESP-MESH differs from traditional infrastructure Wi-Fi networks in that nodes are not required to connect to a central node. Instead, nodes are permitted to connect with neighboring nodes. Nodes are mutually responsible for relaying each others transmissions. This allows an ESP-MESH network to have much greater coverage area as nodes can still achieve interconnectivity without needing to be in range of the central node. Likewise, ESP-MESH is also less susceptible to overloading as the number of nodes permitted on the network is no longer limited by a single central node.

5.27.3 ESP-MESH Concepts

Terminology

Term	Description
Node	Any device that **is** or **can be** part of an ESP-MESH network
Root Node	The top node in the network
Child Node	A node X is a child node when it is connected to another node Y where the connection makes node X more distant from the root node than node Y (in terms of number of connections).
Parent Node	The converse notion of a child node
Sub-Child Node	Any node reachable by repeated proceeding from parent to child
Sibling Nodes	Nodes that share the same parent node
Connection	A traditional Wi-Fi association between an AP and a station. A node in ESP-MESH will use its station interface to associate with the softAP interface of another node, thus forming a connection. The connection process includes the authentication and association processes in Wi-Fi.
Upstream Connection	The connection from a node to its parent node
Downstream Connection	The connection from a node to one of its child nodes
Wireless Hop	The portion of the path between source and destination nodes that corresponds to a single wireless connection. A data packet that traverses a single connection is known as **single-hop** whereas traversing multiple connections is known as **multi-hop**.
Subnetwork	A subnetwork is subdivision of an ESP-MESH network which consists of a node and all of its descendant nodes. Therefore the subnetwork of the root node consists of all nodes in an ESP-MESH network.
MAC Address	Media Access Control Address used to uniquely identify each node or router within an ESP-MESH network.
DS	Distribution System (External IP Network)

Tree Topology

ESP-MESH is built atop the infrastructure Wi-Fi protocol and can be thought of as a networking protocol that combines many individual Wi-Fi networks into a single WLAN. In Wi-Fi, stations are limited to a single connection with an AP (upstream connection) at any time, whilst an AP can be simultaneously connected to multiple stations (downstream connections). However ESP-MESH allows nodes to simultaneously act as a station and an AP. Therefore a node in ESP-MESH can have **multiple downstream connections using its softAP interface**, whilst simultaneously having **a single upstream connection using its station interface**. This naturally results in a tree network topology with a parent-child hierarchy consisting of multiple layers.

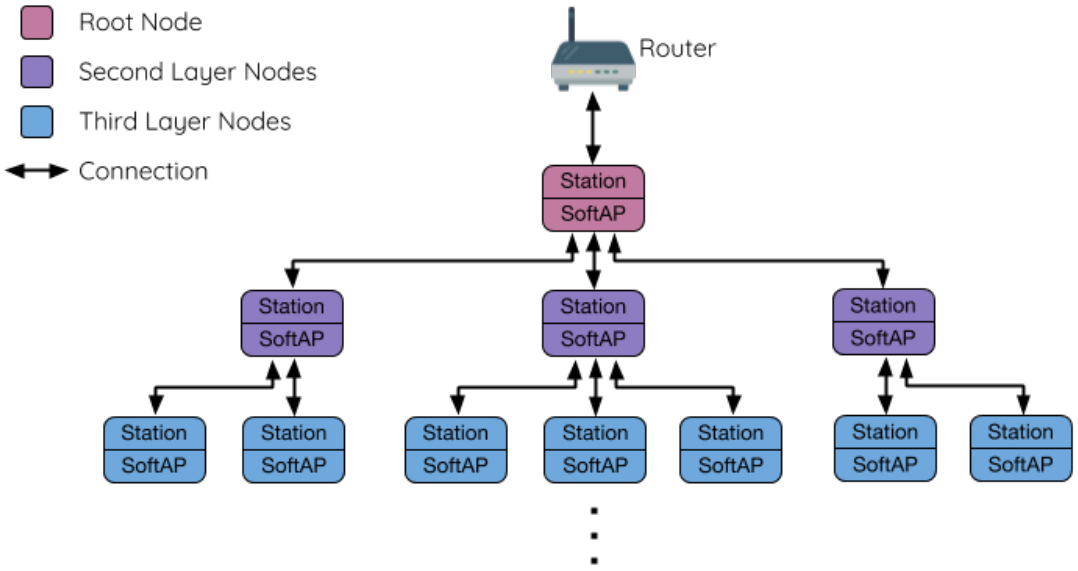


图 57: ESP-MESH Tree Topology

ESP-MESH is a multiple hop (multi-hop) network meaning nodes can transmit packets to other nodes in the network through one or more wireless hops. Therefore, nodes in ESP-MESH not only transmit their own packets, but simultaneously serve as relays for other nodes. Provided that a path exists between any two nodes on the physical layer (via one or more wireless hops), any pair of nodes within an ESP-MESH network can communicate.

注解: The size (total number of nodes) in an ESP-MESH network is dependent on the maximum number of layers permitted in the network, and the maximum number of downstream connections each node can have. Both of these variables can be configured to limit the size of the network.

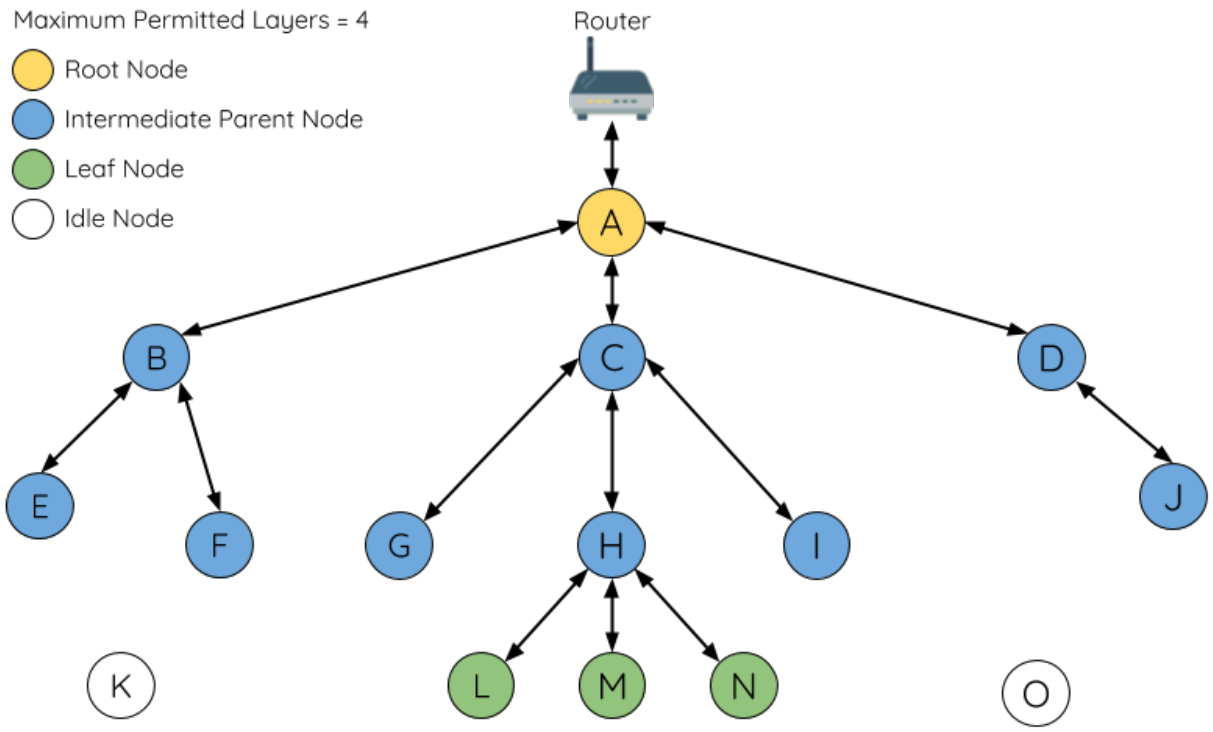


图 58: ESP-MESH Node Types

Node Types

Root Node: The root node is the top node in the network and serves as the only interface between the ESP-MESH network and an external IP network. The root node is connected to a conventional Wi-Fi router and relays packets to/from the external IP network to nodes within the ESP-MESH network. **There can only be one root node within an ESP-MESH network** and the root node' s upstream connection may only be with the router. Referring to the diagram above, node A is the root node of the network.

Leaf Nodes: A leaf node is a node that is not permitted to have any child nodes (no downstream connections). Therefore a leaf node can only transmit or receive its own packets, but cannot forward the packets of other nodes. If a node is situated on the network' s maximum permitted layer, it will be assigned as a leaf node. This prevents the node from forming any downstream connections thus ensuring the network does not add an extra layer. Some nodes without a softAP interface (station only) will also be assigned as leaf nodes due to the requirement of a softAP interface for any downstream connections. Referring to the diagram above, nodes L/M/N are situated on the networks maximum permitted layer hence have been assigned as leaf nodes .

Intermediate Parent Nodes: Connected nodes that are neither the root node or a leaf node are intermediate parent nodes. An intermediate parent node must have a single upstream connection (a single parent node), but can have zero to multiple downstream connections (zero to multiple child nodes). Therefore an intermediate parent node can transmit and receive packets, but also forward packets sent from its upstream

and downstream connections. Referring to the diagram above, nodes B to J are intermediate parent nodes. **Intermediate parent nodes without downstream connections such as nodes E/F/G/I/J are not equivalent to leaf nodes** as they are still permitted to form downstream connections in the future.

Idle Nodes: Nodes that have yet to join the network are assigned as idle nodes. Idle nodes will attempt to form an upstream connection with an intermediate parent node or attempt to become the root node under the correct circumstances (see *Automatic Root Node Selection*). Referring to the diagram above, nodes K and O are idle nodes.

Beacon Frames & RSSI Thresholding

Every node in ESP-MESH that is able to form downstream connections (i.e. has a softAP interface) will periodically transmit Wi-Fi beacon frames. A node uses beacon frames to allow other nodes to detect its presence and know of its status. Idle nodes will listen for beacon frames to generate a list of potential parent nodes, one of which the idle node will form an upstream connection with. ESP-MESH uses the Vendor Information Element to store metadata such as:

- Node Type (Root, Intermediate Parent, Leaf, Idle)
- Current layer of Node
- Maximum number of layers permitted in the network
- Current number of child nodes
- Maximum number of downstream connections to accept

The signal strength of a potential upstream connection is represented by RSSI (Received Signal Strength Indication) of the beacon frames of the potential parent node. To prevent nodes from forming a weak upstream connection, ESP-MESH implements an RSSI threshold mechanism for beacon frames. If a node detects a beacon frame with an RSSI below a preconfigured threshold, the transmitting node will be disregarded when forming an upstream connection.

Panel A of the illustration above demonstrates how the RSSI threshold affects the number of parent node candidates an idle node has.

Panel B of the illustration above demonstrates how an RF shielding object can lower the RSSI of a potential parent node. Due to the RF shielding object, the area in which the RSSI of node X is above the threshold is significantly reduced. This causes the idle node to disregard node X even though node X is physically adjacent. The idle node will instead form an upstream connection with the physically distant node Y due to a stronger RSSI.

注解: Nodes technically still receive all beacon frames on the MAC layer. The RSSI threshold is an ESP-MESH feature that simply filters out all received beacon frames that are below the preconfigured threshold.

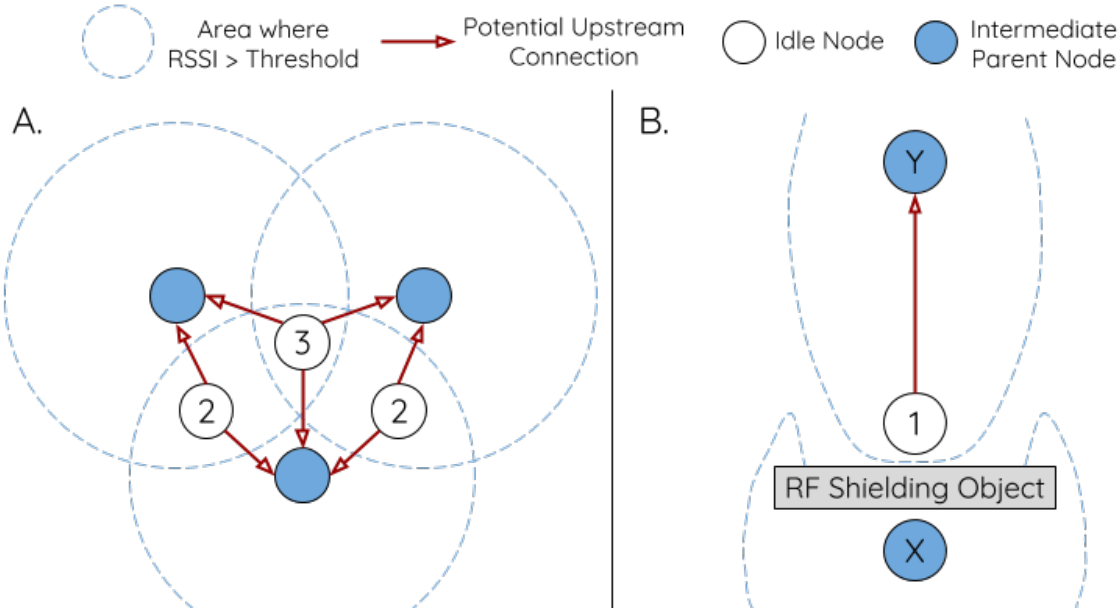


图 59: Effects of RSSI Thresholding

Preferred Parent Node

When an idle node has multiple parent nodes candidates (potential parent nodes), the idle node will form an upstream connection with the **preferred parent node**. The preferred parent node is determined based on the following criteria:

- Which layer the parent node candidate is situated on
- The number of downstream connections (child nodes) the parent node candidate currently has

The selection of the preferred parent node will always prioritize the parent node candidate on the shallowest layer of the network (including the root node). This helps minimize the total number of layers in an ESP-MESH network when upstream connections are formed. For example, given a second layer node and a third layer node, the second layer node will always be preferred.

If there are multiple parent node candidates within the same layer, the parent node candidate with the least child nodes will be preferred. This criteria has the effect of balancing the number of downstream connections amongst nodes of the same layer.

Panel A of the illustration above demonstrates an example of how the idle node G selects a preferred parent node given the five parent node candidates B/C/D/E/F. Nodes on the shallowest layer are preferred, hence nodes B/C are prioritized since they are second layer nodes whereas nodes D/E/F are on the third layer. Node C is selected as the preferred parent node due it having fewer downstream connections (fewer child nodes) compared to node B.

Panel B of the illustration above demonstrates the case where the root node is within range of the idle node G. In other words, the root node' s beacon frames are above the RSSI threshold when received by node G.

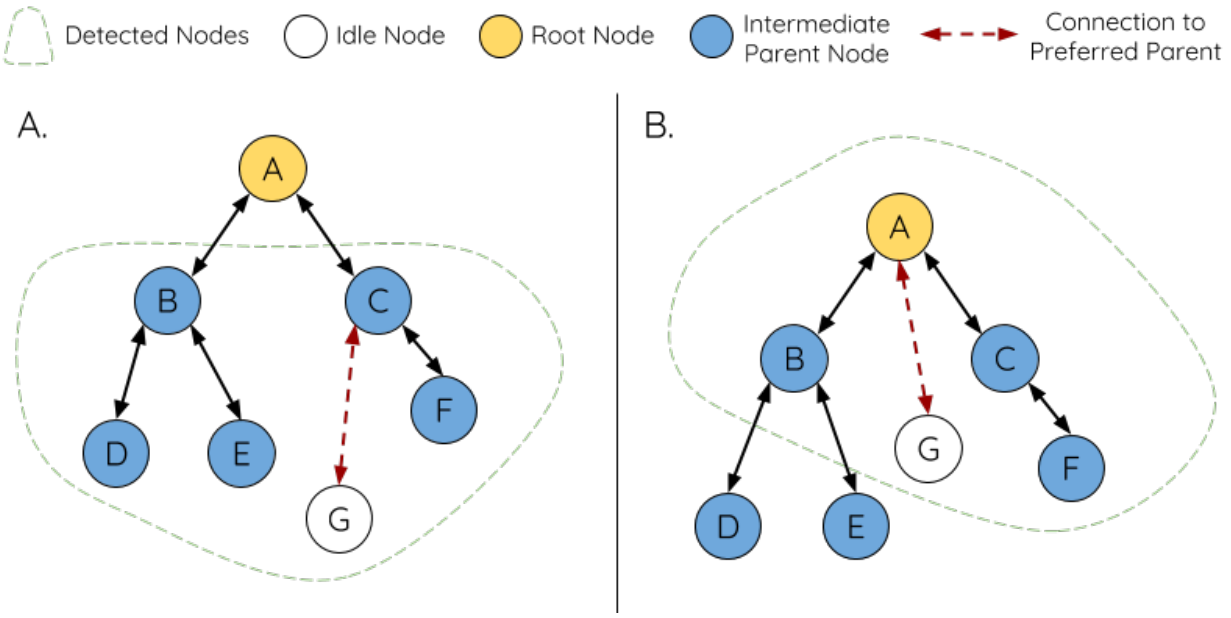


图 60: Preferred Parent Node Selection

The root node is always the shallowest node in an ESP-MESH network hence is always the preferred parent node given multiple parent node candidates.

注解: Users may also define their own algorithm for selecting a preferred parent node, or force a node to only connect with a specific parent node (see the Mesh Manual Networking Example).

Routing Tables

Each node within an ESP-MESH network will maintain its individual routing table used to correctly route ESP-MESH packets (see *ESP-MESH Packet*) to the correct destination node. The routing table of a particular node will **consist of the MAC addresses of all nodes within the particular node' s subnetwork** (including the MAC address of the particular node itself). Each routing table is internally partitioned into multiple subtables with each subtable corresponding to the subnetwork of each child node.

Using the diagram above as an example, the routing table of node B would consist of the MAC addresses of nodes B to I (i.e. equivalent to the subnetwork of node B). Node B' s routing table is internally partitioned into two subtables containing of nodes C to F and nodes G to I (i.e. equivalent to the subnetworks of nodes C and G respectively).

ESP-MESH utilizes routing tables to determine whether an ESP-MESH packet should be forwarded upstream or downstream based on the following rules.

1. If the packet' s destination MAC address is within the current node' s routing table and is not the current node, select the subtable that contains the destination MAC address and forward the data packet

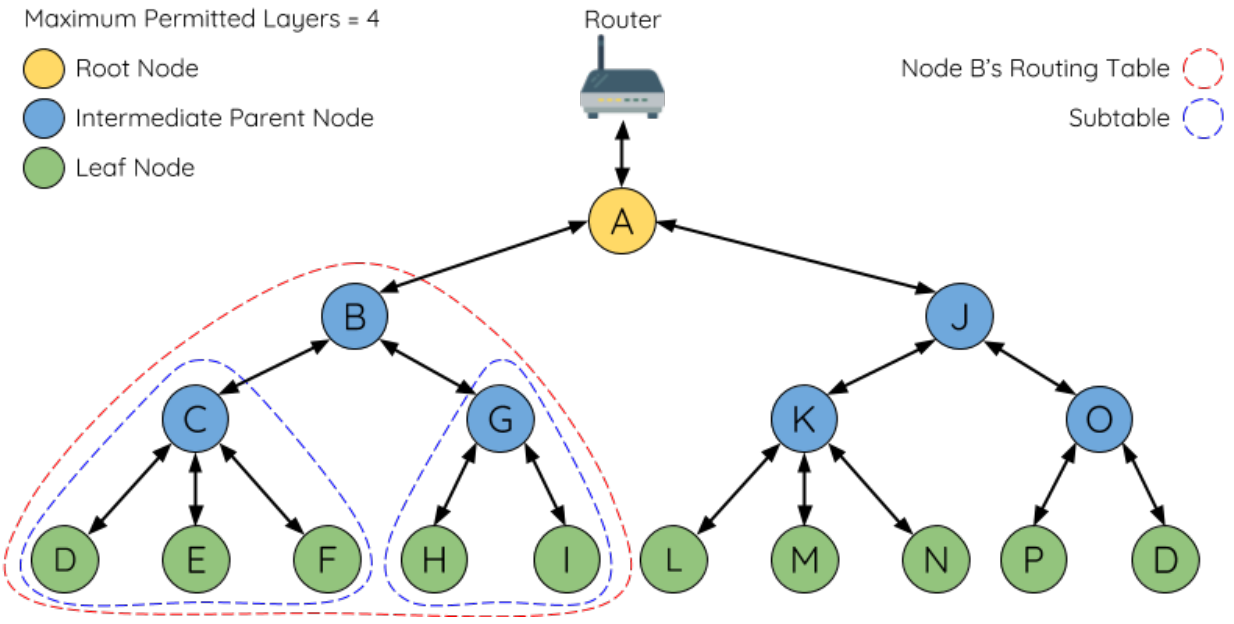


图 61: ESP-MESH Routing Tables Example

downstream to the child node corresponding to the subtable.

2. If the destination MAC address is not within the current node' s routing table, forward the data packet upstream to the current node' s parent node. Doing so repeatedly will result in the packet arriving at the root node where the routing table should contain all nodes within the network.

注解: Users can call *esp_mesh_get_routing_table()* to obtain a node' s routing table, or *esp_mesh_get_routing_table_size()* to obtain the size of a node' s routing table.

esp_mesh_get_subnet_nodes_list() can be used to obtain the corresponding subtable of a specific child node. Likewise *esp_mesh_get_subnet_nodes_num()* can be used to obtain the size of the subtable.

5.27.4 Building a Network

General Process

警告: Before the ESP-MESH network building process can begin, certain parts of the configuration must be uniform across each node in the network (see *mesh_cfg_t*). Each node must be configured with **the same Mesh Network ID, router configuration, and softAP configuration**.

An ESP-MESH network building process involves selecting a root node, then forming downstream connections layer by layer until all nodes have joined the network. The exact layout of the network can be dependent

on factors such as root node selection, parent node selection, and asynchronous power-on reset. However, the ESP-MESH network building process can be generalized into the following steps:

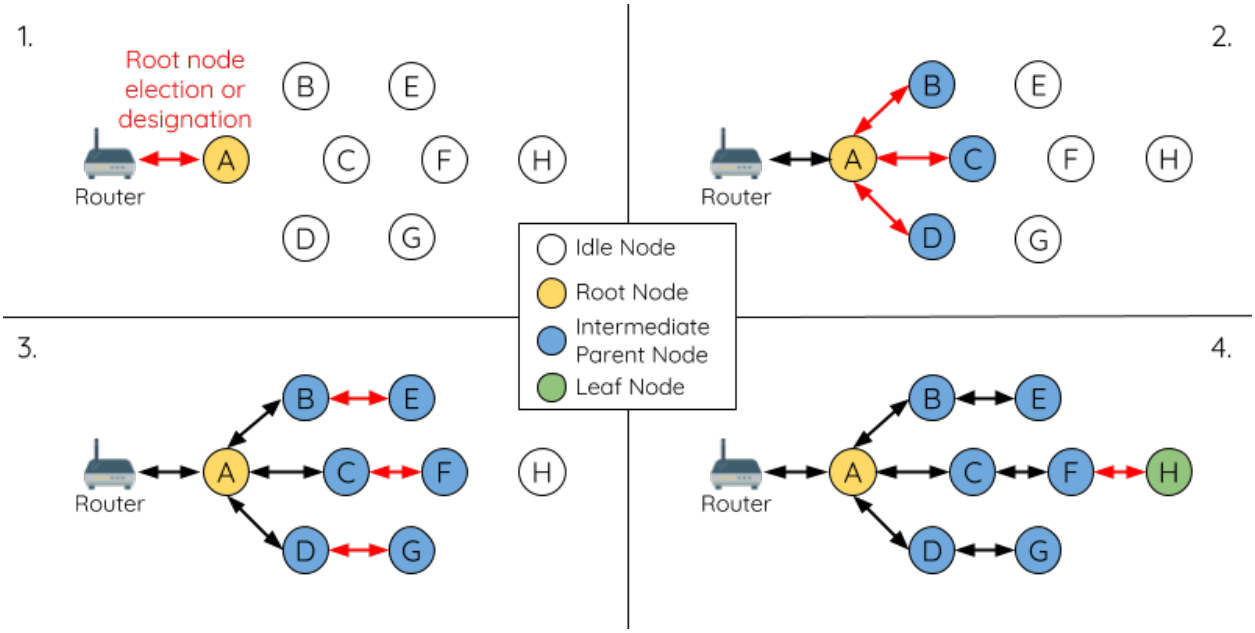


图 62: ESP-MESH Network Building Process

1. Root Node Selection

The root node can be designated during configuration (see section on *User Designated Root Node*), or dynamically elected based on the signal strength between each node and the router (see *Automatic Root Node Selection*). Once selected, the root node will connect with the router and begin allowing downstream connections to form. Referring to the figure above, node A is selected to be the root node hence node A forms an upstream connection with the router.

2. Second Layer Formation

Once the root node has connected to the router, idle nodes in range of the root node will begin connecting with the root node thereby forming the second layer of the network. Once connected, the second layer nodes become intermediate parent nodes (assuming maximum permitted layers > 2) hence the next layer to form. Referring to the figure above, nodes B to D are in range of the root node. Therefore nodes B to D form upstream connections with the root node and become intermediate parent nodes.

3. Formation of remaining layers

The remaining idle nodes will connect with intermediate parent nodes within range thereby forming a new layer in the network. Once connected, the idles nodes become intermediate parent node or leaf nodes

depending on the networks maximum permitted layers. This step is repeated until there are no more idle nodes within the network or until the maximum permitted layer of the network has been reached. Referring to the figure above, nodes E/F/G connect with nodes B/C/D respectively and become intermediate parent nodes themselves.

4. Limiting Tree Depth

To prevent the network from exceeding the maximum permitted number of layers, nodes on the maximum layer will automatically become leaf nodes once connected. This prevents any other idle node from connecting with the leaf node thereby prevent a new layer form forming. However if an idle node has no other potential parent node, it will remain idle indefinitely. Referring to the figure above, the network' s maximum permitted layers is set to four. Therefore when node H connects, it becomes a leaf node to prevent any downstream connections from forming.

Automatic Root Node Selection

The automatic selection of a root node involves an election process amongst all idle nodes based on their signal strengths with the router. Each idle node will transmit their MAC addresses and router RSSI values via Wi-Fi beacon frames. **The MAC address is used to uniquely identify each node in the network** whilst the **router RSSI** is used to indicate a node' s signal strength with reference to the router.

Each node will then simultaneously scan for the beacon frames from other idle nodes. If a node detects a beacon frame with a stronger router RSSI, the node will begin transmitting the contents of that beacon frame (i.e. voting for the node with the stronger router RSSI). The process of transmission and scanning will repeat for a preconfigured minimum number of iterations (10 iterations by default) and result in the beacon frame with the strongest router RSSI being propagated throughout the network.

After all iterations, each node will individually check for its **vote percentage** (`number of votes/number of nodes participating in election`) to determine if it should become the root node. **If a node has a vote percentage larger than a preconfigured threshold (90% by default), the node will become a root node**.

The following diagram demonstrates how an ESP-MESH network is built when the root node is automatically selected.

1. On power-on reset, each node begins transmitting beacon frames consisting of their own MAC addresses and their router RSSIs.

2. Over multiple iterations of transmission and scanning, the beacon frame with the strongest router RSSI is propagated throughout the network. Node C has the strongest router RSSI (-10db) hence its beacon frame is propagated throughout the network. All nodes participating in the election vote for node C thus giving node C a vote percentage of 100%. Therefore node C becomes a root node and connects with the router.

3. Once Node C has connected with the router, nodes A/B/D/E connectwith node C as it is the preferred parent node (i.e. the shallowest node). Nodes A/B/D/E form the second layer of the network.

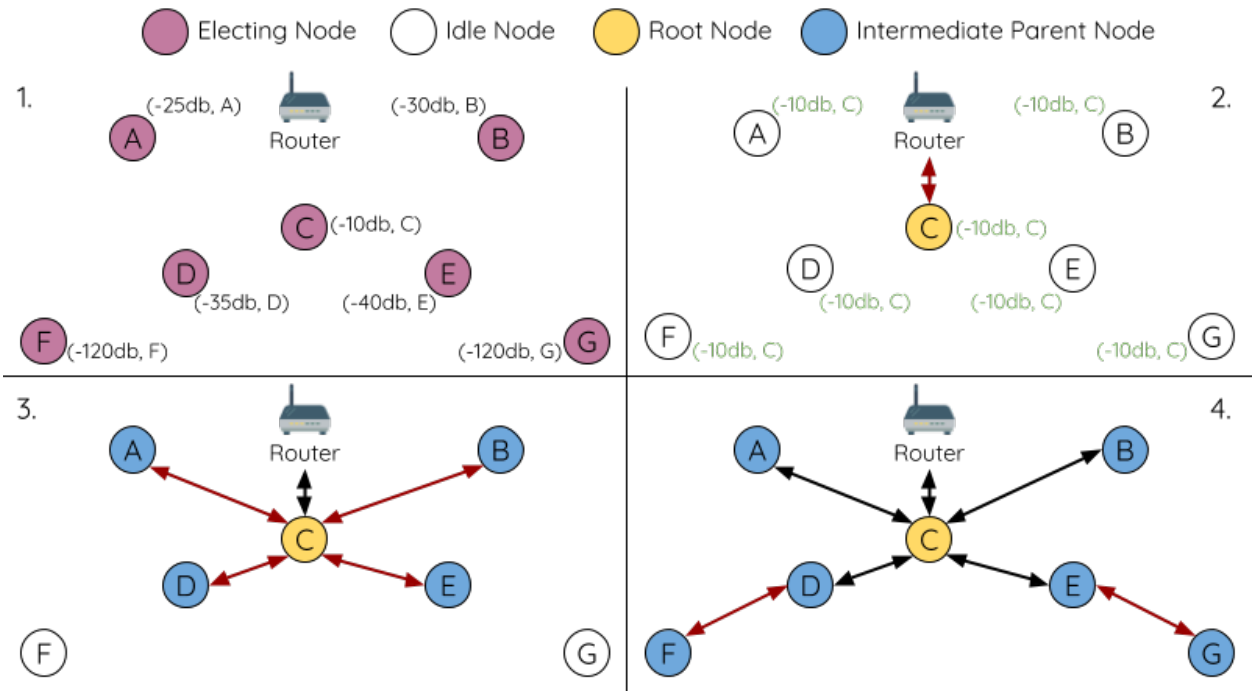


图 63: Root Node Election Example

4. Node F and G connect with nodes D and E respectively and the network building process is complete.

注解: The minimum number of iterations for the election process can be configured using `esp_mesh_set_attempts()`. Users should adjust the number of iterations based on the number of nodes within the network (i.e. the larger the network the larger number of scan iterations required).

警告: **Vote percentage threshold** can also be configured using *esp_mesh_set_vote_percentage()*. Setting a low vote percentage threshold **can result in two or more nodes becoming root nodes** within the same ESP-MESH network leading to the building of multiple networks. If such is the case, ESP-MESH has internal mechanisms to autonomously resolve the **root node conflict**. The networks of the multiple root nodes will be combined into a single network with a single root node. However, root node conflicts where two or more root nodes have the same router SSID but different router BSSID are not handled.

User Designated Root Node

The root node can also be designated by user which will entail the designated root node to directly connect with the router and forgo the election process. When a root node is designated, all other nodes within the network must also forgo the election process to prevent the occurrence of a root node conflict. The following

diagram demonstrates how an ESP-MESH network is built when the root node is designated by the user.

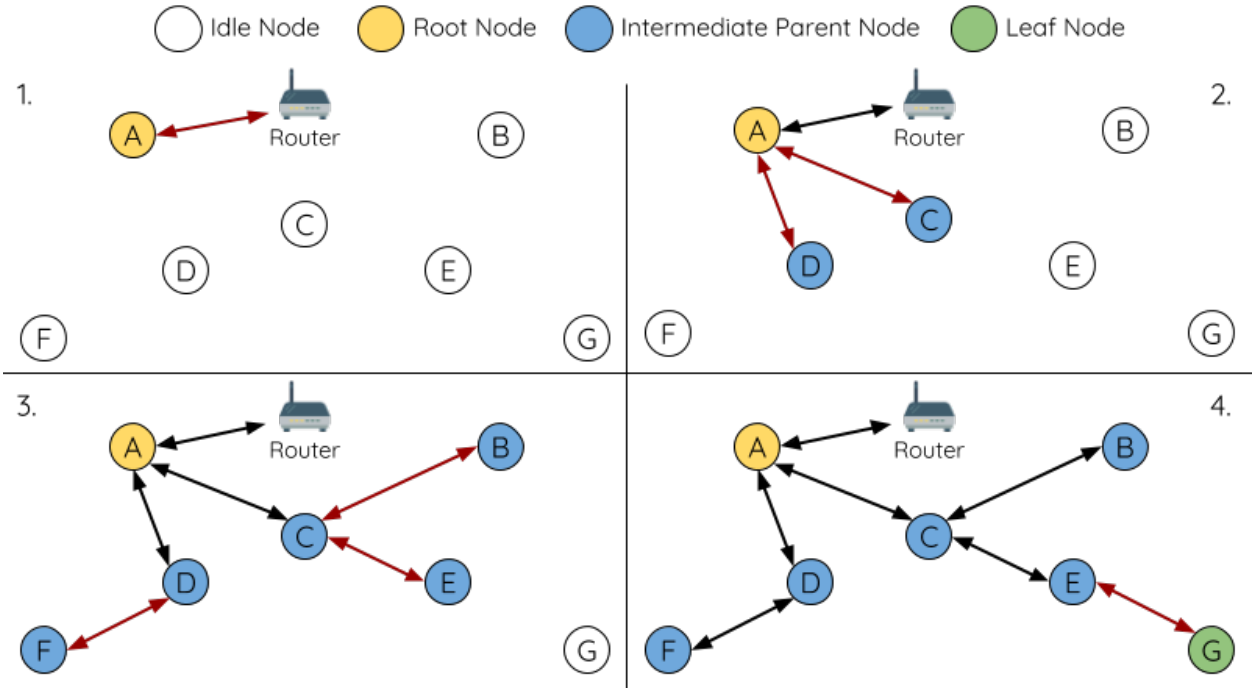


图 64: Root Node Designation Example (Root Node = A, Max Layers = 4)

1. Node A is designated the root node by the user therefore directly connects with the router. All other nodes forgo the election process.

2. Nodes C/D connect with node A as their preferred parent node. Both nodes form the second layer of the network.

3. Likewise, nodes B/E connect with node C, and node F connects with node D. Nodes B/E/F form the third layer of the network.

4. Node G connects with node E, forming the fourth layer of the network. However the maximum permitted number of layers in this network is configured as four, therefore node G becomes a leaf node to prevent any new layers from forming.

注解: When designating a root node, the root node should call *esp_mesh_set_parent()* in order to directly connect with the router. Likewise, all other nodes should call *esp_mesh_fix_root()* to forgo the election process.

Parent Node Selection

By default, ESP-MESH is self organizing meaning that each node will autonomously select which potential parent node to form an upstream connection with. The autonomously selected parent node is known as the

preferred parent node. The criteria used for selecting the preferred parent node is designed to reduce the number of layers in the ESP-MESH network and to balance the number of downstream connections between potential parent nodes (see section on *Preferred Parent Node*).

However ESP-MESH also allows users to disable self-organizing behavior which will allow users to define their own criteria for parent node selection, or to configure nodes to have designated parent nodes (see the Mesh Manual Networking Example).

Asynchronous Power-on Reset

ESP-MESH network building can be affected by the order in which nodes power-on. If certain nodes within the network power-on asynchronously (i.e. separated by several minutes), **the final structure of the network could differ from the ideal case where all nodes are powered on synchronously**. Nodes that are delayed in powering on will adhere to the following rules:

Rule 1: If a root node already exists in the network, the delayed node will not attempt to elect a new root node, even if it has a stronger RSSI with the router. The delayed node will instead join the network like any other idle node by connecting with a preferred parent node. If the delayed node is the designated root node, all other nodes in the network will remain idle until the delayed node powers-on.

Rule 2: If a delayed node forms an upstream connection and becomes an intermediate parent node, it may also become the new preferred parent of other nodes (i.e. being a shallower node). This will cause the other nodes to switch their upstream connections to connect with the delayed node (see *Parent Node Switching*).

Rule 3: If an idle node has a designated parent node which is delayed in powering-on, the idle node will not attempt to form any upstream connections in the absence of its designated parent node. The idle node will remain idle indefinitely until its designated parent node powers-on.

The following example demonstrates the effects of asynchronous power-on with regards to network building.

1. Nodes A/C/D/F/G/H are powered-on synchronously and begin the root node election process by broadcasting their MAC addresses and router RSSIs. Node A is elected as the root node as it has the strongest RSSI.

2. Once node A becomes the root node, the remaining nodes begin forming upstream connections layer by layer with their preferred parent nodes. The result is a network with five layers.

3. Node B/E are delayed in powering-on but neither attempt to become the root node even though they have stronger router RSSIs (-20db and -10db) compared to node A. Instead both delayed nodes form upstream connections with their preferred parent nodes A and C respectively. Both Nodes B/E become intermediate parent nodes after connecting.

4. Nodes D/G switch their upstream connections as node B is the new preferred parent node due to it being on a shallower layer (second layer node). Due to the switch, the resultant network has three layers instead of the original five layers.

Synchronous Power-On: Had all nodes powered-on synchronously, node E would have become the root node as it has the strongest router RSSI (-10db). This would result in a significantly different network

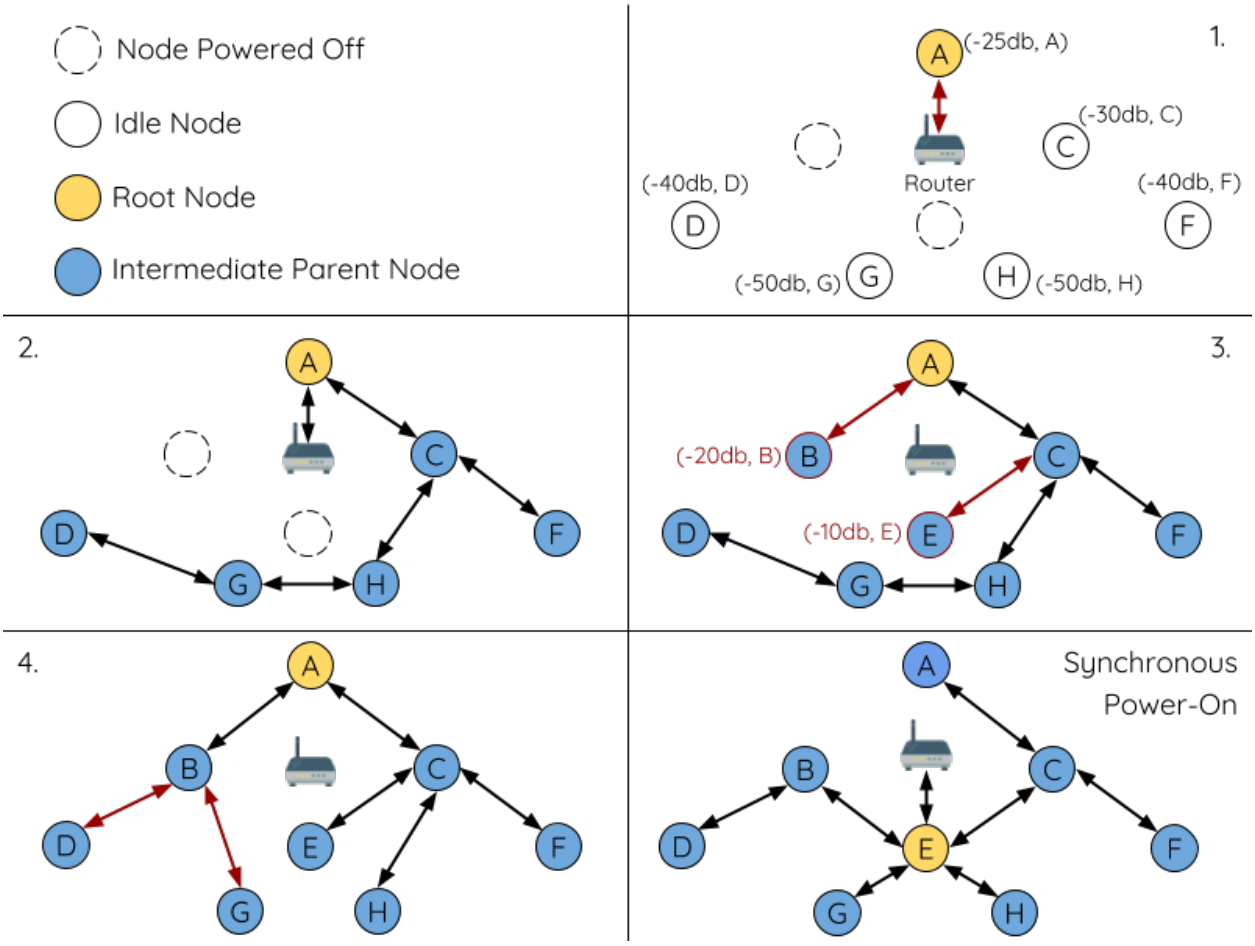


图 65: Network Building with Asynchronous Power On Example

layout compared to the network formed under the conditions of asynchronous power-on. **However the synchronous power-on network layout can still be reached if the user manually switches the root node** (see *esp_mesh_waive_root()*).

注解: Differences in parent node selection caused by asynchronous power-on are autonomously corrected for to some extent in ESP-MESH (see *Parent Node Switching*)

Loop-back Avoidance, Detection, and Handling

A loop-back is the situation where a particular node forms an upstream connection with one of its descendant nodes (a node within the particular node' s subnetwork). This results in a circular connection path thereby breaking the tree topology. ESP-MESH prevents loop-back during parent selection by excluding nodes already present in the selecting node' s routing table (see *Routing Tables*) thus prevents a particular node from attempting to connect to any node within its subnetwork.

In the event that a loop-back occurs, ESP-MESH utilizes a path verification mechanism and energy transfer mechanism to detect the loop-back occurrence. The parent node of the upstream connection that caused the loop-back will then inform the child node of the loop-back and initiate a disconnection.

5.27.5 Managing a Network

ESP-MESH is a self healing network meaning it can detect and correct for failures in network routing. Failures occur when a parent node with one or more child nodes breaks down, or when the connection between a parent node and its child nodes becomes unstable. Child nodes in ESP-MESH will autonomously select a new parent node and form an upstream connection with it to maintain network interconnectivity. ESP-MESH can handle both Root Node Failures and Intermediate Parent Node Failures.

Root Node Failure

If the root node breaks down, the nodes connected with it (second layer nodes) will promptly detect the failure of the root node. The second layer nodes will initially attempt to reconnect with the root node. However after multiple failed attempts, the second layer nodes will initialize a new round of root node election. **The second layer node with the strongest router RSSI will be elected as the new root node** whilst the remaining second layer nodes will form an upstream connection with the new root node (or a neighboring parent node if not in range).

If the root node and multiple downstream layers simultaneously break down (e.g. root node, second layer, and third layer), the shallowest layer that is still functioning will initialize the root node election. The following example illustrates an example of self healing from a root node break down.

1. Node C is the root node of the network. Nodes A/B/D/E are second layer nodes connected to node C.

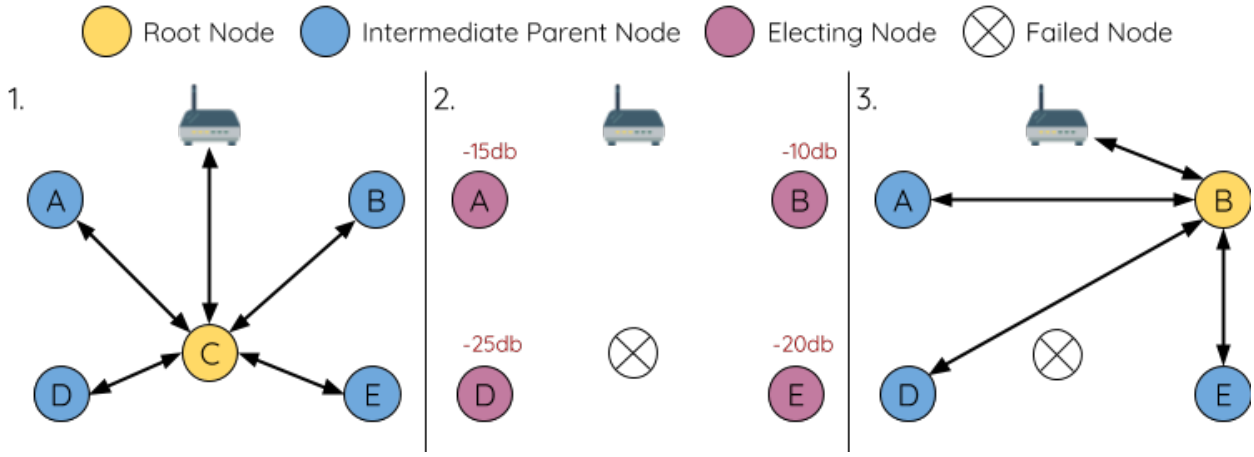


图 66: Self Healing From Root Node Failure

2. Node C breaks down. After multiple failed attempts to reconnect, the second layer nodes begin the election process by broadcasting their router RSSIs. Node B has the strongest router RSSI.

3. Node B is elected as the root node and begins accepting downstream connections. The remaining second layer nodes A/D/E form upstream connections with node B thus the network is healed and can continue operating normally.

注解: If a designated root node breaks down, the remaining nodes **will not autonomously attempt to elect a new root node** as an election process will never be attempted whilst a designated root node is used.

Intermediate Parent Node Failure

If an intermediate parent node breaks down, the disconnected child nodes will initially attempt to reconnect with the parent node. After multiple failed attempts to reconnect, each child node will begin to scan for potential parent nodes (see *Beacon Frames & RSSI Thresholding*).

If other potential parent nodes are available, each child node will individually select a new preferred parent node (see *Preferred Parent Node*) and form an upstream connection with it. If there are no other potential parent nodes for a particular child node, it will remain idle indefinitely.

The following diagram illustrates an example of self healing from an Intermediate Parent Node break down.

1. The following branch of the network consists of nodes A to G.

2. Node C breaks down. Nodes F/G detect the break down and attempt to reconnect with node C. After multiple failed attempts to reconnect, nodes F/G begin to select a new preferred parent node.

3. Node G is out of range from any other parent node hence remains idle for the time being. Node F is in range of nodes B/E, however node B is selected as it is the shallower node. Node F becomes an intermediate

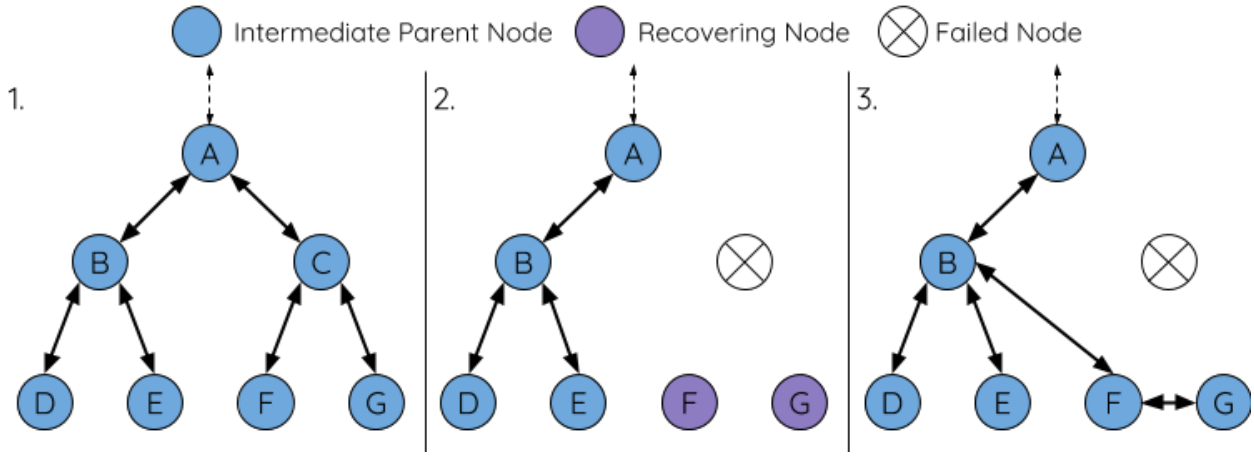


图 67: Self Healing From Intermediate Parent Node Failure

parent node after connecting with Node B thus node G can connect with node F. The network is healed, however the network routing as been affected and an extra layer has been added.

注解: If a child node has a designated parent node that breaks down, the child node will make no attempt to connect with a new parent node. The child node will remain idle indefinitely.

Root Node Switching

ESP-MESH does not automatically switch the root node unless the root node breaks down. Even if the root node' s router RSSI degrades to the point of disconnection, the root node will remain unchanged. Root node switching is the act of explicitly starting a new election such that a node with a stronger router RSSI will be elected as the new root node. This can be a useful method of adapting to degrading root node performance.

To trigger a root node switch, the current root node must explicitly call *esp_mesh_waive_root()* to trigger a new election. The current root node will signal all nodes within the network to begin transmitting and scanning for beacon frames (see *Automatic Root Node Selection*) **whilst remaining connected to the network (i.e. not idle)**. If another node receives more votes than the current root node, a root node switch will be initiated. **The root node will remain unchanged otherwise**.

A newly elected root node sends a **switch request** to the current root node which in turn will respond with an acknowledgment signifying both nodes are ready to switch. Once the acknowledgment is received, the newly elected root node will disconnect from its parent and promptly form an upstream connection with the router thereby becoming the new root node of the network. The previous root node will disconnect from the router **whilst maintaining all of its downstream connections** and enter the idle state. The previous root node will then begin scanning for potential parent nodes and selecting a preferred parent.

The following diagram illustrates an example of a root node switch.

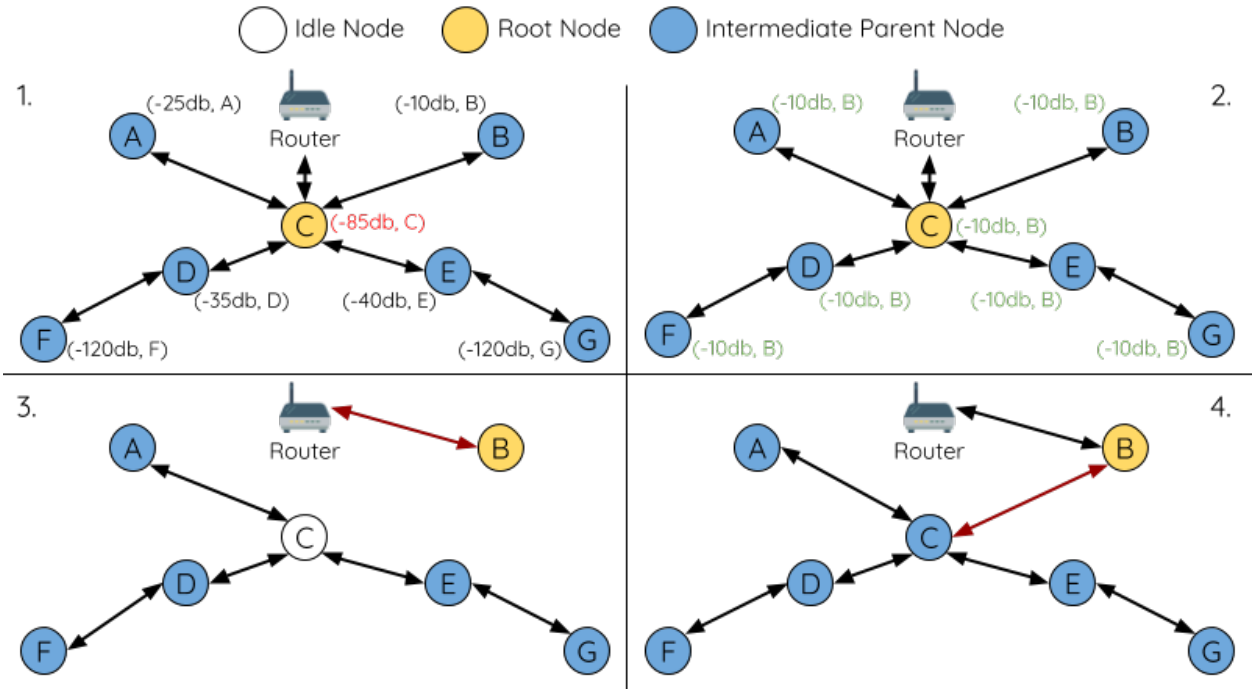


图 68: Root Node Switch Example

1. Node C is the current root node but has degraded signal strength with the router (-85db). The node C triggers a new election and all nodes begin transmitting and scanning for beacon frames **whilst still being connected**.

2. After multiple rounds of transmission and scanning, node B is elected as the new root node. Node B sends node C a **switch request** and node C responds with an acknowledgment.

3. Node B disconnects from its parent and connects with the router becoming the networks new root node. Node C disconnects from the router, enters the idle state, and begins scanning for and selecting a new preferred parent node. **Node C maintains all its downstream connections throughout this process**.

4. Node C selects node B as its preferred parent node, forms an upstream connection, and becomes a second layer node. The network layout is similar after the switch as node C still maintains the same subnetwork. However each node in node C' s subnetwork has been placed one layer deeper as a result of the switch. *Parent Node Switching* may adjust the network layout afterwards if any nodes have a new preferred parent node as a result of the root node switch.

注解: Root node switching must require an election hence is only supported when using a self-organized ESP-MESH network. In other words, root node switching cannot occur if a designated root node is used.

Parent Node Switching

Parent Node Switching entails a child node switching its upstream connection to another parent node of a shallower layer. **Parent Node Switching occurs autonomously** meaning that a child node will change its upstream connection automatically if a potential parent node of a shallower layer becomes available (i.e. due to a *Asynchronous Power-on Reset*).

All potential parent nodes periodically transmit beacon frames (see *Beacon Frames & RSSI Thresholding*) allowing for a child node to scan for the availability of a shallower parent node. Due to parent node switching, a self-organized ESP-MESH network can dynamically adjust its network layout to ensure each connection has a good RSSI and that the number of layers in the network is minimized.

5.27.6 Data Transmission

ESP-MESH Packet

ESP-MESH network data transmissions use ESP-MESH packets. ESP-MESH packets are **entirely contained within the frame body of a Wi-Fi data frame**. A multi-hop data transmission in an ESP-MESH network will involve a single ESP-MESH packet being carried over each wireless hop by a different Wi-Fi data frame.

The following diagram shows the structure of an ESP-MESH packet and its relation with a Wi-Fi data frame.

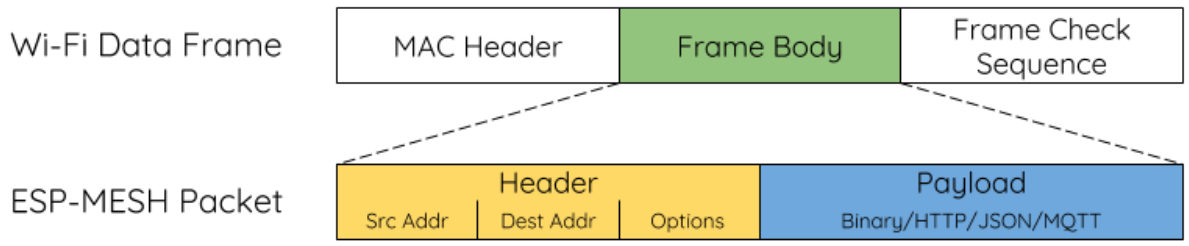


图 69: ESP-MESH Packet

The header of an ESP-MESH packet contains the MAC addresses of the source and destination nodes. The options field contains information pertaining to the special types of ESP-MESH packets such as a group transmission or a packet originating from the external IP network (see *MESH_OPT_SEND_GROUP* and *MESH_OPT_RECV_DS_ADDR*).

The payload of an ESP-MESH packet contains the actual application data. This data can be raw binary data, or encoded under an application layer protocol such as HTTP, MQTT, and JSON (see *mesh_proto_t*).

注解: When sending an ESP-MESH packet to the external IP network, the destination address field of the header will contain the IP address and port of the target server rather than the MAC address of a node (see

mesh_addr_t). Furthermore the root node will handle the formation of the outgoing TCP/IP packet.

Group Control & Multicasting

Multicasting is a feature that allows a single ESP-MESH packet to be transmitted simultaneously to multiple nodes within the network. Multicasting in ESP-MESH can be achieved by either specifying a list of target nodes, or specifying a preconfigured group of nodes. Both methods of multicasting are called via *esp_mesh_send()*.

To multicast by specifying a list of target nodes, users must first set the ESP-MESH packet' s destination address to the **Multicast-Group Address** (`01:00:5E:xx:xx:xx`). This signifies that the ESP-MESH packet is a multicast packet with a group of addresses, and that the address should be obtained from the header options. Users must then list the MAC addresses of the target nodes as options (see *mesh_opt_t* and *MESH_OPT_SEND_GROUP*). This method of multicasting requires no prior setup but can incur a large amount of overhead data as each target node' s MAC address must be listed in the options field of the header.

Multicasting by group allows a ESP-MESH packet to be transmitted to a preconfigured group of nodes. Each grouping is identified by a unique ID, and a node can be placed into a group via *esp_mesh_set_group_id()*. Multicasting to a group involves setting the destination address of the ESP-MESH packet to the target group ID. Furthermore, the *MESH_DATA_GROUP* flag must set. Using groups to multicast incurs less overhead, but requires nodes to previously added into groups.

注解: During a multicast, all nodes within the network still receive the ESP-MESH packet on the MAC layer. However, nodes not included in the MAC address list or the target group will simply filter out the packet.

Broadcasting

Broadcasting is a feature that allows a single ESP-MESH packet to be transmitted simultaneously to all nodes within the network. Each node essentially forwards a broadcast packet to all of its upstream and downstream connections such that the packet propagates throughout the network as quickly as possible. However, ESP-MESH utilizes the following methods to avoid wasting bandwidth during a broadcast.

1. When an intermediate parent node receives a broadcast packet from its parent, it will forward the packet to each of its child nodes whilst storing a copy of the packet for itself.

2. When an intermediate parent node is the source node of the broadcast, it will transmit the broadcast packet upstream to is parent node and downstream to each of its child nodes.

3. When an intermediate parent node receives a broadcast packet from one of its child nodes, it will forward the packet to its parent node and each of its remaining child nodes whilst storing a copy of the packet for itself.

4. When a leaf node is the source node of a broadcast, it will directly transmit the packet to its parent node.

5. When the root node is the source node of a broadcast, the root node will transmit the packet to all of its child nodes.

6. When the root node receives a broadcast packet from one of its child nodes, it will forward the packet to each of its remaining child nodes whilst storing a copy of the packet for itself.

7. When a node receives a broadcast packet with a source address matching its own MAC address, the node will discard the broadcast packet.

8. When an intermediate parent node receives a broadcast packet from its parent node which was originally transmitted from one of its child nodes, it will discard the broadcast packet

Upstream Flow Control

ESP-MESH relies on parent nodes to control the upstream data flow of their immediate child nodes. To prevent a parent node's message buffer from overflowing due to an overload of upstream transmissions, a parent node will allocate a quota for upstream transmissions known as a **receiving window** for each of its child nodes. **Each child node must apply for a receiving window before it is permitted to transmit upstream**. The size of a receiving window can be dynamically adjusted. An upstream transmission from a child node to the parent node consists of the following steps:

1. Before each transmission, the child node sends a window request to its parent node. The window request consists of a sequence number which corresponds to the child node's data packet that is pending transmission.

2. The parent node receives the window request and compares the sequence number with the sequence number of the previous packet sent by the child node. The comparison is used to calculate the size of the receiving window which is transmitted back to the child node.

3. The child node transmits the data packet in accordance with the window size specified by the parent node. If the child node depletes its receiving window, it must obtain another receiving windows by sending a request before it is permitted to continue transmitting.

注解: ESP-MESH does not support any downstream flow control.

警告: Due to *Parent Node Switching*, packet loss may occur during upstream transmissions.

Due to the fact that the root node acts as the sole interface to an external IP network, it is critical that downstream nodes are aware of the root node's connection status with the external IP network. Failing to do so can lead to nodes attempting to pass data upstream to the root node whilst it is disconnected from the IP network. This results in unnecessary transmissions and packet loss. ESP-MESH address this issue by

providing a mechanism to stabilize the throughput of outgoing data based on the connection status between the root node and the external IP network. The root node can broadcast its external IP network connection status to all other nodes by calling *esp_mesh_post_toDS_state()*.

Bi-Directional Data Stream

The following diagram illustrates the various network layers involved in an ESP-MESH Bidirectional Data Stream.

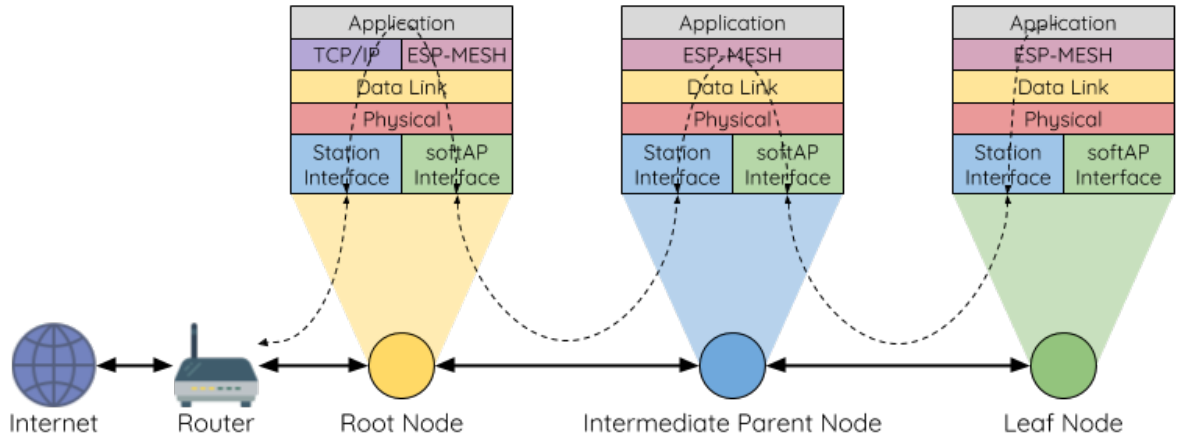


图 70: ESP-MESH Bidirectional Data Stream

Due to the use of *Routing Tables*, **ESP-MESH is able to handle pack forwarding entirely on the mesh layer**. A TCP/IP layer is only required on the root node when it transmits/receives a packet to/from an external IP network.

5.27.7 Channel Switching

Background

In traditional Wi-Fi networks, **channels** are predetermined frequency ranges. In an infrastructure basic service set (BSS), the serving AP and its connected stations must be on the same operating channels (1 to 14) in which beacons are transmitted. Physically adjacent BSS (Basic Service Sets) operating on the same channel can lead to interference and degraded performance.

In order to allow a BSS adapt to changing physical layer conditions and maintain performance, Wi-Fi contains mechanisms for **network channel switching**. A network channel switch is an attempt to move a BSS to a new operating channel whilst minimizing disruption to the BSS during this process. However it should be recognized that a channel switch may be unsuccessful in moving all stations to the new operating channel.

In an infrastructure Wi-Fi network, network channel switches are triggered by the AP with the aim of having the AP and all connected stations synchronously switch to a new channel. Network channel switching is implemented by embedding a **Channel Switch Announcement (CSA)** element within the AP' s periodically transmitted beacon frames. The CSA element is used to advertise to all connected stations regarding an upcoming network channel switch and will be included in multiple beacon frames up until the switch occurs.

A CSA element contains information regarding the **New Channel Number** and a **Channel Switch Count** which indicates the number of beacon frame intervals (TBTTs) remaining until the network channel switch occurs. Therefore, the Channel Switch Count is decremented every beacon frame and allows connected stations to synchronize their channel switch with the AP.

ESP-MESH Network Channel Switching

ESP-MESH Network Channel Switching also utilize beacon frames that contain a CSA element. However, being a multi-hop network makes the switching process in ESP-MESH is more complex due to the fact that a beacon frame might not be able to reach all nodes within the network (i.e. in a single hop). Therefore, an ESP-MESH network relies on nodes to forward the CSA element so that it is propagated throughout the network.

When an intermediate parent node with one or more child nodes receives a beacon frame containing a CSA, the node will forward the CSA element by including the element in its next transmitted beacon frame (i.e. with the same **New Channel Number** and **Channel Switch Count**). Given that all nodes within an ESP-MESH network receive the same CSA, the nodes can synchronize their channel switches using the Channel Switch Count, albeit with a short delay due to CSA element forwarding.

An ESP-MESH network channel switch can be triggered by either the router or the root node.

Root Node Triggered

A root node triggered channel switch can only occur when the ESP-MESH network is not connected to a router. By calling *esp_mesh_switch_channel()*, the root node will set an initial Channel Switch Count value and begin including a CSA element in its beacon frames. Each CSA element is then received by second layer nodes, and forwarded downstream in the their own beacon frames.

Router Triggered

When an ESP-MESH network is connected to a router, the entire network must use the same channel as the router. Therefore, **the root node will not be permitted to trigger a channel switch when it is connected to a router**.

When the root node receives beacon frame containing a CSA element from the router, **the root node will set Channel Switch Count value in the CSA element to a custom value before forwarding it**

downstream via beacon frames. It will also decrement the Channel Switch Count of subsequent CSA elements relative to the custom value. This custom value can be based on factors such as the number of network layers, the current number of nodes etc.

The setting the Channel Switch Count value to a custom value is due to the fact that the ESP-MESH network and its router may have a different and varying beacon intervals. Therefore, the Channel Switch Count value provided by the router is irrelevant to an ESP-MESH network. By using a custom value, nodes within the ESP-MESH network are able to switch channels synchronously relative to the ESP-MESH network' s beacon interval. However, this will also result in the ESP-MESH network' s channel switch being unsynchronized with the channel switch of the router and its connected stations.

Impact of Network Channel Switching

- **Due to the ESP-MESH network channel switch being unsynchronized with the router' s channel switc**
  - The ESP-MESH network' s channel switch time is dependent on the ESP-MESH network' s beacon interval and the root node' s custom Channel Switch Count value.
  - The channel discrepancy prevents any data exchange between the root node and the router during that ESP-MESH network' s switch.
  - In the ESP-MESH network, the root node and intermediate parent nodes will request their connected child nodes to stop transmissions until the channel switch takes place by setting the **Channel Switch Mode** field in the CSA element to 1.
  - Frequent router triggered network channel switches can degrade the ESP-MESH network' s performance. Note that this can be caused by the ESP-MESH network itself (e.g. due to wireless medium contention with ESP-MESH network). If this is the case, users should disable the automatic channel switching on the router and use a specified channel instead.
- **When there is a temporary channel discrepancy, the root node remains technically connected to the r**
  - Disconnection occurs after the root node fails to receive any beacon frames or probe responses from the router over a fixed number of router beacon intervals.
  - Upon disconnection, the root node will automatically re-scan all channels for the presence of a router.
- **If the root node is unable to receive any of the router' s CSA beacon frames (e.g. due to short switch**
  - After the router switches channels, the root node will no longer be able to receive the router' s beacon frames and probe responses and result in a disconnection after a fixed number of beacon intervals.
  - The root node will re-scan all channels for the router after disconnection.

– The root node will maintain downstream connections throughout this process.

注解: Although ESP-MESH network channel switching aims to move all nodes within the network to a new operating channel, it should be recognized that a channel switch might not successfully move all nodes (e.g. due to reasons such as node failures).

Channel and Router Switching Configuration

ESP-MESH allows for autonomous channel switching to be enabled/disabled via configuration. Likewise, autonomous router switching (i.e. when a root node autonomously connects to another router) can also be enabled/disabled by configuration. Autonomous channel switching and router switching is dependent on the following configuration parameters and run-time conditions.

Allow Channel Switch: This parameter is set via the `allow_channel_switch` field of the *`mesh_cfg_t`* structure and permits an ESP-MESH network to dynamically switch channels when set.

Preset Channel: An ESP-MESH network can have a preset channel by setting the `channel` field of the *`mesh_cfg_t`* structure to the desired channel number. If this field is unset, the `allow_channel_switch` parameter is overridden such that channel switches are always permitted.

Allow Router Switch: This parameter is set via the `allow_router_switch` field of the *`mesh_router_t`* and permits an ESP-MESH to dynamically switch to a different router when set.

Preset Router BSSID: An ESP-MESH network can have a preset router by setting the `bssid` field of the *`mesh_router_t`* structure to the BSSID of the desired router. If this field is unset, the `allow_router_switch` parameter is overridden such that router switches are always permitted.

Root Node Present: The presence of a root node will can also affect whether or a channel or router switch is permitted.

The following table illustrates how the different combinations of parameters/conditions affect whether channel switching and/or router switching is permitted. Note that *X* represents a "don't care" for the parameter.

Configuration and Conditions					Result
Preset Channel	Allow Channel Switch	Preset Router BSSID	Allow Router Switch	Root Node Present	Permitted Switches
N	X	N	X	X	Channel & Router
		Y	N		Channel Only
		Y	Y		Channel & Router
Y	Y	N	X	X	Channel & Router
	N			N	Router Only
	N			Y	Channel & Router
	Y	Y	N	X	Channel Only
	N			N	None
	N			Y	Channel Only
	Y		Y	X	Channel & Router
	N			N	Router Only
	N			Y	Channel & Router

5.27.8 Performance

The performance of an ESP-MESH network can be evaluated based on multiple metrics such as the following:

Network Building Time: The amount of time taken to build an ESP-MESH network from scratch.

Healing Time: The amount of time taken for the network to detect a node break down and carry out appropriate actions to heal the network (such as generating a new root node or forming new connections).

Per-hop latency: The latency of data transmission over one wireless hop. In other words, the time taken to transmit a data packet from a parent node to a child node or vice versa.

Network Node Capacity: The total number of nodes the ESP-MESH network can simultaneously support. This number is determined by the maximum number of downstream connections a node can accept and the maximum number of layers permissible in the network.

The following table lists the common performance figures of an ESP-MESH network. However users should note that performance numbers can vary greatly between installations based on network configuration and operating environment.

Function	Description
Networking Building Time	< 60 seconds
Healing time	Root Node Break Down: < 10 seconds Child Node Break Down: < 5 seconds
Per-hop latency	10 to 30 milliseconds

注解: The following test conditions were used to generate the performance figures above.

- Number of test devices: **100**
- Maximum Downstream Connections to Accept: **6**
- Maximum Permissible Layers: **6**

注解: Throughput depends on packet error rate and hop count.

注解: The throughput of root node' s access to the external IP network is directly affected by the number of nodes in the ESP-MESH network and the bandwidth of the router.

5.27.9 Further Notes

- Data transmission uses Wi-Fi WPA2-PSK encryption
- Mesh networking IE uses AES encryption

Router and internet icon made by Smashicons from www.flaticon.com

5.28 BluFi

[English]

5.28.1 概览

BluFi 是一款基于蓝牙通道的 Wi-Fi 网络配置功能，适用于 ESP32。它通过安全协议将 Wi-Fi 配置和证书传输到 ESP32，然后 ESP32 可基于这些信息连接到 AP 或建立 SoftAP。

BluFi 流程的关键部分包括数据的分片、加密、校验和验证。

用户可按需自定义用于对称加密、非对称加密和校验的算法。这里我们采用 DH 算法进行密钥协商、128-AES 算法用于数据加密、CRC16 算法用于校验和验证。

5.28.2 BluFi 流程

BluFi 配网功能包含配置 SoftAP 和 Station 两部分。

下面以配置 Station 为例说明配置步骤。BluFi 配网的配置 Station 包含广播、连接、服务发现、协商共享密钥、传输数据、回传连接状态等步骤。

5.28.3 ESP32 配网流程

1. ESP32 开启 GATT Server 功能，发送带有特定 *adv data* 的广播。你可以自定义该广播，该广播不属于 BluFi Profile。
2. 使用手机 APP 搜索到该特定广播，手机作为 GATT Client 连接 ESP32。你可以决定使用哪款手机 APP。
3. GATT 连接建立成功后，手机向 ESP32 发送“协商过程”数据帧（详情见*BluFi 传输格式* ）。
4. ESP32 收到“协商过程”数据帧后，会按照使用者自定义的协商过程来解析。
5. 手机与 ESP32 进行密钥协商。协商过程可使用 DH/RSA/ECC 等加密算法进行。
6. 协商结束后，手机端向 ESP32 发送“设置安全模式”控制帧。
7. ESP32 收到“设置安全模式”控制帧后，使用经过协商的共享密钥以及配置的安全策略对通信数据进行加密和解密。
8. 手机向 ESP32 发送“BluFi 传输格式”定义的 SSID、Password 等用于 Wi-Fi 连接的必要信息。
9. 手机向 ESP32 发送“Wi-Fi 连接请求”控制帧，ESP32 收到之后，识别为手机已将必要的信息传输完毕，准备连接 Wi-Fi。
10. ESP32 连接到 Wi-Fi 后，发送“Wi-Fi 连接状态报告”控制帧到手机，以报告连接状态。至此配网结束。

注解:

1. 安全模式设置可在任何时候进行，ESP32 收到安全模式的配置后，会根据安全模式指定的模式进行安全相关的操作。
2. 进行对称加密和解密时，加密和解密前后的数据长度必须一致，支持原地加密和解密。

图 71: BluFi Flow Chart

5.28.4 配网流程图

5.28.5 BluFi 传输格式

手机 APP 与 ESP32 之间的 BluFi 通信格式定义如下：

帧不分片情况下的标准格式 (8 bit)：

Description	Value
LSB - Type	1
Frame Control	1
Sequence Number	1
Data Length	1
Data	${Data Length}
MSB - CheckSum	2

如果 **Frame Control** 帧中的 **More Frag** 使能，则 **Total Content Length** 为数据帧中剩余部分的总长度，用于报告终端需要分配多少内存。

帧分片格式（8 bit)：

Description	Value	
LSB - Type	1	
FrameControl(Frag)	1	
SequenceNumber	1	
DataLength	1	
Data	Total Content Length	2
	Content	${Data Length} - 2
MSB - CheckSum	2	

通常情况下，控制帧不包含数据位，Ack 帧类型除外。

Ack 帧格式（8 bit)：

Description	Value	
LSB - Type (Ack)	1	
Frame Control	1	
SequenceNumber	1	
DataLength	1	
Data	Acked Sequence Number	2
MSB - CheckSum	2	

1. Type

   类型域，占 1 byte。分为 Type 和 Subtype（子类型域）两部分, Type 占低 2 bit，Subtype 占高 6 bit。

   - 控制帧，暂不进行加密，可校验；
   - 数据帧，可加密，可校验。

   1.1 控制帧 (0x0 b’ 00)

控制帧(二进制)	含义	解释	备注
0x0 (b'000000)	Ack	用来回复对方发的帧, Ack 帧的 Data 域使用回复对象帧的 Sequence 值。	Data 域使用 1 byte Sequence 值，与恢复对象帧的 Sequence 值相同。
0x1 (b'000001)	Set ESP32 to the security mode.	通知 ESP32 发送数据时使用的安全模式,在该过程中可设置多次,每次设置后影响后续安全模式。在不设置的情况下, ESP32 默认控制帧和数据帧均为无校验、无加密。手机到 ESP32 方向依赖于帧 Control 域。	Data 域占用 1 byte。高 4 bit 为控制帧的安全模式，低 4bit 为数据帧的安全模式。
			b'0000：无校验、无加密；
			b'0001：有校验、无加密；
			b'0010：无校验、有加密；
			b'0011：有校验、有加密。
0x2 (b'000010)	Set the Wi-Fi opmode of ESP32.	设置 ESP32 的 Wi-Fi 模式，帧包含 opmode 信息。	data[0] 用于表示 opmode 类型，包括：
			0x00: NULL;
			0x01: STA;
			0x02: SoftAP;
			0x03: SoftAP&STA.
			如果设置有包含 AP，请尽量优先设置 AP 模式的 SSID/Password/Max Conn Number 等。
0x3 (b'000011)	Connect ESP32 to the AP.	通知 ESP32，必要的信息已经发送完毕，可以连接 AP。	不包含 Data 域。
0x4 (b'000100)	Disconnect ESP32 from the AP.	通知 ESP32 断开与 AP 的连接	不包含 Data 域。
0x5 (b'000101)	Get the status of Wi-Fi.	获取 ESP32 的 Wi-Fi 模式和状态等信息。	不包含 Data 域。ESP32 收到此控制帧后,后续会通过 Wi-Fi 连接状态报告 (Wi-Fi Connection State Report) 数据帧来回复手机端当前所处的 opmode、连接状态、SSID 等信息。提供给手机端的信息由应用决定。
0x6 (b'000110)	Disconnect the STA	处于 SoftAP 模式时，踢掉某个 STA 设备。	data[0~5] 为 STA 设备的 MAC 地址，如有多个 STA，则 [6-11] 为第二个，依次类推

1.2 数据帧 (0x1 b’ 01)

数据帧 (二进制)	含义	解释
0x0 (b’ 000000)	Negotiation data.	用来发送协商数据，传输到应用层注册的回调函数。
0x1 (b’ 000001)	BSSID for STA mode.	STA 将要连接的 AP 的 BSSID（用于隐藏 SSID)。
0x2 (b’ 000010)	SSID for STA mode.	STA 将要连接的 AP 的 SSID。
0x3 (b’ 000011)	Password for STA mode.	STA 将要连接的 AP 的密码。
0x4 (b’ 000100)	SSID for SoftAP mode.	SoftAP 模式使用的 SSID。
0x5 (b’ 000101)	Password for SoftAPmode.	SoftAP 模式使用的密码。
0x6 (b’ 000110)	Max connection number for SoftAP mode.	AP 模式的最大连接数。
0x7 (b’ 000111)	Authentication mode for SoftAP mode.	AP 模式的认证模式。
0x8 (b’ 001000)	Channel for SoftAP mode.	SoftAP 模式的通道数量。
0x9 (b’ 001001)	Username.	使用企业级加密时，Client 端的用户名。
0xa (b’ 001010)	CA certification.	进行企业级加密时使用的 CA 证书。
0xb (b’ 001011)	Client certification.	进行企业级加密时，Client 端的证书。 可包含或不包含私钥，由证书内容决定。
0xc (b’ 001100)	Server certification.	进行企业级加密时，Server 端的证书。 可包含或不包含私钥，由证书内容决定。
0xd (b’ 001101)	Client private key.	进行企业级加密时，Client 端的私钥。
0xe (b’ 001110)	Server private key.	进行企业级加密时，Server 端的私钥。
0xf (b’ 001111)	Wi-Fi connection state report.	通知手机 ESP32 的 Wi-Fi 状态，包括 STA 状态和 SoftAP 状态，用于手机配置

数据帧 (二进制)	含义	解释
		但收到手机询问 Wi-Fi 状态时，除了回复此帧外，还可回复其他数据帧。
0x10 b’ 010000	Version.	
0x11 (b’ 010001)	Wi-Fi list.	通知手机 ESP32 周围的 Wi-Fi 热点列表。
0x12 (b’ 010010)	Report error.	通知手机 BluFi 过程出现异常错误。
0x13 (b’ 010011)	Custom data.	用户发送或者接收自定义数据。

2. Frame Control

   帧控制域，占 1 byte，每个 bit 表示不同含义。

位	含义
0x01	表示帧是否加密。
	1 表示加密，0 表示未加密。
	加密部分帧括完整的 DATA 域加密之前的明文（不帧含末尾的校验）。
	控制帧暂不加密，故控制帧此位为 0。
0x02	表示帧 Data 域结尾是否帧含校验（例如 SHA1,MD5,CRC 等）需要校验的数据域包括 sequcne + data length + 明文 data。
	控制帧和数据帧都可以包含校验位或不包含。
0x04	表示数据方向。
	0 表示手机发向 ESP32；
	1 表示 ESP32 发向手机。
0x08	表示是否要求对方回复 ack。
	0 表示不要求；
	1 表示要求回复 ack。
0x10	表示是否有后续的数据分片。
	0 表示此帧没有后续数据分片；
	1 表示还有后续数据分片，用来传输较长的数据。
	如果是 Frag 帧，则告知当前 content 部分 + 后续 content 部分的总长度，位于 Data 域的前 2 字节 (即最大支持 64K 的 content 数据)。
0x10~0x80 保留	

3. Sequence Control

   序列控制域。帧发送时，无论帧的类型是什么，序列 (Sequence) 都会自动加 1，用来防止重放攻击 (Replay Attack)。每次重现连接后，序列清零。

4. Length

   Data 域的长度，不包含 CheckSum。

5. Data

   不同的 Type 或 Subtype，Data 域的含义均不同。请参考上方表格。

6. CheckSum

   此域为 2 byte 的校验，用来校验『序列 + 数据长度 + 明文数据』。

5.28.6 ESP32 端的安全实现

1. 保证数据安全

   为了保证 Wi-Fi SSID 和密码的传输过程是安全的，需要使用对称加密算法（例如 AES、DES 等）对报文进行加密。在使用对称加密算法之前，需要使用非对称加密算法（DH、RSA、ECC 等）协商出

（或生成出）一个共享密钥。

2. 保证数据完整性

   保证数据完整性，需要加入校验算法（例如 SHA1、MD5、CRC 等）。

3. 身份安全（签名）

   某些算法如 RSA 可以保证身份安全。有些算法如 DH，本身不能保证身份安全，需要添加其他算法来签名。

4. 防止重放攻击 (Replay Attack)

   加入帧发送序列（Sequence），并且序列参与数据校验。

   在 ESP32 端的代码中，你可以决定和开发密钥协商等安全处理的流程参考上述流程图）。手机应用向 ESP32 发送协商数据，将传送给应用层处理。如果应用层不处理，可使用 BluFi 提供的 DH 加密算法来磋商密钥。应用层需向 BluFi 注册以下几个与安全相关的函数：

```
typedef void (*esp_blufi_negotiate_data_handler_t)(uint8_t *data, int len, uint8_t␣
↪**output_data, int *output_len, bool *need_free);
```

该函数用来接收协商期间的正常数据 (normal data)，处理完成后，需要将待发送的数据使用 output_data 和 output_len 传出。

BluFi 会在调用完 negotiate_data_handler 后，发送 negotiate_data_handler 传出的 output_data。

这里的两个『*』，因为需要发出去的数据长度未知，所以需要函数自行分配 (malloc) 或者指向全局变量，通过 need_free 通知是否需要释放内存。

```
typedef int (* esp_blufi_encrypt_func_t)(uint8_t iv8, uint8_t *crypt_data, int cyprt_
↪len);
```

加密和解密的数据长度必须一致。其中 iv8 为帧的 8 bit 序列 (sequence)，可作为 iv 的某 8 bit 来使用。

```
typedef int (* esp_blufi_decrypt_func_t)(uint8_t iv8, uint8_t *crypt_data, int crypt_
↪len);
```

加密和解密的数据长度必须一致。其中 iv8 为帧的 8 bit 序列 (sequence)，可作为 iv 的某 8 bit 来使用。

```
typedef uint16_t (*esp_blufi_checksum_func_t)(uint8_t iv8, uint8_t *data, int len);
```

该函数用来计算 CheckSum，返回值为 CheckSum 的值。BluFi 会使用该函数返回值与包末尾的 CheckSum 做比较。

5.28.7 GATT 相关说明

UUID

BluFi Service UUID：0xFFFF，16 bit

BluFi （手机 -> ESP32）特性：0xFF01，主要权限：可写

BluFi （ESP32 -> 手机）特性：0xFF02，主要权限：可读可通知

5.29 片外 RAM

[English]

5.29.1 简介

ESP32 提供了 520 KB 的片上 SRAM，可以满足大部分需求。但有些场景可能需要更多 RAM，因此 ESP32 另外提供了高达 4 MB 的片外 SPI RAM 存储器以供用户使用。片外 RAM 被添加到内存映射中，在某些范围内与片上 RAM 使用方式相同。

5.29.2 硬件

ESP32 支持与 SPI Flash 芯片并联的 SPI PSRAM。ESP32 支持多种类型的 RAM 芯片，但 ESP32 SDK 当前仅支持 ESP-PSRAM32 芯片。

ESP-PSRAM32 芯片的工作电压为 1.8 V，只能与 1.8 V flash 并联使用。请确保在启动时将 MTDI 管脚设置为高电平，或者将 ESP32 中的熔丝设置为始终使用 1.8 V 的 VDD_SIO 电平，否则有可能会损坏 PSRAM 和/或 flash 芯片。

要将 ESP-PSRAM 芯片连接到 ESP32D0W*，请连接以下信号：

- PSRAM /CE (pin 1) > ESP32 GPIO 16
- PSRAM SO (pin 2) > flash DO
- PSRAM SIO[2] (pin 3) > flash WP
- PSRAM SI (pin 5) > flash DI
- PSRAM SCLK (pin 6) > ESP32 GPIO 17
- PSRAM SIO[3] (pin 7) > flash HOLD
- PSRAM Vcc (pin 8) > ESP32 VCC_SDIO

ESP32D2W* 芯片的连接方式有待确定。

注解： 乐鑫同时提供 ESP32-WROVER 模组，内部搭载 ESP32 芯片，集成 1.8 V flash 和 ESP-PSRAM32，可直接用于终端产品 PCB 中。

5.29.3 配置片外 RAM

ESP-IDF 完全支持将外部存储器集成到您的应用程序中。您可以将 ESP-IDF 配置成启动并完成初始化后以多种方式处理片外 RAM：

- *集成片外 RAM 到 ESP32 内存映射*
- *添加片外 RAM 到内存分配程序*
- *调用 malloc() 分配片外 RAM*（默认）
- *允许.bss 段放入片外存储器*

集成片外 RAM 到 ESP32 内存映射

在*CONFIG_SPIRAM_USE* 中选择“Integrate RAM into ESP32 memory map（集成片外 RAM 到 ESP32 内存映射）”选项。

这是集成片外 RAM 最基础的设置选项，大多数用户需要用到其他更高级的选项。

ESP-IDF 启动过程中，片外 RAM 被映射到以 0x3F800000 起始的数据地址空间（字节可寻址），空间大小正好为 RAM 的大小 (4 MB)。

应用程序可以通过创建指向该区域的指针手动将数据放入片外存储器，同时应用程序全权负责管理片外 RAM，包括协调 Buffer 的使用、防止发生损坏等。

添加片外 RAM 到内存分配程序

在*CONFIG_SPIRAM_USE* 中选择“Make RAM allocatable using heap_caps_malloc(…, MALLOC_CAP_SPIRAM)”选项。

启用上述选项后，片外 RAM 被映射到地址 0x3F800000，并将这个区域添加到*内存分配程序* 里携带 `MALLOC_CAP_SPIRAM` 的标志

程序如果想从片外存储器分配存储空间，则需要调用 `heap_caps_malloc(size, MALLOC_CAP_SPIRAM)`，之后可以调用 `free()` 函数释放这部分存储空间。

调用 malloc() 分配片外 RAM

在*CONFIG_SPIRAM_USE* 中选择“Make RAM allocatable using malloc() as well”选项，该选项为默认选项。

启用此选项后，片外存储器将被添加到内存分配程序（与上一选项相同），同时也将被添加到由标准 `malloc()` 返回的 RAM 中。

这允许应用程序使用片外 RAM 而无需重写代码以使用 `heap_caps_malloc(..., MALLOC_CAP_SPIRAM)`。

如果某次内存分配偏向于片外存储器，您也可以使用*CONFIG_SPIRAM_MALLOC_ALWAYSINTERNAL* 设置分配空间的大小阈值，控制分配结果：

- 如果分配的空间小于阈值，分配程序将首先选择内部存储器。
- 如果分配的空间等于或大于阈值，分配程序将首先选择外部存储器。

如果优先考虑的内部或外部存储器中没有可用的存储块，分配程序则会选择其他类型存储。

由于有些 Buffer 仅可在内部存储器中分配，因此需要使用第二个配置项*CONFIG_SPIRAM_MALLOC_RESERVE_INTERNAL* 定义一个内部存储池，仅限显式的内部存储器分配使用（例如用于 DMA 的存储器）。常规 `malloc()` 将不会从该池中分配，但可以使用*MALLOC_CAP_DMA* 和 `MALLOC_CAP_INTERNAL` 旗标从该池中分配存储器。

允许.bss 段放入片外存储器

设置*CONFIG_SPIRAM_ALLOW_BSS_SEG_EXTERNAL_MEMORY* 启用该选项，此选项配置与上面三个选项互不影响。

启用该选项后，从 0x3F800000 起始的地址空间将用于存储来自 lwip、net80211、libpp 和 bluedroid ESP-IDF 库中零初始化的数据（BSS 段）。

`EXT_RAM_ATTR` 宏应用于任何静态声明（未初始化为非零值）之后，可以将附加数据从内部 BSS 段移到片外 RAM。

启用此选项可以减少 BSS 段占用的内部静态存储。

剩余的片外 RAM 也可以通过上述方法添加到内存分配程序中。

5.29.4 片外 RAM 使用限制

使用片外 RAM 有下面一些限制：

- Flash cache 禁用时（比如，正在写入 flash），片外 RAM 将无法访问；同样，对片外 RAM 的读写操作也将导致 cache 访问异常。出于这个原因，ESP-IDF 不会在片外 RAM 中分配任务堆栈（详见下文）。
- 片外 RAM 不能用于储存 DMA 描述符，也不能用作 DMA 读写操作的缓冲区 (Buffer)。与 DMA 搭配使用的 Buffer 必须先使用 `heap_caps_malloc(size, MALLOC_CAP_DMA)` 进行分配，之后可以调用标准 `free()` 回调释放 Buffer。
- 片外 RAM 与片外 flash 使用相同的 cache 区域，即频繁在片外 RAM 访问的变量可以像在片上 RAM 中一样快速读取和修改。但访问大块数据时（大于 32 KB），cache 空间可能会不足，访问速度将回落到片外 RAM 访问速度。此外，访问大块数据可以“挤出”flash cache，可能会降低代码执行速度。

- 片外 RAM 不可用作任务堆栈存储器。因此*xTaskCreate()* 及类似函数将始终为堆栈和任务 TCB 分配片上储存器，而*xTaskCreateStatic()* 类型的函数将检查传递的 Buffer 是否属于片上存储器。但对于不以任何方式直接或间接调用 ROM 中代码的任务，menuconfig 选项*CONFIG_SPIRAM_ALLOW_STACK_EXTERNAL_MEMORY* 将消除 *xTaskCreateStatic* 中的检查，从而允许任务堆栈存储在外部 RAM 中。但是，不建议使用此方法。
- 默认情况下，片外 RAM 初始化失败将终止 ESP-IDF 启动。如果想禁用此功能，可启用*CONFIG_SPIRAM_IGNORE_NOTFOUND* 配置选项。如果启用*CONFIG_SPIRAM_ALLOW_BSS_SEG_EXTERNAL_MEMORY*，*CONFIG_SPIRAM_IGNORE_NOTFOUND* 选项将不能使用，这是因为在链接时，链接器已经向片外 RAM 分配符号。
- 时钟频率为 80 MHz 时，片外 RAM 须占用 HSPI 总线或 VSPI 总线。请使用*CONFIG_SPIRAM_OCCUPY_SPI_HOST* 选择要用的 SPI 主机。

5.29.5 芯片版本

有些 ESP32 芯片版本存在某些已知问题，可能会影响片外 RAM 的使用。请参考 ESP32 勘误表，查看详细信息。为了解决这些问题，ESP-IDF 采取了以下措施：

ESP32 rev v0

ESP-IDF 尚未提供针对此版本硅片 bug 的解决方法，因此在 ESP32 rev v0 中，ESP-IDF 无法将片外 PSRAM 映射到 ESP32 主内存映射中。

ESP32 rev v1

当某些机器指令序列在片外存储器位置上运行时，此芯片版本中的错误可能会引发芯片故障（详情见 ESP32 勘误表 第 3.2 章节）。为了解决这个问题，用于编译 ESP-IDF 项目的 GCC 编译器扩展了一个旗标：-mfix-esp32-psram-cache-issue。在命令行中将此旗标传递给 GCC，编译器对这些序列进行处理，然后仅输出可以安全执行的代码。如需启用此旗标，请选择*CONFIG_SPIRAM_CACHE_WORKAROUND*。

ESP-IDF 还采取了其他措施确保不同时使用 PSRAM 访问和出错指令集：

- 链接到使用 GCC 旗标重新编译的 Newlib 版本；
- 避免使用某些 ROM 函数；
- 为 Wi-Fi 栈分配静态内存。

5.30 链接脚本生成机制

[English]

5.30.1 概述

ESP32 的代码和数据可以存放在多个内存区域。通常，代码和只读数据存放在 flash 区域，可写数据存放在内存中。我们经常需要更改代码或者数据的默认映射区域，例如为了提高性能，将关键部分的代码和只读数据放置到内存中，或者将代码、数据和只读数据存放到 RTC 内存中以便在唤醒桩 和*ULP 协处理器* 中使用。

IDF 的链接脚本生成机制允许用户在组件级别定义代码和数据的存放区域。组件通过链接片段文件 描述如何映射目标文件的输入段（甚至可以是某个具体的函数或者数据）。在构建应用程序时，链接片段文件会被收集、解析并处理，然后扩充到链接脚本模板 中形成最终的链接脚本文件，该链接脚本会被用于链接最终的二进制应用程序。

5.30.2 快速上手

本节将指导如何快速将代码和数据放入 RAM 和 RTC 内存中，并演示如何使这些放置规则依赖于项目的配置。本节内容重在指导快速入门，因此并未在使用前详细介绍所有涉及的术语和概念，但在首次提及此类术语或概念时，均提供了相应链接，以帮助您的理解。

准备工作

Make

在组件目录中新建一个链接片段文件，该文件是一个扩展名为 `.lf` 的文本文件。为了能够让构建系统收集到此片段文件，需要为组件添加一个条目，在调用 `register_component` 之前设置 `COMPONENT_ADD_LDFRAGMENTS` 变量的值，使其指向刚才的链接片段文件。

```
# 文件路径相对于组件的 Makefile
COMPONENT_ADD_LDFRAGMENTS += "path/to/linker_fragment_file.lf" "path/to/another_linker_
↪fragment_file.lf"
```

CMake

对于 CMake 构建系统来说，需要在调用 `register_component` 之前设置 `COMPONENT_ADD_LDFRAGMENTS` 变量的值，使其指向链接片段文件。

```
# 文件路径相对于组件的 CMakeLists.txt
set(COMPONENT_ADD_LDFRAGMENTS "path/to/linker_fragment_file.lf" "path/to/another_linker_
↪fragment_file.lf")

register_component()
```

也可以使用函数 `ldgen_add_fragment_files` 在项目的 CMakeLists.txt 文件或者组件的 project_include.cmake 文件中指定该片段文件

```
ldgen_add_fragment_files(target files ...)
```

指定放置区域

链接脚本生成机制允许指定以下条目的存放位置：

- 组件中的一个或多个目标文件
- 一个或多个函数/变量（使用它们的名字来指定）
- 整个组件库

在继续讲解之前，假设我们的组件包含以下内容：

- 一个名为 `component` 的组件，在构建期间被归档为 `libcomponent.a` 库文件
- 该库中有三个存档的目标文件：`object1.o`，`object2.o` 和 `object3.o`
- `object1.o` 中定义了 `function1` 函数，`object2.o` 中定义了 `function2` 函数
- 在其中的一个 IDF KConfig 文件中存在 `PERFORMANCE_MODE` 和 `PERFORMANCE_LEVEL` 两个配置，相应地，项目的 sdkconfig 文件会通过 `CONFIG_PERFORMANCE_MODE` 和 `CONFIG_PERFORMANCE_LEVEL` 这两个宏来指示当前设置的值

在新建的链接片段文件中输入以下内容：

```
[mapping]
archive: libcomponent.a
entries:
```

这会创建一个空的*mapping 片段*，它并不会执行任何操作。在链接期间，会使用默认的存放规则 来映射 `libcomponent.a`，除非填充了 `entries` 字段。

放置目标文件

假设整个 `object1.o` 目标文件对性能至关重要，所以最好把它放在 RAM 中。另一方面，假设 “object2.o“ 目标文件包含有从深度睡眠唤醒所需的数据，因此需要将它存放到 RTC 内存中。可以在链接片段文件中写入以下内容：

```
[mapping]
archive: libcomponent.a
entries:
    object1 (noflash)      # 将所有代码和只读数据放置在 IRAM 和 DRAM 中
    object2 (rtc)          # 将所有代码、数据和只读数据放置到 RTC 快速内存和 RTC 慢速内存中
```

那么 `object3.o` 放在哪里呢？由于未指定放置规则，它会被存放到默认区域。

放置函数和数据

假设在 object1.o 目标文件中只有 function1 是与性能密切相关，且在 object2.o 目标文件中只有 function2 需要在深度睡眠唤醒后执行。可以在链接片段文件中写入以下内容：

```
[mapping]
archive: libcomponent.a
entries:
    object1:function1 (noflash)
    object2:function2 (rtc)
```

object1.o 和 object2.o 的剩余函数以及整个 object3.o 目标文件会被存放到默认区域。指定数据存放区域的方法很类似，仅需将 : 之后的函数名，替换为变量名即可。

警告： 使用符号名来指定放置区域有一定的局限。因此，您也可以将相关代码和数据集中在源文件中，然后根据使用目标文件的放置规则 进行放置。

放置整个组件

在这个例子中，假设我们需要将整个组件存放到 RAM 中，可以这样写：

```
[mapping]
archive: libcomponent.a
entries:
    * (noflash)
```

类似的，下面的写法可以将整个组件存放到 RTC 内存中：

```
[mapping]
archive: libcomponent.a
entries:
    * (rtc)
```

依赖于具体配置的存放方式

假设只有当 sdkconfig 文件中存在 CONFIG_PERFORMANCE_MODE == y 时，整个组件才会被放置到指定区域，可以这样写：

```
[mapping]
archive: libcomponent.a
```

(下页继续)

(续上页)

```
entries:
    : PERFORMANCE_MODE = y
    * (noflash)
```

其含义可以通过如下伪代码来表述：

```
if PERFORMANCE_MODE = y
    place entire libcomponent.a in RAM
else
    use default placements
```

此外，您还可以设置多个判断条件。假设有如下需求：当 CONFIG_PERFORMANCE_LEVEL == 1 时，只有 object1.o 存放到 RAM 中；当 CONFIG_PERFORMANCE_LEVEL == 2 时，object1.o 和 object2.o 会被存放到 RAM 中；当 CONFIG_PERFORMANCE_LEVEL == 3 时，归档中的所有目标文件都会被存放到 RAM 中；当这三个条件都不满足时，将整个组件库存放到 RTC 内存中。虽然这种使用场景很罕见，不过，还是可以通过以下方式实现：

```
[mapping]
archive: libcomponent.a
entries:
    : PERFORMANCE_LEVEL = 3
    * (noflash)
    : PERFORMANCE_LEVEL = 2
    object1 (noflash)
    object2 (noflash)
    : PERFORMANCE_LEVEL = 1
    object1 (noflash)
    : default
    * (rtc)
```

用伪代码可以表述为：

```
if CONFIG_PERFORMANCE_LEVEL == 3
    place entire libcomponent.a in RAM
else if CONFIG_PERFORMANCE_LEVEL == 2
    only place object1.o and object2.o in RAM
else if CONFIG_PERFORMANCE_LEVEL == 1
    only place object1.o in RAM
else
    place entire libcomponent.a in RTC memory
```

条件测试还支持其他操作。

默认的存放规则

到目前为止，“默认存放规则”一直作为未指定 `rtc` 和 `noflash` 存放规则时的备选放置方式。`noflash` 或者 `rtc` 标记不仅仅是链接脚本生成机制中的关键字，实际上还是由用户指定且被称为*scheme 片段* 的对象。由于这些存放规则非常常用，所以 IDF 中已经预定义了这些规则。

类似地，还有一个名为 `default` 的 scheme 片段，它定义了默认的存放规则，详情请见*默认 scheme*。

注解： 有关使用此功能的 IDF 组件的示例，请参阅 freertos/CMakeLists.txt。为了提高性能，`freertos` 组件通过该机制将所有目标文件中的代码、字面量和只读数据存放到 IRAM 中。

快速入门指南到此结束，下面的文章将进一步详细讨论这个机制，例如它的组件、基本概念、语法、如何集成到构建系统中等等。以下部分有助于创建自定义的映射或者修改默认行为。

5.30.3 组件

链接片段文件

“链接片段文件”包含称为“片段”的对象，每个片段含有多条信息，放在一起时即可形成寻访规则，共同描述目标文件各个段在二进制输出文件中的存放位置。

换言之，处理“链接片段文件”也就是在 GNU LD 的 `SECTIONS` 命令中，创建段的存放规则，并将其放在一个内部 `target` token 中。

下面讨论三种类型的片段。

注解： 片段具有名称属性（mapping 片段除外）并且是全局可见的。片段的命名遵循 C 语言的基本变量命名规则，即区分大小写；必须以字母或者下划线开头；允许非初始字符使用字母、数字和下划线；不能使用空格等特殊字符。此外，每种片段都有自己的独立命名空间，如果多个片段的类型和名称相同，就会引发异常。

I. sections 片段

sections 片段定义了 GCC 编译器输出的目标文件段的列表，可以是默认的段（比如 `.text` 段、`.data` 段），也可以是用户通过 `__attribute__` 关键字自定义的段。

此外，用户还可以在某类段后增加一个 `+`，表示囊括列表中的“所有这类段”和“所有以这类段开头的段”。相较于显式地罗列所有的段，我们更推荐使用这种方式。

语法

```
[sections:name]
entries:
    .section+
    .section
    ...
```

示例

```
# 不推荐的方式
[sections:text]
entries:
    .text
    .text.*
    .literal
    .literal.*

# 推荐的方式，效果与上面等同
[sections:text]
entries:
    .text+              # 即 .text 和 .text.*
    .literal+           # 即 .literal 和 .literal.*
```

II. scheme 片段

scheme 片段定义了为每个 sections 指定的 `target`。

语法

```
[scheme:name]
entries:
    sections -> target
    sections -> target
    ...
```

示例

```
[scheme:noflash]
entries:
    text -> iram0_text          # 名为 text 的 sections 片段下的所有条目均归入 iram0_text
    rodata -> dram0_data        # 名为 rodata 的 sections 片段下的所有条目均归入 dram0_
↪data
```

default scheme

注意，有一个名为 `default` 的 scheme 很特殊，特殊在于 catch-all 存放规则都是从这个 scheme 中的条目生成的。这意味着，如果该 scheme 有一条 `text -> flash_text` 条目，则将为目标 `flash_text` 生成如下的存放规则:

```
*(.literal .literal.* .text .text.*)
```

此后，这些生成的 catch-all 规则将用于未指定映射规则的情况。

注解: `default` scheme 是在 esp32/ld/esp32_fragments.lf 文件中定义的，此外，快速上手指南中提到的内置 `noflash` scheme 片段和 `rtc` scheme 片段也是在这个文件中定义的。

III. mapping 片段

mapping 片段定义了可映射实体（即目标文件、函数名、变量名）对应的 scheme 片段。具体来说，mapping 片段有两种类型的条目，分别为映射条目和条件条目。

注解: mapping 片段没有具体的名称属性，内部会根据归档条目的值构造其名称。

语法

```
[mapping]
archive: archive                # 构建后输出的存档文件的名称 (即 libxxx.a)
entries:
    : condition                 # 条件条目，非默认
    object:symbol (scheme)      # 映射条目，Type I
    object (scheme)             # 映射条目，Type II
    * (scheme)                  # 映射条目，Type III

    # 为了提高可读性，可以适当增加分隔行或注释，非必须

    : default                   # 条件条目，默认
    * (scheme)                  # 映射条目，Type III
```

映射条目

mapping 片段的映射条目共有三种类型，分别为:

`Type I` 同时指定了目标文件名和符号名。其中，符号名可以是函数名或者变量名。

`Type II` 仅指定了目标文件名。

Type III 指定了 *，也就是指定了归档文件中所有目标文件。

接下来，让我们通过展开一个 Type II 映射条目，更好地理解映射条目的含义。最初：

```
object (scheme)
```

接着，让我们根据条目定义，将这个 scheme 片段展开：

```
object (sections -> target,
        sections -> target,
        ...)
```

然后再根据条目定义，将这个 sections 片段展开：

```
object (.section,
        .section,
        ... -> target, # 根据目标文件将这里所列出的所有段放在该目标位置

        .section,
        .section,
        ... -> target, # 同样的方法指定其他段

        ...)           # 直至所有段均已展开
```

有关 Type I 映射条目的局限性

Type I 映射条目可以工作的大前提是编译器必须支持 -ffunction-sections 和 -ffdata-sections 选项。因此，如果用户主动禁用了这两个选项，Type I 映射条目就无法工作。此外，值得注意的是，Type I 映射条目的实现还与输出段有关。因此，有时及时用户在编译时没有选择禁用这两个选项，也有可能无法使用 Type I 映射条目。

例如，当使用 -ffunction-sections 选项时，编译器会给每个函数都输出一个单独的段，根据段名的构造规则，这些段的名称应该类似 .text.{func_name} 或 .literal.{func_name}。然而，对于函数中的字符串文字，情况并非如此，因为它们会使用池化后或者新创建的段名。

当使用 -fdata-sections 选项时，编译器会给每一个全局可见的数据输出一个单独的段，名字类似于 .data.{var_name}、.rodata.{var_name} 或者 .bss.{var_name}。这种情况下，Type I 映射条目可以使用。然而，对于在函数作用域中声明的静态数据，编译器在为其生成段名时会同时使用其变量名和其他信息，因此当涉及在函数作用域中定义的静态数据时就会出现问题。

条件条目

条件条目允许根据具体项目配置生成不同的链接脚本。也就是说，可以根据一些配置表达式的值，选择使用一套不同的映射条目。由于检查配置的过程是通过 tools/kconfig_new/kconfiglib.py 文件中的 eval_string 完成的，因此条件表达式也必须遵循 eval_string 的语法和限制。

在一个 mapping 片段中，跟着一个条件条目后定义的所有映射条目均属于该条件条目，直至下一个条件条

目的出现或者是该 mapping 片段的结束。在检查配置时，编译器将逐条检查这个 mapping 片段中的所有条件条目，直至找到一个满足条件的条件条目（即表达式为 TRUE），然后使用该条件条目下定义的映射条目。另外，尽管每个映射都已包含一个隐式的空映射，但用户还是可以自定义一个默认条件，即所有条件条目均不满足时（即没有表达式为 TRUE）使用的映射条目。

示例

```
[scheme:noflash]
entries:
    text -> iram0_text
    rodata -> dram0_data

[mapping:lwip]
archive: liblwip.a
entries:
    : LWIP_IRAM_OPTIMIZATION = y          # 如果 CONFIG_LWIP_IRAM_OPTIMIZATION 在␣
↪sdkconfig 中被定义为 'y'
    ip4:ip4_route_src_hook (noflash)      # 将 ip4.o:ip4_route_src_hook, ip4.o:ip4_route_
↪src 和
    ip4:ip4_route_src (noflash)           # ip4.o:ip4_route 映射到 noflash scheme
    ip4:ip4_route (noflash)               # 该 scheme 会将他们存放到 RAM 中

    : default                             # 否则不使用特殊的映射规则
```

链接脚本模板

链接脚本模板与其他链接脚本没有本质区别，但带有特定的标记语法，可以指示放置生成的存放规则的位置，是指定存放规则的放置位置的框架。

语法

如需引用一个 target token 下的所有存放规则，请使用以下语法：

```
mapping[target]
```

示例

以下示例是某个链接脚本模板的摘录。该链接脚定义了一个输出段 .iram0.text，里面包含一个引用目标 iram0_text 的标记。

```
.iram0.text :
{
    /* 标记 IRAM 的边界 */
    _iram_text_start = ABSOLUTE(.);
```

(下页继续)

(续上页)

```
    /* 引用 iram0_text */
    mapping[iram0_text]

    _iram_text_end = ABSOLUTE(.);
} > iram0_0_seg
```

下面，让我们更具体一点。假设某个链接脚本生成器收集到了以下片段：

```
[sections:text]
    .text+
    .literal+

[sections:iram]
    .iram1+

[scheme:default]
entries:
    text -> flash_text
    iram -> iram0_text

[scheme:noflash]
entries:
    text -> iram0_text

[mapping:freertos]
archive: libfreertos.a
entries:
    * (noflash)
```

则该脚本生成器生成的链接脚本文件，其摘录应如下所示：

```
.iram0.text :
{
    /* 标记 IRAM 的边界 */
    _iram_text_start = ABSOLUTE(.);

    /* 将链接片段处理生成的存放规则放置在模板标记的位置处 */
    *(.iram1 .iram1.*)
    *libfreertos.a:(.literal .text .literal.* .text.*)
```

(下页继续)

(续上页)

```
    _iram_text_end = ABSOLUTE(.);
} > iram0_0_seg
```

`*libfreertos.a:(.literal .text .literal.* .text.*)`

这是从 `freertos` mapping 片段的 `* (noflash)` 条目中生成的规则。`libfreertos.a` 归档文件下的所有目标文件的 `text` 段会被收集到 `iram0_text` 目标下（假设采用 `noflash` scheme），并放在模板中被 `iram0_text` 标记的地方。

`*(.iram1 .iram1.*)`

这是从 `default` scheme 的 `iram -> iram0_text` 条目生成的规则，因为 `default` scheme 指定了一个 `iram -> iram0_text` 条目，因此生成的规则也将放在被 `iram0_text` 标记的地方。值得注意的是，由于该规则是从 `default` scheme 中生成的，因此在同一目标下收集的所有规则下排在第一位。

5.30.4 与构建系统的集成

链接脚本是在应用程序的构建过程中生成的，此时尚未链接形成最终的二进制文件。实现该机制的工具位于 `$(IDF_PATH)/tools/ldgen` 目录下。

链接脚本模板

目前使用的链接脚本模板是 esp32/ld/esp32.project.ld.in，仅用于应用程序的构建，生成的链接脚本文件将放在同一组件的构建目录下。值得注意的是，修改此链接描述文件模板会触发应用程序的二进制文件的重新链接。

链接片段文件

任何组件都可以将片段文件添加到构建系统中，方法有两种：设置 `COMPONENT_ADD_LDFRAGMENTS` 变量或者使用 `ldgen_add_fragment_files` 函数（仅限 CMake），具体可以参考添加片段文件 小节中的介绍。值得注意的是，修改构建系统中的任何片段文件都会触发应用程序的二进制文件的重新链接。

5.31 lwIP

ESP-IDF uses the open source lwIP lightweight TCP/IP stack. The ESP-IDF version of lwIP (esp-lwip) has some modifications and additions compared to the upstream project.

5.31.1 Supported APIs

ESP-IDF supports the following lwIP TCP/IP stack functions:

- *BSD Sockets API*
- *Netconn API* is enabled but not officially supported for ESP-IDF applications

Adapted APIs

Some common lwIP "app" APIs are supported indirectly by ESP-IDF:

- DHCP Server & Client are supported indirectly via the *ESP-NETIF* functionality
- Simple Network Time Protocol (SNTP) is supported via the lwip/include/apps/sntp/sntp.h lwip/lwip/src/inlude/lwip/apps/sntp.h functions (see also *SNTP Time Synchronization*)
- ICMP Ping is supported using a variation on the lwIP ping API. See *ICMP Echo*.
- NetBIOS lookup is available using the standard lwIP API. protocols/http_server/restful_server has an option to demonstrate using NetBIOS to look up a host on the LAN.
- mDNS uses a different implementation to the lwIP default mDNS (see *mDNS 服务*), but lwIP can look up mDNS hosts using standard APIs such as `gethostbyname()` and the convention `hostname.local`, provided the *CONFIG_LWIP_DNS_SUPPORT_MDNS_QUERIES* setting is enabled.

5.31.2 BSD Sockets API

The BSD Sockets API is a common cross-platform TCP/IP sockets API that originated in the Berkeley Standard Distribution of UNIX but is now standardized in a section of the POSIX specification. BSD Sockets are sometimes called POSIX Sockets or Berkeley Sockets.

As implemented in ESP-IDF, lwIP supports all of the common usages of the BSD Sockets API.

References

A wide range of BSD Sockets reference material is available, including:

- Single UNIX Specification BSD Sockets page
- Berkeley Sockets Wikipedia page

Examples

A number of ESP-IDF examples show how to use the BSD Sockets APIs:

- protocols/sockets/tcp_server
- protocols/sockets/tcp_client
- protocols/sockets/udp_server
- protocols/sockets/udp_client

- protocols/sockets/udp_multicast
- protocols/http_request (Note: this is a simplified example of using a TCP socket to send an HTTP request. The *ESP HTTP Client* is a much better option for sending HTTP requests.)

Supported functions

The following BSD socket API functions are supported. For full details see lwip/lwip/src/include/lwip/sockets.h.

- `socket()`
- `bind()`
- `accept()`
- `shutdown()`
- `getpeername()`
- `getsockopt()` & `setsockopt()` (see *Socket Options*)
- `close()` (via 虚拟文件系统组件)
- `read()`, `readv()`, `write()`, `writev()` (via 虚拟文件系统组件)
- `recv()`, `recvmsg()`, `recvfrom()`
- `send()`, `sendmsg()`, `sendto()`
- `select()` (via 虚拟文件系统组件)
- `poll()` (Note: on ESP-IDF, `poll()` is implemented by calling select internally, so using `select()` directly is recommended if a choice of methods is available.)
- `fcntl()` (see *fcntl*)

Non-standard functions:

- `ioctl()` (see *ioctls*)

注解: Some lwIP application sample code uses prefixed versions of BSD APIs, for example `lwip_socket()` instead of the standard `socket()`. Both forms can be used with ESP-IDF, but using standard names is recommended.

Socket Error Handling

BSD Socket error handling code is very important for robust socket applications. Normally the socket error handling involves the following aspects:

- Detecting the error.

- Geting the error reason code.
- Handle the error according to the reason code.

In lwIP, we have two different scenarios of handling socket errors:

- Socket API returns an error. For more information, see *Socket API Errors*.
- `select(int maxfdp1, fd_set *readset, fd_set *writeset, fd_set *exceptset, struct timeval *timeout)` has exception descriptor indicating that the socket has an error. For more information, see *select() Errors*.

Socket API Errors

The error detection

- We can know that the socket API fails according to its return value.

Get the error reason code

- When socket API fails, the return value doesn't contain the failure reason and the application can get the error reason code by accessing errno. Different values indicate different meanings. For more information, see <*Socket Error Reason Code*>.

Example:

```
int err;
int sockfd;

if (sockfd = socket(AF_INET,SOCK_STREAM,0) < 0) {
    // the error code is obtained from errno
    err = errno;
    return err;
}
```

select() Errors

The error detection

- Socket error when `select()` has exception descriptor

Get the error reason code

- If the `select` indicates that the socket fails, we can't get the error reason code by accessing errno, instead we should call `getsockopt()` to get the failure reason code. Because `select()` has exception descriptor, the error code will not be given to errno.

注解: `getsockopt` function prototype `int getsockopt(int s, int level, int optname, void *optval, socklen_t *optlen)`. Its function is to get the current value of the option of any type, any state socket, and store the result in optval. For example, when you get the error code on a socket, you can get it by `getsockopt(sockfd, SOL_SOCKET, SO_ERROR, &err, &optlen)`.

Example:

```
int err;

if (select(sockfd + 1, NULL, NULL, &exfds, &tval) <= 0) {
    err = errno;
    return err;
} else {
    if (FD_ISSET(sockfd, &exfds)) {
        // select() exception set using getsockopt()
        int optlen = sizeof(int);
        getsockopt(sockfd, SOL_SOCKET, SO_ERROR, &err, &optlen);
        return err;
    }
}
```

Socket Error Reason Code

Below is a list of common error codes. For more detailed error codes, please query newlib/include/sys/errno.h

Error code	Description
ECONNREFUSED	Connection refused
EADDRINUSE	Address already in use
ECONNABORTED	Software caused connection abort
ENETUNREACH	Network is unreachable
ENETDOWN	Network interface is not configured
ETIMEDOUT	Connection timed out
EHOSTDOWN	Host is down
EHOSTUNREACH	Host is unreachable
EINPROGRESS	Connection already in progress
EALREADY	Socket already connected
EDESTADDRREQ	Destination address required
EPROTONOSUPPORT	Unknown protocol

Socket Options

The `getsockopt()` and `setsockopt()` functions allow getting/setting per-socket options.

Not all standard socket options are supported by lwIP in ESP-IDF. The following socket options are supported:

Common options

Used with level argument `SOL_SOCKET`.

- `SO_REUSEADDR` (available if *CONFIG_LWIP_SO_REUSE* is set, behavior can be customized by setting *CONFIG_LWIP_SO_REUSE_RXTOALL*)
- `SO_KEEPALIVE`
- `SO_BROADCAST`
- `SO_ACCEPTCONN`
- `SO_RCVBUF` (available if *CONFIG_LWIP_SO_RCVBUF* is set)
- `SO_SNDTIMEO` / `SO_RCVTIMEO`
- `SO_ERROR` (this option is only used with `select()`, see *Socket Error Handling*)
- `SO_TYPE`
- `SO_NO_CHECK` (for UDP sockets only)

IP options

Used with level argument `IPPROTO_IP`.

- `IP_TOS`
- `IP_TTL`
- `IP_PKTINFO` (available if *CONFIG_LWIP_NETBUF_RECVINFO* is set)

For multicast UDP sockets:

- `IP_MULTICAST_IF`
- `IP_MULTICAST_LOOP`
- `IP_MULTICAST_TTL`
- `IP_ADD_MEMBERSHIP`
- `IP_DROP_MEMBERSHIP`

TCP options

TCP sockets only. Used with level argument `IPPROTO_TCP`.

- `TCP_NODELAY`

Options relating to TCP keepalive probes:

- `TCP_KEEPALIVE` (int value, TCP keepalive period in milliseconds)
- `TCP_KEEPIDLE` (same as `TCP_KEEPALIVE`, but the value is in seconds)
- `TCP_KEEPINTVL` (int value, interval between keepalive probes in seconds)
- `TCP_KEEPCNT` (int value, number of keepalive probes before timing out)

IPv6 options

IPv6 sockets only. Used with level argument `IPPROTO_IPV6`

- `IPV6_CHECKSUM`
- `IPV6_V6ONLY`

For multicast IPv6 UDP sockets:

- `IPV6_JOIN_GROUP` / `IPV6_ADD_MEMBERSHIP`
- `IPV6_LEAVE_GROUP` / `IPV6_DROP_MEMBERSHIP`
- `IPV6_MULTICAST_IF`
- `IPV6_MULTICAST_HOPS`
- `IPV6_MULTICAST_LOOP`

fcntl

The `fcntl()` function is a standard API for manipulating options related to a file descriptor. In ESP-IDF, the 虚拟文件系统组件 layer is used to implement this function.

When the file descriptor is a socket, only the following `fcntl()` values are supported:

- `O_NONBLOCK` to set/clear non-blocking I/O mode. Also supports `O_NDELAY`, which is identical to `O_NONBLOCK`.
- `O_RDONLY`, `O_WRONLY`, `O_RDWR` flags for different read/write modes. These can read via `F_GETFL` only, they cannot be set using `F_SETFL`. A TCP socket will return a different mode depending on whether the connection has been closed at either end or is still open at both ends. UDP sockets always return `O_RDWR`.

ioctls

The `ioctl()` function provides a semi-standard way to access some internal features of the TCP/IP stack. In ESP-IDF, the 虚拟文件系统组件 layer is used to implement this function.

When the file descriptor is a socket, only the following `ioctl()` values are supported:

- `FIONREAD` returns the number of bytes of pending data already received in the socket' s network buffer.
- `FIONBIO` is an alternative way to set/clear non-blocking I/O status for a socket, equivalent to `fcntl(fd, F_SETFL, O_NONBLOCK, ...)`.

5.31.3 Netconn API

lwIP supports two lower level APIs as well as the BSD Sockets API: the Netconn API and the Raw API.

The lwIP Raw API is designed for single threaded devices and is not supported in ESP-IDF.

The Netconn API is used to implement the BSD Sockets API inside lwIP, and it can also be called directly from ESP-IDF apps. This API has lower resource usage than the BSD Sockets API, in particular it can send and receive data without needing to first copy it into internal lwIP buffers.

重要: Espressif does not test the Netconn API in ESP-IDF. As such, this functionality is *enabled but not supported*. Some functionality may only work correctly when used from the BSD Sockets API.

For more information about the Netconn API, consult lwip/lwip/src/include/lwip/api.h and this wiki page which is part of the unofficial lwIP Application Developers Manual.

5.31.4 lwIP FreeRTOS Task

lwIP creates a dedicated TCP/IP FreeRTOS task to handle socket API requests from other tasks.

A number of configuration items are available to modify the task and the queues ("mailboxes") used to send data to/from the TCP/IP task:

- *CONFIG_LWIP_TCPIP_RECVMBOX_SIZE*
- *CONFIG_LWIP_TCPIP_TASK_STACK_SIZE*
- *CONFIG_LWIP_TCPIP_TASK_AFFINITY*

5.31.5 esp-lwip custom modifications

Additions

The following code is added which is not present in the upstream lwIP release:

Thread-safe sockets

It is possible to `close()` a socket from a different thread to the one that created it. The `close()` call will block until any function calls currently using that socket from other tasks have returned.

On demand timers

lwIP IGMP and MLD6 features both initialize a timer in order to trigger timeout events at certain times.

The default lwIP implementation is to have these timers enabled all the time, even if no timeout events are active. This increases CPU usage and power consumption when using automatic light sleep mode. `esp-lwip` default behaviour is to set each timer "on demand" so it is only enabled when an event is pending.

To return to the default lwIP behaviour (always-on timers), disable *CONFIG_LWIP_TIMERS_ONDEMAND*.

Abort TCP connections when IP changes

CONFIG_LWIP_TCP_KEEP_CONNECTION_WHEN_IP_CHANGES is disabled by default. This disables the default lwIP behaviour of keeping TCP connections open if an interface IP changes, in case the interface IP changes back (for example, if an interface connection goes down and comes back up). Enable this option to keep TCP connections open in this case, until they time out normally. This may increase the number of sockets in use if a network interface goes down temporarily.

Additional Socket Options

- Some standard IPV4 and IPV6 multicast socket options are implemented (see *Socket Options*).
- Possible to set IPV6-only UDP and TCP sockets with `IPV6_V6ONLY` socket option (normal lwIP is TCP only).

IP layer features

- IPV4 source based routing implementation is different.
- IPV4 mapped IPV6 addresses are supported.

Limitations

- Calling `send()` or `sendto()` repeatedly on a UDP socket may eventually fail with `errno` equal to `ENOMEM`. This is a limitation of buffer sizes in the lower layer network interface drivers. If all driver transmit buffers are full then UDP transmission will fail. Applications sending a high volume of UDP datagrams who don't wish for any to be dropped by the sender should check for this error code and

re-send the datagram after a short delay. Increasing the number of TX buffers in the *Wi-Fi* or *Ethernet* project configuration (as applicable) may also help.

5.31.6 Performance Optimization

TCP/IP performance is a complex subject, and performance can be optimized towards multiple goals. The default settings of ESP-IDF are tuned for a compromise between throughput, latency, and moderate memory usage.

Maximum throughput

Espressif tests ESP-IDF TCP/IP throughput using the wifi/iperf example in an RF sealed enclosure.

The wifi/iperf/sdkconfig.defaults file for the iperf example contains settings known to maximize TCP/IP throughput, usually at the expense of higher RAM usage. To get maximum TCP/IP throughput in an application at the expense of other factors then suggest applying settings from this file into the project sdkconfig.

重要: Suggest applying changes a few at a time and checking the performance each time with a particular application workload.

- If a lot of tasks are competing for CPU time on the system, consider that the lwIP task has configurable CPU affinity (*CONFIG_LWIP_TCPIP_TASK_AFFINITY*) and runs at fixed priority `ESP_TASK_TCPIP_PRIO` (18). Configure competing tasks to be pinned to a different core, or to run at a lower priority.
- If using `select()` function with socket arguments only, setting *CONFIG_LWIP_USE_ONLY_LWIP_SELECT* will make `select()` calls faster.

If using a Wi-Fi network interface, please also refer to *Wi-Fi Buffer Usage*.

Minimum latency

Except for increasing buffer sizes, most changes which increase throughput will also decrease latency by reducing the amount of CPU time spent in lwIP functions.

- For TCP sockets, lwIP supports setting the standard `TCP_NODELAY` flag to disable Nagle' s algorithm.

Minimum RAM usage

Most lwIP RAM usage is on-demand, as RAM is allocated from the heap as needed. Therefore, changing lwIP settings to reduce RAM usage may not change RAM usage at idle but can change it at peak.

- Reducing *CONFIG_LWIP_MAX_SOCKETS* reduces the maximum number of sockets in the system. This will also cause TCP sockets in the `WAIT_CLOSE` state to be closed and recycled more rapidly (if needed to open a new socket), further reducing peak RAM usage.
- Reducing *CONFIG_LWIP_TCPIP_RECVMBOX_SIZE*, *CONFIG_LWIP_TCP_RECVMBOX_SIZE* and *CONFIG_LWIP_UDP_RECVMBOX_SIZE* reduce memory usage at the expense of throughput, depending on usage.

If using Wi-Fi, please also refer to *Wi-Fi Buffer Usage*.

Peak Buffer Usage

The peak heap memory that lwIP consumes is the **theoretically-maximum memory** that the lwIP driver consumes. Generally, the peak heap memory that lwIP consumes depends on:

- the memory required to create a UDP connection: lwip_udp_conn
- the memory required to create a TCP connection: lwip_tcp_conn
- the number of UDP connections that the application has: lwip_udp_con_num
- the number of TCP connections that the application has: lwip_tcp_con_num
- the TCP TX window size: lwip_tcp_tx_win_size
- the TCP RX window size: lwip_tcp_rx_win_size

So, the peak heap memory that the LwIP consumes can be calculated with the following formula:
lwip_dynamic_peek_memory = (lwip_udp_con_num * lwip_udp_conn) + (lwip_tcp_con_num * (lwip_tcp_tx_win_size + lwip_tcp_rx_win_size + lwip_tcp_conn))

Some TCP-based applications need only one TCP connection. However, they may choose to close this TCP connection and create a new one when an error (such as a sending failure) occurs. This may result in multiple TCP connections existing in the system simultaneously, because it may take a long time for a TCP connection to close, according to the TCP state machine (refer to RFC793).

5.32 工具

5.32.1 IDF Docker Image

IDF Docker image (`espressif/idf`) is intended for building applications and libraries with specific versions of ESP-IDF, when doing automated builds.

The image contains:

- Common utilities such as git, wget, curl, zip.
- Python 3.6 or newer.

- A copy of a specific version of ESP-IDF (see below for information about versions). `IDF_PATH` environment variable is set, and points to ESP-IDF location in the container.
- All the build tools required for the specific version of ESP-IDF: CMake, make, ninja, cross-compiler toolchains, etc.
- All Python packages required by ESP-IDF are installed in a virtual environment.

The image entrypoint sets up `PATH` environment variable to point to the correct version of tools, and activates the Python virtual environment. As a result, the environment is ready to use the ESP-IDF build system.

The image can also be used as a base for custom images, if additional utilities are required.

Tags

Multiple tags of this image are maintained:

- `latest`: tracks `master` branch of ESP-IDF
- `vX.Y`: corresponds to ESP-IDF release `vX.Y`
- `release-vX.Y`: tracks `release/vX.Y` branch of ESP-IDF

注解: Versions of ESP-IDF released before this feature was introduced do not have corresponding Docker image versions. You can check the up-to-date list of available tags at https://hub.docker.com/r/espressif/idf/tags.

Usage

Setting up Docker

Before using the `espressif/idf` Docker image locally, make sure you have Docker installed. Follow the instructions at https://docs.docker.com/install/, if it is not installed yet.

If using the image in CI environment, consult the documentation of your CI service on how to specify the image used for the build process.

Building a project with CMake

In the project directory, run:

```
docker run --rm -v $PWD:/project -w /project espressif/idf idf.py build
```

The above command explained:

- `docker run`: runs a Docker image. It is a shorter form of the command `docker container run`.

- `--rm`: removes the container when the build is finished
- `-v $PWD:/project`: mounts the current directory on the host (`$PWD`) as `/project` directory in the container
- `espressif/idf`: uses Docker image `espressif/idf` with tag `latest` (implicitly added by Docker when no tag is specified)
- `idf.py build`: runs this command inside the container

To build with a specific docker image tag, specify it as `espressif/idf:TAG`:

```
docker run --rm -v $PWD:/project -w /project espressif/idf:v4.0 idf.py build
```

注解: At the time of writing, v4.0 release of ESP-IDF does not exist, yet, so the above command will not work. You can check the up-to-date list of available tags at https://hub.docker.com/r/espressif/idf/tags.

Building a project with GNU Make

Same as for CMake, except that the build command is different:

```
docker run --rm -v $PWD:/project -w /project espressif/idf make defconfig all -j4
```

注解: If the `sdkconfig` file does not exist, the default behavior of GNU Make build system is to open the menuconfig UI. This may be not desired in automated build environments. To ensure that the `sdkconfig` file exists, `defconfig` target is added before `all`.

Using the image interactively

It is also possible to do builds interactively, to debug build issues or test the automated build scripts. Start the container with *-i -t* flags:

```
docker run --rm -v $PWD:/project -w /project -it espressif/idf
```

Then inside the container, use `idf.py` as usual:

```
idf.py menuconfig
idf.py build
```

注解: Commands which communicate with the development board, such as `idf.py flash` and `idf.py monitor` will not work in the container unless the serial port is passed through into the container. However currently this is not possible with Docker for Windows (https://github.com/docker/for-win/issues/1018) and Docker for Mac (https://github.com/docker/for-mac/issues/900).

CHAPTER 6

Libraries and Frameworks

[中文]

6.1 Cloud Frameworks

ESP32 supports multiple cloud frameworks using agents built on top of ESP-IDF. Here are the pointers to various supported cloud frameworks' agents and examples:

6.1.1 AWS IoT

https://github.com/espressif/esp-aws-iot is an open source repository for ESP32 based on Amazon Web Services' aws-iot-device-sdk-embedded-C.

6.1.2 Azure IoT

https://github.com/espressif/esp-azure is an open source repository for ESP32 based on Microsoft Azure' s azure-iot-sdk-c SDK.

6.1.3 Google IoT Core

https://github.com/espressif/esp-google-iot is an open source repository for ESP32 based on Google' s iot-device-sdk-embedded-c SDK.

CHAPTER 7

Contributions Guide

We welcome contributions to the esp-idf project!

7.1 How to Contribute

Contributions to esp-idf - fixing bugs, adding features, adding documentation - are welcome. We accept contributions via Github Pull Requests.

7.2 Before Contributing

Before sending us a Pull Request, please consider this list of points:

- Is the contribution entirely your own work, or already licensed under an Apache License 2.0 compatible Open Source License? If not then we unfortunately cannot accept it.
- Does any new code conform to the esp-idf *Style Guide*?
- Does the code documentation follow requirements in 编写代码文档?
- Is the code adequately commented for people to understand how it is structured?
- Is there documentation or examples that go with code contributions? There are additional suggestions for writing good examples in examples readme.
- Are comments and documentation written in clear English, with no spelling or grammar errors?
- Example contributions are also welcome. Please check the 创建示例项目 guide for these.

- If the contribution contains multiple commits, are they grouped together into logical changes (one major change per pull request)? Are any commits with names like "fixed typo" squashed into previous commits?
- If you' re unsure about any of these points, please open the Pull Request anyhow and then ask us for feedback.

7.3 Pull Request Process

After you open the Pull Request, there will probably be some discussion in the comments field of the request itself.

Once the Pull Request is ready to merge, it will first be merged into our internal git system for in-house automated testing.

If this process passes, it will be merged onto the public github repository.

7.4 Legal Part

Before a contribution can be accepted, you will need to sign our *Contributor Agreement*. You will be prompted for this automatically as part of the Pull Request process.

7.5 Related Documents

7.5.1 Espressif IoT Development Framework Style Guide

About This Guide

Purpose of this style guide is to encourage use of common coding practices within the ESP-IDF.

Style guide is a set of rules which are aimed to help create readable, maintainable, and robust code. By writing code which looks the same way across the code base we help others read and comprehend the code. By using same conventions for spaces and newlines we reduce chances that future changes will produce huge unreadable diffs. By following common patterns for module structure and by using language features consistently we help others understand code behavior.

We try to keep rules simple enough, which means that they can not cover all potential cases. In some cases one has to bend these simple rules to achieve readability, maintainability, or robustness.

When doing modifications to third-party code used in ESP-IDF, follow the way that particular project is written. That will help propose useful changes for merging into upstream project.

C Code Formatting

Indentation

Use 4 spaces for each indentation level. Don't use tabs for indentation. Configure the editor to emit 4 spaces each time you press tab key.

Vertical Space

Place one empty line between functions. Don't begin or end a function with an empty line.

```
void function1()
{
    do_one_thing();
    do_another_thing();
                                // INCORRECT, don't place empty line here
}
                                // place empty line here
void function2()
{
                                // INCORRECT, don't use an empty line here
    int var = 0;
    while (var < SOME_CONSTANT) {
        do_stuff(&var);
    }
}
```

The maximum line length is 120 characters as long as it doesn't seriously affect the readability.

Horizontal Space

Always add single space after conditional and loop keywords:

```
if (condition) {    // correct
    // ...
}

switch (n) {        // correct
    case 0:
        // ...
}
```

(下页继续)

(续上页)

```
for(int i = 0; i < CONST; ++i) {    // INCORRECT
    // ...
}
```

Add single space around binary operators. No space is necessary for unary operators. It is okay to drop space around multiply and divide operators:

```
const int y = y0 + (x - x0) * (y1 - y0) / (x1 - x0);    // correct

const int y = y0 + (x - x0)*(y1 - y0)/(x1 - x0);        // also okay

int y_cur = -y;                                          // correct
++y_cur;

const int y = y0+(x-x0)*(y1-y0)/(x1-x0);                 // INCORRECT
```

No space is necessary around `.` and `->` operators.

Sometimes adding horizontal space within a line can help make code more readable. For example, you can add space to align function arguments:

```
gpio_matrix_in(PIN_CAM_D6,   I2S0I_DATA_IN14_IDX, false);
gpio_matrix_in(PIN_CAM_D7,   I2S0I_DATA_IN15_IDX, false);
gpio_matrix_in(PIN_CAM_HREF, I2S0I_H_ENABLE_IDX,  false);
gpio_matrix_in(PIN_CAM_PCLK, I2S0I_DATA_IN15_IDX, false);
```

Note however that if someone goes to add new line with a longer identifier as first argument (e.g. `PIN_CAM_VSYNC`), it will not fit. So other lines would have to be realigned, adding meaningless changes to the commit.

Therefore, use horizontal alignment sparingly, especially if you expect new lines to be added to the list later.

Never use TAB characters for horizontal alignment.

Never add trailing whitespace at the end of the line.

Braces

- Function definition should have a brace on a separate line:

  ```
  // This is correct:
  void function(int arg)
  ```

(下页继续)

(续上页)

```
{

}

// NOT like this:
void function(int arg) {

}
```

- Within a function, place opening brace on the same line with conditional and loop statements:

```
if (condition) {
    do_one();
} else if (other_condition) {
    do_two();
}
```

Comments

Use `//` for single line comments. For multi-line comments it is okay to use either `//` on each line or a `/* */` block.

Although not directly related to formatting, here are a few notes about using comments effectively.

- Don't use single comments to disable some functionality:

```
void init_something()
{
    setup_dma();
    // load_resources();                // WHY is this thing commented, asks the␣
↪reader?
    start_timer();
}
```

- If some code is no longer required, remove it completely. If you need it you can always look it up in git history of this file. If you disable some call because of temporary reasons, with an intention to restore it in the future, add explanation on the adjacent line:

```
void init_something()
{
    setup_dma();
```

(下页继续)

(续上页)

```
    // TODO: we should load resources here, but loader is not fully integrated yet.
    // load_resources();
    start_timer();
}
```

- Same goes for `#if 0 ... #endif` blocks. Remove code block completely if it is not used. Otherwise, add comment explaining why the block is disabled. Don't use `#if 0 ... #endif` or comments to store code snippets which you may need in the future.
- Don't add trivial comments about authorship and change date. You can always look up who modified any given line using git. E.g. this comment adds clutter to the code without adding any useful information:

```
void init_something()
{
    setup_dma();
    // XXX add 2016-09-01
    init_dma_list();
    fill_dma_item(0);
    // end XXX add
    start_timer();
}
```

Line Endings

Commits should only contain files with LF (Unix style) endings.

Windows users can configure git to check out CRLF (Windows style) endings locally and commit LF endings by setting the `core.autocrlf` setting. *Github has a document about setting this option <github-line-endings>*. However because MSYS2 uses Unix-style line endings, it is often easier to configure your text editor to use LF (Unix style) endings when editing ESP-IDF source files.

If you accidentally have some commits in your branch that add LF endings, you can convert them to Unix by running this command in an MSYS2 or Unix terminal (change directory to the IDF working directory and check the correct branch is currently checked out, beforehand):

```
git rebase --exec 'git diff-tree --no-commit-id --name-only -r HEAD | xargs dos2unix &&␣
↪git commit -a --amend --no-edit --allow-empty' master
```

(Note that this line rebases on master, change the branch name at the end to rebase on another branch.)

For updating a single commit, it's possible to run `dos2unix FILENAME` and then run `git commit --amend`

Formatting Your Code

You can use `astyle` program to format your code according to the above recommendations.

If you are writing a file from scratch, or doing a complete rewrite, feel free to re-format the entire file. If you are changing a small portion of file, don't re-format the code you didn't change. This will help others when they review your changes.

To re-format a file, run:

```
tools/format.sh components/my_component/file.c
```

C++ Code Formatting

The same rules as for C apply. Where they are not enough, apply the following rules.

File Naming

C++ Header files have the extension `.h`. C++ source files have the extension `.cpp`, which is important for the compiler to distiguish them from normal C source files.

Naming

- **Class and struct** names shall be written in `CamelCase` with a capital letter as beginning. Member variables and methods shall be in `snake_case`.
- **Namespaces** shall be in lower `snake_case`.
- **Templates** are specified in the line above the function declaration.

Member Order in Classes

First put the public members, then the protected, then private ones. Omit public, protected or private sections without any members.

Spacing

- Don't indent inside namespaces.
- Put `public`, `protected` and `private` labels at the same indentation level as the corresponding `class` label.

Simple Example

```
// file spaceship.h
#ifndef SPACESHIP_H_
#define SPACESHIP_H_
#include <cstdlib>

namespace spaceships {

class SpaceShip {
public:
    SpaceShip(size_t crew);
    size_t get_crew_size() const;

private:
    const size_t crew;
};

class SpaceShuttle : public SpaceShip {
public:
    SpaceShuttle();
};

class Sojuz : public SpaceShip {
public:
    Sojuz();
};

template <typename T>
class CargoShip {
public:
    CargoShip(const T &cargo);

private:
    T cargo;
};

} // namespace spaceships

#endif // SPACESHIP_H_
```

(下页继续)

(续上页)

```
// file spaceship.cpp
#include "spaceship.h"

namespace spaceships {

// Putting the curly braces in the same line for constructors is OK if it only␣
↪initializes
// values in the initializer list
SpaceShip::SpaceShip(size_t crew) : crew(crew) { }

size_t SpaceShip::get_crew_size() const
{
    return crew;
}

SpaceShuttle::SpaceShuttle() : SpaceShip(7)
{
    // doing further initialization
}

Sojuz::Sojuz() : SpaceShip(3)
{
    // doing further initialization
}

template <typename T>
CargoShip<T>::CargoShip(const T &cargo) : cargo(cargo) { }

} // namespace spaceships
```

CMake Code Style

- Indent with four spaces.
- Maximum line length 120 characters. When splitting lines, try to focus on readability where possible (for example, by pairing up keyword/argument pairs on individual lines).
- Don’ t put anything in the optional parentheses after `endforeach()`, `endif()`, etc.
- Use lowercase (`with_underscores`) for command, function, and macro names.

- For locally scoped variables, use lowercase (`with_underscores`).
- For globally scoped variables, use uppercase (`WITH_UNDERSCORES`).
- Otherwise follow the defaults of the cmake-lint project.

Configuring the Code Style for a Project Using EditorConfig

EditorConfig helps developers define and maintain consistent coding styles between different editors and IDEs. The EditorConfig project consists of a file format for defining coding styles and a collection of text editor plugins that enable editors to read the file format and adhere to defined styles. EditorConfig files are easily readable and they work nicely with version control systems.

For more information, see EditorConfig Website.

Documenting Code

Please see the guide here: 编写代码文档.

Naming

- Any variable or function which is only used in a single source file should be declared `static`.
- Public names (non-static variables and functions) should be namespaced with a per-component or per-unit prefix, to avoid naming collisions. ie `esp_vfs_register()` or `esp_console_run()`. Starting the prefix with `esp_` for Espressif-specific names is optional, but should be consistent with any other names in the same component.
- Static variables should be prefixed with `s_` for easy identification. For example, `static bool s_invert`.
- Avoid unnecessary abbreviations (ie shortening `data` to `dat`), unless the resulting name would otherwise be very long.

Structure

To be written.

Language Features

To be written.

7.5.2 编写代码文档

[English]

本文简要介绍了 espressif/esp-idf 项目库采用的文件风格以及如何在项目库中添加新文件。

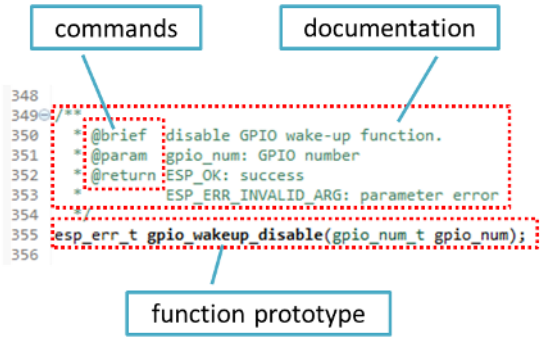
概述

在项目库内编写代码文档时，请遵循 Doxygen 代码注释风格。要采用这一风格，您可以将 `@param` 等特殊命令插入到标准注释块中，比如：

```
/**
 * @param ratio this is oxygen to air ratio
 */
```

Doxygen 会解析代码，提取命令和后续文本，生成代码文档。

注释块通常包含对功能的记述，如下所示。

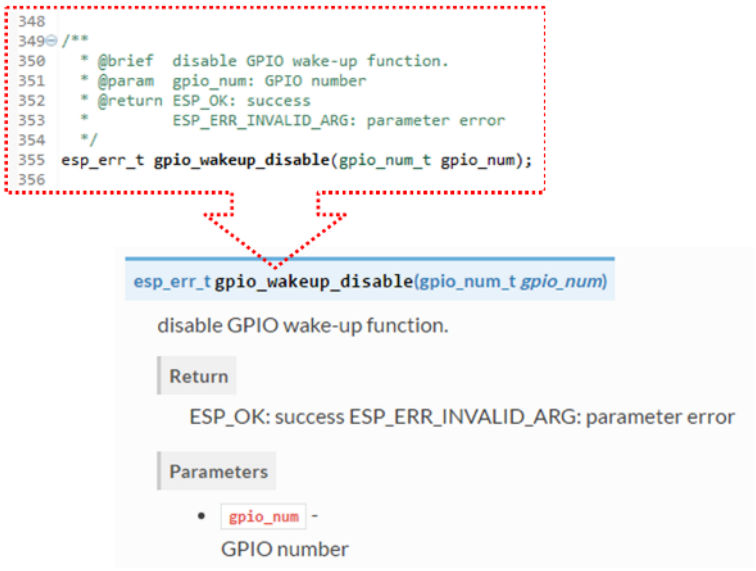


Doxygen 支持多种排版风格，对于文档中可以包含的细节非常灵活。请参考数据丰富、条理清晰的 Doxygen 手册 熟悉 Doxygen 特性。

为什么需要 Doxygen?

使用 Doxygen 的最终目的是确保所有代码编写风格一致，以便在代码变更时使用 Sphinx 和 Breathe 等工具协助筹备、自动更新 API 文档。

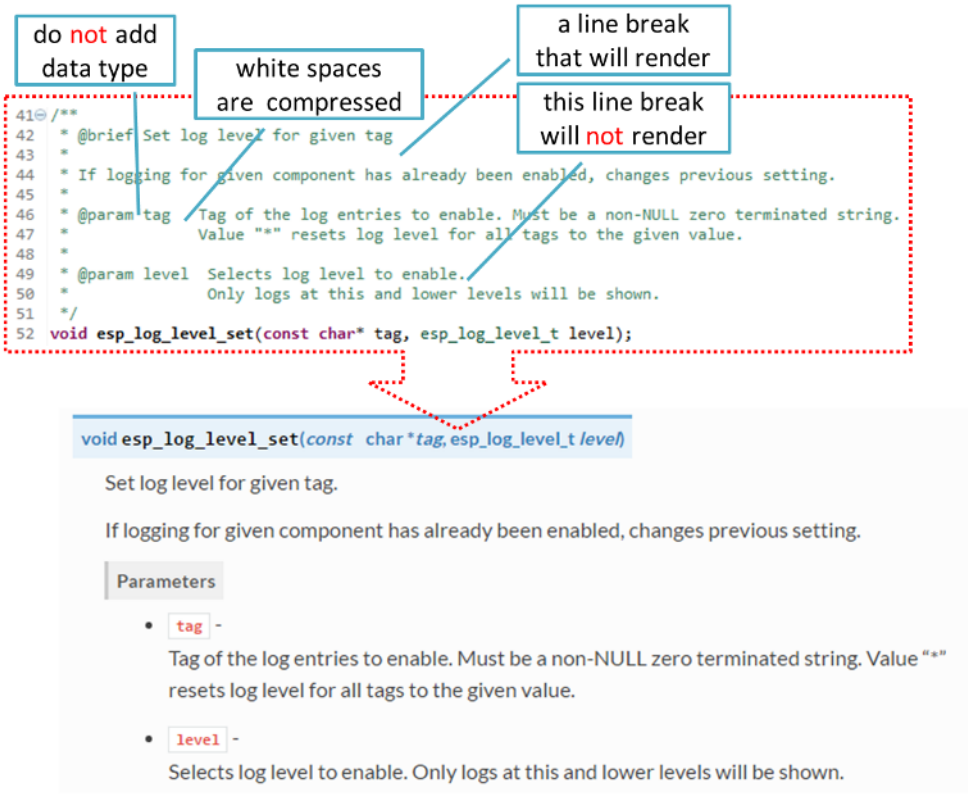
使用这类工具时，上文代码渲染后呈现效果如下：



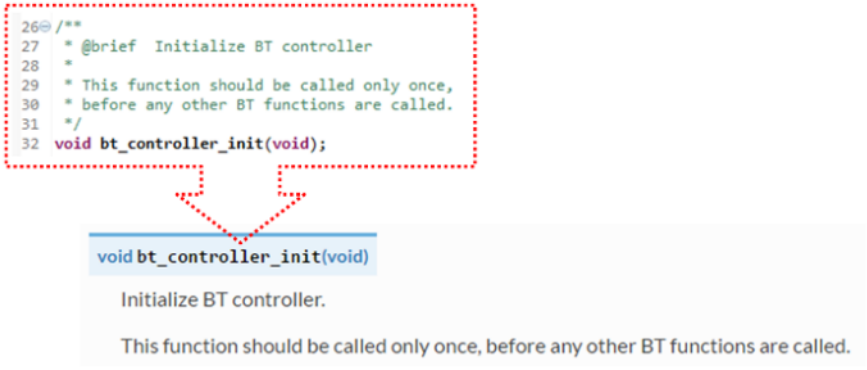
尝试一下！

在本项目库编写代码文档时，请遵守下列准则。

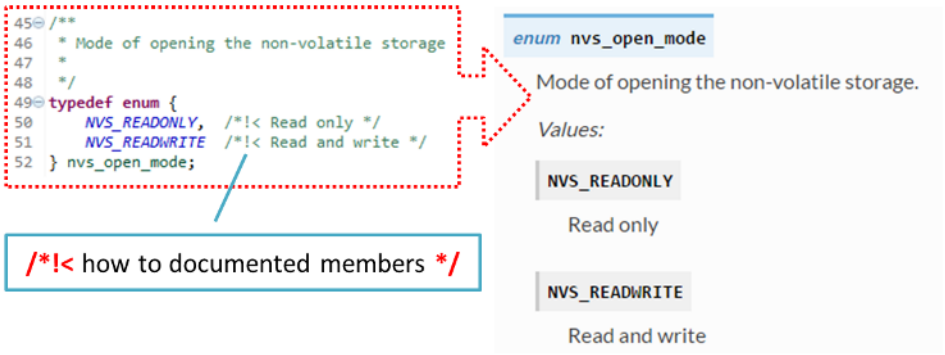
1. 写明代码的基本内容：函数、结构、类型定义、枚举、宏等。请详细说明代码的用途、功能和限制，因为在阅读他人的文档时你也想看到这些信息。
2. 函数文档需简述该函数的功能，并解释输入参数和返回值的含义。
3. 请不要在参数或除空格外的其他字符前面添加数据类型。所有空格和换行符都会压缩为一个空格。如需换行，请执行换行操作两次。



4. 如果函数没有输入参数或返回值，请跳过 `@param` 或 `@return`。



5. 为 `define`、`struct` 和 `enum` 的成员编写文档时，请在每一项后添加注释，如下所示。



6. 请在命令后换行（如下文中的 `@return` ），呈现排版精美的列表。

```
*
* @return
*     - ESP_OK if erase operation was successful
*     - ESP_ERR_NVS_INVALID_HANDLE if handle has been closed or is NULL
*     - ESP_ERR_NVS_READ_ONLY if handle was opened as read only
*     - ESP_ERR_NVS_NOT_FOUND if the requested key doesn't exist
*     - other error codes from the underlying storage driver
*
```

7. 头文件的功能概览和库文件应当存在同一个项目库之下，放入单独的 `README.rst` 文件。如果目录下包含不同 API 的头文件，应将文件命名为 `apiname-readme.rst`。

进阶

以下小贴士可以帮助你进一步提高文档质量，增强可读性。

1. 添加代码片段举例说明。请在片段前后添加 `@code{c}` 和 `@endcode` 命令。

```
*
* @code{c}
* // Example of using nvs_get_i32:
* int32_t max_buffer_size = 4096; // default value
* esp_err_t err = nvs_get_i32(my_handle, "max_buffer_size", &max_buffer_size);
* assert(err == ESP_OK || err == ESP_ERR_NVS_NOT_FOUND);
* // if ESP_ERR_NVS_NOT_FOUND was returned, max_buffer_size will still
* // have its default value.
* @endcode
*
```

   代码片段应放入所介绍功能的注释块中。

2. 使用 `@attention` 或 `@note` 命令高亮显示重要信息。

```
*
* @attention
*     1. This API only impact WIFI_MODE_STA or WIFI_MODE_APSTA mode
*     2. If the ESP32 is connected to an AP, call esp_wifi_disconnect to disconnect.
*
```

上述例子介绍了如何使用编号列表。

3. 给相似的函数编写文档时，可在前后使用 `/**@{*/` 和 `/**@}*/` 标记命令。

```
/**@{*/
/**
 * @brief common description of similar functions
 *
 */
void  first_similar_function (void);
void second_similar_function (void);
/**@}*/
```

示例请参照 nvs_flash/include/nvs.h。

4. 如果想跳过重复的宏定义、枚举项等代码，不添加描述，请在代码前后添加 `/** @cond */` 和 `/** @endcond */` 命令。示例请参照 driver/include/driver/gpio.h。

5. 使用 markdown 增强文档可读性，添加页眉、链接、表格及更多内容。

```
*
* [ESP32 技术参考手册](https://www.espressif.com/sites/default/files/documentation/
↪esp32_technical_reference_manual_cn.pdf)
*
```

注解： 代码片段、注释、链接等内容如没有附在所述对象对应的注释块中，将不会添加到文档中。

6. 准备一个或更多完整的代码示例和描述，将描述放入单独的 `README.md` 文件中，置于 examples 目录的特定文件夹中。

链接到示例

链接到 GitHub 上的示例时，请不要使用绝对 URLs 或硬编码 URLs。请使用 Docutils 自定义角色生成链接。自动生成的链接指向项目库中 git commit 编号（或标记）的 tree 或 blob。这种做法可以确保 master 分支上的文件移动或删除时，链接不会失效。

有如下角色可以选择：

- `:idf:`path`` - 指向 ESP-IDF 内的目录
- `:idf_file:`path`` - 指向 ESP-IDF 内的文件
- `:idf_raw:`path`` - 指向 ESP-IDF 内的原始格式文件
- `:component:`path`` - 指向 ESP-IDF components 内的文件夹
- `:component_file:`path`` - 指向 ESP-IDF components 内的文件
- `:component_raw:`path`` - 指向 ESP-IDF components 内的原始格式文件
- `:example:`path`` - 指向 ESP-IDF examples 内的文件夹
- `:example_file:`path`` - 指向 ESP-IDF examples 内的文件
- `:example_raw:`path`` - 指向 inside ESP-IDF examples 内的原始格式文件

示例:

```
* :example:`get-started/hello_world`
* :example:`Hello World! <get-started/hello_world>`
```

渲染效果:

- get-started/hello_world
- Hello World!

CI build 脚本中添加了检查功能，查找 RST 文件中的硬编码链接（通过 URL 的 tree/master、blob/master 或 raw/master 部分识别）。该功能可通过 `cd docs` 和 `make gh-linkcheck` 命令手动运行。

链接到其他语言文档

要切换不同语言的文档，可使用 `:link_to_translation:` 自定义角色。文档页面中的角色提供其他语言版本的链接。下文的例子说明了如何在文档中添加中英文版本的链接:

```
:link_to_translation:`zh_CN: 中文版 `
:link_to_translation:`en:English`
```

语言用 `en` 或 `zh_CN` 等标准简写表示。最后一个分号后的文本非标准化内容，可根据链接的位置自行输入，如:

```
:link_to_translation:`en:see description in English`
```

添加图例

请考虑使用图表和图片解释表述的概念。

相比于长篇的表述，图例有时可以更好地描述复杂的理念、数据结构或算法。本项目库使用 'blockdiag <http://blockdiag.com/en/index.html>'_ 工具包由简单的文本文件生成图表。

工具包支持下列图表类型：

- 框图
- 时序图
- 活动图
- 逻辑网络图

使用该工具包，可以将简单的文本（与 graphviz 的 DOT 格式类似）转换成美观的图片。图中内容自动排版。图标代码之后会转换为 ".png" 图片，在后台添加进 **Sphinx** 文档中。

要查看图表的渲染效果，可使用线上的 interactive shell 即时显示生成的图片。

下面是一些图表示例：

- 简单的 **框图** / `blockdiag` - Wi-Fi Buffer 配置
- 稍复杂的 **框图** - Wi-Fi 编程模型
- **时序图** / `seqdiag` - 在所有信道中扫描特定 AP
- **包图** / `packetdiag` - NVS 页面结构

尝试修改源代码，看看图表会发生什么变化。

注解： interactive shell 使用的字体和 esp-idf 文档使用的字体略有不同。

添加注释

写文档时，您可能需要：

- 留下建议，说明之后需添加会修改哪些内容。
- 提醒自己或其他人跟进。

这时，您可以使用 `.. todo::` 命令在 reST 文件中添加待做事项。如：

```
.. todo::

   Add a package diagram.
```

如果在 reST 文件中添加 `.. todolist::` 命令，整篇文档中的所有待做事项将会罗列成表。

默认情况下，文档生成器会忽视 `.. todo::` 和 `.. todolist::` 命令。如果您想在本地生成的文档中显示注释和注释列表，请执行下列步骤：

1. 打开本地的 `conf_common.py` 文件。

2. 找到 `todo_include_todos` 参数。
3. 将该参数的值由 `False` 改为 `True`。

将改动推送到远端分支之前，请把 `todo_include_todos` 的值重置为 `False`。

更多关于扩展的信息，请参阅 sphinx.ext.todo 的相关文档。

汇总文档

文档准备好后，请参照*API Documentation Template* 的要求创建一个文件，汇总所有准备好的文档。最后，在文件中添加链接指向 `/docs` 文件夹或子文件夹下 `index.rst` 文件的 `.. toctree::`。

Sphinx 新手怎么办

1. 不要担心。所有需要的软件均有详细文档，并且开源、免费。您可以先查看 Sphinx 文档。如果您不清楚如何用 rst markup 语言写作，请查看 reStructuredText Primer。您也可以使用 markdown (.md) 文件，查找更多在 Recommonmark parser' 文档页面 使用的特定 markdown 句法信息。
2. 查看本文档的源文件，了解本文档使用的代码。源文件存储于 GitHub espressif/esp-idf 项目库的 docs 文件夹下。您可以滑动到页面上方，点击右上角的链接，直接查看本页面的源文件。您也可以通过点击 `Raw` 按键打开源文件，在 GitHub 上查看文件的代码。
3. 想要查看在上传至 GitHub 前文档如何生成、呈现，有两种方式：
   - 安装 'Sphinx <http://www.sphinx-doc.org/>'_、Breathe、Blockdiag 和 Doxygen 本地生成文档，具体可查看下文。
   - 在 Read the Docs 建立账号，在云端生成文档。Read the Docs 免费提供文档生成和存储，且速度快、质量高。
4. 在生成文档前预览，可使用 Sublime Text 编辑器和 OmniMarkupPreviewer 插件。

搭建环境本地生成文档

您可以安装下列包，通过搭建环境在电脑上本地生成文档：

1. Doxygen - https://www.stack.nl/~dimitri/doxygen/
2. Sphinx - https://github.com/sphinx-doc/sphinx/#readme-for-sphinx
3. Breathe - https://github.com/michaeljones/breathe#breathe
4. Document theme "sphinx_rtd_theme" - https://github.com/rtfd/sphinx_rtd_theme
5. Custom 404 page "sphinx-notfound-page" - https://github.com/rtfd/sphinx-notfound-page
6. Blockdiag - http://blockdiag.com/en/index.html
7. Recommonmark - https://github.com/rtfd/recommonmark

添加 “sphinx_rtd_theme” 包之后，文档将与 ESP32 编程指南 的风格保持一致，与 “Read the Docs” 网站的显示效果一样.

不用担心需要安装太多包。除 Doxygen 之外，其他包均使用 Python 语言，可一键安装。

Doxygen 的安装取决于操作系统：

Linux

```
sudo apt-get install doxygen
```

Windows - 在 MSYS2 控制台中安装

```
pacman -S doxygen
```

MacOS

```
brew install doxygen
```

注解： 如果您是在 Windows 系统上安装（Linux 和 MacOS 用户可以跳过此说明），在安装 **之前**，请完成以下两步。这是安装添加图例 提到的 “blockdiag” 依赖项的必须步骤。

1. 更新所有系统包：

   ```
   $ pacman -Syu
   ```

   该过程可能需要重启 MSYS2 MINGW32 控制台重复上述命令，直至更新完成。

2. 安装 *blockdiag* 的依赖项之一 *pillow*：

   ```
   $ pacman -S mingw32/mingw-w64-i686-python-pillow
   ```

   查看屏幕上的记录，确定 `mingw-w64-i686-python-pillow-4.3.0-1` 已安装。旧版本 *pillow* 无法运行。

Windows 安装 Doxygen 的缺点是 *blockdiag pictures <add-illustrations>* 字体不能正确加载，可能会存在乱码。在此问题解决之前，您可以使用 interactive shell 查看完整图片。

其他所有应用都是 Python 包，可以按照下列步骤一键安装：

```
cd ~/esp/esp-idf/docs
pip install --user -r requirements.txt
```

注解： 安装步骤设定将 ESP-IDF 放在 `~/esp/esp-idf` 目录下，这是文档中使用的 ESP-IDF 默认地址。

更换到特定语言文件所在的目录:

```
cd en
```

现在可以调用如下命令生成文档:

```
make html
```

这一步骤需要几分钟时间。完成后，文档会放置在 `~/esp/esp-idf/docs/en/_build/html` 文件夹下。您可以在网页浏览器中打开 `index.html` 查看。

大功告成

我们喜欢可以做酷炫事情的好代码。但我们更喜欢有清晰文档的好代码，可以让读者快速上手，做酷炫的事情。

尝试一下，贡献你的代码和文档!

相关文档

- *API Documentation Template*
- *Documentation Add-ons and Extensions Reference*

7.5.3 Documentation Add-ons and Extensions Reference

This documentation is created using Sphinx application that renders text source files in reStructuredText (`.rst`) format located in docs directory. For some more details on that process, please refer to section 编写代码文档.

Besides Sphinx there are several other applications that help to provide nicely formatted and easy to navigate documentation. These applications are listed in section 搭建环境本地生成文档 with the installed version numbers provided in file docs/requirements.txt.

On top of that we have created a couple of custom add-ons and extensions to help integrate documentation with underlining ESP-IDF repository and further improve navigation as well as maintenance of documentation.

The purpose of this section is to provide a quick reference to the add-ons and the extensions.

Documentation Folder Structure

- The ESP-IDF repository contains a dedicated documentation folder docs in the root.
- The `docs` folder contains localized documentation in docs/en (English) and docs/zh_CN (simplified Chinese) subfolders.

- Graphics files and fonts common to localized documentation are contained in docs/_static subfolder
- Remaining files in the root of `docs` as well as `docs/en` and `docs/zh_CN` provide configuration and scripts used to automate documentation processing including the add-ons and extensions.
- Several folders and files are generated dynamically during documentations build and placed primarily in `docs/[lang]/_build` folders. These folders are temporary and not visible in ESP-IDF repository,

Add-ons and Extensions Reference

docs/conf_common.py This file contains configuration common to each localized documentation (e.g. English, Chinese). The contents of this file is imported to standard Sphinx configuration file `conf.py` located in respective language folders (e.g. `docs/en`, `docs/zh_CN`) during build for each language.

docs/check_doc_warnings.sh If there are any warnings reported during documentation build, then the build is failed. The warnings should be resolved before merging any documentation updates. This script is doing check for warnings in respective log file to fail the build. See also description of `sphinx-known-warnings.txt` below.

docs/check_lang_folder_sync.sh To reduce potential discrepancies when maintaining concurrent language version, the structure and filenames of language folders `docs/en` and `docs/zh_CN` folders should be kept identical. The script `check_lang_folder_sync.sh` is run on each documentation build to verify if this condition is met.

注解: If a new content is provided in e.g. English, and there is no any translation yet, then the corresponding file in `zh_CN` folder should contain an `.. include::` directive pointing to the source file in English. This will automatically include the English version visible to Chinese readers. For example if a file `docs/zh_CN/contribute/documenting-code.rst` does not have a Chinese translation, then it should contain `.. include:: ../../en/contribute/documenting-code.rst` instead.

---

docs/docs_common.mk It contains the common code which is included into the language-specific `Makefiles`. Note that this file contains couple of customizations comparing to what is provided within standard Sphinx installation, e.g. `gh-linkcheck` command has been added.

docs/gen-dxd.py A Python script that generates API reference files based on Doxygen `xml` output. The files have an `inc` extension and are located in `docs/[lang]/_build/inc` directory created dynamically when documentation is build. Please refer to 编写代码文档 and *API Documentation Template*, section **API Reference** for additional details on this process.

docs/gen-toolchain-links.py There couple of places in documentation that provide links to download the toolchain. To provide one source of this information and reduce effort to manually update several files, this script generates toolchain download links and toolchain unpacking code snippets based on information found in tools/toolchain_versions.mk.

docs/gen-version-specific-includes.py Another Python script to automatically generate reStructuredText Text `.inc` snippets with version-based content for this ESP-IDF version.

docs/html_redirects.py During documentation lifetime some source files are moved between folders or renamed. This Python script is adding a mechanism to redirect documentation pages that have changed URL by generating in the Sphinx output static HTML redirect pages. The script is used together with a redirection list `html_redirect_pages` defined in file docs/conf_common.py.

docs/link-roles.py This is an implementation of a custom Sphinx Roles to help linking from documentation to specific files and folders in ESP-IDF. For description of implemented roles please see 链接到示例 and 链接到其他语言文档.

docs/local_util.py A collection of utility functions useful primarily when building documentation locally (see 搭建环境本地生成文档) to reduce the time to generate documentation on a second and subsequent builds. The utility functions check what Doxygen `xml` input files have been changed and copy these files to destination folders, so only the changed files are used during build process.

docs/sphinx-known-warnings.txt There are couple of spurious Sphinx warnings that cannot be resolved without doing update to the Sphinx source code itself. For such specific cases respective warnings are documented in `sphinx-known-warnings.txt` file, that is checked during documentation build, to ignore the spurious warnings.

tools/gen_esp_err_to_name.py This script is traversing the ESP-IDF directory structure looking for error codes and messages in source code header files to generate an `.inc` file to include in documentation under *Error Codes Reference*.

tools/kconfig_new/confgen.py Options to configure ESP-IDF's components are contained in `Kconfig` files located inside directories of individual components, e.g. components/bt/Kconfig. This script is traversing the `component` directories to collect configuration options and generate an `.inc` file to include in documentation under *Configuration Options Reference*.

Related Documents

- 编写代码文档

7.5.4 创建示例项目

[English]

每个 ESP-IDF 的示例都是一个完整的项目，其他人可以将示例复制至本地，并根据实际情况进行一定修改。请注意，示例项目主要是为了展示 ESP-IDF 的功能。

示例项目结构

- `main` 目录需要包含一个名为 `(something)_example_main.c` 的源文件，里面包含示例项目的主要功能。

- 如果该示例项目的子任务比较多，请根据逻辑将其拆分为 `main` 目录下的多个 C 或者 C++ 源文件，并将对应的头文件也放在同一目录下。
- 如果该示例项目具有多种功能，可以考虑在项目中增加一个 `components` 子目录，通过库功能，将示例项目的不同功能划分为不同的组件。注意，如果该组件提供的功能相对完整，且具有一定的通用性，则应该将它们添加到 ESP-IDF 的 `components` 目录中，使其成为 ESP-IDF 的一部分。
- 示例项目需要包含一个 `README.md` 文件，建议使用 示例项目 README 模板 ，并根据项目实际情况进行修改。
- 示例项目需要包含一个 `example_test.py` 文件，用于进行自动化测试。如果在 GitHub 上初次提交 Pull Request 时，可以先不包含这个脚本文件。具体细节，请见有关 Pull Request 的相关内容。

一般准则

示例代码需要遵循《乐鑫物联网开发框架风格指南》。

检查清单

提交一个新的示例项目之前，需要检查以下内容：

- 示例项目的名字（包括 `Makefile` 和 `README.md` 中）应使用 `example`，而不要写 “demo”，“test” 等词汇。
- 每个示例项目只能有一个主要功能。如果某个示例项目有多个主要功能，请将其拆分为两个或更多示例项目。
- 每个示例项目应包含一个 `README.md` 文件，建议使用 示例项目 README 模板 。
- 示例项目中的函数和变量的命令要遵循命名规范 中的要求。对于仅在示例项目源文件中使用的非静态变量/函数，请使用 `example` 或其他类似的前缀。
- 示例项目中的所有代码结构良好，关键代码要有详细注释。
- 示例项目中所有不必要的代码（旧的调试日志，注释掉的代码等）都必须清除掉。
- 示例项目中使用的选项（比如网络名称，地址等）不得直接硬编码，应尽可能地使用配置项，或者定义为宏或常量。
- 配置项可见 `KConfig.projbuild` 文件，该文件中包含一个名为 “Example Configuration” 的菜单。具体情况，请查看现有示例项目。
- 所有的源代码都需要在文件开头指定许可信息（表示该代码是 `in the public domain CC0`）和免责声明。或者，源代码也可以应用 `Apache License 2.0` 许可条款。请查看现有示例项目的许可信息和免责声明，并根据实际情况进行修改。
- 任何第三方代码（无论是直接使用，还是进行了一些改进）均应保留原始代码中的许可信息，且这些代码的许可必须兼容 `Apache License 2.0` 协议。

7.5.5 API Documentation Template

注解: *INSTRUCTIONS*

1. Use this file (docs/api-reference/template.rst) as a template to document API.
2. Change the file name to the name of the header file that represents documented API.
3. Include respective files with descriptions from the API folder using `..include::`
   - README.rst
   - example.rst
   - ...
4. Optionally provide description right in this file.
5. Once done, remove all instructions like this one and any superfluous headers.

Overview

注解: *INSTRUCTIONS*

1. Provide overview where and how this API may be used.
2. Where applicable include code snippets to illustrate functionality of particular functions.
3. To distinguish between sections, use the following heading levels:
   - `#` with overline, for parts
   - `*` with overline, for chapters
   - `=`, for sections
   - `-`, for subsections
   - `^`, for subsubsections
   - `"`, for paragraphs

Application Example

注解: *INSTRUCTIONS*

1. Prepare one or more practical examples to demonstrate functionality of this API.

2. Each example should follow pattern of projects located in `esp-idf/examples/` folder.
3. Place example in this folder complete with `README.md` file.
4. Provide overview of demonstrated functionality in `README.md`.
5. With good overview reader should be able to understand what example does without opening the source code.
6. Depending on complexity of example, break down description of code into parts and provide overview of functionality of each part.
7. Include flow diagram and screenshots of application output if applicable.
8. Finally add in this section synopsis of each example together with link to respective folder in `esp-idf/examples/`.

API Reference

注解: *INSTRUCTIONS*

1. This repository provides for automatic update of API reference documentation using *code markup retrieved by Doxygen from header files*.
2. Update is done on each documentation build by invoking script docs/gen-dxd.py for all header files listed in the `INPUT` statement of docs/Doxyfile.
3. Each line of the `INPUT` statement (other than a comment that begins with `##`) contains a path to header file `*.h` that will be used to generate corresponding `*.inc` files:

```
##
## Wi-Fi - API Reference
##
../components/esp32/include/esp_wifi.h \
../components/esp32/include/esp_smartconfig.h \
```

4. The `*.inc` files contain formatted reference of API members generated automatically on each documentation build. All `*.inc` files are placed in Sphinx `_build` directory. To see directives generated for e.g. `esp_wifi.h`, run `python gen-dxd.py esp32/include/esp_wifi.h`.
5. To show contents of `*.inc` file in documentation, include it as follows:

```
.. include:: /_build/inc/esp_wifi.inc
```

   For example see docs/en/api-reference/wifi/esp_wifi.rst

6. Optionally, rather that using `*.inc` files, you may want to describe API in you own way. See docs/en/api-guides/ulp.rst for example.

   Below is the list of common `.. doxygen...::` directives:

   - Functions - `.. doxygenfunction:: name_of_function`
   - Unions -`.. doxygenunion:: name_of_union`
   - Structures -`.. doxygenstruct:: name_of_structure` together with `:members:`
   - Macros - `.. doxygendefine:: name_of_define`
   - Type Definitions - `.. doxygentypedef:: name_of_type`
   - Enumerations - `.. doxygenenum:: name_of_enumeration`

   See Breathe documentation for additional information.

   To provide a link to header file, use the *link custom role* as follows:

```
* :component_file:`path_to/header_file.h`
```

7. In any case, to generate API reference, the file docs/Doxyfile should be updated with paths to `*.h` headers that are being documented.
8. When changes are committed and documentation is build, check how this section has been rendered. *Correct annotations* in respective header files, if required.

7.5.6 Contributor Agreement

Individual Contributor Non-Exclusive License Agreement

including the Traditional Patent License OPTION

Thank you for your interest in contributing to Espressif IoT Development Framework (esp-idf) (“We” or “Us”).

The purpose of this contributor agreement (“Agreement”) is to clarify and document the rights granted by contributors to Us. To make this document effective, please follow the instructions at CONTRIBUTING.rst

1. DEFINITIONS

“You” means the Individual Copyright owner who submits a Contribution to Us. If You are an employee and submit the Contribution as part of your employment, You have had Your employer approve this Agreement or sign the Entity version of this document.

"Contribution" means any original work of authorship (software and/or documentation) including any modifications or additions to an existing work, Submitted by You to Us, in which You own the Copyright. If You do not own the Copyright in the entire work of authorship, please contact Us at angus@espressif.com.

"Copyright" means all rights protecting works of authorship owned or controlled by You, including copyright, moral and neighboring rights, as appropriate, for the full term of their existence including any extensions by You.

"Material" means the software or documentation made available by Us to third parties. When this Agreement covers more than one software project, the Material means the software or documentation to which the Contribution was Submitted. After You Submit the Contribution, it may be included in the Material.

"Submit" means any form of physical, electronic, or written communication sent to Us, including but not limited to electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, Us, but excluding communication that is conspicuously marked or otherwise designated in writing by You as "Not a Contribution."

"Submission Date" means the date You Submit a Contribution to Us.

"Documentation" means any non-software portion of a Contribution.

2. LICENSE GRANT

2.1 Copyright License to Us

Subject to the terms and conditions of this Agreement, You hereby grant to Us a worldwide, royalty-free, NON-exclusive, perpetual and irrevocable license, with the right to transfer an unlimited number of non-exclusive licenses or to grant sublicenses to third parties, under the Copyright covering the Contribution to use the Contribution by all means, including, but not limited to:

- to publish the Contribution,
- to modify the Contribution, to prepare derivative works based upon or containing the Contribution and to combine the Contribution with other software code,
- to reproduce the Contribution in original or modified form,
- to distribute, to make the Contribution available to the public, display and publicly perform the Contribution in original or modified form.

2.2 Moral Rights remain unaffected to the extent they are recognized and not waivable by applicable law. Notwithstanding, You may add your name in the header of the source code files of Your Contribution and We will respect this attribution when using Your Contribution.

3. PATENTS

3.1 Patent License

Subject to the terms and conditions of this Agreement You hereby grant to us a worldwide, royalty-free, non-exclusive, perpetual and irrevocable (except as stated in Section 3.2) patent license, with the right to transfer an unlimited number of non-exclusive licenses or to grant sublicenses to third parties, to make, have made, use, sell, offer for sale, import and otherwise transfer the Contribution and the Contribution in combination with the Material (and portions of such combination). This license applies to all patents owned or controlled by You, whether already acquired or hereafter acquired, that would be infringed by making, having made, using, selling, offering for sale, importing or otherwise transferring of Your Contribution(s) alone or by combination of Your Contribution(s) with the Material.

3.2 Revocation of Patent License

You reserve the right to revoke the patent license stated in section 3.1 if we make any infringement claim that is targeted at your Contribution and not asserted for a Defensive Purpose. An assertion of claims of the Patents shall be considered for a "Defensive Purpose" if the claims are asserted against an entity that has filed, maintained, threatened, or voluntarily participated in a patent infringement lawsuit against Us or any of Our licensees.

4. DISCLAIMER

THE CONTRIBUTION IS PROVIDED "AS IS". MORE PARTICULARLY, ALL EXPRESS OR IMPLIED WARRANTIES INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT ARE EXPRESSLY DISCLAIMED BY YOU TO US AND BY US TO YOU. TO THE EXTENT THAT ANY SUCH WARRANTIES CANNOT BE DISCLAIMED, SUCH WARRANTY IS LIMITED IN DURATION TO THE MINIMUM PERIOD PERMITTED BY LAW.

5. Consequential Damage Waiver

TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT WILL YOU OR US BE LIABLE FOR ANY LOSS OF PROFITS, LOSS OF ANTICIPATED SAVINGS, LOSS OF DATA, INDIRECT, SPECIAL, INCIDENTAL, CONSEQUENTIAL AND EXEMPLARY DAMAGES ARISING OUT OF THIS AGREEMENT REGARDLESS OF THE LEGAL OR EQUITABLE THEORY (CONTRACT, TORT OR OTHERWISE) UPON WHICH THE CLAIM IS BASED.

6. Approximation of Disclaimer and Damage Waiver

IF THE DISCLAIMER AND DAMAGE WAIVER MENTIONED IN SECTION 4 AND SECTION 5 CANNOT BE GIVEN LEGAL EFFECT UNDER APPLICABLE LOCAL LAW, REVIEWING COURTS SHALL APPLY LOCAL LAW THAT MOST CLOSELY APPROXIMATES AN ABSOLUTE WAIVER OF ALL CIVIL LIABILITY IN CONNECTION WITH THE CONTRIBUTION.

7. Term

7.1 This Agreement shall come into effect upon Your acceptance of the terms and conditions.

7.2 In the event of a termination of this Agreement Sections 4, 5, 6, 7 and 8 shall survive such termination and shall remain in full force thereafter. For the avoidance of doubt, Contributions that are already licensed under a free and open source license at the date of the termination shall remain in full force after the termination of this Agreement.

8. Miscellaneous

8.1 This Agreement and all disputes, claims, actions, suits or other proceedings arising out of this agreement or relating in any way to it shall be governed by the laws of People' s Republic of China excluding its private international law provisions.

8.2 This Agreement sets out the entire agreement between You and Us for Your Contributions to Us and overrides all other agreements or understandings.

8.3 If any provision of this Agreement is found void and unenforceable, such provision will be replaced to the extent possible with a provision that comes closest to the meaning of the original provision and that is enforceable. The terms and conditions set forth in this Agreement shall apply notwithstanding any failure of essential purpose of this Agreement or any limited remedy to the maximum extent possible under law.

8.4 You agree to notify Us of any facts or circumstances of which you become aware that would make this Agreement inaccurate in any respect.

You

Date:	
Name:	
Title:	
Address:	

Us

Date:	
Name:	
Title:	
Address:	

CHAPTER 8

ESP-IDF 版本简介

[English]

ESP-IDF 的 GitHub 仓库时常更新，特别是用于开发新特性的 master 分支。

如有量产需求，请使用稳定版本。

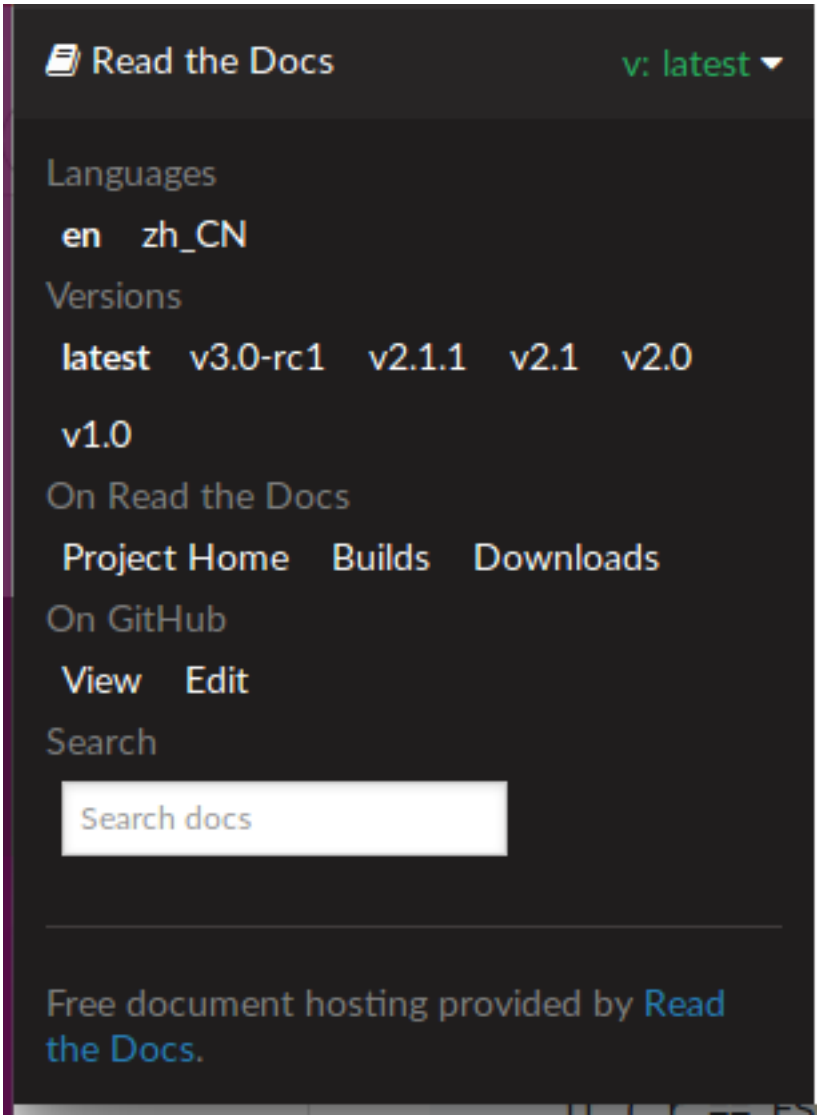
8.1 发布版本

您可以通过以下链接访问各个版本的配套文档：

- 最新稳定版 ESP-IDF：https://docs.espressif.com/projects/esp-idf/zh_CN/stable/
- 最新版 ESP-IDF（即 master 分支）：https://docs.espressif.com/projects/esp-idf/zh_CN/latest/

ESP-IDF 在 GitHub 平台上的完整发布历史请见 发布说明页面。您可以在该页面查看各个版本的发布说明、配套文档及相应获取方式。

此外，您还可以直接前往文档页面，查看部分 ESP-IDF 版本的配套文档，具体可通过点击页面左下角的小三角标志，在不同版本间切换。



8.2 我该选择哪个版本？

- 如有量产需求，请使用 最新稳定版本。稳定版本已通过人工测试，后续更新仅修复 bug，主要特性不受影响（更多详情，请见版本管理）。
- 如需尝试/测试 ESP-IDF 的最新特性，请使用 最新版本（在 master 分支上）。最新版本包含 ESP-IDF 的所有最新特性，已通过自动化测试，但尚未全部完成人工测试（因此存在一定风险）。
- 如需使用稳定版本中没有的新特性，但同时又不希望受到 master 分支更新的影响，您可以将一个最适合您的稳定版本更新至一个预发布版本 或更新至一个发布分支。

有关如何更新 ESP-IDF 本地副本的内容，请参考更新 *ESP-IDF* 章节。

8.3 版本管理

ESP-IDF 采用了 语义版本管理方法，即您可以从字面含义理解每个版本的差异。其中

- 主要版本（例 v3.0）代表有重大更新，包括增加新特性、改变现有特性及移除已弃用的特性。

  升级至一个新的主要版本（例 v2.1 升级至 v3.0）意味着您可能需要更新您的工程代码，并重新测试您的工程，具体可参考 发布说明页面 的重大变更 (Breaking Change) 部分。

- 次要版本（例 v3.1）代表有新增特性和 bug 修复，但现有特性不受影响，公开 API 的使用也不受影响。

  升级至一个新的次要版本（例 v3.0 升级至 v3.1）意味着您可能不需要更新您的工程代码，但需重新测试您的工程，特别是 发布说明页面 中专门提到的部分。

- Bugfix 版本（例 v3.0.1）仅修复 bug，并不增加任何新特性。

  升级至一个新的 Bugfix 版本（例 v3.0 升级至 v3.0.1）意味着您不需要更新您的工程代码，仅需测试与本次发布修复 bug（列表见 发布说明页面）直接相关的特性。

8.4 查看当前版本

查看 ESP-IDF 本地副本的版本，请使用 idf.py 命令:

```
idf.py --version
```

此外，由于 ESP-IDF 的版本也已编译至固件中，因此您也可以使用宏 IDF_VER 查看 ESP-IDF 的版本（以字符串的格式）。ESP-IDF 默认引导程序可以在设备启动时打印 ESP-IDF 的版本，但注意代码中的版本信息仅会在源代码重新编译时才会更新，因此打印出来的版本可能并不是最新的。

几个 ESP-IDF 版本的例子:

版本字符串	含义
v3.2-dev-306-gbeb3611ca	master 分支上的预发布版本。 - v3.2-dev：为 v3.2 进行的开发。 - 306：v3.2 开发启动后的 commit 数量。 - beb3611ca：commit 标识符。
v3.0.2	稳定版本，标签为 v3.0.2。
v3.1-beta1-75-g346d6b0ea	v3.1 的 beta 测试版本（可参考更新至一个发布分支）。 - v3.1-beta1 - 预发布标签。 - 75：添加预发布 beta 标签后的 commit 数量。 - 346d6b0ea：commit 标识符。
v3.0.1-dirty	稳定版本，标签为 v3.0.1。 - dirty 代表 ESP-IDF 的本地副本有修改。

8.5 Git 工作流

乐鑫 ESP-IDF 团队的 (Git) 开发工作流程如下：

- 新的改动总是在 master 分支（最新版本）上进行。master 分支上的 ESP-IDF 版本总带有 -dev 标签，表示“正在开发中”，例 v3.1-dev。
- 这些改动将首先在乐鑫的内部 Git 仓库进行代码审阅与测试，而后在自动化测试完成后推至 GitHub。
- 新版本一旦完成特性开发（在 master 分支上进行）并达到进入 beta 测试的标准，则将该版本签出至一个新分支 (例 release/v3.1)。此外，该分支还打上预发布标签 (例 v3.1-beta1)。您可以在 GitHub 平台上查看 ESP-IDF 的完整 分支列表 和 标签列表。Beta 预发布版本可能仍存在大量“已知问题” (Known Issue)。
- 随着对 beta 版本的不断测试，bug 修复将同时增加至该发布分支和 master 分支。而且，master 分支可能也已经开始为下个版本开发新特性了。
- 当测试快结束时，该发布分支上将增加一个 rc 标签，代表候选发布 (Release Candidate) ，例 v3.1-rc1。此时，该分支仍属于预发布版本。

- 如果一直未发现或报告重大 bug，则该预发布版本将最终增加“主要版本”（例 v4.0）或“次要版本”标记（例 v3.1），成为正式发布版本，并体现在 发布说明页面 。
- 后续，该版本中发现的 bug 都将在该发布分支上进行修复。人工测试完成后，该分支将增加一个 bugfix 版本标签（例 v3.1.1），并体现在 发布说明页面 。

8.6 更新 ESP-IDF

请根据您的实际情况，对 ESP-IDF 进行更新。

- 如有量产用途，建议参考更新至一个稳定发布版本。
- 如需测试/研发/尝试最新特性，建议参考更新至 *master* 分支 。
- 两者折衷建议参考更新至一个发布分支。

注解： 在参考本指南时，请首先获得 ESP-IDF 的本地副本，具体步骤请参考入门指南 中的介绍。

8.6.1 更新至一个稳定发布版本

（推荐量产用户）如需更新至一个新的 ESP-IDF 发布版本，请参考以下步骤：

- 请定期查看 发布说明页面 ，了解最新发布情况。
- 如有新发布的 bugfix 版本（例 v3.0.1 或 v3.0.2）时，请将新的 bugfix 版本更新至您的 ESP-IDF 目录：

```
cd $IDF_PATH
git fetch
git checkout vX.Y.Z
git submodule update --init --recursive
```

- 如有主要版本或次要版本新发布时，请查看发布说明中的具体描述，并决定是否升级您的版本。具体命令与上方描述一致。

注解： 如果您之前在安装 ESP-IDF 时使用了 zip 文件包，而非通过 Git 命令，则您将无法使用 Git 命令进行版本升级，此属正常情况。这种情况下，请重新下载最新 zip 文件包，并替换掉之前 IDF_PATH 下的全部内容。

8.6.2 更新至一个预发布版本

您也可以将您的本地副本签出（命令 `git checkout`）至一个预发布版本或 rc 版本，具体方法请参考更新至一个稳定发布版本 中的描述。

预发布版本通常不体现在 发布说明页面。更多详情，请查看完整 标签列表。使用预发布版本的注意事项，请参考更新至一个发布分支 中的描述。

8.6.3 更新至 master 分支

注解： ESP-IDF 中 master 分支上的代码会时时更新，因此使用 master 分支相当在 “流血的边缘试探”，存在一定风险。

如需使用 ESP-IDF 的 master 分支，请参考以下步骤：

- 本地签出至 master 分支:

```
cd $IDF_PATH
git checkout master
git pull
git submodule update --init --recursive
```

- 此外，您还应在后续工作中不时使用 `git pull` 命令，将远端 master 上的更新同步到本地。注意，在更新 master 分支后，您可能需要更改项代码，也可能遇到新的 bug。
- 如需从 master 分支切换至一个发布分支或稳定版本，请使用 `git checkout` 命令。

重要： 强烈建议您定期使用 `git pull` 和 `git submodule update --init --recursive` 命令，确保本地副本的及时更新。旧的 master 分支相当于一个 “快照”，可能存在未记录的问题，且无法获得支持。对于半稳定版本，请参考更新至一个发布分支。

8.6.4 更新至一个发布分支

从稳定性来说，使用 “发布分支” 相当于在使用 master 分支和稳定版本之间进行折衷，包含一些 master 分支上的新特性，但也同时保证可通过 beta 测试且基本完成了 bug 修复。

更多详情，请前往 GitHub 查看完整 标签列表。

举例，您可以关注 ESP-IDF v3.1 分支，随时关注该分支上的 bugfix 版本发布（例 `v3.1.1` 等）：

```
cd $IDF_PATH
git fetch
git checkout release/v3.1
git pull
git submodule update --init --recursive
```

您每次在该分支上使用 `git pull` 时都相当于把最新的 bugfix 版本发布更新至您的本地副本中。

注解: 发布分支并不会有专门的配套文档，建议您使用与本分支最接近的版本。

CHAPTER 9

资源

[English]

- 您可以在 ESP32 论坛 中提出您的问题，访问社区资源。
- 您可以通过 GitHub 的 Issues 版块提交 bug 或功能请求。在提交新 Issue 之前，请先查看现有的 Issues。
- 您可以在 ESP32 IoT Solution 库中找到基于 ESP-IDF 的 解决方案、应用实例、组件和驱动 等内容。
- 通过 Arduino 平台开发应用，请参考 ESP32 Wi-Fi 芯片的 Arduino 内核。
- 关于 ESP32 的书籍列表，请查看 乐鑫 网站。
- 如果您有兴趣参与到 ESP-IDF 的开发，请查阅*Contributions Guide*。
- 关于 ESP32 的其它信息，请查看官网 文档 版块。
- 关于本文档的 PDF 和 HTML 格式下载（最新版本和早期版本），请点击 下载。

CHAPTER 10

Copyrights and Licenses

10.1 Software Copyrights

All original source code in this repository is Copyright (C) 2015-2019 Espressif Systems. This source code is licensed under the Apache License 2.0 as described in the file LICENSE.

Additional third party copyrighted code is included under the following licenses.

Where source code headers specify Copyright & License information, this information takes precedence over the summaries made here.

10.1.1 Firmware Components

These third party libraries can be included into the application (firmware) produced by ESP-IDF.

- Newlib is licensed under the BSD License and is Copyright of various parties, as described in COPYING.NEWLIB.
- Xtensa header files are Copyright (C) 2013 Tensilica Inc and are licensed under the MIT License as reproduced in the individual header files.
- Original parts of FreeRTOS (components/freertos) are Copyright (C) 2015 Real Time Engineers Ltd and is licensed under the GNU General Public License V2 with the FreeRTOS Linking Exception, as described in license.txt.
- Original parts of LWIP (components/lwip) are Copyright (C) 2001, 2002 Swedish Institute of Computer Science and are licensed under the BSD License as described in COPYING file.

- wpa_supplicant Copyright (c) 2003-2005 Jouni Malinen and licensed under the BSD license.
- FreeBSD net80211 Copyright (c) 2004-2008 Sam Leffler, Errno Consulting and licensed under the BSD license.
- JSMN JSON Parser (components/jsmn) Copyright (c) 2010 Serge A. Zaitsev and licensed under the MIT license.
- argtable3 argument parsing library Copyright (C) 1998-2001,2003-2011,2013 Stewart Heitmann and licensed under 3-clause BSD license.
- linenoise line editing library Copyright (c) 2010-2014 Salvatore Sanfilippo, Copyright (c) 2010-2013 Pieter Noordhuis, licensed under 2-clause BSD license.
- libcoap COAP library Copyright (c) 2010-2017 Olaf Bergmann and others, is licensed under 2-clause BSD license.
- libexpat XML parsing library Copyright (c) 1998-2000 Thai Open Source Software Center Ltd and Clark Cooper, Copyright (c) 2001-2017 Expat maintainers, is licensed under MIT license.
- FatFS library, Copyright (C) 2017 ChaN, is licensed under a BSD-style license .
- cJSON library, Copyright (c) 2009-2017 Dave Gamble and cJSON contributors, is licensed under MIT license.
- libsodium library, Copyright (c) 2013-2018 Frank Denis, is licensed under ISC license.
- micro-ecc library, Copyright (c) 2014 Kenneth MacKay, is licensed under 2-clause BSD license.
- nghttp2 library, Copyright (c) 2012, 2014, 2015, 2016 Tatsuhiro Tsujikawa, Copyright (c) 2012, 2014, 2015, 2016 nghttp2 contributors, is licensed under MIT license.
- Mbed TLS library, Copyright (C) 2006-2018 ARM Limited, is licensed under Apache License 2.0.
- SPIFFS library, Copyright (c) 2013-2017 Peter Andersson, is licensed under MIT license.
- TinyCBOR library, Copyright (c) 2017 Intel Corporation, is licensed under MIT License.
- SD/MMC driver is derived from OpenBSD SD/MMC driver, Copyright (c) 2006 Uwe Stuehler, and is licensed under BSD license.
- Asio , Copyright (c) 2003-2018 Christopher M. Kohlhoff is licensed under the Boost Software License.
- ESP-MQTT MQTT Package (contiki-mqtt) - Copyright (c) 2014, Stephen Robinson, MQTT-ESP - Tuan PM <tuanpm at live dot com> is licensed under Apache License 2.0.
- BLE Mesh is adapted from Zephyr Project, Copyright (c) 2017-2018 Intel Corporation and licensed under Apache License 2.0
- mynewt-nimble Apache Mynewt NimBLE, Copyright 2015-2018, The Apache Software Foundation, is licensed under Apache License 2.0.

10.1.2 Build Tools

This is the list of licenses for tools included in this repository, which are used to build applications. The tools do not become part of the application (firmware), so their license does not affect licensing of the application.

- esptool.py is Copyright (C) 2014-2016 Fredrik Ahlberg, Angus Gratton and is licensed under the GNU General Public License v2, as described in LICENSE file.
- KConfig is Copyright (C) 2002 Roman Zippel and others, and is licensed under the GNU General Public License V2.
- Kconfiglib is Copyright (C) 2011-2019, Ulf Magnusson, and is licensed under the ISC License.
- Menuconfig of Kconfiglib is Copyright (C) 2018-2019, Nordic Semiconductor ASA and Ulf Magnusson, and is licensed under the ISC License.

10.2 ROM Source Code Copyrights

ESP32 mask ROM hardware includes binaries compiled from portions of the following third party software:

- Newlib , licensed under the BSD License and is Copyright of various parties, as described in COPYING.NEWLIB.
- Xtensa libhal, Copyright (c) Tensilica Inc and licensed under the MIT license (see below).
- TinyBasic Plus, Copyright Mike Field & Scott Lawrence and licensed under the MIT license (see below).
- miniz, by Rich Geldreich - placed into the public domain.
- wpa_supplicant Copyright (c) 2003-2005 Jouni Malinen and licensed under the BSD license.
- TJpgDec Copyright (C) 2011, ChaN, all right reserved. See below for license.

10.3 Xtensa libhal MIT License

Copyright (c) 2003, 2006, 2010 Tensilica Inc.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

10.4 TinyBasic Plus MIT License

Copyright (c) 2012-2013

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

10.5 TJpgDec License

TJpgDec - Tiny JPEG Decompressor R0.01 (C)ChaN, 2011 The TJpgDec is a generic JPEG decompressor module for tiny embedded systems. This is a free software that opened for education, research and commercial developments under license policy of following terms.

Copyright (C) 2011, ChaN, all right reserved.

- The TJpgDec module is a free software and there is NO WARRANTY.
- No restriction on use. You can use, modify and redistribute it for personal, non-profit or commercial products UNDER YOUR RESPONSIBILITY.
- Redistributions of source code must retain the above copyright notice.

CHAPTER 11

关于本指南

[English]

本指南为 乐鑫 公司 ESP32 系列芯片 官方应用开发框架 ESP-IDF 的配套文档。

ESP32 芯片是一款 2.4 GHz Wi-Fi 和蓝牙双模芯片，内置 1 或 2 个 32 位处理器，运算能力最高可达 600 DMIPS。

ESP-IDF 即乐鑫物联网开发框架，可为在 Windows、Linux 和 macOS 系统平台上开发 ESP32 应用程序提供工具链、API、组件和工作流的支持。

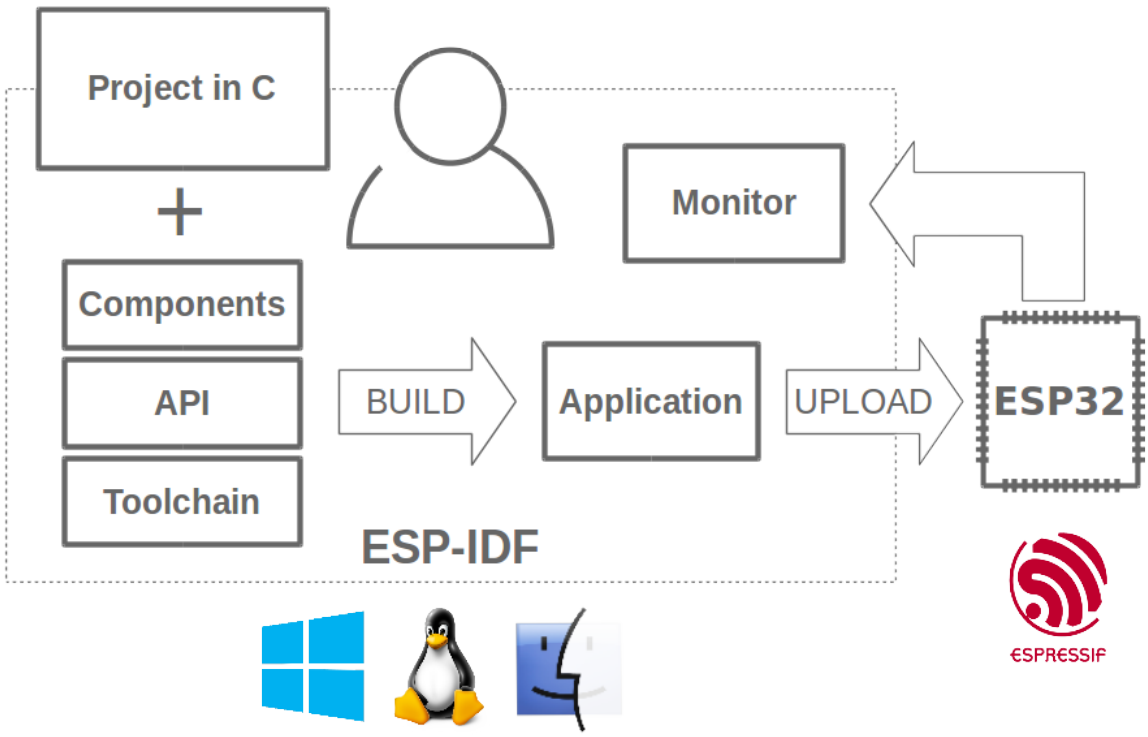


图 1: 乐鑫物联网综合开发框架

CHAPTER 12

Switch Between Languages/切换语言

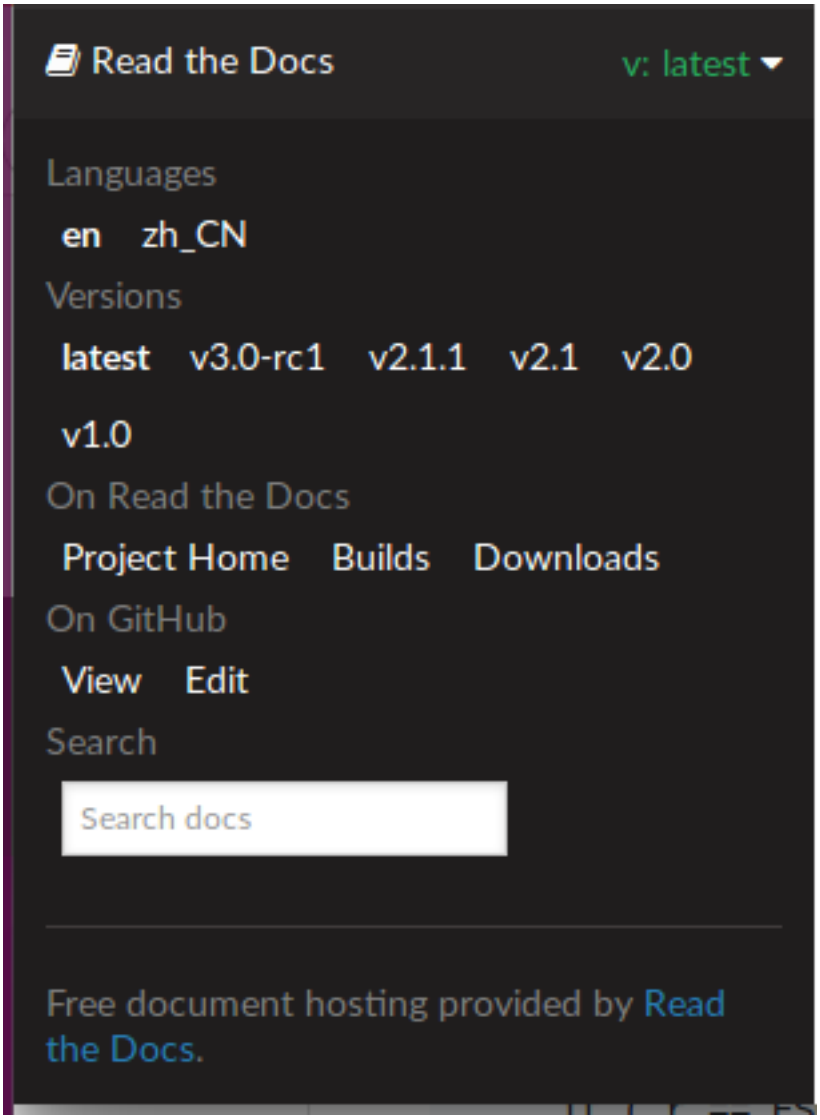
The ESP-IDF Programming Manual is now available in two languages. Please refer to the English version if there is any discrepancy.

《ESP-IDF 编程手册》部分文档现在有两种语言的版本。如有出入请以英文版本为准。

- English/英文
- Chinese/中文

You can easily change from one language to another by the panel on the sidebar like below. Just click on the **Read the Docs** title button on the left-bottom corner if it is folded.

如下图所示，你可使用边栏的面板进行语言的切换。如果该面板被折叠，点击左下角 **Read the Docs** 标题按钮来显示它。


Read the Docs
v: latest
Languages
en zh_CN
Versions
latest v3.0-rc1 v2.1.1 v2.1 v2.0
v1.0
On Read the Docs
Project Home Builds Downloads
On GitHub
View Edit
Search
Search docs
Free document hosting provided by Read the Docs.

- genindex

索引

符号

__anonymous0 (*C++ 类型*), 151
__anonymous8 (*C++ 类型*), 438

A

ADC1_CHANNEL_0 (*C++ 枚举子*), 817
ADC1_CHANNEL_0_GPIO_NUM (*C 宏*), 823
ADC1_CHANNEL_1 (*C++ 枚举子*), 817
ADC1_CHANNEL_1_GPIO_NUM (*C 宏*), 823
ADC1_CHANNEL_2 (*C++ 枚举子*), 817
ADC1_CHANNEL_2_GPIO_NUM (*C 宏*), 823
ADC1_CHANNEL_3 (*C++ 枚举子*), 817
ADC1_CHANNEL_3_GPIO_NUM (*C 宏*), 823
ADC1_CHANNEL_4 (*C++ 枚举子*), 817
ADC1_CHANNEL_4_GPIO_NUM (*C 宏*), 824
ADC1_CHANNEL_5 (*C++ 枚举子*), 817
ADC1_CHANNEL_5_GPIO_NUM (*C 宏*), 824
ADC1_CHANNEL_6 (*C++ 枚举子*), 817
ADC1_CHANNEL_6_GPIO_NUM (*C 宏*), 824
ADC1_CHANNEL_7 (*C++ 枚举子*), 817
ADC1_CHANNEL_7_GPIO_NUM (*C 宏*), 824
adc1_channel_t (*C++ 类型*), 817
adc1_config_channel_atten (*C++ 函数*), 811
adc1_config_width (*C++ 函数*), 811
adc1_get_raw (*C++ 函数*), 812
ADC1_GPIO32_CHANNEL (*C 宏*), 824
ADC1_GPIO33_CHANNEL (*C 宏*), 824
ADC1_GPIO34_CHANNEL (*C 宏*), 824
ADC1_GPIO35_CHANNEL (*C 宏*), 824
ADC1_GPIO36_CHANNEL (*C 宏*), 823
ADC1_GPIO37_CHANNEL (*C 宏*), 823
ADC1_GPIO38_CHANNEL (*C 宏*), 823
ADC1_GPIO39_CHANNEL (*C 宏*), 823
adc1_pad_get_io_num (*C++ 函数*), 810
adc1_ulp_enable (*C++ 函数*), 814
ADC2_CHANNEL_0 (*C++ 枚举子*), 817
ADC2_CHANNEL_0_GPIO_NUM (*C 宏*), 824
ADC2_CHANNEL_1 (*C++ 枚举子*), 817
ADC2_CHANNEL_1_GPIO_NUM (*C 宏*), 824
ADC2_CHANNEL_2 (*C++ 枚举子*), 817
ADC2_CHANNEL_2_GPIO_NUM (*C 宏*), 824
ADC2_CHANNEL_3 (*C++ 枚举子*), 817
ADC2_CHANNEL_3_GPIO_NUM (*C 宏*), 824
ADC2_CHANNEL_4 (*C++ 枚举子*), 817
ADC2_CHANNEL_4_GPIO_NUM (*C 宏*), 824
ADC2_CHANNEL_5 (*C++ 枚举子*), 818
ADC2_CHANNEL_5_GPIO_NUM (*C 宏*), 824
ADC2_CHANNEL_6 (*C++ 枚举子*), 818
ADC2_CHANNEL_6_GPIO_NUM (*C 宏*), 824
ADC2_CHANNEL_7 (*C++ 枚举子*), 818
ADC2_CHANNEL_7_GPIO_NUM (*C 宏*), 824
ADC2_CHANNEL_8 (*C++ 枚举子*), 818
ADC2_CHANNEL_8_GPIO_NUM (*C 宏*), 824
ADC2_CHANNEL_9 (*C++ 枚举子*), 818
ADC2_CHANNEL_9_GPIO_NUM (*C 宏*), 824
ADC2_CHANNEL_MAX (*C++ 枚举子*), 818
adc2_channel_t (*C++ 类型*), 817
adc2_config_channel_atten (*C++ 函数*), 815
adc2_get_raw (*C++ 函数*), 815
ADC2_GPIO0_CHANNEL (*C 宏*), 824

ADC2_GPIO12_CHANNEL (*C 宏*), 824
ADC2_GPIO13_CHANNEL (*C 宏*), 824
ADC2_GPIO14_CHANNEL (*C 宏*), 824
ADC2_GPIO15_CHANNEL (*C 宏*), 824
ADC2_GPIO25_CHANNEL (*C 宏*), 824
ADC2_GPIO26_CHANNEL (*C 宏*), 824
ADC2_GPIO27_CHANNEL (*C 宏*), 824
ADC2_GPIO2_CHANNEL (*C 宏*), 824
ADC2_GPIO4_CHANNEL (*C 宏*), 824
adc2_pad_get_io_num (*C++ 函数*), 815
adc2_vref_to_gpio (*C++ 函数*), 816
ADC_ATTEN_0db (*C 宏*), 816
ADC_ATTEN_11db (*C 宏*), 816
ADC_ATTEN_2_5db (*C 宏*), 816
ADC_ATTEN_6db (*C 宏*), 816
ADC_ATTEN_DB_0 (*C++ 枚举子*), 819
ADC_ATTEN_DB_11 (*C++ 枚举子*), 819
ADC_ATTEN_DB_2_5 (*C++ 枚举子*), 819
ADC_ATTEN_DB_6 (*C++ 枚举子*), 819
ADC_ATTEN_MAX (*C++ 枚举子*), 819
adc_atten_t (*C++ 类型*), 819
adc_bits_width_t (*C++ 类型*), 820
ADC_CHANNEL_0 (*C++ 枚举子*), 819
ADC_CHANNEL_1 (*C++ 枚举子*), 819
ADC_CHANNEL_2 (*C++ 枚举子*), 819
ADC_CHANNEL_3 (*C++ 枚举子*), 819
ADC_CHANNEL_4 (*C++ 枚举子*), 819
ADC_CHANNEL_5 (*C++ 枚举子*), 819
ADC_CHANNEL_6 (*C++ 枚举子*), 819
ADC_CHANNEL_7 (*C++ 枚举子*), 819
ADC_CHANNEL_8 (*C++ 枚举子*), 819
ADC_CHANNEL_9 (*C++ 枚举子*), 819
ADC_CHANNEL_MAX (*C++ 枚举子*), 819
adc_channel_t (*C++ 类型*), 819
ADC_ENCODE_11BIT (*C++ 枚举子*), 818
ADC_ENCODE_12BIT (*C++ 枚举子*), 818
ADC_ENCODE_MAX (*C++ 枚举子*), 818
adc_gpio_init (*C++ 函数*), 813
ADC_I2S_DATA_SRC_ADC (*C++ 枚举子*), 820
ADC_I2S_DATA_SRC_IO_SIG (*C++ 枚举子*), 820
ADC_I2S_DATA_SRC_MAX (*C++ 枚举子*), 820
adc_i2s_encode_t (*C++ 类型*), 818
adc_i2s_mode_init (*C++ 函数*), 814
adc_i2s_source_t (*C++ 类型*), 820
adc_power_off (*C++ 函数*), 813
adc_power_on (*C++ 函数*), 813
adc_set_clk_div (*C++ 函数*), 813
adc_set_data_inv (*C++ 函数*), 813
adc_set_data_width (*C++ 函数*), 811
adc_set_i2s_data_source (*C++ 函数*), 814
ADC_UNIT_1 (*C++ 枚举子*), 818
ADC_UNIT_2 (*C++ 枚举子*), 818
ADC_UNIT_ALTER (*C++ 枚举子*), 818
ADC_UNIT_BOTH (*C++ 枚举子*), 818
ADC_UNIT_MAX (*C++ 枚举子*), 818
adc_unit_t (*C++ 类型*), 818
ADC_WIDTH_10Bit (*C 宏*), 817
ADC_WIDTH_11Bit (*C 宏*), 817
ADC_WIDTH_12Bit (*C 宏*), 817
ADC_WIDTH_9Bit (*C 宏*), 816
ADC_WIDTH_BIT_10 (*C++ 枚举子*), 820
ADC_WIDTH_BIT_11 (*C++ 枚举子*), 820
ADC_WIDTH_BIT_12 (*C++ 枚举子*), 820
ADC_WIDTH_BIT_9 (*C++ 枚举子*), 820
ADC_WIDTH_MAX (*C++ 枚举子*), 820
ADD_DEV_FLUSHABLE_DEV_FLAG (*C 宏*), 416
ADD_DEV_RM_AFTER_PROV_FLAG (*C 宏*), 415
ADD_DEV_START_PROV_NOW_FLAG (*C 宏*), 415
ADV_CHNL_37 (*C++ 枚举子*), 194
ADV_CHNL_38 (*C++ 枚举子*), 194
ADV_CHNL_39 (*C++ 枚举子*), 194
ADV_CHNL_ALL (*C++ 枚举子*), 194
ADV_FILTER_ALLOW_SCAN_ANY_CON_ANY (*C++ 枚举子*), 194
ADV_FILTER_ALLOW_SCAN_ANY_CON_WLST (*C++ 枚举子*), 194
ADV_FILTER_ALLOW_SCAN_WLST_CON_ANY (*C++ 枚举子*), 194
ADV_FILTER_ALLOW_SCAN_WLST_CON_WLST (*C++ 枚举子*), 194
ADV_TYPE_DIRECT_IND_HIGH (*C++ 枚举子*), 194
ADV_TYPE_DIRECT_IND_LOW (*C++ 枚举子*), 194
ADV_TYPE_IND (*C++ 枚举子*), 194
ADV_TYPE_NONCONN_IND (*C++ 枚举子*), 194

ADV_TYPE_SCAN_IND (C++ 枚举子), 194

B

BD_ADDR (C++ 类型), 430
BD_ADDR_LEN (C 宏), 415
BLE_ADDR_TYPE_PUBLIC (C++ 枚举子), 158
BLE_ADDR_TYPE_RANDOM (C++ 枚举子), 158
BLE_ADDR_TYPE_RPA_PUBLIC (C++ 枚举子), 158
BLE_ADDR_TYPE_RPA_RANDOM (C++ 枚举子), 158
BLE_BIT (C 宏), 190
BLE_HCI_UART_H4_ACL (C 宏), 374
BLE_HCI_UART_H4_CMD (C 宏), 374
BLE_HCI_UART_H4_EVT (C 宏), 374
BLE_HCI_UART_H4_NONE (C 宏), 374
BLE_HCI_UART_H4_SCO (C 宏), 374
BLE_SCAN_DUPLICATE_DISABLE (C++ 枚举子), 196
BLE_SCAN_DUPLICATE_ENABLE (C++ 枚举子), 196
BLE_SCAN_DUPLICATE_MAX (C++ 枚举子), 196
BLE_SCAN_FILTER_ALLOW_ALL (C++ 枚举子), 195
BLE_SCAN_FILTER_ALLOW_ONLY_WLST (C++ 枚举子), 195
BLE_SCAN_FILTER_ALLOW_UND_RPA_DIR (C++ 枚举子), 196
BLE_SCAN_FILTER_ALLOW_WLIST_PRA_DIR (C++ 枚举子), 196
BLE_SCAN_TYPE_ACTIVE (C++ 枚举子), 195
BLE_SCAN_TYPE_PASSIVE (C++ 枚举子), 195
BLE_UUID128_VAL_LENGTH (C 宏), 1282
BLE_WL_ADDR_TYPE_PUBLIC (C++ 枚举子), 158
BLE_WL_ADDR_TYPE_RANDOM (C++ 枚举子), 158
BT_CONTROLLER_INIT_CONFIG_DEFAULT (C 宏), 151
BT_OCTET32 (C++ 类型), 430
BT_OCTET32_LEN (C 宏), 415
BTC_HF_CALL_END_TIMEOUT (C 宏), 329
BTC_HF_INVALID_IDX (C 宏), 329
BTC_HF_SECURITY (C 宏), 329
BTC_HF_SERVICE_NAMES (C 宏), 329
BTC_HF_SERVICES (C 宏), 329
BTC_HFAG_SERVICE_NAME (C 宏), 329
BTC_HSAG_SERVICE_NAME (C 宏), 329

C

can_clear_receive_queue (C++ 函数), 844
can_clear_transmit_queue (C++ 函数), 844
can_driver_install (C++ 函数), 840
can_driver_uninstall (C++ 函数), 841
CAN_ERR_PASS_THRESH (C 宏), 839
CAN_EXTD_ID_MASK (C 宏), 839
can_filter_config_t (C++ 类), 839
can_filter_config_t::acceptance_code (C++ 成员), 839
can_filter_config_t::acceptance_mask (C++ 成员), 839
can_filter_config_t::single_filter (C++ 成员), 839
CAN_FRAME_EXTD_ID_LEN_BYTES (C 宏), 839
CAN_FRAME_MAX_DLC (C 宏), 839
CAN_FRAME_STD_ID_LEN_BYTES (C 宏), 839
can_general_config_t (C++ 类), 845
can_general_config_t::alerts_enabled (C++ 成员), 845
can_general_config_t::bus_off_io (C++ 成员), 845
can_general_config_t::clkout_divider (C++ 成员), 845
can_general_config_t::clkout_io (C++ 成员), 845
can_general_config_t::mode (C++ 成员), 845
can_general_config_t::rx_io (C++ 成员), 845
can_general_config_t::rx_queue_len (C++ 成员), 845
can_general_config_t::tx_io (C++ 成员), 845
can_general_config_t::tx_queue_len (C++ 成员), 845
can_get_status_info (C++ 函数), 844
can_initiate_recovery (C++ 函数), 843
CAN_IO_UNUSED (C 宏), 846
can_message_t (C++ 类), 837
can_message_t::data (C++ 成员), 838
can_message_t::data_length_code (C++ 成员), 838
can_message_t::dlc_non_comp (C++ 成员), 838

can_message_t::extd (C++ 成员), 838
can_message_t::flags (C++ 成员), 838
can_message_t::identifier (C++ 成员), 838
can_message_t::reserved (C++ 成员), 838
can_message_t::rtr (C++ 成员), 838
can_message_t::self (C++ 成员), 838
can_message_t::ss (C++ 成员), 838
CAN_MODE_LISTEN_ONLY (C++ 枚举子), 840
CAN_MODE_NO_ACK (C++ 枚举子), 840
CAN_MODE_NORMAL (C++ 枚举子), 840
can_mode_t (C++ 类型), 840
can_read_alerts (C++ 函数), 843
can_receive (C++ 函数), 842
can_reconfigure_alerts (C++ 函数), 843
can_start (C++ 函数), 841
CAN_STATE_BUS_OFF (C++ 枚举子), 846
CAN_STATE_RECOVERING (C++ 枚举子), 846
CAN_STATE_RUNNING (C++ 枚举子), 846
CAN_STATE_STOPPED (C++ 枚举子), 846
can_state_t (C++ 类型), 846
can_status_info_t (C++ 类), 845
can_status_info_t::arb_lost_count (C++ 成员), 846
can_status_info_t::bus_error_count (C++ 成员), 846
can_status_info_t::msgs_to_rx (C++ 成员), 845
can_status_info_t::msgs_to_tx (C++ 成员), 845
can_status_info_t::rx_error_counter (C++ 成员), 846
can_status_info_t::rx_missed_count (C++ 成员), 846
can_status_info_t::state (C++ 成员), 845
can_status_info_t::tx_error_counter (C++ 成员), 846
can_status_info_t::tx_failed_count (C++ 成员), 846
CAN_STD_ID_MASK (C 宏), 839
can_stop (C++ 函数), 841
can_timing_config_t (C++ 类), 838
can_timing_config_t::brp (C++ 成员), 838
can_timing_config_t::sjw (C++ 成员), 839
can_timing_config_t::triple_sampling (C++ 成员), 839
can_timing_config_t::tseg_1 (C++ 成员), 838
can_timing_config_t::tseg_2 (C++ 成员), 838
can_transmit (C++ 函数), 841
CHIP_ESP32 (C++ 枚举子), 1687
CHIP_ESP32S2BETA (C++ 枚举子), 1687
CHIP_FEATURE_BLE (C 宏), 1685
CHIP_FEATURE_BT (C 宏), 1685
CHIP_FEATURE_EMB_FLASH (C 宏), 1685
CHIP_FEATURE_WIFI_BGN (C 宏), 1685
CONFIG_EFUSE_CUSTOM_TABLE, 1428
CONFIG_EFUSE_MAX_BLK_LEN, 1429
CONFIG_EFUSE_VIRTUAL, 1434
CONFIG_ESPTOOLPY_FLASHSIZE, 1307
CONFIG_FEATURE_WPA3_SAE_BIT (C 宏), 676
CONFIG_HEAP_TRACING_STACK_DEPTH (C 宏), 1648
CONFIG_LOG_DEFAULT_LEVEL, 1670, 1671
CONFIG_LWIP_SNTP_UPDATE_DELAY, 1733
CONFIG_LWIP_USE_ONLY_LWIP_SELECT, 1380
CONFIG_SPIRAM_BANKSWITCH_ENABLE, 1655
CONFIG_SPIRAM_BANKSWITCH_RESERVE, 1655

D

DAC_CHANNEL_1 (C++ 枚举子), 850
DAC_CHANNEL_1_GPIO_NUM (C 宏), 849
DAC_CHANNEL_2 (C++ 枚举子), 850
DAC_CHANNEL_2_GPIO_NUM (C 宏), 849
DAC_CHANNEL_MAX (C++ 枚举子), 850
dac_channel_t (C++ 类型), 850
dac_cw_config_t (C++ 类), 850
dac_cw_config_t::en_ch (C++ 成员), 850
dac_cw_config_t::freq (C++ 成员), 850
dac_cw_config_t::offset (C++ 成员), 850
dac_cw_config_t::phase (C++ 成员), 850
dac_cw_config_t::scale (C++ 成员), 850
dac_cw_generator_config (C++ 函数), 849
dac_cw_generator_disable (C++ 函数), 849
dac_cw_generator_enable (C++ 函数), 849
DAC_CW_PHASE_0 (C++ 枚举子), 851
DAC_CW_PHASE_180 (C++ 枚举子), 851
dac_cw_phase_t (C++ 类型), 851
DAC_CW_SCALE_1 (C++ 枚举子), 850

DAC_CW_SCALE_2 (C++ 枚举子), 850
DAC_CW_SCALE_4 (C++ 枚举子), 851
DAC_CW_SCALE_8 (C++ 枚举子), 851
dac_cw_scale_t (C++ 类型), 850
DAC_GPIO25_CHANNEL (C 宏), 849
DAC_GPIO26_CHANNEL (C 宏), 849
dac_i2s_disable (C++ 函数), 848
dac_i2s_enable (C++ 函数), 848
dac_output_disable (C++ 函数), 848
dac_output_enable (C++ 函数), 848
dac_output_voltage (C++ 函数), 848
dac_pad_get_io_num (C++ 函数), 847
DEFAULT_HTTP_BUF_SIZE (C 宏), 1166
DEL_DEV_ADDR_FLAG (C 宏), 416
DEL_DEV_UUID_FLAG (C 宏), 416

E

eAbortSleep (C++ 枚举子), 1505
eBlocked (C++ 枚举子), 1504
eDeleted (C++ 枚举子), 1504
EFUSE_CODE_SCHEME_SELECTOR, 1431
eIncrement (C++ 枚举子), 1504
eNoAction (C++ 枚举子), 1504
eNoTasksWaitingTimeout (C++ 枚举子), 1505
eNotifyAction (C++ 类型), 1504
eReady (C++ 枚举子), 1504
eRunning (C++ 枚举子), 1504
eSetBits (C++ 枚举子), 1504
eSetValueWithoutOverwrite (C++ 枚举子), 1504
eSetValueWithOverwrite (C++ 枚举子), 1504
eSleepModeStatus (C++ 类型), 1505
ESP_A2D_AUDIO_CFG_EVT (C++ 枚举子), 296
ESP_A2D_AUDIO_STATE_EVT (C++ 枚举子), 296
ESP_A2D_AUDIO_STATE_REMOTE_SUSPEND (C++ 枚举子), 295
ESP_A2D_AUDIO_STATE_STARTED (C++ 枚举子), 295
ESP_A2D_AUDIO_STATE_STOPPED (C++ 枚举子), 295
esp_a2d_audio_state_t (C++ 类型), 295
esp_a2d_cb_event_t (C++ 类型), 295
esp_a2d_cb_param_t (C++ 类型), 291
esp_a2d_cb_param_t::a2d_audio_cfg_param (C++ 类), 291
esp_a2d_cb_param_t::a2d_audio_cfg_param::mcc (C++ 成员), 292
esp_a2d_cb_param_t::a2d_audio_cfg_param::remote_bda (C++ 成员), 292
esp_a2d_cb_param_t::a2d_audio_stat_param (C++ 类), 292
esp_a2d_cb_param_t::a2d_audio_stat_param::remote_bda (C++ 成员), 292
esp_a2d_cb_param_t::a2d_audio_stat_param::state (C++ 成员), 292
esp_a2d_cb_param_t::a2d_conn_stat_param (C++ 类), 292
esp_a2d_cb_param_t::a2d_conn_stat_param::disc_rsn (C++ 成员), 292
esp_a2d_cb_param_t::a2d_conn_stat_param::remote_bda (C++ 成员), 292
esp_a2d_cb_param_t::a2d_conn_stat_param::state (C++ 成员), 292
esp_a2d_cb_param_t::audio_cfg (C++ 成员), 291
esp_a2d_cb_param_t::audio_stat (C++ 成员), 291
esp_a2d_cb_param_t::conn_stat (C++ 成员), 291
esp_a2d_cb_param_t::media_ctrl_stat (C++ 成员), 291
esp_a2d_cb_param_t::media_ctrl_stat_param (C++ 类), 292
esp_a2d_cb_param_t::media_ctrl_stat_param::cmd (C++ 成员), 292
esp_a2d_cb_param_t::media_ctrl_stat_param::status (C++ 成员), 292
esp_a2d_cb_t (C++ 类型), 293
ESP_A2D_CIE_LEN_ATRAC (C 宏), 293
ESP_A2D_CIE_LEN_M12 (C 宏), 293
ESP_A2D_CIE_LEN_M24 (C 宏), 293
ESP_A2D_CIE_LEN_SBC (C 宏), 293
ESP_A2D_CONNECTION_STATE_CONNECTED (C++ 枚举子), 294
ESP_A2D_CONNECTION_STATE_CONNECTING (C++ 枚举子), 294
ESP_A2D_CONNECTION_STATE_DISCONNECTED (C++ 枚举子), 294
ESP_A2D_CONNECTION_STATE_DISCONNECTING (C++

枚举子), 294
ESP_A2D_CONNECTION_STATE_EVT (C++ 枚举子), 296
esp_a2d_connection_state_t (C++ 类型), 294
ESP_A2D_DISC_RSN_ABNORMAL (C++ 枚举子), 295
ESP_A2D_DISC_RSN_NORMAL (C++ 枚举子), 294
esp_a2d_disc_rsn_t (C++ 类型), 294
esp_a2d_mcc_t (C++ 类), 293
esp_a2d_mcc_t::cie (C++ 成员), 293
esp_a2d_mcc_t::type (C++ 成员), 293
ESP_A2D_MCT_ATRAC (C 宏), 293
ESP_A2D_MCT_M12 (C 宏), 293
ESP_A2D_MCT_M24 (C 宏), 293
ESP_A2D_MCT_NON_A2DP (C 宏), 293
ESP_A2D_MCT_SBC (C 宏), 293
esp_a2d_mct_t (C++ 类型), 293
esp_a2d_media_ctrl (C++ 函数), 289
ESP_A2D_MEDIA_CTRL_ACK_BUSY (C++ 枚举子), 295
ESP_A2D_MEDIA_CTRL_ACK_EVT (C++ 枚举子), 296
ESP_A2D_MEDIA_CTRL_ACK_FAILURE (C++ 枚举子), 295
ESP_A2D_MEDIA_CTRL_ACK_SUCCESS (C++ 枚举子), 295
esp_a2d_media_ctrl_ack_t (C++ 类型), 295
ESP_A2D_MEDIA_CTRL_CHECK_SRC_RDY (C++ 枚举子), 295
ESP_A2D_MEDIA_CTRL_NONE (C++ 枚举子), 295
ESP_A2D_MEDIA_CTRL_START (C++ 枚举子), 295
ESP_A2D_MEDIA_CTRL_STOP (C++ 枚举子), 295
ESP_A2D_MEDIA_CTRL_SUSPEND (C++ 枚举子), 295
esp_a2d_media_ctrl_t (C++ 类型), 295
esp_a2d_register_callback (C++ 函数), 288
esp_a2d_sink_connect (C++ 函数), 289
esp_a2d_sink_data_cb_t (C++ 类型), 294
esp_a2d_sink_deinit (C++ 函数), 289
esp_a2d_sink_disconnect (C++ 函数), 289
esp_a2d_sink_init (C++ 函数), 288
esp_a2d_sink_register_data_callback (C++ 函数), 288
esp_a2d_source_connect (C++ 函数), 290
esp_a2d_source_data_cb_t (C++ 类型), 294
esp_a2d_source_deinit (C++ 函数), 290
esp_a2d_source_disconnect (C++ 函数), 291
esp_a2d_source_init (C++ 函数), 290
esp_a2d_source_register_data_callback (C++ 函数), 290
esp_adc_cal_characteristics_t (C++ 类), 822
esp_adc_cal_characteristics_t::adc_num (C++ 成员), 822
esp_adc_cal_characteristics_t::atten (C++ 成员), 822
esp_adc_cal_characteristics_t::bit_width (C++ 成员), 822
esp_adc_cal_characteristics_t::coeff_a (C++ 成员), 822
esp_adc_cal_characteristics_t::coeff_b (C++ 成员), 822
esp_adc_cal_characteristics_t::high_curve (C++ 成员), 823
esp_adc_cal_characteristics_t::low_curve (C++ 成员), 823
esp_adc_cal_characteristics_t::vref (C++ 成员), 823
esp_adc_cal_characterize (C++ 函数), 821
esp_adc_cal_check_efuse (C++ 函数), 820
esp_adc_cal_get_voltage (C++ 函数), 822
esp_adc_cal_raw_to_voltage (C++ 函数), 821
ESP_ADC_CAL_VAL_DEFAULT_VREF (C++ 枚举子), 823
ESP_ADC_CAL_VAL_EFUSE_TP (C++ 枚举子), 823
ESP_ADC_CAL_VAL_EFUSE_VREF (C++ 枚举子), 823
esp_adc_cal_value_t (C++ 类型), 823
ESP_APP_DESC_MAGIC_WORD (C 宏), 1418
esp_app_desc_t (C++ 类), 1417
esp_app_desc_t::app_elf_sha256 (C++ 成员), 1418
esp_app_desc_t::date (C++ 成员), 1418
esp_app_desc_t::idf_ver (C++ 成员), 1418
esp_app_desc_t::magic_word (C++ 成员), 1418
esp_app_desc_t::project_name (C++ 成员), 1418
esp_app_desc_t::reserv1 (C++ 成员), 1418
esp_app_desc_t::reserv2 (C++ 成员), 1418
esp_app_desc_t::secure_version (C++ 成员), 1418

esp_app_desc_t::time (C++ 成员), 1418
esp_app_desc_t::version (C++ 成员), 1418
ESP_APP_ID_MAX (C 宏), 156
ESP_APP_ID_MIN (C 宏), 156
esp_apptrace_buffer_get (C++ 函数), 1421
esp_apptrace_buffer_put (C++ 函数), 1421
esp_apptrace_dest_t (C++ 类型), 1426
ESP_APPTRACE_DEST_TRAX (C++ 枚举子), 1426
ESP_APPTRACE_DEST_UART0 (C++ 枚举子), 1426
esp_apptrace_down_buffer_config (C++ 函数), 1420
esp_apptrace_down_buffer_get (C++ 函数), 1423
esp_apptrace_down_buffer_put (C++ 函数), 1423
esp_apptrace_fclose (C++ 函数), 1424
esp_apptrace_flush (C++ 函数), 1422
esp_apptrace_flush_nolock (C++ 函数), 1422
esp_apptrace_fopen (C++ 函数), 1424
esp_apptrace_fread (C++ 函数), 1425
esp_apptrace_fseek (C++ 函数), 1425
esp_apptrace_fstop (C++ 函数), 1425
esp_apptrace_ftell (C++ 函数), 1425
esp_apptrace_fwrite (C++ 函数), 1424
esp_apptrace_host_is_connected (C++ 函数), 1424
esp_apptrace_init (C++ 函数), 1420
esp_apptrace_read (C++ 函数), 1423
esp_apptrace_vprintf (C++ 函数), 1422
esp_apptrace_vprintf_to (C++ 函数), 1422
esp_apptrace_write (C++ 函数), 1421
esp_attr_control_t (C++ 类), 200
esp_attr_control_t::auto_rsp (C++ 成员), 200
esp_attr_desc_t (C++ 类), 199
esp_attr_desc_t::length (C++ 成员), 200
esp_attr_desc_t::max_length (C++ 成员), 200
esp_attr_desc_t::perm (C++ 成员), 200
esp_attr_desc_t::uuid_length (C++ 成员), 200
esp_attr_desc_t::uuid_p (C++ 成员), 200
esp_attr_desc_t::value (C++ 成员), 200
esp_attr_value_t (C++ 类), 200
esp_attr_value_t::attr_len (C++ 成员), 201
esp_attr_value_t::attr_max_len (C++ 成员), 201
esp_attr_value_t::attr_value (C++ 成员), 201
ESP_AVRC_BATT_CRITICAL (C++ 枚举子), 318
ESP_AVRC_BATT_EXTERNAL (C++ 枚举子), 318
ESP_AVRC_BATT_FULL_CHARGE (C++ 枚举子), 318
ESP_AVRC_BATT_NORMAL (C++ 枚举子), 318
esp_avrc_batt_stat_t (C++ 类型), 318
ESP_AVRC_BATT_WARNING (C++ 枚举子), 318
ESP_AVRC_BIT_MASK_OP_CLEAR (C++ 枚举子), 313
ESP_AVRC_BIT_MASK_OP_SET (C++ 枚举子), 313
esp_avrc_bit_mask_op_t (C++ 类型), 313
ESP_AVRC_BIT_MASK_OP_TEST (C++ 枚举子), 313
esp_avrc_ct_cb_event_t (C++ 类型), 314
esp_avrc_ct_cb_param_t (C++ 类型), 303
esp_avrc_ct_cb_param_t::avrc_ct_change_notify_param (C++ 类), 304
esp_avrc_ct_cb_param_t::avrc_ct_change_notify_param::event (C++ 成员), 304
esp_avrc_ct_cb_param_t::avrc_ct_change_notify_param::event (C++ 成员), 304
esp_avrc_ct_cb_param_t::avrc_ct_conn_stat_param (C++ 类), 304
esp_avrc_ct_cb_param_t::avrc_ct_conn_stat_param::connected (C++ 成员), 304
esp_avrc_ct_cb_param_t::avrc_ct_conn_stat_param::remote_bd (C++ 成员), 304
esp_avrc_ct_cb_param_t::avrc_ct_get_rn_caps_rsp_param (C++ 类), 304
esp_avrc_ct_cb_param_t::avrc_ct_get_rn_caps_rsp_param::cap (C++ 成员), 305
esp_avrc_ct_cb_param_t::avrc_ct_get_rn_caps_rsp_param::evt (C++ 成员), 305
esp_avrc_ct_cb_param_t::avrc_ct_meta_rsp_param (C++ 类), 305
esp_avrc_ct_cb_param_t::avrc_ct_meta_rsp_param::attr_id (C++ 成员), 305
esp_avrc_ct_cb_param_t::avrc_ct_meta_rsp_param::attr_lengt (C++ 成员), 305
esp_avrc_ct_cb_param_t::avrc_ct_meta_rsp_param::attr_text (C++ 成员), 305
esp_avrc_ct_cb_param_t::avrc_ct_psth_rsp_param (C++ 类), 305
esp_avrc_ct_cb_param_t::avrc_ct_psth_rsp_param::key_code

(C++ 成员), 305
esp_avrc_ct_cb_param_t::avrc_ct_psth_rsp_param::key_state (C++ 成员), 305
esp_avrc_ct_cb_param_t::avrc_ct_psth_rsp_param::tl (C++ 成员), 305
esp_avrc_ct_cb_param_t::avrc_ct_rmt_feats_param (C++ 类), 305
esp_avrc_ct_cb_param_t::avrc_ct_rmt_feats_param::feat_mask (C++ 成员), 305
esp_avrc_ct_cb_param_t::avrc_ct_rmt_feats_param::peer_mtu (C++ 成员), 305
esp_avrc_ct_cb_param_t::avrc_ct_rmt_feats_param::tg_feat_flag (C++ 成员), 305
esp_avrc_ct_cb_param_t::avrc_ct_set_volume_rsp_param (C++ 类), 306
esp_avrc_ct_cb_param_t::avrc_ct_set_volume_rsp_param::volume (C++ 成员), 306
esp_avrc_ct_cb_param_t::change_ntf (C++ 成员), 304
esp_avrc_ct_cb_param_t::conn_stat (C++ 成员), 304
esp_avrc_ct_cb_param_t::get_rn_caps_rsp (C++ 成员), 304
esp_avrc_ct_cb_param_t::meta_rsp (C++ 成员), 304
esp_avrc_ct_cb_param_t::psth_rsp (C++ 成员), 304
esp_avrc_ct_cb_param_t::rmt_feats (C++ 成员), 304
esp_avrc_ct_cb_param_t::set_volume_rsp (C++ 成员), 304
esp_avrc_ct_cb_t (C++ 类型), 308
ESP_AVRC_CT_CHANGE_NOTIFY_EVT (C++ 枚举子), 314
ESP_AVRC_CT_CONNECTION_STATE_EVT (C++ 枚举子), 314
esp_avrc_ct_deinit (C++ 函数), 297
ESP_AVRC_CT_GET_RN_CAPABILITIES_RSP_EVT (C++ 枚举子), 314
esp_avrc_ct_init (C++ 函数), 297
ESP_AVRC_CT_METADATA_RSP_EVT (C++ 枚举子), 314
ESP_AVRC_CT_PASSTHROUGH_RSP_EVT (C++ 枚举子), 314
ESP_AVRC_CT_PLAY_STATUS_RSP_EVT (C++ 枚举子), 314
esp_avrc_ct_register_callback (C++ 函数), 296
ESP_AVRC_CT_REMOTE_FEATURES_EVT (C++ 枚举子), 314
esp_avrc_ct_send_get_rn_capabilities_cmd (C++ 函数), 297
esp_avrc_ct_send_metadata_cmd (C++ 函数), 299
esp_avrc_ct_send_passthrough_cmd (C++ 函数), 299
esp_avrc_ct_send_register_notification_cmd (C++ 函数), 298
esp_avrc_ct_send_set_absolute_volume_cmd (C++ 函数), 298
esp_avrc_ct_send_set_player_value_cmd (C++ 函数), 297
ESP_AVRC_CT_SET_ABSOLUTE_VOLUME_RSP_EVT (C++ 枚举子), 314
ESP_AVRC_FEAT_ADV_CTRL (C++ 枚举子), 309
ESP_AVRC_FEAT_BROWSE (C++ 枚举子), 309
ESP_AVRC_FEAT_FLAG_BROWSING (C++ 枚举子), 309
ESP_AVRC_FEAT_FLAG_CAT1 (C++ 枚举子), 309
ESP_AVRC_FEAT_FLAG_CAT2 (C++ 枚举子), 309
ESP_AVRC_FEAT_FLAG_CAT3 (C++ 枚举子), 309
ESP_AVRC_FEAT_FLAG_CAT4 (C++ 枚举子), 309
ESP_AVRC_FEAT_FLAG_COVER_ART_GET_IMAGE (C++ 枚举子), 309
ESP_AVRC_FEAT_FLAG_COVER_ART_GET_IMAGE_PROP (C++ 枚举子), 309
ESP_AVRC_FEAT_FLAG_COVER_ART_GET_LINKED_THUMBNAIL (C++ 枚举子), 309
ESP_AVRC_FEAT_META_DATA (C++ 枚举子), 309
ESP_AVRC_FEAT_RCCT (C++ 枚举子), 309
ESP_AVRC_FEAT_RCTG (C++ 枚举子), 308
ESP_AVRC_FEAT_VENDOR (C++ 枚举子), 309
esp_avrc_feature_flag_t (C++ 类型), 309
esp_avrc_features_t (C++ 类型), 308
ESP_AVRC_MD_ATTR_ALBUM (C++ 枚举子), 315
ESP_AVRC_MD_ATTR_ARTIST (C++ 枚举子), 315
ESP_AVRC_MD_ATTR_GENRE (C++ 枚举子), 315

esp_avrc_md_attr_mask_t (C++ 类型), 314
ESP_AVRC_MD_ATTR_NUM_TRACKS (C++ 枚举子), 315
ESP_AVRC_MD_ATTR_PLAYING_TIME (C++ 枚举子), 315
ESP_AVRC_MD_ATTR_TITLE (C++ 枚举子), 314
ESP_AVRC_MD_ATTR_TRACK_NUM (C++ 枚举子), 315
ESP_AVRC_PLAYBACK_ERROR (C++ 枚举子), 319
ESP_AVRC_PLAYBACK_FWD_SEEK (C++ 枚举子), 319
ESP_AVRC_PLAYBACK_PAUSED (C++ 枚举子), 319
ESP_AVRC_PLAYBACK_PLAYING (C++ 枚举子), 318
ESP_AVRC_PLAYBACK_REV_SEEK (C++ 枚举子), 319
esp_avrc_playback_stat_t (C++ 类型), 318
ESP_AVRC_PLAYBACK_STOPPED (C++ 枚举子), 318
esp_avrc_ps_attr_ids_t (C++ 类型), 316
esp_avrc_ps_eq_value_ids_t (C++ 类型), 317
ESP_AVRC_PS_EQUALIZER (C++ 枚举子), 316
ESP_AVRC_PS_EQUALIZER_OFF (C++ 枚举子), 317
ESP_AVRC_PS_EQUALIZER_ON (C++ 枚举子), 317
ESP_AVRC_PS_MAX_ATTR (C++ 枚举子), 317
ESP_AVRC_PS_REPEAT_GROUP (C++ 枚举子), 317
ESP_AVRC_PS_REPEAT_MODE (C++ 枚举子), 316
ESP_AVRC_PS_REPEAT_OFF (C++ 枚举子), 317
ESP_AVRC_PS_REPEAT_SINGLE (C++ 枚举子), 317
esp_avrc_ps_rpt_value_ids_t (C++ 类型), 317
ESP_AVRC_PS_SCAN_ALL (C++ 枚举子), 317
ESP_AVRC_PS_SCAN_GROUP (C++ 枚举子), 317
ESP_AVRC_PS_SCAN_MODE (C++ 枚举子), 316
ESP_AVRC_PS_SCAN_OFF (C++ 枚举子), 317
esp_avrc_ps_scn_value_ids_t (C++ 类型), 317
esp_avrc_ps_shf_value_ids_t (C++ 类型), 317
ESP_AVRC_PS_SHUFFLE_ALL (C++ 枚举子), 317
ESP_AVRC_PS_SHUFFLE_GROUP (C++ 枚举子), 317
ESP_AVRC_PS_SHUFFLE_MODE (C++ 枚举子), 316
ESP_AVRC_PS_SHUFFLE_OFF (C++ 枚举子), 317
esp_avrc_psth_bit_mask_operation (C++ 函数), 301
esp_avrc_psth_bit_mask_t (C++ 类), 307
esp_avrc_psth_bit_mask_t::bits (C++ 成员), 308
ESP_AVRC_PSTH_FILTER_ALLOWED_CMD (C++ 枚举子), 313
ESP_AVRC_PSTH_FILTER_SUPPORT_MAX (C++ 枚举子), 313
ESP_AVRC_PSTH_FILTER_SUPPORTED_CMD (C++ 枚举子), 313
esp_avrc_psth_filter_t (C++ 类型), 313
ESP_AVRC_PT_CMD_0 (C++ 枚举子), 310
ESP_AVRC_PT_CMD_1 (C++ 枚举子), 310
ESP_AVRC_PT_CMD_2 (C++ 枚举子), 310
ESP_AVRC_PT_CMD_3 (C++ 枚举子), 310
ESP_AVRC_PT_CMD_4 (C++ 枚举子), 311
ESP_AVRC_PT_CMD_5 (C++ 枚举子), 311
ESP_AVRC_PT_CMD_6 (C++ 枚举子), 311
ESP_AVRC_PT_CMD_7 (C++ 枚举子), 311
ESP_AVRC_PT_CMD_8 (C++ 枚举子), 311
ESP_AVRC_PT_CMD_9 (C++ 枚举子), 311
ESP_AVRC_PT_CMD_ANGLE (C++ 枚举子), 312
ESP_AVRC_PT_CMD_BACKWARD (C++ 枚举子), 312
ESP_AVRC_PT_CMD_CHAN_DOWN (C++ 枚举子), 311
ESP_AVRC_PT_CMD_CHAN_UP (C++ 枚举子), 311
ESP_AVRC_PT_CMD_CLEAR (C++ 枚举子), 311
ESP_AVRC_PT_CMD_CONT_MENU (C++ 枚举子), 310
ESP_AVRC_PT_CMD_DISP_INFO (C++ 枚举子), 311
ESP_AVRC_PT_CMD_DOT (C++ 枚举子), 311
ESP_AVRC_PT_CMD_DOWN (C++ 枚举子), 310
ESP_AVRC_PT_CMD_EJECT (C++ 枚举子), 312
ESP_AVRC_PT_CMD_ENTER (C++ 枚举子), 311
ESP_AVRC_PT_CMD_EXIT (C++ 枚举子), 310
ESP_AVRC_PT_CMD_F1 (C++ 枚举子), 312
ESP_AVRC_PT_CMD_F2 (C++ 枚举子), 313
ESP_AVRC_PT_CMD_F3 (C++ 枚举子), 313
ESP_AVRC_PT_CMD_F4 (C++ 枚举子), 313
ESP_AVRC_PT_CMD_F5 (C++ 枚举子), 313
ESP_AVRC_PT_CMD_FAST_FORWARD (C++ 枚举子), 312
ESP_AVRC_PT_CMD_FAV_MENU (C++ 枚举子), 310
ESP_AVRC_PT_CMD_FORWARD (C++ 枚举子), 312
ESP_AVRC_PT_CMD_HELP (C++ 枚举子), 311
ESP_AVRC_PT_CMD_INPUT_SEL (C++ 枚举子), 311
ESP_AVRC_PT_CMD_LEFT (C++ 枚举子), 310
ESP_AVRC_PT_CMD_LEFT_DOWN (C++ 枚举子), 310
ESP_AVRC_PT_CMD_LEFT_UP (C++ 枚举子), 310
ESP_AVRC_PT_CMD_MUTE (C++ 枚举子), 312
ESP_AVRC_PT_CMD_PAGE_DOWN (C++ 枚举子), 312

ESP_AVRC_PT_CMD_PAGE_UP (C++ 枚举子), 311
ESP_AVRC_PT_CMD_PAUSE (C++ 枚举子), 312
ESP_AVRC_PT_CMD_PLAY (C++ 枚举子), 312
ESP_AVRC_PT_CMD_POWER (C++ 枚举子), 312
ESP_AVRC_PT_CMD_PREV_CHAN (C++ 枚举子), 311
ESP_AVRC_PT_CMD_RECORD (C++ 枚举子), 312
ESP_AVRC_PT_CMD_REWIND (C++ 枚举子), 312
ESP_AVRC_PT_CMD_RIGHT (C++ 枚举子), 310
ESP_AVRC_PT_CMD_RIGHT_DOWN (C++ 枚举子), 310
ESP_AVRC_PT_CMD_RIGHT_UP (C++ 枚举子), 310
ESP_AVRC_PT_CMD_ROOT_MENU (C++ 枚举子), 310
ESP_AVRC_PT_CMD_SELECT (C++ 枚举子), 309
ESP_AVRC_PT_CMD_SETUP_MENU (C++ 枚举子), 310
ESP_AVRC_PT_CMD_SOUND_SEL (C++ 枚举子), 311
ESP_AVRC_PT_CMD_STATE_PRESSED (C++ 枚举子), 313
ESP_AVRC_PT_CMD_STATE_RELEASED (C++ 枚举子), 313
esp_avrc_pt_cmd_state_t (C++ 类型), 313
ESP_AVRC_PT_CMD_STOP (C++ 枚举子), 312
ESP_AVRC_PT_CMD_SUBPICT (C++ 枚举子), 312
esp_avrc_pt_cmd_t (C++ 类型), 309
ESP_AVRC_PT_CMD_UP (C++ 枚举子), 310
ESP_AVRC_PT_CMD_VENDOR (C++ 枚举子), 313
ESP_AVRC_PT_CMD_VOL_DOWN (C++ 枚举子), 312
ESP_AVRC_PT_CMD_VOL_UP (C++ 枚举子), 312
ESP_AVRC_RN_ADDRESSED_PLAYER_CHANGE (C++ 枚举子), 316
ESP_AVRC_RN_APP_SETTING_CHANGE (C++ 枚举子), 315
ESP_AVRC_RN_AVAILABLE_PLAYERS_CHANGE (C++ 枚举子), 315
ESP_AVRC_RN_BATTERY_STATUS_CHANGE (C++ 枚举子), 315
ESP_AVRC_RN_CAP_ALLOWED_EVT (C++ 枚举子), 316
ESP_AVRC_RN_CAP_MAX (C++ 枚举子), 316
ESP_AVRC_RN_CAP_SUPPORTED_EVT (C++ 枚举子), 316
esp_avrc_rn_event_ids_t (C++ 类型), 315
esp_avrc_rn_evt_bit_mask_operation (C++ 函数), 302
esp_avrc_rn_evt_cap_mask_t (C++ 类), 308
esp_avrc_rn_evt_cap_mask_t::bits (C++ 成员), 308
esp_avrc_rn_evt_cap_t (C++ 类型), 316
ESP_AVRC_RN_MAX_EVT (C++ 枚举子), 316
ESP_AVRC_RN_NOW_PLAYING_CHANGE (C++ 枚举子), 315
esp_avrc_rn_param_t (C++ 类型), 303
esp_avrc_rn_param_t::batt (C++ 成员), 303
esp_avrc_rn_param_t::elm_id (C++ 成员), 303
esp_avrc_rn_param_t::play_pos (C++ 成员), 303
esp_avrc_rn_param_t::playback (C++ 成员), 303
esp_avrc_rn_param_t::volume (C++ 成员), 303
ESP_AVRC_RN_PLAY_POS_CHANGED (C++ 枚举子), 315
ESP_AVRC_RN_PLAY_STATUS_CHANGE (C++ 枚举子), 315
ESP_AVRC_RN_RSP_CHANGED (C++ 枚举子), 316
ESP_AVRC_RN_RSP_INTERIM (C++ 枚举子), 316
esp_avrc_rn_rsp_t (C++ 类型), 316
ESP_AVRC_RN_SYSTEM_STATUS_CHANGE (C++ 枚举子), 315
ESP_AVRC_RN_TRACK_CHANGE (C++ 枚举子), 315
ESP_AVRC_RN_TRACK_REACHED_END (C++ 枚举子), 315
ESP_AVRC_RN_TRACK_REACHED_START (C++ 枚举子), 315
ESP_AVRC_RN_UIDS_CHANGE (C++ 枚举子), 316
ESP_AVRC_RN_VOLUME_CHANGE (C++ 枚举子), 316
ESP_AVRC_RSP_ACCEPT (C++ 枚举子), 318
ESP_AVRC_RSP_CHANGED (C++ 枚举子), 318
ESP_AVRC_RSP_IMPL_STBL (C++ 枚举子), 318
ESP_AVRC_RSP_IN_TRANS (C++ 枚举子), 318
ESP_AVRC_RSP_INTERIM (C++ 枚举子), 318
ESP_AVRC_RSP_NOT_IMPL (C++ 枚举子), 318
ESP_AVRC_RSP_REJECT (C++ 枚举子), 318
esp_avrc_rsp_t (C++ 类型), 317
esp_avrc_tg_cb_event_t (C++ 类型), 314
esp_avrc_tg_cb_param_t (C++ 类型), 306
esp_avrc_tg_cb_param_t::avrc_tg_conn_stat_param (C++ 类), 306
esp_avrc_tg_cb_param_t::avrc_tg_conn_stat_param::connected (C++ 成员), 306

esp_avrc_tg_cb_param_t::avrc_tg_conn_stat_param (C++ 成员), 306
esp_avrc_tg_cb_param_t::avrc_tg_psth_cmd_param (C++ 类), 306
esp_avrc_tg_cb_param_t::avrc_tg_psth_cmd_param::key_code (C++ 成员), 307
esp_avrc_tg_cb_param_t::avrc_tg_psth_cmd_param::key_state (C++ 成员), 307
esp_avrc_tg_cb_param_t::avrc_tg_reg_ntf_param (C++ 类), 307
esp_avrc_tg_cb_param_t::avrc_tg_reg_ntf_param::event_id (C++ 成员), 307
esp_avrc_tg_cb_param_t::avrc_tg_reg_ntf_param::event_parameter (C++ 成员), 307
esp_avrc_tg_cb_param_t::avrc_tg_rmt_feats_param (C++ 类), 307
esp_avrc_tg_cb_param_t::avrc_tg_rmt_feats_param::feat_mask (C++ 成员), 307
esp_avrc_tg_cb_param_t::avrc_tg_rmt_feats_param::peer_features (C++ 成员), 307
esp_avrc_tg_cb_param_t::avrc_tg_rmt_feats_param::tg_feat_flag (C++ 成员), 307
esp_avrc_tg_cb_param_t::avrc_tg_set_abs_vol_param (C++ 类), 307
esp_avrc_tg_cb_param_t::avrc_tg_set_abs_vol_param::volume (C++ 成员), 307
esp_avrc_tg_cb_param_t::conn_stat (C++ 成员), 306
esp_avrc_tg_cb_param_t::psth_cmd (C++ 成员), 306
esp_avrc_tg_cb_param_t::reg_ntf (C++ 成员), 306
esp_avrc_tg_cb_param_t::rmt_feats (C++ 成员), 306
esp_avrc_tg_cb_param_t::set_abs_vol (C++ 成员), 306
esp_avrc_tg_cb_t (C++ 类型), 308
ESP_AVRC_TG_CONNECTION_STATE_EVT (C++ 枚举子), 314
esp_avrc_tg_deinit (C++ 函数), 300
esp_avrc_tg_get_psth_cmd_filter (C++ 函数), 300
esp_avrc_tg_get_rn_evt_cap (C++ 函数), 301
esp_avrc_tg_init (C++ 函数), 300
ESP_AVRC_TG_PASSTHROUGH_CMD_EVT (C++ 枚举子), 314
esp_avrc_tg_register_callback (C++ 函数), 299
ESP_AVRC_TG_REGISTER_NOTIFICATION_EVT (C++ 枚举子), 314
ESP_AVRC_TG_REMOTE_FEATURES_EVT (C++ 枚举子), 314
esp_avrc_tg_send_rn_rsp (C++ 函数), 303
ESP_AVRC_TG_SET_ABSOLUTE_VOLUME_CMD_EVT (C++ 枚举子), 314
esp_avrc_tg_set_psth_cmd_filter (C++ 函数), 301
esp_avrc_tg_set_rn_evt_cap (C++ 函数), 302
ESP_AVRC_TRANS_LABEL_MAX (C 宏), 308
esp_base_mac_addr_get (C++ 函数), 1683
esp_base_mac_addr_set (C++ 函数), 1683
ESP_BD_ADDR_HEX (C 宏), 156
ESP_BD_ADDR_LEN (C 宏), 156
ESP_BD_ADDR_STR (C 宏), 156
esp_bd_addr_t (C++ 类型), 156
ESP_BLE_AD_MANUFACTURER_SPECIFIC_TYPE (C++ 枚举子), 194
ESP_BLE_AD_TYPE_128SERVICE_DATA (C++ 枚举子), 193
ESP_BLE_AD_TYPE_128SOL_SRV_UUID (C++ 枚举子), 193
ESP_BLE_AD_TYPE_128SRV_CMPL (C++ 枚举子), 193
ESP_BLE_AD_TYPE_128SRV_PART (C++ 枚举子), 193
ESP_BLE_AD_TYPE_16SRV_CMPL (C++ 枚举子), 192
ESP_BLE_AD_TYPE_16SRV_PART (C++ 枚举子), 192
ESP_BLE_AD_TYPE_32SERVICE_DATA (C++ 枚举子), 193
ESP_BLE_AD_TYPE_32SOL_SRV_UUID (C++ 枚举子), 193
ESP_BLE_AD_TYPE_32SRV_CMPL (C++ 枚举子), 192
ESP_BLE_AD_TYPE_32SRV_PART (C++ 枚举子), 192
ESP_BLE_AD_TYPE_ADV_INT (C++ 枚举子), 193
ESP_BLE_AD_TYPE_APPEARANCE (C++ 枚举子), 193
ESP_BLE_AD_TYPE_CHAN_MAP_UPDATE (C++ 枚举子), 193

ESP_BLE_AD_TYPE_DEV_CLASS (C++ 枚举子), 193
ESP_BLE_AD_TYPE_FLAG (C++ 枚举子), 192
ESP_BLE_AD_TYPE_INDOOR_POSITION (C++ 枚举子), 193
ESP_BLE_AD_TYPE_INT_RANGE (C++ 枚举子), 193
ESP_BLE_AD_TYPE_LE_DEV_ADDR (C++ 枚举子), 193
ESP_BLE_AD_TYPE_LE_ROLE (C++ 枚举子), 193
ESP_BLE_AD_TYPE_LE_SECURE_CONFIRM (C++ 枚举子), 193
ESP_BLE_AD_TYPE_LE_SECURE_RANDOM (C++ 枚举子), 193
ESP_BLE_AD_TYPE_LE_SUPPORT_FEATURE (C++ 枚举子), 193
ESP_BLE_AD_TYPE_NAME_CMPL (C++ 枚举子), 193
ESP_BLE_AD_TYPE_NAME_SHORT (C++ 枚举子), 193
ESP_BLE_AD_TYPE_PUBLIC_TARGET (C++ 枚举子), 193
ESP_BLE_AD_TYPE_RANDOM_TARGET (C++ 枚举子), 193
ESP_BLE_AD_TYPE_SERVICE_DATA (C++ 枚举子), 193
ESP_BLE_AD_TYPE_SM_OOB_FLAG (C++ 枚举子), 193
ESP_BLE_AD_TYPE_SM_TK (C++ 枚举子), 193
ESP_BLE_AD_TYPE_SOL_SRV_UUID (C++ 枚举子), 193
ESP_BLE_AD_TYPE_SPAIR_C256 (C++ 枚举子), 193
ESP_BLE_AD_TYPE_SPAIR_R256 (C++ 枚举子), 193
ESP_BLE_AD_TYPE_TRANS_DISC_DATA (C++ 枚举子), 193
ESP_BLE_AD_TYPE_TX_PWR (C++ 枚举子), 193
ESP_BLE_AD_TYPE_URI (C++ 枚举子), 193
esp_ble_addr_type_t (C++ 类型), 158
esp_ble_adv_channel_t (C++ 类型), 194
ESP_BLE_ADV_DATA_LEN_MAX (C 宏), 190
esp_ble_adv_data_t (C++ 类), 180
esp_ble_adv_data_t::appearance (C++ 成员), 180
esp_ble_adv_data_t::flag (C++ 成员), 181
esp_ble_adv_data_t::include_name (C++ 成员), 180
esp_ble_adv_data_t::include_txpower (C++ 成员), 180
esp_ble_adv_data_t::manufacturer_len (C++ 成员), 180
esp_ble_adv_data_t::max_interval (C++ 成员), 180
esp_ble_adv_data_t::min_interval (C++ 成员), 180
esp_ble_adv_data_t::p_manufacturer_data (C++ 成员), 180
esp_ble_adv_data_t::p_service_data (C++ 成员), 181
esp_ble_adv_data_t::p_service_uuid (C++ 成员), 181
esp_ble_adv_data_t::service_data_len (C++ 成员), 180
esp_ble_adv_data_t::service_uuid_len (C++ 成员), 181
esp_ble_adv_data_t::set_scan_rsp (C++ 成员), 180
esp_ble_adv_data_type (C++ 类型), 192
esp_ble_adv_filter_t (C++ 类型), 194
ESP_BLE_ADV_FLAG_BREDR_NOT_SPT (C 宏), 187
ESP_BLE_ADV_FLAG_DMT_CONTROLLER_SPT (C 宏), 187
ESP_BLE_ADV_FLAG_DMT_HOST_SPT (C 宏), 187
ESP_BLE_ADV_FLAG_GEN_DISC (C 宏), 187
ESP_BLE_ADV_FLAG_LIMIT_DISC (C 宏), 187
ESP_BLE_ADV_FLAG_NON_LIMIT_DISC (C 宏), 187
esp_ble_adv_params_t (C++ 类), 179
esp_ble_adv_params_t::adv_filter_policy (C++ 成员), 180
esp_ble_adv_params_t::adv_int_max (C++ 成员), 179
esp_ble_adv_params_t::adv_int_min (C++ 成员), 179
esp_ble_adv_params_t::adv_type (C++ 成员), 179
esp_ble_adv_params_t::channel_map (C++ 成员), 180
esp_ble_adv_params_t::own_addr_type (C++ 成员), 179
esp_ble_adv_params_t::peer_addr (C++ 成员), 179

esp_ble_adv_params_t::peer_addr_type (C++ 成员), 180
esp_ble_adv_type_t (C++ 类型), 194
ESP_BLE_APPEARANCE_BLOOD_PRESSURE_ARM (C 宏), 188
ESP_BLE_APPEARANCE_BLOOD_PRESSURE_WRIST (C 宏), 188
ESP_BLE_APPEARANCE_CYCLING_CADENCE (C 宏), 189
ESP_BLE_APPEARANCE_CYCLING_COMPUTER (C 宏), 189
ESP_BLE_APPEARANCE_CYCLING_POWER (C 宏), 189
ESP_BLE_APPEARANCE_CYCLING_SPEED (C 宏), 189
ESP_BLE_APPEARANCE_CYCLING_SPEED_CADENCE (C 宏), 189
ESP_BLE_APPEARANCE_GENERIC_BARCODE_SCANNER (C 宏), 188
ESP_BLE_APPEARANCE_GENERIC_BLOOD_PRESSURE (C 宏), 188
ESP_BLE_APPEARANCE_GENERIC_CLOCK (C 宏), 188
ESP_BLE_APPEARANCE_GENERIC_COMPUTER (C 宏), 188
ESP_BLE_APPEARANCE_GENERIC_CONTINUOUS_GLUCOSE_MONITOR (C 宏), 189
ESP_BLE_APPEARANCE_GENERIC_CYCLING (C 宏), 189
ESP_BLE_APPEARANCE_GENERIC_DISPLAY (C 宏), 188
ESP_BLE_APPEARANCE_GENERIC_EYEGLASSES (C 宏), 188
ESP_BLE_APPEARANCE_GENERIC_GLUCOSE (C 宏), 189
ESP_BLE_APPEARANCE_GENERIC_HEART_RATE (C 宏), 188
ESP_BLE_APPEARANCE_GENERIC_HID (C 宏), 188
ESP_BLE_APPEARANCE_GENERIC_INSULIN_PUMP (C 宏), 189
ESP_BLE_APPEARANCE_GENERIC_KEYRING (C 宏), 188
ESP_BLE_APPEARANCE_GENERIC_MEDIA_PLAYER (C 宏), 188
ESP_BLE_APPEARANCE_GENERIC_MEDICATION_DELIVERY (C 宏), 189
ESP_BLE_APPEARANCE_GENERIC_OUTDOOR_SPORTS (C 宏), 189
ESP_BLE_APPEARANCE_GENERIC_PERSONAL_MOBILITY_DEVICE (C 宏), 189
ESP_BLE_APPEARANCE_GENERIC_PHONE (C 宏), 188
ESP_BLE_APPEARANCE_GENERIC_PULSE_OXIMETER (C 宏), 189
ESP_BLE_APPEARANCE_GENERIC_REMOTE (C 宏), 188
ESP_BLE_APPEARANCE_GENERIC_TAG (C 宏), 188
ESP_BLE_APPEARANCE_GENERIC_THERMOMETER (C 宏), 188
ESP_BLE_APPEARANCE_GENERIC_WALKING (C 宏), 189
ESP_BLE_APPEARANCE_GENERIC_WATCH (C 宏), 188
ESP_BLE_APPEARANCE_GENERIC_WEIGHT (C 宏), 189
ESP_BLE_APPEARANCE_HEART_RATE_BELT (C 宏), 188
ESP_BLE_APPEARANCE_HID_BARCODE_SCANNER (C 宏), 189
ESP_BLE_APPEARANCE_HID_CARD_READER (C 宏), 189
ESP_BLE_APPEARANCE_HID_DIGITAL_PEN (C 宏), 189
ESP_BLE_APPEARANCE_HID_DIGITIZER_TABLET (C 宏), 188
ESP_BLE_APPEARANCE_HID_GAMEPAD (C 宏), 188
ESP_BLE_APPEARANCE_HID_JOYSTICK (C 宏), 188
ESP_BLE_APPEARANCE_HID_KEYBOARD (C 宏), 188
ESP_BLE_APPEARANCE_HID_MOUSE (C 宏), 188
ESP_BLE_APPEARANCE_INSULIN_PEN (C 宏), 189
ESP_BLE_APPEARANCE_INSULIN_PUMP_DURABLE_PUMP (C 宏), 189
ESP_BLE_APPEARANCE_INSULIN_PUMP_PATCH_PUMP (C 宏), 189
ESP_BLE_APPEARANCE_MOBILITY_SCOOTER (C 宏), 189
ESP_BLE_APPEARANCE_OUTDOOR_SPORTS_LOCATION (C 宏), 189
ESP_BLE_APPEARANCE_OUTDOOR_SPORTS_LOCATION_AND_NAV (C 宏), 189
ESP_BLE_APPEARANCE_OUTDOOR_SPORTS_LOCATION_POD (C 宏), 190
ESP_BLE_APPEARANCE_OUTDOOR_SPORTS_LOCATION_POD_AND_NAV (C 宏), 190
ESP_BLE_APPEARANCE_POWERED_WHEELCHAIR (C 宏), 189
ESP_BLE_APPEARANCE_PULSE_OXIMETER_FINGERTIP (C 宏), 189
ESP_BLE_APPEARANCE_PULSE_OXIMETER_WRIST (C 宏), 189

ESP_BLE_APPEARANCE_SPORTS_WATCH (C 宏), 188
ESP_BLE_APPEARANCE_THERMOMETER_EAR (C 宏), 188
ESP_BLE_APPEARANCE_UNKNOWN (C 宏), 188
ESP_BLE_APPEARANCE_WALKING_IN_SHOE (C 宏), 189
ESP_BLE_APPEARANCE_WALKING_ON_HIP (C 宏), 189
ESP_BLE_APPEARANCE_WALKING_ON_SHOE (C 宏), 189
esp_ble_auth_cmpl_t (C++ 类), 186
esp_ble_auth_cmpl_t::addr_type (C++ 成员), 186
esp_ble_auth_cmpl_t::auth_mode (C++ 成员), 186
esp_ble_auth_cmpl_t::bd_addr (C++ 成员), 186
esp_ble_auth_cmpl_t::dev_type (C++ 成员), 186
esp_ble_auth_cmpl_t::fail_reason (C++ 成员), 186
esp_ble_auth_cmpl_t::key (C++ 成员), 186
esp_ble_auth_cmpl_t::key_present (C++ 成员), 186
esp_ble_auth_cmpl_t::key_type (C++ 成员), 186
esp_ble_auth_cmpl_t::success (C++ 成员), 186
esp_ble_auth_req_t (C++ 类型), 190
esp_ble_bond_dev_t (C++ 类), 185
esp_ble_bond_dev_t::bd_addr (C++ 成员), 185
esp_ble_bond_dev_t::bond_key (C++ 成员), 185
esp_ble_bond_key_info_t (C++ 类), 185
esp_ble_bond_key_info_t::key_mask (C++ 成员), 185
esp_ble_bond_key_info_t::pcsrk_key (C++ 成员), 185
esp_ble_bond_key_info_t::penc_key (C++ 成员), 185
esp_ble_bond_key_info_t::pid_key (C++ 成员), 185
esp_ble_confirm_reply (C++ 函数), 169
ESP_BLE_CONN_INT_MAX (C 宏), 155
ESP_BLE_CONN_INT_MIN (C 宏), 155
ESP_BLE_CONN_LATENCY_MAX (C 宏), 155
ESP_BLE_CONN_PARAM_UNDEF (C 宏), 156
ESP_BLE_CONN_SUP_TOUT_MAX (C 宏), 155
ESP_BLE_CONN_SUP_TOUT_MIN (C 宏), 155
esp_ble_conn_update_params_t (C++ 类), 182
esp_ble_conn_update_params_t::bda (C++ 成员), 182
esp_ble_conn_update_params_t::latency (C++ 成员), 182
esp_ble_conn_update_params_t::max_int (C++ 成员), 182
esp_ble_conn_update_params_t::min_int (C++ 成员), 182
esp_ble_conn_update_params_t::timeout (C++ 成员), 182
ESP_BLE_CSR_KEY_MASK (C 宏), 156
esp_ble_duplicate_exceptional_info_type_t (C++ 类型), 197
ESP_BLE_DUPLICATE_EXCEPTIONAL_LIST_ADD (C++ 枚举子), 197
ESP_BLE_DUPLICATE_EXCEPTIONAL_LIST_CLEAN (C++ 枚举子), 197
ESP_BLE_DUPLICATE_EXCEPTIONAL_LIST_REMOVE (C++ 枚举子), 197
ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_ADDR_LIST (C++ 枚举子), 198
ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_ALL_LIST (C++ 枚举子), 198
ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_INFO_ADV_ADDR (C++ 枚举子), 197
ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_INFO_MESH_BEACON_TYPE (C++ 枚举子), 198
ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_INFO_MESH_LINK_ID (C++ 枚举子), 198
ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_INFO_MESH_PROV_SRV_ADV (C++ 枚举子), 198
ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_INFO_MESH_PROXY_SRV_ADV (C++ 枚举子), 198
ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_MESH_BEACON_TYPE_LIST (C++ 枚举子), 198
ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_MESH_LINK_ID_LIST (C++ 枚举子), 198
ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_MESH_PROV_SRV_ADV_LIST (C++ 枚举子), 198
ESP_BLE_DUPLICATE_SCAN_EXCEPTIONAL_MESH_PROXY_SRV_ADV_LIST (C++ 枚举子), 198
ESP_BLE_ENC_KEY_MASK (C 宏), 156
ESP_BLE_EVT_CONN_ADV (C++ 枚举子), 197

ESP_BLE_EVT_CONN_DIR_ADV (C++ 枚举子), 197
ESP_BLE_EVT_DISC_ADV (C++ 枚举子), 197
ESP_BLE_EVT_NON_CONN_ADV (C++ 枚举子), 197
ESP_BLE_EVT_SCAN_RSP (C++ 枚举子), 197
esp_ble_evt_type_t (C++ 类型), 197
esp_ble_gap_add_duplicate_scan_exceptional_device (C++ 函数), 167
esp_ble_gap_cb_param_t (C++ 类型), 172
esp_ble_gap_cb_param_t::adv_data_cmpl (C++ 成员), 172
esp_ble_gap_cb_param_t::adv_data_raw_cmpl (C++ 成员), 172
esp_ble_gap_cb_param_t::adv_start_cmpl (C++ 成员), 173
esp_ble_gap_cb_param_t::adv_stop_cmpl (C++ 成员), 173
esp_ble_gap_cb_param_t::ble_adv_data_cmpl_evt_param (C++ 类), 173
esp_ble_gap_cb_param_t::ble_adv_data_cmpl_evt_param::status (C++ 成员), 174
esp_ble_gap_cb_param_t::ble_adv_data_raw_cmpl_evt_param (C++ 类), 174
esp_ble_gap_cb_param_t::ble_adv_data_raw_cmpl_evt_param::status (C++ 成员), 174
esp_ble_gap_cb_param_t::ble_adv_start_cmpl_evt_param (C++ 类), 174
esp_ble_gap_cb_param_t::ble_adv_start_cmpl_evt_param::status (C++ 成员), 174
esp_ble_gap_cb_param_t::ble_adv_stop_cmpl_evt_param (C++ 类), 174
esp_ble_gap_cb_param_t::ble_adv_stop_cmpl_evt_param::status (C++ 成员), 174
esp_ble_gap_cb_param_t::ble_clear_bond_dev_cmpl_evt_param (C++ 类), 174
esp_ble_gap_cb_param_t::ble_clear_bond_dev_cmpl_evt_param::status (C++ 成员), 174
esp_ble_gap_cb_param_t::ble_get_bond_dev_cmpl_evt_param (C++ 类), 174
esp_ble_gap_cb_param_t::ble_get_bond_dev_cmpl_evt_param::bond_dev_num (C++ 成员), 175
esp_ble_gap_cb_param_t::ble_get_bond_dev_cmpl_evt_param::bond_dev (C++ 成员), 175
esp_ble_gap_cb_param_t::ble_get_bond_dev_cmpl_evt_param::s (C++ 成员), 175
esp_ble_gap_cb_param_t::ble_local_privacy_cmpl_evt_param (C++ 类), 175
esp_ble_gap_cb_param_t::ble_local_privacy_cmpl_evt_param:: (C++ 成员), 175
esp_ble_gap_cb_param_t::ble_pkt_data_length_cmpl_evt_param (C++ 类), 175
esp_ble_gap_cb_param_t::ble_pkt_data_length_cmpl_evt_param (C++ 成员), 175
esp_ble_gap_cb_param_t::ble_pkt_data_length_cmpl_evt_param (C++ 成员), 175
esp_ble_gap_cb_param_t::ble_read_rssi_cmpl_evt_param (C++ 类), 175
esp_ble_gap_cb_param_t::ble_read_rssi_cmpl_evt_param::remo (C++ 成员), 175
esp_ble_gap_cb_param_t::ble_read_rssi_cmpl_evt_param::rssi (C++ 成员), 175
esp_ble_gap_cb_param_t::ble_read_rssi_cmpl_evt_param::stat (C++ 成员), 175
esp_ble_gap_cb_param_t::ble_remove_bond_dev_cmpl_evt_param (C++ 类), 175
esp_ble_gap_cb_param_t::ble_remove_bond_dev_cmpl_evt_param (C++ 成员), 176
esp_ble_gap_cb_param_t::ble_remove_bond_dev_cmpl_evt_param (C++ 成员), 176
esp_ble_gap_cb_param_t::ble_scan_param_cmpl_evt_param (C++ 类), 176
esp_ble_gap_cb_param_t::ble_scan_param_cmpl_evt_param::sta (C++ 成员), 176
esp_ble_gap_cb_param_t::ble_scan_result_evt_param (C++ 类), 176
esp_ble_gap_cb_param_t::ble_scan_result_evt_param::adv_dat (C++ 成员), 176
esp_ble_gap_cb_param_t::ble_scan_result_evt_param::bda (C++ 成员), 176
esp_ble_gap_cb_param_t::ble_scan_result_evt_param::ble_add (C++ 成员), 176
esp_ble_gap_cb_param_t::ble_scan_result_evt_param::ble_adv (C++ 成员), 176
esp_ble_gap_cb_param_t::ble_scan_result_evt_param::ble_evt (C++ 成员), 176

esp_ble_gap_cb_param_t::ble_scan_result_evt_param::ble_evt_type (C++ 成员), 176
esp_ble_gap_cb_param_t::ble_scan_result_evt_param::flag (C++ 成员), 176
esp_ble_gap_cb_param_t::ble_scan_result_evt_param::num_resps (C++ 成员), 177
esp_ble_gap_cb_param_t::ble_scan_result_evt_param::rssi (C++ 成员), 176
esp_ble_gap_cb_param_t::ble_scan_result_evt_param::search_evt (C++ 成员), 176
esp_ble_gap_cb_param_t::ble_scan_rsp_data_cmpl_evt_param (C++ 类), 177
esp_ble_gap_cb_param_t::ble_scan_rsp_data_cmpl_evt_param::status (C++ 成员), 177
esp_ble_gap_cb_param_t::ble_scan_rsp_data_raw_cmpl_evt_param (C++ 类), 177
esp_ble_gap_cb_param_t::ble_scan_rsp_data_raw_cmpl_evt_param::status (C++ 成员), 177
esp_ble_gap_cb_param_t::ble_scan_start_cmpl_evt_param (C++ 类), 177
esp_ble_gap_cb_param_t::ble_scan_start_cmpl_evt_param::status (C++ 成员), 177
esp_ble_gap_cb_param_t::ble_scan_stop_cmpl_evt_param (C++ 类), 177
esp_ble_gap_cb_param_t::ble_scan_stop_cmpl_evt_param::status (C++ 成员), 177
esp_ble_gap_cb_param_t::ble_security (C++ 成员), 173
esp_ble_gap_cb_param_t::ble_set_channels (C++ 成员), 173
esp_ble_gap_cb_param_t::ble_set_channels_evt_param (C++ 类), 177
esp_ble_gap_cb_param_t::ble_set_channels_evt_param::status (C++ 成员), 178
esp_ble_gap_cb_param_t::ble_set_rand_cmpl_evt_param (C++ 类), 178
esp_ble_gap_cb_param_t::ble_set_rand_cmpl_evt_param::status (C++ 成员), 178
esp_ble_gap_cb_param_t::ble_update_conn_params_evt_param (C++ 类), 178
esp_ble_gap_cb_param_t::ble_update_conn_params_evt_param::addr (C++ 成员), 178
esp_ble_gap_cb_param_t::ble_update_conn_params_evt_param::conn_int (C++ 成员), 178
esp_ble_gap_cb_param_t::ble_update_conn_params_evt_param::latency (C++ 成员), 178
esp_ble_gap_cb_param_t::ble_update_conn_params_evt_param::min_int (C++ 成员), 178
esp_ble_gap_cb_param_t::ble_update_conn_params_evt_param::status (C++ 成员), 178
esp_ble_gap_cb_param_t::ble_update_conn_params_evt_param::timeout (C++ 成员), 178
esp_ble_gap_cb_param_t::ble_update_duplicate_exceptional_list_cmpl_evt_param (C++ 类), 178
esp_ble_gap_cb_param_t::ble_update_duplicate_exceptional_list_cmpl_evt_param::num_dup (C++ 成员), 179
esp_ble_gap_cb_param_t::ble_update_duplicate_exceptional_list_cmpl_evt_param::status (C++ 成员), 179
esp_ble_gap_cb_param_t::ble_update_duplicate_exceptional_list_cmpl_evt_param::subcode (C++ 成员), 178
esp_ble_gap_cb_param_t::ble_update_whitelist_cmpl_evt_param (C++ 类), 179
esp_ble_gap_cb_param_t::ble_update_whitelist_cmpl_evt_param::status (C++ 成员), 179
esp_ble_gap_cb_param_t::ble_update_whitelist_cmpl_evt_param::wl_operation (C++ 成员), 179
esp_ble_gap_cb_param_t::clear_bond_dev_cmpl (C++ 成员), 173
esp_ble_gap_cb_param_t::get_bond_dev_cmpl (C++ 成员), 173
esp_ble_gap_cb_param_t::local_privacy_cmpl (C++ 成员), 173
esp_ble_gap_cb_param_t::pkt_data_lenth_cmpl (C++ 成员), 173
esp_ble_gap_cb_param_t::read_rssi_cmpl (C++ 成员), 173

esp_ble_gap_cb_param_t::remove_bond_dev_cmpl (C++ 成员), 173
esp_ble_gap_cb_param_t::scan_param_cmpl (C++ 成员), 172
esp_ble_gap_cb_param_t::scan_rsp_data_cmpl (C++ 成员), 172
esp_ble_gap_cb_param_t::scan_rsp_data_raw_cmpl (C++ 成员), 172
esp_ble_gap_cb_param_t::scan_rst (C++ 成员), 172
esp_ble_gap_cb_param_t::scan_start_cmpl (C++ 成员), 173
esp_ble_gap_cb_param_t::scan_stop_cmpl (C++ 成员), 173
esp_ble_gap_cb_param_t::set_rand_addr_cmpl (C++ 成员), 173
esp_ble_gap_cb_param_t::update_conn_params (C++ 成员), 173
esp_ble_gap_cb_param_t::update_duplicate_exceptional_list_cmpl (C++ 成员), 173
esp_ble_gap_cb_param_t::update_whitelist_cmpl (C++ 成员), 173
esp_ble_gap_clean_duplicate_scan_exceptional_list (C++ 函数), 168
esp_ble_gap_clear_rand_addr (C++ 函数), 164
esp_ble_gap_config_adv_data (C++ 函数), 162
esp_ble_gap_config_adv_data_raw (C++ 函数), 166
esp_ble_gap_config_local_icon (C++ 函数), 164
esp_ble_gap_config_local_privacy (C++ 函数), 164
esp_ble_gap_config_scan_rsp_data_raw (C++ 函数), 166
esp_ble_gap_disconnect (C++ 函数), 170
esp_ble_gap_get_local_used_addr (C++ 函数), 166
esp_ble_gap_get_whitelist_size (C++ 函数), 165
esp_ble_gap_read_rssi (C++ 函数), 167
esp_ble_gap_register_callback (C++ 函数), 161
esp_ble_gap_remove_duplicate_scan_exceptional_device (C++ 函数), 167
esp_ble_gap_security_rsp (C++ 函数), 168
esp_ble_gap_set_device_name (C++ 函数), 165
esp_ble_gap_set_pkt_data_len (C++ 函数), 163
esp_ble_gap_set_prefer_conn_params (C++ 函数), 165
esp_ble_gap_set_rand_addr (C++ 函数), 163
esp_ble_gap_set_scan_params (C++ 函数), 162
esp_ble_gap_set_security_param (C++ 函数), 168
esp_ble_gap_start_advertising (C++ 函数), 163
esp_ble_gap_start_scanning (C++ 函数), 162
esp_ble_gap_stop_advertising (C++ 函数), 163
esp_ble_gap_stop_scanning (C++ 函数), 162
esp_ble_gap_update_conn_params (C++ 函数), 163
esp_ble_gap_update_whitelist (C++ 函数), 164
esp_ble_gattc_app_register (C++ 函数), 230
esp_ble_gattc_app_unregister (C++ 函数), 231
esp_ble_gattc_cache_assoc (C++ 函数), 242
esp_ble_gattc_cache_clean (C++ 函数), 243
esp_ble_gattc_cache_get_addr_list (C++ 函数), 242
esp_ble_gattc_cache_refresh (C++ 函数), 241
esp_ble_gattc_cb_param_t (C++ 类型), 243
esp_ble_gattc_cb_param_t::cfg_mtu (C++ 成员), 243
esp_ble_gattc_cb_param_t::close (C++ 成员), 243
esp_ble_gattc_cb_param_t::congest (C++ 成员), 244
esp_ble_gattc_cb_param_t::connect (C++ 成员), 244
esp_ble_gattc_cb_param_t::dis_srvc_cmpl (C++ 成员), 244
esp_ble_gattc_cb_param_t::disconnect (C++ 成员), 244
esp_ble_gattc_cb_param_t::exec_cmpl (C++ 成员), 243
esp_ble_gattc_cb_param_t::gattc_cfg_mtu_evt_param (C++ 类), 244
esp_ble_gattc_cb_param_t::gattc_cfg_mtu_evt_param::conn_id (C++ 成员), 244

esp_ble_gattc_cb_param_t::gattc_cfg_mtu_evt_param::mtu (C++ 成员), 244
esp_ble_gattc_cb_param_t::gattc_cfg_mtu_evt_param::status (C++ 成员), 244
esp_ble_gattc_cb_param_t::gattc_close_evt_param (C++ 类), 244
esp_ble_gattc_cb_param_t::gattc_close_evt_param::conn_id (C++ 成员), 245
esp_ble_gattc_cb_param_t::gattc_close_evt_param::reason (C++ 成员), 245
esp_ble_gattc_cb_param_t::gattc_close_evt_param::remote_bda (C++ 成员), 245
esp_ble_gattc_cb_param_t::gattc_close_evt_param::status (C++ 成员), 245
esp_ble_gattc_cb_param_t::gattc_congest_evt_param (C++ 类), 245
esp_ble_gattc_cb_param_t::gattc_congest_evt_param::congested (C++ 成员), 245
esp_ble_gattc_cb_param_t::gattc_congest_evt_param::conn_id (C++ 成员), 245
esp_ble_gattc_cb_param_t::gattc_connect_evt_param (C++ 类), 245
esp_ble_gattc_cb_param_t::gattc_connect_evt_param::conn_id (C++ 成员), 245
esp_ble_gattc_cb_param_t::gattc_connect_evt_param::link_role (C++ 成员), 245
esp_ble_gattc_cb_param_t::gattc_connect_evt_param::remote_bda (C++ 成员), 245
esp_ble_gattc_cb_param_t::gattc_dis_srvc_cmpl_evt_param (C++ 类), 245
esp_ble_gattc_cb_param_t::gattc_dis_srvc_cmpl_evt_param::conn_id (C++ 成员), 245
esp_ble_gattc_cb_param_t::gattc_dis_srvc_cmpl_evt_param::status (C++ 成员), 245
esp_ble_gattc_cb_param_t::gattc_disconnect_evt_param (C++ 类), 246
esp_ble_gattc_cb_param_t::gattc_disconnect_evt_param::conn_id (C++ 成员), 246
esp_ble_gattc_cb_param_t::gattc_disconnect_evt_param::reason (C++ 成员), 246
esp_ble_gattc_cb_param_t::gattc_disconnect_evt_param::remote_bda (C++ 成员), 246
esp_ble_gattc_cb_param_t::gattc_exec_cmpl_evt_param (C++ 类), 246
esp_ble_gattc_cb_param_t::gattc_exec_cmpl_evt_param::conn_id (C++ 成员), 246
esp_ble_gattc_cb_param_t::gattc_exec_cmpl_evt_param::status (C++ 成员), 246
esp_ble_gattc_cb_param_t::gattc_get_addr_list_evt_param (C++ 类), 246
esp_ble_gattc_cb_param_t::gattc_get_addr_list_evt_param::addr_list (C++ 成员), 246
esp_ble_gattc_cb_param_t::gattc_get_addr_list_evt_param::num_addr (C++ 成员), 246
esp_ble_gattc_cb_param_t::gattc_get_addr_list_evt_param::status (C++ 成员), 246
esp_ble_gattc_cb_param_t::gattc_notify_evt_param (C++ 类), 246
esp_ble_gattc_cb_param_t::gattc_notify_evt_param::conn_id (C++ 成员), 247
esp_ble_gattc_cb_param_t::gattc_notify_evt_param::handle (C++ 成员), 247
esp_ble_gattc_cb_param_t::gattc_notify_evt_param::is_notify (C++ 成员), 247
esp_ble_gattc_cb_param_t::gattc_notify_evt_param::remote_bda (C++ 成员), 247
esp_ble_gattc_cb_param_t::gattc_notify_evt_param::value (C++ 成员), 247
esp_ble_gattc_cb_param_t::gattc_notify_evt_param::value_len (C++ 成员), 247
esp_ble_gattc_cb_param_t::gattc_open_evt_param (C++ 类), 247
esp_ble_gattc_cb_param_t::gattc_open_evt_param::conn_id (C++ 成员), 247
esp_ble_gattc_cb_param_t::gattc_open_evt_param::mtu (C++ 成员), 247
esp_ble_gattc_cb_param_t::gattc_open_evt_param::remote_bda (C++ 成员), 247
esp_ble_gattc_cb_param_t::gattc_open_evt_param::status (C++ 成员), 247
esp_ble_gattc_cb_param_t::gattc_queue_full_evt_param (C++ 类), 247
esp_ble_gattc_cb_param_t::gattc_queue_full_evt_param::conn_id (C++ 成员), 247

esp_ble_gattc_cb_param_t::gattc_queue_full_evt_param::gatt_if (C++ 成员), 247
esp_ble_gattc_cb_param_t::gattc_queue_full_evt_param::gattc_status (C++ 成员), 247
esp_ble_gattc_cb_param_t::gattc_read_char_evt_param (C++ 类), 248
esp_ble_gattc_cb_param_t::gattc_read_char_evt_param::conn_id (C++ 成员), 248
esp_ble_gattc_cb_param_t::gattc_read_char_evt_param::handle (C++ 成员), 248
esp_ble_gattc_cb_param_t::gattc_read_char_evt_param::status (C++ 成员), 248
esp_ble_gattc_cb_param_t::gattc_read_char_evt_param::value (C++ 成员), 248
esp_ble_gattc_cb_param_t::gattc_read_char_evt_param::value_len (C++ 成员), 248
esp_ble_gattc_cb_param_t::gattc_reg_evt_param (C++ 类), 248
esp_ble_gattc_cb_param_t::gattc_reg_evt_param::app_id (C++ 成员), 248
esp_ble_gattc_cb_param_t::gattc_reg_evt_param::status (C++ 成员), 248
esp_ble_gattc_cb_param_t::gattc_reg_for_notify_evt_param (C++ 类), 248
esp_ble_gattc_cb_param_t::gattc_reg_for_notify_evt_param::gatt_if (C++ 成员), 248
esp_ble_gattc_cb_param_t::gattc_reg_for_notify_evt_param::status (C++ 成员), 248
esp_ble_gattc_cb_param_t::gattc_search_cmpl_evt_param (C++ 类), 248
esp_ble_gattc_cb_param_t::gattc_search_cmpl_evt_param::conn_id (C++ 成员), 249
esp_ble_gattc_cb_param_t::gattc_search_cmpl_evt_param::searched_service_source (C++ 成员), 249
esp_ble_gattc_cb_param_t::gattc_search_cmpl_evt_param::status (C++ 成员), 249
esp_ble_gattc_cb_param_t::gattc_search_res_evt_param (C++ 类), 249
esp_ble_gattc_cb_param_t::gattc_search_res_evt_param::conn_id (C++ 成员), 249
esp_ble_gattc_cb_param_t::gattc_search_res_evt_param::end_handle (C++ 成员), 249
esp_ble_gattc_cb_param_t::gattc_search_res_evt_param::is_primary (C++ 成员), 249
esp_ble_gattc_cb_param_t::gattc_search_res_evt_param::srvc_id (C++ 成员), 249
esp_ble_gattc_cb_param_t::gattc_search_res_evt_param::start_handle (C++ 成员), 249
esp_ble_gattc_cb_param_t::gattc_set_assoc_addr_cmp_evt_param (C++ 类), 249
esp_ble_gattc_cb_param_t::gattc_set_assoc_addr_cmp_evt_param::gatt_if (C++ 成员), 249
esp_ble_gattc_cb_param_t::gattc_srvc_chg_evt_param (C++ 类), 249
esp_ble_gattc_cb_param_t::gattc_srvc_chg_evt_param::remote_bda (C++ 成员), 250
esp_ble_gattc_cb_param_t::gattc_unreg_for_notify_evt_param (C++ 类), 250
esp_ble_gattc_cb_param_t::gattc_unreg_for_notify_evt_param::app_id (C++ 成员), 250
esp_ble_gattc_cb_param_t::gattc_unreg_for_notify_evt_param::status (C++ 成员), 250
esp_ble_gattc_cb_param_t::gattc_write_evt_param (C++ 类), 250
esp_ble_gattc_cb_param_t::gattc_write_evt_param::conn_id (C++ 成员), 250
esp_ble_gattc_cb_param_t::gattc_write_evt_param::handle (C++ 成员), 250
esp_ble_gattc_cb_param_t::gattc_write_evt_param::offset (C++ 成员), 250
esp_ble_gattc_cb_param_t::gattc_write_evt_param::status (C++ 成员), 250
esp_ble_gattc_cb_param_t::get_addr_list (C++ 成员), 244
esp_ble_gattc_cb_param_t::indicate (C++ 成员), 243
esp_ble_gattc_cb_param_t::open (C++ 成员), 243
esp_ble_gattc_cb_param_t::queue_full (C++ 成员), 244
esp_ble_gattc_cb_param_t::read (C++ 成员), 243
esp_ble_gattc_cb_param_t::reg (C++ 成员), 243
esp_ble_gattc_cb_param_t::reg_for_notify

(C++ 成员), 244
esp_ble_gattc_cb_param_t::search_cmpl (C++ 成员), 243
esp_ble_gattc_cb_param_t::search_res (C++ 成员), 243
esp_ble_gattc_cb_param_t::set_assoc_cmp (C++ 成员), 244
esp_ble_gattc_cb_param_t::srvc_chg (C++ 成员), 244
esp_ble_gattc_cb_param_t::unreg_for_notify (C++ 成员), 244
esp_ble_gattc_cb_param_t::write (C++ 成员), 243
esp_ble_gattc_close (C++ 函数), 231
esp_ble_gattc_execute_write (C++ 函数), 240
esp_ble_gattc_get_all_char (C++ 函数), 233
esp_ble_gattc_get_all_descr (C++ 函数), 233
esp_ble_gattc_get_attr_count (C++ 函数), 236
esp_ble_gattc_get_char_by_uuid (C++ 函数), 234
esp_ble_gattc_get_db (C++ 函数), 237
esp_ble_gattc_get_descr_by_char_handle (C++ 函数), 235
esp_ble_gattc_get_descr_by_uuid (C++ 函数), 235
esp_ble_gattc_get_include_service (C++ 函数), 236
esp_ble_gattc_get_service (C++ 函数), 232
esp_ble_gattc_open (C++ 函数), 231
esp_ble_gattc_prepare_write (C++ 函数), 239
esp_ble_gattc_prepare_write_char_descr (C++ 函数), 240
esp_ble_gattc_read_char (C++ 函数), 237
esp_ble_gattc_read_char_descr (C++ 函数), 238
esp_ble_gattc_read_multiple (C++ 函数), 238
esp_ble_gattc_register_callback (C++ 函数), 230
esp_ble_gattc_register_for_notify (C++ 函数), 241
esp_ble_gattc_search_service (C++ 函数), 232
esp_ble_gattc_send_mtu_req (C++ 函数), 232
esp_ble_gattc_unregister_for_notify (C++ 函数), 241
esp_ble_gattc_write_char (C++ 函数), 238
esp_ble_gattc_write_char_descr (C++ 函数), 239
esp_ble_gatts_add_char (C++ 函数), 215
esp_ble_gatts_add_char_descr (C++ 函数), 215
esp_ble_gatts_add_included_service (C++ 函数), 214
esp_ble_gatts_app_register (C++ 函数), 213
esp_ble_gatts_app_unregister (C++ 函数), 213
esp_ble_gatts_cb_param_t (C++ 类型), 219
esp_ble_gatts_cb_param_t::add_attr_tab (C++ 成员), 220
esp_ble_gatts_cb_param_t::add_char (C++ 成员), 219
esp_ble_gatts_cb_param_t::add_char_descr (C++ 成员), 219
esp_ble_gatts_cb_param_t::add_incl_srvc (C++ 成员), 219
esp_ble_gatts_cb_param_t::cancel_open (C++ 成员), 220
esp_ble_gatts_cb_param_t::close (C++ 成员), 220
esp_ble_gatts_cb_param_t::conf (C++ 成员), 219
esp_ble_gatts_cb_param_t::congest (C++ 成员), 220
esp_ble_gatts_cb_param_t::connect (C++ 成员), 220
esp_ble_gatts_cb_param_t::create (C++ 成员), 219
esp_ble_gatts_cb_param_t::del (C++ 成员), 219
esp_ble_gatts_cb_param_t::disconnect (C++ 成员), 220
esp_ble_gatts_cb_param_t::exec_write (C++ 成员), 219
esp_ble_gatts_cb_param_t::gatts_add_attr_tab_evt_param (C++ 类), 220
esp_ble_gatts_cb_param_t::gatts_add_attr_tab_evt_param::ha
(C++ 成员), 221
esp_ble_gatts_cb_param_t::gatts_add_attr_tab_evt_param::nu
(C++ 成员), 221

esp_ble_gatts_cb_param_t::gatts_add_attr_tab_evt_param::status (C++ 成员), 220
esp_ble_gatts_cb_param_t::gatts_add_attr_tab_evt_param::svc_inst_id (C++ 成员), 220
esp_ble_gatts_cb_param_t::gatts_add_attr_tab_evt_param::svc_uuid (C++ 成员), 220
esp_ble_gatts_cb_param_t::gatts_add_char_descr_evt_param (C++ 类), 221
esp_ble_gatts_cb_param_t::gatts_add_char_descr_evt_param::attr_handle (C++ 成员), 221
esp_ble_gatts_cb_param_t::gatts_add_char_descr_evt_param::service_handle (C++ 成员), 221
esp_ble_gatts_cb_param_t::gatts_add_char_descr_evt_param::status (C++ 成员), 221
esp_ble_gatts_cb_param_t::gatts_add_char_evt_param (C++ 类), 221
esp_ble_gatts_cb_param_t::gatts_add_char_evt_param::attr_handle (C++ 成员), 221
esp_ble_gatts_cb_param_t::gatts_add_char_evt_param::char_uuid (C++ 成员), 221
esp_ble_gatts_cb_param_t::gatts_add_char_evt_param::service_handle (C++ 成员), 221
esp_ble_gatts_cb_param_t::gatts_add_char_evt_param::status (C++ 成员), 221
esp_ble_gatts_cb_param_t::gatts_add_incl_srvc_evt_param (C++ 类), 221
esp_ble_gatts_cb_param_t::gatts_add_incl_srvc_evt_param::attr_handle (C++ 成员), 222
esp_ble_gatts_cb_param_t::gatts_add_incl_srvc_evt_param::service_handle (C++ 成员), 222
esp_ble_gatts_cb_param_t::gatts_add_incl_srvc_evt_param::status (C++ 成员), 222
esp_ble_gatts_cb_param_t::gatts_cancel_open_evt_param (C++ 类), 222
esp_ble_gatts_cb_param_t::gatts_cancel_open_evt_param::status (C++ 成员), 222
esp_ble_gatts_cb_param_t::gatts_close_evt_param (C++ 类), 222
esp_ble_gatts_cb_param_t::gatts_close_evt_param::conn_id (C++ 成员), 222
esp_ble_gatts_cb_param_t::gatts_close_evt_param::status (C++ 成员), 222
esp_ble_gatts_cb_param_t::gatts_conf_evt_param (C++ 类), 222
esp_ble_gatts_cb_param_t::gatts_conf_evt_param::conn_id (C++ 成员), 222
esp_ble_gatts_cb_param_t::gatts_conf_evt_param::handle (C++ 成员), 222
esp_ble_gatts_cb_param_t::gatts_conf_evt_param::len (C++ 成员), 222
esp_ble_gatts_cb_param_t::gatts_conf_evt_param::status (C++ 成员), 222
esp_ble_gatts_cb_param_t::gatts_conf_evt_param::value (C++ 成员), 222
esp_ble_gatts_cb_param_t::gatts_congest_evt_param (C++ 类), 223
esp_ble_gatts_cb_param_t::gatts_congest_evt_param::congested (C++ 成员), 223
esp_ble_gatts_cb_param_t::gatts_congest_evt_param::conn_id (C++ 成员), 223
esp_ble_gatts_cb_param_t::gatts_connect_evt_param (C++ 类), 223
esp_ble_gatts_cb_param_t::gatts_connect_evt_param::conn_id (C++ 成员), 223
esp_ble_gatts_cb_param_t::gatts_connect_evt_param::conn_params (C++ 成员), 223
esp_ble_gatts_cb_param_t::gatts_connect_evt_param::remote_bda (C++ 成员), 223
esp_ble_gatts_cb_param_t::gatts_create_evt_param (C++ 类), 223
esp_ble_gatts_cb_param_t::gatts_create_evt_param::service_handle (C++ 成员), 223
esp_ble_gatts_cb_param_t::gatts_create_evt_param::service_id (C++ 成员), 223
esp_ble_gatts_cb_param_t::gatts_create_evt_param::status (C++ 成员), 223
esp_ble_gatts_cb_param_t::gatts_delete_evt_param (C++ 类), 223
esp_ble_gatts_cb_param_t::gatts_delete_evt_param::service_handle (C++ 成员), 224
esp_ble_gatts_cb_param_t::gatts_delete_evt_param::status (C++ 成员), 224

esp_ble_gatts_cb_param_t::gatts_disconnect_evt_param (C++ 类), 224
esp_ble_gatts_cb_param_t::gatts_disconnect_evt_param::conn_id (C++ 成员), 224
esp_ble_gatts_cb_param_t::gatts_disconnect_evt_param::reason (C++ 成员), 224
esp_ble_gatts_cb_param_t::gatts_disconnect_evt_param::remote_bda (C++ 成员), 224
esp_ble_gatts_cb_param_t::gatts_exec_write_evt_param (C++ 类), 224
esp_ble_gatts_cb_param_t::gatts_exec_write_evt_param::conn_id (C++ 成员), 224
esp_ble_gatts_cb_param_t::gatts_exec_write_evt_param::exec_write_flag (C++ 成员), 224
esp_ble_gatts_cb_param_t::gatts_exec_write_evt_param::trans_id (C++ 成员), 224
esp_ble_gatts_cb_param_t::gatts_mtu_evt_param (C++ 类), 224
esp_ble_gatts_cb_param_t::gatts_mtu_evt_param::conn_id (C++ 成员), 224
esp_ble_gatts_cb_param_t::gatts_mtu_evt_param::mtu (C++ 成员), 224
esp_ble_gatts_cb_param_t::gatts_open_evt_param (C++ 类), 225
esp_ble_gatts_cb_param_t::gatts_open_evt_param::status (C++ 成员), 225
esp_ble_gatts_cb_param_t::gatts_read_evt_param (C++ 类), 225
esp_ble_gatts_cb_param_t::gatts_read_evt_param::bda (C++ 成员), 225
esp_ble_gatts_cb_param_t::gatts_read_evt_param::conn_id (C++ 成员), 225
esp_ble_gatts_cb_param_t::gatts_read_evt_param::is_long (C++ 成员), 225
esp_ble_gatts_cb_param_t::gatts_read_evt_param::need_rsp (C++ 成员), 225
esp_ble_gatts_cb_param_t::gatts_read_evt_param::offset (C++ 成员), 225
esp_ble_gatts_cb_param_t::gatts_read_evt_param::trans_id (C++ 成员), 225
esp_ble_gatts_cb_param_t::gatts_reg_evt_param (C++ 类), 225
esp_ble_gatts_cb_param_t::gatts_reg_evt_param::app_id (C++ 成员), 225
esp_ble_gatts_cb_param_t::gatts_reg_evt_param::status (C++ 成员), 225
esp_ble_gatts_cb_param_t::gatts_rsp_evt_param (C++ 类), 225
esp_ble_gatts_cb_param_t::gatts_rsp_evt_param::handle (C++ 成员), 226
esp_ble_gatts_cb_param_t::gatts_rsp_evt_param::status (C++ 成员), 226
esp_ble_gatts_cb_param_t::gatts_send_service_change_evt_param (C++ 类), 226
esp_ble_gatts_cb_param_t::gatts_send_service_change_evt_param::status (C++ 成员), 226
esp_ble_gatts_cb_param_t::gatts_set_attr_val_evt_param (C++ 类), 226
esp_ble_gatts_cb_param_t::gatts_set_attr_val_evt_param::attr_handle (C++ 成员), 226
esp_ble_gatts_cb_param_t::gatts_set_attr_val_evt_param::status (C++ 成员), 226
esp_ble_gatts_cb_param_t::gatts_set_attr_val_evt_param::svc_handle (C++ 成员), 226
esp_ble_gatts_cb_param_t::gatts_start_evt_param (C++ 类), 226
esp_ble_gatts_cb_param_t::gatts_start_evt_param::service_handle (C++ 成员), 226
esp_ble_gatts_cb_param_t::gatts_start_evt_param::status (C++ 成员), 226
esp_ble_gatts_cb_param_t::gatts_stop_evt_param (C++ 类), 226
esp_ble_gatts_cb_param_t::gatts_stop_evt_param::service_handle (C++ 成员), 227
esp_ble_gatts_cb_param_t::gatts_stop_evt_param::status (C++ 成员), 227
esp_ble_gatts_cb_param_t::gatts_write_evt_param (C++ 类), 227
esp_ble_gatts_cb_param_t::gatts_write_evt_param::bda (C++ 成员), 227

esp_ble_gatts_cb_param_t::gatts_write_evt_param::conn_id (C++ 成员), 227
esp_ble_gatts_cb_param_t::gatts_write_evt_param::handle (C++ 成员), 227
esp_ble_gatts_cb_param_t::gatts_write_evt_param::is_prep (C++ 成员), 227
esp_ble_gatts_cb_param_t::gatts_write_evt_param::len (C++ 成员), 227
esp_ble_gatts_cb_param_t::gatts_write_evt_param::need_rsp (C++ 成员), 227
esp_ble_gatts_cb_param_t::gatts_write_evt_param::offset (C++ 成员), 227
esp_ble_gatts_cb_param_t::gatts_write_evt_param::trans_id (C++ 成员), 227
esp_ble_gatts_cb_param_t::gatts_write_evt_param::value (C++ 成员), 227
esp_ble_gatts_cb_param_t::mtu (C++ 成员), 219
esp_ble_gatts_cb_param_t::open (C++ 成员), 220
esp_ble_gatts_cb_param_t::read (C++ 成员), 219
esp_ble_gatts_cb_param_t::reg (C++ 成员), 219
esp_ble_gatts_cb_param_t::rsp (C++ 成员), 220
esp_ble_gatts_cb_param_t::service_change (C++ 成员), 220
esp_ble_gatts_cb_param_t::set_attr_val (C++ 成员), 220
esp_ble_gatts_cb_param_t::start (C++ 成员), 220
esp_ble_gatts_cb_param_t::stop (C++ 成员), 220
esp_ble_gatts_cb_param_t::write (C++ 成员), 219
esp_ble_gatts_close (C++ 函数), 218
esp_ble_gatts_create_attr_tab (C++ 函数), 214
esp_ble_gatts_create_service (C++ 函数), 213
esp_ble_gatts_delete_service (C++ 函数), 216
esp_ble_gatts_get_attr_value (C++ 函数), 217
esp_ble_gatts_open (C++ 函数), 218
esp_ble_gatts_register_callback (C++ 函数), 213
esp_ble_gatts_send_indicate (C++ 函数), 216
esp_ble_gatts_send_response (C++ 函数), 217
esp_ble_gatts_send_service_change_indication (C++ 函数), 218
esp_ble_gatts_set_attr_value (C++ 函数), 217
esp_ble_gatts_start_service (C++ 函数), 216
esp_ble_gatts_stop_service (C++ 函数), 216
esp_ble_get_bond_device_list (C++ 函数), 170
esp_ble_get_bond_device_num (C++ 函数), 169
esp_ble_get_current_conn_params (C++ 函数), 171
ESP_BLE_HOST_STATUS_CHECK (C 宏), 411
ESP_BLE_HOST_STATUS_ENABLED (C 宏), 411
ESP_BLE_ID_KEY_MASK (C 宏), 156
esp_ble_io_cap_t (C++ 类型), 190
ESP_BLE_IS_VALID_PARAM (C 宏), 156
esp_ble_key_mask_t (C++ 类型), 156
esp_ble_key_t (C++ 类), 185
esp_ble_key_t::bd_addr (C++ 成员), 185
esp_ble_key_t::key_type (C++ 成员), 185
esp_ble_key_t::p_key_value (C++ 成员), 185
esp_ble_key_type_t (C++ 类型), 190
esp_ble_key_value_t (C++ 类型), 171
esp_ble_key_value_t::lcsrk_key (C++ 成员), 172
esp_ble_key_value_t::lenc_key (C++ 成员), 172
esp_ble_key_value_t::pcsrk_key (C++ 成员), 171
esp_ble_key_value_t::penc_key (C++ 成员), 171
esp_ble_key_value_t::pid_key (C++ 成员), 171
esp_ble_lcsrk_keys (C++ 类), 184
esp_ble_lcsrk_keys::counter (C++ 成员), 184
esp_ble_lcsrk_keys::csrk (C++ 成员), 184
esp_ble_lcsrk_keys::div (C++ 成员), 184
esp_ble_lcsrk_keys::sec_level (C++ 成员), 184
esp_ble_lenc_keys_t (C++ 类), 183
esp_ble_lenc_keys_t::div (C++ 成员), 184
esp_ble_lenc_keys_t::key_size (C++ 成员), 184
esp_ble_lenc_keys_t::ltk (C++ 成员), 184
esp_ble_lenc_keys_t::sec_level (C++ 成员), 184
ESP_BLE_LINK_KEY_MASK (C 宏), 156
esp_ble_local_id_keys_t (C++ 类), 185

esp_ble_local_id_keys_t::dhk (C++ 成员), 186
esp_ble_local_id_keys_t::ir (C++ 成员), 186
esp_ble_local_id_keys_t::irk (C++ 成员), 186
ESP_BLE_MESH_ACTUATOR_BLOCKED_ERROR (C 宏), 544
ESP_BLE_MESH_ACTUATOR_BLOCKED_WARNING (C 宏), 544
ESP_BLE_MESH_ADDR_ALL_NODES (C 宏), 411
ESP_BLE_MESH_ADDR_FRIENDS (C 宏), 411
ESP_BLE_MESH_ADDR_IS_GROUP (C 宏), 413
ESP_BLE_MESH_ADDR_IS_RFU (C 宏), 413
ESP_BLE_MESH_ADDR_IS_UNICAST (C 宏), 413
ESP_BLE_MESH_ADDR_IS_VIRTUAL (C 宏), 413
ESP_BLE_MESH_ADDR_PROXIES (C 宏), 411
ESP_BLE_MESH_ADDR_RELAYS (C 宏), 412
ESP_BLE_MESH_ADDR_TYPE_PUBLIC (C 宏), 415
ESP_BLE_MESH_ADDR_TYPE_RANDOM (C 宏), 415
ESP_BLE_MESH_ADDR_TYPE_RPA_PUBLIC (C 宏), 415
ESP_BLE_MESH_ADDR_TYPE_RPA_RANDOM (C 宏), 415
esp_ble_mesh_addr_type_t (C++ 类型), 431
ESP_BLE_MESH_ADDR_UNASSIGNED (C 宏), 411
ESP_BLE_MESH_BATTERY_LOW_ERROR (C 宏), 543
ESP_BLE_MESH_BATTERY_LOW_WARNING (C 宏), 543
esp_ble_mesh_bd_addr_t (C++ 类型), 430
ESP_BLE_MESH_BEACON_DISABLED (C 宏), 412
ESP_BLE_MESH_BEACON_ENABLED (C 宏), 412
ESP_BLE_MESH_BEEP (C++ 枚举子), 433
esp_ble_mesh_ble_adv_data_t (C++ 类), 407
esp_ble_mesh_ble_adv_data_t::adv_data (C++ 成员), 408
esp_ble_mesh_ble_adv_data_t::adv_data_len (C++ 成员), 408
esp_ble_mesh_ble_adv_data_t::scan_rsp_data (C++ 成员), 408
esp_ble_mesh_ble_adv_data_t::scan_rsp_data_len (C++ 成员), 408
ESP_BLE_MESH_BLE_ADV_INFINITE (C 宏), 416
esp_ble_mesh_ble_adv_param_t (C++ 类), 407
esp_ble_mesh_ble_adv_param_t::adv_type (C++ 成员), 407
esp_ble_mesh_ble_adv_param_t::count (C++ 成员), 407
esp_ble_mesh_ble_adv_param_t::duration (C++ 成员), 407
esp_ble_mesh_ble_adv_param_t::interval (C++ 成员), 407
esp_ble_mesh_ble_adv_param_t::own_addr_type (C++ 成员), 407
esp_ble_mesh_ble_adv_param_t::peer_addr (C++ 成员), 407
esp_ble_mesh_ble_adv_param_t::peer_addr_type (C++ 成员), 407
esp_ble_mesh_ble_adv_param_t::period (C++ 成员), 407
esp_ble_mesh_ble_adv_param_t::priority (C++ 成员), 407
ESP_BLE_MESH_BLE_ADV_PRIO_HIGH (C++ 枚举子), 434
ESP_BLE_MESH_BLE_ADV_PRIO_LOW (C++ 枚举子), 434
esp_ble_mesh_ble_adv_priority_t (C++ 类型), 434
ESP_BLE_MESH_BLINK (C++ 枚举子), 433
esp_ble_mesh_cb_t (C++ 类型), 430
esp_ble_mesh_cb_type_t (C++ 类型), 432
esp_ble_mesh_cfg_app_key_add_t (C++ 类), 472
esp_ble_mesh_cfg_app_key_add_t::app_idx (C++ 成员), 473
esp_ble_mesh_cfg_app_key_add_t::app_key (C++ 成员), 473
esp_ble_mesh_cfg_app_key_add_t::net_idx (C++ 成员), 473
esp_ble_mesh_cfg_app_key_delete_t (C++ 类), 477
esp_ble_mesh_cfg_app_key_delete_t::app_idx (C++ 成员), 478
esp_ble_mesh_cfg_app_key_delete_t::net_idx (C++ 成员), 478
esp_ble_mesh_cfg_app_key_get_t (C++ 类), 470
esp_ble_mesh_cfg_app_key_get_t::net_idx (C++ 成员), 470
esp_ble_mesh_cfg_app_key_list_cb_t (C++ 类), 485
esp_ble_mesh_cfg_app_key_list_cb_t::app_idx

(C++ 成员), 485
esp_ble_mesh_cfg_app_key_list_cb_t::net_idx (C++ 成员), 485
esp_ble_mesh_cfg_app_key_list_cb_t::status (C++ 成员), 485
esp_ble_mesh_cfg_app_key_status_cb_t (C++ 类), 482
esp_ble_mesh_cfg_app_key_status_cb_t::app_idx (C++ 成员), 482
esp_ble_mesh_cfg_app_key_status_cb_t::net_idx (C++ 成员), 482
esp_ble_mesh_cfg_app_key_status_cb_t::status (C++ 成员), 482
esp_ble_mesh_cfg_app_key_update_t (C++ 类), 477
esp_ble_mesh_cfg_app_key_update_t::app_idx (C++ 成员), 477
esp_ble_mesh_cfg_app_key_update_t::app_key (C++ 成员), 477
esp_ble_mesh_cfg_app_key_update_t::net_idx (C++ 成员), 477
esp_ble_mesh_cfg_beacon_set_t (C++ 类), 471
esp_ble_mesh_cfg_beacon_set_t::beacon (C++ 成员), 471
esp_ble_mesh_cfg_beacon_status_cb_t (C++ 类), 479
esp_ble_mesh_cfg_beacon_status_cb_t::beacon (C++ 成员), 480
esp_ble_mesh_cfg_client_cb_event_t (C++ 类型), 492
esp_ble_mesh_cfg_client_cb_param_t (C++ 类), 486
esp_ble_mesh_cfg_client_cb_param_t::error_code (C++ 成员), 486
esp_ble_mesh_cfg_client_cb_param_t::params (C++ 成员), 486
esp_ble_mesh_cfg_client_cb_param_t::status_cb (C++ 成员), 486
esp_ble_mesh_cfg_client_cb_t (C++ 类型), 491
esp_ble_mesh_cfg_client_common_cb_param_t (C++ 类型), 465
esp_ble_mesh_cfg_client_common_cb_param_t::appkey_list (C++ 成员), 466
esp_ble_mesh_cfg_client_common_cb_param_t::appkey_status (C++ 成员), 466
esp_ble_mesh_cfg_client_common_cb_param_t::beacon_status (C++ 成员), 465
esp_ble_mesh_cfg_client_common_cb_param_t::comp_data_statu (C++ 成员), 465
esp_ble_mesh_cfg_client_common_cb_param_t::default_ttl_sta (C++ 成员), 465
esp_ble_mesh_cfg_client_common_cb_param_t::friend_status (C++ 成员), 466
esp_ble_mesh_cfg_client_common_cb_param_t::gatt_proxy_stat (C++ 成员), 465
esp_ble_mesh_cfg_client_common_cb_param_t::heartbeat_pub_s (C++ 成员), 466
esp_ble_mesh_cfg_client_common_cb_param_t::heartbeat_sub_s (C++ 成员), 466
esp_ble_mesh_cfg_client_common_cb_param_t::kr_phase_status (C++ 成员), 466
esp_ble_mesh_cfg_client_common_cb_param_t::lpn_timeout_sta (C++ 成员), 466
esp_ble_mesh_cfg_client_common_cb_param_t::model_app_list (C++ 成员), 466
esp_ble_mesh_cfg_client_common_cb_param_t::model_app_statu (C++ 成员), 466
esp_ble_mesh_cfg_client_common_cb_param_t::model_pub_statu (C++ 成员), 465
esp_ble_mesh_cfg_client_common_cb_param_t::model_sub_list (C++ 成员), 466
esp_ble_mesh_cfg_client_common_cb_param_t::model_sub_statu (C++ 成员), 466
esp_ble_mesh_cfg_client_common_cb_param_t::net_transmit_st (C++ 成员), 466
esp_ble_mesh_cfg_client_common_cb_param_t::netkey_list (C++ 成员), 466
esp_ble_mesh_cfg_client_common_cb_param_t::netkey_status (C++ 成员), 466
esp_ble_mesh_cfg_client_common_cb_param_t::node_identity_s (C++ 成员), 466
esp_ble_mesh_cfg_client_common_cb_param_t::relay_status (C++ 成员), 465
ESP_BLE_MESH_CFG_CLIENT_EVT_MAX (C++ 枚举

子), 492
ESP_BLE_MESH_CFG_CLIENT_GET_STATE_EVT (C++ 枚举子), 492
esp_ble_mesh_cfg_client_get_state_t (C++ 类型), 462
esp_ble_mesh_cfg_client_get_state_t::app_key_get (C++ 成员), 463
esp_ble_mesh_cfg_client_get_state_t::comp_data_get (C++ 成员), 463
esp_ble_mesh_cfg_client_get_state_t::kr_phase_get (C++ 成员), 463
esp_ble_mesh_cfg_client_get_state_t::lpn_polltimeout_get (C++ 成员), 463
esp_ble_mesh_cfg_client_get_state_t::model_pub_get (C++ 成员), 463
esp_ble_mesh_cfg_client_get_state_t::node_identity_get (C++ 成员), 463
esp_ble_mesh_cfg_client_get_state_t::sig_model_sub_get (C++ 成员), 463
esp_ble_mesh_cfg_client_get_state_t::vnd_model_sub_get (C++ 成员), 463
esp_ble_mesh_cfg_client_get_state_t::vnd_model_sub_get (C++ 成员), 463
ESP_BLE_MESH_CFG_CLIENT_PUBLISH_EVT (C++ 枚举子), 492
ESP_BLE_MESH_CFG_CLIENT_SET_STATE_EVT (C++ 枚举子), 492
esp_ble_mesh_cfg_client_set_state_t (C++ 类型), 463
esp_ble_mesh_cfg_client_set_state_t::app_key_add (C++ 成员), 464
esp_ble_mesh_cfg_client_set_state_t::app_key_delete (C++ 成员), 465
esp_ble_mesh_cfg_client_set_state_t::app_key_update (C++ 成员), 465
esp_ble_mesh_cfg_client_set_state_t::beacon_set (C++ 成员), 464
esp_ble_mesh_cfg_client_set_state_t::default_ttl_set (C++ 成员), 464
esp_ble_mesh_cfg_client_set_state_t::friend_set (C++ 成员), 464
esp_ble_mesh_cfg_client_set_state_t::gatt_proxy_set (C++ 成员), 464
esp_ble_mesh_cfg_client_set_state_t::heartbeat_pub_set (C++ 成员), 464
esp_ble_mesh_cfg_client_set_state_t::heartbeat_sub_set (C++ 成员), 464
esp_ble_mesh_cfg_client_set_state_t::kr_phase_set (C++ 成员), 465
esp_ble_mesh_cfg_client_set_state_t::model_app_bind (C++ 成员), 464
esp_ble_mesh_cfg_client_set_state_t::model_app_unbind (C++ 成员), 465
esp_ble_mesh_cfg_client_set_state_t::model_pub_set (C++ 成员), 464
esp_ble_mesh_cfg_client_set_state_t::model_pub_va_set (C++ 成员), 465
esp_ble_mesh_cfg_client_set_state_t::model_sub_add (C++ 成员), 464
esp_ble_mesh_cfg_client_set_state_t::model_sub_delete (C++ 成员), 464
esp_ble_mesh_cfg_client_set_state_t::model_sub_delete_all (C++ 成员), 465
esp_ble_mesh_cfg_client_set_state_t::model_sub_overwrite (C++ 成员), 464
esp_ble_mesh_cfg_client_set_state_t::model_sub_va_add (C++ 成员), 464
esp_ble_mesh_cfg_client_set_state_t::model_sub_va_delete (C++ 成员), 464
esp_ble_mesh_cfg_client_set_state_t::model_sub_va_overwrit (C++ 成员), 464
esp_ble_mesh_cfg_client_set_state_t::net_key_add (C++ 成员), 464
esp_ble_mesh_cfg_client_set_state_t::net_key_delete (C++ 成员), 465
esp_ble_mesh_cfg_client_set_state_t::net_key_update (C++ 成员), 465
esp_ble_mesh_cfg_client_set_state_t::net_transmit_set (C++ 成员), 465
esp_ble_mesh_cfg_client_set_state_t::node_identity_set (C++ 成员), 465
esp_ble_mesh_cfg_client_set_state_t::relay_set

(C++ 成员), 464
ESP_BLE_MESH_CFG_CLIENT_TIMEOUT_EVT (C++ 枚举子), 492
esp_ble_mesh_cfg_comp_data_status_cb_t (C++ 类), 480
esp_ble_mesh_cfg_comp_data_status_cb_t::composition_data (C++ 成员), 480
esp_ble_mesh_cfg_comp_data_status_cb_t::page (C++ 成员), 480
esp_ble_mesh_cfg_composition_data_get_t (C++ 类), 469
esp_ble_mesh_cfg_composition_data_get_t::page (C++ 成员), 469
esp_ble_mesh_cfg_default_ttl_set_t (C++ 类), 471
esp_ble_mesh_cfg_default_ttl_set_t::ttl (C++ 成员), 472
esp_ble_mesh_cfg_default_ttl_status_cb_t (C++ 类), 480
esp_ble_mesh_cfg_default_ttl_status_cb_t::default_ttl (C++ 成员), 480
esp_ble_mesh_cfg_friend_set_t (C++ 类), 472
esp_ble_mesh_cfg_friend_set_t::friend_state (C++ 成员), 472
esp_ble_mesh_cfg_friend_status_cb_t (C++ 类), 483
esp_ble_mesh_cfg_friend_status_cb_t::friend_state (C++ 成员), 483
esp_ble_mesh_cfg_gatt_proxy_set_t (C++ 类), 472
esp_ble_mesh_cfg_gatt_proxy_set_t::gatt_proxy (C++ 成员), 472
esp_ble_mesh_cfg_gatt_proxy_status_cb_t (C++ 类), 480
esp_ble_mesh_cfg_gatt_proxy_status_cb_t::gatt_proxy (C++ 成员), 480
esp_ble_mesh_cfg_hb_pub_status_cb_t (C++ 类), 483
esp_ble_mesh_cfg_hb_pub_status_cb_t::count (C++ 成员), 483
esp_ble_mesh_cfg_hb_pub_status_cb_t::dst (C++ 成员), 483
esp_ble_mesh_cfg_hb_pub_status_cb_t::features (C++ 成员), 483
esp_ble_mesh_cfg_hb_pub_status_cb_t::net_idx (C++ 成员), 483
esp_ble_mesh_cfg_hb_pub_status_cb_t::period (C++ 成员), 483
esp_ble_mesh_cfg_hb_pub_status_cb_t::status (C++ 成员), 483
esp_ble_mesh_cfg_hb_pub_status_cb_t::ttl (C++ 成员), 483
esp_ble_mesh_cfg_hb_sub_status_cb_t (C++ 类), 483
esp_ble_mesh_cfg_hb_sub_status_cb_t::count (C++ 成员), 484
esp_ble_mesh_cfg_hb_sub_status_cb_t::dst (C++ 成员), 484
esp_ble_mesh_cfg_hb_sub_status_cb_t::max_hops (C++ 成员), 484
esp_ble_mesh_cfg_hb_sub_status_cb_t::min_hops (C++ 成员), 484
esp_ble_mesh_cfg_hb_sub_status_cb_t::period (C++ 成员), 484
esp_ble_mesh_cfg_hb_sub_status_cb_t::src (C++ 成员), 483
esp_ble_mesh_cfg_hb_sub_status_cb_t::status (C++ 成员), 483
esp_ble_mesh_cfg_heartbeat_pub_set_t (C++ 类), 479
esp_ble_mesh_cfg_heartbeat_pub_set_t::count (C++ 成员), 479
esp_ble_mesh_cfg_heartbeat_pub_set_t::dst (C++ 成员), 479
esp_ble_mesh_cfg_heartbeat_pub_set_t::feature (C++ 成员), 479
esp_ble_mesh_cfg_heartbeat_pub_set_t::net_idx (C++ 成员), 479
esp_ble_mesh_cfg_heartbeat_pub_set_t::period (C++ 成员), 479
esp_ble_mesh_cfg_heartbeat_pub_set_t::ttl (C++ 成员), 479
esp_ble_mesh_cfg_heartbeat_sub_set_t (C++ 类), 479

esp_ble_mesh_cfg_heartbeat_sub_set_t::dst (C++ 成员), 479
esp_ble_mesh_cfg_heartbeat_sub_set_t::period (C++ 成员), 479
esp_ble_mesh_cfg_heartbeat_sub_set_t::src (C++ 成员), 479
esp_ble_mesh_cfg_kr_phase_get_t (C++ 类), 471
esp_ble_mesh_cfg_kr_phase_get_t::net_idx (C++ 成员), 471
esp_ble_mesh_cfg_kr_phase_set_t (C++ 类), 478
esp_ble_mesh_cfg_kr_phase_set_t::net_idx (C++ 成员), 478
esp_ble_mesh_cfg_kr_phase_set_t::transition (C++ 成员), 478
esp_ble_mesh_cfg_kr_phase_status_cb_t (C++ 类), 486
esp_ble_mesh_cfg_kr_phase_status_cb_t::net_idx (C++ 成员), 486
esp_ble_mesh_cfg_kr_phase_status_cb_t::phase (C++ 成员), 486
esp_ble_mesh_cfg_kr_phase_status_cb_t::status (C++ 成员), 486
esp_ble_mesh_cfg_lpn_polltimeout_get_t (C++ 类), 471
esp_ble_mesh_cfg_lpn_polltimeout_get_t::lpn_addr (C++ 成员), 471
esp_ble_mesh_cfg_lpn_pollto_status_cb_t (C++ 类), 486
esp_ble_mesh_cfg_lpn_pollto_status_cb_t::lpn_addr (C++ 成员), 486
esp_ble_mesh_cfg_lpn_pollto_status_cb_t::poll_timeout (C++ 成员), 486
esp_ble_mesh_cfg_mod_app_status_cb_t (C++ 类), 482
esp_ble_mesh_cfg_mod_app_status_cb_t::app_idx (C++ 成员), 482
esp_ble_mesh_cfg_mod_app_status_cb_t::company_id (C++ 成员), 482
esp_ble_mesh_cfg_mod_app_status_cb_t::element_addr (C++ 成员), 482
esp_ble_mesh_cfg_mod_app_status_cb_t::model_id (C++ 成员), 483
esp_ble_mesh_cfg_mod_app_status_cb_t::status (C++ 成员), 482
esp_ble_mesh_cfg_model_app_bind_t (C++ 类), 473
esp_ble_mesh_cfg_model_app_bind_t::company_id (C++ 成员), 473
esp_ble_mesh_cfg_model_app_bind_t::element_addr (C++ 成员), 473
esp_ble_mesh_cfg_model_app_bind_t::model_app_idx (C++ 成员), 473
esp_ble_mesh_cfg_model_app_bind_t::model_id (C++ 成员), 473
esp_ble_mesh_cfg_model_app_list_cb_t (C++ 类), 485
esp_ble_mesh_cfg_model_app_list_cb_t::app_idx (C++ 成员), 486
esp_ble_mesh_cfg_model_app_list_cb_t::company_id (C++ 成员), 485
esp_ble_mesh_cfg_model_app_list_cb_t::element_addr (C++ 成员), 485
esp_ble_mesh_cfg_model_app_list_cb_t::model_id (C++ 成员), 486
esp_ble_mesh_cfg_model_app_list_cb_t::status (C++ 成员), 485
esp_ble_mesh_cfg_model_app_unbind_t (C++ 类), 478
esp_ble_mesh_cfg_model_app_unbind_t::company_id (C++ 成员), 478
esp_ble_mesh_cfg_model_app_unbind_t::element_addr (C++ 成员), 478
esp_ble_mesh_cfg_model_app_unbind_t::model_app_idx (C++ 成员), 478
esp_ble_mesh_cfg_model_app_unbind_t::model_id (C++ 成员), 478
esp_ble_mesh_cfg_model_pub_get_t (C++ 类), 469
esp_ble_mesh_cfg_model_pub_get_t::company_id (C++ 成员), 469
esp_ble_mesh_cfg_model_pub_get_t::element_addr (C++ 成员), 469
esp_ble_mesh_cfg_model_pub_get_t::model_id (C++ 成员), 469

esp_ble_mesh_cfg_model_pub_set_t (C++ 类), 473
esp_ble_mesh_cfg_model_pub_set_t::company_id (C++ 成员), 474
esp_ble_mesh_cfg_model_pub_set_t::cred_flag (C++ 成员), 473
esp_ble_mesh_cfg_model_pub_set_t::element_addr (C++ 成员), 473
esp_ble_mesh_cfg_model_pub_set_t::model_id (C++ 成员), 474
esp_ble_mesh_cfg_model_pub_set_t::publish_addr (C++ 成员), 473
esp_ble_mesh_cfg_model_pub_set_t::publish_app_idx (C++ 成员), 473
esp_ble_mesh_cfg_model_pub_set_t::publish_period (C++ 成员), 473
esp_ble_mesh_cfg_model_pub_set_t::publish_retransmit (C++ 成员), 474
esp_ble_mesh_cfg_model_pub_set_t::publish_ttl (C++ 成员), 473
esp_ble_mesh_cfg_model_pub_status_cb_t (C++ 类), 481
esp_ble_mesh_cfg_model_pub_status_cb_t::app_idx (C++ 成员), 481
esp_ble_mesh_cfg_model_pub_status_cb_t::company_id (C++ 成员), 481
esp_ble_mesh_cfg_model_pub_status_cb_t::cred_flag (C++ 成员), 481
esp_ble_mesh_cfg_model_pub_status_cb_t::element_addr (C++ 成员), 481
esp_ble_mesh_cfg_model_pub_status_cb_t::model_id (C++ 成员), 481
esp_ble_mesh_cfg_model_pub_status_cb_t::period (C++ 成员), 481
esp_ble_mesh_cfg_model_pub_status_cb_t::publish_addr (C++ 成员), 481
esp_ble_mesh_cfg_model_pub_status_cb_t::status (C++ 成员), 481
esp_ble_mesh_cfg_model_pub_status_cb_t::transmit (C++ 成员), 481
esp_ble_mesh_cfg_model_pub_status_cb_t::ttl (C++ 成员), 481
esp_ble_mesh_cfg_model_pub_va_set_t (C++ 类), 476
esp_ble_mesh_cfg_model_pub_va_set_t::company_id (C++ 成员), 476
esp_ble_mesh_cfg_model_pub_va_set_t::cred_flag (C++ 成员), 476
esp_ble_mesh_cfg_model_pub_va_set_t::element_addr (C++ 成员), 476
esp_ble_mesh_cfg_model_pub_va_set_t::label_uuid (C++ 成员), 476
esp_ble_mesh_cfg_model_pub_va_set_t::model_id (C++ 成员), 476
esp_ble_mesh_cfg_model_pub_va_set_t::publish_app_idx (C++ 成员), 476
esp_ble_mesh_cfg_model_pub_va_set_t::publish_period (C++ 成员), 476
esp_ble_mesh_cfg_model_pub_va_set_t::publish_retransmit (C++ 成员), 476
esp_ble_mesh_cfg_model_pub_va_set_t::publish_ttl (C++ 成员), 476
esp_ble_mesh_cfg_model_sub_add_t (C++ 类), 474
esp_ble_mesh_cfg_model_sub_add_t::company_id (C++ 成员), 474
esp_ble_mesh_cfg_model_sub_add_t::element_addr (C++ 成员), 474
esp_ble_mesh_cfg_model_sub_add_t::model_id (C++ 成员), 474
esp_ble_mesh_cfg_model_sub_add_t::sub_addr (C++ 成员), 474
esp_ble_mesh_cfg_model_sub_delete_all_t (C++ 类), 476
esp_ble_mesh_cfg_model_sub_delete_all_t::company_id (C++ 成员), 477
esp_ble_mesh_cfg_model_sub_delete_all_t::element_addr (C++ 成员), 477
esp_ble_mesh_cfg_model_sub_delete_all_t::model_id (C++ 成员), 477
esp_ble_mesh_cfg_model_sub_delete_t (C++ 类), 474
esp_ble_mesh_cfg_model_sub_delete_t::company_id (C++ 成员), 474

esp_ble_mesh_cfg_model_sub_delete_t::element_addr (C++ 成员), 474
esp_ble_mesh_cfg_model_sub_delete_t::model_id (C++ 成员), 474
esp_ble_mesh_cfg_model_sub_delete_t::sub_addr (C++ 成员), 474
esp_ble_mesh_cfg_model_sub_list_cb_t (C++ 类), 484
esp_ble_mesh_cfg_model_sub_list_cb_t::company_id (C++ 成员), 484
esp_ble_mesh_cfg_model_sub_list_cb_t::element_addr (C++ 成员), 484
esp_ble_mesh_cfg_model_sub_list_cb_t::model_id (C++ 成员), 484
esp_ble_mesh_cfg_model_sub_list_cb_t::status (C++ 成员), 484
esp_ble_mesh_cfg_model_sub_list_cb_t::sub_addr (C++ 成员), 484
esp_ble_mesh_cfg_model_sub_overwrite_t (C++ 类), 474
esp_ble_mesh_cfg_model_sub_overwrite_t::company_id (C++ 成员), 475
esp_ble_mesh_cfg_model_sub_overwrite_t::element_addr (C++ 成员), 475
esp_ble_mesh_cfg_model_sub_overwrite_t::model_id (C++ 成员), 475
esp_ble_mesh_cfg_model_sub_overwrite_t::sub_addr (C++ 成员), 475
esp_ble_mesh_cfg_model_sub_status_cb_t (C++ 类), 481
esp_ble_mesh_cfg_model_sub_status_cb_t::company_id (C++ 成员), 482
esp_ble_mesh_cfg_model_sub_status_cb_t::element_addr (C++ 成员), 481
esp_ble_mesh_cfg_model_sub_status_cb_t::model_id (C++ 成员), 482
esp_ble_mesh_cfg_model_sub_status_cb_t::status (C++ 成员), 481
esp_ble_mesh_cfg_model_sub_status_cb_t::sub_addr (C++ 成员), 481
esp_ble_mesh_cfg_model_sub_va_add_t (C++ 类), 475
esp_ble_mesh_cfg_model_sub_va_add_t::company_id (C++ 成员), 475
esp_ble_mesh_cfg_model_sub_va_add_t::element_addr (C++ 成员), 475
esp_ble_mesh_cfg_model_sub_va_add_t::label_uuid (C++ 成员), 475
esp_ble_mesh_cfg_model_sub_va_add_t::model_id (C++ 成员), 475
esp_ble_mesh_cfg_model_sub_va_delete_t (C++ 类), 475
esp_ble_mesh_cfg_model_sub_va_delete_t::company_id (C++ 成员), 475
esp_ble_mesh_cfg_model_sub_va_delete_t::element_addr (C++ 成员), 475
esp_ble_mesh_cfg_model_sub_va_delete_t::label_uuid (C++ 成员), 475
esp_ble_mesh_cfg_model_sub_va_delete_t::model_id (C++ 成员), 475
esp_ble_mesh_cfg_model_sub_va_overwrite_t (C++ 类), 475
esp_ble_mesh_cfg_model_sub_va_overwrite_t::company_id (C++ 成员), 476
esp_ble_mesh_cfg_model_sub_va_overwrite_t::element_addr (C++ 成员), 476
esp_ble_mesh_cfg_model_sub_va_overwrite_t::label_uuid (C++ 成员), 476
esp_ble_mesh_cfg_model_sub_va_overwrite_t::model_id (C++ 成员), 476
esp_ble_mesh_cfg_net_key_add_t (C++ 类), 472
esp_ble_mesh_cfg_net_key_add_t::net_idx (C++ 成员), 472
esp_ble_mesh_cfg_net_key_add_t::net_key (C++ 成员), 472
esp_ble_mesh_cfg_net_key_delete_t (C++ 类), 477
esp_ble_mesh_cfg_net_key_delete_t::net_idx (C++ 成员), 477
esp_ble_mesh_cfg_net_key_list_cb_t (C++ 类), 484
esp_ble_mesh_cfg_net_key_list_cb_t::net_idx (C++ 成员), 485
esp_ble_mesh_cfg_net_key_status_cb_t (C++

类), 482
esp_ble_mesh_cfg_net_key_status_cb_t::net_idx (C++ 成员), 482
esp_ble_mesh_cfg_net_key_status_cb_t::status (C++ 成员), 482
esp_ble_mesh_cfg_net_key_update_t (C++ 类), 477
esp_ble_mesh_cfg_net_key_update_t::net_idx (C++ 成员), 477
esp_ble_mesh_cfg_net_key_update_t::net_key (C++ 成员), 477
esp_ble_mesh_cfg_net_trans_status_cb_t (C++ 类), 484
esp_ble_mesh_cfg_net_trans_status_cb_t::net_trans_count (C++ 成员), 484
esp_ble_mesh_cfg_net_trans_status_cb_t::net_trans_step (C++ 成员), 484
esp_ble_mesh_cfg_net_transmit_set_t (C++ 类), 479
esp_ble_mesh_cfg_net_transmit_set_t::net_transmit (C++ 成员), 479
esp_ble_mesh_cfg_node_id_status_cb_t (C++ 类), 485
esp_ble_mesh_cfg_node_id_status_cb_t::identity (C++ 成员), 485
esp_ble_mesh_cfg_node_id_status_cb_t::net_idx (C++ 成员), 485
esp_ble_mesh_cfg_node_id_status_cb_t::status (C++ 成员), 485
esp_ble_mesh_cfg_node_identity_get_t (C++ 类), 470
esp_ble_mesh_cfg_node_identity_get_t::net_idx (C++ 成员), 470
esp_ble_mesh_cfg_node_identity_set_t (C++ 类), 478
esp_ble_mesh_cfg_node_identity_set_t::identity (C++ 成员), 478
esp_ble_mesh_cfg_node_identity_set_t::net_idx (C++ 成员), 478
esp_ble_mesh_cfg_relay_set_t (C++ 类), 472
esp_ble_mesh_cfg_relay_set_t::relay (C++ 成员), 472
esp_ble_mesh_cfg_relay_set_t::relay_retransmit (C++ 成员), 472
esp_ble_mesh_cfg_relay_status_cb_t (C++ 类), 480
esp_ble_mesh_cfg_relay_status_cb_t::relay (C++ 成员), 480
esp_ble_mesh_cfg_relay_status_cb_t::retransmit (C++ 成员), 480
esp_ble_mesh_cfg_server_cb_event_t (C++ 类型), 492
esp_ble_mesh_cfg_server_cb_param_t (C++ 类), 490
esp_ble_mesh_cfg_server_cb_param_t::ctx (C++ 成员), 490
esp_ble_mesh_cfg_server_cb_param_t::model (C++ 成员), 490
esp_ble_mesh_cfg_server_cb_param_t::value (C++ 成员), 490
esp_ble_mesh_cfg_server_cb_t (C++ 类型), 491
esp_ble_mesh_cfg_server_cb_value_t (C++ 类型), 467
esp_ble_mesh_cfg_server_cb_value_t::state_change (C++ 成员), 467
ESP_BLE_MESH_CFG_SERVER_EVT_MAX (C++ 枚举子), 492
ESP_BLE_MESH_CFG_SERVER_STATE_CHANGE_EVT (C++ 枚举子), 492
esp_ble_mesh_cfg_server_state_change_t (C++ 类型), 466
esp_ble_mesh_cfg_server_state_change_t::appkey_add (C++ 成员), 467
esp_ble_mesh_cfg_server_state_change_t::appkey_delete (C++ 成员), 467
esp_ble_mesh_cfg_server_state_change_t::appkey_update (C++ 成员), 467
esp_ble_mesh_cfg_server_state_change_t::kr_phase_set (C++ 成员), 467
esp_ble_mesh_cfg_server_state_change_t::mod_app_bind (C++ 成员), 467
esp_ble_mesh_cfg_server_state_change_t::mod_app_unbind (C++ 成员), 467
esp_ble_mesh_cfg_server_state_change_t::mod_pub_set

(C++ 成员), 467
esp_ble_mesh_cfg_server_state_change_t::mod_sub_add (C++ 成员), 467
esp_ble_mesh_cfg_server_state_change_t::mod_sub_delete (C++ 成员), 467
esp_ble_mesh_cfg_server_state_change_t::netkey_add (C++ 成员), 467
esp_ble_mesh_cfg_server_state_change_t::netkey_delete (C++ 成员), 467
esp_ble_mesh_cfg_server_state_change_t::netkey_update (C++ 成员), 467
esp_ble_mesh_cfg_sig_model_app_get_t (C++ 类), 470
esp_ble_mesh_cfg_sig_model_app_get_t::element_addr (C++ 成员), 470
esp_ble_mesh_cfg_sig_model_app_get_t::model_id (C++ 成员), 470
esp_ble_mesh_cfg_sig_model_sub_get_t (C++ 类), 469
esp_ble_mesh_cfg_sig_model_sub_get_t::element_addr (C++ 成员), 470
esp_ble_mesh_cfg_sig_model_sub_get_t::model_id (C++ 成员), 470
esp_ble_mesh_cfg_srv (C++ 类), 468
esp_ble_mesh_cfg_srv::beacon (C++ 成员), 468
esp_ble_mesh_cfg_srv::count (C++ 成员), 468
esp_ble_mesh_cfg_srv::default_ttl (C++ 成员), 468
esp_ble_mesh_cfg_srv::dst (C++ 成员), 468
esp_ble_mesh_cfg_srv::expiry (C++ 成员), 469
esp_ble_mesh_cfg_srv::feature (C++ 成员), 468
esp_ble_mesh_cfg_srv::friend_state (C++ 成员), 468
esp_ble_mesh_cfg_srv::gatt_proxy (C++ 成员), 468
esp_ble_mesh_cfg_srv::heartbeat_pub (C++ 成员), 469
esp_ble_mesh_cfg_srv::heartbeat_recv_cb (C++ 成员), 469
esp_ble_mesh_cfg_srv::heartbeat_sub (C++ 成员), 469
esp_ble_mesh_cfg_srv::max_hops (C++ 成员), 469
esp_ble_mesh_cfg_srv::min_hops (C++ 成员), 469
esp_ble_mesh_cfg_srv::model (C++ 成员), 468
esp_ble_mesh_cfg_srv::net_idx (C++ 成员), 468
esp_ble_mesh_cfg_srv::net_transmit (C++ 成员), 468
esp_ble_mesh_cfg_srv::period (C++ 成员), 468
esp_ble_mesh_cfg_srv::relay (C++ 成员), 468
esp_ble_mesh_cfg_srv::relay_retransmit (C++ 成员), 468
esp_ble_mesh_cfg_srv::src (C++ 成员), 469
esp_ble_mesh_cfg_srv::timer (C++ 成员), 468
esp_ble_mesh_cfg_srv::ttl (C++ 成员), 468
esp_ble_mesh_cfg_srv_t (C++ 类型), 491
ESP_BLE_MESH_CFG_STATUS_CANNOT_BIND (C 宏), 422
ESP_BLE_MESH_CFG_STATUS_CANNOT_REMOVE (C 宏), 422
ESP_BLE_MESH_CFG_STATUS_CANNOT_SET (C 宏), 422
ESP_BLE_MESH_CFG_STATUS_CANNOT_UPDATE (C 宏), 422
ESP_BLE_MESH_CFG_STATUS_FEATURE_NOT_SUPPORTED (C 宏), 422
ESP_BLE_MESH_CFG_STATUS_INSUFFICIENT_RESOURCES (C 宏), 422
ESP_BLE_MESH_CFG_STATUS_INVALID_ADDRESS (C 宏), 421
ESP_BLE_MESH_CFG_STATUS_INVALID_APPKEY (C 宏), 421
ESP_BLE_MESH_CFG_STATUS_INVALID_BINDING (C 宏), 422
ESP_BLE_MESH_CFG_STATUS_INVALID_MODEL (C 宏), 421
ESP_BLE_MESH_CFG_STATUS_INVALID_NETKEY (C 宏), 421
ESP_BLE_MESH_CFG_STATUS_INVALID_PUBLISH_PARAMETERS (C 宏), 422
ESP_BLE_MESH_CFG_STATUS_KEY_INDEX_ALREADY_STORED (C 宏), 422
ESP_BLE_MESH_CFG_STATUS_NOT_A_SUBSCRIBE_MODEL (C 宏), 422

ESP_BLE_MESH_CFG_STATUS_STORAGE_FAILURE (C 宏), 422
ESP_BLE_MESH_CFG_STATUS_SUCCESS (C 宏), 421
esp_ble_mesh_cfg_status_t (C++ 类型), 431
ESP_BLE_MESH_CFG_STATUS_TEMP_UNABLE_TO_CHANGE_STATE (C 宏), 422
ESP_BLE_MESH_CFG_STATUS_UNSPECIFIED_ERROR (C 宏), 422
esp_ble_mesh_cfg_vnd_model_app_get_t (C++ 类), 471
esp_ble_mesh_cfg_vnd_model_app_get_t::company_id (C++ 成员), 471
esp_ble_mesh_cfg_vnd_model_app_get_t::element_addr (C++ 成员), 471
esp_ble_mesh_cfg_vnd_model_app_get_t::model_id (C++ 成员), 471
esp_ble_mesh_cfg_vnd_model_sub_get_t (C++ 类), 470
esp_ble_mesh_cfg_vnd_model_sub_get_t::company_id (C++ 成员), 470
esp_ble_mesh_cfg_vnd_model_sub_get_t::element_addr (C++ 成员), 470
esp_ble_mesh_cfg_vnd_model_sub_get_t::model_id (C++ 成员), 470
esp_ble_mesh_client_common_param_t (C++ 类), 408
esp_ble_mesh_client_common_param_t::ctx (C++ 成员), 409
esp_ble_mesh_client_common_param_t::model (C++ 成员), 409
esp_ble_mesh_client_common_param_t::msg_role (C++ 成员), 409
esp_ble_mesh_client_common_param_t::msg_timeout (C++ 成员), 409
esp_ble_mesh_client_common_param_t::opcode (C++ 成员), 409
esp_ble_mesh_client_model_deinit (C++ 函数), 445
esp_ble_mesh_client_model_init (C++ 函数), 444
ESP_BLE_MESH_CLIENT_MODEL_RECV_PUBLISH_MSG_EVT (C++ 枚举子), 439
esp_ble_mesh_client_model_send_msg (C++ 函数), 445
ESP_BLE_MESH_CLIENT_MODEL_SEND_TIMEOUT_EVT (C++ 枚举子), 439
esp_ble_mesh_client_op_pair_t (C++ 类), 408
esp_ble_mesh_client_op_pair_t::cli_op (C++ 成员), 408
esp_ble_mesh_client_op_pair_t::status_op (C++ 成员), 408
esp_ble_mesh_client_t (C++ 类), 408
esp_ble_mesh_client_t::internal_data (C++ 成员), 408
esp_ble_mesh_client_t::model (C++ 成员), 408
esp_ble_mesh_client_t::msg_role (C++ 成员), 408
esp_ble_mesh_client_t::op_pair (C++ 成员), 408
esp_ble_mesh_client_t::op_pair_size (C++ 成员), 408
esp_ble_mesh_client_t::publish_status (C++ 成员), 408
esp_ble_mesh_comp_t (C++ 类), 404
esp_ble_mesh_comp_t::cid (C++ 成员), 404
esp_ble_mesh_comp_t::element_count (C++ 成员), 404
esp_ble_mesh_comp_t::elements (C++ 成员), 404
esp_ble_mesh_comp_t::pid (C++ 成员), 404
esp_ble_mesh_comp_t::vid (C++ 成员), 404
ESP_BLE_MESH_CONDENSATION_ERROR (C 宏), 543
ESP_BLE_MESH_CONDENSATION_WARNING (C 宏), 543
esp_ble_mesh_config_client_get_state (C++ 函数), 461
esp_ble_mesh_config_client_set_state (C++ 函数), 462
ESP_BLE_MESH_CONFIGURATION_ERROR (C 宏), 543
ESP_BLE_MESH_CONFIGURATION_WARNING (C 宏), 543
esp_ble_mesh_deinit (C++ 函数), 441
ESP_BLE_MESH_DEINIT_MESH_COMP_EVT (C++ 枚举子), 438
esp_ble_mesh_deinit_param_t (C++ 类), 400
esp_ble_mesh_deinit_param_t::erase_flash (C++ 成员), 400

esp_ble_mesh_dev_add_flag_t (C++ 类型), 431
esp_ble_mesh_dev_role_t (C++ 类型), 434
esp_ble_mesh_device_delete_t (C++ 类), 405
esp_ble_mesh_device_delete_t::addr (C++ 成员), 405
esp_ble_mesh_device_delete_t::addr_type (C++ 成员), 405
esp_ble_mesh_device_delete_t::flag (C++ 成员), 405
esp_ble_mesh_device_delete_t::uuid (C++ 成员), 405
ESP_BLE_MESH_DEVICE_DROPPED_ERROR (C 宏), 544
ESP_BLE_MESH_DEVICE_DROPPED_WARNING (C 宏), 544
ESP_BLE_MESH_DEVICE_MOVED_ERROR (C 宏), 544
ESP_BLE_MESH_DEVICE_MOVED_WARNING (C 宏), 544
ESP_BLE_MESH_DEVICE_NAME_MAX_LEN (C 宏), 411
ESP_BLE_MESH_DISPLAY_NUMBER (C++ 枚举子), 433
ESP_BLE_MESH_DISPLAY_STRING (C++ 枚举子), 433
esp_ble_mesh_elem_t (C++ 类), 400
esp_ble_mesh_elem_t::element_addr (C++ 成员), 400
esp_ble_mesh_elem_t::location (C++ 成员), 400
esp_ble_mesh_elem_t::sig_model_count (C++ 成员), 400
esp_ble_mesh_elem_t::sig_models (C++ 成员), 401
esp_ble_mesh_elem_t::vnd_model_count (C++ 成员), 401
esp_ble_mesh_elem_t::vnd_models (C++ 成员), 401
ESP_BLE_MESH_ELEMENT (C 宏), 414
ESP_BLE_MESH_ELEMENT_NOT_CALIBRATED_ERROR (C 宏), 543
ESP_BLE_MESH_ELEMENT_NOT_CALIBRATED_WARNING (C 宏), 543
ESP_BLE_MESH_EMPTY_ERROR (C 宏), 544
ESP_BLE_MESH_EMPTY_WARNING (C 宏), 544
ESP_BLE_MESH_ENTER_NUMBER (C++ 枚举子), 433
ESP_BLE_MESH_ENTER_STRING (C++ 枚举子), 433
esp_ble_mesh_fast_prov_action_t (C++ 类型), 434
esp_ble_mesh_fast_prov_info_t (C++ 类), 406
esp_ble_mesh_fast_prov_info_t::flags (C++ 成员), 406
esp_ble_mesh_fast_prov_info_t::iv_index (C++ 成员), 407
esp_ble_mesh_fast_prov_info_t::match_len (C++ 成员), 407
esp_ble_mesh_fast_prov_info_t::match_val (C++ 成员), 407
esp_ble_mesh_fast_prov_info_t::net_idx (C++ 成员), 406
esp_ble_mesh_fast_prov_info_t::offset (C++ 成员), 407
esp_ble_mesh_fast_prov_info_t::unicast_max (C++ 成员), 406
esp_ble_mesh_fast_prov_info_t::unicast_min (C++ 成员), 406
ESP_BLE_MESH_FEATURE_ALL_SUPPORTED (C 宏), 412
ESP_BLE_MESH_FEATURE_FRIEND (C 宏), 412
ESP_BLE_MESH_FEATURE_LOW_POWER (C 宏), 412
ESP_BLE_MESH_FEATURE_PROXY (C 宏), 412
ESP_BLE_MESH_FEATURE_RELAY (C 宏), 412
esp_ble_mesh_find_element (C++ 函数), 442
esp_ble_mesh_find_sig_model (C++ 函数), 442
esp_ble_mesh_find_vendor_model (C++ 函数), 442
ESP_BLE_MESH_FRIEND_DISABLED (C 宏), 412
ESP_BLE_MESH_FRIEND_ENABLED (C 宏), 412
ESP_BLE_MESH_FRIEND_FRIENDSHIP_ESTABLISH_EVT (C++ 枚举子), 437
ESP_BLE_MESH_FRIEND_FRIENDSHIP_TERMINATE_EVT (C++ 枚举子), 438
ESP_BLE_MESH_FRIEND_NOT_SUPPORTED (C 宏), 412
ESP_BLE_MESH_GATT_PROXY_DISABLED (C 宏), 412
ESP_BLE_MESH_GATT_PROXY_ENABLED (C 宏), 412
ESP_BLE_MESH_GATT_PROXY_NOT_SUPPORTED (C 宏), 412
ESP_BLE_MESH_GEN_ADMIN_ACCESS_READ (C++ 枚举子), 533
ESP_BLE_MESH_GEN_ADMIN_ACCESS_READ_WRITE (C++ 枚举子), 533
ESP_BLE_MESH_GEN_ADMIN_ACCESS_WRITE (C++ 枚

举子), 533
ESP_BLE_MESH_GEN_ADMIN_NOT_USER_PROP (C++ 枚举子), 533
esp_ble_mesh_gen_admin_prop_access_t (C++ 类型), 533
esp_ble_mesh_gen_admin_prop_srv_t (C++ 类), 516
esp_ble_mesh_gen_admin_prop_srv_t::model (C++ 成员), 516
esp_ble_mesh_gen_admin_prop_srv_t::properties (C++ 成员), 516
esp_ble_mesh_gen_admin_prop_srv_t::property_count (C++ 成员), 516
esp_ble_mesh_gen_admin_prop_srv_t::rsp_ctrl (C++ 成员), 516
esp_ble_mesh_gen_admin_properties_status_cb_t (C++ 类), 508
esp_ble_mesh_gen_admin_properties_status_cb_t::property_ids (C++ 成员), 508
esp_ble_mesh_gen_admin_property_get_t (C++ 类), 503
esp_ble_mesh_gen_admin_property_get_t::property_id (C++ 成员), 503
esp_ble_mesh_gen_admin_property_set_t (C++ 类), 503
esp_ble_mesh_gen_admin_property_set_t::property_id (C++ 成员), 503
esp_ble_mesh_gen_admin_property_set_t::property_value (C++ 成员), 503
esp_ble_mesh_gen_admin_property_set_t::user_access (C++ 成员), 503
esp_ble_mesh_gen_admin_property_status_cb_t (C++ 类), 508
esp_ble_mesh_gen_admin_property_status_cb_t::op_en (C++ 成员), 508
esp_ble_mesh_gen_admin_property_status_cb_t::property_id (C++ 成员), 508
esp_ble_mesh_gen_admin_property_status_cb_t::property_value (C++ 成员), 508
esp_ble_mesh_gen_admin_property_status_cb_t::user_access (C++ 成员), 508
esp_ble_mesh_gen_battery_srv_t (C++ 类), 514
esp_ble_mesh_gen_battery_srv_t::model (C++ 成员), 514
esp_ble_mesh_gen_battery_srv_t::rsp_ctrl (C++ 成员), 514
esp_ble_mesh_gen_battery_srv_t::state (C++ 成员), 514
esp_ble_mesh_gen_battery_state_t (C++ 类), 513
esp_ble_mesh_gen_battery_state_t::battery_flags (C++ 成员), 514
esp_ble_mesh_gen_battery_state_t::battery_level (C++ 成员), 514
esp_ble_mesh_gen_battery_state_t::time_to_charge (C++ 成员), 514
esp_ble_mesh_gen_battery_state_t::time_to_discharge (C++ 成员), 514
esp_ble_mesh_gen_battery_status_cb_t (C++ 类), 506
esp_ble_mesh_gen_battery_status_cb_t::battery_level (C++ 成员), 506
esp_ble_mesh_gen_battery_status_cb_t::flags (C++ 成员), 506
esp_ble_mesh_gen_battery_status_cb_t::time_to_charge (C++ 成员), 506
esp_ble_mesh_gen_battery_status_cb_t::time_to_discharge (C++ 成员), 506
esp_ble_mesh_gen_client_prop_srv_t (C++ 类), 517
esp_ble_mesh_gen_client_prop_srv_t::id_count (C++ 成员), 517
esp_ble_mesh_gen_client_prop_srv_t::model (C++ 成员), 517
esp_ble_mesh_gen_client_prop_srv_t::property_ids (C++ 成员), 517
esp_ble_mesh_gen_client_prop_srv_t::rsp_ctrl (C++ 成员), 517
esp_ble_mesh_gen_client_properties_get_t (C++ 类), 504
esp_ble_mesh_gen_client_properties_get_t::property_id (C++ 成员), 504
esp_ble_mesh_gen_client_properties_status_cb_t (C++ 类), 509

esp_ble_mesh_gen_client_properties_status_cb_t::properties (C++ 成员), 509
esp_ble_mesh_gen_client_status_cb_t (C++ 类型), 495
esp_ble_mesh_gen_client_status_cb_t::admin_properties_status (C++ 成员), 496
esp_ble_mesh_gen_client_status_cb_t::admin_property_status (C++ 成员), 496
esp_ble_mesh_gen_client_status_cb_t::battery_status (C++ 成员), 496
esp_ble_mesh_gen_client_status_cb_t::client_properties_status (C++ 成员), 496
esp_ble_mesh_gen_client_status_cb_t::def_trans_time_status (C++ 成员), 495
esp_ble_mesh_gen_client_status_cb_t::level_status (C++ 成员), 495
esp_ble_mesh_gen_client_status_cb_t::location_global_status (C++ 成员), 496
esp_ble_mesh_gen_client_status_cb_t::location_local_status (C++ 成员), 496
esp_ble_mesh_gen_client_status_cb_t::manufacturer_properties_status (C++ 成员), 496
esp_ble_mesh_gen_client_status_cb_t::manufacturer_property_status (C++ 成员), 496
esp_ble_mesh_gen_client_status_cb_t::onoff_status (C++ 成员), 495
esp_ble_mesh_gen_client_status_cb_t::onpowerup_status (C++ 成员), 495
esp_ble_mesh_gen_client_status_cb_t::power_default_status (C++ 成员), 495
esp_ble_mesh_gen_client_status_cb_t::power_last_status (C++ 成员), 495
esp_ble_mesh_gen_client_status_cb_t::power_level_status (C++ 成员), 495
esp_ble_mesh_gen_client_status_cb_t::power_range_status (C++ 成员), 495
esp_ble_mesh_gen_client_status_cb_t::user_properties_status (C++ 成员), 496
esp_ble_mesh_gen_client_status_cb_t::user_property_status (C++ 成员), 496
esp_ble_mesh_gen_def_trans_time_set_t (C++ 类), 500
esp_ble_mesh_gen_def_trans_time_set_t::property_id (C++ 成员), 501
esp_ble_mesh_gen_def_trans_time_set_t::trans_time (C++ 成员), 501
esp_ble_mesh_gen_def_trans_time_srv_t (C++ 类), 511
esp_ble_mesh_gen_def_trans_time_srv_t::model (C++ 成员), 511
esp_ble_mesh_gen_def_trans_time_srv_t::rsp_ctrl (C++ 成员), 511
esp_ble_mesh_gen_def_trans_time_srv_t::state (C++ 成员), 511
esp_ble_mesh_gen_def_trans_time_state_t (C++ 类), 511
esp_ble_mesh_gen_def_trans_time_state_t::trans_time (C++ 成员), 511
esp_ble_mesh_gen_def_trans_time_status_cb_t (C++ 类), 505
esp_ble_mesh_gen_def_trans_time_status_cb_t::trans_time (C++ 成员), 505
esp_ble_mesh_gen_delta_set_t (C++ 类), 500
esp_ble_mesh_gen_delta_set_t::delay (C++ 成员), 500
esp_ble_mesh_gen_delta_set_t::level (C++ 成员), 500
esp_ble_mesh_gen_delta_set_t::op_en (C++ 成员), 500
esp_ble_mesh_gen_delta_set_t::tid (C++ 成员), 500
esp_ble_mesh_gen_delta_set_t::trans_time (C++ 成员), 500
esp_ble_mesh_gen_level_set_t (C++ 类), 499
esp_ble_mesh_gen_level_set_t::delay (C++ 成员), 500
esp_ble_mesh_gen_level_set_t::level (C++ 成员), 499
esp_ble_mesh_gen_level_set_t::op_en (C++ 成员), 499
esp_ble_mesh_gen_level_set_t::tid (C++ 成员), 499
esp_ble_mesh_gen_level_set_t::trans_time (C++ 成员), 499
esp_ble_mesh_gen_level_srv_t (C++ 类), 510
esp_ble_mesh_gen_level_srv_t::last (C++ 成

员), 511
esp_ble_mesh_gen_level_srv_t::model (C++ 成员), 511
esp_ble_mesh_gen_level_srv_t::rsp_ctrl (C++ 成员), 511
esp_ble_mesh_gen_level_srv_t::state (C++ 成员), 511
esp_ble_mesh_gen_level_srv_t::transition (C++ 成员), 511
esp_ble_mesh_gen_level_srv_t::tt_delta_level (C++ 成员), 511
esp_ble_mesh_gen_level_state_t (C++ 类), 510
esp_ble_mesh_gen_level_state_t::last_delta (C++ 成员), 510
esp_ble_mesh_gen_level_state_t::last_level (C++ 成员), 510
esp_ble_mesh_gen_level_state_t::level (C++ 成员), 510
esp_ble_mesh_gen_level_state_t::move_start (C++ 成员), 510
esp_ble_mesh_gen_level_state_t::positive (C++ 成员), 510
esp_ble_mesh_gen_level_state_t::target_level (C++ 成员), 510
esp_ble_mesh_gen_level_status_cb_t (C++ 类), 504
esp_ble_mesh_gen_level_status_cb_t::op_en (C++ 成员), 504
esp_ble_mesh_gen_level_status_cb_t::present_level (C++ 成员), 504
esp_ble_mesh_gen_level_status_cb_t::remain_time (C++ 成员), 505
esp_ble_mesh_gen_level_status_cb_t::target_level (C++ 成员), 505
esp_ble_mesh_gen_loc_global_set_t (C++ 类), 502
esp_ble_mesh_gen_loc_global_set_t::global_altitude (C++ 成员), 502
esp_ble_mesh_gen_loc_global_set_t::global_latitude (C++ 成员), 502
esp_ble_mesh_gen_loc_global_set_t::global_longitude (C++ 成员), 502
esp_ble_mesh_gen_loc_global_status_cb_t (C++ 类), 506
esp_ble_mesh_gen_loc_global_status_cb_t::global_altitude (C++ 成员), 507
esp_ble_mesh_gen_loc_global_status_cb_t::global_latitude (C++ 成员), 507
esp_ble_mesh_gen_loc_global_status_cb_t::global_longitude (C++ 成员), 507
esp_ble_mesh_gen_loc_local_set_t (C++ 类), 502
esp_ble_mesh_gen_loc_local_set_t::floor_number (C++ 成员), 502
esp_ble_mesh_gen_loc_local_set_t::local_altitude (C++ 成员), 502
esp_ble_mesh_gen_loc_local_set_t::local_east (C++ 成员), 502
esp_ble_mesh_gen_loc_local_set_t::local_north (C++ 成员), 502
esp_ble_mesh_gen_loc_local_set_t::uncertainty (C++ 成员), 502
esp_ble_mesh_gen_loc_local_status_cb_t (C++ 类), 507
esp_ble_mesh_gen_loc_local_status_cb_t::floor_number (C++ 成员), 507
esp_ble_mesh_gen_loc_local_status_cb_t::local_altitude (C++ 成员), 507
esp_ble_mesh_gen_loc_local_status_cb_t::local_east (C++ 成员), 507
esp_ble_mesh_gen_loc_local_status_cb_t::local_north (C++ 成员), 507
esp_ble_mesh_gen_loc_local_status_cb_t::uncertainty (C++ 成员), 507
esp_ble_mesh_gen_location_setup_srv_t (C++ 类), 515
esp_ble_mesh_gen_location_setup_srv_t::model (C++ 成员), 515
esp_ble_mesh_gen_location_setup_srv_t::rsp_ctrl (C++ 成员), 515
esp_ble_mesh_gen_location_setup_srv_t::state (C++ 成员), 515
esp_ble_mesh_gen_location_srv_t (C++ 类), 515
esp_ble_mesh_gen_location_srv_t::model (C++

成员), 515
esp_ble_mesh_gen_location_srv_t::rsp_ctrl (C++ 成员), 515
esp_ble_mesh_gen_location_srv_t::state (C++ 成员), 515
esp_ble_mesh_gen_location_state_t (C++ 类), 514
esp_ble_mesh_gen_location_state_t::floor_number (C++ 成员), 515
esp_ble_mesh_gen_location_state_t::global_altitude (C++ 成员), 514
esp_ble_mesh_gen_location_state_t::global_latitude (C++ 成员), 514
esp_ble_mesh_gen_location_state_t::global_longitude (C++ 成员), 514
esp_ble_mesh_gen_location_state_t::local_altitude (C++ 成员), 515
esp_ble_mesh_gen_location_state_t::local_east (C++ 成员), 514
esp_ble_mesh_gen_location_state_t::local_north (C++ 成员), 514
esp_ble_mesh_gen_location_state_t::uncertainty (C++ 成员), 515
ESP_BLE_MESH_GEN_MANU_ACCESS_READ (C++ 枚举子), 534
ESP_BLE_MESH_GEN_MANU_NOT_USER_PROP (C++ 枚举子), 533
esp_ble_mesh_gen_manu_prop_access_t (C++ 类型), 533
esp_ble_mesh_gen_manu_prop_srv_t (C++ 类), 516
esp_ble_mesh_gen_manu_prop_srv_t::model (C++ 成员), 517
esp_ble_mesh_gen_manu_prop_srv_t::properties (C++ 成员), 517
esp_ble_mesh_gen_manu_prop_srv_t::property_count (C++ 成员), 517
esp_ble_mesh_gen_manu_prop_srv_t::rsp_ctrl (C++ 成员), 517
esp_ble_mesh_gen_manufacturer_properties_status_cb_t (C++ 类), 508
esp_ble_mesh_gen_manufacturer_properties_status_cb_t::properties (C++ 成员), 508
esp_ble_mesh_gen_manufacturer_property_get_t (C++ 类), 503
esp_ble_mesh_gen_manufacturer_property_get_t::property_id (C++ 成员), 503
esp_ble_mesh_gen_manufacturer_property_set_t (C++ 类), 503
esp_ble_mesh_gen_manufacturer_property_set_t::property_id (C++ 成员), 504
esp_ble_mesh_gen_manufacturer_property_set_t::user_access (C++ 成员), 504
esp_ble_mesh_gen_manufacturer_property_status_cb_t (C++ 类), 508
esp_ble_mesh_gen_manufacturer_property_status_cb_t::op_en (C++ 成员), 509
esp_ble_mesh_gen_manufacturer_property_status_cb_t::proper (C++ 成员), 509
esp_ble_mesh_gen_manufacturer_property_status_cb_t::proper (C++ 成员), 509
esp_ble_mesh_gen_manufacturer_property_status_cb_t::user_a (C++ 成员), 509
esp_ble_mesh_gen_move_set_t (C++ 类), 500
esp_ble_mesh_gen_move_set_t::delay (C++ 成员), 500
esp_ble_mesh_gen_move_set_t::delta_level (C++ 成员), 500
esp_ble_mesh_gen_move_set_t::op_en (C++ 成员), 500
esp_ble_mesh_gen_move_set_t::tid (C++ 成员), 500
esp_ble_mesh_gen_move_set_t::trans_time (C++ 成员), 500
esp_ble_mesh_gen_onoff_set_t (C++ 类), 499
esp_ble_mesh_gen_onoff_set_t::delay (C++ 成员), 499
esp_ble_mesh_gen_onoff_set_t::onoff (C++ 成员), 499
esp_ble_mesh_gen_onoff_set_t::op_en (C++ 成员), 499
esp_ble_mesh_gen_onoff_set_t::tid (C++ 成员), 499
esp_ble_mesh_gen_onoff_set_t::trans_time

(C++ 成员), 499
esp_ble_mesh_gen_onoff_srv_t (C++ 类), 509
esp_ble_mesh_gen_onoff_srv_t::last (C++ 成员), 510
esp_ble_mesh_gen_onoff_srv_t::model (C++ 成员), 510
esp_ble_mesh_gen_onoff_srv_t::rsp_ctrl (C++ 成员), 510
esp_ble_mesh_gen_onoff_srv_t::state (C++ 成员), 510
esp_ble_mesh_gen_onoff_srv_t::transition (C++ 成员), 510
esp_ble_mesh_gen_onoff_state_t (C++ 类), 509
esp_ble_mesh_gen_onoff_state_t::onoff (C++ 成员), 509
esp_ble_mesh_gen_onoff_state_t::target_onoff (C++ 成员), 509
esp_ble_mesh_gen_onoff_status_cb_t (C++ 类), 504
esp_ble_mesh_gen_onoff_status_cb_t::op_en (C++ 成员), 504
esp_ble_mesh_gen_onoff_status_cb_t::present_onoff (C++ 成员), 504
esp_ble_mesh_gen_onoff_status_cb_t::remain_time (C++ 成员), 504
esp_ble_mesh_gen_onoff_status_cb_t::target_onoff (C++ 成员), 504
esp_ble_mesh_gen_onpowerup_set_t (C++ 类), 501
esp_ble_mesh_gen_onpowerup_set_t::onpowerup (C++ 成员), 501
esp_ble_mesh_gen_onpowerup_state_t (C++ 类), 511
esp_ble_mesh_gen_onpowerup_state_t::onpowerup (C++ 成员), 512
esp_ble_mesh_gen_onpowerup_status_cb_t (C++ 类), 505
esp_ble_mesh_gen_onpowerup_status_cb_t::onpowerup (C++ 成员), 505
esp_ble_mesh_gen_power_default_set_t (C++ 类), 501
esp_ble_mesh_gen_power_default_set_t::power (C++ 成员), 501
esp_ble_mesh_gen_power_default_status_cb_t (C++ 类), 506
esp_ble_mesh_gen_power_default_status_cb_t::power (C++ 成员), 506
esp_ble_mesh_gen_power_last_status_cb_t (C++ 类), 505
esp_ble_mesh_gen_power_last_status_cb_t::power (C++ 成员), 506
esp_ble_mesh_gen_power_level_set_t (C++ 类), 501
esp_ble_mesh_gen_power_level_set_t::delay (C++ 成员), 501
esp_ble_mesh_gen_power_level_set_t::op_en (C++ 成员), 501
esp_ble_mesh_gen_power_level_set_t::power (C++ 成员), 501
esp_ble_mesh_gen_power_level_set_t::tid (C++ 成员), 501
esp_ble_mesh_gen_power_level_set_t::trans_time (C++ 成员), 501
esp_ble_mesh_gen_power_level_setup_srv_t (C++ 类), 513
esp_ble_mesh_gen_power_level_setup_srv_t::model (C++ 成员), 513
esp_ble_mesh_gen_power_level_setup_srv_t::rsp_ctrl (C++ 成员), 513
esp_ble_mesh_gen_power_level_setup_srv_t::state (C++ 成员), 513
esp_ble_mesh_gen_power_level_srv_t (C++ 类), 513
esp_ble_mesh_gen_power_level_srv_t::last (C++ 成员), 513
esp_ble_mesh_gen_power_level_srv_t::model (C++ 成员), 513
esp_ble_mesh_gen_power_level_srv_t::rsp_ctrl (C++ 成员), 513
esp_ble_mesh_gen_power_level_srv_t::state (C++ 成员), 513
esp_ble_mesh_gen_power_level_srv_t::transition (C++ 成员), 513
esp_ble_mesh_gen_power_level_srv_t::tt_delta_level

(C++ 成员), 513
esp_ble_mesh_gen_power_level_state_t (C++ 类), 512
esp_ble_mesh_gen_power_level_state_t::power_actual (C++ 成员), 512
esp_ble_mesh_gen_power_level_state_t::power_default (C++ 成员), 512
esp_ble_mesh_gen_power_level_state_t::power_last (C++ 成员), 512
esp_ble_mesh_gen_power_level_state_t::power_range_max (C++ 成员), 513
esp_ble_mesh_gen_power_level_state_t::power_range_min (C++ 成员), 513
esp_ble_mesh_gen_power_level_state_t::status_code (C++ 成员), 513
esp_ble_mesh_gen_power_level_state_t::target_power_actual (C++ 成员), 512
esp_ble_mesh_gen_power_level_status_cb_t (C++ 类), 505
esp_ble_mesh_gen_power_level_status_cb_t::op_en (C++ 成员), 505
esp_ble_mesh_gen_power_level_status_cb_t::present_power (C++ 成员), 505
esp_ble_mesh_gen_power_level_status_cb_t::remain_time (C++ 成员), 505
esp_ble_mesh_gen_power_level_status_cb_t::target_power (C++ 成员), 505
esp_ble_mesh_gen_power_onoff_setup_srv_t (C++ 类), 512
esp_ble_mesh_gen_power_onoff_setup_srv_t::model (C++ 成员), 512
esp_ble_mesh_gen_power_onoff_setup_srv_t::rsp_ctrl (C++ 成员), 512
esp_ble_mesh_gen_power_onoff_setup_srv_t::state (C++ 成员), 512
esp_ble_mesh_gen_power_onoff_srv_t (C++ 类), 512
esp_ble_mesh_gen_power_onoff_srv_t::model (C++ 成员), 512
esp_ble_mesh_gen_power_onoff_srv_t::rsp_ctrl (C++ 成员), 512
esp_ble_mesh_gen_power_onoff_srv_t::state (C++ 成员), 512
esp_ble_mesh_gen_power_range_set_t (C++ 类), 501
esp_ble_mesh_gen_power_range_set_t::range_max (C++ 成员), 502
esp_ble_mesh_gen_power_range_set_t::range_min (C++ 成员), 502
esp_ble_mesh_gen_power_range_status_cb_t (C++ 类), 506
esp_ble_mesh_gen_power_range_status_cb_t::range_max (C++ 成员), 506
esp_ble_mesh_gen_power_range_status_cb_t::range_min (C++ 成员), 506
esp_ble_mesh_gen_power_range_status_cb_t::status_code (C++ 成员), 506
ESP_BLE_MESH_GEN_USER_ACCESS_PROHIBIT (C++ 枚举子), 533
ESP_BLE_MESH_GEN_USER_ACCESS_READ (C++ 枚举子), 533
ESP_BLE_MESH_GEN_USER_ACCESS_READ_WRITE (C++ 枚举子), 533
ESP_BLE_MESH_GEN_USER_ACCESS_WRITE (C++ 枚举子), 533
esp_ble_mesh_gen_user_prop_access_t (C++ 类型), 533
esp_ble_mesh_gen_user_prop_srv_t (C++ 类), 516
esp_ble_mesh_gen_user_prop_srv_t::model (C++ 成员), 516
esp_ble_mesh_gen_user_prop_srv_t::properties (C++ 成员), 516
esp_ble_mesh_gen_user_prop_srv_t::property_count (C++ 成员), 516
esp_ble_mesh_gen_user_prop_srv_t::rsp_ctrl (C++ 成员), 516
esp_ble_mesh_gen_user_properties_status_cb_t (C++ 类), 507
esp_ble_mesh_gen_user_properties_status_cb_t::property_ids (C++ 成员), 507
esp_ble_mesh_gen_user_property_get_t (C++ 类), 502
esp_ble_mesh_gen_user_property_get_t::property_id

(C++ 成员), 503
esp_ble_mesh_gen_user_property_set_t (C++ 类), 503
esp_ble_mesh_gen_user_property_set_t::property_id (C++ 成员), 503
esp_ble_mesh_gen_user_property_set_t::property_value (C++ 成员), 503
esp_ble_mesh_gen_user_property_status_cb_t (C++ 类), 507
esp_ble_mesh_gen_user_property_status_cb_t::op_en (C++ 成员), 507
esp_ble_mesh_gen_user_property_status_cb_t::property_id (C++ 成员), 507
esp_ble_mesh_gen_user_property_status_cb_t::property_value (C++ 成员), 508
esp_ble_mesh_gen_user_property_status_cb_t::user_access (C++ 成员), 508
esp_ble_mesh_generic_client_cb_event_t (C++ 类型), 533
esp_ble_mesh_generic_client_cb_param_t (C++ 类), 509
esp_ble_mesh_generic_client_cb_param_t::error_code (C++ 成员), 509
esp_ble_mesh_generic_client_cb_param_t::params (C++ 成员), 509
esp_ble_mesh_generic_client_cb_param_t::status_cb (C++ 成员), 509
esp_ble_mesh_generic_client_cb_t (C++ 类型), 532
ESP_BLE_MESH_GENERIC_CLIENT_EVT_MAX (C++ 枚举子), 533
esp_ble_mesh_generic_client_get_state (C++ 函数), 492
ESP_BLE_MESH_GENERIC_CLIENT_GET_STATE_EVT (C++ 枚举子), 533
esp_ble_mesh_generic_client_get_state_t (C++ 类型), 493
esp_ble_mesh_generic_client_get_state_t::admin_property_get (C++ 成员), 494
esp_ble_mesh_generic_client_get_state_t::client_property_get (C++ 成员), 494
esp_ble_mesh_generic_client_get_state_t::manufacturer_property_get (C++ 成员), 494
esp_ble_mesh_generic_client_get_state_t::user_property_get (C++ 成员), 494
ESP_BLE_MESH_GENERIC_CLIENT_PUBLISH_EVT (C++ 枚举子), 533
esp_ble_mesh_generic_client_set_state (C++ 函数), 493
ESP_BLE_MESH_GENERIC_CLIENT_SET_STATE_EVT (C++ 枚举子), 533
esp_ble_mesh_generic_client_set_state_t (C++ 类型), 494
esp_ble_mesh_generic_client_set_state_t::admin_property_set (C++ 成员), 495
esp_ble_mesh_generic_client_set_state_t::def_trans_time_set (C++ 成员), 494
esp_ble_mesh_generic_client_set_state_t::delta_set (C++ 成员), 494
esp_ble_mesh_generic_client_set_state_t::level_set (C++ 成员), 494
esp_ble_mesh_generic_client_set_state_t::loc_global_set (C++ 成员), 495
esp_ble_mesh_generic_client_set_state_t::loc_local_set (C++ 成员), 495
esp_ble_mesh_generic_client_set_state_t::manufacturer_property_set (C++ 成员), 495
esp_ble_mesh_generic_client_set_state_t::move_set (C++ 成员), 494
esp_ble_mesh_generic_client_set_state_t::onoff_set (C++ 成员), 494
esp_ble_mesh_generic_client_set_state_t::power_default_set (C++ 成员), 494
esp_ble_mesh_generic_client_set_state_t::power_level_set (C++ 成员), 494
esp_ble_mesh_generic_client_set_state_t::power_range_set (C++ 成员), 494
esp_ble_mesh_generic_client_set_state_t::power_set (C++ 成员), 494
esp_ble_mesh_generic_client_set_state_t::user_property_set (C++ 成员), 495
ESP_BLE_MESH_GENERIC_CLIENT_TIMEOUT_EVT (C++ 枚举子), 533
ESP_BLE_MESH_GENERIC_LEVEL_STATE (C++ 枚举

子), 439
esp_ble_mesh_generic_message_opcode_t (C++ 类型), 431
ESP_BLE_MESH_GENERIC_ONOFF_STATE (C++ 枚举子), 439
ESP_BLE_MESH_GENERIC_ONPOWERUP_STATE (C++ 枚举子), 439
ESP_BLE_MESH_GENERIC_POWER_ACTUAL_STATE (C++ 枚举子), 439
esp_ble_mesh_generic_property_t (C++ 类), 515
esp_ble_mesh_generic_property_t::admin_access (C++ 成员), 516
esp_ble_mesh_generic_property_t::id (C++ 成员), 515
esp_ble_mesh_generic_property_t::manu_access (C++ 成员), 516
esp_ble_mesh_generic_property_t::user_access (C++ 成员), 515
esp_ble_mesh_generic_property_t::val (C++ 成员), 516
esp_ble_mesh_generic_server_cb_event_t (C++ 类型), 534
esp_ble_mesh_generic_server_cb_param_t (C++ 类), 525
esp_ble_mesh_generic_server_cb_param_t::ctx (C++ 成员), 525
esp_ble_mesh_generic_server_cb_param_t::model (C++ 成员), 525
esp_ble_mesh_generic_server_cb_param_t::value (C++ 成员), 526
esp_ble_mesh_generic_server_cb_t (C++ 类型), 532
esp_ble_mesh_generic_server_cb_value_t (C++ 类型), 498
esp_ble_mesh_generic_server_cb_value_t::get (C++ 成员), 499
esp_ble_mesh_generic_server_cb_value_t::set (C++ 成员), 499
esp_ble_mesh_generic_server_cb_value_t::state_change (C++ 成员), 499
ESP_BLE_MESH_GENERIC_SERVER_EVT_MAX (C++ 枚举子), 534
ESP_BLE_MESH_GENERIC_SERVER_RECV_GET_MSG_EVT (C++ 枚举子), 534
esp_ble_mesh_generic_server_recv_get_msg_t (C++ 类型), 497
esp_ble_mesh_generic_server_recv_get_msg_t::admin_property (C++ 成员), 497
esp_ble_mesh_generic_server_recv_get_msg_t::client_propert (C++ 成员), 497
esp_ble_mesh_generic_server_recv_get_msg_t::manu_property (C++ 成员), 497
esp_ble_mesh_generic_server_recv_get_msg_t::user_property (C++ 成员), 497
ESP_BLE_MESH_GENERIC_SERVER_RECV_SET_MSG_EVT (C++ 枚举子), 534
esp_ble_mesh_generic_server_recv_set_msg_t (C++ 类型), 497
esp_ble_mesh_generic_server_recv_set_msg_t::admin_property (C++ 成员), 498
esp_ble_mesh_generic_server_recv_set_msg_t::def_trans_time (C++ 成员), 498
esp_ble_mesh_generic_server_recv_set_msg_t::delta (C++ 成员), 498
esp_ble_mesh_generic_server_recv_set_msg_t::level (C++ 成员), 498
esp_ble_mesh_generic_server_recv_set_msg_t::location_globa (C++ 成员), 498
esp_ble_mesh_generic_server_recv_set_msg_t::location_local (C++ 成员), 498
esp_ble_mesh_generic_server_recv_set_msg_t::manu_property (C++ 成员), 498
esp_ble_mesh_generic_server_recv_set_msg_t::move (C++ 成员), 498
esp_ble_mesh_generic_server_recv_set_msg_t::onoff (C++ 成员), 498
esp_ble_mesh_generic_server_recv_set_msg_t::onpowerup (C++ 成员), 498
esp_ble_mesh_generic_server_recv_set_msg_t::power_default (C++ 成员), 498
esp_ble_mesh_generic_server_recv_set_msg_t::power_level (C++ 成员), 498
esp_ble_mesh_generic_server_recv_set_msg_t::power_range (C++ 成员), 498

esp_ble_mesh_generic_server_recv_set_msg_t::user_property (C++ 成员), 498
ESP_BLE_MESH_GENERIC_SERVER_STATE_CHANGE_EVT (C++ 枚举子), 534
esp_ble_mesh_generic_server_state_change_t (C++ 类型), 496
esp_ble_mesh_generic_server_state_change_t::addr (C++ 成员), 497
esp_ble_mesh_generic_server_state_change_t::delay (C++ 成员), 496
esp_ble_mesh_generic_server_state_change_t::delta_level (C++ 成员), 496
esp_ble_mesh_generic_server_state_change_t::level (C++ 成员), 496
esp_ble_mesh_generic_server_state_change_t::local_addr (C++ 成员), 497
esp_ble_mesh_generic_server_state_change_t::local_net_idx (C++ 成员), 497
esp_ble_mesh_generic_server_state_change_t::mask (C++ 成员), 497
esp_ble_mesh_generic_server_state_change_t::move_level (C++ 成员), 496
esp_ble_mesh_generic_server_state_change_t::onoff (C++ 成员), 496
esp_ble_mesh_generic_server_state_change_t::ontime (C++ 成员), 497
esp_ble_mesh_generic_server_state_change_t::power_default (C++ 成员), 497
esp_ble_mesh_generic_server_state_change_t::power_level (C++ 成员), 497
esp_ble_mesh_generic_server_state_change_t::power_range (C++ 成员), 497
esp_ble_mesh_generic_server_state_change_t::user_property (C++ 成员), 497
esp_ble_mesh_get_composition_data (C++ 函数), 442
esp_ble_mesh_get_element_count (C++ 函数), 442
esp_ble_mesh_get_fast_prov_app_key (C++ 函数), 451
esp_ble_mesh_get_model_publish_period (C++ 函数), 441
esp_ble_mesh_get_primary_element_address (C++ 函数), 441
ESP_BLE_MESH_GET_PUBLISH_TRANSMIT_COUNT (C 宏), 414
ESP_BLE_MESH_GET_PUBLISH_TRANSMIT_INTERVAL (C 宏), 414
ESP_BLE_MESH_GET_SENSOR_DATA_FORMAT (C 宏), 620
ESP_BLE_MESH_GET_SENSOR_DATA_LENGTH (C 宏), 620
ESP_BLE_MESH_GET_SENSOR_DATA_PROPERTY_ID (C 宏), 620
ESP_BLE_MESH_GET_TRANSMIT_COUNT (C 宏), 413
ESP_BLE_MESH_GET_TRANSMIT_INTERVAL (C 宏), 413
esp_ble_mesh_health_attention_off_cb_t (C++ 类), 542
esp_ble_mesh_health_attention_off_cb_t::model (C++ 成员), 542
esp_ble_mesh_health_attention_on_cb_t (C++ 类), 541
esp_ble_mesh_health_attention_on_cb_t::model (C++ 成员), 542
esp_ble_mesh_health_attention_on_cb_t::time (C++ 成员), 542
esp_ble_mesh_health_attention_set_t (C++ 类), 539
esp_ble_mesh_health_attention_set_t::attention (C++ 成员), 539
esp_ble_mesh_health_attention_status_cb_t (C++ 类), 540
esp_ble_mesh_health_attention_status_cb_t::attention (C++ 成员), 540
esp_ble_mesh_health_client_cb_event_t (C++ 类型), 545
esp_ble_mesh_health_client_cb_param_t (C++ 类), 540
esp_ble_mesh_health_client_cb_param_t::error_code (C++ 成员), 541
esp_ble_mesh_health_client_cb_param_t::params (C++ 成员), 541
esp_ble_mesh_health_client_cb_param_t::status_cb (C++ 成员), 541

esp_ble_mesh_health_client_cb_t (C++ 类型), 545
esp_ble_mesh_health_client_common_cb_param_t (C++ 类型), 536
esp_ble_mesh_health_client_common_cb_param_t::[illegible] (C++ 成员), 537
esp_ble_mesh_health_client_common_cb_param_t::[illegible] (C++ 成员), 537
esp_ble_mesh_health_client_common_cb_param_t::[illegible] (C++ 成员), 537
esp_ble_mesh_health_client_common_cb_param_t::[illegible] (C++ 成员), 537
ESP_BLE_MESH_HEALTH_CLIENT_EVT_MAX (C++ 枚举子), 545
esp_ble_mesh_health_client_get_state (C++ 函数), 535
ESP_BLE_MESH_HEALTH_CLIENT_GET_STATE_EVT (C++ 枚举子), 545
esp_ble_mesh_health_client_get_state_t (C++ 类型), 536
esp_ble_mesh_health_client_get_state_t::fault_get (C++ 成员), 536
ESP_BLE_MESH_HEALTH_CLIENT_PUBLISH_EVT (C++ 枚举子), 545
esp_ble_mesh_health_client_set_state (C++ 函数), 535
ESP_BLE_MESH_HEALTH_CLIENT_SET_STATE_EVT (C++ 枚举子), 545
esp_ble_mesh_health_client_set_state_t (C++ 类型), 536
esp_ble_mesh_health_client_set_state_t::attention_set (C++ 成员), 536
esp_ble_mesh_health_client_set_state_t::fault_clear (C++ 成员), 536
esp_ble_mesh_health_client_set_state_t::fault_test (C++ 成员), 536
esp_ble_mesh_health_client_set_state_t::period_set (C++ 成员), 536
ESP_BLE_MESH_HEALTH_CLIENT_TIMEOUT_EVT (C++ 枚举子), 545
esp_ble_mesh_health_current_status_cb_t (C++ 类), 539
esp_ble_mesh_health_current_status_cb_t::company_id (C++ 成员), 540
esp_ble_mesh_health_current_status_cb_t::fault_array (C++ 成员), 540
esp_ble_mesh_health_current_status_cb_t::test_id (C++ 成员), 540
ESP_BLE_MESH_HEALTH_FAULT_ARRAY_SIZE (C 宏), 544
esp_ble_mesh_health_fault_clear_cb_t (C++ 类), 541
esp_ble_mesh_health_fault_clear_cb_t::company_id (C++ 成员), 541
esp_ble_mesh_health_fault_clear_cb_t::model (C++ 成员), 541
esp_ble_mesh_health_fault_clear_t (C++ 类), 539
esp_ble_mesh_health_fault_clear_t::company_id (C++ 成员), 539
esp_ble_mesh_health_fault_get_t (C++ 类), 538
esp_ble_mesh_health_fault_get_t::company_id (C++ 成员), 539
esp_ble_mesh_health_fault_status_cb_t (C++ 类), 540
esp_ble_mesh_health_fault_status_cb_t::company_id (C++ 成员), 540
esp_ble_mesh_health_fault_status_cb_t::fault_array (C++ 成员), 540
esp_ble_mesh_health_fault_status_cb_t::test_id (C++ 成员), 540
esp_ble_mesh_health_fault_test_cb_t (C++ 类), 541
esp_ble_mesh_health_fault_test_cb_t::company_id (C++ 成员), 541
esp_ble_mesh_health_fault_test_cb_t::model (C++ 成员), 541
esp_ble_mesh_health_fault_test_cb_t::test_id (C++ 成员), 541
esp_ble_mesh_health_fault_test_t (C++ 类), 539
esp_ble_mesh_health_fault_test_t::company_id (C++ 成员), 539
esp_ble_mesh_health_fault_test_t::test_id

(C++ 成员), 539
esp_ble_mesh_health_fault_update_comp_cb_t (C++ 类), 541
esp_ble_mesh_health_fault_update_comp_cb_t::element (C++ 成员), 541
esp_ble_mesh_health_fault_update_comp_cb_t::error_code (C++ 成员), 541
esp_ble_mesh_health_model_status_t (C++ 类型), 431
esp_ble_mesh_health_period_set_t (C++ 类), 539
esp_ble_mesh_health_period_set_t::fast_period_divisor (C++ 成员), 539
esp_ble_mesh_health_period_status_cb_t (C++ 类), 540
esp_ble_mesh_health_period_status_cb_t::fast_period_divisor (C++ 成员), 540
ESP_BLE_MESH_HEALTH_PUB_DEFINE (C 宏), 542
ESP_BLE_MESH_HEALTH_SERVER_ATTENTION_OFF_EVT (C++ 枚举子), 546
ESP_BLE_MESH_HEALTH_SERVER_ATTENTION_ON_EVT (C++ 枚举子), 545
esp_ble_mesh_health_server_cb_event_t (C++ 类型), 545
esp_ble_mesh_health_server_cb_param_t (C++ 类型), 537
esp_ble_mesh_health_server_cb_param_t::attention (C++ 成员), 537
esp_ble_mesh_health_server_cb_param_t::attention_on (C++ 成员), 537
esp_ble_mesh_health_server_cb_param_t::fault_clear (C++ 成员), 537
esp_ble_mesh_health_server_cb_param_t::fault_test (C++ 成员), 537
esp_ble_mesh_health_server_cb_param_t::fault_update_comp (C++ 成员), 537
esp_ble_mesh_health_server_cb_t (C++ 类型), 545
ESP_BLE_MESH_HEALTH_SERVER_EVT_MAX (C++ 枚举子), 546
ESP_BLE_MESH_HEALTH_SERVER_FAULT_CLEAR_EVT (C++ 枚举子), 545
ESP_BLE_MESH_HEALTH_SERVER_FAULT_TEST_EVT (C++ 枚举子), 545
esp_ble_mesh_health_server_fault_update (C++ 函数), 535
ESP_BLE_MESH_HEALTH_SERVER_FAULT_UPDATE_COMP_EVT (C++ 枚举子), 545
esp_ble_mesh_health_srv_cb_t (C++ 类), 537
esp_ble_mesh_health_srv_cb_t::attention_off (C++ 成员), 538
esp_ble_mesh_health_srv_cb_t::attention_on (C++ 成员), 537
esp_ble_mesh_health_srv_cb_t::fault_clear (C++ 成员), 537
esp_ble_mesh_health_srv_cb_t::fault_test (C++ 成员), 537
esp_ble_mesh_health_srv_t (C++ 类), 538
esp_ble_mesh_health_srv_t::attention_timer (C++ 成员), 538
esp_ble_mesh_health_srv_t::attention_timer_start (C++ 成员), 538
esp_ble_mesh_health_srv_t::health_cb (C++ 成员), 538
esp_ble_mesh_health_srv_t::health_test (C++ 成员), 538
esp_ble_mesh_health_srv_t::model (C++ 成员), 538
ESP_BLE_MESH_HEALTH_STANDARD_TEST (C 宏), 543
esp_ble_mesh_health_test_t (C++ 类), 538
esp_ble_mesh_health_test_t::company_id (C++ 成员), 538
esp_ble_mesh_health_test_t::current_faults (C++ 成员), 538
esp_ble_mesh_health_test_t::id_count (C++ 成员), 538
esp_ble_mesh_health_test_t::prev_test_id (C++ 成员), 538
esp_ble_mesh_health_test_t::registered_faults (C++ 成员), 538
esp_ble_mesh_health_test_t::test_ids (C++ 成员), 538
ESP_BLE_MESH_HEARTBEAT_MESSAGE_RECV_EVT (C++ 枚举子), 437

ESP_BLE_MESH_HOUSING_OPENED_ERROR (C 宏), 544
ESP_BLE_MESH_HOUSING_OPENED_WARNING (C 宏), 544
esp_ble_mesh_init (C++ 函数), 440
esp_ble_mesh_input_action_t (C++ 类型), 433
ESP_BLE_MESH_INPUT_NO_CHANGE_ERROR (C 宏), 544
ESP_BLE_MESH_INPUT_NO_CHANGE_WARNING (C 宏), 544
ESP_BLE_MESH_INPUT_OOB (C++ 枚举子), 432
ESP_BLE_MESH_INPUT_TOO_HIGH_ERROR (C 宏), 544
ESP_BLE_MESH_INPUT_TOO_HIGH_WARNING (C 宏), 544
ESP_BLE_MESH_INPUT_TOO_LOW_ERROR (C 宏), 544
ESP_BLE_MESH_INPUT_TOO_LOW_WARNING (C 宏), 544
ESP_BLE_MESH_INTERNAL_BUS_ERROR (C 宏), 544
ESP_BLE_MESH_INTERNAL_BUS_WARNING (C 宏), 544
ESP_BLE_MESH_INVALID_NODE_INDEX (C 宏), 413
ESP_BLE_MESH_INVALID_SCENE_NUMBER (C 宏), 649
ESP_BLE_MESH_INVALID_SENSOR_PROPERTY_ID (C 宏), 619
ESP_BLE_MESH_INVALID_SENSOR_SETTING_PROPERTY_ID (C 宏), 619
esp_ble_mesh_is_model_subscribed_to_group (C++ 函数), 441
ESP_BLE_MESH_KEY_ANY (C 宏), 412
ESP_BLE_MESH_KEY_DEV (C 宏), 412
ESP_BLE_MESH_KEY_PRIMARY (C 宏), 412
ESP_BLE_MESH_KEY_UNUSED (C 宏), 412
esp_ble_mesh_last_msg_info_t (C++ 类), 410
esp_ble_mesh_last_msg_info_t::dst (C++ 成员), 410
esp_ble_mesh_last_msg_info_t::src (C++ 成员), 410
esp_ble_mesh_last_msg_info_t::tid (C++ 成员), 410
esp_ble_mesh_last_msg_info_t::timestamp (C++ 成员), 410
ESP_BLE_MESH_LC_FADE (C++ 枚举子), 602
ESP_BLE_MESH_LC_FADE_ON (C++ 枚举子), 602
ESP_BLE_MESH_LC_FADE_STANDBY_AUTO (C++ 枚举子), 602
ESP_BLE_MESH_LC_FADE_STANDBY_MANUAL (C++ 枚举子), 602
ESP_BLE_MESH_LC_OFF (C++ 枚举子), 601
ESP_BLE_MESH_LC_PROLONG (C++ 枚举子), 602
ESP_BLE_MESH_LC_RUN (C++ 枚举子), 602
ESP_BLE_MESH_LC_STANDBY (C++ 枚举子), 602
esp_ble_mesh_lc_state_t (C++ 类型), 601
esp_ble_mesh_light_client_cb_event_t (C++ 类型), 601
esp_ble_mesh_light_client_cb_param_t (C++ 类), 568
esp_ble_mesh_light_client_cb_param_t::error_code (C++ 成员), 569
esp_ble_mesh_light_client_cb_param_t::params (C++ 成员), 569
esp_ble_mesh_light_client_cb_param_t::status_cb (C++ 成员), 569
esp_ble_mesh_light_client_cb_t (C++ 类型), 601
ESP_BLE_MESH_LIGHT_CLIENT_EVT_MAX (C++ 枚举子), 601
esp_ble_mesh_light_client_get_state (C++ 函数), 546
ESP_BLE_MESH_LIGHT_CLIENT_GET_STATE_EVT (C++ 枚举子), 601
esp_ble_mesh_light_client_get_state_t (C++ 类型), 547
esp_ble_mesh_light_client_get_state_t::lc_property_get (C++ 成员), 547
ESP_BLE_MESH_LIGHT_CLIENT_PUBLISH_EVT (C++ 枚举子), 601
esp_ble_mesh_light_client_set_state (C++ 函数), 546
ESP_BLE_MESH_LIGHT_CLIENT_SET_STATE_EVT (C++ 枚举子), 601
esp_ble_mesh_light_client_set_state_t (C++ 类型), 547
esp_ble_mesh_light_client_set_state_t::ctl_default_set (C++ 成员), 548
esp_ble_mesh_light_client_set_state_t::ctl_set (C++ 成员), 547
esp_ble_mesh_light_client_set_state_t::ctl_temperature_ran (C++ 成员), 548

esp_ble_mesh_light_client_set_state_t::ctl_temperature_set (C++ 成员), 548
esp_ble_mesh_light_client_set_state_t::hsl_default_set (C++ 成员), 548
esp_ble_mesh_light_client_set_state_t::hsl_hue_set (C++ 成员), 548
esp_ble_mesh_light_client_set_state_t::hsl_range_set (C++ 成员), 548
esp_ble_mesh_light_client_set_state_t::hsl_saturation_set (C++ 成员), 548
esp_ble_mesh_light_client_set_state_t::hsl_set (C++ 成员), 548
esp_ble_mesh_light_client_set_state_t::lc_light_onoff_set (C++ 成员), 549
esp_ble_mesh_light_client_set_state_t::lc_mode_set (C++ 成员), 548
esp_ble_mesh_light_client_set_state_t::lc_om_set (C++ 成员), 548
esp_ble_mesh_light_client_set_state_t::lc_property_set (C++ 成员), 549
esp_ble_mesh_light_client_set_state_t::lightness_default_set (C++ 成员), 547
esp_ble_mesh_light_client_set_state_t::lightness_linear_set (C++ 成员), 547
esp_ble_mesh_light_client_set_state_t::lightness_range_set (C++ 成员), 547
esp_ble_mesh_light_client_set_state_t::lightness_set (C++ 成员), 547
esp_ble_mesh_light_client_set_state_t::xyl_default_set (C++ 成员), 548
esp_ble_mesh_light_client_set_state_t::xyl_range_set (C++ 成员), 548
esp_ble_mesh_light_client_set_state_t::xyl_set (C++ 成员), 548
esp_ble_mesh_light_client_status_cb_t (C++ 类型), 549
esp_ble_mesh_light_client_status_cb_t::ctl_default_status (C++ 成员), 549
esp_ble_mesh_light_client_status_cb_t::ctl_status (C++ 成员), 549
esp_ble_mesh_light_client_status_cb_t::ctl_temperature_status (C++ 成员), 549
esp_ble_mesh_light_client_status_cb_t::ctl_temperature_status (C++ 成员), 549
esp_ble_mesh_light_client_status_cb_t::hsl_default_status (C++ 成员), 550
esp_ble_mesh_light_client_status_cb_t::hsl_hue_status (C++ 成员), 549
esp_ble_mesh_light_client_status_cb_t::hsl_range_status (C++ 成员), 550
esp_ble_mesh_light_client_status_cb_t::hsl_saturation_status (C++ 成员), 550
esp_ble_mesh_light_client_status_cb_t::hsl_status (C++ 成员), 549
esp_ble_mesh_light_client_status_cb_t::hsl_target_status (C++ 成员), 549
esp_ble_mesh_light_client_status_cb_t::lc_light_onoff_status (C++ 成员), 550
esp_ble_mesh_light_client_status_cb_t::lc_mode_status (C++ 成员), 550
esp_ble_mesh_light_client_status_cb_t::lc_om_status (C++ 成员), 550
esp_ble_mesh_light_client_status_cb_t::lc_property_status (C++ 成员), 550
esp_ble_mesh_light_client_status_cb_t::lightness_default_status (C++ 成员), 549
esp_ble_mesh_light_client_status_cb_t::lightness_last_status (C++ 成员), 549
esp_ble_mesh_light_client_status_cb_t::lightness_linear_status (C++ 成员), 549
esp_ble_mesh_light_client_status_cb_t::lightness_range_status (C++ 成员), 549
esp_ble_mesh_light_client_status_cb_t::lightness_status (C++ 成员), 549
esp_ble_mesh_light_client_status_cb_t::xyl_default_status (C++ 成员), 550
esp_ble_mesh_light_client_status_cb_t::xyl_range_status (C++ 成员), 550
esp_ble_mesh_light_client_status_cb_t::xyl_status (C++ 成员), 550
esp_ble_mesh_light_client_status_cb_t::xyl_target_status (C++ 成员), 550
ESP_BLE_MESH_LIGHT_CLIENT_TIMEOUT_EVT (C++ 枚举子), 601

esp_ble_mesh_light_control_t (C++ 类), 581
esp_ble_mesh_light_control_t::prop_state (C++ 成员), 581
esp_ble_mesh_light_control_t::state (C++ 成员), 581
esp_ble_mesh_light_control_t::state_machine (C++ 成员), 581
esp_ble_mesh_light_ctl_default_set_t (C++ 类), 556
esp_ble_mesh_light_ctl_default_set_t::delta_uv (C++ 成员), 556
esp_ble_mesh_light_ctl_default_set_t::lightness (C++ 成员), 556
esp_ble_mesh_light_ctl_default_set_t::temperature (C++ 成员), 556
esp_ble_mesh_light_ctl_default_status_cb_t (C++ 类), 563
esp_ble_mesh_light_ctl_default_status_cb_t::delta_uv (C++ 成员), 564
esp_ble_mesh_light_ctl_default_status_cb_t::lightness (C++ 成员), 564
esp_ble_mesh_light_ctl_default_status_cb_t::temperature (C++ 成员), 564
ESP_BLE_MESH_LIGHT_CTL_LIGHTNESS_STATE (C++ 枚举子), 439
esp_ble_mesh_light_ctl_set_t (C++ 类), 555
esp_ble_mesh_light_ctl_set_t::ctl_delta_uv (C++ 成员), 555
esp_ble_mesh_light_ctl_set_t::ctl_lightness (C++ 成员), 555
esp_ble_mesh_light_ctl_set_t::ctl_temperatrue (C++ 成员), 555
esp_ble_mesh_light_ctl_set_t::delay (C++ 成员), 555
esp_ble_mesh_light_ctl_set_t::op_en (C++ 成员), 555
esp_ble_mesh_light_ctl_set_t::tid (C++ 成员), 555
esp_ble_mesh_light_ctl_set_t::trans_time (C++ 成员), 555
esp_ble_mesh_light_ctl_setup_srv_t (C++ 类), 572
esp_ble_mesh_light_ctl_setup_srv_t::model (C++ 成员), 572
esp_ble_mesh_light_ctl_setup_srv_t::rsp_ctrl (C++ 成员), 572
esp_ble_mesh_light_ctl_setup_srv_t::state (C++ 成员), 572
esp_ble_mesh_light_ctl_srv_t (C++ 类), 571
esp_ble_mesh_light_ctl_srv_t::last (C++ 成员), 571
esp_ble_mesh_light_ctl_srv_t::model (C++ 成员), 571
esp_ble_mesh_light_ctl_srv_t::rsp_ctrl (C++ 成员), 571
esp_ble_mesh_light_ctl_srv_t::state (C++ 成员), 571
esp_ble_mesh_light_ctl_srv_t::transition (C++ 成员), 571
esp_ble_mesh_light_ctl_srv_t::tt_delta_delta_uv (C++ 成员), 572
esp_ble_mesh_light_ctl_srv_t::tt_delta_lightness (C++ 成员), 572
esp_ble_mesh_light_ctl_srv_t::tt_delta_temperature (C++ 成员), 572
esp_ble_mesh_light_ctl_state_t (C++ 类), 570
esp_ble_mesh_light_ctl_state_t::delta_uv (C++ 成员), 571
esp_ble_mesh_light_ctl_state_t::delta_uv_default (C++ 成员), 571
esp_ble_mesh_light_ctl_state_t::lightness (C++ 成员), 570
esp_ble_mesh_light_ctl_state_t::lightness_default (C++ 成员), 571
esp_ble_mesh_light_ctl_state_t::status_code (C++ 成员), 571
esp_ble_mesh_light_ctl_state_t::target_delta_uv (C++ 成员), 571
esp_ble_mesh_light_ctl_state_t::target_lightness (C++ 成员), 570
esp_ble_mesh_light_ctl_state_t::target_temperature (C++ 成员), 571
esp_ble_mesh_light_ctl_state_t::temperature (C++ 成员), 571

esp_ble_mesh_light_ctl_state_t::temperature_default (C++ 成员), 571
esp_ble_mesh_light_ctl_state_t::temperature_range_max (C++ 成员), 571
esp_ble_mesh_light_ctl_state_t::temperature_range_min (C++ 成员), 571
esp_ble_mesh_light_ctl_status_cb_t (C++ 类), 562
esp_ble_mesh_light_ctl_status_cb_t::op_en (C++ 成员), 562
esp_ble_mesh_light_ctl_status_cb_t::present_ctl_lightness (C++ 成员), 562
esp_ble_mesh_light_ctl_status_cb_t::present_ctl_temperature (C++ 成员), 562
esp_ble_mesh_light_ctl_status_cb_t::remain_time (C++ 成员), 563
esp_ble_mesh_light_ctl_status_cb_t::target_ctl_lightness (C++ 成员), 562
esp_ble_mesh_light_ctl_status_cb_t::target_ctl_temperature (C++ 成员), 563
ESP_BLE_MESH_LIGHT_CTL_TEMP_DELTA_UV_STATE (C++ 枚举子), 439
esp_ble_mesh_light_ctl_temp_srv_t (C++ 类), 572
esp_ble_mesh_light_ctl_temp_srv_t::last (C++ 成员), 572
esp_ble_mesh_light_ctl_temp_srv_t::model (C++ 成员), 572
esp_ble_mesh_light_ctl_temp_srv_t::rsp_ctrl (C++ 成员), 572
esp_ble_mesh_light_ctl_temp_srv_t::state (C++ 成员), 572
esp_ble_mesh_light_ctl_temp_srv_t::transition (C++ 成员), 572
esp_ble_mesh_light_ctl_temp_srv_t::tt_delta_delta_uv (C++ 成员), 572
esp_ble_mesh_light_ctl_temp_srv_t::tt_delta_temperature (C++ 成员), 572
esp_ble_mesh_light_ctl_temperature_range_set_t (C++ 类), 556
esp_ble_mesh_light_ctl_temperature_range_set_t::range_max (C++ 成员), 556
esp_ble_mesh_light_ctl_temperature_range_set_t::range_min (C++ 成员), 556
esp_ble_mesh_light_ctl_temperature_range_status_cb_t (C++ 类), 563
esp_ble_mesh_light_ctl_temperature_range_status_cb_t::range_max (C++ 成员), 563
esp_ble_mesh_light_ctl_temperature_range_status_cb_t::range_min (C++ 成员), 563
esp_ble_mesh_light_ctl_temperature_range_status_cb_t::status_code (C++ 成员), 563
esp_ble_mesh_light_ctl_temperature_set_t (C++ 类), 555
esp_ble_mesh_light_ctl_temperature_set_t::ctl_delta_uv (C++ 成员), 556
esp_ble_mesh_light_ctl_temperature_set_t::ctl_temperature (C++ 成员), 556
esp_ble_mesh_light_ctl_temperature_set_t::delay (C++ 成员), 556
esp_ble_mesh_light_ctl_temperature_set_t::op_en (C++ 成员), 556
esp_ble_mesh_light_ctl_temperature_set_t::tid (C++ 成员), 556
esp_ble_mesh_light_ctl_temperature_set_t::trans_time (C++ 成员), 556
esp_ble_mesh_light_ctl_temperature_status_cb_t (C++ 类), 563
esp_ble_mesh_light_ctl_temperature_status_cb_t::op_en (C++ 成员), 563
esp_ble_mesh_light_ctl_temperature_status_cb_t::present_ctl_delta_uv (C++ 成员), 563
esp_ble_mesh_light_ctl_temperature_status_cb_t::present_ctl_temperature (C++ 成员), 563
esp_ble_mesh_light_ctl_temperature_status_cb_t::remain_time (C++ 成员), 563
esp_ble_mesh_light_ctl_temperature_status_cb_t::target_ctl_delta_uv (C++ 成员), 563
esp_ble_mesh_light_ctl_temperature_status_cb_t::target_ctl_temperature (C++ 成员), 563
esp_ble_mesh_light_hsl_default_set_t (C++ 类), 558
esp_ble_mesh_light_hsl_default_set_t::hue (C++ 成员), 558

esp_ble_mesh_light_hsl_default_set_t::lightness (C++ 成员), 558
esp_ble_mesh_light_hsl_default_set_t::saturation (C++ 成员), 558
esp_ble_mesh_light_hsl_default_status_cb_t (C++ 类), 565
esp_ble_mesh_light_hsl_default_status_cb_t::hue (C++ 成员), 565
esp_ble_mesh_light_hsl_default_status_cb_t::lightness (C++ 成员), 565
esp_ble_mesh_light_hsl_default_status_cb_t::saturation (C++ 成员), 565
esp_ble_mesh_light_hsl_hue_set_t (C++ 类), 557
esp_ble_mesh_light_hsl_hue_set_t::delay (C++ 成员), 557
esp_ble_mesh_light_hsl_hue_set_t::hue (C++ 成员), 557
esp_ble_mesh_light_hsl_hue_set_t::op_en (C++ 成员), 557
esp_ble_mesh_light_hsl_hue_set_t::tid (C++ 成员), 557
esp_ble_mesh_light_hsl_hue_set_t::trans_time (C++ 成员), 557
esp_ble_mesh_light_hsl_hue_srv_t (C++ 类), 574
esp_ble_mesh_light_hsl_hue_srv_t::last (C++ 成员), 575
esp_ble_mesh_light_hsl_hue_srv_t::model (C++ 成员), 575
esp_ble_mesh_light_hsl_hue_srv_t::rsp_ctrl (C++ 成员), 575
esp_ble_mesh_light_hsl_hue_srv_t::state (C++ 成员), 575
esp_ble_mesh_light_hsl_hue_srv_t::transition (C++ 成员), 575
esp_ble_mesh_light_hsl_hue_srv_t::tt_delta_hue (C++ 成员), 575
ESP_BLE_MESH_LIGHT_HSL_HUE_STATE (C++ 枚举子), 439
esp_ble_mesh_light_hsl_hue_status_cb_t (C++ 类), 565
esp_ble_mesh_light_hsl_hue_status_cb_t::op_en (C++ 成员), 565
esp_ble_mesh_light_hsl_hue_status_cb_t::present_hue (C++ 成员), 565
esp_ble_mesh_light_hsl_hue_status_cb_t::remain_time (C++ 成员), 565
esp_ble_mesh_light_hsl_hue_status_cb_t::target_hue (C++ 成员), 565
ESP_BLE_MESH_LIGHT_HSL_LIGHTNESS_STATE (C++ 枚举子), 439
esp_ble_mesh_light_hsl_range_set_t (C++ 类), 558
esp_ble_mesh_light_hsl_range_set_t::hue_range_max (C++ 成员), 558
esp_ble_mesh_light_hsl_range_set_t::hue_range_min (C++ 成员), 558
esp_ble_mesh_light_hsl_range_set_t::saturation_range_max (C++ 成员), 558
esp_ble_mesh_light_hsl_range_set_t::saturation_range_min (C++ 成员), 558
esp_ble_mesh_light_hsl_range_status_cb_t (C++ 类), 566
esp_ble_mesh_light_hsl_range_status_cb_t::hue_range_max (C++ 成员), 566
esp_ble_mesh_light_hsl_range_status_cb_t::hue_range_min (C++ 成员), 566
esp_ble_mesh_light_hsl_range_status_cb_t::saturation_range (C++ 成员), 566
esp_ble_mesh_light_hsl_range_status_cb_t::saturation_range (C++ 成员), 566
esp_ble_mesh_light_hsl_range_status_cb_t::status_code (C++ 成员), 566
esp_ble_mesh_light_hsl_sat_srv_t (C++ 类), 575
esp_ble_mesh_light_hsl_sat_srv_t::last (C++ 成员), 575
esp_ble_mesh_light_hsl_sat_srv_t::model (C++ 成员), 575
esp_ble_mesh_light_hsl_sat_srv_t::rsp_ctrl (C++ 成员), 575
esp_ble_mesh_light_hsl_sat_srv_t::state (C++ 成员), 575

esp_ble_mesh_light_hsl_sat_srv_t::transition (C++ 成员), 575
esp_ble_mesh_light_hsl_sat_srv_t::tt_delta_saturation (C++ 成员), 575
esp_ble_mesh_light_hsl_saturation_set_t (C++ 类), 557
esp_ble_mesh_light_hsl_saturation_set_t::delay (C++ 成员), 558
esp_ble_mesh_light_hsl_saturation_set_t::op_en (C++ 成员), 558
esp_ble_mesh_light_hsl_saturation_set_t::saturation (C++ 成员), 558
esp_ble_mesh_light_hsl_saturation_set_t::tid (C++ 成员), 558
esp_ble_mesh_light_hsl_saturation_set_t::trans_time (C++ 成员), 558
ESP_BLE_MESH_LIGHT_HSL_SATURATION_STATE (C++ 枚举子), 439
esp_ble_mesh_light_hsl_saturation_status_cb_t (C++ 类), 565
esp_ble_mesh_light_hsl_saturation_status_cb_t::saturation (C++ 成员), 565
esp_ble_mesh_light_hsl_saturation_status_cb_t::present_saturation (C++ 成员), 565
esp_ble_mesh_light_hsl_saturation_status_cb_t::remain_time (C++ 成员), 565
esp_ble_mesh_light_hsl_saturation_status_cb_t::target_saturation (C++ 成员), 565
esp_ble_mesh_light_hsl_set_t (C++ 类), 556
esp_ble_mesh_light_hsl_set_t::delay (C++ 成员), 557
esp_ble_mesh_light_hsl_set_t::hsl_hue (C++ 成员), 557
esp_ble_mesh_light_hsl_set_t::hsl_lightness (C++ 成员), 557
esp_ble_mesh_light_hsl_set_t::hsl_saturation (C++ 成员), 557
esp_ble_mesh_light_hsl_set_t::op_en (C++ 成员), 557
esp_ble_mesh_light_hsl_set_t::tid (C++ 成员), 557
esp_ble_mesh_light_hsl_set_t::trans_time (C++ 成员), 557
esp_ble_mesh_light_hsl_setup_srv_t (C++ 类), 574
esp_ble_mesh_light_hsl_setup_srv_t::model (C++ 成员), 574
esp_ble_mesh_light_hsl_setup_srv_t::rsp_ctrl (C++ 成员), 574
esp_ble_mesh_light_hsl_setup_srv_t::state (C++ 成员), 574
esp_ble_mesh_light_hsl_srv_t (C++ 类), 573
esp_ble_mesh_light_hsl_srv_t::last (C++ 成员), 574
esp_ble_mesh_light_hsl_srv_t::model (C++ 成员), 574
esp_ble_mesh_light_hsl_srv_t::rsp_ctrl (C++ 成员), 574
esp_ble_mesh_light_hsl_srv_t::state (C++ 成员), 574
esp_ble_mesh_light_hsl_srv_t::transition (C++ 成员), 574
esp_ble_mesh_light_hsl_srv_t::tt_delta_hue (C++ 成员), 574
esp_ble_mesh_light_hsl_srv_t::tt_delta_lightness (C++ 成员), 574
esp_ble_mesh_light_hsl_srv_t::tt_delta_saturation (C++ 成员), 574
esp_ble_mesh_light_hsl_state_t (C++ 类), 573
esp_ble_mesh_light_hsl_state_t::hue (C++ 成员), 573
esp_ble_mesh_light_hsl_state_t::hue_default (C++ 成员), 573
esp_ble_mesh_light_hsl_state_t::hue_range_max (C++ 成员), 573
esp_ble_mesh_light_hsl_state_t::hue_range_min (C++ 成员), 573
esp_ble_mesh_light_hsl_state_t::lightness (C++ 成员), 573
esp_ble_mesh_light_hsl_state_t::lightness_default (C++ 成员), 573
esp_ble_mesh_light_hsl_state_t::saturation (C++ 成员), 573
esp_ble_mesh_light_hsl_state_t::saturation_default

(C++ 成员), 573
esp_ble_mesh_light_hsl_state_t::saturation_range_max
(C++ 成员), 573
esp_ble_mesh_light_hsl_state_t::saturation_range_min
(C++ 成员), 573
esp_ble_mesh_light_hsl_state_t::status_code
(C++ 成员), 573
esp_ble_mesh_light_hsl_state_t::target_hue
(C++ 成员), 573
esp_ble_mesh_light_hsl_state_t::target_lightness
(C++ 成员), 573
esp_ble_mesh_light_hsl_state_t::target_saturation
(C++ 成员), 573
esp_ble_mesh_light_hsl_status_cb_t (C++ 类), 564
esp_ble_mesh_light_hsl_status_cb_t::hsl_hue
(C++ 成员), 564
esp_ble_mesh_light_hsl_status_cb_t::hsl_lightness
(C++ 成员), 564
esp_ble_mesh_light_hsl_status_cb_t::hsl_saturation
(C++ 成员), 564
esp_ble_mesh_light_hsl_status_cb_t::op_en
(C++ 成员), 564
esp_ble_mesh_light_hsl_status_cb_t::remain_time
(C++ 成员), 564
esp_ble_mesh_light_hsl_target_status_cb_t
(C++ 类), 564
esp_ble_mesh_light_hsl_target_status_cb_t::hsl_hue_target
(C++ 成员), 564
esp_ble_mesh_light_hsl_target_status_cb_t::hsl_lightness_target
(C++ 成员), 564
esp_ble_mesh_light_hsl_target_status_cb_t::hsl_saturation_target
(C++ 成员), 564
esp_ble_mesh_light_hsl_target_status_cb_t::op_en
(C++ 成员), 564
esp_ble_mesh_light_hsl_target_status_cb_t::remain_time
(C++ 成员), 564
esp_ble_mesh_light_lc_light_onoff_set_t
(C++ 类), 560
esp_ble_mesh_light_lc_light_onoff_set_t::delay
(C++ 成员), 560
esp_ble_mesh_light_lc_light_onoff_set_t::light_onoff
(C++ 成员), 560
esp_ble_mesh_light_lc_light_onoff_set_t::op_en
(C++ 成员), 560
esp_ble_mesh_light_lc_light_onoff_set_t::tid
(C++ 成员), 560
esp_ble_mesh_light_lc_light_onoff_set_t::trans_time
(C++ 成员), 560
ESP_BLE_MESH_LIGHT_LC_LIGHT_ONOFF_STATE
(C++ 枚举子), 439
esp_ble_mesh_light_lc_light_onoff_status_cb_t
(C++ 类), 568
esp_ble_mesh_light_lc_light_onoff_status_cb_t::op_en
(C++ 成员), 568
esp_ble_mesh_light_lc_light_onoff_status_cb_t::present_lig
(C++ 成员), 568
esp_ble_mesh_light_lc_light_onoff_status_cb_t::remain_time
(C++ 成员), 568
esp_ble_mesh_light_lc_light_onoff_status_cb_t::target_ligh
(C++ 成员), 568
esp_ble_mesh_light_lc_mode_set_t (C++ 类),
560
esp_ble_mesh_light_lc_mode_set_t::mode (C++
成员), 560
esp_ble_mesh_light_lc_mode_status_cb_t (C++
类), 568
esp_ble_mesh_light_lc_mode_status_cb_t::mode
(C++ 成员), 568
esp_ble_mesh_light_lc_om_set_t (C++ 类), 560
esp_ble_mesh_light_lc_om_set_t::mode (C++
成员), 560
esp_ble_mesh_light_lc_om_status_cb_t (C++
类), 568
esp_ble_mesh_light_lc_om_status_cb_t::mode
(C++ 成员), 568
esp_ble_mesh_light_lc_property_get_t (C++
类), 560
esp_ble_mesh_light_lc_property_get_t::property_id
(C++ 成员), 561
esp_ble_mesh_light_lc_property_set_t (C++
类), 561
esp_ble_mesh_light_lc_property_set_t::property_id
(C++ 成员), 561

esp_ble_mesh_light_lc_property_set_t::property_value (C++ 成员), 561
esp_ble_mesh_light_lc_property_state_t (C++ 类), 578
esp_ble_mesh_light_lc_property_state_t::ambient_luxlevel_on (C++ 成员), 579
esp_ble_mesh_light_lc_property_state_t::ambient_luxlevel_prolong (C++ 成员), 579
esp_ble_mesh_light_lc_property_state_t::ambient_luxlevel_standby (C++ 成员), 579
esp_ble_mesh_light_lc_property_state_t::lightness_on (C++ 成员), 579
esp_ble_mesh_light_lc_property_state_t::lightness_prolong (C++ 成员), 579
esp_ble_mesh_light_lc_property_state_t::lightness_standby (C++ 成员), 579
esp_ble_mesh_light_lc_property_state_t::regulator_accuracy (C++ 成员), 580
esp_ble_mesh_light_lc_property_state_t::regulator_kid (C++ 成员), 580
esp_ble_mesh_light_lc_property_state_t::regulator_kiu (C++ 成员), 579
esp_ble_mesh_light_lc_property_state_t::regulator_kpd (C++ 成员), 580
esp_ble_mesh_light_lc_property_state_t::regulator_kpu (C++ 成员), 580
esp_ble_mesh_light_lc_property_state_t::set_occupancy_to (C++ 成员), 580
esp_ble_mesh_light_lc_property_state_t::time_fade (C++ 成员), 578
esp_ble_mesh_light_lc_property_state_t::time_fade_on (C++ 成员), 578
esp_ble_mesh_light_lc_property_state_t::time_fade_standby_auto (C++ 成员), 579
esp_ble_mesh_light_lc_property_state_t::time_fade_standby_manual (C++ 成员), 579
esp_ble_mesh_light_lc_property_state_t::time_occupancy_delay (C++ 成员), 578
esp_ble_mesh_light_lc_property_state_t::time_prolong (C++ 成员), 579
esp_ble_mesh_light_lc_property_state_t::time_run_on (C++ 成员), 578
esp_ble_mesh_light_lc_property_status_cb_t (C++ 类), 568
esp_ble_mesh_light_lc_property_status_cb_t::property_id (C++ 成员), 568
esp_ble_mesh_light_lc_property_status_cb_t::property_value (C++ 成员), 568
esp_ble_mesh_light_lc_setup_srv_t (C++ 类), 581
esp_ble_mesh_light_lc_setup_srv_t::lc (C++ 成员), 582
esp_ble_mesh_light_lc_setup_srv_t::model (C++ 成员), 582
esp_ble_mesh_light_lc_setup_srv_t::rsp_ctrl (C++ 成员), 582
esp_ble_mesh_light_lc_srv_t (C++ 类), 581
esp_ble_mesh_light_lc_srv_t::last (C++ 成员), 581
esp_ble_mesh_light_lc_srv_t::lc (C++ 成员), 581
esp_ble_mesh_light_lc_srv_t::model (C++ 成员), 581
esp_ble_mesh_light_lc_srv_t::rsp_ctrl (C++ 成员), 581
esp_ble_mesh_light_lc_srv_t::transition (C++ 成员), 581
esp_ble_mesh_light_lc_state_machine_t (C++ 类), 580
esp_ble_mesh_light_lc_state_machine_t::delay (C++ 成员), 580
esp_ble_mesh_light_lc_state_machine_t::fade (C++ 成员), 580
esp_ble_mesh_light_lc_state_machine_t::fade_on (C++ 成员), 580
esp_ble_mesh_light_lc_state_machine_t::fade_standby_auto (C++ 成员), 580
esp_ble_mesh_light_lc_state_machine_t::fade_standby_manual (C++ 成员), 580
esp_ble_mesh_light_lc_state_machine_t::state (C++ 成员), 581
esp_ble_mesh_light_lc_state_machine_t::timer (C++ 成员), 581
esp_ble_mesh_light_lc_state_machine_t::trans_time (C++ 成员), 581
esp_ble_mesh_light_lc_state_t (C++ 类), 577

esp_ble_mesh_light_lc_state_t::ambient_luxlevel (C++ 成员), 578
esp_ble_mesh_light_lc_state_t::light_onoff (C++ 成员), 578
esp_ble_mesh_light_lc_state_t::linear_output (C++ 成员), 578
esp_ble_mesh_light_lc_state_t::mode (C++ 成员), 577
esp_ble_mesh_light_lc_state_t::occupancy (C++ 成员), 578
esp_ble_mesh_light_lc_state_t::occupancy_mode (C++ 成员), 578
esp_ble_mesh_light_lc_state_t::target_light_onoff (C++ 成员), 578
ESP_BLE_MESH_LIGHT_LIGHTNESS_ACTUAL_STATE (C++ 枚举子), 439
esp_ble_mesh_light_lightness_default_set_t (C++ 类), 554
esp_ble_mesh_light_lightness_default_set_t::lightness (C++ 成员), 555
esp_ble_mesh_light_lightness_default_status_cb_t (C++ 类), 562
esp_ble_mesh_light_lightness_default_status_cb_t::lightness (C++ 成员), 562
esp_ble_mesh_light_lightness_last_status_cb_t (C++ 类), 562
esp_ble_mesh_light_lightness_last_status_cb_t::lightness (C++ 成员), 562
esp_ble_mesh_light_lightness_linear_set_t (C++ 类), 554
esp_ble_mesh_light_lightness_linear_set_t::delay (C++ 成员), 554
esp_ble_mesh_light_lightness_linear_set_t::lightness (C++ 成员), 554
esp_ble_mesh_light_lightness_linear_set_t::op_en (C++ 成员), 554
esp_ble_mesh_light_lightness_linear_set_t::tid (C++ 成员), 554
esp_ble_mesh_light_lightness_linear_set_t::trans_time (C++ 成员), 554
ESP_BLE_MESH_LIGHT_LIGHTNESS_LINEAR_STATE (C++ 枚举子), 439
esp_ble_mesh_light_lightness_linear_status_cb_t (C++ 类), 561
esp_ble_mesh_light_lightness_linear_status_cb_t::op_en (C++ 成员), 561
esp_ble_mesh_light_lightness_linear_status_cb_t::present_l (C++ 成员), 561
esp_ble_mesh_light_lightness_linear_status_cb_t::remain_ti (C++ 成员), 561
esp_ble_mesh_light_lightness_linear_status_cb_t::target_li (C++ 成员), 561
esp_ble_mesh_light_lightness_range_set_t (C++ 类), 555
esp_ble_mesh_light_lightness_range_set_t::range_max (C++ 成员), 555
esp_ble_mesh_light_lightness_range_set_t::range_min (C++ 成员), 555
esp_ble_mesh_light_lightness_range_status_cb_t (C++ 类), 562
esp_ble_mesh_light_lightness_range_status_cb_t::range_max (C++ 成员), 562
esp_ble_mesh_light_lightness_range_status_cb_t::range_min (C++ 成员), 562
esp_ble_mesh_light_lightness_range_status_cb_t::status_cod (C++ 成员), 562
esp_ble_mesh_light_lightness_set_t (C++ 类), 554
esp_ble_mesh_light_lightness_set_t::delay (C++ 成员), 554
esp_ble_mesh_light_lightness_set_t::lightness (C++ 成员), 554
esp_ble_mesh_light_lightness_set_t::op_en (C++ 成员), 554
esp_ble_mesh_light_lightness_set_t::tid (C++ 成员), 554
esp_ble_mesh_light_lightness_set_t::trans_time (C++ 成员), 554
esp_ble_mesh_light_lightness_setup_srv_t (C++ 类), 570
esp_ble_mesh_light_lightness_setup_srv_t::model (C++ 成员), 570
esp_ble_mesh_light_lightness_setup_srv_t::rsp_ctrl (C++ 成员), 570

esp_ble_mesh_light_lightness_setup_srv_t::state (C++ 成员), 570
esp_ble_mesh_light_lightness_srv_t (C++ 类), 569
esp_ble_mesh_light_lightness_srv_t::actual_transmitter (C++ 成员), 570
esp_ble_mesh_light_lightness_srv_t::last (C++ 成员), 570
esp_ble_mesh_light_lightness_srv_t::linear_transmitter (C++ 成员), 570
esp_ble_mesh_light_lightness_srv_t::model (C++ 成员), 570
esp_ble_mesh_light_lightness_srv_t::rsp_ctrl (C++ 成员), 570
esp_ble_mesh_light_lightness_srv_t::state (C++ 成员), 570
esp_ble_mesh_light_lightness_srv_t::tt_delta_lightness_actual (C++ 成员), 570
esp_ble_mesh_light_lightness_srv_t::tt_delta_lightness_linear (C++ 成员), 570
esp_ble_mesh_light_lightness_state_t (C++ 类), 569
esp_ble_mesh_light_lightness_state_t::lightness_actual (C++ 成员), 569
esp_ble_mesh_light_lightness_state_t::lightness_default (C++ 成员), 569
esp_ble_mesh_light_lightness_state_t::lightness_last (C++ 成员), 569
esp_ble_mesh_light_lightness_state_t::lightness_linear (C++ 成员), 569
esp_ble_mesh_light_lightness_state_t::lightness_range_max (C++ 成员), 569
esp_ble_mesh_light_lightness_state_t::lightness_range_min (C++ 成员), 569
esp_ble_mesh_light_lightness_state_t::status_code (C++ 成员), 569
esp_ble_mesh_light_lightness_state_t::target_lightness_actual (C++ 成员), 569
esp_ble_mesh_light_lightness_state_t::target_lightness_linear (C++ 成员), 569
esp_ble_mesh_light_lightness_status_cb_t (C++ 类), 561
esp_ble_mesh_light_lightness_status_cb_t::op_en (C++ 成员), 561
esp_ble_mesh_light_lightness_status_cb_t::present_lightness (C++ 成员), 561
esp_ble_mesh_light_lightness_status_cb_t::remain_time (C++ 成员), 561
esp_ble_mesh_light_lightness_status_cb_t::target_lightness (C++ 成员), 561
esp_ble_mesh_light_message_opcode_t (C++ 类型), 432
esp_ble_mesh_light_xyl_default_set_t (C++ 类), 559
esp_ble_mesh_light_xyl_default_set_t::lightness (C++ 成员), 559
esp_ble_mesh_light_xyl_default_set_t::xyl_x (C++ 成员), 559
esp_ble_mesh_light_xyl_default_set_t::xyl_y (C++ 成员), 559
esp_ble_mesh_light_xyl_default_status_cb_t (C++ 类), 567
esp_ble_mesh_light_xyl_default_status_cb_t::lightness (C++ 成员), 567
esp_ble_mesh_light_xyl_default_status_cb_t::xyl_x (C++ 成员), 567
esp_ble_mesh_light_xyl_default_status_cb_t::xyl_y (C++ 成员), 567
ESP_BLE_MESH_LIGHT_XYL_LIGHTNESS_STATE (C++ 枚举子), 439
esp_ble_mesh_light_xyl_range_set_t (C++ 类), 559
esp_ble_mesh_light_xyl_range_set_t::xyl_x_range_max (C++ 成员), 559
esp_ble_mesh_light_xyl_range_set_t::xyl_x_range_min (C++ 成员), 559
esp_ble_mesh_light_xyl_range_set_t::xyl_y_range_max (C++ 成员), 560
esp_ble_mesh_light_xyl_range_set_t::xyl_y_range_min (C++ 成员), 560
esp_ble_mesh_light_xyl_range_status_cb_t (C++ 类), 567
esp_ble_mesh_light_xyl_range_status_cb_t::status_code (C++ 成员), 567

esp_ble_mesh_light_xyl_range_status_cb_t::xyl_range_max (C++ 成员), 567
esp_ble_mesh_light_xyl_range_status_cb_t::xyl_range_min (C++ 成员), 567
esp_ble_mesh_light_xyl_range_status_cb_t::xyl_status (C++ 成员), 567
esp_ble_mesh_light_xyl_range_status_cb_t::xyl_y_range_max (C++ 成员), 567
esp_ble_mesh_light_xyl_set_t (C++ 类), 558
esp_ble_mesh_light_xyl_set_t::delay (C++ 成员), 559
esp_ble_mesh_light_xyl_set_t::op_en (C++ 成员), 559
esp_ble_mesh_light_xyl_set_t::tid (C++ 成员), 559
esp_ble_mesh_light_xyl_set_t::trans_time (C++ 成员), 559
esp_ble_mesh_light_xyl_set_t::xyl_lightness (C++ 成员), 559
esp_ble_mesh_light_xyl_set_t::xyl_x (C++ 成员), 559
esp_ble_mesh_light_xyl_set_t::xyl_y (C++ 成员), 559
esp_ble_mesh_light_xyl_setup_srv_t (C++ 类), 577
esp_ble_mesh_light_xyl_setup_srv_t::model (C++ 成员), 577
esp_ble_mesh_light_xyl_setup_srv_t::rsp_ctrl (C++ 成员), 577
esp_ble_mesh_light_xyl_setup_srv_t::state (C++ 成员), 577
esp_ble_mesh_light_xyl_srv_t (C++ 类), 576
esp_ble_mesh_light_xyl_srv_t::last (C++ 成员), 577
esp_ble_mesh_light_xyl_srv_t::model (C++ 成员), 577
esp_ble_mesh_light_xyl_srv_t::rsp_ctrl (C++ 成员), 577
esp_ble_mesh_light_xyl_srv_t::state (C++ 成员), 577
esp_ble_mesh_light_xyl_srv_t::transition (C++ 成员), 577
esp_ble_mesh_light_xyl_srv_t::tt_delta_lightness (C++ 成员), 577
esp_ble_mesh_light_xyl_srv_t::tt_delta_x (C++ 成员), 577
esp_ble_mesh_light_xyl_srv_t::tt_delta_y (C++ 成员), 577
esp_ble_mesh_light_xyl_state_t (C++ 类), 575
esp_ble_mesh_light_xyl_state_t::lightness (C++ 成员), 576
esp_ble_mesh_light_xyl_state_t::lightness_default (C++ 成员), 576
esp_ble_mesh_light_xyl_state_t::status_code (C++ 成员), 576
esp_ble_mesh_light_xyl_state_t::target_lightness (C++ 成员), 576
esp_ble_mesh_light_xyl_state_t::target_x (C++ 成员), 576
esp_ble_mesh_light_xyl_state_t::target_y (C++ 成员), 576
esp_ble_mesh_light_xyl_state_t::x (C++ 成员), 576
esp_ble_mesh_light_xyl_state_t::x_default (C++ 成员), 576
esp_ble_mesh_light_xyl_state_t::x_range_max (C++ 成员), 576
esp_ble_mesh_light_xyl_state_t::x_range_min (C++ 成员), 576
esp_ble_mesh_light_xyl_state_t::y (C++ 成员), 576
esp_ble_mesh_light_xyl_state_t::y_default (C++ 成员), 576
esp_ble_mesh_light_xyl_state_t::y_range_max (C++ 成员), 576
esp_ble_mesh_light_xyl_state_t::y_range_min (C++ 成员), 576
esp_ble_mesh_light_xyl_status_cb_t (C++ 类), 566
esp_ble_mesh_light_xyl_status_cb_t::op_en (C++ 成员), 566
esp_ble_mesh_light_xyl_status_cb_t::remain_time (C++ 成员), 566
esp_ble_mesh_light_xyl_status_cb_t::xyl_lightness

(C++ 成员), 566
esp_ble_mesh_light_xyl_status_cb_t::xyl_x (C++ 成员), 566
esp_ble_mesh_light_xyl_status_cb_t::xyl_y (C++ 成员), 566
esp_ble_mesh_light_xyl_target_status_cb_t (C++ 类), 566
esp_ble_mesh_light_xyl_target_status_cb_t::op_en (C++ 成员), 567
esp_ble_mesh_light_xyl_target_status_cb_t::remain_time (C++ 成员), 567
esp_ble_mesh_light_xyl_target_status_cb_t::target_xyl_lightness (C++ 成员), 567
esp_ble_mesh_light_xyl_target_status_cb_t::target_xyl_x (C++ 成员), 567
esp_ble_mesh_light_xyl_target_status_cb_t::target_xyl_y (C++ 成员), 567
esp_ble_mesh_lighting_server_cb_event_t (C++ 类型), 602
esp_ble_mesh_lighting_server_cb_param_t (C++ 类), 594
esp_ble_mesh_lighting_server_cb_param_t::ctx (C++ 成员), 595
esp_ble_mesh_lighting_server_cb_param_t::model (C++ 成员), 595
esp_ble_mesh_lighting_server_cb_param_t::value (C++ 成员), 595
esp_ble_mesh_lighting_server_cb_t (C++ 类型), 601
esp_ble_mesh_lighting_server_cb_value_t (C++ 类型), 553
esp_ble_mesh_lighting_server_cb_value_t::get (C++ 成员), 553
esp_ble_mesh_lighting_server_cb_value_t::set (C++ 成员), 553
esp_ble_mesh_lighting_server_cb_value_t::state_change (C++ 成员), 553
esp_ble_mesh_lighting_server_cb_value_t::status (C++ 成员), 553
ESP_BLE_MESH_LIGHTING_SERVER_EVT_MAX (C++ 枚举子), 602
ESP_BLE_MESH_LIGHTING_SERVER_RECV_GET_MSG_EVT (C++ 枚举子), 602
esp_ble_mesh_lighting_server_recv_get_msg_t (C++ 类型), 552
esp_ble_mesh_lighting_server_recv_get_msg_t::lc_property (C++ 成员), 552
ESP_BLE_MESH_LIGHTING_SERVER_RECV_SET_MSG_EVT (C++ 枚举子), 602
esp_ble_mesh_lighting_server_recv_set_msg_t (C++ 类型), 552
esp_ble_mesh_lighting_server_recv_set_msg_t::ctl (C++ 成员), 552
esp_ble_mesh_lighting_server_recv_set_msg_t::ctl_default (C++ 成员), 552
esp_ble_mesh_lighting_server_recv_set_msg_t::ctl_temp (C++ 成员), 552
esp_ble_mesh_lighting_server_recv_set_msg_t::ctl_temp_rang (C++ 成员), 552
esp_ble_mesh_lighting_server_recv_set_msg_t::hsl (C++ 成员), 552
esp_ble_mesh_lighting_server_recv_set_msg_t::hsl_default (C++ 成员), 552
esp_ble_mesh_lighting_server_recv_set_msg_t::hsl_hue (C++ 成员), 552
esp_ble_mesh_lighting_server_recv_set_msg_t::hsl_range (C++ 成员), 553
esp_ble_mesh_lighting_server_recv_set_msg_t::hsl_saturatio (C++ 成员), 552
esp_ble_mesh_lighting_server_recv_set_msg_t::lc_light_onof (C++ 成员), 553
esp_ble_mesh_lighting_server_recv_set_msg_t::lc_mode (C++ 成员), 553
esp_ble_mesh_lighting_server_recv_set_msg_t::lc_om (C++ 成员), 553
esp_ble_mesh_lighting_server_recv_set_msg_t::lc_property (C++ 成员), 553
esp_ble_mesh_lighting_server_recv_set_msg_t::lightness (C++ 成员), 552
esp_ble_mesh_lighting_server_recv_set_msg_t::lightness_def (C++ 成员), 552
esp_ble_mesh_lighting_server_recv_set_msg_t::lightness_lin (C++ 成员), 552
esp_ble_mesh_lighting_server_recv_set_msg_t::lightness_ran

(C++ 成员), 552
esp_ble_mesh_lighting_server_recv_set_msg_t::xyl (C++ 成员), 553
esp_ble_mesh_lighting_server_recv_set_msg_t::xyl_default (C++ 成员), 553
esp_ble_mesh_lighting_server_recv_set_msg_t::xyl_range (C++ 成员), 553
ESP_BLE_MESH_LIGHTING_SERVER_RECV_STATUS_MSG_EVT (C++ 枚举子), 602
esp_ble_mesh_lighting_server_recv_status_msg_t (C++ 类型), 553
esp_ble_mesh_lighting_server_recv_status_msg_t::ctl_temp_range (C++ 成员), 553
ESP_BLE_MESH_LIGHTING_SERVER_STATE_CHANGE_EVT (C++ 枚举子), 602
esp_ble_mesh_lighting_server_state_change_t (C++ 类型), 550
esp_ble_mesh_lighting_server_state_change_t::ctl_default_lightness (C++ 成员), 551
esp_ble_mesh_lighting_server_state_change_t::ctl_set (C++ 成员), 551
esp_ble_mesh_lighting_server_state_change_t::ctl_temp_range_set (C++ 成员), 551
esp_ble_mesh_lighting_server_state_change_t::ctl_temp_set (C++ 成员), 551
esp_ble_mesh_lighting_server_state_change_t::hsl_default_set (C++ 成员), 551
esp_ble_mesh_lighting_server_state_change_t::hsl_hue_set (C++ 成员), 551
esp_ble_mesh_lighting_server_state_change_t::hsl_range_set (C++ 成员), 551
esp_ble_mesh_lighting_server_state_change_t::hsl_saturation_set (C++ 成员), 551
esp_ble_mesh_lighting_server_state_change_t::hsl_set (C++ 成员), 551
esp_ble_mesh_lighting_server_state_change_t::light_hsl_range_set (C++ 成员), 551
esp_ble_mesh_lighting_server_state_change_t::lightness_set (C++ 成员), 551
esp_ble_mesh_lighting_server_state_change_t::lc_property_set (C++ 成员), 551
esp_ble_mesh_lighting_server_state_change_t::lightness_default (C++ 成员), 550
esp_ble_mesh_lighting_server_state_change_t::lightness_linear (C++ 成员), 550
esp_ble_mesh_lighting_server_state_change_t::lightness_range (C++ 成员), 550
esp_ble_mesh_lighting_server_state_change_t::lightness_set (C++ 成员), 550
esp_ble_mesh_lighting_server_state_change_t::sensor_status (C++ 成员), 551
esp_ble_mesh_lighting_server_state_change_t::xyl_default_set (C++ 成员), 551
esp_ble_mesh_lighting_server_state_change_t::xyl_range_set (C++ 成员), 551
esp_ble_mesh_lighting_server_state_change_t::xyl_set (C++ 成员), 551
esp_ble_mesh_lpn_disable (C++ 函数), 443
ESP_BLE_MESH_LPN_DISABLE_COMP_EVT (C++ 枚举子), 437
esp_ble_mesh_lpn_enable (C++ 函数), 443
ESP_BLE_MESH_LPN_ENABLE_COMP_EVT (C++ 枚举子), 437
ESP_BLE_MESH_LPN_FRIENDSHIP_ESTABLISH_EVT (C++ 枚举子), 437
ESP_BLE_MESH_LPN_FRIENDSHIP_TERMINATE_EVT (C++ 枚举子), 437
esp_ble_mesh_lpn_poll (C++ 函数), 443
ESP_BLE_MESH_LPN_POLL_COMP_EVT (C++ 枚举子), 437
ESP_BLE_MESH_MECHANISM_JAMMED_ERROR (C 宏), 544
ESP_BLE_MESH_MECHANISM_JAMMED_WARNING (C 宏), 544
ESP_BLE_MESH_MEMORY_ERROR (C 宏), 544
ESP_BLE_MESH_MEMORY_WARNING (C 宏), 543
ESP_BLE_MESH_MIC_LONG (C 宏), 411
ESP_BLE_MESH_MIC_SHORT (C 宏), 411
esp_ble_mesh_model (C++ 类), 402
esp_ble_mesh_model::element (C++ 成员), 403
esp_ble_mesh_model::element_idx (C++ 成员), 402

esp_ble_mesh_model::flags (C++ 成员), 402
esp_ble_mesh_model::groups (C++ 成员), 403
esp_ble_mesh_model::keys (C++ 成员), 403
esp_ble_mesh_model::model_idx (C++ 成员), 402
esp_ble_mesh_model::op (C++ 成员), 403
esp_ble_mesh_model::pub (C++ 成员), 403
esp_ble_mesh_model::user_data (C++ 成员), 403
esp_ble_mesh_model::[anonymous] (C++ 成员), 402
esp_ble_mesh_model_cb_event_t (C++ 类型), 439
esp_ble_mesh_model_cb_param_t (C++ 类型), 397
esp_ble_mesh_model_cb_param_t::ble_mesh_client_model_send_comp_param (C++ 类), 398
esp_ble_mesh_model_cb_param_t::ble_mesh_client_model_send_comp_param::ctx (C++ 成员), 398
esp_ble_mesh_model_cb_param_t::ble_mesh_client_model_send_comp_param::model (C++ 成员), 398
esp_ble_mesh_model_cb_param_t::ble_mesh_client_model_send_comp_param::opcode (C++ 成员), 398
esp_ble_mesh_model_cb_param_t::ble_mesh_mod_recv_publish_msg_param (C++ 类), 398
esp_ble_mesh_model_cb_param_t::ble_mesh_mod_recv_publish_msg_param::ctx (C++ 成员), 398
esp_ble_mesh_model_cb_param_t::ble_mesh_mod_recv_publish_msg_param::length (C++ 成员), 398
esp_ble_mesh_model_cb_param_t::ble_mesh_mod_recv_publish_msg_param::model (C++ 成员), 398
esp_ble_mesh_model_cb_param_t::ble_mesh_mod_recv_publish_msg_param::msg (C++ 成员), 398
esp_ble_mesh_model_cb_param_t::ble_mesh_mod_recv_publish_msg_param::opcode (C++ 成员), 398
esp_ble_mesh_model_cb_param_t::ble_mesh_model_operation_comp_param (C++ 类), 398
esp_ble_mesh_model_cb_param_t::ble_mesh_model_operation_comp_param::ctx (C++ 成员), 399
esp_ble_mesh_model_cb_param_t::ble_mesh_model_operation_comp_param::length (C++ 成员), 399
esp_ble_mesh_model_cb_param_t::ble_mesh_model_operation_comp_param::model (C++ 成员), 399
esp_ble_mesh_model_cb_param_t::ble_mesh_model_operation_comp_param::msg (C++ 成员), 399
esp_ble_mesh_model_cb_param_t::ble_mesh_model_operation_comp_param::opcode (C++ 成员), 399
esp_ble_mesh_model_cb_param_t::ble_mesh_model_publish_comp_param (C++ 类), 399
esp_ble_mesh_model_cb_param_t::ble_mesh_model_publish_comp_param::model (C++ 成员), 399
esp_ble_mesh_model_cb_param_t::ble_mesh_model_publish_comp_param::status (C++ 成员), 399
esp_ble_mesh_model_cb_param_t::ble_mesh_model_publish_update_comp_param (C++ 类), 399
esp_ble_mesh_model_cb_param_t::ble_mesh_model_publish_update_comp_param::model (C++ 成员), 399
esp_ble_mesh_model_cb_param_t::ble_mesh_model_send_comp_param (C++ 类), 399
esp_ble_mesh_model_cb_param_t::ble_mesh_model_send_comp_param::ctx (C++ 成员), 400
esp_ble_mesh_model_cb_param_t::ble_mesh_model_send_comp_param::model (C++ 成员), 399
esp_ble_mesh_model_cb_param_t::ble_mesh_model_send_comp_param::opcode (C++ 成员), 400
esp_ble_mesh_model_cb_param_t::ble_mesh_model_send_comp_param::state_change (C++ 成员), 399
esp_ble_mesh_model_cb_param_t::ble_mesh_server_model_update_state_comp_param (C++ 类), 400
esp_ble_mesh_model_cb_param_t::ble_mesh_server_model_update_state_comp_param::ctx (C++ 成员), 400
esp_ble_mesh_model_cb_param_t::ble_mesh_server_model_update_state_comp_param::model (C++ 成员), 400
esp_ble_mesh_model_cb_param_t::ble_mesh_server_model_update_state_comp_param::msg (C++ 成员), 400
esp_ble_mesh_model_cb_param_t::client_recv_publish_msg (C++ 成员), 398
esp_ble_mesh_model_cb_param_t::client_send_timeout (C++ 成员), 398
esp_ble_mesh_model_cb_param_t::ctx (C++ 成员), 397
esp_ble_mesh_model_cb_param_t::length (C++ 成员), 397
esp_ble_mesh_model_cb_param_t::model (C++ 成员), 398
esp_ble_mesh_model_cb_param_t::msg (C++ 成员), 397
esp_ble_mesh_model_cb_param_t::server_model_update_state

(C++ 成员), 398
esp_ble_mesh_model_cb_t (C++ 类型), 451
ESP_BLE_MESH_MODEL_CFG_CLI (C 宏), 491
ESP_BLE_MESH_MODEL_CFG_SRV (C 宏), 491
ESP_BLE_MESH_MODEL_EVT_MAX (C++ 枚举子), 440
ESP_BLE_MESH_MODEL_GEN_ADMIN_PROP_SRV (C 宏), 531
ESP_BLE_MESH_MODEL_GEN_BATTERY_CLI (C 宏), 527
ESP_BLE_MESH_MODEL_GEN_BATTERY_SRV (C 宏), 530
ESP_BLE_MESH_MODEL_GEN_CLIENT_PROP_SRV (C 宏), 532
ESP_BLE_MESH_MODEL_GEN_DEF_TRANS_TIME_CLI (C 宏), 526
ESP_BLE_MESH_MODEL_GEN_DEF_TRANS_TIME_SRV (C 宏), 528
ESP_BLE_MESH_MODEL_GEN_LEVEL_CLI (C 宏), 526
ESP_BLE_MESH_MODEL_GEN_LEVEL_SRV (C 宏), 528
ESP_BLE_MESH_MODEL_GEN_LOCATION_CLI (C 宏), 527
ESP_BLE_MESH_MODEL_GEN_LOCATION_SETUP_SRV (C 宏), 531
ESP_BLE_MESH_MODEL_GEN_LOCATION_SRV (C 宏), 530
ESP_BLE_MESH_MODEL_GEN_MANUFACTURER_PROP_SRV (C 宏), 532
ESP_BLE_MESH_MODEL_GEN_ONOFF_CLI (C 宏), 526
ESP_BLE_MESH_MODEL_GEN_ONOFF_SRV (C 宏), 528
ESP_BLE_MESH_MODEL_GEN_POWER_LEVEL_CLI (C 宏), 527
ESP_BLE_MESH_MODEL_GEN_POWER_LEVEL_SETUP_SRV (C 宏), 530
ESP_BLE_MESH_MODEL_GEN_POWER_LEVEL_SRV (C 宏), 529
ESP_BLE_MESH_MODEL_GEN_POWER_ONOFF_CLI (C 宏), 526
ESP_BLE_MESH_MODEL_GEN_POWER_ONOFF_SETUP_SRV (C 宏), 529
ESP_BLE_MESH_MODEL_GEN_POWER_ONOFF_SRV (C 宏), 529
ESP_BLE_MESH_MODEL_GEN_PROPERTY_CLI (C 宏), 528
ESP_BLE_MESH_MODEL_GEN_USER_PROP_SRV (C 宏), 531
ESP_BLE_MESH_MODEL_HEALTH_CLI (C 宏), 542
ESP_BLE_MESH_MODEL_HEALTH_SRV (C 宏), 542
ESP_BLE_MESH_MODEL_ID_CONFIG_CLI (C 宏), 416
ESP_BLE_MESH_MODEL_ID_CONFIG_SRV (C 宏), 416
ESP_BLE_MESH_MODEL_ID_GEN_ADMIN_PROP_SRV (C 宏), 417
ESP_BLE_MESH_MODEL_ID_GEN_BATTERY_CLI (C 宏), 416
ESP_BLE_MESH_MODEL_ID_GEN_BATTERY_SRV (C 宏), 416
ESP_BLE_MESH_MODEL_ID_GEN_CLIENT_PROP_SRV (C 宏), 417
ESP_BLE_MESH_MODEL_ID_GEN_DEF_TRANS_TIME_CLI (C 宏), 416
ESP_BLE_MESH_MODEL_ID_GEN_DEF_TRANS_TIME_SRV (C 宏), 416
ESP_BLE_MESH_MODEL_ID_GEN_LEVEL_CLI (C 宏), 416
ESP_BLE_MESH_MODEL_ID_GEN_LEVEL_SRV (C 宏), 416
ESP_BLE_MESH_MODEL_ID_GEN_LOCATION_CLI (C 宏), 417
ESP_BLE_MESH_MODEL_ID_GEN_LOCATION_SETUP_SRV (C 宏), 417
ESP_BLE_MESH_MODEL_ID_GEN_LOCATION_SRV (C 宏), 417
ESP_BLE_MESH_MODEL_ID_GEN_MANUFACTURER_PROP_SRV (C 宏), 417
ESP_BLE_MESH_MODEL_ID_GEN_ONOFF_CLI (C 宏), 416
ESP_BLE_MESH_MODEL_ID_GEN_ONOFF_SRV (C 宏), 416
ESP_BLE_MESH_MODEL_ID_GEN_POWER_LEVEL_CLI (C 宏), 416
ESP_BLE_MESH_MODEL_ID_GEN_POWER_LEVEL_SETUP_SRV (C 宏), 416
ESP_BLE_MESH_MODEL_ID_GEN_POWER_LEVEL_SRV (C 宏), 416
ESP_BLE_MESH_MODEL_ID_GEN_POWER_ONOFF_CLI (C 宏), 416
ESP_BLE_MESH_MODEL_ID_GEN_POWER_ONOFF_SETUP_SRV

(C 宏), 416
ESP_BLE_MESH_MODEL_ID_GEN_POWER_ONOFF_SRV (C 宏), 416
ESP_BLE_MESH_MODEL_ID_GEN_PROP_CLI (C 宏), 417
ESP_BLE_MESH_MODEL_ID_GEN_USER_PROP_SRV (C 宏), 417
ESP_BLE_MESH_MODEL_ID_HEALTH_CLI (C 宏), 416
ESP_BLE_MESH_MODEL_ID_HEALTH_SRV (C 宏), 416
ESP_BLE_MESH_MODEL_ID_LIGHT_CTL_CLI (C 宏), 417
ESP_BLE_MESH_MODEL_ID_LIGHT_CTL_SETUP_SRV (C 宏), 417
ESP_BLE_MESH_MODEL_ID_LIGHT_CTL_SRV (C 宏), 417
ESP_BLE_MESH_MODEL_ID_LIGHT_CTL_TEMP_SRV (C 宏), 417
ESP_BLE_MESH_MODEL_ID_LIGHT_HSL_CLI (C 宏), 417
ESP_BLE_MESH_MODEL_ID_LIGHT_HSL_HUE_SRV (C 宏), 418
ESP_BLE_MESH_MODEL_ID_LIGHT_HSL_SAT_SRV (C 宏), 418
ESP_BLE_MESH_MODEL_ID_LIGHT_HSL_SETUP_SRV (C 宏), 417
ESP_BLE_MESH_MODEL_ID_LIGHT_HSL_SRV (C 宏), 417
ESP_BLE_MESH_MODEL_ID_LIGHT_LC_CLI (C 宏), 418
ESP_BLE_MESH_MODEL_ID_LIGHT_LC_SETUP_SRV (C 宏), 418
ESP_BLE_MESH_MODEL_ID_LIGHT_LC_SRV (C 宏), 418
ESP_BLE_MESH_MODEL_ID_LIGHT_LIGHTNESS_CLI (C 宏), 417
ESP_BLE_MESH_MODEL_ID_LIGHT_LIGHTNESS_SETUP_SRV (C 宏), 417
ESP_BLE_MESH_MODEL_ID_LIGHT_LIGHTNESS_SRV (C 宏), 417
ESP_BLE_MESH_MODEL_ID_LIGHT_XYL_CLI (C 宏), 418
ESP_BLE_MESH_MODEL_ID_LIGHT_XYL_SETUP_SRV (C 宏), 418
ESP_BLE_MESH_MODEL_ID_LIGHT_XYL_SRV (C 宏), 418
ESP_BLE_MESH_MODEL_ID_SCENE_CLI (C 宏), 417
ESP_BLE_MESH_MODEL_ID_SCENE_SETUP_SRV (C 宏), 417
ESP_BLE_MESH_MODEL_ID_SCENE_SRV (C 宏), 417
ESP_BLE_MESH_MODEL_ID_SCHEDULER_CLI (C 宏), 417
ESP_BLE_MESH_MODEL_ID_SCHEDULER_SETUP_SRV (C 宏), 417
ESP_BLE_MESH_MODEL_ID_SCHEDULER_SRV (C 宏), 417
ESP_BLE_MESH_MODEL_ID_SENSOR_CLI (C 宏), 417
ESP_BLE_MESH_MODEL_ID_SENSOR_SETUP_SRV (C 宏), 417
ESP_BLE_MESH_MODEL_ID_SENSOR_SRV (C 宏), 417
ESP_BLE_MESH_MODEL_ID_TIME_CLI (C 宏), 417
ESP_BLE_MESH_MODEL_ID_TIME_SETUP_SRV (C 宏), 417
ESP_BLE_MESH_MODEL_ID_TIME_SRV (C 宏), 417
ESP_BLE_MESH_MODEL_LIGHT_CTL_CLI (C 宏), 595
ESP_BLE_MESH_MODEL_LIGHT_CTL_SETUP_SRV (C 宏), 597
ESP_BLE_MESH_MODEL_LIGHT_CTL_SRV (C 宏), 597
ESP_BLE_MESH_MODEL_LIGHT_CTL_TEMP_SRV (C 宏), 598
ESP_BLE_MESH_MODEL_LIGHT_HSL_CLI (C 宏), 595
ESP_BLE_MESH_MODEL_LIGHT_HSL_HUE_SRV (C 宏), 599
ESP_BLE_MESH_MODEL_LIGHT_HSL_SAT_SRV (C 宏), 599
ESP_BLE_MESH_MODEL_LIGHT_HSL_SETUP_SRV (C 宏), 598
ESP_BLE_MESH_MODEL_LIGHT_HSL_SRV (C 宏), 598
ESP_BLE_MESH_MODEL_LIGHT_LC_CLI (C 宏), 596
ESP_BLE_MESH_MODEL_LIGHT_LC_SETUP_SRV (C 宏), 600
ESP_BLE_MESH_MODEL_LIGHT_LC_SRV (C 宏), 600
ESP_BLE_MESH_MODEL_LIGHT_LIGHTNESS_CLI (C 宏), 595
ESP_BLE_MESH_MODEL_LIGHT_LIGHTNESS_SETUP_SRV (C 宏), 596
ESP_BLE_MESH_MODEL_LIGHT_LIGHTNESS_SRV (C 宏), 596

ESP_BLE_MESH_MODEL_LIGHT_XYL_CLI (C 宏), 596
ESP_BLE_MESH_MODEL_LIGHT_XYL_SETUP_SRV (C 宏), 600
ESP_BLE_MESH_MODEL_LIGHT_XYL_SRV (C 宏), 599
esp_ble_mesh_model_msg_opcode_init (C++ 函数), 444
ESP_BLE_MESH_MODEL_NONE (C 宏), 415
ESP_BLE_MESH_MODEL_OP (C 宏), 415
ESP_BLE_MESH_MODEL_OP_1 (C 宏), 414
ESP_BLE_MESH_MODEL_OP_2 (C 宏), 414
ESP_BLE_MESH_MODEL_OP_3 (C 宏), 414
ESP_BLE_MESH_MODEL_OP_APP_KEY_ADD (C 宏), 420
ESP_BLE_MESH_MODEL_OP_APP_KEY_DELETE (C 宏), 420
ESP_BLE_MESH_MODEL_OP_APP_KEY_GET (C 宏), 418
ESP_BLE_MESH_MODEL_OP_APP_KEY_LIST (C 宏), 421
ESP_BLE_MESH_MODEL_OP_APP_KEY_STATUS (C 宏), 421
ESP_BLE_MESH_MODEL_OP_APP_KEY_UPDATE (C 宏), 420
ESP_BLE_MESH_MODEL_OP_ATTENTION_GET (C 宏), 422
ESP_BLE_MESH_MODEL_OP_ATTENTION_SET (C 宏), 422
ESP_BLE_MESH_MODEL_OP_ATTENTION_SET_UNACK (C 宏), 423
ESP_BLE_MESH_MODEL_OP_ATTENTION_STATUS (C 宏), 423
ESP_BLE_MESH_MODEL_OP_BEACON_GET (C 宏), 418
ESP_BLE_MESH_MODEL_OP_BEACON_SET (C 宏), 419
ESP_BLE_MESH_MODEL_OP_BEACON_STATUS (C 宏), 421
ESP_BLE_MESH_MODEL_OP_COMPOSITION_DATA_GET (C 宏), 418
ESP_BLE_MESH_MODEL_OP_COMPOSITION_DATA_STATUS (C 宏), 421
ESP_BLE_MESH_MODEL_OP_DEFAULT_TTL_GET (C 宏), 418
ESP_BLE_MESH_MODEL_OP_DEFAULT_TTL_SET (C 宏), 419
ESP_BLE_MESH_MODEL_OP_DEFAULT_TTL_STATUS (C 宏), 421
ESP_BLE_MESH_MODEL_OP_END (C 宏), 415
ESP_BLE_MESH_MODEL_OP_FRIEND_GET (C 宏), 418
ESP_BLE_MESH_MODEL_OP_FRIEND_SET (C 宏), 420
ESP_BLE_MESH_MODEL_OP_FRIEND_STATUS (C 宏), 421
ESP_BLE_MESH_MODEL_OP_GATT_PROXY_GET (C 宏), 418
ESP_BLE_MESH_MODEL_OP_GATT_PROXY_SET (C 宏), 419
ESP_BLE_MESH_MODEL_OP_GATT_PROXY_STATUS (C 宏), 421
ESP_BLE_MESH_MODEL_OP_GEN_ADMIN_PROPERTIES_GET (C 宏), 425
ESP_BLE_MESH_MODEL_OP_GEN_ADMIN_PROPERTIES_STATUS (C 宏), 425
ESP_BLE_MESH_MODEL_OP_GEN_ADMIN_PROPERTY_GET (C 宏), 425
ESP_BLE_MESH_MODEL_OP_GEN_ADMIN_PROPERTY_SET (C 宏), 425
ESP_BLE_MESH_MODEL_OP_GEN_ADMIN_PROPERTY_SET_UNACK (C 宏), 425
ESP_BLE_MESH_MODEL_OP_GEN_ADMIN_PROPERTY_STATUS (C 宏), 425
ESP_BLE_MESH_MODEL_OP_GEN_BATTERY_GET (C 宏), 424
ESP_BLE_MESH_MODEL_OP_GEN_BATTERY_STATUS (C 宏), 424
ESP_BLE_MESH_MODEL_OP_GEN_CLIENT_PROPERTIES_GET (C 宏), 425
ESP_BLE_MESH_MODEL_OP_GEN_CLIENT_PROPERTIES_STATUS (C 宏), 425
ESP_BLE_MESH_MODEL_OP_GEN_DEF_TRANS_TIME_GET (C 宏), 423
ESP_BLE_MESH_MODEL_OP_GEN_DEF_TRANS_TIME_SET (C 宏), 423
ESP_BLE_MESH_MODEL_OP_GEN_DEF_TRANS_TIME_SET_UNACK (C 宏), 423
ESP_BLE_MESH_MODEL_OP_GEN_DEF_TRANS_TIME_STATUS (C 宏), 423
ESP_BLE_MESH_MODEL_OP_GEN_DELTA_SET (C 宏), 423
ESP_BLE_MESH_MODEL_OP_GEN_DELTA_SET_UNACK (C

宏), 423
ESP_BLE_MESH_MODEL_OP_GEN_LEVEL_GET (C 宏), 423
ESP_BLE_MESH_MODEL_OP_GEN_LEVEL_SET (C 宏), 423
ESP_BLE_MESH_MODEL_OP_GEN_LEVEL_SET_UNACK (C 宏), 423
ESP_BLE_MESH_MODEL_OP_GEN_LEVEL_STATUS (C 宏), 423
ESP_BLE_MESH_MODEL_OP_GEN_LOC_GLOBAL_GET (C 宏), 424
ESP_BLE_MESH_MODEL_OP_GEN_LOC_GLOBAL_SET (C 宏), 424
ESP_BLE_MESH_MODEL_OP_GEN_LOC_GLOBAL_SET_UNACK (C 宏), 424
ESP_BLE_MESH_MODEL_OP_GEN_LOC_GLOBAL_STATUS (C 宏), 424
ESP_BLE_MESH_MODEL_OP_GEN_LOC_LOCAL_GET (C 宏), 424
ESP_BLE_MESH_MODEL_OP_GEN_LOC_LOCAL_SET (C 宏), 424
ESP_BLE_MESH_MODEL_OP_GEN_LOC_LOCAL_SET_UNACK (C 宏), 424
ESP_BLE_MESH_MODEL_OP_GEN_LOC_LOCAL_STATUS (C 宏), 424
ESP_BLE_MESH_MODEL_OP_GEN_MANUFACTURER_PROPERTIES_GET (C 宏), 424
ESP_BLE_MESH_MODEL_OP_GEN_MANUFACTURER_PROPERTIES_STATUS (C 宏), 424
ESP_BLE_MESH_MODEL_OP_GEN_MANUFACTURER_PROPERTY_GET (C 宏), 425
ESP_BLE_MESH_MODEL_OP_GEN_MANUFACTURER_PROPERTY_SET (C 宏), 425
ESP_BLE_MESH_MODEL_OP_GEN_MANUFACTURER_PROPERTY_SET_UNACK (C 宏), 425
ESP_BLE_MESH_MODEL_OP_GEN_MANUFACTURER_PROPERTY_STATUS (C 宏), 425
ESP_BLE_MESH_MODEL_OP_GEN_MOVE_SET (C 宏), 423
ESP_BLE_MESH_MODEL_OP_GEN_MOVE_SET_UNACK (C 宏), 423
ESP_BLE_MESH_MODEL_OP_GEN_ONOFF_GET (C 宏), 423
ESP_BLE_MESH_MODEL_OP_GEN_ONOFF_SET (C 宏), 423
ESP_BLE_MESH_MODEL_OP_GEN_ONOFF_SET_UNACK (C 宏), 423
ESP_BLE_MESH_MODEL_OP_GEN_ONOFF_STATUS (C 宏), 423
ESP_BLE_MESH_MODEL_OP_GEN_ONPOWERUP_GET (C 宏), 423
ESP_BLE_MESH_MODEL_OP_GEN_ONPOWERUP_SET (C 宏), 423
ESP_BLE_MESH_MODEL_OP_GEN_ONPOWERUP_SET_UNACK (C 宏), 423
ESP_BLE_MESH_MODEL_OP_GEN_ONPOWERUP_STATUS (C 宏), 423
ESP_BLE_MESH_MODEL_OP_GEN_POWER_DEFAULT_GET (C 宏), 424
ESP_BLE_MESH_MODEL_OP_GEN_POWER_DEFAULT_SET (C 宏), 424
ESP_BLE_MESH_MODEL_OP_GEN_POWER_DEFAULT_SET_UNACK (C 宏), 424
ESP_BLE_MESH_MODEL_OP_GEN_POWER_DEFAULT_STATUS (C 宏), 424
ESP_BLE_MESH_MODEL_OP_GEN_POWER_LAST_GET (C 宏), 424
ESP_BLE_MESH_MODEL_OP_GEN_POWER_LAST_STATUS (C 宏), 424
ESP_BLE_MESH_MODEL_OP_GEN_POWER_LEVEL_GET (C 宏), 424
ESP_BLE_MESH_MODEL_OP_GEN_POWER_LEVEL_SET (C 宏), 424
ESP_BLE_MESH_MODEL_OP_GEN_POWER_LEVEL_SET_UNACK (C 宏), 424
ESP_BLE_MESH_MODEL_OP_GEN_POWER_LEVEL_STATUS (C 宏), 424
ESP_BLE_MESH_MODEL_OP_GEN_POWER_RANGE_GET (C 宏), 424
ESP_BLE_MESH_MODEL_OP_GEN_POWER_RANGE_SET (C 宏), 424
ESP_BLE_MESH_MODEL_OP_GEN_POWER_RANGE_SET_UNACK (C 宏), 424
ESP_BLE_MESH_MODEL_OP_GEN_POWER_RANGE_STATUS (C 宏), 424

ESP_BLE_MESH_MODEL_OP_GEN_USER_PROPERTIES_GET (C 宏), 425
ESP_BLE_MESH_MODEL_OP_GEN_USER_PROPERTIES_STATUS (C 宏), 425
ESP_BLE_MESH_MODEL_OP_GEN_USER_PROPERTY_GET (C 宏), 425
ESP_BLE_MESH_MODEL_OP_GEN_USER_PROPERTY_SET (C 宏), 425
ESP_BLE_MESH_MODEL_OP_GEN_USER_PROPERTY_SET_UNACK (C 宏), 425
ESP_BLE_MESH_MODEL_OP_GEN_USER_PROPERTY_STATUS (C 宏), 425
ESP_BLE_MESH_MODEL_OP_HEALTH_CURRENT_STATUS (C 宏), 423
ESP_BLE_MESH_MODEL_OP_HEALTH_FAULT_CLEAR (C 宏), 422
ESP_BLE_MESH_MODEL_OP_HEALTH_FAULT_CLEAR_UNACK (C 宏), 422
ESP_BLE_MESH_MODEL_OP_HEALTH_FAULT_GET (C 宏), 422
ESP_BLE_MESH_MODEL_OP_HEALTH_FAULT_STATUS (C 宏), 423
ESP_BLE_MESH_MODEL_OP_HEALTH_FAULT_TEST (C 宏), 422
ESP_BLE_MESH_MODEL_OP_HEALTH_FAULT_TEST_UNACK (C 宏), 422
ESP_BLE_MESH_MODEL_OP_HEALTH_PERIOD_GET (C 宏), 422
ESP_BLE_MESH_MODEL_OP_HEALTH_PERIOD_SET (C 宏), 422
ESP_BLE_MESH_MODEL_OP_HEALTH_PERIOD_SET_UNACK (C 宏), 422
ESP_BLE_MESH_MODEL_OP_HEALTH_PERIOD_STATUS (C 宏), 423
ESP_BLE_MESH_MODEL_OP_HEARTBEAT_PUB_GET (C 宏), 418
ESP_BLE_MESH_MODEL_OP_HEARTBEAT_PUB_SET (C 宏), 420
ESP_BLE_MESH_MODEL_OP_HEARTBEAT_PUB_STATUS (C 宏), 421
ESP_BLE_MESH_MODEL_OP_HEARTBEAT_SUB_GET (C 宏), 418
ESP_BLE_MESH_MODEL_OP_HEARTBEAT_SUB_SET (C 宏), 420
ESP_BLE_MESH_MODEL_OP_HEARTBEAT_SUB_STATUS (C 宏), 421
ESP_BLE_MESH_MODEL_OP_KEY_REFRESH_PHASE_GET (C 宏), 419
ESP_BLE_MESH_MODEL_OP_KEY_REFRESH_PHASE_SET (C 宏), 420
ESP_BLE_MESH_MODEL_OP_KEY_REFRESH_PHASE_STATUS (C 宏), 421
ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_DEFAULT_GET (C 宏), 428
ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_DEFAULT_SET (C 宏), 428
ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_DEFAULT_SET_UNACK (C 宏), 428
ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_DEFAULT_STATUS (C 宏), 428
ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_GET (C 宏), 427
ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_SET (C 宏), 428
ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_SET_UNACK (C 宏), 428
ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_STATUS (C 宏), 428
ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_TEMPERATURE_GET (C 宏), 428
ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_TEMPERATURE_RANGE_GET (C 宏), 428
ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_TEMPERATURE_RANGE_SET (C 宏), 428
ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_TEMPERATURE_RANGE_SET_UNAC (C 宏), 428
ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_TEMPERATURE_RANGE_STATUS (C 宏), 428
ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_TEMPERATURE_SET (C 宏), 428
ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_TEMPERATURE_SET_UNACK (C 宏), 428
ESP_BLE_MESH_MODEL_OP_LIGHT_CTL_TEMPERATURE_STATUS (C 宏), 428

ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_DEFAULT_GET (C 宏), 429
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_DEFAULT_SET (C 宏), 429
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_DEFAULT_SET_UNACK (C 宏), 429
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_DEFAULT_STATUS (C 宏), 429
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_GET (C 宏), 428
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_HUE_GET (C 宏), 428
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_HUE_SET (C 宏), 428
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_HUE_SET_UNACK (C 宏), 428
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_HUE_STATUS (C 宏), 428
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_RANGE_GET (C 宏), 429
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_RANGE_SET (C 宏), 429
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_RANGE_SET_UNACK (C 宏), 429
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_RANGE_STATUS (C 宏), 429
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_SATURATION_GET (C 宏), 428
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_SATURATION_SET (C 宏), 428
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_SATURATION_SET_UNACK (C 宏), 428
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_SATURATION_STATUS (C 宏), 428
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_SET (C 宏), 428
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_SET_UNACK (C 宏), 428
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_STATUS (C 宏), 428
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_TARGET_GET (C 宏), 428
ESP_BLE_MESH_MODEL_OP_LIGHT_HSL_TARGET_STATUS (C 宏), 428
ESP_BLE_MESH_MODEL_OP_LIGHT_LC_LIGHT_ONOFF_GET (C 宏), 430
ESP_BLE_MESH_MODEL_OP_LIGHT_LC_LIGHT_ONOFF_SET (C 宏), 430
ESP_BLE_MESH_MODEL_OP_LIGHT_LC_LIGHT_ONOFF_SET_UNACK (C 宏), 430
ESP_BLE_MESH_MODEL_OP_LIGHT_LC_LIGHT_ONOFF_STATUS (C 宏), 430
ESP_BLE_MESH_MODEL_OP_LIGHT_LC_MODE_GET (C 宏), 429
ESP_BLE_MESH_MODEL_OP_LIGHT_LC_MODE_SET (C 宏), 429
ESP_BLE_MESH_MODEL_OP_LIGHT_LC_MODE_SET_UNACK (C 宏), 429
ESP_BLE_MESH_MODEL_OP_LIGHT_LC_MODE_STATUS (C 宏), 429
ESP_BLE_MESH_MODEL_OP_LIGHT_LC_OM_GET (C 宏), 429
ESP_BLE_MESH_MODEL_OP_LIGHT_LC_OM_SET (C 宏), 430
ESP_BLE_MESH_MODEL_OP_LIGHT_LC_OM_SET_UNACK (C 宏), 430
ESP_BLE_MESH_MODEL_OP_LIGHT_LC_OM_STATUS (C 宏), 430
ESP_BLE_MESH_MODEL_OP_LIGHT_LC_PROPERTY_GET (C 宏), 430
ESP_BLE_MESH_MODEL_OP_LIGHT_LC_PROPERTY_SET (C 宏), 430
ESP_BLE_MESH_MODEL_OP_LIGHT_LC_PROPERTY_SET_UNACK (C 宏), 430
ESP_BLE_MESH_MODEL_OP_LIGHT_LC_PROPERTY_STATUS (C 宏), 430
ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_DEFAULT_GET (C 宏), 427
ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_DEFAULT_SET (C 宏), 427
ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_DEFAULT_SET_UNACK (C 宏), 427
ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_DEFAULT_STATUS (C 宏), 427

ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_GET (C 宏), 427
ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_LAST_GET (C 宏), 427
ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_LAST_STATUS (C 宏), 427
ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_LINEAR_GET (C 宏), 427
ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_LINEAR_SET (C 宏), 427
ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_LINEAR_SET_UNACK (C 宏), 427
ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_LINEAR_STATUS (C 宏), 427
ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_RANGE_GET (C 宏), 427
ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_RANGE_SET (C 宏), 427
ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_RANGE_SET_UNACK (C 宏), 427
ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_RANGE_STATUS (C 宏), 427
ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_SET (C 宏), 427
ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_SET_UNACK (C 宏), 427
ESP_BLE_MESH_MODEL_OP_LIGHT_LIGHTNESS_STATUS (C 宏), 427
ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_DEFAULT_GET (C 宏), 429
ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_DEFAULT_SET (C 宏), 429
ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_DEFAULT_SET_UNACK (C 宏), 429
ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_DEFAULT_STATUS (C 宏), 429
ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_GET (C 宏), 429
ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_RANGE_GET (C 宏), 429
ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_RANGE_SET (C 宏), 429
ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_RANGE_SET_UNACK (C 宏), 429
ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_RANGE_STATUS (C 宏), 429
ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_SET (C 宏), 429
ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_SET_UNACK (C 宏), 429
ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_STATUS (C 宏), 429
ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_TARGET_GET (C 宏), 429
ESP_BLE_MESH_MODEL_OP_LIGHT_XYL_TARGET_STATUS (C 宏), 429
ESP_BLE_MESH_MODEL_OP_LPN_POLLTIMEOUT_GET (C 宏), 419
ESP_BLE_MESH_MODEL_OP_LPN_POLLTIMEOUT_STATUS (C 宏), 421
ESP_BLE_MESH_MODEL_OP_MODEL_APP_BIND (C 宏), 420
ESP_BLE_MESH_MODEL_OP_MODEL_APP_STATUS (C 宏), 421
ESP_BLE_MESH_MODEL_OP_MODEL_APP_UNBIND (C 宏), 420
ESP_BLE_MESH_MODEL_OP_MODEL_PUB_GET (C 宏), 418
ESP_BLE_MESH_MODEL_OP_MODEL_PUB_SET (C 宏), 419
ESP_BLE_MESH_MODEL_OP_MODEL_PUB_STATUS (C 宏), 421
ESP_BLE_MESH_MODEL_OP_MODEL_PUB_VIRTUAL_ADDR_SET (C 宏), 420
ESP_BLE_MESH_MODEL_OP_MODEL_SUB_ADD (C 宏), 419
ESP_BLE_MESH_MODEL_OP_MODEL_SUB_DELETE (C 宏), 419
ESP_BLE_MESH_MODEL_OP_MODEL_SUB_DELETE_ALL (C 宏), 420
ESP_BLE_MESH_MODEL_OP_MODEL_SUB_OVERWRITE (C 宏), 419
ESP_BLE_MESH_MODEL_OP_MODEL_SUB_STATUS (C 宏), 421

ESP_BLE_MESH_MODEL_OP_MODEL_SUB_VIRTUAL_ADDR_ADD (C 宏), 419
ESP_BLE_MESH_MODEL_OP_MODEL_SUB_VIRTUAL_ADDR_DELETE (C 宏), 419
ESP_BLE_MESH_MODEL_OP_MODEL_SUB_VIRTUAL_ADDR_OVERWRITE (C 宏), 419
ESP_BLE_MESH_MODEL_OP_NET_KEY_ADD (C 宏), 420
ESP_BLE_MESH_MODEL_OP_NET_KEY_DELETE (C 宏), 420
ESP_BLE_MESH_MODEL_OP_NET_KEY_GET (C 宏), 418
ESP_BLE_MESH_MODEL_OP_NET_KEY_LIST (C 宏), 421
ESP_BLE_MESH_MODEL_OP_NET_KEY_STATUS (C 宏), 421
ESP_BLE_MESH_MODEL_OP_NET_KEY_UPDATE (C 宏), 420
ESP_BLE_MESH_MODEL_OP_NETWORK_TRANSMIT_GET (C 宏), 419
ESP_BLE_MESH_MODEL_OP_NETWORK_TRANSMIT_SET (C 宏), 420
ESP_BLE_MESH_MODEL_OP_NETWORK_TRANSMIT_STATUS (C 宏), 421
ESP_BLE_MESH_MODEL_OP_NODE_IDENTITY_GET (C 宏), 418
ESP_BLE_MESH_MODEL_OP_NODE_IDENTITY_SET (C 宏), 420
ESP_BLE_MESH_MODEL_OP_NODE_IDENTITY_STATUS (C 宏), 421
ESP_BLE_MESH_MODEL_OP_NODE_RESET (C 宏), 420
ESP_BLE_MESH_MODEL_OP_NODE_RESET_STATUS (C 宏), 421
ESP_BLE_MESH_MODEL_OP_RELAY_GET (C 宏), 418
ESP_BLE_MESH_MODEL_OP_RELAY_SET (C 宏), 419
ESP_BLE_MESH_MODEL_OP_RELAY_STATUS (C 宏), 421
ESP_BLE_MESH_MODEL_OP_SCENE_DELETE (C 宏), 427
ESP_BLE_MESH_MODEL_OP_SCENE_DELETE_UNACK (C 宏), 427
ESP_BLE_MESH_MODEL_OP_SCENE_GET (C 宏), 426
ESP_BLE_MESH_MODEL_OP_SCENE_RECALL (C 宏), 426
ESP_BLE_MESH_MODEL_OP_SCENE_RECALL_UNACK (C 宏), 426
ESP_BLE_MESH_MODEL_OP_SCENE_REGISTER_GET (C 宏), 426
ESP_BLE_MESH_MODEL_OP_SCENE_REGISTER_STATUS (C 宏), 426
ESP_BLE_MESH_MODEL_OP_SCENE_STATUS (C 宏), 426
ESP_BLE_MESH_MODEL_OP_SCENE_STORE (C 宏), 426
ESP_BLE_MESH_MODEL_OP_SCENE_STORE_UNACK (C 宏), 426
ESP_BLE_MESH_MODEL_OP_SCHEDULER_ACT_GET (C 宏), 427
ESP_BLE_MESH_MODEL_OP_SCHEDULER_ACT_SET (C 宏), 427
ESP_BLE_MESH_MODEL_OP_SCHEDULER_ACT_SET_UNACK (C 宏), 427
ESP_BLE_MESH_MODEL_OP_SCHEDULER_ACT_STATUS (C 宏), 427
ESP_BLE_MESH_MODEL_OP_SCHEDULER_GET (C 宏), 427
ESP_BLE_MESH_MODEL_OP_SCHEDULER_STATUS (C 宏), 427
ESP_BLE_MESH_MODEL_OP_SENSOR_CADENCE_GET (C 宏), 425
ESP_BLE_MESH_MODEL_OP_SENSOR_CADENCE_SET (C 宏), 426
ESP_BLE_MESH_MODEL_OP_SENSOR_CADENCE_SET_UNACK (C 宏), 426
ESP_BLE_MESH_MODEL_OP_SENSOR_CADENCE_STATUS (C 宏), 426
ESP_BLE_MESH_MODEL_OP_SENSOR_COLUMN_GET (C 宏), 425
ESP_BLE_MESH_MODEL_OP_SENSOR_COLUMN_STATUS (C 宏), 425
ESP_BLE_MESH_MODEL_OP_SENSOR_DESCRIPTOR_GET (C 宏), 425
ESP_BLE_MESH_MODEL_OP_SENSOR_DESCRIPTOR_STATUS (C 宏), 425
ESP_BLE_MESH_MODEL_OP_SENSOR_GET (C 宏), 425
ESP_BLE_MESH_MODEL_OP_SENSOR_SERIES_GET (C 宏), 425
ESP_BLE_MESH_MODEL_OP_SENSOR_SERIES_STATUS (C 宏), 425
ESP_BLE_MESH_MODEL_OP_SENSOR_SETTING_GET (C 宏), 426
ESP_BLE_MESH_MODEL_OP_SENSOR_SETTING_SET (C

宏), 426
ESP_BLE_MESH_MODEL_OP_SENSOR_SETTING_SET_UNACK (C 宏), 426
ESP_BLE_MESH_MODEL_OP_SENSOR_SETTING_STATUS (C 宏), 426
ESP_BLE_MESH_MODEL_OP_SENSOR_SETTINGS_GET (C 宏), 426
ESP_BLE_MESH_MODEL_OP_SENSOR_SETTINGS_STATUS (C 宏), 426
ESP_BLE_MESH_MODEL_OP_SENSOR_STATUS (C 宏), 425
ESP_BLE_MESH_MODEL_OP_SIG_MODEL_APP_GET (C 宏), 419
ESP_BLE_MESH_MODEL_OP_SIG_MODEL_APP_LIST (C 宏), 421
ESP_BLE_MESH_MODEL_OP_SIG_MODEL_SUB_GET (C 宏), 418
ESP_BLE_MESH_MODEL_OP_SIG_MODEL_SUB_LIST (C 宏), 421
esp_ble_mesh_model_op_t (C++ 类), 402
esp_ble_mesh_model_op_t::min_len (C++ 成员), 402
esp_ble_mesh_model_op_t::opcode (C++ 成员), 402
esp_ble_mesh_model_op_t::param_cb (C++ 成员), 402
ESP_BLE_MESH_MODEL_OP_TAI_UTC_DELTA_GET (C 宏), 426
ESP_BLE_MESH_MODEL_OP_TAI_UTC_DELTA_SET (C 宏), 426
ESP_BLE_MESH_MODEL_OP_TAI_UTC_DELTA_STATUS (C 宏), 426
ESP_BLE_MESH_MODEL_OP_TIME_GET (C 宏), 426
ESP_BLE_MESH_MODEL_OP_TIME_ROLE_GET (C 宏), 426
ESP_BLE_MESH_MODEL_OP_TIME_ROLE_SET (C 宏), 426
ESP_BLE_MESH_MODEL_OP_TIME_ROLE_STATUS (C 宏), 426
ESP_BLE_MESH_MODEL_OP_TIME_SET (C 宏), 426
ESP_BLE_MESH_MODEL_OP_TIME_STATUS (C 宏), 426
ESP_BLE_MESH_MODEL_OP_TIME_ZONE_GET (C 宏), 426
ESP_BLE_MESH_MODEL_OP_TIME_ZONE_SET (C 宏), 426
ESP_BLE_MESH_MODEL_OP_TIME_ZONE_STATUS (C 宏), 426
ESP_BLE_MESH_MODEL_OP_VENDOR_MODEL_APP_GET (C 宏), 419
ESP_BLE_MESH_MODEL_OP_VENDOR_MODEL_APP_LIST (C 宏), 421
ESP_BLE_MESH_MODEL_OP_VENDOR_MODEL_SUB_GET (C 宏), 419
ESP_BLE_MESH_MODEL_OP_VENDOR_MODEL_SUB_LIST (C 宏), 421
ESP_BLE_MESH_MODEL_OPERATION_EVT (C++ 枚举子), 439
ESP_BLE_MESH_MODEL_PUB_DEFINE (C 宏), 415
esp_ble_mesh_model_pub_t (C++ 类), 401
esp_ble_mesh_model_pub_t::app_idx (C++ 成员), 401
esp_ble_mesh_model_pub_t::count (C++ 成员), 401
esp_ble_mesh_model_pub_t::cred (C++ 成员), 401
esp_ble_mesh_model_pub_t::dev_role (C++ 成员), 402
esp_ble_mesh_model_pub_t::fast_period (C++ 成员), 401
esp_ble_mesh_model_pub_t::model (C++ 成员), 401
esp_ble_mesh_model_pub_t::msg (C++ 成员), 402
esp_ble_mesh_model_pub_t::period (C++ 成员), 401
esp_ble_mesh_model_pub_t::period_div (C++ 成员), 401
esp_ble_mesh_model_pub_t::period_start (C++ 成员), 401
esp_ble_mesh_model_pub_t::publish_addr (C++ 成员), 401
esp_ble_mesh_model_pub_t::retransmit (C++ 成员), 401
esp_ble_mesh_model_pub_t::send_rel (C++ 成员), 401

esp_ble_mesh_model_pub_t::timer (C++ 成员), 402
esp_ble_mesh_model_pub_t::ttl (C++ 成员), 401
esp_ble_mesh_model_pub_t::update (C++ 成员), 402
esp_ble_mesh_model_publish (C++ 函数), 446
ESP_BLE_MESH_MODEL_PUBLISH_COMP_EVT (C++ 枚举子), 439
ESP_BLE_MESH_MODEL_PUBLISH_UPDATE_EVT (C++ 枚举子), 439
ESP_BLE_MESH_MODEL_SCENE_CLI (C 宏), 647
ESP_BLE_MESH_MODEL_SCENE_SETUP_SRV (C 宏), 648
ESP_BLE_MESH_MODEL_SCENE_SRV (C 宏), 648
ESP_BLE_MESH_MODEL_SCHEDULER_CLI (C 宏), 647
ESP_BLE_MESH_MODEL_SCHEDULER_SETUP_SRV (C 宏), 649
ESP_BLE_MESH_MODEL_SCHEDULER_SRV (C 宏), 648
ESP_BLE_MESH_MODEL_SEND_COMP_EVT (C++ 枚举子), 439
ESP_BLE_MESH_MODEL_SENSOR_CLI (C 宏), 618
ESP_BLE_MESH_MODEL_SENSOR_SETUP_SRV (C 宏), 618
ESP_BLE_MESH_MODEL_SENSOR_SRV (C 宏), 618
ESP_BLE_MESH_MODEL_STATUS_CANNOT_SET_RANGE_MAX (C 宏), 430
ESP_BLE_MESH_MODEL_STATUS_CANNOT_SET_RANGE_MIN (C 宏), 430
ESP_BLE_MESH_MODEL_STATUS_SUCCESS (C 宏), 430
esp_ble_mesh_model_status_t (C++ 类型), 432
esp_ble_mesh_model_t (C++ 类型), 431
ESP_BLE_MESH_MODEL_TIME_CLI (C 宏), 646
ESP_BLE_MESH_MODEL_TIME_SETUP_SRV (C 宏), 647
ESP_BLE_MESH_MODEL_TIME_SRV (C 宏), 647
esp_ble_mesh_msg_ctx_t (C++ 类), 403
esp_ble_mesh_msg_ctx_t::addr (C++ 成员), 403
esp_ble_mesh_msg_ctx_t::app_idx (C++ 成员), 403
esp_ble_mesh_msg_ctx_t::model (C++ 成员), 404
esp_ble_mesh_msg_ctx_t::net_idx (C++ 成员), 403
esp_ble_mesh_msg_ctx_t::recv_dst (C++ 成员), 403
esp_ble_mesh_msg_ctx_t::recv_op (C++ 成员), 403
esp_ble_mesh_msg_ctx_t::recv_rssi (C++ 成员), 403
esp_ble_mesh_msg_ctx_t::recv_ttl (C++ 成员), 403
esp_ble_mesh_msg_ctx_t::send_rel (C++ 成员), 403
esp_ble_mesh_msg_ctx_t::send_ttl (C++ 成员), 403
esp_ble_mesh_msg_ctx_t::srv_send (C++ 成员), 404
ESP_BLE_MESH_NET_PRIMARY (C 宏), 412
ESP_BLE_MESH_NO_FAULT (C 宏), 543
ESP_BLE_MESH_NO_INPUT (C++ 枚举子), 433
ESP_BLE_MESH_NO_LOAD_ERROR (C 宏), 543
ESP_BLE_MESH_NO_LOAD_WARNING (C 宏), 543
ESP_BLE_MESH_NO_OOB (C++ 枚举子), 432
ESP_BLE_MESH_NO_OUTPUT (C++ 枚举子), 433
ESP_BLE_MESH_NODE_IDENTITY_NOT_SUPPORTED (C 宏), 412
ESP_BLE_MESH_NODE_IDENTITY_RUNNING (C 宏), 412
ESP_BLE_MESH_NODE_IDENTITY_STOPPED (C 宏), 412
esp_ble_mesh_node_input_number (C++ 函数), 452
esp_ble_mesh_node_input_string (C++ 函数), 452
esp_ble_mesh_node_is_provisioned (C++ 函数), 451
esp_ble_mesh_node_local_reset (C++ 函数), 447
ESP_BLE_MESH_NODE_NAME_MAX_LEN (C 宏), 411
ESP_BLE_MESH_NODE_PROV_COMPLETE_EVT (C++ 枚举子), 435
esp_ble_mesh_node_prov_disable (C++ 函数), 452
ESP_BLE_MESH_NODE_PROV_DISABLE_COMP_EVT (C++ 枚举子), 434
esp_ble_mesh_node_prov_enable (C++ 函数), 451
ESP_BLE_MESH_NODE_PROV_ENABLE_COMP_EVT (C++ 枚举子), 434
ESP_BLE_MESH_NODE_PROV_INPUT_EVT (C++ 枚举子), 435

ESP_BLE_MESH_NODE_PROV_INPUT_NUMBER_COMP_EVT (C++ 枚举子), 435
ESP_BLE_MESH_NODE_PROV_INPUT_STRING_COMP_EVT (C++ 枚举子), 435
ESP_BLE_MESH_NODE_PROV_LINK_CLOSE_EVT (C++ 枚举子), 435
ESP_BLE_MESH_NODE_PROV_LINK_OPEN_EVT (C++ 枚举子), 434
ESP_BLE_MESH_NODE_PROV_OOB_PUB_KEY_EVT (C++ 枚举子), 435
ESP_BLE_MESH_NODE_PROV_OUTPUT_NUMBER_EVT (C++ 枚举子), 435
ESP_BLE_MESH_NODE_PROV_OUTPUT_STRING_EVT (C++ 枚举子), 435
ESP_BLE_MESH_NODE_PROV_RESET_EVT (C++ 枚举子), 435
ESP_BLE_MESH_NODE_PROV_SET_OOB_PUB_KEY_COMP_EVT (C++ 枚举子), 435
ESP_BLE_MESH_NODE_PROXY_GATT_DISABLE_COMP_EVT (C++ 枚举子), 435
ESP_BLE_MESH_NODE_PROXY_GATT_ENABLE_COMP_EVT (C++ 枚举子), 435
ESP_BLE_MESH_NODE_PROXY_IDENTITY_ENABLE_COMP_EVT (C++ 枚举子), 435
esp_ble_mesh_node_set_oob_pub_key (C++ 函数), 452
ESP_BLE_MESH_NODE_SET_UNPROV_DEV_NAME_COMP_EVT (C++ 枚举子), 434
esp_ble_mesh_node_t (C++ 类), 405
esp_ble_mesh_node_t::addr (C++ 成员), 405
esp_ble_mesh_node_t::addr_type (C++ 成员), 405
esp_ble_mesh_node_t::comp_data (C++ 成员), 406
esp_ble_mesh_node_t::comp_length (C++ 成员), 406
esp_ble_mesh_node_t::dev_key (C++ 成员), 406
esp_ble_mesh_node_t::dev_uuid (C++ 成员), 406
esp_ble_mesh_node_t::element_num (C++ 成员), 406
esp_ble_mesh_node_t::flags (C++ 成员), 406
esp_ble_mesh_node_t::iv_index (C++ 成员), 406
esp_ble_mesh_node_t::name (C++ 成员), 406
esp_ble_mesh_node_t::net_idx (C++ 成员), 406
esp_ble_mesh_node_t::oob_info (C++ 成员), 406
esp_ble_mesh_node_t::unicast_addr (C++ 成员), 406
ESP_BLE_MESH_OCTET16_LEN (C 宏), 411
esp_ble_mesh_octet16_t (C++ 类型), 430
ESP_BLE_MESH_OCTET8_LEN (C 宏), 411
esp_ble_mesh_octet8_t (C++ 类型), 430
esp_ble_mesh_oob_method_t (C++ 类型), 432
esp_ble_mesh_opcode_config_client_get_t (C++ 类型), 431
esp_ble_mesh_opcode_config_client_set_t (C++ 类型), 431
esp_ble_mesh_opcode_config_status_t (C++ 类型), 431
esp_ble_mesh_opcode_health_client_get_t (C++ 类型), 431
esp_ble_mesh_opcode_health_client_set_t (C++ 类型), 431
esp_ble_mesh_opcode_t (C++ 类型), 432
esp_ble_mesh_output_action_t (C++ 类型), 433
ESP_BLE_MESH_OUTPUT_OOB (C++ 枚举子), 432
ESP_BLE_MESH_OVERFLOW_ERROR (C 宏), 544
ESP_BLE_MESH_OVERFLOW_WARNING (C 宏), 544
ESP_BLE_MESH_OVERHEAT_ERROR (C 宏), 543
ESP_BLE_MESH_OVERHEAT_WARNING (C 宏), 543
ESP_BLE_MESH_OVERLOAD_ERROR (C 宏), 543
ESP_BLE_MESH_OVERLOAD_WARNING (C 宏), 543
ESP_BLE_MESH_POWER_SUPPLY_INTERRUPTED_ERROR (C 宏), 543
ESP_BLE_MESH_POWER_SUPPLY_INTERRUPTED_WARNING (C 宏), 543
ESP_BLE_MESH_PROV (C 宏), 415
ESP_BLE_MESH_PROV_ADV (C++ 枚举子), 433
esp_ble_mesh_prov_adv_cb_t (C++ 类型), 458
esp_ble_mesh_prov_bearer_t (C++ 类型), 433
esp_ble_mesh_prov_cb_event_t (C++ 类型), 434
esp_ble_mesh_prov_cb_param_t (C++ 类型), 375
esp_ble_mesh_prov_cb_param_t::__anonymous20 (C++ 类型), 380
esp_ble_mesh_prov_cb_param_t::ble_mesh_deinit_mesh_comp_pa

(C++ 类), 379
esp_ble_mesh_prov_cb_param_t::ble_mesh_deinit_mesh_comp_param::err_code (C++ 成员), 379
esp_ble_mesh_prov_cb_param_t::ble_mesh_friend_establish_param (C++ 类), 379
esp_ble_mesh_prov_cb_param_t::ble_mesh_friend_establish_param::lpn_addr (C++ 成员), 379
esp_ble_mesh_prov_cb_param_t::ble_mesh_friend_establish_param::lpn_num (C++ 成员), 379
esp_ble_mesh_prov_cb_param_t::ble_mesh_heartbeat_msg_recv_param (C++ 类), 380
esp_ble_mesh_prov_cb_param_t::ble_mesh_heartbeat_msg_recv_param::feature (C++ 成员), 380
esp_ble_mesh_prov_cb_param_t::ble_mesh_heartbeat_msg_recv_param::hops (C++ 成员), 380
esp_ble_mesh_prov_cb_param_t::ble_mesh_input_event_param (C++ 类), 380
esp_ble_mesh_prov_cb_param_t::ble_mesh_input_event_param::action (C++ 成员), 381
esp_ble_mesh_prov_cb_param_t::ble_mesh_input_event_param::size (C++ 成员), 381
esp_ble_mesh_prov_cb_param_t::ble_mesh_input_number_comp_param (C++ 类), 381
esp_ble_mesh_prov_cb_param_t::ble_mesh_input_number_comp_param::err_code (C++ 成员), 381
esp_ble_mesh_prov_cb_param_t::ble_mesh_input_string_comp_param (C++ 类), 381
esp_ble_mesh_prov_cb_param_t::ble_mesh_input_string_comp_param::err_code (C++ 成员), 381
esp_ble_mesh_prov_cb_param_t::ble_mesh_link_close_param (C++ 类), 381
esp_ble_mesh_prov_cb_param_t::ble_mesh_link_close_param::reason (C++ 成员), 381
esp_ble_mesh_prov_cb_param_t::ble_mesh_link_open_param (C++ 类), 381
esp_ble_mesh_prov_cb_param_t::ble_mesh_link_open_param::bearer (C++ 成员), 381
esp_ble_mesh_prov_cb_param_t::ble_mesh_lpn_disable_comp_param (C++ 类), 381
esp_ble_mesh_prov_cb_param_t::ble_mesh_lpn_disable_comp_param::err_code (C++ 成员), 382
esp_ble_mesh_prov_cb_param_t::ble_mesh_lpn_enable_comp_param (C++ 类), 382
esp_ble_mesh_prov_cb_param_t::ble_mesh_lpn_enable_comp_param::lpn_addr (C++ 成员), 382
esp_ble_mesh_prov_cb_param_t::ble_mesh_lpn_friendship_establish_param (C++ 类), 382
esp_ble_mesh_prov_cb_param_t::ble_mesh_lpn_friendship_establish_param::lpn_addr (C++ 成员), 382
esp_ble_mesh_prov_cb_param_t::ble_mesh_lpn_friendship_terminate_param (C++ 类), 382
esp_ble_mesh_prov_cb_param_t::ble_mesh_lpn_friendship_terminate_param::reason (C++ 成员), 382
esp_ble_mesh_prov_cb_param_t::ble_mesh_lpn_friendship_terminate_param::src (C++ 成员), 382
esp_ble_mesh_prov_cb_param_t::ble_mesh_lpn_poll_comp_param (C++ 类), 382
esp_ble_mesh_prov_cb_param_t::ble_mesh_lpn_poll_comp_param::err_code (C++ 成员), 382
esp_ble_mesh_prov_cb_param_t::ble_mesh_output_num_evt_param (C++ 类), 382
esp_ble_mesh_prov_cb_param_t::ble_mesh_output_num_evt_param::action (C++ 成员), 382
esp_ble_mesh_prov_cb_param_t::ble_mesh_output_num_evt_param::size (C++ 成员), 382
esp_ble_mesh_prov_cb_param_t::ble_mesh_output_str_evt_param (C++ 类), 383
esp_ble_mesh_prov_cb_param_t::ble_mesh_output_str_evt_param::err_code (C++ 成员), 383
esp_ble_mesh_prov_cb_param_t::ble_mesh_prov_disable_comp_param (C++ 类), 383
esp_ble_mesh_prov_cb_param_t::ble_mesh_prov_disable_comp_param::err_code (C++ 成员), 383
esp_ble_mesh_prov_cb_param_t::ble_mesh_prov_enable_comp_param (C++ 类), 383
esp_ble_mesh_prov_cb_param_t::ble_mesh_prov_enable_comp_param::err_code (C++ 成员), 383
esp_ble_mesh_prov_cb_param_t::ble_mesh_prov_register_comp_param (C++ 类), 383
esp_ble_mesh_prov_cb_param_t::ble_mesh_prov_register_comp_param::err_code (C++ 成员), 383
esp_ble_mesh_prov_cb_param_t::ble_mesh_provision_complete_param (C++ 类), 383

(C++ 类), 383
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_delete_comp_evt_param::addr (C++ 成员), 384
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_delete_comp_evt_param::addr_type (C++ 成员), 384
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_delete_comp_evt_param::app_idx (C++ 成员), 384
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_delete_comp_evt_param::element_num (C++ 成员), 383
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_delete_comp_evt_param::net_idx (C++ 成员), 383
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_delete_comp_evt_param::status (C++ 成员), 383
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_delete_comp_evt_param::uuid (C++ 成员), 384
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_delete_dev_param (C++ 类), 384
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_delete_dev_param::addr (C++ 成员), 384
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_delete_dev_param::addr_type (C++ 成员), 384
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_delete_dev_param::flag (C++ 成员), 384
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_delete_dev_param::index (C++ 成员), 384
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_delete_dev_param::net_idx (C++ 成员), 384
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_delete_dev_param::status (C++ 成员), 384
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_delete_dev_param::uuid (C++ 成员), 384
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_link_close_param (C++ 类), 384
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_link_close_param::addr (C++ 成员), 384
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_link_close_param::addr_type (C++ 成员), 384
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_link_close_param::bearer (C++ 成员), 384
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_link_close_param::net_idx (C++ 成员), 384
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_link_close_param::reason (C++ 成员), 384
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_link_open_param (C++ 类), 384
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_link_open_param::addr (C++ 成员), 384
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_link_open_param::addr_type (C++ 成员), 384
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_link_open_param::bearer (C++ 成员), 384
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_prov_complete_param (C++ 类), 384
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_prov_complete_param::addr (C++ 成员), 384
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_prov_complete_param::addr_type (C++ 成员), 384
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_prov_complete_param::bearer (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_prov_complete_param::element_num (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_prov_complete_param::net_idx (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_prov_complete_param::oob_info (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_prov_complete_param::provisioner_addr (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_prov_complete_param::provisioner_addr_type (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_prov_complete_param::uuid (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_prov_dev_comp_param (C++ 类), 385
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_prov_dev_comp_param::err_code (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_prov_dev_comp_param::num_comp_data_pages (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_recv_unprov_adv_pkt_param (C++ 类), 385
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_recv_unprov_adv_pkt_param::addr (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_recv_unprov_adv_pkt_param::addr_type (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_recv_unprov_adv_pkt_param::adv_type (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_recv_unprov_adv_pkt_param::bearer (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_recv_unprov_adv_pkt_param::dev_type (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_recv_unprov_adv_pkt_param::oob_info (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_recv_unprov_adv_pkt_param::oob_type (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_recv_unprov_adv_pkt_param::uri_hash (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_recv_unprov_adv_pkt_param::uuid (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::ble_mesh_set_oob_pub_key_comp_param (C++ 类), 385
esp_ble_mesh_prov_cb_param_t::ble_mesh_set_oob_pub_key_comp_param::err_code (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::ble_mesh_set_unprov_dev_name_comp_param (C++ 类), 385
esp_ble_mesh_prov_cb_param_t::ble_mesh_set_unprov_dev_name_comp_param::err_code (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::ble_mesh_unprovisioned_beacon_param (C++ 类), 385
esp_ble_mesh_prov_cb_param_t::ble_mesh_unprovisioned_beacon_param::oob_info (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::ble_mesh_unprovisioned_beacon_param::uri_hash (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::ble_mesh_unprovisioned_beacon_param::uuid (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::node_prov_complete_param (C++ 类), 385
esp_ble_mesh_prov_cb_param_t::node_prov_complete_param::addr (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::node_prov_complete_param::bearer (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::node_prov_complete_param::net_idx (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::node_prov_complete_param::oob_info (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::node_prov_complete_param::primary_addr (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::node_prov_complete_param::uuid (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::node_prov_enable_comp_param (C++ 类), 385
esp_ble_mesh_prov_cb_param_t::node_prov_enable_comp_param::err_code (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::node_prov_input_num_param (C++ 类), 385
esp_ble_mesh_prov_cb_param_t::node_prov_input_num_param::number (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::node_prov_input_num_param::size (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::node_prov_input_param (C++ 类), 385
esp_ble_mesh_prov_cb_param_t::node_prov_input_param::action (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::node_prov_input_param::size (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::node_prov_link_close_param (C++ 类), 385
esp_ble_mesh_prov_cb_param_t::node_prov_link_close_param::bearer (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::node_prov_link_close_param::reason (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::node_prov_link_open_param (C++ 类), 385
esp_ble_mesh_prov_cb_param_t::node_prov_link_open_param::bearer (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::node_prov_output_num_param (C++ 类), 385
esp_ble_mesh_prov_cb_param_t::node_prov_output_num_param::number (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::node_prov_output_num_param::size (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::node_prov_output_param (C++ 类), 385
esp_ble_mesh_prov_cb_param_t::node_prov_output_param::action (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::node_prov_output_param::size (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::node_prov_reset_param (C++ 类), 385
esp_ble_mesh_prov_cb_param_t::node_prov_reset_param::bearer (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::node_prov_set_node_name_comp_param (C++ 类), 385
esp_ble_mesh_prov_cb_param_t::node_prov_set_node_name_comp_param::err_code (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::node_prov_set_node_name_comp_param::node_index (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::node_prov_set_oob_pub_key_param (C++ 类), 385
esp_ble_mesh_prov_cb_param_t::node_prov_set_oob_pub_key_param::pub_key_x (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::node_prov_set_oob_pub_key_param::pub_key_y (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::node_prov_set_oob_pub_key_param::private_key (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::node_prov_set_unprov_dev_name_comp_param (C++ 类), 385
esp_ble_mesh_prov_cb_param_t::node_prov_set_unprov_dev_name_comp_param::err_code (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::node_prov_unprov_beacon_param (C++ 类), 385
esp_ble_mesh_prov_cb_param_t::node_prov_unprov_beacon_param::oob_info (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::node_prov_unprov_beacon_param::uri_hash (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::node_prov_unprov_beacon_param::uuid (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_add_unprov_dev_comp_param (C++ 类), 385
esp_ble_mesh_prov_cb_param_t::provisioner_add_unprov_dev_comp_param::err_code (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_delete_comp_param (C++ 类), 385
esp_ble_mesh_prov_cb_param_t::provisioner_delete_comp_param::err_code (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_delete_comp_param::num_devices (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_delete_comp_param::type (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_dev_add_comp_param (C++ 类), 385
esp_ble_mesh_prov_cb_param_t::provisioner_dev_add_comp_param::addr (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_dev_add_comp_param::addr_type (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_dev_add_comp_param::err_code (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_dev_add_comp_param::flag (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_dev_add_comp_param::net_idx (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_dev_add_comp_param::oob_info (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_dev_add_comp_param::uuid (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_prov_read_oob_pub_key_param (C++ 类), 385
esp_ble_mesh_prov_cb_param_t::provisioner_prov_read_oob_pub_key_param::bearer (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_prov_read_oob_pub_key_param::link_idx (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_prov_read_oob_pub_key_param::pub_key_x (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_prov_read_oob_pub_key_param::pub_key_y (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_prov_read_oob_pub_key_param::prov_idx (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_prov_read_oob_pub_key_param::provisioner_int_pub_key_x (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_prov_read_oob_pub_key_param::provisioner_int_pub_key_y (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_prov_read_oob_pub_key_param::provisioner_int_private_key (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_prov_read_oob_pub_key_param::provisioner_link_idx (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_recv_unprov_adv_pkt_param (C++ 类), 385
esp_ble_mesh_prov_cb_param_t::provisioner_recv_unprov_adv_pkt_param::addr (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_recv_unprov_adv_pkt_param::addr_type (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_recv_unprov_adv_pkt_param::adv_type (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_recv_unprov_adv_pkt_param::bearer (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_recv_unprov_adv_pkt_param::dev_type (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_recv_unprov_adv_pkt_param::oob_info (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_recv_unprov_adv_pkt_param::oob_type (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_recv_unprov_adv_pkt_param::uri_hash (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_recv_unprov_adv_pkt_param::uuid (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_set_dev_uuid_match_comp_param (C++ 类), 385
esp_ble_mesh_prov_cb_param_t::provisioner_set_dev_uuid_match_comp_param::err_code (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_set_node_name_comp_param (C++ 类), 385
esp_ble_mesh_prov_cb_param_t::provisioner_set_node_name_comp_param::err_code (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_set_node_name_comp_param::node_index (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_set_oob_pub_key_param (C++ 类), 385
esp_ble_mesh_prov_cb_param_t::provisioner_set_oob_pub_key_param::pub_key_x (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_set_oob_pub_key_param::pub_key_y (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_set_oob_pub_key_param::private_key (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_set_prov_data_info_comp_param (C++ 类), 385
esp_ble_mesh_prov_cb_param_t::provisioner_set_prov_data_info_comp_param::err_code (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_set_static_oob_val_comp_param (C++ 类), 385
esp_ble_mesh_prov_cb_param_t::provisioner_set_static_oob_val_comp_param::err_code (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_set_unprov_dev_name_comp_param (C++ 类), 385
esp_ble_mesh_prov_cb_param_t::provisioner_set_unprov_dev_name_comp_param::err_code (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_upper_ack_recv_param (C++ 类), 385
esp_ble_mesh_prov_cb_param_t::provisioner_upper_ack_recv_param::addr (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_upper_ack_recv_param::ctx (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_upper_ack_recv_param::opcode (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_upper_ack_recv_param::status (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_upper_ack_recv_param::timeout (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_upper_ack_recv_param::type (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_upper_adv_recv_param (C++ 类), 385
esp_ble_mesh_prov_cb_param_t::provisioner_upper_adv_recv_param::ctx (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_upper_adv_recv_param::length (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_upper_adv_recv_param::msg (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_upper_adv_recv_param::opcode (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_upper_adv_recv_param::type (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_upper_param (C++ 类), 385
esp_ble_mesh_prov_cb_param_t::provisioner_upper_param::ctx (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_upper_param::length (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_upper_param::msg (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_upper_param::opcode (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_upper_param::type (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_upper_unseg_recv_param (C++ 类), 385
esp_ble_mesh_prov_cb_param_t::provisioner_upper_unseg_recv_param::ctx (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_upper_unseg_recv_param::length (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_upper_unseg_recv_param::msg (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_upper_unseg_recv_param::opcode (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_upper_unseg_recv_param::type (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_upper_update_comp_param (C++ 类), 385
esp_ble_mesh_prov_cb_param_t::provisioner_upper_update_comp_param::err_code (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_upper_update_comp_param::type (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_upper_update_comp_param::update (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_upper_update_comp_param::value (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_upper_update_param (C++ 类), 385
esp_ble_mesh_prov_cb_param_t::provisioner_upper_update_param::type (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_upper_update_param::update (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_upper_update_param::value (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_upper_update_recv_param (C++ 类), 385
esp_ble_mesh_prov_cb_param_t::provisioner_upper_update_recv_param::ctx (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_upper_update_recv_param::type (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_upper_update_recv_param::update (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_upper_update_recv_param::value (C++ 成员), 385
esp_ble_mesh_prov_cb_param_t::provisioner_upper_update_recv_param::value_len

(C++ 类), 387
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_prov_disable_comp_param::err_code (C++ 成员), 387
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_prov_disable_comp_param::node_num (C++ 成员), 387
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_prov_disable_comp_param::nodes (C++ 成员), 387
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_recv_unprov_adv_pkt_param (C++ 类), 387
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_recv_unprov_adv_pkt_param::addr (C++ 成员), 387
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_recv_unprov_adv_pkt_param::adv_type (C++ 成员), 387
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_recv_unprov_adv_pkt_param::bearer (C++ 成员), 387
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_recv_unprov_adv_pkt_param::dev_uuid (C++ 成员), 387
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_recv_unprov_adv_pkt_param::oob_info (C++ 成员), 387
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_recv_unprov_adv_pkt_param::oob_type (C++ 成员), 387
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_recv_unprov_adv_pkt_param::uri_hash (C++ 成员), 387
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_set_dev_uuid_match_comp_param (C++ 类), 387
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_set_dev_uuid_match_comp_param::addr (C++ 成员), 388
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_set_dev_uuid_match_comp_param::addr_type (C++ 成员), 388
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_set_node_name_comp_param (C++ 类), 388
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_set_node_name_comp_param::err_code (C++ 成员), 388
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_set_node_name_comp_param::index (C++ 成员), 388
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_set_node_name_comp_param::name (C++ 成员), 388
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_set_prov_data_info_comp_param (C++ 类), 388
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_set_prov_data_info_comp_param::err_code (C++ 成员), 388
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_set_prov_data_info_comp_param::flag (C++ 成员), 388
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_set_static_oob_val_comp_param (C++ 类), 388
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisioner_set_static_oob_val_comp_param::err_code (C++ 成员), 388
esp_ble_mesh_prov_cb_param_t::ble_mesh_set_oob_pub_key_comp_param (C++ 类), 388
esp_ble_mesh_prov_cb_param_t::ble_mesh_set_oob_pub_key_comp_param::err_code (C++ 成员), 388
esp_ble_mesh_prov_cb_param_t::ble_mesh_set_unprov_dev_name_comp_param (C++ 类), 388
esp_ble_mesh_prov_cb_param_t::ble_mesh_set_unprov_dev_name_comp_param::err_code (C++ 成员), 388
esp_ble_mesh_prov_cb_param_t::ble_mesh_unprov_beacon_recv_param (C++ 类), 388
esp_ble_mesh_prov_cb_param_t::ble_mesh_unprov_beacon_recv_param::oob_info (C++ 成员), 388
esp_ble_mesh_prov_cb_param_t::ble_mesh_unprov_beacon_recv_param::uri_hash (C++ 成员), 388
esp_ble_mesh_prov_cb_param_t::ble_mesh_unprov_dev_add_comp_param (C++ 类), 388
esp_ble_mesh_prov_cb_param_t::ble_mesh_unprov_dev_add_comp_param::addr (C++ 成员), 388
esp_ble_mesh_prov_cb_param_t::ble_mesh_unprov_dev_add_comp_param::addr_type (C++ 成员), 389
esp_ble_mesh_prov_cb_param_t::ble_mesh_unprov_dev_add_comp_param::err_code (C++ 成员), 389
esp_ble_mesh_prov_cb_param_t::ble_mesh_unprov_dev_add_comp_param::flag (C++ 成员), 389
esp_ble_mesh_prov_cb_param_t::ble_mesh_unprov_dev_add_comp_param::method (C++ 成员), 389
esp_ble_mesh_prov_cb_param_t::ble_mesh_unprov_dev_add_comp_param::num (C++ 成员), 389
esp_ble_mesh_prov_cb_param_t::ble_mesh_unprov_dev_add_comp_param::oob_type (C++ 成员), 389
esp_ble_mesh_prov_cb_param_t::ble_mesh_unprov_dev_add_comp_param::size (C++ 成员), 389
esp_ble_mesh_prov_cb_param_t::ble_mesh_unprov_dev_add_comp_param::uuid (C++ 成员), 389
esp_ble_mesh_prov_cb_param_t::ble_mesh_unprov_dev_add_comp_param::[anonymous] (C++ 成员), 389
esp_ble_mesh_prov_cb_param_t::ble_mesh_unprov_dev_add_comp_param::[anonymous] (C++ 成员), 390
esp_ble_mesh_prov_cb_param_t::ble_mesh_unprov_dev_add_comp_param::[anonymous] (C++ 类), 390
esp_ble_mesh_prov_cb_param_t::ble_mesh_unprov_dev_add_comp_param::[anonymous] (C++ 成员

(C++ 类), 390
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisio (C++ 成员), 390
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisio (C++ 类), 390
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisio (C++ 成员), 390
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisio (C++ 成员), 390
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisio (C++ 类), 390
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisio (C++ 成员), 390
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisio (C++ 成员), 390
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisio (C++ 类), 390
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisio (C++ 类), 390
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisio (C++ 成员), 391
esp_ble_mesh_prov_cb_param_t::ble_mesh_provisio (C++ 成员), 391
esp_ble_mesh_prov_cb_param_t::ble_mesh_proxy_cl (C++ 类), 391
esp_ble_mesh_prov_cb_param_t::ble_mesh_proxy_cl (C++ 成员), 391
esp_ble_mesh_prov_cb_param_t::ble_mesh_proxy_cl (C++ 成员), 391
esp_ble_mesh_prov_cb_param_t::ble_mesh_proxy_cl (C++ 成员), 391
esp_ble_mesh_prov_cb_param_t::ble_mesh_proxy_cl (C++ 类), 391
esp_ble_mesh_prov_cb_param_t::ble_mesh_proxy_cl (C++ 成员

(C++ 成员), 393
esp_ble_mesh_prov_cb_param_t::ble_mesh_proxy_client_recv_filter_status_param::filter_type (C++ 成员), 393
esp_ble_mesh_prov_cb_param_t::ble_mesh_proxy_client_recv_filter_status_param::list_size (C++ 成员), 393
esp_ble_mesh_prov_cb_param_t::ble_mesh_proxy_client_recv_filter_status_param::net_idx (C++ 成员), 393
esp_ble_mesh_prov_cb_param_t::ble_mesh_proxy_client_recv_filter_status_param::server_addr (C++ 成员), 393
esp_ble_mesh_prov_cb_param_t::ble_mesh_proxy_client_remove_filter_comp_param (C++ 类), 393
esp_ble_mesh_prov_cb_param_t::ble_mesh_proxy_client_remove_filter_comp_param::err_code (C++ 成员), 393
esp_ble_mesh_prov_cb_param_t::ble_mesh_proxy_client_remove_filter_comp_param::net_idx (C++ 成员), 393
esp_ble_mesh_prov_cb_param_t::ble_mesh_proxy_client_remove_filter_comp_param::server_addr (C++ 成员), 393
esp_ble_mesh_prov_cb_param_t::ble_mesh_set_fast_prov_info_comp_param (C++ 类), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_set_fast_prov_info_comp_param::err_code (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_set_fast_prov_info_comp_param::status_unicast (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_set_fast_prov_info_comp_param::status_net_idx (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_set_fast_prov_info_comp_param::status_app_idx (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_set_fast_prov_info_comp_param::status_matrix (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_set_fast_prov_info_comp_param::status_flags (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_set_fast_prov_info_comp_param::status_iv_index (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_set_oob_pub_key_comp_param (C++ 类), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_set_oob_pub_key_comp_param::err_code (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_start_ble_advertising_comp_param (C++ 类), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_start_ble_advertising_comp_param::err_code (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_stop_ble_advertising_comp_param (C++ 类), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_stop_ble_advertising_comp_param::err_code (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_store_node_comp_data_comp_param (C++ 类), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_store_node_comp_data_comp_param::err_code (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_store_node_comp_data_comp_param::status_comp_data (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_store_node_comp_data_comp_param::status_len (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_store_node_comp_data_comp_param::status_offset (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_store_node_comp_data_comp_param::type (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_store_node_comp_data_comp_param::status_page (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_store_node_comp_data_comp_param::status_features (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_store_node_comp_data_comp_param::status_net_idx (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_store_node_comp_data_comp_param::status_relay (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_store_node_comp_data_comp_param::status_gatt_proxy (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_store_node_comp_data_comp_param::status_friend (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_store_node_comp_data_comp_param::status_low_power (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_store_node_comp_data_comp_param::status_flags (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_store_node_comp_data_comp_param::status_iv_index (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_store_node_comp_data_comp_param::status_element_num (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_store_node_comp_data_comp_param::status_model_num (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_store_node_comp_data_comp_param::status_model_id (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_store_node_comp_data_comp_param::status_vnd_model_id (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_store_node_comp_data_comp_param::status_sig_model_id (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_store_node_comp_data_comp_param::status_company_id (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_store_node_comp_data_comp_param::status_element_addr (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_store_node_comp_data_comp_param::status_loc (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_store_node_comp_data_comp_param::status_nums (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_store_node_comp_data_comp_param::status_sig (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_store_node_comp_data_comp_param::status_vnd (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_store_node_comp_data_comp_param::status_sig_models (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_store_node_comp_data_comp_param::status_vnd_models (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_store_node_comp_data_comp_param::status_model_idx (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_store_node_comp_data_comp_param::status_model_sub_num (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_store_node_comp_data_comp_param::status_model_sub_addr (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_store_node_comp_data_comp_param::status_model_bind_num (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_store_node_comp_data_comp_param::status_model_bind_app_idx (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_store_node_comp_data_comp_param::status_model_pub_addr (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_store_node_comp_data_comp_param::status_model_pub_app_idx (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_store_node_comp_data_comp_param::status_model_pub_period (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_store_node_comp_data_comp_param::status_model_pub_ttl (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_store_node_comp_data_comp_param::status_model_pub_retransmit (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_store_node_comp_data_comp_param::status_model_pub_net_idx (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_store_node_comp_data_comp_param::status_model_pub_fast_period (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_store_node_comp_data_comp_param::status_model_pub_group_addr (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_store_node_comp_data_comp_param::status_model_pub_fast_ttl (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_store_node_comp_data_comp_param::status_model_pub_fast_retransmit (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_store_node_comp_data_comp_param::status_model_pub_fast_net_idx (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_store_node_comp_data_comp_param::status_model_pub_fast_group_addr (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_store_node_comp_data_comp_param::status_model_pub_fast_app_idx (C++ 成员), 394
esp_ble_mesh_prov_cb_param_t::ble_mesh_store_node_comp_data_comp_param::status_model_pub_fast_period

(C++ 成员), 378
esp_ble_mesh_prov_cb_param_t::heartbeat_msg_recv
(C++ 成员), 378
esp_ble_mesh_prov_cb_param_t::lpn_disable_comp
(C++ 成员), 378
esp_ble_mesh_prov_cb_param_t::lpn_enable_comp
(C++ 成员), 378
esp_ble_mesh_prov_cb_param_t::lpn_friendship_establish
(C++ 成员), 378
esp_ble_mesh_prov_cb_param_t::lpn_friendship_terminate
(C++ 成员), 378
esp_ble_mesh_prov_cb_param_t::lpn_poll_comp
(C++ 成员), 378
esp_ble_mesh_prov_cb_param_t::node_prov_complete
(C++ 成员), 376
esp_ble_mesh_prov_cb_param_t::node_prov_disable_comp
(C++ 成员), 375
esp_ble_mesh_prov_cb_param_t::node_prov_enable_comp
(C++ 成员), 375
esp_ble_mesh_prov_cb_param_t::node_prov_input
(C++ 成员), 376
esp_ble_mesh_prov_cb_param_t::node_prov_input_num
(C++ 成员), 376
esp_ble_mesh_prov_cb_param_t::node_prov_input_str
(C++ 成员), 376
esp_ble_mesh_prov_cb_param_t::node_prov_link_close
(C++ 成员), 375
esp_ble_mesh_prov_cb_param_t::node_prov_link_open
(C++ 成员), 375
esp_ble_mesh_prov_cb_param_t::node_prov_output_num
(C++ 成员), 375
esp_ble_mesh_prov_cb_param_t::node_prov_output_str
(C++ 成员), 376
esp_ble_mesh_prov_cb_param_t::node_prov_reset
(C++ 成员), 376
esp_ble_mesh_prov_cb_param_t::node_prov_set_oob_pub_key_comp
(C++ 成员), 376
esp_ble_mesh_prov_cb_param_t::node_proxy_gatt_disable_comp
(C++ 成员), 376
esp_ble_mesh_prov_cb_param_t::node_proxy_gatt_enable_comp
(C++ 成员), 376
esp_ble_mesh_prov_cb_param_t::node_proxy_identity_enable_comp
(C++ 成员), 376
esp_ble_mesh_prov_cb_param_t::node_set_unprov_dev_name_com
(C++ 成员), 375
esp_ble_mesh_prov_cb_param_t::prov_register_comp
(C++ 成员), 375
esp_ble_mesh_prov_cb_param_t::provisioner_add_app_key_comp
(C++ 成员), 377
esp_ble_mesh_prov_cb_param_t::provisioner_add_net_key_comp
(C++ 成员), 377
esp_ble_mesh_prov_cb_param_t::provisioner_add_unprov_dev_c
(C++ 成员), 377
esp_ble_mesh_prov_cb_param_t::provisioner_bind_app_key_to_
(C++ 成员), 377
esp_ble_mesh_prov_cb_param_t::provisioner_delete_dev_comp
(C++ 成员), 377
esp_ble_mesh_prov_cb_param_t::provisioner_delete_node_with
(C++ 成员), 378
esp_ble_mesh_prov_cb_param_t::provisioner_delete_node_with
(C++ 成员), 378
esp_ble_mesh_prov_cb_param_t::provisioner_prov_complete
(C++ 成员), 377
esp_ble_mesh_prov_cb_param_t::provisioner_prov_dev_with_ad
(C++ 成员), 377
esp_ble_mesh_prov_cb_param_t::provisioner_prov_disable_com
(C++ 成员), 376
esp_ble_mesh_prov_cb_param_t::provisioner_prov_enable_comp
(C++ 成员), 376
esp_ble_mesh_prov_cb_param_t::provisioner_prov_input
(C++ 成员), 376
esp_ble_mesh_prov_cb_param_t::provisioner_prov_input_num_c
(C++ 成员), 377
esp_ble_mesh_prov_cb_param_t::provisioner_prov_input_str_c
(C++ 成员), 377
esp_ble_mesh_prov_cb_param_t::provisioner_prov_link_close
(C++ 成员), 377
esp_ble_mesh_prov_cb_param_t::provisioner_prov_link_open
(C++ 成员), 376
esp_ble_mesh_prov_cb_param_t::provisioner_prov_output
(C++ 成员), 376
esp_ble_mesh_prov_cb_param_t::provisioner_prov_read_oob_pu
(C++ 成员), 376
esp_ble_mesh_prov_cb_param_t::provisioner_prov_read_oob_pu

(C++ 成员), 377
esp_ble_mesh_prov_cb_param_t::provisioner_recv_unprov_adv_pkt_comp (C++ 成员), 376
esp_ble_mesh_prov_cb_param_t::provisioner_set_dev_uuid_match_comp (C++ 成员), 377
esp_ble_mesh_prov_cb_param_t::provisioner_set_node_name_comp (C++ 成员), 377
esp_ble_mesh_prov_cb_param_t::provisioner_set_primary_elem_addr_comp (C++ 成员), 377
esp_ble_mesh_prov_cb_param_t::provisioner_set_prov_data_info_comp (C++ 成员), 377
esp_ble_mesh_prov_cb_param_t::provisioner_set_static_oob_val_comp (C++ 成员), 377
esp_ble_mesh_prov_cb_param_t::provisioner_store_node_comp_data_comp (C++ 成员), 378
esp_ble_mesh_prov_cb_param_t::provisioner_update_app_key_comp (C++ 成员), 377
esp_ble_mesh_prov_cb_param_t::provisioner_update_net_key_comp (C++ 成员), 378
esp_ble_mesh_prov_cb_param_t::proxy_client_add_filter_comp (C++ 成员), 379
esp_ble_mesh_prov_cb_param_t::proxy_client_connect_comp (C++ 成员), 379
esp_ble_mesh_prov_cb_param_t::proxy_client_connected (C++ 成员), 378
esp_ble_mesh_prov_cb_param_t::proxy_client_disconnect_comp (C++ 成员), 379
esp_ble_mesh_prov_cb_param_t::proxy_client_disconnected (C++ 成员), 378
esp_ble_mesh_prov_cb_param_t::proxy_client_recv_adv_pkt (C++ 成员), 378
esp_ble_mesh_prov_cb_param_t::proxy_client_recv_filter_status (C++ 成员), 379
esp_ble_mesh_prov_cb_param_t::proxy_client_remove_filter_comp (C++ 成员), 379
esp_ble_mesh_prov_cb_param_t::proxy_client_set_filter_type_comp (C++ 成员), 379
esp_ble_mesh_prov_cb_param_t::set_fast_prov_action_comp (C++ 成员), 378
esp_ble_mesh_prov_cb_param_t::set_fast_prov_info_comp (C++ 成员), 378
esp_ble_mesh_prov_cb_param_t::start_ble_advertising_comp (C++ 成员), 379
esp_ble_mesh_prov_cb_param_t::stop_ble_advertising_comp (C++ 成员), 379
esp_ble_mesh_prov_cb_t (C++ 类型), 458
esp_ble_mesh_prov_data_info_t (C++ 类), 405
esp_ble_mesh_prov_data_info_t::flag (C++ 成员), 405
esp_ble_mesh_prov_data_info_t::flags (C++ 成员), 405
esp_ble_mesh_prov_data_info_t::iv_index (C++ 成员), 405
esp_ble_mesh_prov_data_info_t::net_idx (C++ 成员), 405
ESP_BLE_MESH_PROV_EVT_MAX (C++ 枚举子), 438
ESP_BLE_MESH_PROV_GATT (C++ 枚举子), 433
ESP_BLE_MESH_PROV_INPUT_OOB_MAX_LEN (C 宏), 414
ESP_BLE_MESH_PROV_OOB_2D_CODE (C++ 枚举子), 433
ESP_BLE_MESH_PROV_OOB_BAR_CODE (C++ 枚举子), 433
ESP_BLE_MESH_PROV_OOB_IN_BOX (C++ 枚举子), 433
ESP_BLE_MESH_PROV_OOB_IN_MANUAL (C++ 枚举子), 434
esp_ble_mesh_prov_oob_info_t (C++ 类型), 433
ESP_BLE_MESH_PROV_OOB_NFC (C++ 枚举子), 433
ESP_BLE_MESH_PROV_OOB_NUMBER (C++ 枚举子), 433
ESP_BLE_MESH_PROV_OOB_ON_BOX (C++ 枚举子), 433
ESP_BLE_MESH_PROV_OOB_ON_DEV (C++ 枚举子), 434
ESP_BLE_MESH_PROV_OOB_ON_PAPER (C++ 枚举子), 433
ESP_BLE_MESH_PROV_OOB_OTHER (C++ 枚举子), 433
ESP_BLE_MESH_PROV_OOB_STRING (C++ 枚举子), 433
ESP_BLE_MESH_PROV_OOB_URI (C++ 枚举子), 433
ESP_BLE_MESH_PROV_OUTPUT_OOB_MAX_LEN (C 宏), 414
ESP_BLE_MESH_PROV_REGISTER_COMP_EVT (C++ 枚

举子), 434
ESP_BLE_MESH_PROV_STATIC_OOB_MAX_LEN (C 宏), 414
esp_ble_mesh_prov_t (C++ 类), 404
esp_ble_mesh_provisioner_add_local_app_key (C++ 函数), 448
ESP_BLE_MESH_PROVISIONER_ADD_LOCAL_APP_KEY_COMP_EVT (C++ 枚举子), 437
esp_ble_mesh_provisioner_add_local_net_key (C++ 函数), 450
ESP_BLE_MESH_PROVISIONER_ADD_LOCAL_NET_KEY_COMP_EVT (C++ 枚举子), 437
esp_ble_mesh_provisioner_add_unprov_dev (C++ 函数), 454
ESP_BLE_MESH_PROVISIONER_ADD_UNPROV_DEV_COMP_EVT (C++ 枚举子), 436
esp_ble_mesh_provisioner_bind_app_key_to_local_model (C++ 函数), 449
ESP_BLE_MESH_PROVISIONER_BIND_APP_KEY_TO_MODEL_COMP_EVT (C++ 枚举子), 437
esp_ble_mesh_provisioner_delete_dev (C++ 函数), 456
ESP_BLE_MESH_PROVISIONER_DELETE_DEV_COMP_EVT (C++ 枚举子), 436
esp_ble_mesh_provisioner_delete_node_with_addr (C++ 函数), 448
ESP_BLE_MESH_PROVISIONER_DELETE_NODE_WITH_ADDR_COMP_EVT (C++ 枚举子), 437
esp_ble_mesh_provisioner_delete_node_with_uuid (C++ 函数), 448
ESP_BLE_MESH_PROVISIONER_DELETE_NODE_WITH_UUID_COMP_EVT (C++ 枚举子), 437
esp_ble_mesh_provisioner_get_local_app_key (C++ 函数), 449
esp_ble_mesh_provisioner_get_local_net_key (C++ 函数), 450
esp_ble_mesh_provisioner_get_node_index (C++ 函数), 447
esp_ble_mesh_provisioner_get_node_name (C++ 函数), 447
esp_ble_mesh_provisioner_get_node_with_addr (C++ 函数), 448
esp_ble_mesh_provisioner_get_node_with_uuid (C++ 函数), 448
esp_ble_mesh_provisioner_get_prov_node_count (C++ 函数), 450
esp_ble_mesh_provisioner_input_number (C++ 函数), 453
esp_ble_mesh_provisioner_input_string (C++ 函数), 453
ESP_BLE_MESH_PROVISIONER_PROV_COMPLETE_EVT (C++ 枚举子), 436
ESP_BLE_MESH_PROVISIONER_PROV_DEV_WITH_ADDR_COMP_EVT (C++ 枚举子), 436
esp_ble_mesh_provisioner_prov_device_with_addr (C++ 函数), 455
esp_ble_mesh_provisioner_prov_disable (C++ 函数), 454
ESP_BLE_MESH_PROVISIONER_PROV_DISABLE_COMP_EVT (C++ 枚举子), 435
esp_ble_mesh_provisioner_prov_enable (C++ 函数), 454
ESP_BLE_MESH_PROVISIONER_PROV_ENABLE_COMP_EVT (C++ 枚举子), 435
ESP_BLE_MESH_PROVISIONER_PROV_INPUT_EVT (C++ 枚举子), 436
ESP_BLE_MESH_PROVISIONER_PROV_INPUT_NUMBER_COMP_EVT (C++ 枚举子), 436
ESP_BLE_MESH_PROVISIONER_PROV_INPUT_STRING_COMP_EVT (C++ 枚举子), 436
ESP_BLE_MESH_PROVISIONER_PROV_LINK_CLOSE_EVT (C++ 枚举子), 436
ESP_BLE_MESH_PROVISIONER_PROV_LINK_OPEN_EVT (C++ 枚举子), 436
ESP_BLE_MESH_PROVISIONER_PROV_OUTPUT_EVT (C++ 枚举子), 436
ESP_BLE_MESH_PROVISIONER_PROV_READ_OOB_PUB_KEY_COMP_EVT (C++ 枚举子), 436
ESP_BLE_MESH_PROVISIONER_PROV_READ_OOB_PUB_KEY_EVT (C++ 枚举子), 435
esp_ble_mesh_provisioner_read_oob_pub_key (C++ 函数), 453
ESP_BLE_MESH_PROVISIONER_RECV_UNPROV_ADV_PKT_EVT (C++ 枚举子), 435

esp_ble_mesh_provisioner_set_dev_uuid_match (C++ 函数), 456
ESP_BLE_MESH_PROVISIONER_SET_DEV_UUID_MATCH_COMP_EVT (C++ 枚举子), 436
esp_ble_mesh_provisioner_set_node_name (C++ 函数), 447
ESP_BLE_MESH_PROVISIONER_SET_NODE_NAME_COMP_EVT (C++ 枚举子), 436
esp_ble_mesh_provisioner_set_primary_elem_addr (C++ 函数), 457
ESP_BLE_MESH_PROVISIONER_SET_PRIMARY_ELEM_ADDR_COMP_EVT (C++ 枚举子), 436
esp_ble_mesh_provisioner_set_prov_data_info (C++ 函数), 457
ESP_BLE_MESH_PROVISIONER_SET_PROV_DATA_INFO_COMP_EVT (C++ 枚举子), 436
esp_ble_mesh_provisioner_set_static_oob_value (C++ 函数), 457
ESP_BLE_MESH_PROVISIONER_SET_STATIC_OOB_VALUE_COMP_EVT (C++ 枚举子), 436
esp_ble_mesh_provisioner_store_node_comp_data (C++ 函数), 447
ESP_BLE_MESH_PROVISIONER_STORE_NODE_COMP_DATA_COMP_EVT (C++ 枚举子), 437
esp_ble_mesh_provisioner_update_local_app_key (C++ 函数), 449
ESP_BLE_MESH_PROVISIONER_UPDATE_LOCAL_APP_KEY_COMP_EVT (C++ 枚举子), 437
esp_ble_mesh_provisioner_update_local_net_key (C++ 函数), 450
ESP_BLE_MESH_PROVISIONER_UPDATE_LOCAL_NET_KEY_COMP_EVT (C++ 枚举子), 437
esp_ble_mesh_proxy_client_add_filter_addr (C++ 函数), 460
ESP_BLE_MESH_PROXY_CLIENT_ADD_FILTER_ADDR_COMP_EVT (C++ 枚举子), 438
esp_ble_mesh_proxy_client_connect (C++ 函数), 459
ESP_BLE_MESH_PROXY_CLIENT_CONNECT_COMP_EVT (C++ 枚举子), 438
ESP_BLE_MESH_PROXY_CLIENT_CONNECTED_EVT (C++ 枚举子), 438
esp_ble_mesh_proxy_client_disconnect (C++ 函数), 459
ESP_BLE_MESH_PROXY_CLIENT_DISCONNECT_COMP_EVT (C++ 枚举子), 438
ESP_BLE_MESH_PROXY_CLIENT_DISCONNECTED_EVT (C++ 枚举子), 438
ESP_BLE_MESH_PROXY_CLIENT_RECV_ADV_PKT_EVT (C++ 枚举子), 438
ESP_BLE_MESH_PROXY_CLIENT_RECV_FILTER_STATUS_EVT (C++ 枚举子), 438
esp_ble_mesh_proxy_client_remove_filter_addr (C++ 函数), 460
ESP_BLE_MESH_PROXY_CLIENT_REMOVE_FILTER_ADDR_COMP_EVT (C++ 枚举子), 438
esp_ble_mesh_proxy_client_set_filter_type (C++ 函数), 459
ESP_BLE_MESH_PROXY_CLIENT_SET_FILTER_TYPE_COMP_EVT (C++ 枚举子), 438
esp_ble_mesh_proxy_filter_type_t (C++ 类型), 434
esp_ble_mesh_proxy_gatt_disable (C++ 函数), 459
esp_ble_mesh_proxy_gatt_enable (C++ 函数), 459
esp_ble_mesh_proxy_identity_enable (C++ 函数), 459
ESP_BLE_MESH_PUBLISH_TRANSMIT (C 宏), 413
ESP_BLE_MESH_PUSH (C++ 枚举子), 433
esp_ble_mesh_register_config_client_callback (C++ 函数), 461
esp_ble_mesh_register_config_server_callback (C++ 函数), 461
esp_ble_mesh_register_custom_model_callback (C++ 函数), 444
esp_ble_mesh_register_generic_client_callback (C++ 函数), 492
esp_ble_mesh_register_generic_server_callback (C++ 函数), 493
esp_ble_mesh_register_health_client_callback (C++ 函数), 534
esp_ble_mesh_register_health_server_callback (C++ 函数), 534

esp_ble_mesh_register_light_client_callback (C++ 函数), 546
esp_ble_mesh_register_lighting_server_callback (C++ 函数), 547
esp_ble_mesh_register_prov_callback (C++ 函数), 451
esp_ble_mesh_register_sensor_client_callback (C++ 函数), 603
esp_ble_mesh_register_sensor_server_callback (C++ 函数), 603
esp_ble_mesh_register_time_scene_client_callback (C++ 函数), 623
esp_ble_mesh_register_time_scene_server_callback (C++ 函数), 624
ESP_BLE_MESH_RELAY_DISABLED (C 宏), 412
ESP_BLE_MESH_RELAY_ENABLED (C 宏), 412
ESP_BLE_MESH_RELAY_NOT_SUPPORTED (C 宏), 412
ESP_BLE_MESH_SAMPLE_FUNC_ACCUMULATED (C++ 枚举子), 623
ESP_BLE_MESH_SAMPLE_FUNC_ARITHMETIC_MEAN (C++ 枚举子), 623
ESP_BLE_MESH_SAMPLE_FUNC_COUNT (C++ 枚举子), 623
ESP_BLE_MESH_SAMPLE_FUNC_INSTANTANEOUS (C++ 枚举子), 622
ESP_BLE_MESH_SAMPLE_FUNC_MAXIMUM (C++ 枚举子), 623
ESP_BLE_MESH_SAMPLE_FUNC_MINIMUM (C++ 枚举子), 623
ESP_BLE_MESH_SAMPLE_FUNC_RMS (C++ 枚举子), 623
ESP_BLE_MESH_SAMPLE_FUNC_UNSPECIFIED (C++ 枚举子), 622
esp_ble_mesh_scene_delete_t (C++ 类), 630
esp_ble_mesh_scene_delete_t::scene_number (C++ 成员), 630
ESP_BLE_MESH_SCENE_NOT_FOUND (C 宏), 651
ESP_BLE_MESH_SCENE_NUMBER_LEN (C 宏), 650
esp_ble_mesh_scene_recall_t (C++ 类), 629
esp_ble_mesh_scene_recall_t::delay (C++ 成员), 630
esp_ble_mesh_scene_recall_t::op_en (C++ 成员), 629
esp_ble_mesh_scene_recall_t::scene_number (C++ 成员), 629
esp_ble_mesh_scene_recall_t::tid (C++ 成员), 630
esp_ble_mesh_scene_recall_t::trans_time (C++ 成员), 630
ESP_BLE_MESH_SCENE_REG_FULL (C 宏), 651
esp_ble_mesh_scene_register_status_cb_t (C++ 类), 633
esp_ble_mesh_scene_register_status_cb_t::current_scene (C++ 成员), 633
esp_ble_mesh_scene_register_status_cb_t::scenes (C++ 成员), 633
esp_ble_mesh_scene_register_status_cb_t::status_code (C++ 成员), 633
esp_ble_mesh_scene_register_t (C++ 类), 636
esp_ble_mesh_scene_register_t::scene_number (C++ 成员), 636
esp_ble_mesh_scene_register_t::scene_type (C++ 成员), 636
esp_ble_mesh_scene_register_t::scene_value (C++ 成员), 636
esp_ble_mesh_scene_setup_srv_t (C++ 类), 638
esp_ble_mesh_scene_setup_srv_t::model (C++ 成员), 638
esp_ble_mesh_scene_setup_srv_t::rsp_ctrl (C++ 成员), 638
esp_ble_mesh_scene_setup_srv_t::state (C++ 成员), 638
esp_ble_mesh_scene_srv_t (C++ 类), 637
esp_ble_mesh_scene_srv_t::last (C++ 成员), 638
esp_ble_mesh_scene_srv_t::model (C++ 成员), 637
esp_ble_mesh_scene_srv_t::rsp_ctrl (C++ 成员), 637
esp_ble_mesh_scene_srv_t::state (C++ 成员), 637
esp_ble_mesh_scene_srv_t::transition (C++ 成员), 638
esp_ble_mesh_scene_status_cb_t (C++ 类), 632

esp_ble_mesh_scene_status_cb_t::current_scene (C++ 成员), 632
esp_ble_mesh_scene_status_cb_t::op_en (C++ 成员), 632
esp_ble_mesh_scene_status_cb_t::remain_time (C++ 成员), 633
esp_ble_mesh_scene_status_cb_t::status_code (C++ 成员), 632
esp_ble_mesh_scene_status_cb_t::target_scene (C++ 成员), 632
esp_ble_mesh_scene_store_t (C++ 类), 629
esp_ble_mesh_scene_store_t::scene_number (C++ 成员), 629
ESP_BLE_MESH_SCENE_SUCCESS (C 宏), 651
esp_ble_mesh_scenes_state_t (C++ 类), 636
esp_ble_mesh_scenes_state_t::current_scene (C++ 成员), 637
esp_ble_mesh_scenes_state_t::in_progress (C++ 成员), 637
esp_ble_mesh_scenes_state_t::scene_count (C++ 成员), 636
esp_ble_mesh_scenes_state_t::scenes (C++ 成员), 636
esp_ble_mesh_scenes_state_t::status_code (C++ 成员), 637
esp_ble_mesh_scenes_state_t::target_scene (C++ 成员), 637
ESP_BLE_MESH_SCHEDULE_ACT_NO_ACTION (C 宏), 650
ESP_BLE_MESH_SCHEDULE_ACT_SCENE_RECALL (C 宏), 650
ESP_BLE_MESH_SCHEDULE_ACT_TURN_OFF (C 宏), 650
ESP_BLE_MESH_SCHEDULE_ACT_TURN_ON (C 宏), 650
ESP_BLE_MESH_SCHEDULE_DAY_ANY_DAY (C 宏), 650
ESP_BLE_MESH_SCHEDULE_ENTRY_MAX_INDEX (C 宏), 651
ESP_BLE_MESH_SCHEDULE_HOUR_ANY_HOUR (C 宏), 650
ESP_BLE_MESH_SCHEDULE_HOUR_ONCE_A_DAY (C 宏), 650
esp_ble_mesh_schedule_register_t (C++ 类), 638
esp_ble_mesh_schedule_register_t::action (C++ 成员), 639
esp_ble_mesh_schedule_register_t::day (C++ 成员), 638
esp_ble_mesh_schedule_register_t::day_of_week (C++ 成员), 638
esp_ble_mesh_schedule_register_t::hour (C++ 成员), 638
esp_ble_mesh_schedule_register_t::in_use (C++ 成员), 638
esp_ble_mesh_schedule_register_t::minute (C++ 成员), 638
esp_ble_mesh_schedule_register_t::month (C++ 成员), 638
esp_ble_mesh_schedule_register_t::scene_number (C++ 成员), 639
esp_ble_mesh_schedule_register_t::second (C++ 成员), 638
esp_ble_mesh_schedule_register_t::trans_time (C++ 成员), 639
esp_ble_mesh_schedule_register_t::year (C++ 成员), 638
ESP_BLE_MESH_SCHEDULE_SCENE_NO_SCENE (C 宏), 651
ESP_BLE_MESH_SCHEDULE_SEC_ANY_OF_HOUR (C 宏), 650
ESP_BLE_MESH_SCHEDULE_SEC_ANY_OF_MIN (C 宏), 650
ESP_BLE_MESH_SCHEDULE_SEC_EVERY_15_MIN (C 宏), 650
ESP_BLE_MESH_SCHEDULE_SEC_EVERY_15_SEC (C 宏), 650
ESP_BLE_MESH_SCHEDULE_SEC_EVERY_20_MIN (C 宏), 650
ESP_BLE_MESH_SCHEDULE_SEC_EVERY_20_SEC (C 宏), 650
ESP_BLE_MESH_SCHEDULE_SEC_ONCE_AN_HOUR (C 宏), 650
ESP_BLE_MESH_SCHEDULE_SEC_ONCE_AN_MIN (C 宏), 650
ESP_BLE_MESH_SCHEDULE_YEAR_ANY_YEAR (C 宏), 650

esp_ble_mesh_scheduler_act_get_t (C++ 类), 630
esp_ble_mesh_scheduler_act_get_t::index (C++ 成员), 630
esp_ble_mesh_scheduler_act_set_t (C++ 类), 630
esp_ble_mesh_scheduler_act_set_t::action (C++ 成员), 631
esp_ble_mesh_scheduler_act_set_t::day (C++ 成员), 630
esp_ble_mesh_scheduler_act_set_t::day_of_week (C++ 成员), 631
esp_ble_mesh_scheduler_act_set_t::hour (C++ 成员), 630
esp_ble_mesh_scheduler_act_set_t::index (C++ 成员), 630
esp_ble_mesh_scheduler_act_set_t::minute (C++ 成员), 630
esp_ble_mesh_scheduler_act_set_t::month (C++ 成员), 630
esp_ble_mesh_scheduler_act_set_t::scene_number (C++ 成员), 631
esp_ble_mesh_scheduler_act_set_t::second (C++ 成员), 631
esp_ble_mesh_scheduler_act_set_t::trans_time (C++ 成员), 631
esp_ble_mesh_scheduler_act_set_t::year (C++ 成员), 630
esp_ble_mesh_scheduler_act_status_cb_t (C++ 类), 633
esp_ble_mesh_scheduler_act_status_cb_t::action (C++ 成员), 634
esp_ble_mesh_scheduler_act_status_cb_t::day (C++ 成员), 633
esp_ble_mesh_scheduler_act_status_cb_t::day_of_week (C++ 成员), 634
esp_ble_mesh_scheduler_act_status_cb_t::hour (C++ 成员), 633
esp_ble_mesh_scheduler_act_status_cb_t::index (C++ 成员), 633
esp_ble_mesh_scheduler_act_status_cb_t::minute (C++ 成员), 633
esp_ble_mesh_scheduler_act_status_cb_t::month (C++ 成员), 633
esp_ble_mesh_scheduler_act_status_cb_t::scene_number (C++ 成员), 634
esp_ble_mesh_scheduler_act_status_cb_t::second (C++ 成员), 634
esp_ble_mesh_scheduler_act_status_cb_t::trans_time (C++ 成员), 634
esp_ble_mesh_scheduler_act_status_cb_t::year (C++ 成员), 633
esp_ble_mesh_scheduler_setup_srv_t (C++ 类), 639
esp_ble_mesh_scheduler_setup_srv_t::model (C++ 成员), 639
esp_ble_mesh_scheduler_setup_srv_t::rsp_ctrl (C++ 成员), 639
esp_ble_mesh_scheduler_setup_srv_t::state (C++ 成员), 639
esp_ble_mesh_scheduler_srv_t (C++ 类), 639
esp_ble_mesh_scheduler_srv_t::model (C++ 成员), 639
esp_ble_mesh_scheduler_srv_t::rsp_ctrl (C++ 成员), 639
esp_ble_mesh_scheduler_srv_t::state (C++ 成员), 639
esp_ble_mesh_scheduler_state_t (C++ 类), 639
esp_ble_mesh_scheduler_state_t::schedule_count (C++ 成员), 639
esp_ble_mesh_scheduler_state_t::schedules (C++ 成员), 639
esp_ble_mesh_scheduler_status_cb_t (C++ 类), 633
esp_ble_mesh_scheduler_status_cb_t::schedules (C++ 成员), 633
ESP_BLE_MESH_SDU_MAX_LEN (C 宏), 411
ESP_BLE_MESH_SELF_TEST_ERROR (C 宏), 544
ESP_BLE_MESH_SELF_TEST_WARNING (C 宏), 544
esp_ble_mesh_sensor_cadence_get_t (C++ 类), 607
esp_ble_mesh_sensor_cadence_get_t::property_id (C++ 成员), 607
esp_ble_mesh_sensor_cadence_set_t (C++ 类),

607
esp_ble_mesh_sensor_cadence_set_t::fast_cadence_high (C++ 成员), 607
esp_ble_mesh_sensor_cadence_set_t::fast_cadence_low (C++ 成员), 607
esp_ble_mesh_sensor_cadence_set_t::fast_cadence_period_divisor (C++ 成员), 607
esp_ble_mesh_sensor_cadence_set_t::property_id (C++ 成员), 607
esp_ble_mesh_sensor_cadence_set_t::status_min_interval (C++ 成员), 607
esp_ble_mesh_sensor_cadence_set_t::status_trigger_delta_down (C++ 成员), 607
esp_ble_mesh_sensor_cadence_set_t::status_trigger_delta_up (C++ 成员), 607
esp_ble_mesh_sensor_cadence_set_t::status_trigger_type (C++ 成员), 607
esp_ble_mesh_sensor_cadence_status_cb_t (C++ 类), 609
esp_ble_mesh_sensor_cadence_status_cb_t::property_id (C++ 成员), 609
esp_ble_mesh_sensor_cadence_status_cb_t::sensor_cadence_value (C++ 成员), 609
esp_ble_mesh_sensor_cadence_t (C++ 类), 612
esp_ble_mesh_sensor_cadence_t::fast_cadence_high (C++ 成员), 612
esp_ble_mesh_sensor_cadence_t::fast_cadence_low (C++ 成员), 612
esp_ble_mesh_sensor_cadence_t::min_interval (C++ 成员), 612
esp_ble_mesh_sensor_cadence_t::period_divisor (C++ 成员), 612
esp_ble_mesh_sensor_cadence_t::trigger_delta_down (C++ 成员), 612
esp_ble_mesh_sensor_cadence_t::trigger_delta_up (C++ 成员), 612
esp_ble_mesh_sensor_cadence_t::trigger_type (C++ 成员), 612
esp_ble_mesh_sensor_client_cb_event_t (C++ 类型), 622
esp_ble_mesh_sensor_client_cb_param_t (C++ 类), 611
esp_ble_mesh_sensor_client_cb_param_t::error_code (C++ 成员), 611
esp_ble_mesh_sensor_client_cb_param_t::params (C++ 成员), 611
esp_ble_mesh_sensor_client_cb_param_t::status_cb (C++ 成员), 611
esp_ble_mesh_sensor_client_cb_t (C++ 类型), 622
ESP_BLE_MESH_SENSOR_CLIENT_EVT_MAX (C++ 枚举子), 622
esp_ble_mesh_sensor_client_get_state (C++ 函数), 603
ESP_BLE_MESH_SENSOR_CLIENT_GET_STATE_EVT (C++ 枚举子), 622
esp_ble_mesh_sensor_client_get_state_t (C++ 类型), 604
esp_ble_mesh_sensor_client_get_state_t::cadence_get (C++ 成员), 604
esp_ble_mesh_sensor_client_get_state_t::column_get (C++ 成员), 604
esp_ble_mesh_sensor_client_get_state_t::descriptor_get (C++ 成员), 604
esp_ble_mesh_sensor_client_get_state_t::sensor_get (C++ 成员), 604
esp_ble_mesh_sensor_client_get_state_t::series_get (C++ 成员), 604
esp_ble_mesh_sensor_client_get_state_t::setting_get (C++ 成员), 604
esp_ble_mesh_sensor_client_get_state_t::settings_get (C++ 成员), 604
ESP_BLE_MESH_SENSOR_CLIENT_PUBLISH_EVT (C++ 枚举子), 622
esp_ble_mesh_sensor_client_set_state (C++ 函数), 603
ESP_BLE_MESH_SENSOR_CLIENT_SET_STATE_EVT (C++ 枚举子), 622
esp_ble_mesh_sensor_client_set_state_t (C++ 类型), 604
esp_ble_mesh_sensor_client_set_state_t::cadence_set (C++ 成员), 604
esp_ble_mesh_sensor_client_set_state_t::setting_set (C++ 成员), 604

esp_ble_mesh_sensor_client_status_cb_t (C++ 类型), 604
esp_ble_mesh_sensor_client_status_cb_t::cadence_status (C++ 成员), 605
esp_ble_mesh_sensor_client_status_cb_t::column_status (C++ 成员), 605
esp_ble_mesh_sensor_client_status_cb_t::descriptor_status (C++ 成员), 605
esp_ble_mesh_sensor_client_status_cb_t::sensor_status (C++ 成员), 605
esp_ble_mesh_sensor_client_status_cb_t::series_status (C++ 成员), 605
esp_ble_mesh_sensor_client_status_cb_t::setting_status (C++ 成员), 605
esp_ble_mesh_sensor_client_status_cb_t::settings_status (C++ 成员), 605
ESP_BLE_MESH_SENSOR_CLIENT_TIMEOUT_EVT (C++ 枚举子), 622
esp_ble_mesh_sensor_column_get_t (C++ 类), 608
esp_ble_mesh_sensor_column_get_t::property_id (C++ 成员), 609
esp_ble_mesh_sensor_column_get_t::raw_value_x (C++ 成员), 609
esp_ble_mesh_sensor_column_status_cb_t (C++ 类), 610
esp_ble_mesh_sensor_column_status_cb_t::property_id (C++ 成员), 610
esp_ble_mesh_sensor_column_status_cb_t::sensor_column_value (C++ 成员), 610
ESP_BLE_MESH_SENSOR_DATA_FORMAT_A (C 宏), 620
ESP_BLE_MESH_SENSOR_DATA_FORMAT_A_MPID (C 宏), 621
ESP_BLE_MESH_SENSOR_DATA_FORMAT_A_MPID_LEN (C 宏), 620
ESP_BLE_MESH_SENSOR_DATA_FORMAT_B (C 宏), 620
ESP_BLE_MESH_SENSOR_DATA_FORMAT_B_MPID (C 宏), 621
ESP_BLE_MESH_SENSOR_DATA_FORMAT_B_MPID_LEN (C 宏), 620
esp_ble_mesh_sensor_data_t (C++ 类), 612
esp_ble_mesh_sensor_data_t::format (C++ 成员), 612
esp_ble_mesh_sensor_data_t::length (C++ 成员), 613
esp_ble_mesh_sensor_data_t::raw_value (C++ 成员), 613
ESP_BLE_MESH_SENSOR_DATA_ZERO_LEN (C 宏), 620
esp_ble_mesh_sensor_descriptor_get_t (C++ 类), 606
esp_ble_mesh_sensor_descriptor_get_t::op_en (C++ 成员), 607
esp_ble_mesh_sensor_descriptor_get_t::property_id (C++ 成员), 607
ESP_BLE_MESH_SENSOR_DESCRIPTOR_LEN (C 宏), 619
esp_ble_mesh_sensor_descriptor_status_cb_t (C++ 类), 609
esp_ble_mesh_sensor_descriptor_status_cb_t::descriptor (C++ 成员), 609
esp_ble_mesh_sensor_descriptor_t (C++ 类), 611
esp_ble_mesh_sensor_descriptor_t::measure_period (C++ 成员), 611
esp_ble_mesh_sensor_descriptor_t::negative_tolerance (C++ 成员), 611
esp_ble_mesh_sensor_descriptor_t::positive_tolerance (C++ 成员), 611
esp_ble_mesh_sensor_descriptor_t::sampling_function (C++ 成员), 611
esp_ble_mesh_sensor_descriptor_t::update_interval (C++ 成员), 611
ESP_BLE_MESH_SENSOR_DIVISOR_TRIGGER_TYPE_LEN (C 宏), 619
esp_ble_mesh_sensor_get_t (C++ 类), 608
esp_ble_mesh_sensor_get_t::op_en (C++ 成员), 608
esp_ble_mesh_sensor_get_t::property_id (C++ 成员), 608
esp_ble_mesh_sensor_message_opcode_t (C++ 类型), 431
ESP_BLE_MESH_SENSOR_NOT_APPL_MEASURE_PERIOD (C 宏), 619
ESP_BLE_MESH_SENSOR_NOT_APPL_UPDATE_INTERVAL (C 宏), 619

ESP_BLE_MESH_SENSOR_PERIOD_DIVISOR_MAX_VALUE (C 宏), 619
ESP_BLE_MESH_SENSOR_PROPERTY_ID_LEN (C 宏), 619
esp_ble_mesh_sensor_sample_func (C++ 类型), 622
esp_ble_mesh_sensor_series_column_t (C++ 类), 613
esp_ble_mesh_sensor_series_column_t::column_width (C++ 成员), 613
esp_ble_mesh_sensor_series_column_t::raw_value1 (C++ 成员), 613
esp_ble_mesh_sensor_series_column_t::raw_value2 (C++ 成员), 613
esp_ble_mesh_sensor_series_get_t (C++ 类), 609
esp_ble_mesh_sensor_series_get_t::op_en (C++ 成员), 609
esp_ble_mesh_sensor_series_get_t::property_id (C++ 成员), 609
esp_ble_mesh_sensor_series_get_t::raw_value_x1 (C++ 成员), 609
esp_ble_mesh_sensor_series_get_t::raw_value_x2 (C++ 成员), 609
esp_ble_mesh_sensor_series_status_cb_t (C++ 类), 610
esp_ble_mesh_sensor_series_status_cb_t::property_id (C++ 成员), 611
esp_ble_mesh_sensor_series_status_cb_t::sensor_series (C++ 成员), 611
esp_ble_mesh_sensor_server_cb_event_t (C++ 类型), 623
esp_ble_mesh_sensor_server_cb_param_t (C++ 类), 617
esp_ble_mesh_sensor_server_cb_param_t::ctx (C++ 成员), 617
esp_ble_mesh_sensor_server_cb_param_t::model (C++ 成员), 617
esp_ble_mesh_sensor_server_cb_param_t::value (C++ 成员), 618
esp_ble_mesh_sensor_server_cb_t (C++ 类型), 622
esp_ble_mesh_sensor_server_cb_value_t (C++ 类型), 606
esp_ble_mesh_sensor_server_cb_value_t::get (C++ 成员), 606
esp_ble_mesh_sensor_server_cb_value_t::set (C++ 成员), 606
esp_ble_mesh_sensor_server_cb_value_t::state_change (C++ 成员), 606
ESP_BLE_MESH_SENSOR_SERVER_EVT_MAX (C++ 枚举子), 623
ESP_BLE_MESH_SENSOR_SERVER_RECV_GET_MSG_EVT (C++ 枚举子), 623
esp_ble_mesh_sensor_server_recv_get_msg_t (C++ 类型), 605
esp_ble_mesh_sensor_server_recv_get_msg_t::sensor_cadence (C++ 成员), 605
esp_ble_mesh_sensor_server_recv_get_msg_t::sensor_column (C++ 成员), 606
esp_ble_mesh_sensor_server_recv_get_msg_t::sensor_data (C++ 成员), 606
esp_ble_mesh_sensor_server_recv_get_msg_t::sensor_descript (C++ 成员), 605
esp_ble_mesh_sensor_server_recv_get_msg_t::sensor_series (C++ 成员), 606
esp_ble_mesh_sensor_server_recv_get_msg_t::sensor_setting (C++ 成员), 605
esp_ble_mesh_sensor_server_recv_get_msg_t::sensor_settings (C++ 成员), 605
ESP_BLE_MESH_SENSOR_SERVER_RECV_SET_MSG_EVT (C++ 枚举子), 623
esp_ble_mesh_sensor_server_recv_set_msg_t (C++ 类型), 606
esp_ble_mesh_sensor_server_recv_set_msg_t::sensor_cadence (C++ 成员), 606
esp_ble_mesh_sensor_server_recv_set_msg_t::sensor_setting (C++ 成员), 606
ESP_BLE_MESH_SENSOR_SERVER_STATE_CHANGE_EVT (C++ 枚举子), 623
esp_ble_mesh_sensor_server_state_change_t (C++ 类型), 605
esp_ble_mesh_sensor_server_state_change_t::sensor_cadence_ (C++ 成员), 605

esp_ble_mesh_sensor_server_state_change_t::sensor_setting_set (C++ 成员), 605
ESP_BLE_MESH_SENSOR_SETTING_ACCESS_LEN (C 宏), 619
ESP_BLE_MESH_SENSOR_SETTING_ACCESS_READ (C 宏), 619
ESP_BLE_MESH_SENSOR_SETTING_ACCESS_READ_WRITE (C 宏), 619
esp_ble_mesh_sensor_setting_get_t (C++ 类), 608
esp_ble_mesh_sensor_setting_get_t::sensor_property_id (C++ 成员), 608
esp_ble_mesh_sensor_setting_get_t::sensor_setting_property_id (C++ 成员), 608
ESP_BLE_MESH_SENSOR_SETTING_PROPERTY_ID_LEN (C 宏), 619
esp_ble_mesh_sensor_setting_set_t (C++ 类), 608
esp_ble_mesh_sensor_setting_set_t::sensor_property_id (C++ 成员), 608
esp_ble_mesh_sensor_setting_set_t::sensor_setting_property_id (C++ 成员), 608
esp_ble_mesh_sensor_setting_set_t::sensor_setting_raw (C++ 成员), 608
esp_ble_mesh_sensor_setting_status_cb_t (C++ 类), 610
esp_ble_mesh_sensor_setting_status_cb_t::op_en (C++ 成员), 610
esp_ble_mesh_sensor_setting_status_cb_t::sensor_property_id (C++ 成员), 610
esp_ble_mesh_sensor_setting_status_cb_t::sensor_setting_access (C++ 成员), 610
esp_ble_mesh_sensor_setting_status_cb_t::sensor_setting_property_id (C++ 成员), 610
esp_ble_mesh_sensor_setting_status_cb_t::sensor_setting_raw (C++ 成员), 610
esp_ble_mesh_sensor_setting_t (C++ 类), 611
esp_ble_mesh_sensor_setting_t::access (C++ 成员), 612
esp_ble_mesh_sensor_setting_t::property_id (C++ 成员), 612
esp_ble_mesh_sensor_setting_t::raw (C++ 成员), 612
esp_ble_mesh_sensor_settings_get_t (C++ 类), 607
esp_ble_mesh_sensor_settings_get_t::sensor_property_id (C++ 成员), 608
esp_ble_mesh_sensor_settings_status_cb_t (C++ 类), 609
esp_ble_mesh_sensor_settings_status_cb_t::sensor_property_id (C++ 成员), 610
esp_ble_mesh_sensor_settings_status_cb_t::sensor_setting_property_id (C++ 成员), 610
esp_ble_mesh_sensor_setup_srv_t (C++ 类), 614
esp_ble_mesh_sensor_setup_srv_t::model (C++ 成员), 614
esp_ble_mesh_sensor_setup_srv_t::rsp_ctrl (C++ 成员), 614
esp_ble_mesh_sensor_setup_srv_t::state_count (C++ 成员), 614
esp_ble_mesh_sensor_setup_srv_t::states (C++ 成员), 614
esp_ble_mesh_sensor_srv_t (C++ 类), 614
esp_ble_mesh_sensor_srv_t::model (C++ 成员), 614
esp_ble_mesh_sensor_srv_t::rsp_ctrl (C++ 成员), 614
esp_ble_mesh_sensor_srv_t::state_count (C++ 成员), 614
esp_ble_mesh_sensor_srv_t::states (C++ 成员), 614
esp_ble_mesh_sensor_state_t (C++ 类), 613
esp_ble_mesh_sensor_state_t::cadence (C++ 成员), 613
esp_ble_mesh_sensor_state_t::descriptor (C++ 成员), 613
esp_ble_mesh_sensor_state_t::sensor_data (C++ 成员), 613
esp_ble_mesh_sensor_state_t::sensor_property_id (C++ 成员), 613
esp_ble_mesh_sensor_state_t::series_column (C++ 成员), 613
esp_ble_mesh_sensor_state_t::setting_count (C++ 成员), 613

esp_ble_mesh_sensor_state_t::settings (C++ 成员), 613
esp_ble_mesh_sensor_status_cb_t (C++ 类), 610
esp_ble_mesh_sensor_status_cb_t::marshalled_sensor_data (C++ 成员), 610
ESP_BLE_MESH_SENSOR_STATUS_MIN_INTERVAL_LEN (C 宏), 619
ESP_BLE_MESH_SENSOR_STATUS_MIN_INTERVAL_MAX (C 宏), 619
ESP_BLE_MESH_SENSOR_STATUS_TRIGGER_TYPE_CHAR (C 宏), 619
ESP_BLE_MESH_SENSOR_STATUS_TRIGGER_TYPE_UINT16 (C 宏), 620
ESP_BLE_MESH_SENSOR_UNSPECIFIED_NEG_TOLERANCE (C 宏), 619
ESP_BLE_MESH_SENSOR_UNSPECIFIED_POS_TOLERANCE (C 宏), 619
ESP_BLE_MESH_SERVER_AUTO_RSP (C 宏), 430
ESP_BLE_MESH_SERVER_FLAG_MAX (C++ 枚举子), 438
esp_ble_mesh_server_model_send_msg (C++ 函数), 445
ESP_BLE_MESH_SERVER_MODEL_STATE_MAX (C++ 枚举子), 439
esp_ble_mesh_server_model_update_state (C++ 函数), 446
ESP_BLE_MESH_SERVER_MODEL_UPDATE_STATE_COMP_EVT (C++ 枚举子), 440
esp_ble_mesh_server_recv_gen_admin_property_get_t (C++ 类), 520
esp_ble_mesh_server_recv_gen_admin_property_get_t::property_id (C++ 成员), 521
esp_ble_mesh_server_recv_gen_admin_property_set_t (C++ 类), 525
esp_ble_mesh_server_recv_gen_admin_property_set_t::property_id (C++ 成员), 525
esp_ble_mesh_server_recv_gen_admin_property_set_t::property_value (C++ 成员), 525
esp_ble_mesh_server_recv_gen_admin_property_set_t::user_access (C++ 成员), 525
esp_ble_mesh_server_recv_gen_client_properties_get_t (C++ 类), 521
esp_ble_mesh_server_recv_gen_client_properties_get_t::prop (C++ 成员), 521
esp_ble_mesh_server_recv_gen_def_trans_time_set_t (C++ 类), 523
esp_ble_mesh_server_recv_gen_def_trans_time_set_t::trans_t (C++ 成员), 523
esp_ble_mesh_server_recv_gen_delta_set_t (C++ 类), 522
esp_ble_mesh_server_recv_gen_delta_set_t::delay (C++ 成员), 522
esp_ble_mesh_server_recv_gen_delta_set_t::delta_level (C++ 成员), 522
esp_ble_mesh_server_recv_gen_delta_set_t::op_en (C++ 成员), 522
esp_ble_mesh_server_recv_gen_delta_set_t::tid (C++ 成员), 522
esp_ble_mesh_server_recv_gen_delta_set_t::trans_time (C++ 成员), 522
esp_ble_mesh_server_recv_gen_level_set_t (C++ 类), 521
esp_ble_mesh_server_recv_gen_level_set_t::delay (C++ 成员), 522
esp_ble_mesh_server_recv_gen_level_set_t::level (C++ 成员), 522
esp_ble_mesh_server_recv_gen_level_set_t::op_en (C++ 成员), 522
esp_ble_mesh_server_recv_gen_level_set_t::tid (C++ 成员), 522
esp_ble_mesh_server_recv_gen_level_set_t::trans_time (C++ 成员), 522
esp_ble_mesh_server_recv_gen_loc_global_set_t (C++ 类), 524
esp_ble_mesh_server_recv_gen_loc_global_set_t::global_alti (C++ 成员), 524
esp_ble_mesh_server_recv_gen_loc_global_set_t::global_lati (C++ 成员), 524
esp_ble_mesh_server_recv_gen_loc_global_set_t::global_long (C++ 成员), 524
esp_ble_mesh_server_recv_gen_loc_local_set_t (C++ 类), 524
esp_ble_mesh_server_recv_gen_loc_local_set_t::floor_number (C++ 成员), 524

esp_ble_mesh_server_recv_gen_loc_local_set_t::esp_ble_mesh_server_recv_gen_onpowerup_set_t
(C++ 成员), 524 (C++ 类), 523
esp_ble_mesh_server_recv_gen_loc_local_set_t::esp_ble_mesh_server_recv_gen_onpowerup_set_t::onpowerup
(C++ 成员), 524 (C++ 成员), 523
esp_ble_mesh_server_recv_gen_loc_local_set_t::esp_ble_mesh_server_recv_gen_power_default_set_t
(C++ 成员), 524 (C++ 类), 523
esp_ble_mesh_server_recv_gen_loc_local_set_t::esp_ble_mesh_server_recv_gen_power_default_set_t::power
(C++ 成员), 524 (C++ 成员), 524
esp_ble_mesh_server_recv_gen_manufacturer_property_get_t esp_ble_mesh_server_recv_gen_power_level_set_t
(C++ 类), 521 (C++ 类), 523
esp_ble_mesh_server_recv_gen_manufacturer_property_get_t::property_id esp_ble_mesh_server_recv_gen_power_level_set_t::delay
(C++ 成员), 521 (C++ 成员), 523
esp_ble_mesh_server_recv_gen_manufacturer_property_set_t esp_ble_mesh_server_recv_gen_power_level_set_t::op_en
(C++ 类), 525 (C++ 成员), 523
esp_ble_mesh_server_recv_gen_manufacturer_property_set_t::property_id esp_ble_mesh_server_recv_gen_power_level_set_t::power
(C++ 成员), 525 (C++ 成员), 523
esp_ble_mesh_server_recv_gen_manufacturer_property_set_t::user_access esp_ble_mesh_server_recv_gen_power_level_set_t::tid
(C++ 成员), 525 (C++ 成员), 523
esp_ble_mesh_server_recv_gen_move_set_t esp_ble_mesh_server_recv_gen_power_level_set_t::trans_time
(C++ 类), 522 (C++ 成员), 523
esp_ble_mesh_server_recv_gen_move_set_t::delay esp_ble_mesh_server_recv_gen_power_range_set_t
(C++ 成员), 523 (C++ 类), 524
esp_ble_mesh_server_recv_gen_move_set_t::delta_level esp_ble_mesh_server_recv_gen_power_range_set_t::range_max
(C++ 成员), 522 (C++ 成员), 524
esp_ble_mesh_server_recv_gen_move_set_t::op_en esp_ble_mesh_server_recv_gen_power_range_set_t::range_min
(C++ 成员), 522 (C++ 成员), 524
esp_ble_mesh_server_recv_gen_move_set_t::tid esp_ble_mesh_server_recv_gen_user_property_get_t
(C++ 成员), 522 (C++ 类), 520
esp_ble_mesh_server_recv_gen_move_set_t::trans_time esp_ble_mesh_server_recv_gen_user_property_get_t::property
(C++ 成员), 523 (C++ 成员), 520
esp_ble_mesh_server_recv_gen_onoff_set_t esp_ble_mesh_server_recv_gen_user_property_set_t
(C++ 类), 521 (C++ 类), 525
esp_ble_mesh_server_recv_gen_onoff_set_t::delay esp_ble_mesh_server_recv_gen_user_property_set_t::property
(C++ 成员), 521 (C++ 成员), 525
esp_ble_mesh_server_recv_gen_onoff_set_t::onoff esp_ble_mesh_server_recv_gen_user_property_set_t::property
(C++ 成员), 521 (C++ 成员), 525
esp_ble_mesh_server_recv_gen_onoff_set_t::op_en esp_ble_mesh_server_recv_light_ctl_default_set_t
(C++ 成员), 521 (C++ 类), 590
esp_ble_mesh_server_recv_gen_onoff_set_t::tid esp_ble_mesh_server_recv_light_ctl_default_set_t::delta_uv
(C++ 成员), 521 (C++ 成员), 590
esp_ble_mesh_server_recv_gen_onoff_set_t::trans_time esp_ble_mesh_server_recv_light_ctl_default_set_t::lightnes
(C++ 成员), 521 (C++ 成员), 590

esp_ble_mesh_server_recv_light_ctl_default_set_t::temperature (C++ 成员), 590
esp_ble_mesh_server_recv_light_ctl_set_t (C++ 类), 588
esp_ble_mesh_server_recv_light_ctl_set_t::delay (C++ 成员), 589
esp_ble_mesh_server_recv_light_ctl_set_t::delta_uv (C++ 成员), 589
esp_ble_mesh_server_recv_light_ctl_set_t::lightness (C++ 成员), 589
esp_ble_mesh_server_recv_light_ctl_set_t::op_en (C++ 成员), 589
esp_ble_mesh_server_recv_light_ctl_set_t::temperature (C++ 成员), 589
esp_ble_mesh_server_recv_light_ctl_set_t::tid (C++ 成员), 589
esp_ble_mesh_server_recv_light_ctl_set_t::trans_time (C++ 成员), 589
esp_ble_mesh_server_recv_light_ctl_temperature_range_set_t (C++ 类), 589
esp_ble_mesh_server_recv_light_ctl_temperature_range_set_t::range_max (C++ 成员), 590
esp_ble_mesh_server_recv_light_ctl_temperature_range_set_t::range_min (C++ 成员), 590
esp_ble_mesh_server_recv_light_ctl_temperature_set_t (C++ 类), 589
esp_ble_mesh_server_recv_light_ctl_temperature_set_t::delay (C++ 成员), 589
esp_ble_mesh_server_recv_light_ctl_temperature_set_t::delta_uv (C++ 成员), 589
esp_ble_mesh_server_recv_light_ctl_temperature_set_t::op_en (C++ 成员), 589
esp_ble_mesh_server_recv_light_ctl_temperature_set_t::temperature (C++ 成员), 589
esp_ble_mesh_server_recv_light_ctl_temperature_set_t::tid (C++ 成员), 589
esp_ble_mesh_server_recv_light_ctl_temperature_set_t::trans_time (C++ 成员), 589
esp_ble_mesh_server_recv_light_hsl_default_set_t (C++ 类), 591
esp_ble_mesh_server_recv_light_hsl_default_set_t::lightness (C++ 成员), 592
esp_ble_mesh_server_recv_light_hsl_default_set_t::saturation (C++ 成员), 592
esp_ble_mesh_server_recv_light_hsl_hue_set_t (C++ 类), 591
esp_ble_mesh_server_recv_light_hsl_hue_set_t::delay (C++ 成员), 591
esp_ble_mesh_server_recv_light_hsl_hue_set_t::hue (C++ 成员), 591
esp_ble_mesh_server_recv_light_hsl_hue_set_t::op_en (C++ 成员), 591
esp_ble_mesh_server_recv_light_hsl_hue_set_t::tid (C++ 成员), 591
esp_ble_mesh_server_recv_light_hsl_hue_set_t::trans_time (C++ 成员), 591
esp_ble_mesh_server_recv_light_hsl_range_set_t (C++ 类), 592
esp_ble_mesh_server_recv_light_hsl_range_set_t::hue_range_max (C++ 成员), 592
esp_ble_mesh_server_recv_light_hsl_range_set_t::hue_range_min (C++ 成员), 592
esp_ble_mesh_server_recv_light_hsl_range_set_t::saturation_max (C++ 成员), 592
esp_ble_mesh_server_recv_light_hsl_range_set_t::saturation_min (C++ 成员), 592
esp_ble_mesh_server_recv_light_hsl_saturation_set_t (C++ 类), 591
esp_ble_mesh_server_recv_light_hsl_saturation_set_t::delay (C++ 成员), 591
esp_ble_mesh_server_recv_light_hsl_saturation_set_t::op_en (C++ 成员), 591
esp_ble_mesh_server_recv_light_hsl_saturation_set_t::saturation (C++ 成员), 591
esp_ble_mesh_server_recv_light_hsl_saturation_set_t::tid (C++ 成员), 591
esp_ble_mesh_server_recv_light_hsl_saturation_set_t::trans_time (C++ 成员), 591
esp_ble_mesh_server_recv_light_hsl_set_t (C++ 类), 590
esp_ble_mesh_server_recv_light_hsl_set_t::delay (C++ 成员), 590

esp_ble_mesh_server_recv_light_hsl_set_t::hue (C++ 成员), 590
esp_ble_mesh_server_recv_light_hsl_set_t::lightness (C++ 成员), 590
esp_ble_mesh_server_recv_light_hsl_set_t::op_en (C++ 成员), 590
esp_ble_mesh_server_recv_light_hsl_set_t::saturation (C++ 成员), 590
esp_ble_mesh_server_recv_light_hsl_set_t::tid (C++ 成员), 590
esp_ble_mesh_server_recv_light_hsl_set_t::trans_time (C++ 成员), 590
esp_ble_mesh_server_recv_light_lc_light_onoff_set_t (C++ 类), 594
esp_ble_mesh_server_recv_light_lc_light_onoff_set_t::delay (C++ 成员), 594
esp_ble_mesh_server_recv_light_lc_light_onoff_set_t::light_onoff (C++ 成员), 594
esp_ble_mesh_server_recv_light_lc_light_onoff_set_t::op_en (C++ 成员), 594
esp_ble_mesh_server_recv_light_lc_light_onoff_set_t::tid (C++ 成员), 594
esp_ble_mesh_server_recv_light_lc_light_onoff_set_t::trans_time (C++ 成员), 594
esp_ble_mesh_server_recv_light_lc_mode_set_t (C++ 类), 593
esp_ble_mesh_server_recv_light_lc_mode_set_t::mode (C++ 成员), 593
esp_ble_mesh_server_recv_light_lc_om_set_t (C++ 类), 593
esp_ble_mesh_server_recv_light_lc_om_set_t::mode (C++ 成员), 594
esp_ble_mesh_server_recv_light_lc_property_get_t (C++ 类), 587
esp_ble_mesh_server_recv_light_lc_property_get_t::property_id (C++ 成员), 587
esp_ble_mesh_server_recv_light_lc_property_set_t (C++ 类), 594
esp_ble_mesh_server_recv_light_lc_property_set_t::property_id (C++ 成员), 594
esp_ble_mesh_server_recv_light_lc_property_set_t::property_value (C++ 成员), 594
esp_ble_mesh_server_recv_light_lightness_default_set_t (C++ 类), 588
esp_ble_mesh_server_recv_light_lightness_default_set_t::li (C++ 成员), 588
esp_ble_mesh_server_recv_light_lightness_linear_set_t (C++ 类), 588
esp_ble_mesh_server_recv_light_lightness_linear_set_t::del (C++ 成员), 588
esp_ble_mesh_server_recv_light_lightness_linear_set_t::lig (C++ 成员), 588
esp_ble_mesh_server_recv_light_lightness_linear_set_t::op_ (C++ 成员), 588
esp_ble_mesh_server_recv_light_lightness_linear_set_t::tid (C++ 成员), 588
esp_ble_mesh_server_recv_light_lightness_linear_set_t::tra (C++ 成员), 588
esp_ble_mesh_server_recv_light_lightness_range_set_t (C++ 类), 588
esp_ble_mesh_server_recv_light_lightness_range_set_t::rang (C++ 成员), 588
esp_ble_mesh_server_recv_light_lightness_range_set_t::rang (C++ 成员), 588
esp_ble_mesh_server_recv_light_lightness_set_t (C++ 类), 587
esp_ble_mesh_server_recv_light_lightness_set_t::delay (C++ 成员), 588
esp_ble_mesh_server_recv_light_lightness_set_t::lightness (C++ 成员), 587
esp_ble_mesh_server_recv_light_lightness_set_t::op_en (C++ 成员), 587
esp_ble_mesh_server_recv_light_lightness_set_t::tid (C++ 成员), 587
esp_ble_mesh_server_recv_light_lightness_set_t::trans_time (C++ 成员), 588
esp_ble_mesh_server_recv_light_xyl_default_set_t (C++ 类), 593
esp_ble_mesh_server_recv_light_xyl_default_set_t::lightnes (C++ 成员), 593
esp_ble_mesh_server_recv_light_xyl_default_set_t::x (C++ 成员), 593
esp_ble_mesh_server_recv_light_xyl_default_set_t::y (C++ 成员), 593

esp_ble_mesh_server_recv_light_xyl_range_set_t (C++ 类), 593
esp_ble_mesh_server_recv_light_xyl_range_set_t::range_max (C++ 成员), 593
esp_ble_mesh_server_recv_light_xyl_range_set_t::range_min (C++ 成员), 593
esp_ble_mesh_server_recv_light_xyl_range_set_t::y_range_max (C++ 成员), 593
esp_ble_mesh_server_recv_light_xyl_range_set_t::y_range_min (C++ 成员), 593
esp_ble_mesh_server_recv_light_xyl_set_t (C++ 类), 592
esp_ble_mesh_server_recv_light_xyl_set_t::delay (C++ 成员), 593
esp_ble_mesh_server_recv_light_xyl_set_t::lightness (C++ 成员), 592
esp_ble_mesh_server_recv_light_xyl_set_t::op_en (C++ 成员), 592
esp_ble_mesh_server_recv_light_xyl_set_t::tid (C++ 成员), 592
esp_ble_mesh_server_recv_light_xyl_set_t::trans_time (C++ 成员), 592
esp_ble_mesh_server_recv_light_xyl_set_t::x (C++ 成员), 592
esp_ble_mesh_server_recv_light_xyl_set_t::y (C++ 成员), 592
esp_ble_mesh_server_recv_scene_delete_t (C++ 类), 644
esp_ble_mesh_server_recv_scene_delete_t::scene_number (C++ 成员), 645
esp_ble_mesh_server_recv_scene_recall_t (C++ 类), 644
esp_ble_mesh_server_recv_scene_recall_t::delay (C++ 成员), 644
esp_ble_mesh_server_recv_scene_recall_t::op_en (C++ 成员), 644
esp_ble_mesh_server_recv_scene_recall_t::scene_number (C++ 成员), 644
esp_ble_mesh_server_recv_scene_recall_t::tid (C++ 成员), 644
esp_ble_mesh_server_recv_scene_recall_t::trans_time (C++ 成员), 644
esp_ble_mesh_server_recv_scene_store_t (C++ 类), 644
esp_ble_mesh_server_recv_scene_store_t::scene_number (C++ 成员), 644
esp_ble_mesh_server_recv_scheduler_act_get_t (C++ 类), 642
esp_ble_mesh_server_recv_scheduler_act_get_t::index (C++ 成员), 643
esp_ble_mesh_server_recv_scheduler_act_set_t (C++ 类), 645
esp_ble_mesh_server_recv_scheduler_act_set_t::action (C++ 成员), 645
esp_ble_mesh_server_recv_scheduler_act_set_t::day (C++ 成员), 645
esp_ble_mesh_server_recv_scheduler_act_set_t::day_of_week (C++ 成员), 645
esp_ble_mesh_server_recv_scheduler_act_set_t::hour (C++ 成员), 645
esp_ble_mesh_server_recv_scheduler_act_set_t::index (C++ 成员), 645
esp_ble_mesh_server_recv_scheduler_act_set_t::minute (C++ 成员), 645
esp_ble_mesh_server_recv_scheduler_act_set_t::month (C++ 成员), 645
esp_ble_mesh_server_recv_scheduler_act_set_t::scene_number (C++ 成员), 645
esp_ble_mesh_server_recv_scheduler_act_set_t::second (C++ 成员), 645
esp_ble_mesh_server_recv_scheduler_act_set_t::trans_time (C++ 成员), 645
esp_ble_mesh_server_recv_scheduler_act_set_t::year (C++ 成员), 645
esp_ble_mesh_server_recv_sensor_cadence_get_t (C++ 类), 615
esp_ble_mesh_server_recv_sensor_cadence_get_t::property_id (C++ 成员), 616
esp_ble_mesh_server_recv_sensor_cadence_set_t (C++ 类), 617
esp_ble_mesh_server_recv_sensor_cadence_set_t::cadence (C++ 成员), 617
esp_ble_mesh_server_recv_sensor_cadence_set_t::property_id (C++ 成员), 617

esp_ble_mesh_server_recv_sensor_column_get_t (C++ 类), 616
esp_ble_mesh_server_recv_sensor_column_get_t::property_id (C++ 成员), 616
esp_ble_mesh_server_recv_sensor_column_get_t::raw_value (C++ 成员), 616
esp_ble_mesh_server_recv_sensor_descriptor_get_t (C++ 类), 615
esp_ble_mesh_server_recv_sensor_descriptor_get_t::op_en (C++ 成员), 615
esp_ble_mesh_server_recv_sensor_descriptor_get_t::property_id (C++ 成员), 615
esp_ble_mesh_server_recv_sensor_get_t (C++ 类), 616
esp_ble_mesh_server_recv_sensor_get_t::op_en (C++ 成员), 616
esp_ble_mesh_server_recv_sensor_get_t::property_id (C++ 成员), 616
esp_ble_mesh_server_recv_sensor_series_get_t (C++ 类), 616
esp_ble_mesh_server_recv_sensor_series_get_t::op_en (C++ 成员), 617
esp_ble_mesh_server_recv_sensor_series_get_t::property_id (C++ 成员), 617
esp_ble_mesh_server_recv_sensor_series_get_t::raw_value (C++ 成员), 617
esp_ble_mesh_server_recv_sensor_setting_get_t (C++ 类), 616
esp_ble_mesh_server_recv_sensor_setting_get_t::property_id (C++ 成员), 616
esp_ble_mesh_server_recv_sensor_setting_get_t::setting_property_id (C++ 成员), 616
esp_ble_mesh_server_recv_sensor_setting_set_t (C++ 类), 617
esp_ble_mesh_server_recv_sensor_setting_set_t::property_id (C++ 成员), 617
esp_ble_mesh_server_recv_sensor_setting_set_t::setting_property_id (C++ 成员), 617
esp_ble_mesh_server_recv_sensor_setting_set_t::setting_raw (C++ 成员), 617
esp_ble_mesh_server_recv_sensor_settings_get_t (C++ 类), 616
esp_ble_mesh_server_recv_sensor_settings_get_t::property_id (C++ 成员), 616
esp_ble_mesh_server_recv_sensor_status_t (C++ 类), 594
esp_ble_mesh_server_recv_sensor_status_t::data (C++ 成员), 594
esp_ble_mesh_server_recv_tai_utc_delta_set_t (C++ 类), 643
esp_ble_mesh_server_recv_tai_utc_delta_set_t::padding (C++ 成员), 643
esp_ble_mesh_server_recv_tai_utc_delta_set_t::tai_delta_ch (C++ 成员), 644
esp_ble_mesh_server_recv_tai_utc_delta_set_t::tai_utc_delt (C++ 成员), 643
esp_ble_mesh_server_recv_time_role_set_t (C++ 类), 644
esp_ble_mesh_server_recv_time_role_set_t::time_role (C++ 成员), 644
esp_ble_mesh_server_recv_time_set_t (C++ 类), 643
esp_ble_mesh_server_recv_time_set_t::subsecond (C++ 成员), 643
esp_ble_mesh_server_recv_time_set_t::tai_seconds (C++ 成员), 643
esp_ble_mesh_server_recv_time_set_t::tai_utc_delta (C++ 成员), 643
esp_ble_mesh_server_recv_time_set_t::time_authority (C++ 成员), 643
esp_ble_mesh_server_recv_time_set_t::time_zone_offset (C++ 成员), 643
esp_ble_mesh_server_recv_time_set_t::uncertainty (C++ 成员), 643
esp_ble_mesh_server_recv_time_status_t (C++ 类), 645
esp_ble_mesh_server_recv_time_status_t::subsecond (C++ 成员), 646
esp_ble_mesh_server_recv_time_status_t::tai_seconds (C++ 成员), 646
esp_ble_mesh_server_recv_time_status_t::tai_utc_delta (C++ 成员), 646
esp_ble_mesh_server_recv_time_status_t::time_authority (C++ 成员), 646

esp_ble_mesh_server_recv_time_status_t::time_zone_offset (C++ 成员), 646
esp_ble_mesh_server_recv_time_status_t::uncertainty (C++ 成员), 646
esp_ble_mesh_server_recv_time_zone_set_t (C++ 类), 643
esp_ble_mesh_server_recv_time_zone_set_t::tai_zone_change (C++ 成员), 643
esp_ble_mesh_server_recv_time_zone_set_t::time_zone_offset_new (C++ 成员), 643
ESP_BLE_MESH_SERVER_RSP_BY_APP (C 宏), 430
esp_ble_mesh_server_rsp_ctrl_t (C++ 类), 410
esp_ble_mesh_server_rsp_ctrl_t::get_auto_rsp (C++ 成员), 410
esp_ble_mesh_server_rsp_ctrl_t::set_auto_rsp (C++ 成员), 411
esp_ble_mesh_server_rsp_ctrl_t::status_auto_rsp (C++ 成员), 411
esp_ble_mesh_server_state_type_t (C++ 类型), 439
esp_ble_mesh_server_state_value_t (C++ 类型), 396
esp_ble_mesh_server_state_value_t::delta_uv (C++ 成员), 397
esp_ble_mesh_server_state_value_t::gen_level (C++ 成员), 396
esp_ble_mesh_server_state_value_t::gen_onoff (C++ 成员), 396
esp_ble_mesh_server_state_value_t::gen_onpowerup (C++ 成员), 396
esp_ble_mesh_server_state_value_t::gen_power_actual (C++ 成员), 396
esp_ble_mesh_server_state_value_t::hue (C++ 成员), 397
esp_ble_mesh_server_state_value_t::level (C++ 成员), 396
esp_ble_mesh_server_state_value_t::light_ctl_lightness (C++ 成员), 397
esp_ble_mesh_server_state_value_t::light_ctl_temp_delta_uv (C++ 成员), 397
esp_ble_mesh_server_state_value_t::light_hsl_hue (C++ 成员), 397
esp_ble_mesh_server_state_value_t::light_hsl_lightness (C++ 成员), 397
esp_ble_mesh_server_state_value_t::light_hsl_saturation (C++ 成员), 397
esp_ble_mesh_server_state_value_t::light_lc_light_onoff (C++ 成员), 397
esp_ble_mesh_server_state_value_t::light_lightness_actual (C++ 成员), 397
esp_ble_mesh_server_state_value_t::light_lightness_linear (C++ 成员), 397
esp_ble_mesh_server_state_value_t::light_xyl_lightness (C++ 成员), 397
esp_ble_mesh_server_state_value_t::lightness (C++ 成员), 396
esp_ble_mesh_server_state_value_t::onoff (C++ 成员), 396
esp_ble_mesh_server_state_value_t::onpowerup (C++ 成员), 396
esp_ble_mesh_server_state_value_t::power (C++ 成员), 396
esp_ble_mesh_server_state_value_t::saturation (C++ 成员), 397
esp_ble_mesh_server_state_value_t::temperature (C++ 成员), 397
ESP_BLE_MESH_SERVER_TRANS_TIMER_START (C++ 枚举子), 438
esp_ble_mesh_set_fast_prov_action (C++ 函数), 458
ESP_BLE_MESH_SET_FAST_PROV_ACTION_COMP_EVT (C++ 枚举子), 437
esp_ble_mesh_set_fast_prov_info (C++ 函数), 457
ESP_BLE_MESH_SET_FAST_PROV_INFO_COMP_EVT (C++ 枚举子), 437
esp_ble_mesh_set_unprovisioned_device_name (C++ 函数), 453
ESP_BLE_MESH_SIG_MODEL (C 宏), 414
ESP_BLE_MESH_START_BLE_ADVERTISING_COMP_EVT (C++ 枚举子), 438
esp_ble_mesh_state_change_cfg_appkey_add_t (C++ 类), 488
esp_ble_mesh_state_change_cfg_appkey_add_t::app_idx

(C++ 成员), 489
esp_ble_mesh_state_change_cfg_appkey_add_t::appkey_idx
(C++ 成员), 489
esp_ble_mesh_state_change_cfg_appkey_add_t::net_idx
(C++ 成员), 489
esp_ble_mesh_state_change_cfg_appkey_delete_t
(C++ 类), 489
esp_ble_mesh_state_change_cfg_appkey_delete_t::app_idx
(C++ 成员), 489
esp_ble_mesh_state_change_cfg_appkey_delete_t::net_idx
(C++ 成员), 489
esp_ble_mesh_state_change_cfg_appkey_update_t
(C++ 类), 489
esp_ble_mesh_state_change_cfg_appkey_update_t::app_idx
(C++ 成员), 489
esp_ble_mesh_state_change_cfg_appkey_update_t::appkey
(C++ 成员), 489
esp_ble_mesh_state_change_cfg_appkey_update_t::net_idx
(C++ 成员), 489
esp_ble_mesh_state_change_cfg_kr_phase_set_t
(C++ 类), 490
esp_ble_mesh_state_change_cfg_kr_phase_set_t::kr_phase
(C++ 成员), 490
esp_ble_mesh_state_change_cfg_kr_phase_set_t::net_idx
(C++ 成员), 490
esp_ble_mesh_state_change_cfg_mod_pub_set_t
(C++ 类), 486
esp_ble_mesh_state_change_cfg_mod_pub_set_t::app_idx
(C++ 成员), 487
esp_ble_mesh_state_change_cfg_mod_pub_set_t::company_id
(C++ 成员), 487
esp_ble_mesh_state_change_cfg_mod_pub_set_t::cred_flag
(C++ 成员), 487
esp_ble_mesh_state_change_cfg_mod_pub_set_t::element_addr
(C++ 成员), 487
esp_ble_mesh_state_change_cfg_mod_pub_set_t::model_id
(C++ 成员), 487
esp_ble_mesh_state_change_cfg_mod_pub_set_t::pub_addr
(C++ 成员), 487
esp_ble_mesh_state_change_cfg_mod_pub_set_t::pub_period
(C++ 成员), 487
esp_ble_mesh_state_change_cfg_mod_pub_set_t::pub_retransmit
(C++ 成员), 487
esp_ble_mesh_state_change_cfg_mod_pub_set_t::pub_ttl
(C++ 成员), 487
esp_ble_mesh_state_change_cfg_model_app_bind_t
(C++ 类), 489
esp_ble_mesh_state_change_cfg_model_app_bind_t::app_idx
(C++ 成员), 489
esp_ble_mesh_state_change_cfg_model_app_bind_t::company_id
(C++ 成员), 490
esp_ble_mesh_state_change_cfg_model_app_bind_t::element_addr
(C++ 成员), 489
esp_ble_mesh_state_change_cfg_model_app_bind_t::model_id
(C++ 成员), 490
esp_ble_mesh_state_change_cfg_model_app_unbind_t
(C++ 类), 490
esp_ble_mesh_state_change_cfg_model_app_unbind_t::app_idx
(C++ 成员), 490
esp_ble_mesh_state_change_cfg_model_app_unbind_t::company_id
(C++ 成员), 490
esp_ble_mesh_state_change_cfg_model_app_unbind_t::element_addr
(C++ 成员), 490
esp_ble_mesh_state_change_cfg_model_app_unbind_t::model_id
(C++ 成员), 490
esp_ble_mesh_state_change_cfg_model_sub_add_t
(C++ 类), 487
esp_ble_mesh_state_change_cfg_model_sub_add_t::company_id
(C++ 成员), 487
esp_ble_mesh_state_change_cfg_model_sub_add_t::element_addr
(C++ 成员), 487
esp_ble_mesh_state_change_cfg_model_sub_add_t::model_id
(C++ 成员), 487
esp_ble_mesh_state_change_cfg_model_sub_add_t::sub_addr
(C++ 成员), 487
esp_ble_mesh_state_change_cfg_model_sub_delete_t
(C++ 类), 487
esp_ble_mesh_state_change_cfg_model_sub_delete_t::company_id
(C++ 成员), 488
esp_ble_mesh_state_change_cfg_model_sub_delete_t::element_addr
(C++ 成员), 488
esp_ble_mesh_state_change_cfg_model_sub_delete_t::model_id
(C++ 成员), 488
esp_ble_mesh_state_change_cfg_model_sub_delete_t::sub_addr

(C++ 成员), 488
esp_ble_mesh_state_change_cfg_netkey_add_t (C++ 类), 488
esp_ble_mesh_state_change_cfg_netkey_add_t::net_idx (C++ 成员), 488
esp_ble_mesh_state_change_cfg_netkey_add_t::net_key (C++ 成员), 488
esp_ble_mesh_state_change_cfg_netkey_delete_t (C++ 类), 488
esp_ble_mesh_state_change_cfg_netkey_delete_t::net_idx (C++ 成员), 488
esp_ble_mesh_state_change_cfg_netkey_update_t (C++ 类), 488
esp_ble_mesh_state_change_cfg_netkey_update_t::net_idx (C++ 成员), 488
esp_ble_mesh_state_change_cfg_netkey_update_t::net_key (C++ 成员), 488
esp_ble_mesh_state_change_gen_admin_property_set_t (C++ 类), 520
esp_ble_mesh_state_change_gen_admin_property_set_t::access (C++ 成员), 520
esp_ble_mesh_state_change_gen_admin_property_set_t::id (C++ 成员), 520
esp_ble_mesh_state_change_gen_admin_property_set_t::user_access (C++ 成员), 520
esp_ble_mesh_state_change_gen_def_trans_time_set_t (C++ 类), 518
esp_ble_mesh_state_change_gen_def_trans_time_set_t::trans_time (C++ 成员), 518
esp_ble_mesh_state_change_gen_delta_set_t (C++ 类), 517
esp_ble_mesh_state_change_gen_delta_set_t::level (C++ 成员), 518
esp_ble_mesh_state_change_gen_level_set_t (C++ 类), 517
esp_ble_mesh_state_change_gen_level_set_t::level (C++ 成员), 517
esp_ble_mesh_state_change_gen_loc_global_set_t (C++ 类), 519
esp_ble_mesh_state_change_gen_loc_global_set_t::altitude (C++ 成员), 519
esp_ble_mesh_state_change_gen_loc_global_set_t::global_altitude (C++ 成员), 519
esp_ble_mesh_state_change_gen_loc_global_set_t::global_east (C++ 成员), 519
esp_ble_mesh_state_change_gen_loc_global_set_t::global_north (C++ 成员), 519
esp_ble_mesh_state_change_gen_loc_global_set_t::longitude (C++ 成员), 519
esp_ble_mesh_state_change_gen_loc_local_set_t (C++ 类), 519
esp_ble_mesh_state_change_gen_loc_local_set_t::altitude (C++ 成员), 519
esp_ble_mesh_state_change_gen_loc_local_set_t::east (C++ 成员), 519
esp_ble_mesh_state_change_gen_loc_local_set_t::floor_number (C++ 成员), 519
esp_ble_mesh_state_change_gen_loc_local_set_t::north (C++ 成员), 519
esp_ble_mesh_state_change_gen_loc_local_set_t::uncertainty (C++ 成员), 519
esp_ble_mesh_state_change_gen_manu_property_set_t (C++ 类), 520
esp_ble_mesh_state_change_gen_manu_property_set_t::access (C++ 成员), 520
esp_ble_mesh_state_change_gen_manu_property_set_t::id (C++ 成员), 520
esp_ble_mesh_state_change_gen_move_set_t (C++ 类), 518
esp_ble_mesh_state_change_gen_move_set_t::level (C++ 成员), 518
esp_ble_mesh_state_change_gen_onoff_set_t (C++ 类), 517
esp_ble_mesh_state_change_gen_onoff_set_t::onoff (C++ 成员), 517
esp_ble_mesh_state_change_gen_onpowerup_set_t (C++ 类), 518
esp_ble_mesh_state_change_gen_onpowerup_set_t::onpowerup (C++ 成员), 518
esp_ble_mesh_state_change_gen_power_default_set_t (C++ 类), 518
esp_ble_mesh_state_change_gen_power_default_set_t::power (C++ 成员), 519
esp_ble_mesh_state_change_gen_power_level_set_t (C++ 类), 518
esp_ble_mesh_state_change_gen_power_level_set_t::power (C++ 成员), 518
esp_ble_mesh_state_change_gen_power_range_set_t

(C++ 类), 519
esp_ble_mesh_state_change_gen_power_range_set_t::range_max (C++ 成员), 519
esp_ble_mesh_state_change_gen_power_range_set_t::range_min (C++ 成员), 519
esp_ble_mesh_state_change_gen_user_property_set_t (C++ 类), 520
esp_ble_mesh_state_change_gen_user_property_set_t::ble_id (C++ 成员), 520
esp_ble_mesh_state_change_gen_user_property_set_t::value (C++ 成员), 520
esp_ble_mesh_state_change_light_ctl_default_set_t (C++ 类), 583
esp_ble_mesh_state_change_light_ctl_default_set_t::blue_uv (C++ 成员), 584
esp_ble_mesh_state_change_light_ctl_default_set_t::lightness (C++ 成员), 583
esp_ble_mesh_state_change_light_ctl_default_set_t::temperature (C++ 成员), 583
esp_ble_mesh_state_change_light_ctl_set_t (C++ 类), 583
esp_ble_mesh_state_change_light_ctl_set_t::delta_uv (C++ 成员), 583
esp_ble_mesh_state_change_light_ctl_set_t::lightness (C++ 成员), 583
esp_ble_mesh_state_change_light_ctl_set_t::temperature (C++ 成员), 583
esp_ble_mesh_state_change_light_ctl_temperature_range_set_t (C++ 类), 583
esp_ble_mesh_state_change_light_ctl_temperature_range_set_t::range_max (C++ 成员), 583
esp_ble_mesh_state_change_light_ctl_temperature_range_set_t::range_min (C++ 成员), 583
esp_ble_mesh_state_change_light_ctl_temperature_set_t (C++ 类), 583
esp_ble_mesh_state_change_light_ctl_temperature_set_t::delta_level (C++ 成员), 583
esp_ble_mesh_state_change_light_ctl_temperature_set_t::temperature (C++ 成员), 583
esp_ble_mesh_state_change_light_hsl_default_set_t (C++ 类), 584
esp_ble_mesh_state_change_light_hsl_default_set_t::hue (C++ 成员), 584
esp_ble_mesh_state_change_light_hsl_default_set_t::lightness (C++ 成员), 584
esp_ble_mesh_state_change_light_hsl_default_set_t::saturation (C++ 成员), 584
esp_ble_mesh_state_change_light_hsl_hue_set_t (C++ 类), 584
esp_ble_mesh_state_change_light_hsl_hue_set_t::hue (C++ 成员), 584
esp_ble_mesh_state_change_light_hsl_range_set_t (C++ 类), 585
esp_ble_mesh_state_change_light_hsl_range_set_t::hue_range_max (C++ 成员), 585
esp_ble_mesh_state_change_light_hsl_range_set_t::hue_range_min (C++ 成员), 585
esp_ble_mesh_state_change_light_hsl_range_set_t::saturation_range_max (C++ 成员), 585
esp_ble_mesh_state_change_light_hsl_range_set_t::saturation_range_min (C++ 成员), 585
esp_ble_mesh_state_change_light_hsl_saturation_set_t (C++ 类), 584
esp_ble_mesh_state_change_light_hsl_saturation_set_t::saturation (C++ 成员), 584
esp_ble_mesh_state_change_light_hsl_set_t (C++ 类), 584
esp_ble_mesh_state_change_light_hsl_set_t::hue (C++ 成员), 584
esp_ble_mesh_state_change_light_hsl_set_t::lightness (C++ 成员), 584
esp_ble_mesh_state_change_light_hsl_set_t::saturation (C++ 成员), 584
esp_ble_mesh_state_change_light_lc_light_onoff_set_t (C++ 类), 586
esp_ble_mesh_state_change_light_lc_light_onoff_set_t::onoff (C++ 成员), 586
esp_ble_mesh_state_change_light_lc_mode_set_t (C++ 类), 586
esp_ble_mesh_state_change_light_lc_mode_set_t::mode (C++ 成员), 586
esp_ble_mesh_state_change_light_lc_om_set_t (C++ 类), 586
esp_ble_mesh_state_change_light_lc_om_set_t::mode

(C++ 成员), 586
esp_ble_mesh_state_change_light_lc_property_set_t (C++ 类), 586
esp_ble_mesh_state_change_light_lc_property_set_t::property_id (C++ 成员), 587
esp_ble_mesh_state_change_light_lc_property_set_t::property_value (C++ 成员), 587
esp_ble_mesh_state_change_light_lightness_default_set_t (C++ 类), 582
esp_ble_mesh_state_change_light_lightness_default_set_t::lightness (C++ 成员), 582
esp_ble_mesh_state_change_light_lightness_linear_set_t (C++ 类), 582
esp_ble_mesh_state_change_light_lightness_linear_set_t::lightness (C++ 成员), 582
esp_ble_mesh_state_change_light_lightness_range_set_t (C++ 类), 582
esp_ble_mesh_state_change_light_lightness_range_set_t::range_max (C++ 成员), 582
esp_ble_mesh_state_change_light_lightness_range_set_t::range_min (C++ 成员), 582
esp_ble_mesh_state_change_light_lightness_set_t (C++ 类), 582
esp_ble_mesh_state_change_light_lightness_set_t::lightness (C++ 成员), 582
esp_ble_mesh_state_change_light_xyl_default_set_t (C++ 类), 585
esp_ble_mesh_state_change_light_xyl_default_set_t::lightness (C++ 成员), 585
esp_ble_mesh_state_change_light_xyl_default_set_t::tid (C++ 成员), 585
esp_ble_mesh_state_change_light_xyl_default_set_t::y (C++ 成员), 585
esp_ble_mesh_state_change_light_xyl_range_set_t (C++ 类), 585
esp_ble_mesh_state_change_light_xyl_range_set_t::y0 (C++ 成员), 586
esp_ble_mesh_state_change_light_xyl_range_set_t::y1 (C++ 成员), 586
esp_ble_mesh_state_change_light_xyl_range_set_t::y_range_max (C++ 成员), 586
esp_ble_mesh_state_change_light_xyl_range_set_t::y_range_min (C++ 成员), 586
esp_ble_mesh_state_change_light_xyl_set_t (C++ 类), 585
esp_ble_mesh_state_change_light_xyl_set_t::lightness (C++ 成员), 585
esp_ble_mesh_state_change_light_xyl_set_t::tid (C++ 成员), 585
esp_ble_mesh_state_change_light_xyl_set_t::x (C++ 成员), 585
esp_ble_mesh_state_change_light_xyl_set_t::y (C++ 成员), 585
esp_ble_mesh_state_change_scene_delete_t (C++ 类), 641
esp_ble_mesh_state_change_scene_delete_t::scene_number (C++ 成员), 642
esp_ble_mesh_state_change_scene_recall_t (C++ 类), 641
esp_ble_mesh_state_change_scene_recall_t::scene_number (C++ 成员), 641
esp_ble_mesh_state_change_scene_store_t (C++ 类), 641
esp_ble_mesh_state_change_scene_store_t::scene_number (C++ 成员), 641
esp_ble_mesh_state_change_scheduler_act_set_t (C++ 类), 642
esp_ble_mesh_state_change_scheduler_act_set_t::action (C++ 成员), 642
esp_ble_mesh_state_change_scheduler_act_set_t::day (C++ 成员), 642
esp_ble_mesh_state_change_scheduler_act_set_t::day_of_week (C++ 成员), 642
esp_ble_mesh_state_change_scheduler_act_set_t::hour (C++ 成员), 642
esp_ble_mesh_state_change_scheduler_act_set_t::index (C++ 成员), 642
esp_ble_mesh_state_change_scheduler_act_set_t::minute (C++ 成员), 642
esp_ble_mesh_state_change_scheduler_act_set_t::month (C++ 成员), 642
esp_ble_mesh_state_change_scheduler_act_set_t::scene_number (C++ 成员), 642
esp_ble_mesh_state_change_scheduler_act_set_t::second (C++ 成员), 642
esp_ble_mesh_state_change_scheduler_act_set_t::trans_time

(C++ 成员), 642
esp_ble_mesh_state_change_scheduler_act_set_t::year
(C++ 成员), 642
esp_ble_mesh_state_change_sensor_cadence_set_t
(C++ 类), 614
esp_ble_mesh_state_change_sensor_cadence_set_t::fast_cadence_high
(C++ 成员), 615
esp_ble_mesh_state_change_sensor_cadence_set_t::fast_cadence_low
(C++ 成员), 615
esp_ble_mesh_state_change_sensor_cadence_set_t::min_interval
(C++ 成员), 615
esp_ble_mesh_state_change_sensor_cadence_set_t::period_divisor
(C++ 成员), 614
esp_ble_mesh_state_change_sensor_cadence_set_t::property_id
(C++ 成员), 614
esp_ble_mesh_state_change_sensor_cadence_set_t::trigger_delta_down
(C++ 成员), 615
esp_ble_mesh_state_change_sensor_cadence_set_t::trigger_delta_up
(C++ 成员), 615
esp_ble_mesh_state_change_sensor_cadence_set_t::trigger_type
(C++ 成员), 614
esp_ble_mesh_state_change_sensor_setting_set_t
(C++ 类), 615
esp_ble_mesh_state_change_sensor_setting_set_t::property_id
(C++ 成员), 615
esp_ble_mesh_state_change_sensor_setting_set_t::setting_property_id
(C++ 成员), 615
esp_ble_mesh_state_change_sensor_setting_set_t::setting_raw
(C++ 成员), 615
esp_ble_mesh_state_change_sensor_status_t
(C++ 类), 587
esp_ble_mesh_state_change_sensor_status_t::ambient_luxlevel
(C++ 成员), 587
esp_ble_mesh_state_change_sensor_status_t::occupancy
(C++ 成员), 587
esp_ble_mesh_state_change_sensor_status_t::property_id
(C++ 成员), 587
esp_ble_mesh_state_change_sensor_status_t::setting_property_id
(C++ 成员), 587
esp_ble_mesh_state_change_sensor_status_t::status
(C++ 成员), 587
esp_ble_mesh_state_change_tai_utc_delta_set_t
(C++ 类), 641
esp_ble_mesh_state_change_tai_utc_delta_set_t::tai_delta_c
(C++ 成员), 641
esp_ble_mesh_state_change_tai_utc_delta_set_t::tai_utc_del
(C++ 成员), 641
esp_ble_mesh_state_change_time_role_set_t
(C++ 类), 641
esp_ble_mesh_state_change_time_role_set_t::time_role
(C++ 成员), 641
esp_ble_mesh_state_change_time_set_t (C++
类), 640
esp_ble_mesh_state_change_time_set_t::subsecond
(C++ 成员), 640
esp_ble_mesh_state_change_time_set_t::tai_seconds
(C++ 成员), 640
esp_ble_mesh_state_change_time_set_t::tai_utc_delta_curr
(C++ 成员), 640
esp_ble_mesh_state_change_time_set_t::time_authority
(C++ 成员), 640
esp_ble_mesh_state_change_time_set_t::time_zone_offset_cur
(C++ 成员), 640
esp_ble_mesh_state_change_time_set_t::uncertainty
(C++ 成员), 640
esp_ble_mesh_state_change_time_status_t
(C++ 类), 640
esp_ble_mesh_state_change_time_status_t::subsecond
(C++ 成员), 640
esp_ble_mesh_state_change_time_status_t::tai_seconds
(C++ 成员), 640
esp_ble_mesh_state_change_time_status_t::tai_utc_delta_cur
(C++ 成员), 640
esp_ble_mesh_state_change_time_status_t::time_authority
(C++ 成员), 640
esp_ble_mesh_state_change_time_status_t::time_zone_offset_
(C++ 成员), 640
esp_ble_mesh_state_change_time_status_t::uncertainty
(C++ 成员), 640
esp_ble_mesh_state_change_time_zone_set_t
(C++ 类), 640
esp_ble_mesh_state_change_time_zone_set_t::tai_zone_change
(C++ 成员), 641
esp_ble_mesh_state_change_time_zone_set_t::time_zone_offse

(C++ 成员), 641
esp_ble_mesh_state_transition_t (C++ 类), 409
esp_ble_mesh_state_transition_t::BLE_MESH_ATOMIC_DEFINE (C++ 函数), 409
esp_ble_mesh_state_transition_t::counter (C++ 成员), 410
esp_ble_mesh_state_transition_t::delay (C++ 成员), 409
esp_ble_mesh_state_transition_t::just_started (C++ 成员), 409
esp_ble_mesh_state_transition_t::quo_tt (C++ 成员), 410
esp_ble_mesh_state_transition_t::remain_time (C++ 成员), 409
esp_ble_mesh_state_transition_t::start_timestamp (C++ 成员), 410
esp_ble_mesh_state_transition_t::timer (C++ 成员), 410
esp_ble_mesh_state_transition_t::total_duration (C++ 成员), 410
esp_ble_mesh_state_transition_t::trans_time (C++ 成员), 409
ESP_BLE_MESH_STATIC_OOB (C++ 枚举子), 432
ESP_BLE_MESH_STOP_BLE_ADVERTISING_COMP_EVT (C++ 枚举子), 438
ESP_BLE_MESH_SUPPLY_VOLTAGE_TOO_HIGH_ERROR (C 宏), 543
ESP_BLE_MESH_SUPPLY_VOLTAGE_TOO_HIGH_WARNING (C 宏), 543
ESP_BLE_MESH_SUPPLY_VOLTAGE_TOO_LOW_ERROR (C 宏), 543
ESP_BLE_MESH_SUPPLY_VOLTAGE_TOO_LOW_WARNING (C 宏), 543
ESP_BLE_MESH_TAI_OF_DELTA_CHANGE_LEN (C 宏), 649
ESP_BLE_MESH_TAI_OF_ZONE_CHANGE_LEN (C 宏), 649
ESP_BLE_MESH_TAI_SECONDS_LEN (C 宏), 649
ESP_BLE_MESH_TAI_UTC_DELTA_MAX_VALUE (C 宏), 649
esp_ble_mesh_tai_utc_delta_set_t (C++ 类), 629
esp_ble_mesh_tai_utc_delta_set_t::padding (C++ 成员), 629
esp_ble_mesh_tai_utc_delta_set_t::tai_delta_change (C++ 成员), 629
esp_ble_mesh_tai_utc_delta_set_t::tai_utc_delta_new (C++ 成员), 629
esp_ble_mesh_tai_utc_delta_status_cb_t (C++ 类), 632
esp_ble_mesh_tai_utc_delta_status_cb_t::padding_1 (C++ 成员), 632
esp_ble_mesh_tai_utc_delta_status_cb_t::padding_2 (C++ 成员), 632
esp_ble_mesh_tai_utc_delta_status_cb_t::tai_delta_change (C++ 成员), 632
esp_ble_mesh_tai_utc_delta_status_cb_t::tai_utc_delta_curr (C++ 成员), 632
esp_ble_mesh_tai_utc_delta_status_cb_t::tai_utc_delta_new (C++ 成员), 632
ESP_BLE_MESH_TAMPER_ERROR (C 宏), 544
ESP_BLE_MESH_TAMPER_WARNING (C 宏), 544
ESP_BLE_MESH_TIME_AUTHORITY (C 宏), 651
ESP_BLE_MESH_TIME_CLINET (C 宏), 651
ESP_BLE_MESH_TIME_NONE (C 宏), 651
ESP_BLE_MESH_TIME_RELAY (C 宏), 651
esp_ble_mesh_time_role_set_t (C++ 类), 629
esp_ble_mesh_time_role_set_t::time_role (C++ 成员), 629
esp_ble_mesh_time_role_status_cb_t (C++ 类), 632
esp_ble_mesh_time_role_status_cb_t::time_role (C++ 成员), 632
esp_ble_mesh_time_scene_client_cb_event_t (C++ 类型), 652
esp_ble_mesh_time_scene_client_cb_param_t (C++ 类), 634
esp_ble_mesh_time_scene_client_cb_param_t::error_code (C++ 成员), 634
esp_ble_mesh_time_scene_client_cb_param_t::params (C++ 成员), 634
esp_ble_mesh_time_scene_client_cb_param_t::status_cb (C++ 成员), 634
esp_ble_mesh_time_scene_client_cb_t (C++ 类

型), 651
ESP_BLE_MESH_TIME_SCENE_CLIENT_EVT_MAX (C++ 枚举子), 652
esp_ble_mesh_time_scene_client_get_state (C++ 函数), 624
ESP_BLE_MESH_TIME_SCENE_CLIENT_GET_STATE_EVT (C++ 枚举子), 652
esp_ble_mesh_time_scene_client_get_state_t (C++ 类型), 624
esp_ble_mesh_time_scene_client_get_state_t::scheduler_mesh_time (C++ 成员), 625
ESP_BLE_MESH_TIME_SCENE_CLIENT_PUBLISH_EVT (C++ 枚举子), 652
esp_ble_mesh_time_scene_client_set_state (C++ 函数), 624
ESP_BLE_MESH_TIME_SCENE_CLIENT_SET_STATE_EVT (C++ 枚举子), 652
esp_ble_mesh_time_scene_client_set_state_t (C++ 类型), 625
esp_ble_mesh_time_scene_client_set_state_t::scene_delete (C++ 成员), 625
esp_ble_mesh_time_scene_client_set_state_t::scene_recall (C++ 成员), 625
esp_ble_mesh_time_scene_client_set_state_t::scene_store (C++ 成员), 625
esp_ble_mesh_time_scene_client_set_state_t::scheduler_mesh_time (C++ 成员), 625
esp_ble_mesh_time_scene_client_set_state_t::tai_utc_delta_set (C++ 成员), 625
esp_ble_mesh_time_scene_client_set_state_t::time_role (C++ 成员), 625
esp_ble_mesh_time_scene_client_set_state_t::time_set (C++ 成员), 625
esp_ble_mesh_time_scene_client_set_state_t::time_zone_set (C++ 成员), 625
esp_ble_mesh_time_scene_client_status_cb_t (C++ 类型), 625
esp_ble_mesh_time_scene_client_status_cb_t::scene_delete_status (C++ 成员), 626
esp_ble_mesh_time_scene_client_status_cb_t::scene_recall_status (C++ 成员), 626
esp_ble_mesh_time_scene_client_status_cb_t::scene_status (C++ 成员), 626
esp_ble_mesh_time_scene_client_status_cb_t::scene_store_status (C++ 成员), 626
esp_ble_mesh_time_scene_client_status_cb_t::scheduler_mesh_time (C++ 成员), 626
esp_ble_mesh_time_scene_client_status_cb_t::scheduler_status (C++ 成员), 626
esp_ble_mesh_time_scene_client_status_cb_t::tai_utc_delta_status (C++ 成员), 625
esp_ble_mesh_time_scene_client_status_cb_t::time_role_status (C++ 成员), 626
esp_ble_mesh_time_scene_client_status_cb_t::time_status (C++ 成员), 625
esp_ble_mesh_time_scene_client_status_cb_t::time_zone_status (C++ 成员), 625
ESP_BLE_MESH_TIME_SCENE_CLIENT_TIMEOUT_EVT (C++ 枚举子), 652
esp_ble_mesh_time_scene_message_opcode_t (C++ 类型), 432
esp_ble_mesh_time_scene_server_cb_event_t (C++ 类型), 652
esp_ble_mesh_time_scene_server_cb_param_t (C++ 类), 646
esp_ble_mesh_time_scene_server_cb_param_t::ctx (C++ 成员), 646
esp_ble_mesh_time_scene_server_cb_param_t::model (C++ 成员), 646
esp_ble_mesh_time_scene_server_cb_param_t::value (C++ 成员), 646
esp_ble_mesh_time_scene_server_cb_t (C++ 类型), 651
esp_ble_mesh_time_scene_server_cb_value_t (C++ 类型), 627
esp_ble_mesh_time_scene_server_cb_value_t::get (C++ 成员), 628
esp_ble_mesh_time_scene_server_cb_value_t::set (C++ 成员), 628
esp_ble_mesh_time_scene_server_cb_value_t::state_change (C++ 成员), 628
esp_ble_mesh_time_scene_server_cb_value_t::status (C++ 成员), 628
ESP_BLE_MESH_TIME_SCENE_SERVER_EVT_MAX (C++ 枚举子), 652
ESP_BLE_MESH_TIME_SCENE_SERVER_RECV_GET_MSG_EVT (C++ 枚举子), 652
esp_ble_mesh_time_scene_server_recv_get_msg_t

(C++ 类型), 626
esp_ble_mesh_time_scene_server_recv_get_msg_t::scheduler_act (C++ 成员), 627
ESP_BLE_MESH_TIME_SCENE_SERVER_RECV_SET_MSG_EVT (C++ 枚举子), 652
esp_ble_mesh_time_scene_server_recv_set_msg_t (C++ 类型), 627
esp_ble_mesh_time_scene_server_recv_set_msg_t::scene_delete (C++ 成员), 627
esp_ble_mesh_time_scene_server_recv_set_msg_t::scene_recall (C++ 成员), 627
esp_ble_mesh_time_scene_server_recv_set_msg_t::scene_store (C++ 成员), 627
esp_ble_mesh_time_scene_server_recv_set_msg_t::scheduler_act (C++ 成员), 627
esp_ble_mesh_time_scene_server_recv_set_msg_t::tai_utc_delta (C++ 成员), 627
esp_ble_mesh_time_scene_server_recv_set_msg_t::time (C++ 成员), 627
esp_ble_mesh_time_scene_server_recv_set_msg_t::time_role (C++ 成员), 627
esp_ble_mesh_time_scene_server_recv_set_msg_t::time_zone_offset (C++ 成员), 627
ESP_BLE_MESH_TIME_SCENE_SERVER_RECV_STATUS_MSG_EVT (C++ 枚举子), 652
esp_ble_mesh_time_scene_server_recv_status_msg_t (C++ 类型), 627
esp_ble_mesh_time_scene_server_recv_status_msg_t::time_status (C++ 成员), 627
ESP_BLE_MESH_TIME_SCENE_SERVER_STATE_CHANGE_EVT (C++ 枚举子), 652
esp_ble_mesh_time_scene_server_state_change_t (C++ 类型), 626
esp_ble_mesh_time_scene_server_state_change_t::scene_delete (C++ 成员), 626
esp_ble_mesh_time_scene_server_state_change_t::scene_recall (C++ 成员), 626
esp_ble_mesh_time_scene_server_state_change_t::scene_store (C++ 成员), 626
esp_ble_mesh_time_scene_server_state_change_t::scheduler_act (C++ 成员), 626
esp_ble_mesh_time_scene_server_state_change_t::tai_utc_delta (C++ 成员), 626
esp_ble_mesh_time_scene_server_state_change_t::time (C++ 成员), 626
esp_ble_mesh_time_scene_server_state_change_t::time_role (C++ 成员), 626
esp_ble_mesh_time_scene_server_state_change_t::time_set (C++ 成员), 626
esp_ble_mesh_time_scene_server_state_change_t::time_status (C++ 成员), 626
esp_ble_mesh_time_scene_server_state_change_t::time_zone_offset (C++ 成员), 626
esp_ble_mesh_time_set_t (C++ 类), 628
esp_ble_mesh_time_set_t::sub_second (C++ 成员), 628
esp_ble_mesh_time_set_t::tai_seconds (C++ 成员), 628
esp_ble_mesh_time_set_t::tai_utc_delta (C++ 成员), 628
esp_ble_mesh_time_set_t::time_authority (C++ 成员), 628
esp_ble_mesh_time_set_t::time_zone_offset (C++ 成员), 628
esp_ble_mesh_time_set_t::uncertainty (C++ 成员), 628
esp_ble_mesh_time_setup_srv_t (C++ 类), 635
esp_ble_mesh_time_setup_srv_t::model (C++ 成员), 635
esp_ble_mesh_time_setup_srv_t::rsp_ctrl (C++ 成员), 635
esp_ble_mesh_time_setup_srv_t::state (C++ 成员), 635
esp_ble_mesh_time_srv_t (C++ 类), 635
esp_ble_mesh_time_srv_t::model (C++ 成员), 635
esp_ble_mesh_time_srv_t::rsp_ctrl (C++ 成员), 635
esp_ble_mesh_time_srv_t::state (C++ 成员), 635
esp_ble_mesh_time_state_t (C++ 类), 634
esp_ble_mesh_time_state_t::subsecond (C++ 成员), 634
esp_ble_mesh_time_state_t::tai_delta_change (C++ 成员), 635
esp_ble_mesh_time_state_t::tai_seconds (C++

成员), 634
esp_ble_mesh_time_state_t::tai_utc_delta_curr (C++ 成员), 635
esp_ble_mesh_time_state_t::tai_utc_delta_new (C++ 成员), 635
esp_ble_mesh_time_state_t::tai_zone_change (C++ 成员), 635
esp_ble_mesh_time_state_t::time (C++ 成员), 635
esp_ble_mesh_time_state_t::time_authority (C++ 成员), 635
esp_ble_mesh_time_state_t::time_role (C++ 成员), 635
esp_ble_mesh_time_state_t::time_zone_offset_curr (C++ 成员), 634
esp_ble_mesh_time_state_t::time_zone_offset_new (C++ 成员), 634
esp_ble_mesh_time_state_t::uncertainty (C++ 成员), 634
esp_ble_mesh_time_status_cb_t (C++ 类), 631
esp_ble_mesh_time_status_cb_t::sub_second (C++ 成员), 631
esp_ble_mesh_time_status_cb_t::tai_seconds (C++ 成员), 631
esp_ble_mesh_time_status_cb_t::tai_utc_delta (C++ 成员), 631
esp_ble_mesh_time_status_cb_t::time_authority (C++ 成员), 631
esp_ble_mesh_time_status_cb_t::time_zone_offset (C++ 成员), 631
esp_ble_mesh_time_status_cb_t::uncertainty (C++ 成员), 631
esp_ble_mesh_time_zone_set_t (C++ 类), 628
esp_ble_mesh_time_zone_set_t::tai_zone_change (C++ 成员), 629
esp_ble_mesh_time_zone_set_t::time_zone_offset_new (C++ 成员), 629
esp_ble_mesh_time_zone_status_cb_t (C++ 类), 631
esp_ble_mesh_time_zone_status_cb_t::tai_zone_change (C++ 成员), 632
esp_ble_mesh_time_zone_status_cb_t::time_zone_offset_curr (C++ 成员), 631
esp_ble_mesh_time_zone_status_cb_t::time_zone_offset_new (C++ 成员), 632
ESP_BLE_MESH_TRANSMIT (C 宏), 413
ESP_BLE_MESH_TWIST (C++ 枚举子), 433
ESP_BLE_MESH_TYPE_COMPLETE_CB (C++ 枚举子), 432
ESP_BLE_MESH_TYPE_INTPUT_CB (C++ 枚举子), 432
ESP_BLE_MESH_TYPE_LINK_CLOSE_CB (C++ 枚举子), 432
ESP_BLE_MESH_TYPE_LINK_OPEN_CB (C++ 枚举子), 432
ESP_BLE_MESH_TYPE_OUTPUT_NUM_CB (C++ 枚举子), 432
ESP_BLE_MESH_TYPE_OUTPUT_STR_CB (C++ 枚举子), 432
ESP_BLE_MESH_TYPE_PROV_CB (C++ 枚举子), 432
ESP_BLE_MESH_TYPE_RESET_CB (C++ 枚举子), 432
ESP_BLE_MESH_UNKNOWN_TAI_DELTA_CHANGE (C 宏), 649
ESP_BLE_MESH_UNKNOWN_TAI_SECONDS (C 宏), 649
ESP_BLE_MESH_UNKNOWN_TAI_ZONE_CHANGE (C 宏), 649
esp_ble_mesh_unprov_dev_add_t (C++ 类), 404
esp_ble_mesh_unprov_dev_add_t::addr (C++ 成员), 404
esp_ble_mesh_unprov_dev_add_t::addr_type (C++ 成员), 404
esp_ble_mesh_unprov_dev_add_t::bearer (C++ 成员), 405
esp_ble_mesh_unprov_dev_add_t::oob_info (C++ 成员), 404
esp_ble_mesh_unprov_dev_add_t::uuid (C++ 成员), 404
ESP_BLE_MESH_VENDOR_MODEL (C 宏), 414
ESP_BLE_MESH_VIBRATE (C++ 枚举子), 433
ESP_BLE_MESH_VIBRATION_ERROR (C 宏), 543
ESP_BLE_MESH_VIBRATION_WARNING (C 宏), 543
ESP_BLE_ONLY_ACCEPT_SPECIFIED_AUTH_DISABLE (C 宏), 187
ESP_BLE_ONLY_ACCEPT_SPECIFIED_AUTH_ENABLE (C 宏), 187

ESP_BLE_OOB_DISABLE (C 宏), 187
ESP_BLE_OOB_ENABLE (C 宏), 187
esp_ble_oob_req_reply (C++ 函数), 170
esp_ble_passkey_reply (C++ 函数), 169
esp_ble_pcsrk_keys_t (C++ 类), 183
esp_ble_pcsrk_keys_t::counter (C++ 成员), 183
esp_ble_pcsrk_keys_t::csrk (C++ 成员), 183
esp_ble_pcsrk_keys_t::sec_level (C++ 成员), 183
esp_ble_penc_keys_t (C++ 类), 182
esp_ble_penc_keys_t::ediv (C++ 成员), 183
esp_ble_penc_keys_t::key_size (C++ 成员), 183
esp_ble_penc_keys_t::ltk (C++ 成员), 183
esp_ble_penc_keys_t::rand (C++ 成员), 183
esp_ble_penc_keys_t::sec_level (C++ 成员), 183
esp_ble_pid_keys_t (C++ 类), 183
esp_ble_pid_keys_t::addr_type (C++ 成员), 183
esp_ble_pid_keys_t::irk (C++ 成员), 183
esp_ble_pid_keys_t::static_addr (C++ 成员), 183
esp_ble_pkt_data_length_params_t (C++ 类), 182
esp_ble_pkt_data_length_params_t::rx_len (C++ 成员), 182
esp_ble_pkt_data_length_params_t::tx_len (C++ 成员), 182
esp_ble_power_type_t (C++ 类型), 152
ESP_BLE_PWR_TYPE_ADV (C++ 枚举子), 153
ESP_BLE_PWR_TYPE_CONN_HDL0 (C++ 枚举子), 152
ESP_BLE_PWR_TYPE_CONN_HDL1 (C++ 枚举子), 152
ESP_BLE_PWR_TYPE_CONN_HDL2 (C++ 枚举子), 152
ESP_BLE_PWR_TYPE_CONN_HDL3 (C++ 枚举子), 152
ESP_BLE_PWR_TYPE_CONN_HDL4 (C++ 枚举子), 153
ESP_BLE_PWR_TYPE_CONN_HDL5 (C++ 枚举子), 153
ESP_BLE_PWR_TYPE_CONN_HDL6 (C++ 枚举子), 153
ESP_BLE_PWR_TYPE_CONN_HDL7 (C++ 枚举子), 153
ESP_BLE_PWR_TYPE_CONN_HDL8 (C++ 枚举子), 153
ESP_BLE_PWR_TYPE_DEFAULT (C++ 枚举子), 153
ESP_BLE_PWR_TYPE_NUM (C++ 枚举子), 153
ESP_BLE_PWR_TYPE_SCAN (C++ 枚举子), 153
esp_ble_remove_bond_device (C++ 函数), 169
esp_ble_resolve_adv_data (C++ 函数), 166
ESP_BLE_SCA_100PPM (C++ 枚举子), 152
ESP_BLE_SCA_150PPM (C++ 枚举子), 152
ESP_BLE_SCA_20PPM (C++ 枚举子), 152
ESP_BLE_SCA_250PPM (C++ 枚举子), 151
ESP_BLE_SCA_30PPM (C++ 枚举子), 152
ESP_BLE_SCA_500PPM (C++ 枚举子), 151
ESP_BLE_SCA_50PPM (C++ 枚举子), 152
ESP_BLE_SCA_75PPM (C++ 枚举子), 152
esp_ble_scan_dupilcate_list_flush (C++ 函数), 149
esp_ble_scan_duplicate_t (C++ 类型), 196
esp_ble_scan_filter_t (C++ 类型), 195
ESP_BLE_SCAN_PARAM_UNDEF (C 宏), 156
esp_ble_scan_params_t (C++ 类), 181
esp_ble_scan_params_t::own_addr_type (C++ 成员), 181
esp_ble_scan_params_t::scan_duplicate (C++ 成员), 181
esp_ble_scan_params_t::scan_filter_policy (C++ 成员), 181
esp_ble_scan_params_t::scan_interval (C++ 成员), 181
esp_ble_scan_params_t::scan_type (C++ 成员), 181
esp_ble_scan_params_t::scan_window (C++ 成员), 181
ESP_BLE_SCAN_RSP_DATA_LEN_MAX (C 宏), 190
esp_ble_scan_type_t (C++ 类型), 195
esp_ble_sec_act_t (C++ 类型), 194
ESP_BLE_SEC_ENCRYPT (C++ 枚举子), 194
ESP_BLE_SEC_ENCRYPT_MITM (C++ 枚举子), 194
ESP_BLE_SEC_ENCRYPT_NO_MITM (C++ 枚举子), 194
esp_ble_sec_key_notif_t (C++ 类), 184
esp_ble_sec_key_notif_t::bd_addr (C++ 成员), 184
esp_ble_sec_key_notif_t::passkey (C++ 成员), 184
esp_ble_sec_req_t (C++ 类), 184
esp_ble_sec_req_t::bd_addr (C++ 成员), 185
esp_ble_sec_t (C++ 类型), 172
esp_ble_sec_t::auth_cmpl (C++ 成员), 172

esp_ble_sec_t::ble_id_keys (C++ 成员), 172
esp_ble_sec_t::ble_key (C++ 成员), 172
esp_ble_sec_t::ble_req (C++ 成员), 172
esp_ble_sec_t::key_notif (C++ 成员), 172
esp_ble_set_encryption (C++ 函数), 169
ESP_BLE_SM_AUTHEN_REQ_MODE (C++ 枚举子), 195
ESP_BLE_SM_CLEAR_STATIC_PASSKEY (C++ 枚举子), 195
ESP_BLE_SM_IOCAP_MODE (C++ 枚举子), 195
ESP_BLE_SM_MAX_KEY_SIZE (C++ 枚举子), 195
ESP_BLE_SM_MAX_PARAM (C++ 枚举子), 195
ESP_BLE_SM_MIN_KEY_SIZE (C++ 枚举子), 195
ESP_BLE_SM_ONLY_ACCEPT_SPECIFIED_SEC_AUTH (C++ 枚举子), 195
ESP_BLE_SM_OOB_SUPPORT (C++ 枚举子), 195
esp_ble_sm_param_t (C++ 类型), 194
ESP_BLE_SM_PASSKEY (C++ 枚举子), 195
ESP_BLE_SM_SET_INIT_KEY (C++ 枚举子), 195
ESP_BLE_SM_SET_RSP_KEY (C++ 枚举子), 195
ESP_BLE_SM_SET_STATIC_PASSKEY (C++ 枚举子), 195
esp_ble_tx_power_get (C++ 函数), 144
esp_ble_tx_power_set (C++ 函数), 144
ESP_BLE_WHITELIST_ADD (C++ 枚举子), 197
ESP_BLE_WHITELIST_REMOVE (C++ 枚举子), 197
esp_ble_wl_addr_type_t (C++ 类型), 158
esp_ble_wl_opration_t (C++ 类型), 197
esp_bluedroid_deinit (C++ 函数), 159
esp_bluedroid_disable (C++ 函数), 159
esp_bluedroid_enable (C++ 函数), 159
esp_bluedroid_get_status (C++ 函数), 158
esp_bluedroid_init (C++ 函数), 159
ESP_BLUEDROID_STATUS_CHECK (C 宏), 155
ESP_BLUEDROID_STATUS_ENABLED (C++ 枚举子), 159
ESP_BLUEDROID_STATUS_INITIALIZED (C++ 枚举子), 159
esp_bluedroid_status_t (C++ 类型), 159
ESP_BLUEDROID_STATUS_UNINITIALIZED (C++ 枚举子), 159
esp_blufi_ap_record_t (C++ 类), 263
esp_blufi_ap_record_t::rssi (C++ 成员), 263
esp_blufi_ap_record_t::ssid (C++ 成员), 263
esp_blufi_callbacks_t (C++ 类), 263
esp_blufi_callbacks_t::checksum_func (C++ 成员), 263
esp_blufi_callbacks_t::decrypt_func (C++ 成员), 263
esp_blufi_callbacks_t::encrypt_func (C++ 成员), 263
esp_blufi_callbacks_t::event_cb (C++ 成员), 263
esp_blufi_callbacks_t::negotiate_data_handler (C++ 成员), 263
esp_blufi_cb_event_t (C++ 类型), 265
esp_blufi_cb_param_t (C++ 类型), 256
esp_blufi_cb_param_t::blufi_connect_evt_param (C++ 类), 257
esp_blufi_cb_param_t::blufi_connect_evt_param::conn_id (C++ 成员), 257
esp_blufi_cb_param_t::blufi_connect_evt_param::remote_bda (C++ 成员), 257
esp_blufi_cb_param_t::blufi_connect_evt_param::server_if (C++ 成员), 257
esp_blufi_cb_param_t::blufi_deinit_finish_evt_param (C++ 类), 257
esp_blufi_cb_param_t::blufi_deinit_finish_evt_param::state (C++ 成员), 257
esp_blufi_cb_param_t::blufi_disconnect_evt_param (C++ 类), 257
esp_blufi_cb_param_t::blufi_disconnect_evt_param::remote_b (C++ 成员), 258
esp_blufi_cb_param_t::blufi_get_error_evt_param (C++ 类), 258
esp_blufi_cb_param_t::blufi_get_error_evt_param::state (C++ 成员), 258
esp_blufi_cb_param_t::blufi_init_finish_evt_param (C++ 类), 258
esp_blufi_cb_param_t::blufi_init_finish_evt_param::state (C++ 成员), 258
esp_blufi_cb_param_t::blufi_recv_ca_evt_param (C++ 类), 258
esp_blufi_cb_param_t::blufi_recv_ca_evt_param::cert (C++ 成员), 258

esp_blufi_cb_param_t::blufi_recv_ca_evt_param::esp_blufi_cb_param_t::blufi_recv_softap_max_conn_num_evt_p
(C++ 成员), 258 (C++ 成员), 260
esp_blufi_cb_param_t::blufi_recv_client_cert_evt_param esp_blufi_cb_param_t::blufi_recv_softap_passwd_evt_param
(C++ 类), 258 (C++ 类), 260
esp_blufi_cb_param_t::blufi_recv_client_cert_evt_param::esp_blufi_cb_param_t::blufi_recv_softap_passwd_evt_param::
(C++ 成员), 258 (C++ 成员), 260
esp_blufi_cb_param_t::blufi_recv_client_cert_evt_param::esp_blufi_cb_param_t::blufi_recv_softap_passwd_evt_param::
(C++ 成员), 258 (C++ 成员), 260
esp_blufi_cb_param_t::blufi_recv_client_pkey_evt_param esp_blufi_cb_param_t::blufi_recv_softap_ssid_evt_param
(C++ 类), 258 (C++ 类), 260
esp_blufi_cb_param_t::blufi_recv_client_pkey_evt_param::esp_blufi_cb_param_t::blufi_recv_softap_ssid_evt_param::ss
(C++ 成员), 259 (C++ 成员), 260
esp_blufi_cb_param_t::blufi_recv_client_pkey_evt_param::esp_blufi_cb_param_t::blufi_recv_softap_ssid_evt_param::ss
(C++ 成员), 259 (C++ 成员), 260
esp_blufi_cb_param_t::blufi_recv_custom_data_evt_param esp_blufi_cb_param_t::blufi_recv_sta_bssid_evt_param
(C++ 类), 259 (C++ 类), 260
esp_blufi_cb_param_t::blufi_recv_custom_data_evt_param::esp_blufi_cb_param_t::blufi_recv_sta_bssid_evt_param::bssi
(C++ 成员), 259 (C++ 成员), 261
esp_blufi_cb_param_t::blufi_recv_custom_data_evt_param::esp_blufi_cb_param_t::blufi_recv_sta_passwd_evt_param
(C++ 成员), 259 (C++ 类), 261
esp_blufi_cb_param_t::blufi_recv_server_cert_evt_param esp_blufi_cb_param_t::blufi_recv_sta_passwd_evt_param::pas
(C++ 类), 259 (C++ 成员), 261
esp_blufi_cb_param_t::blufi_recv_server_cert_evt_param::esp_blufi_cb_param_t::blufi_recv_sta_passwd_evt_param::pas
(C++ 成员), 259 (C++ 成员), 261
esp_blufi_cb_param_t::blufi_recv_server_cert_evt_param::esp_blufi_cb_param_t::blufi_recv_sta_ssid_evt_param
(C++ 成员), 259 (C++ 类), 261
esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param esp_blufi_cb_param_t::blufi_recv_sta_ssid_evt_param::ssid
(C++ 类), 259 (C++ 成员), 261
esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::esp_blufi_cb_param_t::blufi_recv_sta_ssid_evt_param::ssid_
(C++ 成员), 259 (C++ 成员), 261
esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param::esp_blufi_cb_param_t::blufi_recv_username_evt_param
(C++ 成员), 259 (C++ 类), 261
esp_blufi_cb_param_t::blufi_recv_softap_auth_mode_evt_paesp_blufi_cb_param_t::blufi_recv_username_evt_param::name
(C++ 类), 259 (C++ 成员), 261
esp_blufi_cb_param_t::blufi_recv_softap_auth_mode_evt_paesp_blufi_cb_param_t::blufi_recv_username_evt_param::name_
(C++ 成员), 260 (C++ 成员), 261
esp_blufi_cb_param_t::blufi_recv_softap_channel_evt_paramesp_blufi_cb_param_t::blufi_set_wifi_mode_evt_param
(C++ 类), 260 (C++ 类), 261
esp_blufi_cb_param_t::blufi_recv_softap_channel_evt_paramesp_blufi_cb_param_t::blufi_set_wifi_mode_evt_param::op_mo
(C++ 成员), 260 (C++ 成员), 261
esp_blufi_cb_param_t::blufi_recv_softap_max_conn_num_evesp_blufi_cb_param_t::ca (C++ 成员), 257
(C++ 类), 260 esp_blufi_cb_param_t::client_cert (C++ 成

员), 257
esp_blufi_cb_param_t::client_pkey (C++ 成员), 257
esp_blufi_cb_param_t::connect (C++ 成员), 256
esp_blufi_cb_param_t::custom_data (C++ 成员), 257
esp_blufi_cb_param_t::deinit_finish (C++ 成员), 256
esp_blufi_cb_param_t::disconnect (C++ 成员), 256
esp_blufi_cb_param_t::init_finish (C++ 成员), 256
esp_blufi_cb_param_t::report_error (C++ 成员), 257
esp_blufi_cb_param_t::server_cert (C++ 成员), 257
esp_blufi_cb_param_t::server_pkey (C++ 成员), 257
esp_blufi_cb_param_t::softap_auth_mode (C++ 成员), 256
esp_blufi_cb_param_t::softap_channel (C++ 成员), 256
esp_blufi_cb_param_t::softap_max_conn_num (C++ 成员), 256
esp_blufi_cb_param_t::softap_passwd (C++ 成员), 256
esp_blufi_cb_param_t::softap_ssid (C++ 成员), 256
esp_blufi_cb_param_t::sta_bssid (C++ 成员), 256
esp_blufi_cb_param_t::sta_passwd (C++ 成员), 256
esp_blufi_cb_param_t::sta_ssid (C++ 成员), 256
esp_blufi_cb_param_t::username (C++ 成员), 256
esp_blufi_cb_param_t::wifi_mode (C++ 成员), 256
ESP_BLUFI_CHECKSUM_ERROR (C++ 枚举子), 266
esp_blufi_checksum_func_t (C++ 类型), 264
esp_blufi_close (C++ 函数), 255
ESP_BLUFI_DATA_FORMAT_ERROR (C++ 枚举子), 266
ESP_BLUFI_DECRYPT_ERROR (C++ 枚举子), 266
esp_blufi_decrypt_func_t (C++ 类型), 264
ESP_BLUFI_DEINIT_FAILED (C++ 枚举子), 266
ESP_BLUFI_DEINIT_OK (C++ 枚举子), 266
esp_blufi_deinit_state_t (C++ 类型), 266
ESP_BLUFI_DH_MALLOC_ERROR (C++ 枚举子), 266
ESP_BLUFI_DH_PARAM_ERROR (C++ 枚举子), 266
ESP_BLUFI_ENCRYPT_ERROR (C++ 枚举子), 266
esp_blufi_encrypt_func_t (C++ 类型), 264
esp_blufi_error_state_t (C++ 类型), 266
ESP_BLUFI_EVENT_BLE_CONNECT (C++ 枚举子), 265
ESP_BLUFI_EVENT_BLE_DISCONNECT (C++ 枚举子), 265
esp_blufi_event_cb_t (C++ 类型), 263
ESP_BLUFI_EVENT_DEAUTHENTICATE_STA (C++ 枚举子), 265
ESP_BLUFI_EVENT_DEINIT_FINISH (C++ 枚举子), 265
ESP_BLUFI_EVENT_GET_WIFI_LIST (C++ 枚举子), 265
ESP_BLUFI_EVENT_GET_WIFI_STATUS (C++ 枚举子), 265
ESP_BLUFI_EVENT_INIT_FINISH (C++ 枚举子), 265
ESP_BLUFI_EVENT_RECV_CA_CERT (C++ 枚举子), 265
ESP_BLUFI_EVENT_RECV_CLIENT_CERT (C++ 枚举子), 265
ESP_BLUFI_EVENT_RECV_CLIENT_PRIV_KEY (C++ 枚举子), 265
ESP_BLUFI_EVENT_RECV_CUSTOM_DATA (C++ 枚举子), 265
ESP_BLUFI_EVENT_RECV_SERVER_CERT (C++ 枚举子), 265
ESP_BLUFI_EVENT_RECV_SERVER_PRIV_KEY (C++ 枚举子), 265
ESP_BLUFI_EVENT_RECV_SLAVE_DISCONNECT_BLE (C++ 枚举子), 265
ESP_BLUFI_EVENT_RECV_SOFTAP_AUTH_MODE (C++ 枚举子), 265
ESP_BLUFI_EVENT_RECV_SOFTAP_CHANNEL (C++ 枚举子), 265
ESP_BLUFI_EVENT_RECV_SOFTAP_MAX_CONN_NUM

(C++ 枚举子), 265
ESP_BLUFI_EVENT_RECV_SOFTAP_PASSWD (C++ 枚举子), 265
ESP_BLUFI_EVENT_RECV_SOFTAP_SSID (C++ 枚举子), 265
ESP_BLUFI_EVENT_RECV_STA_BSSID (C++ 枚举子), 265
ESP_BLUFI_EVENT_RECV_STA_PASSWD (C++ 枚举子), 265
ESP_BLUFI_EVENT_RECV_STA_SSID (C++ 枚举子), 265
ESP_BLUFI_EVENT_RECV_USERNAME (C++ 枚举子), 265
ESP_BLUFI_EVENT_REPORT_ERROR (C++ 枚举子), 265
ESP_BLUFI_EVENT_REQ_CONNECT_TO_AP (C++ 枚举子), 265
ESP_BLUFI_EVENT_REQ_DISCONNECT_FROM_AP (C++ 枚举子), 265
ESP_BLUFI_EVENT_SET_WIFI_OPMODE (C++ 枚举子), 265
esp_blufi_extra_info_t (C++ 类), 262
esp_blufi_extra_info_t::softap_authmode (C++ 成员), 262
esp_blufi_extra_info_t::softap_authmode_set (C++ 成员), 262
esp_blufi_extra_info_t::softap_channel (C++ 成员), 262
esp_blufi_extra_info_t::softap_channel_set (C++ 成员), 263
esp_blufi_extra_info_t::softap_max_conn_num (C++ 成员), 262
esp_blufi_extra_info_t::softap_max_conn_num_set (C++ 成员), 262
esp_blufi_extra_info_t::softap_passwd (C++ 成员), 262
esp_blufi_extra_info_t::softap_passwd_len (C++ 成员), 262
esp_blufi_extra_info_t::softap_ssid (C++ 成员), 262
esp_blufi_extra_info_t::softap_ssid_len (C++ 成员), 262
esp_blufi_extra_info_t::sta_bssid (C++ 成员), 262
esp_blufi_extra_info_t::sta_bssid_set (C++ 成员), 262
esp_blufi_extra_info_t::sta_passwd (C++ 成员), 262
esp_blufi_extra_info_t::sta_passwd_len (C++ 成员), 262
esp_blufi_extra_info_t::sta_ssid (C++ 成员), 262
esp_blufi_extra_info_t::sta_ssid_len (C++ 成员), 262
esp_blufi_get_version (C++ 函数), 255
ESP_BLUFI_INIT_FAILED (C++ 枚举子), 266
ESP_BLUFI_INIT_OK (C++ 枚举子), 266
ESP_BLUFI_INIT_SECURITY_ERROR (C++ 枚举子), 266
esp_blufi_init_state_t (C++ 类型), 266
ESP_BLUFI_MAKE_PUBLIC_ERROR (C++ 枚举子), 266
esp_blufi_negotiate_data_handler_t (C++ 类型), 264
esp_blufi_profile_deinit (C++ 函数), 254
esp_blufi_profile_init (C++ 函数), 254
ESP_BLUFI_READ_PARAM_ERROR (C++ 枚举子), 266
esp_blufi_register_callbacks (C++ 函数), 254
esp_blufi_send_custom_data (C++ 函数), 255
esp_blufi_send_error_info (C++ 函数), 255
esp_blufi_send_wifi_conn_report (C++ 函数), 254
esp_blufi_send_wifi_list (C++ 函数), 255
ESP_BLUFI_SEQUENCE_ERROR (C++ 枚举子), 266
ESP_BLUFI_STA_CONN_FAIL (C++ 枚举子), 266
esp_blufi_sta_conn_state_t (C++ 类型), 265
ESP_BLUFI_STA_CONN_SUCCESS (C++ 枚举子), 266
esp_bredr_sco_datapath_set (C++ 函数), 145
esp_bredr_tx_power_get (C++ 函数), 145
esp_bredr_tx_power_set (C++ 函数), 144
ESP_BT_CLR_COD_SERVICE_CLASS (C++ 枚举子), 283
ESP_BT_COD_FORMAT_TYPE_1 (C 宏), 282
ESP_BT_COD_FORMAT_TYPE_BIT_MASK (C 宏), 282
ESP_BT_COD_FORMAT_TYPE_BIT_OFFSET (C 宏), 282

ESP_BT_COD_MAJOR_DEV_AV (C++ 枚举子), 285
ESP_BT_COD_MAJOR_DEV_BIT_MASK (C 宏), 282
ESP_BT_COD_MAJOR_DEV_BIT_OFFSET (C 宏), 282
ESP_BT_COD_MAJOR_DEV_COMPUTER (C++ 枚举子), 285
ESP_BT_COD_MAJOR_DEV_HEALTH (C++ 枚举子), 286
ESP_BT_COD_MAJOR_DEV_IMAGING (C++ 枚举子), 285
ESP_BT_COD_MAJOR_DEV_LAN_NAP (C++ 枚举子), 285
ESP_BT_COD_MAJOR_DEV_MISC (C++ 枚举子), 285
ESP_BT_COD_MAJOR_DEV_PERIPHERAL (C++ 枚举子), 285
ESP_BT_COD_MAJOR_DEV_PHONE (C++ 枚举子), 285
esp_bt_cod_major_dev_t (C++ 类型), 285
ESP_BT_COD_MAJOR_DEV_TOY (C++ 枚举子), 286
ESP_BT_COD_MAJOR_DEV_UNCATEGORIZED (C++ 枚举子), 286
ESP_BT_COD_MAJOR_DEV_WEARABLE (C++ 枚举子), 286
ESP_BT_COD_MINOR_DEV_BIT_MASK (C 宏), 282
ESP_BT_COD_MINOR_DEV_BIT_OFFSET (C 宏), 282
esp_bt_cod_mode_t (C++ 类型), 283
ESP_BT_COD_SRVC_AUDIO (C++ 枚举子), 285
ESP_BT_COD_SRVC_BIT_MASK (C 宏), 282
ESP_BT_COD_SRVC_BIT_OFFSET (C 宏), 282
ESP_BT_COD_SRVC_CAPTURING (C++ 枚举子), 284
ESP_BT_COD_SRVC_INFORMATION (C++ 枚举子), 285
ESP_BT_COD_SRVC_LMTD_DISCOVER (C++ 枚举子), 284
ESP_BT_COD_SRVC_NETWORKING (C++ 枚举子), 284
ESP_BT_COD_SRVC_NONE (C++ 枚举子), 284
ESP_BT_COD_SRVC_OBJ_TRANSFER (C++ 枚举子), 285
ESP_BT_COD_SRVC_POSITIONING (C++ 枚举子), 284
ESP_BT_COD_SRVC_RENDERING (C++ 枚举子), 284
esp_bt_cod_srvc_t (C++ 类型), 284
ESP_BT_COD_SRVC_TELEPHONY (C++ 枚举子), 285
esp_bt_cod_t (C++ 类), 279
esp_bt_cod_t::major (C++ 成员), 279
esp_bt_cod_t::minor (C++ 成员), 279
esp_bt_cod_t::reserved_2 (C++ 成员), 279
esp_bt_cod_t::reserved_8 (C++ 成员), 279
esp_bt_cod_t::service (C++ 成员), 279
ESP_BT_CONNECTABLE (C++ 枚举子), 283
esp_bt_connection_mode_t (C++ 类型), 283
ESP_BT_CONTROLLER_CONFIG_MAGIC_VAL (C 宏), 151
esp_bt_controller_config_t (C++ 类), 149
esp_bt_controller_config_t::auto_latency (C++ 成员), 150
esp_bt_controller_config_t::ble_max_conn (C++ 成员), 150
esp_bt_controller_config_t::ble_sca (C++ 成员), 150
esp_bt_controller_config_t::bt_max_acl_conn (C++ 成员), 150
esp_bt_controller_config_t::bt_max_sync_conn (C++ 成员), 150
esp_bt_controller_config_t::bt_sco_datapath (C++ 成员), 150
esp_bt_controller_config_t::controller_debug_flag (C++ 成员), 150
esp_bt_controller_config_t::controller_task_prio (C++ 成员), 149
esp_bt_controller_config_t::controller_task_stack_size (C++ 成员), 149
esp_bt_controller_config_t::hci_uart_baudrate (C++ 成员), 150
esp_bt_controller_config_t::hci_uart_no (C++ 成员), 149
esp_bt_controller_config_t::magic (C++ 成员), 150
esp_bt_controller_config_t::mesh_adv_size (C++ 成员), 150
esp_bt_controller_config_t::mode (C++ 成员), 150
esp_bt_controller_config_t::normal_adv_size (C++ 成员), 150
esp_bt_controller_config_t::scan_duplicate_mode (C++ 成员), 150
esp_bt_controller_config_t::scan_duplicate_type (C++ 成员), 150
esp_bt_controller_config_t::send_adv_reserved_size (C++ 成员), 150

esp_bt_controller_deinit (C++ 函数), 145
esp_bt_controller_disable (C++ 函数), 146
esp_bt_controller_enable (C++ 函数), 145
esp_bt_controller_get_status (C++ 函数), 146
esp_bt_controller_init (C++ 函数), 145
esp_bt_controller_is_sleeping (C++ 函数), 149
esp_bt_controller_mem_release (C++ 函数), 146
ESP_BT_CONTROLLER_STATUS_ENABLED (C++ 枚举子), 152
ESP_BT_CONTROLLER_STATUS_IDLE (C++ 枚举子), 152
ESP_BT_CONTROLLER_STATUS_INITED (C++ 枚举子), 152
ESP_BT_CONTROLLER_STATUS_NUM (C++ 枚举子), 152
esp_bt_controller_status_t (C++ 类型), 152
esp_bt_controller_wakeup_request (C++ 函数), 149
esp_bt_dev_get_address (C++ 函数), 160
esp_bt_dev_set_device_name (C++ 函数), 160
esp_bt_dev_type_t (C++ 类型), 157
ESP_BT_DEVICE_TYPE_BLE (C++ 枚举子), 158
ESP_BT_DEVICE_TYPE_BREDR (C++ 枚举子), 157
ESP_BT_DEVICE_TYPE_DUMO (C++ 枚举子), 158
esp_bt_discovery_mode_t (C++ 类型), 284
esp_bt_duplicate_exceptional_subcode_type_t (C++ 类型), 197
esp_bt_eir_data_t (C++ 类), 280
esp_bt_eir_data_t::fec_required (C++ 成员), 280
esp_bt_eir_data_t::flag (C++ 成员), 280
esp_bt_eir_data_t::include_txpower (C++ 成员), 280
esp_bt_eir_data_t::include_uuid (C++ 成员), 280
esp_bt_eir_data_t::manufacturer_len (C++ 成员), 280
esp_bt_eir_data_t::p_manufacturer_data (C++ 成员), 280
esp_bt_eir_data_t::p_url (C++ 成员), 280
esp_bt_eir_data_t::url_len (C++ 成员), 280
ESP_BT_EIR_FLAG_BREDR_NOT_SPT (C 宏), 281
ESP_BT_EIR_FLAG_DMT_CONTROLLER_SPT (C 宏), 281
ESP_BT_EIR_FLAG_DMT_HOST_SPT (C 宏), 281
ESP_BT_EIR_FLAG_GEN_DISC (C 宏), 281
ESP_BT_EIR_FLAG_LIMIT_DISC (C 宏), 281
ESP_BT_EIR_MAX_LEN (C 宏), 281
ESP_BT_EIR_TYPE_CMPL_128BITS_UUID (C 宏), 281
ESP_BT_EIR_TYPE_CMPL_16BITS_UUID (C 宏), 281
ESP_BT_EIR_TYPE_CMPL_32BITS_UUID (C 宏), 281
ESP_BT_EIR_TYPE_CMPL_LOCAL_NAME (C 宏), 281
ESP_BT_EIR_TYPE_FLAGS (C 宏), 280
ESP_BT_EIR_TYPE_INCMPL_128BITS_UUID (C 宏), 281
ESP_BT_EIR_TYPE_INCMPL_16BITS_UUID (C 宏), 281
ESP_BT_EIR_TYPE_INCMPL_32BITS_UUID (C 宏), 281
ESP_BT_EIR_TYPE_MANU_SPECIFIC (C 宏), 281
ESP_BT_EIR_TYPE_MAX_NUM (C 宏), 281
ESP_BT_EIR_TYPE_SHORT_LOCAL_NAME (C 宏), 281
esp_bt_eir_type_t (C++ 类型), 283
ESP_BT_EIR_TYPE_TX_POWER_LEVEL (C 宏), 281
ESP_BT_EIR_TYPE_URL (C 宏), 281
esp_bt_gap_afh_channels (C++ 类型), 283
ESP_BT_GAP_AFH_CHANNELS_LEN (C 宏), 280
ESP_BT_GAP_AUTH_CMPL_EVT (C++ 枚举子), 286
esp_bt_gap_cancel_discovery (C++ 函数), 269
esp_bt_gap_cb_event_t (C++ 类型), 286
esp_bt_gap_cb_param_t (C++ 类型), 274
esp_bt_gap_cb_param_t::auth_cmpl (C++ 成员), 275
esp_bt_gap_cb_param_t::auth_cmpl_param (C++ 类), 275
esp_bt_gap_cb_param_t::auth_cmpl_param::bda (C++ 成员), 275
esp_bt_gap_cb_param_t::auth_cmpl_param::device_name (C++ 成员), 276
esp_bt_gap_cb_param_t::auth_cmpl_param::stat (C++ 成员), 275
esp_bt_gap_cb_param_t::cfm_req (C++ 成员), 275
esp_bt_gap_cb_param_t::cfm_req_param (C++ 类), 276
esp_bt_gap_cb_param_t::cfm_req_param::bda (C++ 成员), 276

esp_bt_gap_cb_param_t::cfm_req_param::num_val (C++ 成员), 276
esp_bt_gap_cb_param_t::config_eir_data (C++ 成员), 275
esp_bt_gap_cb_param_t::config_eir_data_param (C++ 类), 276
esp_bt_gap_cb_param_t::config_eir_data_param::eir_type (C++ 成员), 276
esp_bt_gap_cb_param_t::config_eir_data_param::eir_type_num (C++ 成员), 276
esp_bt_gap_cb_param_t::config_eir_data_param::status (C++ 成员), 276
esp_bt_gap_cb_param_t::disc_res (C++ 成员), 275
esp_bt_gap_cb_param_t::disc_res_param (C++ 类), 276
esp_bt_gap_cb_param_t::disc_res_param::bda (C++ 成员), 276
esp_bt_gap_cb_param_t::disc_res_param::num_prop (C++ 成员), 276
esp_bt_gap_cb_param_t::disc_res_param::prop (C++ 成员), 276
esp_bt_gap_cb_param_t::disc_st_chg (C++ 成员), 275
esp_bt_gap_cb_param_t::disc_state_changed_param (C++ 类), 276
esp_bt_gap_cb_param_t::disc_state_changed_param::state (C++ 成员), 277
esp_bt_gap_cb_param_t::key_notif (C++ 成员), 275
esp_bt_gap_cb_param_t::key_notif_param (C++ 类), 277
esp_bt_gap_cb_param_t::key_notif_param::bda (C++ 成员), 277
esp_bt_gap_cb_param_t::key_notif_param::passkey (C++ 成员), 277
esp_bt_gap_cb_param_t::key_req (C++ 成员), 275
esp_bt_gap_cb_param_t::key_req_param (C++ 类), 277
esp_bt_gap_cb_param_t::key_req_param::bda (C++ 成员), 277
esp_bt_gap_cb_param_t::pin_req (C++ 成员), 275
esp_bt_gap_cb_param_t::pin_req_param (C++ 类), 277
esp_bt_gap_cb_param_t::pin_req_param::bda (C++ 成员), 277
esp_bt_gap_cb_param_t::pin_req_param::min_16_digit (C++ 成员), 277
esp_bt_gap_cb_param_t::read_rmt_name (C++ 成员), 275
esp_bt_gap_cb_param_t::read_rmt_name_param (C++ 类), 277
esp_bt_gap_cb_param_t::read_rmt_name_param::rmt_name (C++ 成员), 278
esp_bt_gap_cb_param_t::read_rmt_name_param::stat (C++ 成员), 278
esp_bt_gap_cb_param_t::read_rssi_delta (C++ 成员), 275
esp_bt_gap_cb_param_t::read_rssi_delta_param (C++ 类), 278
esp_bt_gap_cb_param_t::read_rssi_delta_param::bda (C++ 成员), 278
esp_bt_gap_cb_param_t::read_rssi_delta_param::rssi_delta (C++ 成员), 278
esp_bt_gap_cb_param_t::read_rssi_delta_param::stat (C++ 成员), 278
esp_bt_gap_cb_param_t::rmt_srvc_rec (C++ 成员), 275
esp_bt_gap_cb_param_t::rmt_srvc_rec_param (C++ 类), 278
esp_bt_gap_cb_param_t::rmt_srvc_rec_param::bda (C++ 成员), 278
esp_bt_gap_cb_param_t::rmt_srvc_rec_param::stat (C++ 成员), 278
esp_bt_gap_cb_param_t::rmt_srvcs (C++ 成员), 275
esp_bt_gap_cb_param_t::rmt_srvcs_param (C++ 类), 278
esp_bt_gap_cb_param_t::rmt_srvcs_param::bda (C++ 成员), 278
esp_bt_gap_cb_param_t::rmt_srvcs_param::num_uuids (C++ 成员), 278

esp_bt_gap_cb_param_t::rmt_srvcs_param::stat (C++ 成员), 278
esp_bt_gap_cb_param_t::rmt_srvcs_param::uuid_list (C++ 成员), 279
esp_bt_gap_cb_param_t::set_afh_channels (C++ 成员), 275
esp_bt_gap_cb_param_t::set_afh_channels_param (C++ 类), 279
esp_bt_gap_cb_param_t::set_afh_channels_param::stat (C++ 成员), 279
esp_bt_gap_cb_t (C++ 类型), 283
ESP_BT_GAP_CFM_REQ_EVT (C++ 枚举子), 286
esp_bt_gap_config_eir_data (C++ 函数), 270
ESP_BT_GAP_CONFIG_EIR_DATA_EVT (C++ 枚举子), 287
ESP_BT_GAP_DEV_PROP_BDNAME (C++ 枚举子), 284
ESP_BT_GAP_DEV_PROP_COD (C++ 枚举子), 284
ESP_BT_GAP_DEV_PROP_EIR (C++ 枚举子), 284
ESP_BT_GAP_DEV_PROP_RSSI (C++ 枚举子), 284
esp_bt_gap_dev_prop_t (C++ 类), 279
esp_bt_gap_dev_prop_t::len (C++ 成员), 279
esp_bt_gap_dev_prop_t::type (C++ 成员), 279
esp_bt_gap_dev_prop_t::val (C++ 成员), 279
esp_bt_gap_dev_prop_type_t (C++ 类型), 284
ESP_BT_GAP_DISC_RES_EVT (C++ 枚举子), 286
ESP_BT_GAP_DISC_STATE_CHANGED_EVT (C++ 枚举子), 286
ESP_BT_GAP_DISCOVERY_STARTED (C++ 枚举子), 286
esp_bt_gap_discovery_state_t (C++ 类型), 286
ESP_BT_GAP_DISCOVERY_STOPPED (C++ 枚举子), 286
ESP_BT_GAP_EIR_DATA_LEN (C 宏), 280
ESP_BT_GAP_EVT_MAX (C++ 枚举子), 287
esp_bt_gap_get_bond_device_list (C++ 函数), 272
esp_bt_gap_get_bond_device_num (C++ 函数), 271
esp_bt_gap_get_cod (C++ 函数), 271
esp_bt_gap_get_cod_format_type (C++ 函数), 267
esp_bt_gap_get_cod_major_dev (C++ 函数), 267
esp_bt_gap_get_cod_minor_dev (C++ 函数), 267
esp_bt_gap_get_cod_srvc (C++ 函数), 267
esp_bt_gap_get_remote_service_record (C++ 函数), 269
esp_bt_gap_get_remote_services (C++ 函数), 269
esp_bt_gap_is_valid_cod (C++ 函数), 268
ESP_BT_GAP_KEY_NOTIF_EVT (C++ 枚举子), 286
ESP_BT_GAP_KEY_REQ_EVT (C++ 枚举子), 287
ESP_BT_GAP_MAX_BDNAME_LEN (C 宏), 280
ESP_BT_GAP_MAX_INQ_LEN (C 宏), 282
ESP_BT_GAP_MIN_INQ_LEN (C 宏), 282
esp_bt_gap_pin_reply (C++ 函数), 272
ESP_BT_GAP_PIN_REQ_EVT (C++ 枚举子), 286
esp_bt_gap_read_remote_name (C++ 函数), 274
ESP_BT_GAP_READ_REMOTE_NAME_EVT (C++ 枚举子), 287
esp_bt_gap_read_rssi_delta (C++ 函数), 271
ESP_BT_GAP_READ_RSSI_DELTA_EVT (C++ 枚举子), 287
esp_bt_gap_register_callback (C++ 函数), 268
esp_bt_gap_remove_bond_device (C++ 函数), 271
esp_bt_gap_resolve_eir_data (C++ 函数), 270
ESP_BT_GAP_RMT_SRVC_REC_EVT (C++ 枚举子), 286
ESP_BT_GAP_RMT_SRVCS_EVT (C++ 枚举子), 286
ESP_BT_GAP_RSSI_HIGH_THRLD (C 宏), 280
ESP_BT_GAP_RSSI_LOW_THRLD (C 宏), 280
esp_bt_gap_set_afh_channels (C++ 函数), 274
ESP_BT_GAP_SET_AFH_CHANNELS_EVT (C++ 枚举子), 287
esp_bt_gap_set_cod (C++ 函数), 270
esp_bt_gap_set_pin (C++ 函数), 272
esp_bt_gap_set_scan_mode (C++ 函数), 268
esp_bt_gap_set_security_param (C++ 函数), 273
esp_bt_gap_ssp_confirm_reply (C++ 函数), 273
esp_bt_gap_ssp_passkey_reply (C++ 函数), 273
esp_bt_gap_start_discovery (C++ 函数), 268
ESP_BT_GENERAL_DISCOVERABLE (C++ 枚举子), 284
esp_bt_hf_answer_call (C++ 函数), 361
esp_bt_hf_bsir (C++ 函数), 361
esp_bt_hf_cind_response (C++ 函数), 359
esp_bt_hf_clcc_response (C++ 函数), 360

ESP_BT_HF_CLIENT_NUMBER_LEN (C 宏), 351
ESP_BT_HF_CLIENT_OPERATOR_NAME_LEN (C 宏), 351
esp_bt_hf_cmee_response (C++ 函数), 358
esp_bt_hf_cnum_response (C++ 函数), 361
esp_bt_hf_connect (C++ 函数), 356
esp_bt_hf_connect_audio (C++ 函数), 356
esp_bt_hf_cops_response (C++ 函数), 360
esp_bt_hf_deinit (C++ 函数), 356
esp_bt_hf_disconnect (C++ 函数), 356
esp_bt_hf_disconnect_audio (C++ 函数), 357
esp_bt_hf_end_call (C++ 函数), 363
esp_bt_hf_indchange_notification (C++ 函数), 358
esp_bt_hf_init (C++ 函数), 355
ESP_BT_HF_NUMBER_LEN (C 宏), 329
ESP_BT_HF_OPERATOR_NAME_LEN (C 宏), 329
esp_bt_hf_out_call (C++ 函数), 362
esp_bt_hf_register_callback (C++ 函数), 355
esp_bt_hf_register_data_callback (C++ 函数), 363
esp_bt_hf_reject_call (C++ 函数), 362
esp_bt_hf_volume_control (C++ 函数), 357
esp_bt_hf_vra (C++ 函数), 357
ESP_BT_INIT_COD (C++ 枚举子), 283
ESP_BT_INQ_MODE_GENERAL_INQUIRY (C++ 枚举子), 287
ESP_BT_INQ_MODE_LIMITED_INQUIRY (C++ 枚举子), 287
esp_bt_inq_mode_t (C++ 类型), 287
ESP_BT_IO_CAP_IN (C 宏), 282
ESP_BT_IO_CAP_IO (C 宏), 282
ESP_BT_IO_CAP_NONE (C 宏), 282
ESP_BT_IO_CAP_OUT (C 宏), 282
esp_bt_io_cap_t (C++ 类型), 283
ESP_BT_LIMITED_DISCOVERABLE (C++ 枚举子), 284
esp_bt_mem_release (C++ 函数), 147
ESP_BT_MODE_BLE (C++ 枚举子), 151
ESP_BT_MODE_BTDM (C++ 枚举子), 151
ESP_BT_MODE_CLASSIC_BT (C++ 枚举子), 151
ESP_BT_MODE_IDLE (C++ 枚举子), 151
esp_bt_mode_t (C++ 类型), 151
ESP_BT_NON_CONNECTABLE (C++ 枚举子), 283
ESP_BT_NON_DISCOVERABLE (C++ 枚举子), 284
ESP_BT_OCTET16_LEN (C 宏), 155
esp_bt_octet16_t (C++ 类型), 156
ESP_BT_OCTET8_LEN (C 宏), 155
esp_bt_octet8_t (C++ 类型), 156
ESP_BT_PIN_CODE_LEN (C 宏), 281
esp_bt_pin_code_t (C++ 类型), 283
ESP_BT_PIN_TYPE_FIXED (C++ 枚举子), 285
esp_bt_pin_type_t (C++ 类型), 285
ESP_BT_PIN_TYPE_VARIABLE (C++ 枚举子), 285
ESP_BT_SET_COD_ALL (C++ 枚举子), 283
ESP_BT_SET_COD_MAJOR_MINOR (C++ 枚举子), 283
ESP_BT_SET_COD_SERVICE_CLASS (C++ 枚举子), 283
esp_bt_sleep_disable (C++ 函数), 148
esp_bt_sleep_enable (C++ 函数), 148
ESP_BT_SP_IOCAP_MODE (C++ 枚举子), 285
esp_bt_sp_param_t (C++ 类型), 285
ESP_BT_STATUS_AUTH_FAILURE (C++ 枚举子), 157
ESP_BT_STATUS_AUTH_REJECTED (C++ 枚举子), 157
ESP_BT_STATUS_BUSY (C++ 枚举子), 157
ESP_BT_STATUS_CONTROL_LE_DATA_LEN_UNSUPPORTED (C++ 枚举子), 157
ESP_BT_STATUS_DONE (C++ 枚举子), 157
ESP_BT_STATUS_EIR_TOO_LARGE (C++ 枚举子), 157
ESP_BT_STATUS_ERR_ILLEGAL_PARAMETER_FMT (C++ 枚举子), 157
ESP_BT_STATUS_FAIL (C++ 枚举子), 157
ESP_BT_STATUS_INVALID_STATIC_RAND_ADDR (C++ 枚举子), 157
ESP_BT_STATUS_MEMORY_FULL (C++ 枚举子), 157
ESP_BT_STATUS_NOMEM (C++ 枚举子), 157
ESP_BT_STATUS_NOT_READY (C++ 枚举子), 157
ESP_BT_STATUS_PARAM_OUT_OF_RANGE (C++ 枚举子), 157
ESP_BT_STATUS_PARM_INVALID (C++ 枚举子), 157
ESP_BT_STATUS_PEER_LE_DATA_LEN_UNSUPPORTED (C++ 枚举子), 157
ESP_BT_STATUS_PENDING (C++ 枚举子), 157
ESP_BT_STATUS_RMT_DEV_DOWN (C++ 枚举子), 157
ESP_BT_STATUS_SUCCESS (C++ 枚举子), 157
esp_bt_status_t (C++ 类型), 157

ESP_BT_STATUS_TIMEOUT (C++ 枚举子), 157
ESP_BT_STATUS_UNACCEPT_CONN_INTERVAL (C++ 枚举子), 157
ESP_BT_STATUS_UNHANDLED (C++ 枚举子), 157
ESP_BT_STATUS_UNSUPPORTED (C++ 枚举子), 157
esp_bt_uuid_t (C++ 类), 155
esp_bt_uuid_t::len (C++ 成员), 155
esp_bt_uuid_t::uuid (C++ 成员), 155
ESP_CHIP_ID_ESP32 (C++ 枚举子), 1418
ESP_CHIP_ID_INVALID (C++ 枚举子), 1419
esp_chip_id_t (C++ 类型), 1418
esp_chip_info (C++ 函数), 1685
esp_chip_info_t (C++ 类), 1685
esp_chip_info_t::cores (C++ 成员), 1685
esp_chip_info_t::features (C++ 成员), 1685
esp_chip_info_t::model (C++ 成员), 1685
esp_chip_info_t::revision (C++ 成员), 1685
esp_chip_model_t (C++ 类型), 1686
esp_deep_sleep (C++ 函数), 1722
esp_deep_sleep_disable_rom_logging (C++ 函数), 1723
esp_deep_sleep_start (C++ 函数), 1722
esp_deep_sleep_wake_stub_fn_t (C++ 类型), 1724
ESP_DEFAULT_GATT_IF (C 宏), 155
esp_default_wake_deep_sleep (C++ 函数), 1723
esp_deregister_freertos_idle_hook (C++ 函数), 1615
esp_deregister_freertos_idle_hook_for_cpu (C++ 函数), 1615
esp_deregister_freertos_tick_hook (C++ 函数), 1616
esp_deregister_freertos_tick_hook_for_cpu (C++ 函数), 1615
esp_derive_local_mac (C++ 函数), 1684
esp_duplicate_info_t (C++ 类型), 190
esp_duplicate_scan_exceptional_list_type_t (C++ 类型), 198
ESP_EARLY_LOGD (C 宏), 1676
ESP_EARLY_LOGE (C 宏), 1675
ESP_EARLY_LOGI (C 宏), 1676
ESP_EARLY_LOGV (C 宏), 1676
ESP_EARLY_LOGW (C 宏), 1676
esp_efuse_batch_write_begin (C++ 函数), 1441
esp_efuse_batch_write_cancel (C++ 函数), 1441
esp_efuse_batch_write_commit (C++ 函数), 1441
esp_efuse_burn_new_values (C++ 函数), 1441
esp_efuse_check_secure_version (C++ 函数), 1441
esp_efuse_desc_t (C++ 类), 1441
esp_efuse_desc_t::bit_count (C++ 成员), 1441
esp_efuse_desc_t::bit_start (C++ 成员), 1441
esp_efuse_desc_t::efuse_block (C++ 成员), 1441
esp_efuse_disable_basic_rom_console (C++ 函数), 1441
esp_efuse_get_chip_ver (C++ 函数), 1440
esp_efuse_get_coding_scheme (C++ 函数), 1439
esp_efuse_get_field_size (C++ 函数), 1439
esp_efuse_get_pkg_ver (C++ 函数), 1441
esp_efuse_init (C++ 函数), 1441
esp_efuse_mac_get_custom (C++ 函数), 1683
esp_efuse_mac_get_default (C++ 函数), 1684
esp_efuse_read_block (C++ 函数), 1440
esp_efuse_read_field_blob (C++ 函数), 1436
esp_efuse_read_field_cnt (C++ 函数), 1437
esp_efuse_read_reg (C++ 函数), 1439
esp_efuse_read_secure_version (C++ 函数), 1441
esp_efuse_reset (C++ 函数), 1441
esp_efuse_set_read_protect (C++ 函数), 1438
esp_efuse_set_write_protect (C++ 函数), 1438
esp_efuse_update_secure_version (C++ 函数), 1441
esp_efuse_write_block (C++ 函数), 1440
esp_efuse_write_field_blob (C++ 函数), 1437
esp_efuse_write_field_cnt (C++ 函数), 1438
esp_efuse_write_random_key (C++ 函数), 1441
esp_efuse_write_reg (C++ 函数), 1439
ESP_ERR_CODING (C 宏), 1442
ESP_ERR_EFUSE (C 宏), 1441
ESP_ERR_EFUSE_CNT_IS_FULL (C 宏), 1442
ESP_ERR_EFUSE_REPEATED_PROG (C 宏), 1442
ESP_ERR_ESP_TLS_BASE (C 宏), 1149

ESP_ERR_ESP_TLS_CANNOT_CREATE_SOCKET (C 宏), 1149
ESP_ERR_ESP_TLS_CANNOT_RESOLVE_HOSTNAME (C 宏), 1149
ESP_ERR_ESP_TLS_CONNECTION_TIMEOUT (C 宏), 1150
ESP_ERR_ESP_TLS_FAILED_CONNECT_TO_HOST (C 宏), 1149
ESP_ERR_ESP_TLS_SOCKET_SETOPT_FAILED (C 宏), 1149
ESP_ERR_ESP_TLS_UNSUPPORTED_PROTOCOL_FAMILY (C 宏), 1149
ESP_ERR_ESPNOW_ARG (C 宏), 712
ESP_ERR_ESPNOW_BASE (C 宏), 712
ESP_ERR_ESPNOW_EXIST (C 宏), 712
ESP_ERR_ESPNOW_FULL (C 宏), 712
ESP_ERR_ESPNOW_IF (C 宏), 712
ESP_ERR_ESPNOW_INTERNAL (C 宏), 712
ESP_ERR_ESPNOW_NO_MEM (C 宏), 712
ESP_ERR_ESPNOW_NOT_FOUND (C 宏), 712
ESP_ERR_ESPNOW_NOT_INIT (C 宏), 712
ESP_ERR_FLASH_BASE (C 宏), 1444
ESP_ERR_HTTP_BASE (C 宏), 1166
ESP_ERR_HTTP_CONNECT (C 宏), 1166
ESP_ERR_HTTP_CONNECTING (C 宏), 1167
ESP_ERR_HTTP_EAGAIN (C 宏), 1167
ESP_ERR_HTTP_FETCH_HEADER (C 宏), 1167
ESP_ERR_HTTP_INVALID_TRANSPORT (C 宏), 1167
ESP_ERR_HTTP_MAX_REDIRECT (C 宏), 1166
ESP_ERR_HTTP_WRITE_DATA (C 宏), 1167
ESP_ERR_HTTPD_ALLOC_MEM (C 宏), 1205
ESP_ERR_HTTPD_BASE (C 宏), 1204
ESP_ERR_HTTPD_HANDLER_EXISTS (C 宏), 1204
ESP_ERR_HTTPD_HANDLERS_FULL (C 宏), 1204
ESP_ERR_HTTPD_INVALID_REQ (C 宏), 1204
ESP_ERR_HTTPD_RESP_HDR (C 宏), 1205
ESP_ERR_HTTPD_RESP_SEND (C 宏), 1205
ESP_ERR_HTTPD_RESULT_TRUNC (C 宏), 1204
ESP_ERR_HTTPD_TASK (C 宏), 1205
ESP_ERR_HTTPS_OTA_BASE (C 宏), 1449
ESP_ERR_HTTPS_OTA_IN_PROGRESS (C 宏), 1449
ESP_ERR_INVALID_ARG (C 宏), 1443
ESP_ERR_INVALID_CRC (C 宏), 1443
ESP_ERR_INVALID_MAC (C 宏), 1444
ESP_ERR_INVALID_RESPONSE (C 宏), 1443
ESP_ERR_INVALID_SIZE (C 宏), 1443
ESP_ERR_INVALID_STATE (C 宏), 1443
ESP_ERR_INVALID_VERSION (C 宏), 1444
ESP_ERR_MBEDTLS_CERT_PARTLY_OK (C 宏), 1149
ESP_ERR_MBEDTLS_CTR_DRBG_SEED_FAILED (C 宏), 1149
ESP_ERR_MBEDTLS_PK_PARSE_KEY_FAILED (C 宏), 1150
ESP_ERR_MBEDTLS_SSL_CONF_ALPN_PROTOCOLS_FAILED (C 宏), 1150
ESP_ERR_MBEDTLS_SSL_CONF_OWN_CERT_FAILED (C 宏), 1150
ESP_ERR_MBEDTLS_SSL_CONF_PSK_FAILED (C 宏), 1150
ESP_ERR_MBEDTLS_SSL_CONFIG_DEFAULTS_FAILED (C 宏), 1149
ESP_ERR_MBEDTLS_SSL_HANDSHAKE_FAILED (C 宏), 1150
ESP_ERR_MBEDTLS_SSL_SET_HOSTNAME_FAILED (C 宏), 1149
ESP_ERR_MBEDTLS_SSL_SETUP_FAILED (C 宏), 1150
ESP_ERR_MBEDTLS_SSL_WRITE_FAILED (C 宏), 1150
ESP_ERR_MBEDTLS_X509_CRT_PARSE_FAILED (C 宏), 1150
ESP_ERR_MESH_ARGUMENT (C 宏), 750
ESP_ERR_MESH_BASE (C 宏), 1444
ESP_ERR_MESH_DISCARD (C 宏), 751
ESP_ERR_MESH_DISCARD_DUPLICATE (C 宏), 751
ESP_ERR_MESH_DISCONNECTED (C 宏), 750
ESP_ERR_MESH_EXCEED_MTU (C 宏), 750
ESP_ERR_MESH_INTERFACE (C 宏), 751
ESP_ERR_MESH_NO_MEMORY (C 宏), 750
ESP_ERR_MESH_NO_PARENT_FOUND (C 宏), 750
ESP_ERR_MESH_NO_ROUTE_FOUND (C 宏), 751
ESP_ERR_MESH_NOT_ALLOWED (C 宏), 750
ESP_ERR_MESH_NOT_CONFIG (C 宏), 750
ESP_ERR_MESH_NOT_INIT (C 宏), 750
ESP_ERR_MESH_NOT_START (C 宏), 750
ESP_ERR_MESH_NOT_SUPPORT (C 宏), 750

ESP_ERR_MESH_OPTION_NULL (C 宏), 751
ESP_ERR_MESH_OPTION_UNKNOWN (C 宏), 751
ESP_ERR_MESH_QUEUE_FAIL (C 宏), 750
ESP_ERR_MESH_QUEUE_FULL (C 宏), 750
ESP_ERR_MESH_TIMEOUT (C 宏), 750
ESP_ERR_MESH_VOTING (C 宏), 751
ESP_ERR_MESH_WIFI_NOT_START (C 宏), 750
ESP_ERR_MESH_XON_NO_WINDOW (C 宏), 751
ESP_ERR_NO_MEM (C 宏), 1443
ESP_ERR_NOT_FINISHED (C 宏), 1260
ESP_ERR_NOT_FOUND (C 宏), 1443
ESP_ERR_NOT_SUPPORTED (C 宏), 1443
ESP_ERR_NVS_BASE (C 宏), 1369
ESP_ERR_NVS_CONTENT_DIFFERS (C 宏), 1370
ESP_ERR_NVS_CORRUPT_KEY_PART (C 宏), 1370
ESP_ERR_NVS_ENCR_NOT_SUPPORTED (C 宏), 1370
ESP_ERR_NVS_INVALID_HANDLE (C 宏), 1369
ESP_ERR_NVS_INVALID_LENGTH (C 宏), 1369
ESP_ERR_NVS_INVALID_NAME (C 宏), 1369
ESP_ERR_NVS_INVALID_STATE (C 宏), 1369
ESP_ERR_NVS_KEY_TOO_LONG (C 宏), 1369
ESP_ERR_NVS_KEYS_NOT_INITIALIZED (C 宏), 1370
ESP_ERR_NVS_NEW_VERSION_FOUND (C 宏), 1370
ESP_ERR_NVS_NO_FREE_PAGES (C 宏), 1370
ESP_ERR_NVS_NOT_ENOUGH_SPACE (C 宏), 1369
ESP_ERR_NVS_NOT_FOUND (C 宏), 1369
ESP_ERR_NVS_NOT_INITIALIZED (C 宏), 1369
ESP_ERR_NVS_PAGE_FULL (C 宏), 1369
ESP_ERR_NVS_PART_NOT_FOUND (C 宏), 1370
ESP_ERR_NVS_READ_ONLY (C 宏), 1369
ESP_ERR_NVS_REMOVE_FAILED (C 宏), 1369
ESP_ERR_NVS_TYPE_MISMATCH (C 宏), 1369
ESP_ERR_NVS_VALUE_TOO_LONG (C 宏), 1370
ESP_ERR_NVS_XTS_CFG_FAILED (C 宏), 1370
ESP_ERR_NVS_XTS_CFG_NOT_FOUND (C 宏), 1370
ESP_ERR_NVS_XTS_DECR_FAILED (C 宏), 1370
ESP_ERR_NVS_XTS_ENCR_FAILED (C 宏), 1370
ESP_ERR_OTA_BASE (C 宏), 1702
ESP_ERR_OTA_PARTITION_CONFLICT (C 宏), 1702
ESP_ERR_OTA_ROLLBACK_FAILED (C 宏), 1702
ESP_ERR_OTA_ROLLBACK_INVALID_STATE (C 宏), 1702
ESP_ERR_OTA_SELECT_INFO_INVALID (C 宏), 1702
ESP_ERR_OTA_SMALL_SEC_VER (C 宏), 1702
ESP_ERR_OTA_VALIDATE_FAILED (C 宏), 1702
esp_err_t (C++ 类型), 1444
ESP_ERR_TIMEOUT (C 宏), 1443
esp_err_to_name (C++ 函数), 1442
esp_err_to_name_r (C++ 函数), 1442
ESP_ERR_ULP_BASE (C 宏), 2231
ESP_ERR_ULP_BRANCH_OUT_OF_RANGE (C 宏), 2231
ESP_ERR_ULP_DUPLICATE_LABEL (C 宏), 2231
ESP_ERR_ULP_INVALID_LOAD_ADDR (C 宏), 2231
ESP_ERR_ULP_SIZE_TOO_BIG (C 宏), 2231
ESP_ERR_ULP_UNDEFINED_LABEL (C 宏), 2231
ESP_ERR_WIFI_BASE (C 宏), 1444
ESP_ERR_WIFI_CONN (C 宏), 675
ESP_ERR_WIFI_IF (C 宏), 675
ESP_ERR_WIFI_INIT_STATE (C 宏), 676
ESP_ERR_WIFI_MAC (C 宏), 675
ESP_ERR_WIFI_MODE (C 宏), 675
ESP_ERR_WIFI_NOT_CONNECT (C 宏), 676
ESP_ERR_WIFI_NOT_INIT (C 宏), 675
ESP_ERR_WIFI_NOT_STARTED (C 宏), 675
ESP_ERR_WIFI_NOT_STOPPED (C 宏), 675
ESP_ERR_WIFI_NVS (C 宏), 675
ESP_ERR_WIFI_PASSWORD (C 宏), 675
ESP_ERR_WIFI_POST (C 宏), 676
ESP_ERR_WIFI_SSID (C 宏), 675
ESP_ERR_WIFI_STATE (C 宏), 675
ESP_ERR_WIFI_STOP_STATE (C 宏), 676
ESP_ERR_WIFI_TIMEOUT (C 宏), 675
ESP_ERR_WIFI_WAKE_FAIL (C 宏), 675
ESP_ERR_WIFI_WOULD_BLOCK (C 宏), 675
ESP_ERROR_CHECK (C 宏), 1444
ESP_ERROR_CHECK_WITHOUT_ABORT (C 宏), 1444
esp_esptouch_set_timeout (C++ 函数), 701
esp_eth_clear_default_handlers (C++ 函数), 779
esp_eth_config_t (C++ 类), 761
esp_eth_config_t::check_link_period_ms (C++ 成员), 761
esp_eth_config_t::mac (C++ 成员), 761

esp_eth_config_t::on_lowlevel_deinit_done (C++ 成员), 762
esp_eth_config_t::on_lowlevel_init_done (C++ 成员), 761
esp_eth_config_t::phy (C++ 成员), 761
esp_eth_config_t::stack_input (C++ 成员), 761
esp_eth_decrease_reference (C++ 函数), 761
esp_eth_del_netif_glue (C++ 函数), 778
esp_eth_detect_phy_addr (C++ 函数), 762
esp_eth_driver_install (C++ 函数), 757
esp_eth_driver_uninstall (C++ 函数), 758
esp_eth_handle_t (C++ 类型), 762
esp_eth_increase_reference (C++ 函数), 760
esp_eth_io_cmd_t (C++ 类型), 765
esp_eth_ioctl (C++ 函数), 760
esp_eth_mac_s (C++ 类), 767
esp_eth_mac_s::deinit (C++ 成员), 768
esp_eth_mac_s::del (C++ 成员), 772
esp_eth_mac_s::get_addr (C++ 成员), 770
esp_eth_mac_s::init (C++ 成员), 767
esp_eth_mac_s::read_phy_reg (C++ 成员), 769
esp_eth_mac_s::receive (C++ 成员), 769
esp_eth_mac_s::set_addr (C++ 成员), 770
esp_eth_mac_s::set_duplex (C++ 成员), 771
esp_eth_mac_s::set_link (C++ 成员), 771
esp_eth_mac_s::set_mediator (C++ 成员), 767
esp_eth_mac_s::set_promiscuous (C++ 成员), 772
esp_eth_mac_s::set_speed (C++ 成员), 771
esp_eth_mac_s::start (C++ 成员), 768
esp_eth_mac_s::stop (C++ 成员), 768
esp_eth_mac_s::transmit (C++ 成员), 768
esp_eth_mac_s::write_phy_reg (C++ 成员), 770
esp_eth_mac_t (C++ 类型), 773
esp_eth_mediator_s (C++ 类), 763
esp_eth_mediator_s::on_state_changed (C++ 成员), 764
esp_eth_mediator_s::phy_reg_read (C++ 成员), 763
esp_eth_mediator_s::phy_reg_write (C++ 成员), 763
esp_eth_mediator_s::stack_input (C++ 成员), 764
esp_eth_mediator_t (C++ 类型), 765
esp_eth_new_netif_glue (C++ 函数), 778
ESP_ETH_PHY_ADDR_AUTO (C 宏), 778
esp_eth_phy_new_dp83848 (C++ 函数), 774
esp_eth_phy_new_ip101 (C++ 函数), 773
esp_eth_phy_new_lan8720 (C++ 函数), 774
esp_eth_phy_new_rtl8201 (C++ 函数), 773
esp_eth_phy_s (C++ 类), 774
esp_eth_phy_s::deinit (C++ 成员), 775
esp_eth_phy_s::del (C++ 成员), 777
esp_eth_phy_s::get_addr (C++ 成员), 777
esp_eth_phy_s::get_link (C++ 成员), 776
esp_eth_phy_s::init (C++ 成员), 775
esp_eth_phy_s::negotiate (C++ 成员), 776
esp_eth_phy_s::pwrctl (C++ 成员), 776
esp_eth_phy_s::reset (C++ 成员), 775
esp_eth_phy_s::reset_hw (C++ 成员), 775
esp_eth_phy_s::set_addr (C++ 成员), 776
esp_eth_phy_s::set_mediator (C++ 成员), 774
esp_eth_phy_t (C++ 类型), 778
esp_eth_receive (C++ 函数), 759
esp_eth_set_default_handlers (C++ 函数), 778
esp_eth_start (C++ 函数), 758
esp_eth_state_t (C++ 类型), 765
esp_eth_stop (C++ 函数), 758
esp_eth_transmit (C++ 函数), 759
esp_eth_update_input_path (C++ 函数), 759
ESP_EVENT_ANY_BASE (C 宏), 1464
ESP_EVENT_ANY_ID (C 宏), 1464
esp_event_base_t (C++ 类型), 1465
ESP_EVENT_DECLARE_BASE (C 宏), 1464
ESP_EVENT_DEFINE_BASE (C 宏), 1464
esp_event_dump (C++ 函数), 1463
esp_event_handler_register (C++ 函数), 1459
esp_event_handler_register_with (C++ 函数), 1460
esp_event_handler_t (C++ 类型), 1465
esp_event_handler_unregister (C++ 函数), 1460
esp_event_handler_unregister_with (C++ 函数), 1461
esp_event_loop_args_t (C++ 类), 1464

esp_event_loop_args_t::queue_size (C++ 成员), 1464
esp_event_loop_args_t::task_core_id (C++ 成员), 1464
esp_event_loop_args_t::task_name (C++ 成员), 1464
esp_event_loop_args_t::task_priority (C++ 成员), 1464
esp_event_loop_args_t::task_stack_size (C++ 成员), 1464
esp_event_loop_create (C++ 函数), 1457
esp_event_loop_create_default (C++ 函数), 1458
esp_event_loop_delete (C++ 函数), 1458
esp_event_loop_delete_default (C++ 函数), 1458
esp_event_loop_handle_t (C++ 类型), 1465
esp_event_loop_init (C++ 函数), 1466
esp_event_loop_run (C++ 函数), 1458
esp_event_loop_set_cb (C++ 函数), 1467
esp_event_post (C++ 函数), 1461
esp_event_post_to (C++ 函数), 1462
esp_event_process_default (C++ 函数), 1466
esp_event_send (C++ 函数), 1465
esp_event_send_internal (C++ 函数), 1465
esp_event_set_default_eth_handlers (C++ 函数), 1466
esp_event_set_default_wifi_handlers (C++ 函数), 1466
ESP_EXECUTE_EXPRESSION_WITH_STACK (C 宏), 1662
ESP_EXT1_WAKEUP_ALL_LOW (C++ 枚举子), 1724
ESP_EXT1_WAKEUP_ANY_HIGH (C++ 枚举子), 1724
ESP_FAIL (C 宏), 1443
esp_fill_random (C++ 函数), 1683
ESP_FLASH_10MHZ (C++ 枚举子), 1322
ESP_FLASH_20MHZ (C++ 枚举子), 1322
ESP_FLASH_26MHZ (C++ 枚举子), 1322
ESP_FLASH_40MHZ (C++ 枚举子), 1322
ESP_FLASH_5MHZ (C++ 枚举子), 1322
ESP_FLASH_80MHZ (C++ 枚举子), 1322
esp_flash_chip_driver_initialized (C++ 函数), 1313
ESP_FLASH_ENC_MODE_DEVELOPMENT (C++ 枚举子), 1333
ESP_FLASH_ENC_MODE_DISABLED (C++ 枚举子), 1333
ESP_FLASH_ENC_MODE_RELEASE (C++ 枚举子), 1333
esp_flash_enc_mode_t (C++ 类型), 1333
esp_flash_encrypt_check_and_update (C++ 函数), 1332
esp_flash_encrypt_region (C++ 函数), 1332
esp_flash_encryption_enabled (C++ 函数), 1332
esp_flash_encryption_init_checks (C++ 函数), 1333
esp_flash_erase_chip (C++ 函数), 1313
esp_flash_erase_region (C++ 函数), 1314
esp_flash_get_chip_write_protect (C++ 函数), 1314
esp_flash_get_protectable_regions (C++ 函数), 1315
esp_flash_get_protected_region (C++ 函数), 1315
esp_flash_get_size (C++ 函数), 1313
esp_flash_init (C++ 函数), 1312
esp_flash_io_mode_t (C++ 类型), 1322
esp_flash_is_quad_mode (C++ 函数), 1318
esp_flash_os_functions_t (C++ 类), 1318
esp_flash_os_functions_t::delay_ms (C++ 成员), 1319
esp_flash_os_functions_t::end (C++ 成员), 1318
esp_flash_os_functions_t::region_protected (C++ 成员), 1319
esp_flash_os_functions_t::start (C++ 成员), 1318
esp_flash_read (C++ 函数), 1316
esp_flash_read_encrypted (C++ 函数), 1317
esp_flash_read_id (C++ 函数), 1313
esp_flash_region_t (C++ 类), 1318
esp_flash_region_t::offset (C++ 成员), 1318
esp_flash_region_t::size (C++ 成员), 1318
esp_flash_set_chip_write_protect (C++ 函数), 1314
esp_flash_set_protected_region (C++ 函数),

1315
ESP_FLASH_SPEED_MAX (C++ 枚举子), 1322
ESP_FLASH_SPEED_MIN (C 宏), 1322
esp_flash_speed_t (C++ 类型), 1322
esp_flash_spi_device_config_t (C++ 类), 1312
esp_flash_spi_device_config_t::cs_id (C++ 成员), 1312
esp_flash_spi_device_config_t::cs_io_num (C++ 成员), 1312
esp_flash_spi_device_config_t::host_id (C++ 成员), 1312
esp_flash_spi_device_config_t::input_delay_ns (C++ 成员), 1312
esp_flash_spi_device_config_t::io_mode (C++ 成员), 1312
esp_flash_spi_device_config_t::speed (C++ 成员), 1312
esp_flash_t (C++ 类), 1319
esp_flash_t (C++ 类型), 1319
esp_flash_t::chip_drv (C++ 成员), 1319
esp_flash_t::chip_id (C++ 成员), 1319
esp_flash_t::host (C++ 成员), 1319
esp_flash_t::os_func (C++ 成员), 1319
esp_flash_t::os_func_data (C++ 成员), 1319
esp_flash_t::read_mode (C++ 成员), 1319
esp_flash_t::size (C++ 成员), 1319
esp_flash_write (C++ 函数), 1316
esp_flash_write_encrypted (C++ 函数), 1317
esp_flash_write_protect_crypt_cnt (C++ 函数), 1333
esp_freertos_idle_cb_t (C++ 类型), 1616
esp_freertos_tick_cb_t (C++ 类型), 1616
ESP_GAP_BLE_ADD_WHITELIST_COMPLETE_EVT (C 宏), 190
ESP_GAP_BLE_ADV_DATA_RAW_SET_COMPLETE_EVT (C++ 枚举子), 191
ESP_GAP_BLE_ADV_DATA_SET_COMPLETE_EVT (C++ 枚举子), 190
ESP_GAP_BLE_ADV_START_COMPLETE_EVT (C++ 枚举子), 191
ESP_GAP_BLE_ADV_STOP_COMPLETE_EVT (C++ 枚举子), 191
ESP_GAP_BLE_AUTH_CMPL_EVT (C++ 枚举子), 191
esp_gap_ble_cb_event_t (C++ 类型), 190
esp_gap_ble_cb_t (C++ 类型), 190
esp_gap_ble_channels (C++ 类型), 190
ESP_GAP_BLE_CHANNELS_LEN (C 宏), 190
ESP_GAP_BLE_CLEAR_BOND_DEV_COMPLETE_EVT (C++ 枚举子), 192
ESP_GAP_BLE_EVT_MAX (C++ 枚举子), 192
ESP_GAP_BLE_GET_BOND_DEV_COMPLETE_EVT (C++ 枚举子), 192
ESP_GAP_BLE_KEY_EVT (C++ 枚举子), 191
ESP_GAP_BLE_LOCAL_ER_EVT (C++ 枚举子), 191
ESP_GAP_BLE_LOCAL_IR_EVT (C++ 枚举子), 191
ESP_GAP_BLE_NC_REQ_EVT (C++ 枚举子), 191
ESP_GAP_BLE_OOB_REQ_EVT (C++ 枚举子), 191
ESP_GAP_BLE_PASSKEY_NOTIF_EVT (C++ 枚举子), 191
ESP_GAP_BLE_PASSKEY_REQ_EVT (C++ 枚举子), 191
ESP_GAP_BLE_READ_RSSI_COMPLETE_EVT (C++ 枚举子), 192
ESP_GAP_BLE_REMOVE_BOND_DEV_COMPLETE_EVT (C++ 枚举子), 192
ESP_GAP_BLE_SCAN_PARAM_SET_COMPLETE_EVT (C++ 枚举子), 191
ESP_GAP_BLE_SCAN_RESULT_EVT (C++ 枚举子), 191
ESP_GAP_BLE_SCAN_RSP_DATA_RAW_SET_COMPLETE_EVT (C++ 枚举子), 191
ESP_GAP_BLE_SCAN_RSP_DATA_SET_COMPLETE_EVT (C++ 枚举子), 191
ESP_GAP_BLE_SCAN_START_COMPLETE_EVT (C++ 枚举子), 191
ESP_GAP_BLE_SCAN_STOP_COMPLETE_EVT (C++ 枚举子), 192
ESP_GAP_BLE_SEC_REQ_EVT (C++ 枚举子), 191
esp_gap_ble_set_channels (C++ 函数), 171
ESP_GAP_BLE_SET_CHANNELS_EVT (C++ 枚举子), 192
ESP_GAP_BLE_SET_LOCAL_PRIVACY_COMPLETE_EVT (C++ 枚举子), 192
ESP_GAP_BLE_SET_PKT_LENGTH_COMPLETE_EVT (C++ 枚举子), 192
ESP_GAP_BLE_SET_STATIC_RAND_ADDR_EVT (C++

枚举子), 192
ESP_GAP_BLE_UPDATE_CONN_PARAMS_EVT (C++ 枚举子), 192
ESP_GAP_BLE_UPDATE_DUPLICATE_EXCEPTIONAL_LIST_COMPLETE_EVT (C++ 枚举子), 192
ESP_GAP_BLE_UPDATE_WHITELIST_COMPLETE_EVT (C++ 枚举子), 192
esp_gap_conn_params_t (C++ 类), 181
esp_gap_conn_params_t::interval (C++ 成员), 182
esp_gap_conn_params_t::latency (C++ 成员), 182
esp_gap_conn_params_t::timeout (C++ 成员), 182
ESP_GAP_SEARCH_DI_DISC_CMPL_EVT (C++ 枚举子), 196
ESP_GAP_SEARCH_DISC_BLE_RES_EVT (C++ 枚举子), 196
ESP_GAP_SEARCH_DISC_CMPL_EVT (C++ 枚举子), 196
ESP_GAP_SEARCH_DISC_RES_EVT (C++ 枚举子), 196
esp_gap_search_evt_t (C++ 类型), 196
ESP_GAP_SEARCH_INQ_CMPL_EVT (C++ 枚举子), 196
ESP_GAP_SEARCH_INQ_DISCARD_NUM_EVT (C++ 枚举子), 197
ESP_GAP_SEARCH_INQ_RES_EVT (C++ 枚举子), 196
ESP_GAP_SEARCH_SEARCH_CANCEL_CMPL_EVT (C++ 枚举子), 196
ESP_GATT_ALREADY_OPEN (C++ 枚举子), 210
ESP_GATT_APP_RSP (C++ 枚举子), 210
ESP_GATT_ATTR_HANDLE_MAX (C 宏), 207
ESP_GATT_AUTH_FAIL (C++ 枚举子), 210
ESP_GATT_AUTH_REQ_MITM (C++ 枚举子), 211
ESP_GATT_AUTH_REQ_NO_MITM (C++ 枚举子), 211
ESP_GATT_AUTH_REQ_NONE (C++ 枚举子), 211
ESP_GATT_AUTH_REQ_SIGNED_MITM (C++ 枚举子), 211
ESP_GATT_AUTH_REQ_SIGNED_NO_MITM (C++ 枚举子), 211
esp_gatt_auth_req_t (C++ 类型), 211
ESP_GATT_AUTO_RSP (C 宏), 208
ESP_GATT_BODY_SENSOR_LOCATION (C 宏), 207
ESP_GATT_BUSY (C++ 枚举子), 210
ESP_GATT_CANCEL (C++ 枚举子), 210
ESP_GATT_CCC_CFG_ERR (C++ 枚举子), 210
ESP_GATT_CHAR_PROP_BIT_AUTH (C 宏), 208
ESP_GATT_CHAR_PROP_BIT_BROADCAST (C 宏), 208
ESP_GATT_CHAR_PROP_BIT_EXT_PROP (C 宏), 208
ESP_GATT_CHAR_PROP_BIT_INDICATE (C 宏), 208
ESP_GATT_CHAR_PROP_BIT_NOTIFY (C 宏), 208
ESP_GATT_CHAR_PROP_BIT_READ (C 宏), 208
ESP_GATT_CHAR_PROP_BIT_WRITE (C 宏), 208
ESP_GATT_CHAR_PROP_BIT_WRITE_NR (C 宏), 208
esp_gatt_char_prop_t (C++ 类型), 208
ESP_GATT_CMD_STARTED (C++ 枚举子), 210
ESP_GATT_CONGESTED (C++ 枚举子), 210
ESP_GATT_CONN_CONN_CANCEL (C++ 枚举子), 211
ESP_GATT_CONN_FAIL_ESTABLISH (C++ 枚举子), 211
ESP_GATT_CONN_L2C_FAILURE (C++ 枚举子), 210
ESP_GATT_CONN_LMP_TIMEOUT (C++ 枚举子), 211
ESP_GATT_CONN_NONE (C++ 枚举子), 211
esp_gatt_conn_params_t (C++ 类), 202
esp_gatt_conn_params_t::interval (C++ 成员), 202
esp_gatt_conn_params_t::latency (C++ 成员), 202
esp_gatt_conn_params_t::timeout (C++ 成员), 202
esp_gatt_conn_reason_t (C++ 类型), 210
ESP_GATT_CONN_TERMINATE_LOCAL_HOST (C++ 枚举子), 211
ESP_GATT_CONN_TERMINATE_PEER_USER (C++ 枚举子), 211
ESP_GATT_CONN_TIMEOUT (C++ 枚举子), 210
ESP_GATT_CONN_UNKNOWN (C++ 枚举子), 210
ESP_GATT_DB_ALL (C++ 枚举子), 212
esp_gatt_db_attr_type_t (C++ 类型), 212
ESP_GATT_DB_CHARACTERISTIC (C++ 枚举子), 212
ESP_GATT_DB_DESCRIPTOR (C++ 枚举子), 212
ESP_GATT_DB_FULL (C++ 枚举子), 210
ESP_GATT_DB_INCLUDED_SERVICE (C++ 枚举子), 212
ESP_GATT_DB_PRIMARY_SERVICE (C++ 枚举子), 212

ESP_GATT_DB_SECONDARY_SERVICE (C++ 枚举子), 212
ESP_GATT_DUP_REG (C++ 枚举子), 210
ESP_GATT_ENCRYPED_MITM (C++ 枚举子), 210
ESP_GATT_ENCRYPED_NO_MITM (C++ 枚举子), 210
ESP_GATT_ERR_UNLIKELY (C++ 枚举子), 209
ESP_GATT_ERROR (C++ 枚举子), 210
ESP_GATT_HEART_RATE_CNTL_POINT (C 宏), 207
ESP_GATT_HEART_RATE_MEAS (C 宏), 207
esp_gatt_id_t (C++ 类), 199
esp_gatt_id_t::inst_id (C++ 成员), 199
esp_gatt_id_t::uuid (C++ 成员), 199
ESP_GATT_IF_NONE (C 宏), 208
esp_gatt_if_t (C++ 类型), 208
ESP_GATT_ILLEGAL_HANDLE (C 宏), 207
ESP_GATT_ILLEGAL_PARAMETER (C++ 枚举子), 210
ESP_GATT_ILLEGAL_UUID (C 宏), 207
ESP_GATT_INSUF_AUTHENTICATION (C++ 枚举子), 209
ESP_GATT_INSUF_AUTHORIZATION (C++ 枚举子), 209
ESP_GATT_INSUF_ENCRYPTION (C++ 枚举子), 209
ESP_GATT_INSUF_KEY_SIZE (C++ 枚举子), 209
ESP_GATT_INSUF_RESOURCE (C++ 枚举子), 209
ESP_GATT_INTERNAL_ERROR (C++ 枚举子), 209
ESP_GATT_INVALID_ATTR_LEN (C++ 枚举子), 209
ESP_GATT_INVALID_CFG (C++ 枚举子), 210
ESP_GATT_INVALID_HANDLE (C++ 枚举子), 209
ESP_GATT_INVALID_OFFSET (C++ 枚举子), 209
ESP_GATT_INVALID_PDU (C++ 枚举子), 209
ESP_GATT_MAX_ATTR_LEN (C 宏), 208
ESP_GATT_MAX_READ_MULTI_HANDLES (C 宏), 207
ESP_GATT_MORE (C++ 枚举子), 210
ESP_GATT_NO_RESOURCES (C++ 枚举子), 209
ESP_GATT_NOT_ENCRYPTED (C++ 枚举子), 210
ESP_GATT_NOT_FOUND (C++ 枚举子), 209
ESP_GATT_NOT_LONG (C++ 枚举子), 209
ESP_GATT_OK (C++ 枚举子), 209
ESP_GATT_OUT_OF_RANGE (C++ 枚举子), 210
ESP_GATT_PENDING (C++ 枚举子), 210
ESP_GATT_PERM_READ (C 宏), 207
ESP_GATT_PERM_READ_ENC_MITM (C 宏), 208
ESP_GATT_PERM_READ_ENCRYPTED (C 宏), 208
esp_gatt_perm_t (C++ 类型), 208
ESP_GATT_PERM_WRITE (C 宏), 208
ESP_GATT_PERM_WRITE_ENC_MITM (C 宏), 208
ESP_GATT_PERM_WRITE_ENCRYPTED (C 宏), 208
ESP_GATT_PERM_WRITE_SIGNED (C 宏), 208
ESP_GATT_PERM_WRITE_SIGNED_MITM (C 宏), 208
ESP_GATT_PRC_IN_PROGRESS (C++ 枚举子), 210
ESP_GATT_PREP_WRITE_CANCEL (C 宏), 227
ESP_GATT_PREP_WRITE_CANCEL (C++ 枚举子), 209
ESP_GATT_PREP_WRITE_EXEC (C 宏), 227
ESP_GATT_PREP_WRITE_EXEC (C++ 枚举子), 209
esp_gatt_prep_write_type (C++ 类型), 209
ESP_GATT_PREPARE_Q_FULL (C++ 枚举子), 209
ESP_GATT_READ_NOT_PERMIT (C++ 枚举子), 209
ESP_GATT_REQ_NOT_SUPPORTED (C++ 枚举子), 209
ESP_GATT_RSP_BY_APP (C 宏), 208
esp_gatt_rsp_t (C++ 类型), 199
esp_gatt_rsp_t::attr_value (C++ 成员), 199
esp_gatt_rsp_t::handle (C++ 成员), 199
ESP_GATT_SERVICE_FROM_NVS_FLASH (C++ 枚举子), 211
ESP_GATT_SERVICE_FROM_REMOTE_DEVICE (C++ 枚举子), 211
ESP_GATT_SERVICE_FROM_UNKNOWN (C++ 枚举子), 211
ESP_GATT_SERVICE_STARTED (C++ 枚举子), 210
esp_gatt_srvc_id_t (C++ 类), 199
esp_gatt_srvc_id_t::id (C++ 成员), 199
esp_gatt_srvc_id_t::is_primary (C++ 成员), 199
ESP_GATT_STACK_RSP (C++ 枚举子), 210
esp_gatt_status_t (C++ 类型), 209
ESP_GATT_UNKNOWN_ERROR (C++ 枚举子), 210
ESP_GATT_UNSUPPORT_GRP_TYPE (C++ 枚举子), 209
ESP_GATT_UUID_ALERT_LEVEL (C 宏), 206
ESP_GATT_UUID_ALERT_NTF_SVC (C 宏), 205
ESP_GATT_UUID_ALERT_STATUS (C 宏), 206
ESP_GATT_UUID_Automation_IO_SVC (C 宏), 205
ESP_GATT_UUID_BATTERY_LEVEL (C 宏), 207
ESP_GATT_UUID_BATTERY_SERVICE_SVC (C 宏), 204
ESP_GATT_UUID_BLOOD_PRESSURE_SVC (C 宏), 204

ESP_GATT_UUID_BODY_COMPOSITION (C 宏), 205
ESP_GATT_UUID_BOND_MANAGEMENT_SVC (C 宏), 205
ESP_GATT_UUID_CHAR_AGG_FORMAT (C 宏), 205
ESP_GATT_UUID_CHAR_CLIENT_CONFIG (C 宏), 205
ESP_GATT_UUID_CHAR_DECLARE (C 宏), 205
ESP_GATT_UUID_CHAR_DESCRIPTION (C 宏), 205
ESP_GATT_UUID_CHAR_EXT_PROP (C 宏), 205
ESP_GATT_UUID_CHAR_PRESENT_FORMAT (C 宏), 205
ESP_GATT_UUID_CHAR_SRVR_CONFIG (C 宏), 205
ESP_GATT_UUID_CHAR_VALID_RANGE (C 宏), 205
ESP_GATT_UUID_CONT_GLUCOSE_MONITOR_SVC (C 宏), 205
ESP_GATT_UUID_CSC_FEATURE (C 宏), 207
ESP_GATT_UUID_CSC_MEASUREMENT (C 宏), 207
ESP_GATT_UUID_CURRENT_TIME (C 宏), 206
ESP_GATT_UUID_CURRENT_TIME_SVC (C 宏), 204
ESP_GATT_UUID_CYCLING_POWER_SVC (C 宏), 205
ESP_GATT_UUID_CYCLING_SPEED_CADENCE_SVC (C 宏), 205
ESP_GATT_UUID_DEVICE_INFO_SVC (C 宏), 204
ESP_GATT_UUID_ENV_SENSING_CONFIG_DESCR (C 宏), 205
ESP_GATT_UUID_ENV_SENSING_MEASUREMENT_DESCR (C 宏), 206
ESP_GATT_UUID_ENV_SENSING_TRIGGER_DESCR (C 宏), 206
ESP_GATT_UUID_ENVIRONMENTAL_SENSING_SVC (C 宏), 205
ESP_GATT_UUID_EXT_RPT_REF_DESCR (C 宏), 205
ESP_GATT_UUID_FW_VERSION_STR (C 宏), 206
ESP_GATT_UUID_GAP_CENTRAL_ADDR_RESOL (C 宏), 206
ESP_GATT_UUID_GAP_DEVICE_NAME (C 宏), 206
ESP_GATT_UUID_GAP_ICON (C 宏), 206
ESP_GATT_UUID_GAP_PREF_CONN_PARAM (C 宏), 206
ESP_GATT_UUID_GATT_SRV_CHGD (C 宏), 206
ESP_GATT_UUID_GLUCOSE_SVC (C 宏), 204
ESP_GATT_UUID_GM_CONTEXT (C 宏), 206
ESP_GATT_UUID_GM_CONTROL_POINT (C 宏), 206
ESP_GATT_UUID_GM_FEATURE (C 宏), 206
ESP_GATT_UUID_GM_MEASUREMENT (C 宏), 206
ESP_GATT_UUID_HEALTH_THERMOM_SVC (C 宏), 204
ESP_GATT_UUID_HEART_RATE_SVC (C 宏), 204
ESP_GATT_UUID_HID_BT_KB_INPUT (C 宏), 207
ESP_GATT_UUID_HID_BT_KB_OUTPUT (C 宏), 207
ESP_GATT_UUID_HID_BT_MOUSE_INPUT (C 宏), 207
ESP_GATT_UUID_HID_CONTROL_POINT (C 宏), 207
ESP_GATT_UUID_HID_INFORMATION (C 宏), 207
ESP_GATT_UUID_HID_PROTO_MODE (C 宏), 207
ESP_GATT_UUID_HID_REPORT (C 宏), 207
ESP_GATT_UUID_HID_REPORT_MAP (C 宏), 207
ESP_GATT_UUID_HID_SVC (C 宏), 205
ESP_GATT_UUID_HW_VERSION_STR (C 宏), 206
ESP_GATT_UUID_IEEE_DATA (C 宏), 206
ESP_GATT_UUID_IMMEDIATE_ALERT_SVC (C 宏), 204
ESP_GATT_UUID_INCLUDE_SERVICE (C 宏), 205
ESP_GATT_UUID_LINK_LOSS_SVC (C 宏), 204
ESP_GATT_UUID_LOCAL_TIME_INFO (C 宏), 206
ESP_GATT_UUID_LOCATION_AND_NAVIGATION_SVC (C 宏), 205
ESP_GATT_UUID_MANU_NAME (C 宏), 206
ESP_GATT_UUID_MODEL_NUMBER_STR (C 宏), 206
ESP_GATT_UUID_NEXT_DST_CHANGE_SVC (C 宏), 204
ESP_GATT_UUID_NUM_DIGITALS_DESCR (C 宏), 205
ESP_GATT_UUID_NW_STATUS (C 宏), 206
ESP_GATT_UUID_NW_TRIGGER (C 宏), 206
ESP_GATT_UUID_PHONE_ALERT_STATUS_SVC (C 宏), 204
ESP_GATT_UUID_PNP_ID (C 宏), 207
ESP_GATT_UUID_PRI_SERVICE (C 宏), 205
ESP_GATT_UUID_REF_TIME_INFO (C 宏), 206
ESP_GATT_UUID_REF_TIME_UPDATE_SVC (C 宏), 204
ESP_GATT_UUID_RINGER_CP (C 宏), 206
ESP_GATT_UUID_RINGER_SETTING (C 宏), 206
ESP_GATT_UUID_RPT_REF_DESCR (C 宏), 205
ESP_GATT_UUID_RSC_FEATURE (C 宏), 207
ESP_GATT_UUID_RSC_MEASUREMENT (C 宏), 207
ESP_GATT_UUID_RUNNING_SPEED_CADENCE_SVC (C 宏), 205
ESP_GATT_UUID_SC_CONTROL_POINT (C 宏), 207
ESP_GATT_UUID_SCAN_INT_WINDOW (C 宏), 207
ESP_GATT_UUID_SCAN_PARAMETERS_SVC (C 宏), 205
ESP_GATT_UUID_SCAN_REFRESH (C 宏), 207
ESP_GATT_UUID_SEC_SERVICE (C 宏), 205

ESP_GATT_UUID_SENSOR_LOCATION (C 宏), 207
ESP_GATT_UUID_SERIAL_NUMBER_STR (C 宏), 206
ESP_GATT_UUID_SW_VERSION_STR (C 宏), 206
ESP_GATT_UUID_SYSTEM_ID (C 宏), 206
ESP_GATT_UUID_TIME_TRIGGER_DESCR (C 宏), 206
ESP_GATT_UUID_TX_POWER_LEVEL (C 宏), 206
ESP_GATT_UUID_TX_POWER_SVC (C 宏), 204
ESP_GATT_UUID_USER_DATA_SVC (C 宏), 205
ESP_GATT_UUID_VALUE_TRIGGER_DESCR (C 宏), 205
ESP_GATT_UUID_WEIGHT_SCALE_SVC (C 宏), 205
esp_gatt_value_t (C++ 类), 201
esp_gatt_value_t::auth_req (C++ 成员), 202
esp_gatt_value_t::handle (C++ 成员), 201
esp_gatt_value_t::len (C++ 成员), 202
esp_gatt_value_t::offset (C++ 成员), 201
esp_gatt_value_t::value (C++ 成员), 201
ESP_GATT_WRITE_NOT_PERMIT (C++ 枚举子), 209
ESP_GATT_WRITE_TYPE_NO_RSP (C++ 枚举子), 211
ESP_GATT_WRITE_TYPE_RSP (C++ 枚举子), 211
esp_gatt_write_type_t (C++ 类型), 211
ESP_GATT_WRONG_STATE (C++ 枚举子), 209
ESP_GATTC_ACL_EVT (C++ 枚举子), 251
ESP_GATTC_ADV_DATA_EVT (C++ 枚举子), 252
ESP_GATTC_ADV_VSC_EVT (C++ 枚举子), 253
ESP_GATTC_BTH_SCAN_CFG_EVT (C++ 枚举子), 252
ESP_GATTC_BTH_SCAN_DIS_EVT (C++ 枚举子), 252
ESP_GATTC_BTH_SCAN_ENB_EVT (C++ 枚举子), 252
ESP_GATTC_BTH_SCAN_PARAM_EVT (C++ 枚举子), 252
ESP_GATTC_BTH_SCAN_RD_EVT (C++ 枚举子), 252
ESP_GATTC_BTH_SCAN_THR_EVT (C++ 枚举子), 252
ESP_GATTC_CANCEL_OPEN_EVT (C++ 枚举子), 251
esp_gattc_cb_event_t (C++ 类型), 251
esp_gattc_cb_t (C++ 类型), 250
ESP_GATTC_CFG_MTU_EVT (C++ 枚举子), 252
esp_gattc_char_elem_t (C++ 类), 203
esp_gattc_char_elem_t::char_handle (C++ 成员), 203
esp_gattc_char_elem_t::properties (C++ 成员), 203
esp_gattc_char_elem_t::uuid (C++ 成员), 203
ESP_GATTC_CLOSE_EVT (C++ 枚举子), 251
ESP_GATTC_CONGEST_EVT (C++ 枚举子), 252
ESP_GATTC_CONNECT_EVT (C++ 枚举子), 253
esp_gattc_db_elem_t (C++ 类), 202
esp_gattc_db_elem_t::attribute_handle (C++ 成员), 202
esp_gattc_db_elem_t::end_handle (C++ 成员), 203
esp_gattc_db_elem_t::properties (C++ 成员), 203
esp_gattc_db_elem_t::start_handle (C++ 成员), 202
esp_gattc_db_elem_t::type (C++ 成员), 202
esp_gattc_db_elem_t::uuid (C++ 成员), 203
esp_gattc_descr_elem_t (C++ 类), 203
esp_gattc_descr_elem_t::handle (C++ 成员), 204
esp_gattc_descr_elem_t::uuid (C++ 成员), 204
ESP_GATTC_DIS_SRVC_CMPL_EVT (C++ 枚举子), 253
ESP_GATTC_DISCONNECT_EVT (C++ 枚举子), 253
ESP_GATTC_ENC_CMPL_CB_EVT (C++ 枚举子), 252
ESP_GATTC_EXEC_EVT (C++ 枚举子), 251
ESP_GATTC_GET_ADDR_LIST_EVT (C++ 枚举子), 253
esp_gattc_incl_svc_elem_t (C++ 类), 204
esp_gattc_incl_svc_elem_t::handle (C++ 成员), 204
esp_gattc_incl_svc_elem_t::incl_srvc_e_handle (C++ 成员), 204
esp_gattc_incl_svc_elem_t::incl_srvc_s_handle (C++ 成员), 204
esp_gattc_incl_svc_elem_t::uuid (C++ 成员), 204
ESP_GATTC_MULT_ADV_DATA_EVT (C++ 枚举子), 252
ESP_GATTC_MULT_ADV_DIS_EVT (C++ 枚举子), 252
ESP_GATTC_MULT_ADV_ENB_EVT (C++ 枚举子), 252
ESP_GATTC_MULT_ADV_UPD_EVT (C++ 枚举子), 252
esp_gattc_multi_t (C++ 类), 202
esp_gattc_multi_t::handles (C++ 成员), 202
esp_gattc_multi_t::num_attr (C++ 成员), 202
ESP_GATTC_NOTIFY_EVT (C++ 枚举子), 251
ESP_GATTC_OPEN_EVT (C++ 枚举子), 251
ESP_GATTC_PREP_WRITE_EVT (C++ 枚举子), 251
ESP_GATTC_QUEUE_FULL_EVT (C++ 枚举子), 253

ESP_GATTC_READ_CHAR_EVT (C++ 枚举子), 251
ESP_GATTC_READ_DESCR_EVT (C++ 枚举子), 251
ESP_GATTC_READ_MULTIPLE_EVT (C++ 枚举子), 253
ESP_GATTC_REG_EVT (C++ 枚举子), 251
ESP_GATTC_REG_FOR_NOTIFY_EVT (C++ 枚举子), 253
ESP_GATTC_SCAN_FLT_CFG_EVT (C++ 枚举子), 252
ESP_GATTC_SCAN_FLT_PARAM_EVT (C++ 枚举子), 252
ESP_GATTC_SCAN_FLT_STATUS_EVT (C++ 枚举子), 253
ESP_GATTC_SEARCH_CMPL_EVT (C++ 枚举子), 251
ESP_GATTC_SEARCH_RES_EVT (C++ 枚举子), 251
esp_gattc_service_elem_t (C++ 类), 203
esp_gattc_service_elem_t::end_handle (C++ 成员), 203
esp_gattc_service_elem_t::is_primary (C++ 成员), 203
esp_gattc_service_elem_t::start_handle (C++ 成员), 203
esp_gattc_service_elem_t::uuid (C++ 成员), 203
ESP_GATTC_SET_ASSOC_EVT (C++ 枚举子), 253
ESP_GATTC_SRVC_CHG_EVT (C++ 枚举子), 252
ESP_GATTC_UNREG_EVT (C++ 枚举子), 251
ESP_GATTC_UNREG_FOR_NOTIFY_EVT (C++ 枚举子), 253
ESP_GATTC_WRITE_CHAR_EVT (C++ 枚举子), 251
ESP_GATTC_WRITE_DESCR_EVT (C++ 枚举子), 251
ESP_GATTS_ADD_CHAR_DESCR_EVT (C++ 枚举子), 229
ESP_GATTS_ADD_CHAR_EVT (C++ 枚举子), 228
ESP_GATTS_ADD_INCL_SRVC_EVT (C++ 枚举子), 228
esp_gatts_attr_db_t (C++ 类), 200
esp_gatts_attr_db_t::att_desc (C++ 成员), 200
esp_gatts_attr_db_t::attr_control (C++ 成员), 200
ESP_GATTS_CANCEL_OPEN_EVT (C++ 枚举子), 229
esp_gatts_cb_event_t (C++ 类型), 228
esp_gatts_cb_t (C++ 类型), 228
ESP_GATTS_CLOSE_EVT (C++ 枚举子), 229
ESP_GATTS_CONF_EVT (C++ 枚举子), 228
ESP_GATTS_CONGEST_EVT (C++ 枚举子), 229
ESP_GATTS_CONNECT_EVT (C++ 枚举子), 229
ESP_GATTS_CREAT_ATTR_TAB_EVT (C++ 枚举子), 229
ESP_GATTS_CREATE_EVT (C++ 枚举子), 228
ESP_GATTS_DELETE_EVT (C++ 枚举子), 229
ESP_GATTS_DISCONNECT_EVT (C++ 枚举子), 229
ESP_GATTS_EXEC_WRITE_EVT (C++ 枚举子), 228
esp_gatts_incl128_svc_desc_t (C++ 类), 201
esp_gatts_incl128_svc_desc_t::end_hdl (C++ 成员), 201
esp_gatts_incl128_svc_desc_t::start_hdl (C++ 成员), 201
esp_gatts_incl_svc_desc_t (C++ 类), 201
esp_gatts_incl_svc_desc_t::end_hdl (C++ 成员), 201
esp_gatts_incl_svc_desc_t::start_hdl (C++ 成员), 201
esp_gatts_incl_svc_desc_t::uuid (C++ 成员), 201
ESP_GATTS_LISTEN_EVT (C++ 枚举子), 229
ESP_GATTS_MTU_EVT (C++ 枚举子), 228
ESP_GATTS_OPEN_EVT (C++ 枚举子), 229
ESP_GATTS_READ_EVT (C++ 枚举子), 228
ESP_GATTS_REG_EVT (C++ 枚举子), 228
ESP_GATTS_RESPONSE_EVT (C++ 枚举子), 229
ESP_GATTS_SEND_SERVICE_CHANGE_EVT (C++ 枚举子), 229
ESP_GATTS_SET_ATTR_VAL_EVT (C++ 枚举子), 229
ESP_GATTS_START_EVT (C++ 枚举子), 229
ESP_GATTS_STOP_EVT (C++ 枚举子), 229
ESP_GATTS_UNREG_EVT (C++ 枚举子), 228
ESP_GATTS_WRITE_EVT (C++ 枚举子), 228
esp_gcov_dump (C++ 函数), 1426
esp_get_deep_sleep_wake_stub (C++ 函数), 1723
esp_get_flash_encryption_mode (C++ 函数), 1333
esp_get_free_heap_size (C++ 函数), 1682
esp_get_idf_version (C++ 函数), 1687
esp_get_minimum_free_heap_size (C++ 函数), 1682
esp_hf_answer_call_cmd_callback (C++ 类型),

329
ESP_HF_AT_RESPONSE_CODE_BLACKLISTED (C++ 枚举子), 336
ESP_HF_AT_RESPONSE_CODE_BUSY (C++ 枚举子), 336
ESP_HF_AT_RESPONSE_CODE_CME (C++ 枚举子), 336
ESP_HF_AT_RESPONSE_CODE_DELAYED (C++ 枚举子), 336
ESP_HF_AT_RESPONSE_CODE_ERR (C++ 枚举子), 336
ESP_HF_AT_RESPONSE_CODE_NO_ANSWER (C++ 枚举子), 336
ESP_HF_AT_RESPONSE_CODE_NO_CARRIER (C++ 枚举子), 336
ESP_HF_AT_RESPONSE_CODE_OK (C++ 枚举子), 336
esp_hf_at_response_code_t (C++ 类型), 336
ESP_HF_AT_RESPONSE_ERROR (C++ 枚举子), 336
ESP_HF_AT_RESPONSE_OK (C++ 枚举子), 337
esp_hf_at_response_t (C++ 类型), 336
ESP_HF_ATA_RESPONSE_EVT (C++ 枚举子), 370
esp_hf_audio_state_callback (C++ 类型), 329
ESP_HF_AUDIO_STATE_CONNECTED (C++ 枚举子), 331
ESP_HF_AUDIO_STATE_CONNECTED_MSBC (C++ 枚举子), 331
ESP_HF_AUDIO_STATE_CONNECTING (C++ 枚举子), 331
ESP_HF_AUDIO_STATE_DISCONNECTED (C++ 枚举子), 331
ESP_HF_AUDIO_STATE_EVT (C++ 枚举子), 369
esp_hf_audio_state_t (C++ 类型), 331
ESP_HF_BCS_RESPONSE_EVT (C++ 枚举子), 370
ESP_HF_BTRH_CMD_ACCEPT (C++ 枚举子), 334
ESP_HF_BTRH_CMD_HOLD (C++ 枚举子), 334
ESP_HF_BTRH_CMD_REJECT (C++ 枚举子), 335
esp_hf_btrh_cmd_t (C++ 类型), 334
ESP_HF_BTRH_STATUS_ACCEPTED (C++ 枚举子), 334
ESP_HF_BTRH_STATUS_HELD (C++ 枚举子), 334
ESP_HF_BTRH_STATUS_REJECTED (C++ 枚举子), 334
esp_hf_btrh_status_t (C++ 类型), 334
ESP_HF_BVRA_RESPONSE_EVT (C++ 枚举子), 369
ESP_HF_CALL_ADDR_TYPE_INTERNATIONAL (C++ 枚举子), 334
esp_hf_call_addr_type_t (C++ 类型), 334
ESP_HF_CALL_ADDR_TYPE_UNKNOWN (C++ 枚举子), 334
ESP_HF_CALL_HELD_STATUS_HELD (C++ 枚举子), 332
ESP_HF_CALL_HELD_STATUS_HELD_AND_ACTIVE (C++ 枚举子), 332
ESP_HF_CALL_HELD_STATUS_NONE (C++ 枚举子), 332
esp_hf_call_held_status_t (C++ 类型), 332
ESP_HF_CALL_SETUP_STATUS_IDLE (C++ 枚举子), 332
ESP_HF_CALL_SETUP_STATUS_INCOMING (C++ 枚举子), 332
ESP_HF_CALL_SETUP_STATUS_OUTGOING_ALERTING (C++ 枚举子), 332
ESP_HF_CALL_SETUP_STATUS_OUTGOING_DIALING (C++ 枚举子), 332
esp_hf_call_setup_status_t (C++ 类型), 332
ESP_HF_CALL_STATUS_CALL_IN_PROGRESS (C++ 枚举子), 332
ESP_HF_CALL_STATUS_NO_CALLS (C++ 枚举子), 332
esp_hf_call_status_t (C++ 类型), 331
ESP_HF_CALL_WAITING_ACTIVE (C++ 枚举子), 335
ESP_HF_CALL_WAITING_INACTIVE (C++ 枚举子), 335
esp_hf_call_waiting_status_t (C++ 类型), 335
esp_hf_cb_event_t (C++ 类型), 369
esp_hf_cb_param_t (C++ 类型), 364
esp_hf_cb_param_t::audio_stat (C++ 成员), 364
esp_hf_cb_param_t::bcs_rep (C++ 成员), 365
esp_hf_cb_param_t::cind (C++ 成员), 364
esp_hf_cb_param_t::conn_stat (C++ 成员), 364
esp_hf_cb_param_t::hf_audio_stat_param (C++ 类), 365
esp_hf_cb_param_t::hf_audio_stat_param::remote_addr (C++ 成员), 365
esp_hf_cb_param_t::hf_audio_stat_param::state (C++ 成员), 365
esp_hf_cb_param_t::hf_bcs_rep_param (C++ 类), 365
esp_hf_cb_param_t::hf_bcs_rep_param::mode

(C++ 成员), 365
esp_hf_cb_param_t::hf_cind_param (C++ 类), 365
esp_hf_cb_param_t::hf_cind_param::battery_level (C++ 成员), 366
esp_hf_cb_param_t::hf_cind_param::call_held_status (C++ 成员), 366
esp_hf_cb_param_t::hf_cind_param::call_setup_status (C++ 成员), 365
esp_hf_cb_param_t::hf_cind_param::call_status (C++ 成员), 365
esp_hf_cb_param_t::hf_cind_param::roam (C++ 成员), 366
esp_hf_cb_param_t::hf_cind_param::signal_strength (C++ 成员), 365
esp_hf_cb_param_t::hf_cind_param::svc (C++ 成员), 365
esp_hf_cb_param_t::hf_conn_stat_param (C++ 类), 366
esp_hf_cb_param_t::hf_conn_stat_param::chld_feat (C++ 成员), 366
esp_hf_cb_param_t::hf_conn_stat_param::peer_feat (C++ 成员), 366
esp_hf_cb_param_t::hf_conn_stat_param::remote_bda (C++ 成员), 366
esp_hf_cb_param_t::hf_conn_stat_param::state (C++ 成员), 366
esp_hf_cb_param_t::hf_nrec_param (C++ 类), 366
esp_hf_cb_param_t::hf_nrec_param::state (C++ 成员), 366
esp_hf_cb_param_t::hf_out_call_param (C++ 类), 366
esp_hf_cb_param_t::hf_out_call_param::num_or_loc (C++ 成员), 366
esp_hf_cb_param_t::hf_out_call_param::remote_addr (C++ 成员), 366
esp_hf_cb_param_t::hf_unat_rep_param (C++ 类), 366
esp_hf_cb_param_t::hf_unat_rep_param::unat (C++ 成员), 367
esp_hf_cb_param_t::hf_volume_control_param (C++ 类), 367
esp_hf_cb_param_t::hf_volume_control_param::type (C++ 成员), 367
esp_hf_cb_param_t::hf_volume_control_param::volume (C++ 成员), 367
esp_hf_cb_param_t::hf_vra_rep_param (C++ 类), 367
esp_hf_cb_param_t::hf_vra_rep_param::remote_addr (C++ 成员), 367
esp_hf_cb_param_t::hf_vra_rep_param::value (C++ 成员), 367
esp_hf_cb_param_t::hf_vts_rep_param (C++ 类), 367
esp_hf_cb_param_t::hf_vts_rep_param::code (C++ 成员), 367
esp_hf_cb_param_t::hf_wbs_rep_param (C++ 类), 367
esp_hf_cb_param_t::hf_wbs_rep_param::codec (C++ 成员), 367
esp_hf_cb_param_t::nrec (C++ 成员), 365
esp_hf_cb_param_t::out_call (C++ 成员), 364
esp_hf_cb_param_t::unat_rep (C++ 成员), 364
esp_hf_cb_param_t::volume_control (C++ 成员), 364
esp_hf_cb_param_t::vra_rep (C++ 成员), 364
esp_hf_cb_param_t::vts_rep (C++ 成员), 365
esp_hf_cb_param_t::wbs_rep (C++ 成员), 365
esp_hf_cb_t (C++ 类型), 369
esp_hf_chld_cmd_callback (C++ 类型), 330
ESP_HF_CHLD_FEAT_HOLD_ACC (C 宏), 368
ESP_HF_CHLD_FEAT_MERGE (C 宏), 368
ESP_HF_CHLD_FEAT_MERGE_DETACH (C 宏), 368
ESP_HF_CHLD_FEAT_PRIV_X (C 宏), 368
ESP_HF_CHLD_FEAT_REL (C 宏), 368
ESP_HF_CHLD_FEAT_REL_ACC (C 宏), 368
ESP_HF_CHLD_FEAT_REL_X (C 宏), 368
ESP_HF_CHLD_TYPE_HOLD_ACC (C++ 枚举子), 336
ESP_HF_CHLD_TYPE_MERGE (C++ 枚举子), 336
ESP_HF_CHLD_TYPE_MERGE_DETACH (C++ 枚举子), 336
ESP_HF_CHLD_TYPE_PRIV_X (C++ 枚举子), 336
ESP_HF_CHLD_TYPE_REL (C++ 枚举子), 336

ESP_HF_CHLD_TYPE_REL_ACC (C++ 枚举子), 336
ESP_HF_CHLD_TYPE_REL_X (C++ 枚举子), 336
esp_hf_chld_type_t (C++ 类型), 335
ESP_HF_CHUP_RESPONSE_EVT (C++ 枚举子), 370
esp_hf_cind_cmd_callback (C++ 类型), 330
ESP_HF_CIND_RESPONSE_EVT (C++ 枚举子), 369
esp_hf_clcc_cmd_callback (C++ 类型), 330
ESP_HF_CLCC_RESPONSE_EVT (C++ 枚举子), 370
esp_hf_client_answer_call (C++ 函数), 342
ESP_HF_CLIENT_AT_RESPONSE_EVT (C++ 枚举子), 354
ESP_HF_CLIENT_AUDIO_STATE_CONNECTED (C++ 枚举子), 353
ESP_HF_CLIENT_AUDIO_STATE_CONNECTED_MSBC (C++ 枚举子), 353
ESP_HF_CLIENT_AUDIO_STATE_CONNECTING (C++ 枚举子), 353
ESP_HF_CLIENT_AUDIO_STATE_DISCONNECTED (C++ 枚举子), 353
ESP_HF_CLIENT_AUDIO_STATE_EVT (C++ 枚举子), 353
esp_hf_client_audio_state_t (C++ 类型), 353
ESP_HF_CLIENT_BINP_EVT (C++ 枚举子), 354
ESP_HF_CLIENT_BSIR_EVT (C++ 枚举子), 354
ESP_HF_CLIENT_BTRH_EVT (C++ 枚举子), 354
ESP_HF_CLIENT_BVRA_EVT (C++ 枚举子), 353
esp_hf_client_cb_event_t (C++ 类型), 353
esp_hf_client_cb_param_t (C++ 类型), 345
esp_hf_client_cb_param_t::at_response (C++ 成员), 346
esp_hf_client_cb_param_t::audio_stat (C++ 成员), 345
esp_hf_client_cb_param_t::battery_level (C++ 成员), 346
esp_hf_client_cb_param_t::binp (C++ 成员), 346
esp_hf_client_cb_param_t::bsir (C++ 成员), 346
esp_hf_client_cb_param_t::btrh (C++ 成员), 346
esp_hf_client_cb_param_t::bvra (C++ 成员), 345
esp_hf_client_cb_param_t::call (C++ 成员), 346
esp_hf_client_cb_param_t::call_held (C++ 成员), 346
esp_hf_client_cb_param_t::call_setup (C++ 成员), 346
esp_hf_client_cb_param_t::ccwa (C++ 成员), 346
esp_hf_client_cb_param_t::clcc (C++ 成员), 346
esp_hf_client_cb_param_t::clip (C++ 成员), 346
esp_hf_client_cb_param_t::cnum (C++ 成员), 346
esp_hf_client_cb_param_t::conn_stat (C++ 成员), 345
esp_hf_client_cb_param_t::cops (C++ 成员), 346
esp_hf_client_cb_param_t::hf_client_at_response_param (C++ 类), 347
esp_hf_client_cb_param_t::hf_client_at_response_param::cme (C++ 成员), 347
esp_hf_client_cb_param_t::hf_client_at_response_param::cod (C++ 成员), 347
esp_hf_client_cb_param_t::hf_client_audio_stat_param (C++ 类), 347
esp_hf_client_cb_param_t::hf_client_audio_stat_param::remo (C++ 成员), 347
esp_hf_client_cb_param_t::hf_client_audio_stat_param::stat (C++ 成员), 347
esp_hf_client_cb_param_t::hf_client_battery_level_ind_para (C++ 类), 347
esp_hf_client_cb_param_t::hf_client_battery_level_ind_para (C++ 成员), 347
esp_hf_client_cb_param_t::hf_client_binp_param (C++ 类), 347
esp_hf_client_cb_param_t::hf_client_binp_param::number (C++ 成员), 347
esp_hf_client_cb_param_t::hf_client_bsirparam (C++ 类), 347
esp_hf_client_cb_param_t::hf_client_bsirparam::state (C++ 成员), 348

esp_hf_client_cb_param_t::hf_client_btrh_param (C++ 类), 348
esp_hf_client_cb_param_t::hf_client_btrh_param::status (C++ 成员), 348
esp_hf_client_cb_param_t::hf_client_bvra_param (C++ 类), 348
esp_hf_client_cb_param_t::hf_client_bvra_param::value (C++ 成员), 348
esp_hf_client_cb_param_t::hf_client_call_held_ind_param (C++ 类), 348
esp_hf_client_cb_param_t::hf_client_call_held_ind_param::status (C++ 成员), 348
esp_hf_client_cb_param_t::hf_client_call_ind_param (C++ 类), 348
esp_hf_client_cb_param_t::hf_client_call_ind_param::status (C++ 成员), 348
esp_hf_client_cb_param_t::hf_client_call_setup_ind_param (C++ 类), 348
esp_hf_client_cb_param_t::hf_client_call_setup_ind_param::status (C++ 成员), 348
esp_hf_client_cb_param_t::hf_client_ccwa_param (C++ 类), 349
esp_hf_client_cb_param_t::hf_client_ccwa_param::number (C++ 成员), 349
esp_hf_client_cb_param_t::hf_client_clcc_param (C++ 类), 349
esp_hf_client_cb_param_t::hf_client_clcc_param::dir (C++ 成员), 349
esp_hf_client_cb_param_t::hf_client_clcc_param::idx (C++ 成员), 349
esp_hf_client_cb_param_t::hf_client_clcc_param::mpty (C++ 成员), 349
esp_hf_client_cb_param_t::hf_client_clcc_param::number (C++ 成员), 349
esp_hf_client_cb_param_t::hf_client_clcc_param::status (C++ 成员), 349
esp_hf_client_cb_param_t::hf_client_clip_param (C++ 类), 349
esp_hf_client_cb_param_t::hf_client_clip_param::number (C++ 成员), 349
esp_hf_client_cb_param_t::hf_client_cnum_param (C++ 类), 349
esp_hf_client_cb_param_t::hf_client_cnum_param::number (C++ 成员), 349
esp_hf_client_cb_param_t::hf_client_cnum_param::type (C++ 成员), 349
esp_hf_client_cb_param_t::hf_client_conn_stat_param (C++ 类), 350
esp_hf_client_cb_param_t::hf_client_conn_stat_param::chld_ (C++ 成员), 350
esp_hf_client_cb_param_t::hf_client_conn_stat_param::peer_ (C++ 成员), 350
esp_hf_client_cb_param_t::hf_client_conn_stat_param::remot (C++ 成员), 350
esp_hf_client_cb_param_t::hf_client_conn_stat_param::state (C++ 成员), 350
esp_hf_client_cb_param_t::hf_client_current_operator_param (C++ 类), 350
esp_hf_client_cb_param_t::hf_client_current_operator_param (C++ 成员), 350
esp_hf_client_cb_param_t::hf_client_network_roaming_param (C++ 类), 350
esp_hf_client_cb_param_t::hf_client_network_roaming_param: (C++ 成员), 350
esp_hf_client_cb_param_t::hf_client_service_availability_p (C++ 类), 350
esp_hf_client_cb_param_t::hf_client_service_availability_p (C++ 成员), 350
esp_hf_client_cb_param_t::hf_client_signal_strength_ind_pa (C++ 类), 350
esp_hf_client_cb_param_t::hf_client_signal_strength_ind_pa (C++ 成员), 351
esp_hf_client_cb_param_t::hf_client_volume_control_param (C++ 类), 351
esp_hf_client_cb_param_t::hf_client_volume_control_param:: (C++ 成员), 351
esp_hf_client_cb_param_t::hf_client_volume_control_param:: (C++ 成员), 351
esp_hf_client_cb_param_t::roaming (C++ 成员), 346
esp_hf_client_cb_param_t::service_availability (C++ 成员), 346
esp_hf_client_cb_param_t::signal_strength (C++ 成员), 346

esp_hf_client_cb_param_t::volume_control (C++ 成员), 346
esp_hf_client_cb_t (C++ 类型), 352
ESP_HF_CLIENT_CCWA_EVT (C++ 枚举子), 354
ESP_HF_CLIENT_CHLD_FEAT_HOLD_ACC (C 宏), 351
ESP_HF_CLIENT_CHLD_FEAT_MERGE (C 宏), 351
ESP_HF_CLIENT_CHLD_FEAT_MERGE_DETACH (C 宏), 351
ESP_HF_CLIENT_CHLD_FEAT_PRIV_X (C 宏), 351
ESP_HF_CLIENT_CHLD_FEAT_REL (C 宏), 351
ESP_HF_CLIENT_CHLD_FEAT_REL_ACC (C 宏), 351
ESP_HF_CLIENT_CHLD_FEAT_REL_X (C 宏), 351
ESP_HF_CLIENT_CIND_BATTERY_LEVEL_EVT (C++ 枚举子), 354
ESP_HF_CLIENT_CIND_CALL_EVT (C++ 枚举子), 354
ESP_HF_CLIENT_CIND_CALL_HELD_EVT (C++ 枚举子), 354
ESP_HF_CLIENT_CIND_CALL_SETUP_EVT (C++ 枚举子), 354
ESP_HF_CLIENT_CIND_ROAMING_STATUS_EVT (C++ 枚举子), 354
ESP_HF_CLIENT_CIND_SERVICE_AVAILABILITY_EVT (C++ 枚举子), 354
ESP_HF_CLIENT_CIND_SIGNAL_STRENGTH_EVT (C++ 枚举子), 354
ESP_HF_CLIENT_CLCC_EVT (C++ 枚举子), 354
ESP_HF_CLIENT_CLIP_EVT (C++ 枚举子), 354
ESP_HF_CLIENT_CNUM_EVT (C++ 枚举子), 354
esp_hf_client_connect (C++ 函数), 339
esp_hf_client_connect_audio (C++ 函数), 340
ESP_HF_CLIENT_CONNECTION_STATE_CONNECTED (C++ 枚举子), 353
ESP_HF_CLIENT_CONNECTION_STATE_CONNECTING (C++ 枚举子), 353
ESP_HF_CLIENT_CONNECTION_STATE_DISCONNECTED (C++ 枚举子), 352
ESP_HF_CLIENT_CONNECTION_STATE_DISCONNECTING (C++ 枚举子), 353
ESP_HF_CLIENT_CONNECTION_STATE_EVT (C++ 枚举子), 353
ESP_HF_CLIENT_CONNECTION_STATE_SLC_CONNECTED (C++ 枚举子), 353
esp_hf_client_connection_state_t (C++ 类型), 352
ESP_HF_CLIENT_COPS_CURRENT_OPERATOR_EVT (C++ 枚举子), 354
esp_hf_client_deinit (C++ 函数), 339
esp_hf_client_dial (C++ 函数), 341
esp_hf_client_dial_memory (C++ 函数), 341
esp_hf_client_disconnect (C++ 函数), 339
esp_hf_client_disconnect_audio (C++ 函数), 340
esp_hf_client_in_band_ring_state_t (C++ 类型), 353
ESP_HF_CLIENT_IN_BAND_RINGTONE_NOT_PROVIDED (C++ 枚举子), 353
ESP_HF_CLIENT_IN_BAND_RINGTONE_PROVIDED (C++ 枚举子), 353
esp_hf_client_incoming_data_cb_t (C++ 类型), 352
esp_hf_client_init (C++ 函数), 339
esp_hf_client_outgoing_data_cb_t (C++ 类型), 352
esp_hf_client_outgoing_data_ready (C++ 函数), 345
esp_hf_client_pcm_resample (C++ 函数), 345
esp_hf_client_pcm_resample_deinit (C++ 函数), 345
esp_hf_client_pcm_resample_init (C++ 函数), 345
ESP_HF_CLIENT_PEER_FEAT_3WAY (C 宏), 351
ESP_HF_CLIENT_PEER_FEAT_CODEC (C 宏), 351
ESP_HF_CLIENT_PEER_FEAT_ECC (C 宏), 351
ESP_HF_CLIENT_PEER_FEAT_ECNR (C 宏), 351
ESP_HF_CLIENT_PEER_FEAT_ECS (C 宏), 351
ESP_HF_CLIENT_PEER_FEAT_EXTERR (C 宏), 351
ESP_HF_CLIENT_PEER_FEAT_INBAND (C 宏), 351
ESP_HF_CLIENT_PEER_FEAT_REJECT (C 宏), 351
ESP_HF_CLIENT_PEER_FEAT_VREC (C 宏), 351
ESP_HF_CLIENT_PEER_FEAT_VTAG (C 宏), 351
esp_hf_client_query_current_calls (C++ 函数), 343
esp_hf_client_query_current_operator_name (C++ 函数), 343

esp_hf_client_register_callback (C++ 函数), 338
esp_hf_client_register_data_callback (C++ 函数), 344
esp_hf_client_reject_call (C++ 函数), 342
esp_hf_client_request_last_voice_tag_number (C++ 函数), 344
esp_hf_client_retrieve_subscriber_info (C++ 函数), 343
ESP_HF_CLIENT_RING_IND_EVT (C++ 枚举子), 355
esp_hf_client_send_btrh_cmd (C++ 函数), 342
esp_hf_client_send_chld_cmd (C++ 函数), 342
esp_hf_client_send_dtmf (C++ 函数), 343
esp_hf_client_send_nrec (C++ 函数), 344
esp_hf_client_start_voice_recognition (C++ 函数), 340
esp_hf_client_stop_voice_recognition (C++ 函数), 340
ESP_HF_CLIENT_VOLUME_CONTROL_EVT (C++ 枚举子), 354
esp_hf_client_volume_update (C++ 函数), 341
ESP_HF_CME_AG_FAILURE (C++ 枚举子), 337
ESP_HF_CME_DIAL_STRING_TOO_LONG (C++ 枚举子), 338
esp_hf_cme_err_t (C++ 类型), 337
ESP_HF_CME_INCORRECT_PASSWORD (C++ 枚举子), 337
ESP_HF_CME_INVALID_CHARACTERS_IN_DIAL_STRING (C++ 枚举子), 338
ESP_HF_CME_INVALID_CHARACTERS_IN_TEXT_STRING (C++ 枚举子), 338
ESP_HF_CME_INVALID_INDEX (C++ 枚举子), 337
ESP_HF_CME_MEMEORY_FAILURE (C++ 枚举子), 337
ESP_HF_CME_MEMEORY_FULL (C++ 枚举子), 337
ESP_HF_CME_NETWORK_NOT_ALLOWED (C++ 枚举子), 338
ESP_HF_CME_NETWORK_TIMEOUT (C++ 枚举子), 338
ESP_HF_CME_NO_CONNECTION_TO_PHONE (C++ 枚举子), 337
ESP_HF_CME_NO_NETWORK_SERVICE (C++ 枚举子), 338
ESP_HF_CME_OPERATION_NOT_ALLOWED (C++ 枚举子), 337
ESP_HF_CME_OPERATION_NOT_SUPPORTED (C++ 枚举子), 337
ESP_HF_CME_PH_SIM_PIN_REQUIRED (C++ 枚举子), 337
ESP_HF_CME_SIM_BUSY (C++ 枚举子), 337
ESP_HF_CME_SIM_FAILURE (C++ 枚举子), 337
ESP_HF_CME_SIM_NOT_INSERTED (C++ 枚举子), 337
ESP_HF_CME_SIM_PIN2_REQUIRED (C++ 枚举子), 337
ESP_HF_CME_SIM_PIN_REQUIRED (C++ 枚举子), 337
ESP_HF_CME_SIM_PUK2_REQUIRED (C++ 枚举子), 337
ESP_HF_CME_SIM_PUK_REQUIRED (C++ 枚举子), 337
ESP_HF_CME_TEXT_STRING_TOO_LONG (C++ 枚举子), 338
esp_hf_cnum_cmd_callback (C++ 类型), 330
ESP_HF_CNUM_RESPONSE_EVT (C++ 枚举子), 370
esp_hf_connection_state_callback (C++ 类型), 329
ESP_HF_CONNECTION_STATE_CONNECTED (C++ 枚举子), 335
ESP_HF_CONNECTION_STATE_CONNECTING (C++ 枚举子), 335
ESP_HF_CONNECTION_STATE_DISCONNECTED (C++ 枚举子), 335
ESP_HF_CONNECTION_STATE_DISCONNECTING (C++ 枚举子), 335
ESP_HF_CONNECTION_STATE_EVT (C++ 枚举子), 369
ESP_HF_CONNECTION_STATE_SLC_CONNECTED (C++ 枚举子), 335
esp_hf_connection_state_t (C++ 类型), 335
esp_hf_cops_cmd_callback (C++ 类型), 330
ESP_HF_COPS_RESPONSE_EVT (C++ 枚举子), 369
ESP_HF_CURRENT_CALL_DIRECTION_INCOMING (C++ 枚举子), 333
ESP_HF_CURRENT_CALL_DIRECTION_OUTGOING (C++ 枚举子), 333
esp_hf_current_call_direction_t (C++ 类型), 333
ESP_HF_CURRENT_CALL_MODE_DATA (C++ 枚举子), 334

ESP_HF_CURRENT_CALL_MODE_FAX (C++ 枚举子), 334
esp_hf_current_call_mode_t (C++ 类型), 333
ESP_HF_CURRENT_CALL_MODE_VOICE (C++ 枚举子), 333
ESP_HF_CURRENT_CALL_MPTY_TYPE_MULTI (C++ 枚举子), 333
ESP_HF_CURRENT_CALL_MPTY_TYPE_SINGLE (C++ 枚举子), 333
esp_hf_current_call_mpty_type_t (C++ 类型), 333
ESP_HF_CURRENT_CALL_STATUS_ACTIVE (C++ 枚举子), 333
ESP_HF_CURRENT_CALL_STATUS_ALERTING (C++ 枚举子), 333
ESP_HF_CURRENT_CALL_STATUS_DIALING (C++ 枚举子), 333
ESP_HF_CURRENT_CALL_STATUS_HELD (C++ 枚举子), 333
ESP_HF_CURRENT_CALL_STATUS_HELD_BY_RESP_HOLD (C++ 枚举子), 333
ESP_HF_CURRENT_CALL_STATUS_INCOMING (C++ 枚举子), 333
esp_hf_current_call_status_t (C++ 类型), 332
ESP_HF_CURRENT_CALL_STATUS_WAITING (C++ 枚举子), 333
esp_hf_dial_call_cmd_callback (C++ 类型), 329
ESP_HF_DIAL_EVT (C++ 枚举子), 370
esp_hf_dtmf_cmd_callback (C++ 类型), 330
esp_hf_hangup_call_cmd_callback (C++ 类型), 329
esp_hf_in_band_ring_state_t (C++ 类型), 330
ESP_HF_IN_BAND_RINGTONE_NOT_PROVIDED (C++ 枚举子), 330
ESP_HF_IN_BAND_RINGTONE_PROVIDED (C++ 枚举子), 330
esp_hf_incoming_data_cb_t (C++ 类型), 368
ESP_HF_IND_UPDATE_EVT (C++ 枚举子), 369
esp_hf_key_pressed_cmd_callback (C++ 类型), 330
ESP_HF_NETWORK_STATE_AVAILABLE (C++ 枚举子), 331
ESP_HF_NETWORK_STATE_NOT_AVAILABLE (C++ 枚举子), 331
esp_hf_network_state_t (C++ 类型), 331
esp_hf_nrec_cmd_callback (C++ 类型), 330
ESP_HF_NREC_RESPONSE_EVT (C++ 枚举子), 370
ESP_HF_NREC_START (C++ 枚举子), 335
ESP_HF_NREC_STOP (C++ 枚举子), 335
esp_hf_nrec_t (C++ 类型), 335
esp_hf_outgoing_data_cb_t (C++ 类型), 368
esp_hf_outgoing_data_ready (C++ 函数), 364
ESP_HF_PEER_FEAT_3WAY (C 宏), 368
ESP_HF_PEER_FEAT_CODEC (C 宏), 368
ESP_HF_PEER_FEAT_ECC (C 宏), 368
ESP_HF_PEER_FEAT_ECNR (C 宏), 368
ESP_HF_PEER_FEAT_ECS (C 宏), 368
ESP_HF_PEER_FEAT_EXTERR (C 宏), 368
ESP_HF_PEER_FEAT_INBAND (C 宏), 368
ESP_HF_PEER_FEAT_REJECT (C 宏), 368
ESP_HF_PEER_FEAT_VREC (C 宏), 368
ESP_HF_PEER_FEAT_VTAG (C 宏), 368
ESP_HF_ROAMING_STATUS_ACTIVE (C++ 枚举子), 332
ESP_HF_ROAMING_STATUS_INACTIVE (C++ 枚举子), 332
esp_hf_roaming_status_t (C++ 类型), 332
ESP_HF_SERVICE_TYPE_HOME (C++ 枚举子), 331
ESP_HF_SERVICE_TYPE_ROAMING (C++ 枚举子), 331
esp_hf_service_type_t (C++ 类型), 331
ESP_HF_SUBSCRIBER_SERVICE_TYPE_FAX (C++ 枚举子), 334
esp_hf_subscriber_service_type_t (C++ 类型), 334
ESP_HF_SUBSCRIBER_SERVICE_TYPE_UNKNOWN (C++ 枚举子), 334
ESP_HF_SUBSCRIBER_SERVICE_TYPE_VOICE (C++ 枚举子), 334
esp_hf_unat_response (C++ 函数), 358
ESP_HF_UNAT_RESPONSE_EVT (C++ 枚举子), 369
esp_hf_unknown_at_cmd_callback (C++ 类型), 330
esp_hf_volume_cmd_callback (C++ 类型), 329
ESP_HF_VOLUME_CONTROL_EVT (C++ 枚举子), 369

ESP_HF_VOLUME_CONTROL_TARGET_MIC (C++ 枚举子), 331
ESP_HF_VOLUME_CONTROL_TARGET_SPK (C++ 枚举子), 331
esp_hf_volume_control_target_t (C++ 类型), 330
ESP_HF_VOLUME_TYPE_MIC (C++ 枚举子), 331
ESP_HF_VOLUME_TYPE_SPK (C++ 枚举子), 331
esp_hf_volume_type_t (C++ 类型), 331
esp_hf_vr_cmd_callback (C++ 类型), 329
ESP_HF_VR_STATE_DISABLED (C++ 枚举子), 330
ESP_HF_VR_STATE_ENABLED (C++ 枚举子), 330
esp_hf_vr_state_t (C++ 类型), 330
ESP_HF_VTS_RESPONSE_EVT (C++ 枚举子), 370
esp_hf_wbs_callback (C++ 类型), 330
esp_hf_wbs_config_t (C++ 类型), 335
ESP_HF_WBS_NO (C++ 枚举子), 335
ESP_HF_WBS_NONE (C++ 枚举子), 335
ESP_HF_WBS_RESPONSE_EVT (C++ 枚举子), 370
ESP_HF_WBS_YES (C++ 枚举子), 335
esp_himem_alloc (C++ 函数), 1655
esp_himem_alloc_map_range (C++ 函数), 1655
ESP_HIMEM_BLKSZ (C 宏), 1658
esp_himem_free (C++ 函数), 1656
esp_himem_free_map_range (C++ 函数), 1657
esp_himem_get_free_size (C++ 函数), 1657
esp_himem_get_phys_size (C++ 函数), 1657
esp_himem_handle_t (C++ 类型), 1658
esp_himem_map (C++ 函数), 1656
ESP_HIMEM_MAPFLAG_RO (C 宏), 1658
esp_himem_rangehandle_t (C++ 类型), 1658
esp_himem_reserved_area_size (C++ 函数), 1657
esp_himem_unmap (C++ 函数), 1657
esp_http_client_add_auth (C++ 函数), 1164
esp_http_client_auth_type_t (C++ 类型), 1169
esp_http_client_cleanup (C++ 函数), 1163
esp_http_client_close (C++ 函数), 1163
esp_http_client_config_t (C++ 类), 1165
esp_http_client_config_t::auth_type (C++ 成员), 1165
esp_http_client_config_t::buffer_size (C++ 成员), 1166
esp_http_client_config_t::buffer_size_tx (C++ 成员), 1166
esp_http_client_config_t::cert_pem (C++ 成员), 1165
esp_http_client_config_t::client_cert_pem (C++ 成员), 1165
esp_http_client_config_t::client_key_pem (C++ 成员), 1165
esp_http_client_config_t::disable_auto_redirect (C++ 成员), 1166
esp_http_client_config_t::event_handler (C++ 成员), 1166
esp_http_client_config_t::host (C++ 成员), 1165
esp_http_client_config_t::is_async (C++ 成员), 1166
esp_http_client_config_t::max_redirection_count (C++ 成员), 1166
esp_http_client_config_t::method (C++ 成员), 1166
esp_http_client_config_t::password (C++ 成员), 1165
esp_http_client_config_t::path (C++ 成员), 1165
esp_http_client_config_t::port (C++ 成员), 1165
esp_http_client_config_t::query (C++ 成员), 1165
esp_http_client_config_t::skip_cert_common_name_check (C++ 成员), 1166
esp_http_client_config_t::timeout_ms (C++ 成员), 1166
esp_http_client_config_t::transport_type (C++ 成员), 1166
esp_http_client_config_t::url (C++ 成员), 1165
esp_http_client_config_t::use_global_ca_store (C++ 成员), 1166
esp_http_client_config_t::user_data (C++ 成员), 1166
esp_http_client_config_t::username (C++ 成员), 1165

esp_http_client_delete_header (C++ 函数), 1160
esp_http_client_event (C++ 类), 1164
esp_http_client_event::client (C++ 成员), 1164
esp_http_client_event::data (C++ 成员), 1164
esp_http_client_event::data_len (C++ 成员), 1165
esp_http_client_event::event_id (C++ 成员), 1164
esp_http_client_event::header_key (C++ 成员), 1165
esp_http_client_event::header_value (C++ 成员), 1165
esp_http_client_event::user_data (C++ 成员), 1165
esp_http_client_event_handle_t (C++ 类型), 1167
esp_http_client_event_id_t (C++ 类型), 1167
esp_http_client_event_t (C++ 类型), 1167
esp_http_client_fetch_headers (C++ 函数), 1161
esp_http_client_get_content_length (C++ 函数), 1162
esp_http_client_get_header (C++ 函数), 1158
esp_http_client_get_password (C++ 函数), 1159
esp_http_client_get_post_field (C++ 函数), 1158
esp_http_client_get_status_code (C++ 函数), 1162
esp_http_client_get_transport_type (C++ 函数), 1163
esp_http_client_get_username (C++ 函数), 1159
esp_http_client_handle_t (C++ 类型), 1167
esp_http_client_init (C++ 函数), 1156
esp_http_client_is_chunked_response (C++ 函数), 1162
esp_http_client_is_complete_data_received (C++ 函数), 1164
esp_http_client_method_t (C++ 类型), 1168
esp_http_client_open (C++ 函数), 1161
esp_http_client_perform (C++ 函数), 1156
esp_http_client_read (C++ 函数), 1162
esp_http_client_set_authtype (C++ 函数), 1160
esp_http_client_set_header (C++ 函数), 1158
esp_http_client_set_method (C++ 函数), 1160
esp_http_client_set_password (C++ 函数), 1160
esp_http_client_set_post_field (C++ 函数), 1157
esp_http_client_set_redirection (C++ 函数), 1163
esp_http_client_set_url (C++ 函数), 1157
esp_http_client_set_username (C++ 函数), 1159
esp_http_client_transport_t (C++ 类型), 1168
esp_http_client_write (C++ 函数), 1161
esp_https_ota (C++ 函数), 1445
esp_https_ota_begin (C++ 函数), 1446
esp_https_ota_config_t (C++ 类), 1448
esp_https_ota_config_t::http_config (C++ 成员), 1448
esp_https_ota_finish (C++ 函数), 1447
esp_https_ota_get_image_len_read (C++ 函数), 1448
esp_https_ota_get_img_desc (C++ 函数), 1448
esp_https_ota_handle_t (C++ 类型), 1449
esp_https_ota_is_complete_data_received (C++ 函数), 1447
esp_https_ota_perform (C++ 函数), 1446
ESP_IDF_VERSION (C 宏), 1687
ESP_IDF_VERSION_MAJOR (C 宏), 1687
ESP_IDF_VERSION_MINOR (C 宏), 1687
ESP_IDF_VERSION_PATCH (C 宏), 1687
ESP_IDF_VERSION_VAL (C 宏), 1687
ESP_IMAGE_FLASH_SIZE_16MB (C++ 枚举子), 1420
ESP_IMAGE_FLASH_SIZE_1MB (C++ 枚举子), 1419
ESP_IMAGE_FLASH_SIZE_2MB (C++ 枚举子), 1419
ESP_IMAGE_FLASH_SIZE_4MB (C++ 枚举子), 1420
ESP_IMAGE_FLASH_SIZE_8MB (C++ 枚举子), 1420
ESP_IMAGE_FLASH_SIZE_MAX (C++ 枚举子), 1420
esp_image_flash_size_t (C++ 类型), 1419
ESP_IMAGE_HEADER_MAGIC (C 宏), 1418
esp_image_header_t (C++ 类), 1416
esp_image_header_t::chip_id (C++ 成员), 1417
esp_image_header_t::entry_addr (C++ 成员),

1417
esp_image_header_t::hash_appended (C++ 成员), 1417
esp_image_header_t::magic (C++ 成员), 1416
esp_image_header_t::min_chip_rev (C++ 成员), 1417
esp_image_header_t::reserved (C++ 成员), 1417
esp_image_header_t::segment_count (C++ 成员), 1416
esp_image_header_t::spi_mode (C++ 成员), 1417
esp_image_header_t::spi_pin_drv (C++ 成员), 1417
esp_image_header_t::spi_size (C++ 成员), 1417
esp_image_header_t::spi_speed (C++ 成员), 1417
esp_image_header_t::wp_pin (C++ 成员), 1417
ESP_IMAGE_MAX_SEGMENTS (C 宏), 1418
esp_image_segment_header_t (C++ 类), 1417
esp_image_segment_header_t::data_len (C++ 成员), 1417
esp_image_segment_header_t::load_addr (C++ 成员), 1417
esp_image_spi_freq_t (C++ 类型), 1419
ESP_IMAGE_SPI_MODE_DIO (C++ 枚举子), 1419
ESP_IMAGE_SPI_MODE_DOUT (C++ 枚举子), 1419
ESP_IMAGE_SPI_MODE_FAST_READ (C++ 枚举子), 1419
ESP_IMAGE_SPI_MODE_QIO (C++ 枚举子), 1419
ESP_IMAGE_SPI_MODE_QOUT (C++ 枚举子), 1419
ESP_IMAGE_SPI_MODE_SLOW_READ (C++ 枚举子), 1419
esp_image_spi_mode_t (C++ 类型), 1419
ESP_IMAGE_SPI_SPEED_20M (C++ 枚举子), 1419
ESP_IMAGE_SPI_SPEED_26M (C++ 枚举子), 1419
ESP_IMAGE_SPI_SPEED_40M (C++ 枚举子), 1419
ESP_IMAGE_SPI_SPEED_80M (C++ 枚举子), 1419
esp_int_wdt_init (C++ 函数), 1727
esp_intr_alloc (C++ 函数), 1665
esp_intr_alloc_intrstatus (C++ 函数), 1666
ESP_INTR_DISABLE (C 宏), 1670
esp_intr_disable (C++ 函数), 1668
ESP_INTR_ENABLE (C 宏), 1670
esp_intr_enable (C++ 函数), 1668
ESP_INTR_FLAG_EDGE (C 宏), 1669
ESP_INTR_FLAG_HIGH (C 宏), 1669
ESP_INTR_FLAG_INTRDISABLED (C 宏), 1669
ESP_INTR_FLAG_IRAM (C 宏), 1669
ESP_INTR_FLAG_LEVEL1 (C 宏), 1669
ESP_INTR_FLAG_LEVEL2 (C 宏), 1669
ESP_INTR_FLAG_LEVEL3 (C 宏), 1669
ESP_INTR_FLAG_LEVEL4 (C 宏), 1669
ESP_INTR_FLAG_LEVEL5 (C 宏), 1669
ESP_INTR_FLAG_LEVEL6 (C 宏), 1669
ESP_INTR_FLAG_LEVELMASK (C 宏), 1669
ESP_INTR_FLAG_LOWMED (C 宏), 1669
ESP_INTR_FLAG_NMI (C 宏), 1669
ESP_INTR_FLAG_SHARED (C 宏), 1669
esp_intr_free (C++ 函数), 1667
esp_intr_get_cpu (C++ 函数), 1667
esp_intr_get_intno (C++ 函数), 1667
esp_intr_mark_shared (C++ 函数), 1665
esp_intr_noniram_disable (C++ 函数), 1668
esp_intr_noniram_enable (C++ 函数), 1668
esp_intr_reserve (C++ 函数), 1665
esp_intr_set_in_iram (C++ 函数), 1668
ESP_IO_CAP_IN (C 宏), 188
ESP_IO_CAP_IO (C 宏), 188
ESP_IO_CAP_KBDISP (C 宏), 188
ESP_IO_CAP_NONE (C 宏), 188
ESP_IO_CAP_OUT (C 宏), 187
esp_ip4addr_aton (C++ 函数), 795
esp_ip4addr_ntoa (C++ 函数), 795
esp_ipc_call (C++ 函数), 1659
esp_ipc_call_blocking (C++ 函数), 1659
ESP_LE_AUTH_BOND (C 宏), 187
ESP_LE_AUTH_NO_BOND (C 宏), 187
ESP_LE_AUTH_REQ_BOND_MITM (C 宏), 187
ESP_LE_AUTH_REQ_MITM (C 宏), 187
ESP_LE_AUTH_REQ_SC_BOND (C 宏), 187
ESP_LE_AUTH_REQ_SC_MITM (C 宏), 187
ESP_LE_AUTH_REQ_SC_MITM_BOND (C 宏), 187
ESP_LE_AUTH_REQ_SC_ONLY (C 宏), 187
ESP_LE_KEY_LCSRK (C 宏), 187
ESP_LE_KEY_LENC (C 宏), 187

ESP_LE_KEY_LID (C 宏), 187
ESP_LE_KEY_LLK (C 宏), 187
ESP_LE_KEY_NONE (C 宏), 187
ESP_LE_KEY_PCSRK (C 宏), 187
ESP_LE_KEY_PENC (C 宏), 187
ESP_LE_KEY_PID (C 宏), 187
ESP_LE_KEY_PLK (C 宏), 187
esp_light_sleep_start (C++ 函数), 1722
ESP_LINE_ENDINGS_CR (C++ 枚举子), 1390
ESP_LINE_ENDINGS_CRLF (C++ 枚举子), 1390
ESP_LINE_ENDINGS_LF (C++ 枚举子), 1390
esp_line_endings_t (C++ 类型), 1390
esp_link_key (C++ 类型), 156
esp_local_ctrl_add_property (C++ 函数), 1246
esp_local_ctrl_config (C++ 类), 1251
esp_local_ctrl_config::handlers (C++ 成员), 1251
esp_local_ctrl_config::max_properties (C++ 成员), 1251
esp_local_ctrl_config::transport (C++ 成员), 1251
esp_local_ctrl_config::transport_config (C++ 成员), 1251
esp_local_ctrl_config_t (C++ 类型), 1252
esp_local_ctrl_get_property (C++ 函数), 1247
esp_local_ctrl_get_transport_ble (C++ 函数), 1246
esp_local_ctrl_get_transport_httpd (C++ 函数), 1246
esp_local_ctrl_handlers (C++ 类), 1249
esp_local_ctrl_handlers::get_prop_values (C++ 成员), 1250
esp_local_ctrl_handlers::set_prop_values (C++ 成员), 1250
esp_local_ctrl_handlers::usr_ctx (C++ 成员), 1251
esp_local_ctrl_handlers::usr_ctx_free_fn (C++ 成员), 1251
esp_local_ctrl_handlers_t (C++ 类型), 1252
esp_local_ctrl_prop (C++ 类), 1248
esp_local_ctrl_prop::ctx (C++ 成员), 1249
esp_local_ctrl_prop::ctx_free_fn (C++ 成员), 1249
esp_local_ctrl_prop::flags (C++ 成员), 1249
esp_local_ctrl_prop::name (C++ 成员), 1248
esp_local_ctrl_prop::size (C++ 成员), 1249
esp_local_ctrl_prop::type (C++ 成员), 1248
esp_local_ctrl_prop_t (C++ 类型), 1251
esp_local_ctrl_prop_val (C++ 类), 1249
esp_local_ctrl_prop_val::data (C++ 成员), 1249
esp_local_ctrl_prop_val::free_fn (C++ 成员), 1249
esp_local_ctrl_prop_val::size (C++ 成员), 1249
esp_local_ctrl_prop_val_t (C++ 类型), 1252
esp_local_ctrl_remove_property (C++ 函数), 1247
esp_local_ctrl_set_handler (C++ 函数), 1247
esp_local_ctrl_start (C++ 函数), 1246
esp_local_ctrl_stop (C++ 函数), 1246
ESP_LOCAL_CTRL_TRANSPORT_BLE (C 宏), 1251
esp_local_ctrl_transport_config_ble_t (C++ 类型), 1252
esp_local_ctrl_transport_config_httpd_t (C++ 类型), 1252
esp_local_ctrl_transport_config_t (C++ 类型), 1248
esp_local_ctrl_transport_config_t::ble (C++ 成员), 1248
esp_local_ctrl_transport_config_t::httpd (C++ 成员), 1248
ESP_LOCAL_CTRL_TRANSPORT_HTTPD (C 宏), 1251
esp_local_ctrl_transport_t (C++ 类型), 1252
ESP_LOG_BUFFER_CHAR (C 宏), 1675
ESP_LOG_BUFFER_CHAR_LEVEL (C 宏), 1674
ESP_LOG_BUFFER_HEX (C 宏), 1675
ESP_LOG_BUFFER_HEX_LEVEL (C 宏), 1674
ESP_LOG_BUFFER_HEXDUMP (C 宏), 1674
ESP_LOG_DEBUG (C++ 枚举子), 1677
ESP_LOG_EARLY_IMPL (C 宏), 1676
esp_log_early_timestamp (C++ 函数), 1673
ESP_LOG_ERROR (C++ 枚举子), 1677
ESP_LOG_INFO (C++ 枚举子), 1677

ESP_LOG_LEVEL_LOCAL (C 宏), 1676
esp_log_level_set (C++ 函数), 1672
esp_log_level_t (C++ 类型), 1677
ESP_LOG_NONE (C++ 枚举子), 1677
esp_log_set_vprintf (C++ 函数), 1673
esp_log_system_timestamp (C++ 函数), 1673
esp_log_timestamp (C++ 函数), 1673
ESP_LOG_VERBOSE (C++ 枚举子), 1677
ESP_LOG_WARN (C++ 枚举子), 1677
esp_log_write (C++ 函数), 1674
esp_log_writev (C++ 函数), 1674
ESP_LOGD (C 宏), 1676
ESP_LOGE (C 宏), 1676
ESP_LOGI (C 宏), 1676
ESP_LOGV (C 宏), 1676
ESP_LOGW (C 宏), 1676
ESP_MAC_BT (C++ 枚举子), 1686
ESP_MAC_ETH (C++ 枚举子), 1686
esp_mac_type_t (C++ 类型), 1686
ESP_MAC_WIFI_SOFTAP (C++ 枚举子), 1686
ESP_MAC_WIFI_STA (C++ 枚举子), 1686
esp_mesh_allow_root_conflicts (C++ 函数), 735
esp_mesh_available_txupQ_num (C++ 函数), 734
esp_mesh_connect (C++ 函数), 741
esp_mesh_deinit (C++ 函数), 721
esp_mesh_delete_group_id (C++ 函数), 735
esp_mesh_disconnect (C++ 函数), 740
esp_mesh_fix_root (C++ 函数), 738
esp_mesh_flush_scan_result (C++ 函数), 741
esp_mesh_flush_upstream_packets (C++ 函数), 740
esp_mesh_get_ap_assoc_expire (C++ 函数), 733
esp_mesh_get_ap_authmode (C++ 函数), 729
esp_mesh_get_ap_connections (C++ 函数), 730
esp_mesh_get_capacity_num (C++ 函数), 736
esp_mesh_get_config (C++ 函数), 727
esp_mesh_get_group_list (C++ 函数), 736
esp_mesh_get_group_num (C++ 函数), 736
esp_mesh_get_id (C++ 函数), 728
esp_mesh_get_ie_crypto_key (C++ 函数), 737
esp_mesh_get_layer (C++ 函数), 730
esp_mesh_get_max_layer (C++ 函数), 729
esp_mesh_get_parent_bssid (C++ 函数), 730
esp_mesh_get_root_healing_delay (C++ 函数), 738
esp_mesh_get_router (C++ 函数), 727
esp_mesh_get_router_bssid (C++ 函数), 741
esp_mesh_get_routing_table (C++ 函数), 733
esp_mesh_get_routing_table_size (C++ 函数), 733
esp_mesh_get_rx_pending (C++ 函数), 734
esp_mesh_get_self_organized (C++ 函数), 731
esp_mesh_get_subnet_nodes_list (C++ 函数), 740
esp_mesh_get_subnet_nodes_num (C++ 函数), 740
esp_mesh_get_total_node_num (C++ 函数), 733
esp_mesh_get_tsf_time (C++ 函数), 742
esp_mesh_get_tx_pending (C++ 函数), 734
esp_mesh_get_type (C++ 函数), 728
esp_mesh_get_vote_percentage (C++ 函数), 732
esp_mesh_get_xon_qsize (C++ 函数), 735
esp_mesh_init (C++ 函数), 721
esp_mesh_is_my_group (C++ 函数), 736
esp_mesh_is_root (C++ 函数), 730
esp_mesh_is_root_conflicts_allowed (C++ 函数), 735
esp_mesh_is_root_fixed (C++ 函数), 738
esp_mesh_post_toDS_state (C++ 函数), 733
esp_mesh_recv (C++ 函数), 724
esp_mesh_recv_toDS (C++ 函数), 725
esp_mesh_scan_get_ap_ie_len (C++ 函数), 739
esp_mesh_scan_get_ap_record (C++ 函数), 739
esp_mesh_send (C++ 函数), 722
esp_mesh_set_ap_assoc_expire (C++ 函数), 732
esp_mesh_set_ap_authmode (C++ 函数), 729
esp_mesh_set_ap_connections (C++ 函数), 729
esp_mesh_set_ap_password (C++ 函数), 729
esp_mesh_set_capacity_num (C++ 函数), 736
esp_mesh_set_config (C++ 函数), 726
esp_mesh_set_group_id (C++ 函数), 735
esp_mesh_set_id (C++ 函数), 727
esp_mesh_set_ie_crypto_funcs (C++ 函数), 737
esp_mesh_set_ie_crypto_key (C++ 函数), 737
esp_mesh_set_max_layer (C++ 函数), 728

esp_mesh_set_parent (C++ 函数), 738
esp_mesh_set_root_healing_delay (C++ 函数), 737
esp_mesh_set_router (C++ 函数), 727
esp_mesh_set_self_organized (C++ 函数), 730
esp_mesh_set_type (C++ 函数), 728
esp_mesh_set_vote_percentage (C++ 函数), 732
esp_mesh_set_xon_qsize (C++ 函数), 734
esp_mesh_start (C++ 函数), 721
esp_mesh_stop (C++ 函数), 721
esp_mesh_switch_channel (C++ 函数), 741
esp_mesh_waive_root (C++ 函数), 731
esp_mqtt_client_config_t (C++ 类), 1228
esp_mqtt_client_config_t::alpn_protos (C++ 成员), 1231
esp_mqtt_client_config_t::buffer_size (C++ 成员), 1230
esp_mqtt_client_config_t::cert_len (C++ 成员), 1230
esp_mqtt_client_config_t::cert_pem (C++ 成员), 1230
esp_mqtt_client_config_t::client_cert_len (C++ 成员), 1230
esp_mqtt_client_config_t::client_cert_pem (C++ 成员), 1230
esp_mqtt_client_config_t::client_id (C++ 成员), 1229
esp_mqtt_client_config_t::client_key_len (C++ 成员), 1230
esp_mqtt_client_config_t::client_key_pem (C++ 成员), 1230
esp_mqtt_client_config_t::clientkey_password (C++ 成员), 1231
esp_mqtt_client_config_t::clientkey_password_len (C++ 成员), 1231
esp_mqtt_client_config_t::disable_auto_reconnect (C++ 成员), 1229
esp_mqtt_client_config_t::disable_clean_session (C++ 成员), 1229
esp_mqtt_client_config_t::event_handle (C++ 成员), 1229
esp_mqtt_client_config_t::event_loop_handle (C++ 成员), 1229
esp_mqtt_client_config_t::host (C++ 成员), 1229
esp_mqtt_client_config_t::keepalive (C++ 成员), 1229
esp_mqtt_client_config_t::lwt_msg (C++ 成员), 1229
esp_mqtt_client_config_t::lwt_msg_len (C++ 成员), 1229
esp_mqtt_client_config_t::lwt_qos (C++ 成员), 1229
esp_mqtt_client_config_t::lwt_retain (C++ 成员), 1229
esp_mqtt_client_config_t::lwt_topic (C++ 成员), 1229
esp_mqtt_client_config_t::out_buffer_size (C++ 成员), 1231
esp_mqtt_client_config_t::password (C++ 成员), 1229
esp_mqtt_client_config_t::port (C++ 成员), 1229
esp_mqtt_client_config_t::protocol_ver (C++ 成员), 1231
esp_mqtt_client_config_t::psk_hint_key (C++ 成员), 1230
esp_mqtt_client_config_t::reconnect_timeout_ms (C++ 成员), 1231
esp_mqtt_client_config_t::refresh_connection_after_ms (C++ 成员), 1230
esp_mqtt_client_config_t::task_prio (C++ 成员), 1230
esp_mqtt_client_config_t::task_stack (C++ 成员), 1230
esp_mqtt_client_config_t::transport (C++ 成员), 1230
esp_mqtt_client_config_t::uri (C++ 成员), 1229
esp_mqtt_client_config_t::use_global_ca_store (C++ 成员), 1230
esp_mqtt_client_config_t::user_context (C++ 成员), 1229
esp_mqtt_client_config_t::username (C++ 成

员), 1229
esp_mqtt_client_destroy (C++ 函数), 1226
esp_mqtt_client_disconnect (C++ 函数), 1225
esp_mqtt_client_handle_t (C++ 类型), 1231
esp_mqtt_client_init (C++ 函数), 1224
esp_mqtt_client_publish (C++ 函数), 1226
esp_mqtt_client_reconnect (C++ 函数), 1224
esp_mqtt_client_register_event (C++ 函数), 1227
esp_mqtt_client_set_uri (C++ 函数), 1224
esp_mqtt_client_start (C++ 函数), 1224
esp_mqtt_client_stop (C++ 函数), 1225
esp_mqtt_client_subscribe (C++ 函数), 1225
esp_mqtt_client_unsubscribe (C++ 函数), 1225
esp_mqtt_connect_return_code_t (C++ 类型), 1233
esp_mqtt_error_codes (C++ 类), 1227
esp_mqtt_error_codes::connect_return_code (C++ 成员), 1228
esp_mqtt_error_codes::error_type (C++ 成员), 1227
esp_mqtt_error_codes::esp_tls_cert_verify_flags (C++ 成员), 1227
esp_mqtt_error_codes::esp_tls_last_esp_err (C++ 成员), 1227
esp_mqtt_error_codes::esp_tls_stack_err (C++ 成员), 1227
esp_mqtt_error_codes_t (C++ 类型), 1231
esp_mqtt_error_type_t (C++ 类型), 1233
esp_mqtt_event_handle_t (C++ 类型), 1231
esp_mqtt_event_id_t (C++ 类型), 1232
esp_mqtt_event_t (C++ 类), 1228
esp_mqtt_event_t::client (C++ 成员), 1228
esp_mqtt_event_t::current_data_offset (C++ 成员), 1228
esp_mqtt_event_t::data (C++ 成员), 1228
esp_mqtt_event_t::data_len (C++ 成员), 1228
esp_mqtt_event_t::error_handle (C++ 成员), 1228
esp_mqtt_event_t::event_id (C++ 成员), 1228
esp_mqtt_event_t::msg_id (C++ 成员), 1228
esp_mqtt_event_t::session_present (C++ 成员), 1228
esp_mqtt_event_t::topic (C++ 成员), 1228
esp_mqtt_event_t::topic_len (C++ 成员), 1228
esp_mqtt_event_t::total_data_len (C++ 成员), 1228
esp_mqtt_event_t::user_context (C++ 成员), 1228
esp_mqtt_protocol_ver_t (C++ 类型), 1234
esp_mqtt_set_config (C++ 函数), 1226
esp_mqtt_transport_t (C++ 类型), 1233
esp_netif_action_connected (C++ 函数), 785
esp_netif_action_disconnected (C++ 函数), 786
esp_netif_action_got_ip (C++ 函数), 786
esp_netif_action_start (C++ 函数), 785
esp_netif_action_stop (C++ 函数), 785
esp_netif_attach (C++ 函数), 784
esp_netif_attach_wifi_ap (C++ 函数), 797
esp_netif_attach_wifi_station (C++ 函数), 797
esp_netif_create_default_wifi_ap (C++ 函数), 798
esp_netif_create_default_wifi_mesh_netifs (C++ 函数), 798
esp_netif_create_default_wifi_sta (C++ 函数), 798
esp_netif_create_ip6_linklocal (C++ 函数), 793
esp_netif_deinit (C++ 函数), 783
esp_netif_destroy (C++ 函数), 784
esp_netif_dhcpc_get_status (C++ 函数), 791
esp_netif_dhcpc_option (C++ 函数), 790
esp_netif_dhcpc_start (C++ 函数), 790
esp_netif_dhcpc_stop (C++ 函数), 791
esp_netif_dhcps_get_status (C++ 函数), 791
esp_netif_dhcps_option (C++ 函数), 789
esp_netif_dhcps_start (C++ 函数), 792
esp_netif_dhcps_stop (C++ 函数), 792
esp_netif_free_rx_buffer (C++ 函数), 803
esp_netif_get_desc (C++ 函数), 796
esp_netif_get_dns_info (C++ 函数), 793
esp_netif_get_event_id (C++ 函数), 796
esp_netif_get_flags (C++ 函数), 795
esp_netif_get_handle_from_ifkey (C++ 函数),

795
esp_netif_get_handle_from_netif_impl (C++ 函数), 802
esp_netif_get_hostname (C++ 函数), 787
esp_netif_get_ifkey (C++ 函数), 796
esp_netif_get_io_driver (C++ 函数), 795
esp_netif_get_ip6_global (C++ 函数), 794
esp_netif_get_ip6_linklocal (C++ 函数), 794
esp_netif_get_ip_info (C++ 函数), 787
esp_netif_get_netif_impl (C++ 函数), 802
esp_netif_get_netif_impl_index (C++ 函数), 789
esp_netif_get_nr_of_ifs (C++ 函数), 796
esp_netif_get_old_ip_info (C++ 函数), 788
esp_netif_init (C++ 函数), 783
esp_netif_is_netif_up (C++ 函数), 787
esp_netif_new (C++ 函数), 784
esp_netif_next (C++ 函数), 796
esp_netif_receive (C++ 函数), 785
esp_netif_set_dns_info (C++ 函数), 792
esp_netif_set_driver_config (C++ 函数), 784
esp_netif_set_hostname (C++ 函数), 787
esp_netif_set_ip4_addr (C++ 函数), 794
esp_netif_set_ip_info (C++ 函数), 788
esp_netif_set_mac (C++ 函数), 786
esp_netif_set_old_ip_info (C++ 函数), 789
esp_netif_transmit (C++ 函数), 803
esp_nimble_hci_and_controller_deinit (C++ 函数), 373
esp_nimble_hci_and_controller_init (C++ 函数), 372
esp_nimble_hci_deinit (C++ 函数), 373
esp_nimble_hci_init (C++ 函数), 372
esp_now_add_peer (C++ 函数), 708
esp_now_deinit (C++ 函数), 706
esp_now_del_peer (C++ 函数), 709
ESP_NOW_ETH_ALEN (C 宏), 712
esp_now_fetch_peer (C++ 函数), 710
esp_now_get_peer (C++ 函数), 709
esp_now_get_peer_num (C++ 函数), 710
esp_now_get_version (C++ 函数), 706
esp_now_init (C++ 函数), 706
esp_now_is_peer_exist (C++ 函数), 710
ESP_NOW_KEY_LEN (C 宏), 712
ESP_NOW_MAX_DATA_LEN (C 宏), 712
ESP_NOW_MAX_ENCRYPT_PEER_NUM (C 宏), 712
ESP_NOW_MAX_TOTAL_PEER_NUM (C 宏), 712
esp_now_mod_peer (C++ 函数), 709
esp_now_peer_info (C++ 类), 711
esp_now_peer_info::channel (C++ 成员), 711
esp_now_peer_info::encrypt (C++ 成员), 711
esp_now_peer_info::ifidx (C++ 成员), 711
esp_now_peer_info::lmk (C++ 成员), 711
esp_now_peer_info::peer_addr (C++ 成员), 711
esp_now_peer_info::priv (C++ 成员), 711
esp_now_peer_info_t (C++ 类型), 713
esp_now_peer_num (C++ 类), 711
esp_now_peer_num::encrypt_num (C++ 成员), 712
esp_now_peer_num::total_num (C++ 成员), 712
esp_now_peer_num_t (C++ 类型), 713
esp_now_recv_cb_t (C++ 类型), 713
esp_now_register_recv_cb (C++ 函数), 707
esp_now_register_send_cb (C++ 函数), 707
esp_now_send (C++ 函数), 708
esp_now_send_cb_t (C++ 类型), 713
ESP_NOW_SEND_FAIL (C++ 枚举子), 713
esp_now_send_status_t (C++ 类型), 713
ESP_NOW_SEND_SUCCESS (C++ 枚举子), 713
esp_now_set_pmk (C++ 函数), 710
esp_now_unregister_recv_cb (C++ 函数), 707
esp_now_unregister_send_cb (C++ 函数), 707
ESP_OK (C 宏), 1443
ESP_OK_EFUSE_CNT (C 宏), 1441
esp_ota_begin (C++ 函数), 1696
esp_ota_check_rollback_is_possible (C++ 函数), 1701
esp_ota_end (C++ 函数), 1698
esp_ota_erase_last_boot_app_partition (C++ 函数), 1701
esp_ota_get_app_description (C++ 函数), 1696
esp_ota_get_app_elf_sha256 (C++ 函数), 1696
esp_ota_get_boot_partition (C++ 函数), 1698
esp_ota_get_last_invalid_partition (C++ 函数), 1700

esp_ota_get_next_update_partition (C++ 函数), 1699
esp_ota_get_partition_description (C++ 函数), 1700
esp_ota_get_running_partition (C++ 函数), 1699
esp_ota_get_state_partition (C++ 函数), 1701
esp_ota_handle_t (C++ 类型), 1702
esp_ota_mark_app_invalid_rollback_and_reboot (C++ 函数), 1700
esp_ota_mark_app_valid_cancel_rollback (C++ 函数), 1700
esp_ota_set_boot_partition (C++ 函数), 1698
esp_ota_write (C++ 函数), 1697
esp_partition_check_identity (C++ 函数), 1327
esp_partition_deregister_external (C++ 函数), 1328
esp_partition_erase_range (C++ 函数), 1326
esp_partition_find (C++ 函数), 1323
esp_partition_find_first (C++ 函数), 1323
esp_partition_get (C++ 函数), 1324
esp_partition_get_sha256 (C++ 函数), 1327
esp_partition_iterator_release (C++ 函数), 1324
esp_partition_iterator_t (C++ 类型), 1330
esp_partition_mmap (C++ 函数), 1326
esp_partition_next (C++ 函数), 1324
esp_partition_read (C++ 函数), 1325
esp_partition_register_external (C++ 函数), 1328
ESP_PARTITION_SUBTYPE_ANY (C++ 枚举子), 1332
ESP_PARTITION_SUBTYPE_APP_FACTORY (C++ 枚举子), 1330
ESP_PARTITION_SUBTYPE_APP_OTA_0 (C++ 枚举子), 1330
ESP_PARTITION_SUBTYPE_APP_OTA_1 (C++ 枚举子), 1330
ESP_PARTITION_SUBTYPE_APP_OTA_10 (C++ 枚举子), 1331
ESP_PARTITION_SUBTYPE_APP_OTA_11 (C++ 枚举子), 1331
ESP_PARTITION_SUBTYPE_APP_OTA_12 (C++ 枚举子), 1331
ESP_PARTITION_SUBTYPE_APP_OTA_13 (C++ 枚举子), 1331
ESP_PARTITION_SUBTYPE_APP_OTA_14 (C++ 枚举子), 1331
ESP_PARTITION_SUBTYPE_APP_OTA_15 (C++ 枚举子), 1331
ESP_PARTITION_SUBTYPE_APP_OTA_2 (C++ 枚举子), 1330
ESP_PARTITION_SUBTYPE_APP_OTA_3 (C++ 枚举子), 1330
ESP_PARTITION_SUBTYPE_APP_OTA_4 (C++ 枚举子), 1331
ESP_PARTITION_SUBTYPE_APP_OTA_5 (C++ 枚举子), 1331
ESP_PARTITION_SUBTYPE_APP_OTA_6 (C++ 枚举子), 1331
ESP_PARTITION_SUBTYPE_APP_OTA_7 (C++ 枚举子), 1331
ESP_PARTITION_SUBTYPE_APP_OTA_8 (C++ 枚举子), 1331
ESP_PARTITION_SUBTYPE_APP_OTA_9 (C++ 枚举子), 1331
ESP_PARTITION_SUBTYPE_APP_OTA_MAX (C++ 枚举子), 1331
ESP_PARTITION_SUBTYPE_APP_OTA_MIN (C++ 枚举子), 1330
ESP_PARTITION_SUBTYPE_APP_TEST (C++ 枚举子), 1331
ESP_PARTITION_SUBTYPE_DATA_COREDUMP (C++ 枚举子), 1332
ESP_PARTITION_SUBTYPE_DATA_EFUSE_EM (C++ 枚举子), 1332
ESP_PARTITION_SUBTYPE_DATA_ESPHTTPD (C++ 枚举子), 1332
ESP_PARTITION_SUBTYPE_DATA_FAT (C++ 枚举子), 1332
ESP_PARTITION_SUBTYPE_DATA_NVS (C++ 枚举子), 1331
ESP_PARTITION_SUBTYPE_DATA_NVS_KEYS (C++ 枚举子), 1332
ESP_PARTITION_SUBTYPE_DATA_OTA (C++ 枚举子),

1331
ESP_PARTITION_SUBTYPE_DATA_PHY (C++ 枚举子), 1331
ESP_PARTITION_SUBTYPE_DATA_SPIFFS (C++ 枚举子), 1332
ESP_PARTITION_SUBTYPE_OTA (C 宏), 1330
esp_partition_subtype_t (C++ 类型), 1330
esp_partition_t (C++ 类), 1329
esp_partition_t::address (C++ 成员), 1329
esp_partition_t::encrypted (C++ 成员), 1329
esp_partition_t::flash_chip (C++ 成员), 1329
esp_partition_t::label (C++ 成员), 1329
esp_partition_t::size (C++ 成员), 1329
esp_partition_t::subtype (C++ 成员), 1329
esp_partition_t::type (C++ 成员), 1329
ESP_PARTITION_TYPE_APP (C++ 枚举子), 1330
ESP_PARTITION_TYPE_DATA (C++ 枚举子), 1330
esp_partition_type_t (C++ 类型), 1330
esp_partition_verify (C++ 函数), 1324
esp_partition_write (C++ 函数), 1325
ESP_PD_DOMAIN_MAX (C++ 枚举子), 1724
ESP_PD_DOMAIN_RTC_FAST_MEM (C++ 枚举子), 1724
ESP_PD_DOMAIN_RTC_PERIPH (C++ 枚举子), 1724
ESP_PD_DOMAIN_RTC_SLOW_MEM (C++ 枚举子), 1724
ESP_PD_DOMAIN_XTAL (C++ 枚举子), 1724
ESP_PD_OPTION_AUTO (C++ 枚举子), 1724
ESP_PD_OPTION_OFF (C++ 枚举子), 1724
ESP_PD_OPTION_ON (C++ 枚举子), 1724
esp_ping_callbacks_t (C++ 类), 1217
esp_ping_callbacks_t::cb_args (C++ 成员), 1217
esp_ping_callbacks_t::on_ping_end (C++ 成员), 1217
esp_ping_callbacks_t::on_ping_success (C++ 成员), 1217
esp_ping_callbacks_t::on_ping_timeout (C++ 成员), 1217
esp_ping_config_t (C++ 类), 1217
esp_ping_config_t::count (C++ 成员), 1217
esp_ping_config_t::data_size (C++ 成员), 1217
esp_ping_config_t::interval_ms (C++ 成员), 1217
esp_ping_config_t::target_addr (C++ 成员), 1217
esp_ping_config_t::task_prio (C++ 成员), 1217
esp_ping_config_t::task_stack_size (C++ 成员), 1217
esp_ping_config_t::timeout_ms (C++ 成员), 1217
esp_ping_config_t::tos (C++ 成员), 1217
ESP_PING_COUNT_INFINITE (C 宏), 1218
ESP_PING_DEFAULT_CONFIG (C 宏), 1218
esp_ping_delete_session (C++ 函数), 1215
esp_ping_get_profile (C++ 函数), 1216
esp_ping_handle_t (C++ 类型), 1218
esp_ping_new_session (C++ 函数), 1215
ESP_PING_PROF_DURATION (C++ 枚举子), 1218
ESP_PING_PROF_IPADDR (C++ 枚举子), 1218
ESP_PING_PROF_REPLY (C++ 枚举子), 1218
ESP_PING_PROF_REQUEST (C++ 枚举子), 1218
ESP_PING_PROF_SEQNO (C++ 枚举子), 1218
ESP_PING_PROF_SIZE (C++ 枚举子), 1218
ESP_PING_PROF_TIMEGAP (C++ 枚举子), 1218
ESP_PING_PROF_TTL (C++ 枚举子), 1218
esp_ping_profile_t (C++ 类型), 1218
esp_ping_start (C++ 函数), 1216
esp_ping_stop (C++ 函数), 1216
ESP_PM_APB_FREQ_MAX (C++ 枚举子), 1712
esp_pm_config_esp32_t (C++ 类), 1713
esp_pm_config_esp32_t::light_sleep_enable (C++ 成员), 1713
esp_pm_config_esp32_t::max_freq_mhz (C++ 成员), 1713
esp_pm_config_esp32_t::min_freq_mhz (C++ 成员), 1713
esp_pm_configure (C++ 函数), 1710
ESP_PM_CPU_FREQ_MAX (C++ 枚举子), 1712
esp_pm_dump_locks (C++ 函数), 1712
esp_pm_lock_acquire (C++ 函数), 1710
esp_pm_lock_create (C++ 函数), 1710
esp_pm_lock_delete (C++ 函数), 1711
esp_pm_lock_handle_t (C++ 类型), 1712
esp_pm_lock_release (C++ 函数), 1711
esp_pm_lock_type_t (C++ 类型), 1712

ESP_PM_NO_LIGHT_SLEEP (C++ 枚举子), 1713
esp_power_level_t (C++ 类型), 153
esp_pthread_cfg_t (C++ 类), 1451
esp_pthread_cfg_t::inherit_cfg (C++ 成员), 1452
esp_pthread_cfg_t::pin_to_core (C++ 成员), 1452
esp_pthread_cfg_t::prio (C++ 成员), 1452
esp_pthread_cfg_t::stack_size (C++ 成员), 1452
esp_pthread_cfg_t::thread_name (C++ 成员), 1452
esp_pthread_get_cfg (C++ 函数), 1451
esp_pthread_get_default_config (C++ 函数), 1450
esp_pthread_set_cfg (C++ 函数), 1451
ESP_PWR_LVL_N0 (C++ 枚举子), 153
ESP_PWR_LVL_N11 (C++ 枚举子), 154
ESP_PWR_LVL_N12 (C++ 枚举子), 153
ESP_PWR_LVL_N14 (C++ 枚举子), 154
ESP_PWR_LVL_N2 (C++ 枚举子), 154
ESP_PWR_LVL_N3 (C++ 枚举子), 153
ESP_PWR_LVL_N5 (C++ 枚举子), 154
ESP_PWR_LVL_N6 (C++ 枚举子), 153
ESP_PWR_LVL_N8 (C++ 枚举子), 154
ESP_PWR_LVL_N9 (C++ 枚举子), 153
ESP_PWR_LVL_P1 (C++ 枚举子), 154
ESP_PWR_LVL_P3 (C++ 枚举子), 153
ESP_PWR_LVL_P4 (C++ 枚举子), 154
ESP_PWR_LVL_P6 (C++ 枚举子), 153
ESP_PWR_LVL_P7 (C++ 枚举子), 154
ESP_PWR_LVL_P9 (C++ 枚举子), 154
esp_random (C++ 函数), 1682
esp_read_mac (C++ 函数), 1684
esp_register_freertos_idle_hook (C++ 函数), 1614
esp_register_freertos_idle_hook_for_cpu (C++ 函数), 1614
esp_register_freertos_tick_hook (C++ 函数), 1615
esp_register_freertos_tick_hook_for_cpu (C++ 函数), 1614
esp_register_shutdown_handler (C++ 函数), 1681
esp_reset_reason (C++ 函数), 1682
esp_reset_reason_t (C++ 类型), 1686
esp_restart (C++ 函数), 1682
ESP_RST_BROWNOUT (C++ 枚举子), 1686
ESP_RST_DEEPSLEEP (C++ 枚举子), 1686
ESP_RST_EXT (C++ 枚举子), 1686
ESP_RST_INT_WDT (C++ 枚举子), 1686
ESP_RST_PANIC (C++ 枚举子), 1686
ESP_RST_POWERON (C++ 枚举子), 1686
ESP_RST_SDIO (C++ 枚举子), 1686
ESP_RST_SW (C++ 枚举子), 1686
ESP_RST_TASK_WDT (C++ 枚举子), 1686
ESP_RST_UNKNOWN (C++ 枚举子), 1686
ESP_RST_WDT (C++ 枚举子), 1686
ESP_SCO_DATA_PATH_HCI (C++ 枚举子), 154
ESP_SCO_DATA_PATH_PCM (C++ 枚举子), 154
esp_sco_data_path_t (C++ 类型), 154
esp_service_source_t (C++ 类型), 211
esp_set_deep_sleep_wake_stub (C++ 函数), 1723
esp_sleep_disable_wakeup_source (C++ 函数), 1718
esp_sleep_enable_ext0_wakeup (C++ 函数), 1720
esp_sleep_enable_ext1_wakeup (C++ 函数), 1720
esp_sleep_enable_gpio_wakeup (C++ 函数), 1721
esp_sleep_enable_timer_wakeup (C++ 函数), 1719
esp_sleep_enable_touchpad_wakeup (C++ 函数), 1719
esp_sleep_enable_uart_wakeup (C++ 函数), 1721
esp_sleep_enable_ulp_wakeup (C++ 函数), 1718
esp_sleep_ext1_wakeup_mode_t (C++ 类型), 1724
esp_sleep_get_ext1_wakeup_status (C++ 函数), 1722
esp_sleep_get_touchpad_wakeup_status (C++ 函数), 1719
esp_sleep_get_wakeup_cause (C++ 函数), 1723
esp_sleep_pd_config (C++ 函数), 1722
esp_sleep_pd_domain_t (C++ 类型), 1724
esp_sleep_pd_option_t (C++ 类型), 1724
esp_sleep_source_t (C++ 类型), 1725

ESP_SLEEP_WAKEUP_ALL (C++ 枚举子), 1725
esp_sleep_wakeup_cause_t (C++ 类型), 1724
ESP_SLEEP_WAKEUP_EXT0 (C++ 枚举子), 1725
ESP_SLEEP_WAKEUP_EXT1 (C++ 枚举子), 1725
ESP_SLEEP_WAKEUP_GPIO (C++ 枚举子), 1725
ESP_SLEEP_WAKEUP_TIMER (C++ 枚举子), 1725
ESP_SLEEP_WAKEUP_TOUCHPAD (C++ 枚举子), 1725
ESP_SLEEP_WAKEUP_UART (C++ 枚举子), 1725
ESP_SLEEP_WAKEUP_ULP (C++ 枚举子), 1725
ESP_SLEEP_WAKEUP_UNDEFINED (C++ 枚举子), 1725
esp_smartconfig_fast_mode (C++ 函数), 702
esp_smartconfig_get_version (C++ 函数), 701
esp_smartconfig_set_type (C++ 函数), 702
esp_smartconfig_start (C++ 函数), 701
esp_smartconfig_stop (C++ 函数), 701
esp_spiffs_format (C++ 函数), 1407
esp_spiffs_info (C++ 函数), 1407
esp_spiffs_mounted (C++ 函数), 1406
ESP_SPP_BUSY (C++ 枚举子), 327
esp_spp_cb_event_t (C++ 类型), 328
esp_spp_cb_param_t (C++ 类型), 322
esp_spp_cb_param_t::cl_init (C++ 成员), 322
esp_spp_cb_param_t::close (C++ 成员), 322
esp_spp_cb_param_t::cong (C++ 成员), 323
esp_spp_cb_param_t::data_ind (C++ 成员), 322
esp_spp_cb_param_t::disc_comp (C++ 成员), 322
esp_spp_cb_param_t::init (C++ 成员), 322
esp_spp_cb_param_t::open (C++ 成员), 322
esp_spp_cb_param_t::spp_cl_init_evt_param (C++ 类), 323
esp_spp_cb_param_t::spp_cl_init_evt_param::handle (C++ 成员), 323
esp_spp_cb_param_t::spp_cl_init_evt_param::sec_id (C++ 成员), 323
esp_spp_cb_param_t::spp_cl_init_evt_param::status (C++ 成员), 323
esp_spp_cb_param_t::spp_cl_init_evt_param::use_co (C++ 成员), 323
esp_spp_cb_param_t::spp_close_evt_param (C++ 类), 323
esp_spp_cb_param_t::spp_close_evt_param::async (C++ 成员), 323
esp_spp_cb_param_t::spp_close_evt_param::handle (C++ 成员), 323
esp_spp_cb_param_t::spp_close_evt_param::port_status (C++ 成员), 323
esp_spp_cb_param_t::spp_close_evt_param::status (C++ 成员), 323
esp_spp_cb_param_t::spp_cong_evt_param (C++ 类), 323
esp_spp_cb_param_t::spp_cong_evt_param::cong (C++ 成员), 324
esp_spp_cb_param_t::spp_cong_evt_param::handle (C++ 成员), 323
esp_spp_cb_param_t::spp_cong_evt_param::status (C++ 成员), 323
esp_spp_cb_param_t::spp_data_ind_evt_param (C++ 类), 324
esp_spp_cb_param_t::spp_data_ind_evt_param::data (C++ 成员), 324
esp_spp_cb_param_t::spp_data_ind_evt_param::handle (C++ 成员), 324
esp_spp_cb_param_t::spp_data_ind_evt_param::len (C++ 成员), 324
esp_spp_cb_param_t::spp_data_ind_evt_param::status (C++ 成员), 324
esp_spp_cb_param_t::spp_discovery_comp_evt_param (C++ 类), 324
esp_spp_cb_param_t::spp_discovery_comp_evt_param::scn (C++ 成员), 324
esp_spp_cb_param_t::spp_discovery_comp_evt_param::scn_num (C++ 成员), 324
esp_spp_cb_param_t::spp_discovery_comp_evt_param::status (C++ 成员), 324
esp_spp_cb_param_t::spp_init_evt_param (C++ 类), 324
esp_spp_cb_param_t::spp_init_evt_param::status (C++ 成员), 324
esp_spp_cb_param_t::spp_open_evt_param (C++ 类), 324
esp_spp_cb_param_t::spp_open_evt_param::fd (C++ 成员), 325
esp_spp_cb_param_t::spp_open_evt_param::handle (C++ 成员), 325

esp_spp_cb_param_t::spp_open_evt_param::rem_bda (C++ 成员), 325
esp_spp_cb_param_t::spp_open_evt_param::status (C++ 成员), 325
esp_spp_cb_param_t::spp_srv_open_evt_param (C++ 类), 325
esp_spp_cb_param_t::spp_srv_open_evt_param::fd (C++ 成员), 325
esp_spp_cb_param_t::spp_srv_open_evt_param::handle (C++ 成员), 325
esp_spp_cb_param_t::spp_srv_open_evt_param::new_listen_handle (C++ 成员), 325
esp_spp_cb_param_t::spp_srv_open_evt_param::rem_bda (C++ 成员), 325
esp_spp_cb_param_t::spp_srv_open_evt_param::status (C++ 成员), 325
esp_spp_cb_param_t::spp_start_evt_param (C++ 类), 325
esp_spp_cb_param_t::spp_start_evt_param::handle (C++ 成员), 325
esp_spp_cb_param_t::spp_start_evt_param::sec_id (C++ 成员), 325
esp_spp_cb_param_t::spp_start_evt_param::status (C++ 成员), 325
esp_spp_cb_param_t::spp_start_evt_param::use_co (C++ 成员), 325
esp_spp_cb_param_t::spp_write_evt_param (C++ 类), 326
esp_spp_cb_param_t::spp_write_evt_param::cong (C++ 成员), 326
esp_spp_cb_param_t::spp_write_evt_param::handle (C++ 成员), 326
esp_spp_cb_param_t::spp_write_evt_param::len (C++ 成员), 326
esp_spp_cb_param_t::spp_write_evt_param::status (C++ 成员), 326
esp_spp_cb_param_t::srv_open (C++ 成员), 322
esp_spp_cb_param_t::start (C++ 成员), 322
esp_spp_cb_param_t::write (C++ 成员), 322
ESP_SPP_CL_INIT_EVT (C++ 枚举子), 328
ESP_SPP_CLOSE_EVT (C++ 枚举子), 328
ESP_SPP_CONG_EVT (C++ 枚举子), 328
esp_spp_connect (C++ 函数), 320
ESP_SPP_DATA_IND_EVT (C++ 枚举子), 328
esp_spp_deinit (C++ 函数), 320
esp_spp_disconnect (C++ 函数), 321
ESP_SPP_DISCOVERY_COMP_EVT (C++ 枚举子), 328
ESP_SPP_FAILURE (C++ 枚举子), 327
esp_spp_init (C++ 函数), 320
ESP_SPP_INIT_EVT (C++ 枚举子), 328
ESP_SPP_MAX_MTU (C 宏), 326
ESP_SPP_MAX_SCN (C 宏), 326
ESP_SPP_MODE_CB (C++ 枚举子), 327
esp_spp_mode_t (C++ 类型), 327
ESP_SPP_MODE_VFS (C++ 枚举子), 328
ESP_SPP_NO_DATA (C++ 枚举子), 327
ESP_SPP_NO_RESOURCE (C++ 枚举子), 327
ESP_SPP_OPEN_EVT (C++ 枚举子), 328
esp_spp_register_callback (C++ 函数), 319
ESP_SPP_ROLE_MASTER (C++ 枚举子), 327
ESP_SPP_ROLE_SLAVE (C++ 枚举子), 327
esp_spp_role_t (C++ 类型), 327
ESP_SPP_SEC_AUTHENTICATE (C 宏), 326
ESP_SPP_SEC_AUTHORIZE (C 宏), 326
ESP_SPP_SEC_ENCRYPT (C 宏), 326
ESP_SPP_SEC_IN_16_DIGITS (C 宏), 326
ESP_SPP_SEC_MITM (C 宏), 326
ESP_SPP_SEC_MODE4_LEVEL4 (C 宏), 326
ESP_SPP_SEC_NONE (C 宏), 326
esp_spp_sec_t (C++ 类型), 327
ESP_SPP_SRV_OPEN_EVT (C++ 枚举子), 328
esp_spp_start_discovery (C++ 函数), 320
ESP_SPP_START_EVT (C++ 枚举子), 328
esp_spp_start_srv (C++ 函数), 321
esp_spp_status_t (C++ 类型), 327
ESP_SPP_SUCCESS (C++ 枚举子), 327
esp_spp_vfs_register (C++ 函数), 322
esp_spp_write (C++ 函数), 321
ESP_SPP_WRITE_EVT (C++ 枚举子), 328
esp_switch_stack_enter (C++ 函数), 1661
esp_switch_stack_exit (C++ 函数), 1661
esp_switch_stack_setup (C++ 函数), 1661
esp_sysview_flush (C++ 函数), 1426
esp_sysview_heap_trace_alloc (C++ 函数), 1427

esp_sysview_heap_trace_free (C++ 函数), 1427
esp_sysview_heap_trace_start (C++ 函数), 1427
esp_sysview_heap_trace_stop (C++ 函数), 1427
esp_sysview_vprintf (C++ 函数), 1426
esp_task_wdt_add (C++ 函数), 1728
esp_task_wdt_deinit (C++ 函数), 1728
esp_task_wdt_delete (C++ 函数), 1729
esp_task_wdt_init (C++ 函数), 1727
esp_task_wdt_reset (C++ 函数), 1729
esp_task_wdt_status (C++ 函数), 1729
esp_timer_cb_t (C++ 类型), 1654
esp_timer_create (C++ 函数), 1651
esp_timer_create_args_t (C++ 类), 1653
esp_timer_create_args_t::arg (C++ 成员), 1654
esp_timer_create_args_t::callback (C++ 成员), 1654
esp_timer_create_args_t::dispatch_method (C++ 成员), 1654
esp_timer_create_args_t::name (C++ 成员), 1654
esp_timer_deinit (C++ 函数), 1650
esp_timer_delete (C++ 函数), 1652
esp_timer_dispatch_t (C++ 类型), 1654
esp_timer_dump (C++ 函数), 1653
esp_timer_get_next_alarm (C++ 函数), 1653
esp_timer_get_time (C++ 函数), 1652
esp_timer_handle_t (C++ 类型), 1654
esp_timer_init (C++ 函数), 1650
esp_timer_start_once (C++ 函数), 1651
esp_timer_start_periodic (C++ 函数), 1651
esp_timer_stop (C++ 函数), 1652
ESP_TIMER_TASK (C++ 枚举子), 1654
esp_tls (C++ 类), 1148
esp_tls::conn_state (C++ 成员), 1149
esp_tls::error_handle (C++ 成员), 1149
esp_tls::is_tls (C++ 成员), 1149
esp_tls::read (C++ 成员), 1148
esp_tls::role (C++ 成员), 1149
esp_tls::rset (C++ 成员), 1149
esp_tls::sockfd (C++ 成员), 1148
esp_tls::write (C++ 成员), 1148
esp_tls::wset (C++ 成员), 1149
esp_tls_cfg (C++ 类), 1146
esp_tls_cfg::alpn_protos (C++ 成员), 1147
esp_tls_cfg::cacert_buf (C++ 成员), 1147
esp_tls_cfg::cacert_bytes (C++ 成员), 1147
esp_tls_cfg::cacert_pem_buf (C++ 成员), 1147
esp_tls_cfg::cacert_pem_bytes (C++ 成员), 1147
esp_tls_cfg::clientcert_buf (C++ 成员), 1147
esp_tls_cfg::clientcert_bytes (C++ 成员), 1147
esp_tls_cfg::clientcert_pem_buf (C++ 成员), 1147
esp_tls_cfg::clientcert_pem_bytes (C++ 成员), 1147
esp_tls_cfg::clientkey_buf (C++ 成员), 1147
esp_tls_cfg::clientkey_bytes (C++ 成员), 1148
esp_tls_cfg::clientkey_password (C++ 成员), 1148
esp_tls_cfg::clientkey_password_len (C++ 成员), 1148
esp_tls_cfg::clientkey_pem_buf (C++ 成员), 1147
esp_tls_cfg::clientkey_pem_bytes (C++ 成员), 1148
esp_tls_cfg::common_name (C++ 成员), 1148
esp_tls_cfg::non_block (C++ 成员), 1148
esp_tls_cfg::psk_hint_key (C++ 成员), 1148
esp_tls_cfg::skip_common_name (C++ 成员), 1148
esp_tls_cfg::timeout_ms (C++ 成员), 1148
esp_tls_cfg::use_global_ca_store (C++ 成员), 1148
esp_tls_cfg_t (C++ 类型), 1150
ESP_TLS_CLIENT (C++ 枚举子), 1151
esp_tls_conn_delete (C++ 函数), 1144
esp_tls_conn_http_new (C++ 函数), 1142
esp_tls_conn_http_new_async (C++ 函数), 1143
esp_tls_conn_new (C++ 函数), 1141
esp_tls_conn_new_async (C++ 函数), 1142
esp_tls_conn_new_sync (C++ 函数), 1141
esp_tls_conn_read (C++ 函数), 1144
esp_tls_conn_state (C++ 类型), 1151

esp_tls_conn_state_t (C++ 类型), 1150
esp_tls_conn_write (C++ 函数), 1143
ESP_TLS_CONNECTING (C++ 枚举子), 1151
ESP_TLS_DONE (C++ 枚举子), 1151
esp_tls_error_handle_t (C++ 类型), 1150
ESP_TLS_FAIL (C++ 枚举子), 1151
esp_tls_free_global_ca_store (C++ 函数), 1145
esp_tls_get_and_clear_last_error (C++ 函数), 1145
esp_tls_get_bytes_avail (C++ 函数), 1144
ESP_TLS_HANDSHAKE (C++ 枚举子), 1151
esp_tls_init (C++ 函数), 1141
ESP_TLS_INIT (C++ 枚举子), 1151
esp_tls_init_global_ca_store (C++ 函数), 1144
esp_tls_last_error (C++ 类), 1146
esp_tls_last_error::esp_tls_error_code (C++ 成员), 1146
esp_tls_last_error::esp_tls_flags (C++ 成员), 1146
esp_tls_last_error::last_error (C++ 成员), 1146
esp_tls_last_error_t (C++ 类型), 1150
esp_tls_role (C++ 类型), 1151
esp_tls_role_t (C++ 类型), 1150
ESP_TLS_SERVER (C++ 枚举子), 1151
esp_tls_set_global_ca_store (C++ 函数), 1145
esp_tls_t (C++ 类型), 1151
esp_unregister_shutdown_handler (C++ 函数), 1682
ESP_UUID_LEN_128 (C 宏), 156
ESP_UUID_LEN_16 (C 宏), 156
ESP_UUID_LEN_32 (C 宏), 156
esp_vendor_ie_cb_t (C++ 类型), 677
esp_vfs_close (C++ 函数), 1383
esp_vfs_dev_uart_register (C++ 函数), 1388
esp_vfs_dev_uart_set_rx_line_endings (C++ 函数), 1388
esp_vfs_dev_uart_set_tx_line_endings (C++ 函数), 1389
esp_vfs_dev_uart_use_driver (C++ 函数), 1389
esp_vfs_dev_uart_use_nonblocking (C++ 函数), 1389
esp_vfs_fat_mount_config_t (C++ 类), 1393, 1399
esp_vfs_fat_mount_config_t::allocation_unit_size (C++ 成员), 1393, 1399
esp_vfs_fat_mount_config_t::format_if_mount_failed (C++ 成员), 1393, 1399
esp_vfs_fat_mount_config_t::max_files (C++ 成员), 1393, 1399
esp_vfs_fat_rawflash_mount (C++ 函数), 1394
esp_vfs_fat_rawflash_unmount (C++ 函数), 1394
esp_vfs_fat_register (C++ 函数), 1391
esp_vfs_fat_sdmmc_mount (C++ 函数), 1392
esp_vfs_fat_sdmmc_unmount (C++ 函数), 1393
esp_vfs_fat_spiflash_mount (C++ 函数), 1398
esp_vfs_fat_spiflash_unmount (C++ 函数), 1399
esp_vfs_fat_unregister_path (C++ 函数), 1391
ESP_VFS_FLAG_CONTEXT_PTR (C 宏), 1388
ESP_VFS_FLAG_DEFAULT (C 宏), 1388
esp_vfs_fstat (C++ 函数), 1383
esp_vfs_id_t (C++ 类型), 1388
esp_vfs_link (C++ 函数), 1383
esp_vfs_lseek (C++ 函数), 1382
esp_vfs_open (C++ 函数), 1383
ESP_VFS_PATH_MAX (C 宏), 1388
esp_vfs_poll (C++ 函数), 1386
esp_vfs_pread (C++ 函数), 1386
esp_vfs_pwrite (C++ 函数), 1386
esp_vfs_read (C++ 函数), 1383
esp_vfs_register (C++ 函数), 1383
esp_vfs_register_fd (C++ 函数), 1384
esp_vfs_register_fd_range (C++ 函数), 1383
esp_vfs_register_with_id (C++ 函数), 1384
esp_vfs_rename (C++ 函数), 1383
esp_vfs_select (C++ 函数), 1385
esp_vfs_select_sem_t (C++ 类), 1387
esp_vfs_select_sem_t::is_sem_local (C++ 成员), 1387
esp_vfs_select_sem_t::sem (C++ 成员), 1387
esp_vfs_select_triggered (C++ 函数), 1385
esp_vfs_select_triggered_isr (C++ 函数), 1385
esp_vfs_spiffs_conf_t (C++ 类), 1407
esp_vfs_spiffs_conf_t::base_path (C++ 成员),

1407
esp_vfs_spiffs_conf_t::format_if_mount_failed (C++ 成员), 1408
esp_vfs_spiffs_conf_t::max_files (C++ 成员), 1408
esp_vfs_spiffs_conf_t::partition_label (C++ 成员), 1407
esp_vfs_spiffs_register (C++ 函数), 1406
esp_vfs_spiffs_unregister (C++ 函数), 1406
esp_vfs_stat (C++ 函数), 1383
esp_vfs_t (C++ 类), 1387
esp_vfs_t::end_select (C++ 成员), 1388
esp_vfs_t::flags (C++ 成员), 1387
esp_vfs_t::get_socket_select_semaphore (C++ 成员), 1388
esp_vfs_t::socket_select (C++ 成员), 1387
esp_vfs_t::start_select (C++ 成员), 1387
esp_vfs_t::stop_socket_select (C++ 成员), 1387
esp_vfs_t::stop_socket_select_isr (C++ 成员), 1388
esp_vfs_unlink (C++ 函数), 1383
esp_vfs_unregister (C++ 函数), 1384
esp_vfs_unregister_fd (C++ 函数), 1384
esp_vfs_utime (C++ 函数), 1383
esp_vfs_write (C++ 函数), 1382
esp_vhci_host_callback (C++ 类), 150
esp_vhci_host_callback::notify_host_recv (C++ 成员), 151
esp_vhci_host_callback::notify_host_send_available (C++ 成员), 151
esp_vhci_host_callback_t (C++ 类型), 151
esp_vhci_host_check_send_available (C++ 函数), 146
esp_vhci_host_register_callback (C++ 函数), 146
esp_vhci_host_send_packet (C++ 函数), 146
esp_wake_deep_sleep (C++ 函数), 1723
esp_websocket_client_config_t (C++ 类), 1174
esp_websocket_client_config_t::buffer_size (C++ 成员), 1175
esp_websocket_client_config_t::cert_pem (C++ 成员), 1175
esp_websocket_client_config_t::disable_auto_reconnect (C++ 成员), 1175
esp_websocket_client_config_t::headers (C++ 成员), 1175
esp_websocket_client_config_t::host (C++ 成员), 1174
esp_websocket_client_config_t::password (C++ 成员), 1175
esp_websocket_client_config_t::path (C++ 成员), 1175
esp_websocket_client_config_t::port (C++ 成员), 1174
esp_websocket_client_config_t::subprotocol (C++ 成员), 1175
esp_websocket_client_config_t::task_prio (C++ 成员), 1175
esp_websocket_client_config_t::task_stack (C++ 成员), 1175
esp_websocket_client_config_t::transport (C++ 成员), 1175
esp_websocket_client_config_t::uri (C++ 成员), 1174
esp_websocket_client_config_t::user_agent (C++ 成员), 1175
esp_websocket_client_config_t::user_context (C++ 成员), 1175
esp_websocket_client_config_t::username (C++ 成员), 1174
esp_websocket_client_destroy (C++ 函数), 1172
esp_websocket_client_handle_t (C++ 类型), 1175
esp_websocket_client_init (C++ 函数), 1171
esp_websocket_client_is_connected (C++ 函数), 1173
esp_websocket_client_send (C++ 函数), 1172
esp_websocket_client_send_bin (C++ 函数), 1172
esp_websocket_client_send_text (C++ 函数), 1173
esp_websocket_client_set_uri (C++ 函数), 1171
esp_websocket_client_start (C++ 函数), 1171

esp_websocket_client_stop (C++ 函数), 1172
esp_websocket_event_data_t (C++ 类), 1174
esp_websocket_event_data_t::client (C++ 成员), 1174
esp_websocket_event_data_t::data_len (C++ 成员), 1174
esp_websocket_event_data_t::data_ptr (C++ 成员), 1174
esp_websocket_event_data_t::op_code (C++ 成员), 1174
esp_websocket_event_data_t::payload_len (C++ 成员), 1174
esp_websocket_event_data_t::payload_offset (C++ 成员), 1174
esp_websocket_event_data_t::user_context (C++ 成员), 1174
esp_websocket_event_id_t (C++ 类型), 1175
esp_websocket_register_events (C++ 函数), 1173
esp_websocket_transport_t (C++ 类型), 1176
esp_wifi_80211_tx (C++ 函数), 670
esp_wifi_ap_get_sta_aid (C++ 函数), 667
esp_wifi_ap_get_sta_list (C++ 函数), 667
esp_wifi_clear_default_wifi_driver_and_handlers (C++ 函数), 797
esp_wifi_clear_fast_connect (C++ 函数), 657
esp_wifi_connect (C++ 函数), 656
esp_wifi_deauth_sta (C++ 函数), 657
esp_wifi_deinit (C++ 函数), 654
esp_wifi_disconnect (C++ 函数), 657
esp_wifi_get_ant (C++ 函数), 673
esp_wifi_get_ant_gpio (C++ 函数), 672
esp_wifi_get_bandwidth (C++ 函数), 661
esp_wifi_get_channel (C++ 函数), 662
esp_wifi_get_config (C++ 函数), 667
esp_wifi_get_country (C++ 函数), 663
esp_wifi_get_event_mask (C++ 函数), 670
esp_wifi_get_mac (C++ 函数), 664
esp_wifi_get_max_tx_power (C++ 函数), 669
esp_wifi_get_mode (C++ 函数), 655
esp_wifi_get_promiscuous (C++ 函数), 665
esp_wifi_get_promiscuous_ctrl_filter (C++ 函数), 666
esp_wifi_get_promiscuous_filter (C++ 函数), 665
esp_wifi_get_protocol (C++ 函数), 660
esp_wifi_get_ps (C++ 函数), 659
esp_wifi_init (C++ 函数), 654
ESP_WIFI_MAX_CONN_NUM (C 宏), 690
esp_wifi_restore (C++ 函数), 656
esp_wifi_scan_get_ap_num (C++ 函数), 658
esp_wifi_scan_get_ap_records (C++ 函数), 658
esp_wifi_scan_start (C++ 函数), 657
esp_wifi_scan_stop (C++ 函数), 658
esp_wifi_set_ant (C++ 函数), 673
esp_wifi_set_ant_gpio (C++ 函数), 672
esp_wifi_set_bandwidth (C++ 函数), 660
esp_wifi_set_channel (C++ 函数), 661
esp_wifi_set_config (C++ 函数), 666
esp_wifi_set_country (C++ 函数), 662
esp_wifi_set_csi (C++ 函数), 672
esp_wifi_set_csi_config (C++ 函数), 671
esp_wifi_set_csi_rx_cb (C++ 函数), 671
esp_wifi_set_default_wifi_ap_handlers (C++ 函数), 797
esp_wifi_set_default_wifi_sta_handlers (C++ 函数), 797
esp_wifi_set_event_mask (C++ 函数), 670
esp_wifi_set_mac (C++ 函数), 663
esp_wifi_set_max_tx_power (C++ 函数), 669
esp_wifi_set_mode (C++ 函数), 654
esp_wifi_set_promiscuous (C++ 函数), 664
esp_wifi_set_promiscuous_ctrl_filter (C++ 函数), 665
esp_wifi_set_promiscuous_filter (C++ 函数), 665
esp_wifi_set_promiscuous_rx_cb (C++ 函数), 664
esp_wifi_set_protocol (C++ 函数), 660
esp_wifi_set_ps (C++ 函数), 659
esp_wifi_set_storage (C++ 函数), 668
esp_wifi_set_vendor_ie (C++ 函数), 668
esp_wifi_set_vendor_ie_cb (C++ 函数), 669
esp_wifi_sta_get_ap_info (C++ 函数), 659

esp_wifi_start (C++ 函数), 655
esp_wifi_stop (C++ 函数), 655
essl_clear_intr (C++ 函数), 1259
essl_get_intr (C++ 函数), 1259
essl_get_intr_ena (C++ 函数), 1260
essl_get_packet (C++ 函数), 1257
essl_get_rx_data_size (C++ 函数), 1256
essl_get_tx_buffer_num (C++ 函数), 1256
essl_handle_t (C++ 类型), 1260
essl_init (C++ 函数), 1255
essl_read_reg (C++ 函数), 1258
essl_reset_cnt (C++ 函数), 1256
essl_sdio_config_t (C++ 类), 1261
essl_sdio_config_t::card (C++ 成员), 1261
essl_sdio_config_t::recv_buffer_size (C++ 成员), 1261
essl_sdio_deinit_dev (C++ 函数), 1261
essl_sdio_init_dev (C++ 函数), 1261
essl_send_packet (C++ 函数), 1256
essl_send_slave_intr (C++ 函数), 1260
essl_set_intr_ena (C++ 函数), 1259
essl_wait_for_ready (C++ 函数), 1255
essl_wait_int (C++ 函数), 1258
essl_write_reg (C++ 函数), 1257
eStandardSleep (C++ 枚举子), 1505
eSuspended (C++ 枚举子), 1504
eTaskGetState (C++ 函数), 1481
eTaskState (C++ 类型), 1504
ETH_CMD_G_MAC_ADDR (C++ 枚举子), 765
ETH_CMD_G_PHY_ADDR (C++ 枚举子), 766
ETH_CMD_G_SPEED (C++ 枚举子), 766
ETH_CMD_S_MAC_ADDR (C++ 枚举子), 766
ETH_CMD_S_PHY_ADDR (C++ 枚举子), 766
ETH_CMD_S_PROMISCUOUS (C++ 枚举子), 766
ETH_CRC_LEN (C 宏), 765
ETH_DEFAULT_CONFIG (C 宏), 762
ETH_DUPLEX_FULL (C++ 枚举子), 766
ETH_DUPLEX_HALF (C++ 枚举子), 766
eth_duplex_t (C++ 类型), 766
eth_event_t (C++ 类型), 766
ETH_HEADER_LEN (C 宏), 764
ETH_JUMBO_FRAME_PAYLOAD_LEN (C 宏), 765
ETH_LINK_DOWN (C++ 枚举子), 766
eth_link_t (C++ 类型), 766
ETH_LINK_UP (C++ 枚举子), 766
eth_mac_config_t (C++ 类), 772
eth_mac_config_t::flags (C++ 成员), 773
eth_mac_config_t::rx_task_prio (C++ 成员), 772
eth_mac_config_t::rx_task_stack_size (C++ 成员), 772
eth_mac_config_t::smi_mdc_gpio_num (C++ 成员), 773
eth_mac_config_t::smi_mdio_gpio_num (C++ 成员), 773
eth_mac_config_t::sw_reset_timeout_ms (C++ 成员), 772
ETH_MAC_DEFAULT_CONFIG (C 宏), 773
ETH_MAC_FLAG_WORK_WITH_CACHE_DISABLE (C 宏), 773
ETH_MAX_PACKET_SIZE (C 宏), 765
ETH_MAX_PAYLOAD_LEN (C 宏), 764
ETH_MIN_PACKET_SIZE (C 宏), 765
ETH_MIN_PAYLOAD_LEN (C 宏), 764
eth_phy_config_t (C++ 类), 777
eth_phy_config_t::autonego_timeout_ms (C++ 成员), 777
eth_phy_config_t::phy_addr (C++ 成员), 777
eth_phy_config_t::reset_gpio_num (C++ 成员), 778
eth_phy_config_t::reset_timeout_ms (C++ 成员), 777
ETH_PHY_DEFAULT_CONFIG (C 宏), 778
ETH_SPEED_100M (C++ 枚举子), 766
ETH_SPEED_10M (C++ 枚举子), 766
eth_speed_t (C++ 类型), 766
ETH_STATE_DEINIT (C++ 枚举子), 765
ETH_STATE_DUPLEX (C++ 枚举子), 765
ETH_STATE_LINK (C++ 枚举子), 765
ETH_STATE_LLINIT (C++ 枚举子), 765
ETH_STATE_SPEED (C++ 枚举子), 765
ETH_VLAN_TAG_LEN (C 宏), 765
ETHERNET_EVENT_CONNECTED (C++ 枚举子), 767
ETHERNET_EVENT_DISCONNECTED (C++ 枚举子), 767

ETHERNET_EVENT_START (C++ 枚举子), 767
ETHERNET_EVENT_STOP (C++ 枚举子), 767
ETS_INTERNAL_INTR_SOURCE_OFF (C 宏), 1670
ETS_INTERNAL_PROFILING_INTR_SOURCE (C 宏), 1670
ETS_INTERNAL_SW0_INTR_SOURCE (C 宏), 1670
ETS_INTERNAL_SW1_INTR_SOURCE (C 宏), 1670
ETS_INTERNAL_TIMER0_INTR_SOURCE (C 宏), 1669
ETS_INTERNAL_TIMER1_INTR_SOURCE (C 宏), 1670
ETS_INTERNAL_TIMER2_INTR_SOURCE (C 宏), 1670
EventBits_t (C++ 类型), 1591
EventGroupHandle_t (C++ 类型), 1590

F

FAST_PROV_ACT_ENTER (C++ 枚举子), 434
FAST_PROV_ACT_EXIT (C++ 枚举子), 434
FAST_PROV_ACT_MAX (C++ 枚举子), 434
FAST_PROV_ACT_NONE (C++ 枚举子), 434
FAST_PROV_ACT_SUSPEND (C++ 枚举子), 434
ff_diskio_impl_t (C++ 类), 1395
ff_diskio_impl_t::init (C++ 成员), 1395
ff_diskio_impl_t::ioctl (C++ 成员), 1395
ff_diskio_impl_t::read (C++ 成员), 1395
ff_diskio_impl_t::status (C++ 成员), 1395
ff_diskio_impl_t::write (C++ 成员), 1395
ff_diskio_register (C++ 函数), 1395
ff_diskio_register_raw_partition (C++ 函数), 1396
ff_diskio_register_sdmmc (C++ 函数), 1396
ff_diskio_register_wl_partition (C++ 函数), 1396

G

gpio_config (C++ 函数), 852
gpio_config_t (C++ 类), 860
gpio_config_t::intr_type (C++ 成员), 860
gpio_config_t::mode (C++ 成员), 860
gpio_config_t::pin_bit_mask (C++ 成员), 860
gpio_config_t::pull_down_en (C++ 成员), 860
gpio_config_t::pull_up_en (C++ 成员), 860
gpio_deep_sleep_hold_dis (C++ 函数), 859
gpio_deep_sleep_hold_en (C++ 函数), 859
GPIO_DRIVE_CAP_0 (C++ 枚举子), 865
GPIO_DRIVE_CAP_1 (C++ 枚举子), 865
GPIO_DRIVE_CAP_2 (C++ 枚举子), 865
GPIO_DRIVE_CAP_3 (C++ 枚举子), 865
GPIO_DRIVE_CAP_DEFAULT (C++ 枚举子), 865
GPIO_DRIVE_CAP_MAX (C++ 枚举子), 865
gpio_drive_cap_t (C++ 类型), 865
GPIO_FLOATING (C++ 枚举子), 864
gpio_get_drive_capability (C++ 函数), 858
gpio_get_level (C++ 函数), 853
gpio_hold_dis (C++ 函数), 859
gpio_hold_en (C++ 函数), 858
gpio_install_isr_service (C++ 函数), 856
gpio_int_type_t (C++ 类型), 863
GPIO_INTR_ANYEDGE (C++ 枚举子), 863
gpio_intr_disable (C++ 函数), 853
GPIO_INTR_DISABLE (C++ 枚举子), 863
gpio_intr_enable (C++ 函数), 852
GPIO_INTR_HIGH_LEVEL (C++ 枚举子), 863
GPIO_INTR_LOW_LEVEL (C++ 枚举子), 863
GPIO_INTR_MAX (C++ 枚举子), 863
GPIO_INTR_NEGEDGE (C++ 枚举子), 863
GPIO_INTR_POSEDGE (C++ 枚举子), 863
gpio_iomux_in (C++ 函数), 859
gpio_iomux_out (C++ 函数), 860
gpio_isr_handle_t (C++ 类型), 860
gpio_isr_handler_add (C++ 函数), 857
gpio_isr_handler_remove (C++ 函数), 857
gpio_isr_register (C++ 函数), 855
gpio_isr_t (C++ 类型), 861
GPIO_MODE_DISABLE (C++ 枚举子), 864
GPIO_MODE_INPUT (C++ 枚举子), 864
GPIO_MODE_INPUT_OUTPUT (C++ 枚举子), 864
GPIO_MODE_INPUT_OUTPUT_OD (C++ 枚举子), 864
GPIO_MODE_OUTPUT (C++ 枚举子), 864
GPIO_MODE_OUTPUT_OD (C++ 枚举子), 864
gpio_mode_t (C++ 类型), 863
GPIO_NUM_0 (C++ 枚举子), 861
GPIO_NUM_1 (C++ 枚举子), 861
GPIO_NUM_10 (C++ 枚举子), 861
GPIO_NUM_11 (C++ 枚举子), 862
GPIO_NUM_12 (C++ 枚举子), 862

GPIO_NUM_13 (C++ 枚举子), 862
GPIO_NUM_14 (C++ 枚举子), 862
GPIO_NUM_15 (C++ 枚举子), 862
GPIO_NUM_16 (C++ 枚举子), 862
GPIO_NUM_17 (C++ 枚举子), 862
GPIO_NUM_18 (C++ 枚举子), 862
GPIO_NUM_19 (C++ 枚举子), 862
GPIO_NUM_2 (C++ 枚举子), 861
GPIO_NUM_20 (C++ 枚举子), 862
GPIO_NUM_21 (C++ 枚举子), 862
GPIO_NUM_26 (C++ 枚举子), 862
GPIO_NUM_27 (C++ 枚举子), 862
GPIO_NUM_28 (C++ 枚举子), 862
GPIO_NUM_29 (C++ 枚举子), 862
GPIO_NUM_3 (C++ 枚举子), 861
GPIO_NUM_30 (C++ 枚举子), 862
GPIO_NUM_31 (C++ 枚举子), 862
GPIO_NUM_32 (C++ 枚举子), 863
GPIO_NUM_33 (C++ 枚举子), 863
GPIO_NUM_34 (C++ 枚举子), 863
GPIO_NUM_35 (C++ 枚举子), 863
GPIO_NUM_36 (C++ 枚举子), 863
GPIO_NUM_37 (C++ 枚举子), 863
GPIO_NUM_38 (C++ 枚举子), 863
GPIO_NUM_39 (C++ 枚举子), 863
GPIO_NUM_4 (C++ 枚举子), 861
GPIO_NUM_5 (C++ 枚举子), 861
GPIO_NUM_6 (C++ 枚举子), 861
GPIO_NUM_7 (C++ 枚举子), 861
GPIO_NUM_8 (C++ 枚举子), 861
GPIO_NUM_9 (C++ 枚举子), 861
GPIO_NUM_MAX (C++ 枚举子), 863
GPIO_NUM_NC (C++ 枚举子), 861
gpio_num_t (C++ 类型), 861
GPIO_PORT_0 (C++ 枚举子), 861
GPIO_PORT_MAX (C++ 枚举子), 861
gpio_port_t (C++ 类型), 861
gpio_pull_mode_t (C++ 类型), 864
gpio_pulldown_dis (C++ 函数), 856
GPIO_PULLDOWN_DISABLE (C++ 枚举子), 864
gpio_pulldown_en (C++ 函数), 856
GPIO_PULLDOWN_ENABLE (C++ 枚举子), 864
GPIO_PULLDOWN_ONLY (C++ 枚举子), 864
gpio_pulldown_t (C++ 类型), 864
gpio_pullup_dis (C++ 函数), 856
GPIO_PULLUP_DISABLE (C++ 枚举子), 864
gpio_pullup_en (C++ 函数), 855
GPIO_PULLUP_ENABLE (C++ 枚举子), 864
GPIO_PULLUP_ONLY (C++ 枚举子), 864
GPIO_PULLUP_PULLDOWN (C++ 枚举子), 864
gpio_pullup_t (C++ 类型), 864
gpio_reset_pin (C++ 函数), 852
gpio_set_direction (C++ 函数), 854
gpio_set_drive_capability (C++ 函数), 858
gpio_set_intr_type (C++ 函数), 852
gpio_set_level (C++ 函数), 853
gpio_set_pull_mode (C++ 函数), 854
gpio_uninstall_isr_service (C++ 函数), 857
gpio_wakeup_disable (C++ 函数), 855
gpio_wakeup_enable (C++ 函数), 854

H

hall_sensor_read (C++ 函数), 814
heap_caps_add_region (C++ 函数), 1627
heap_caps_add_region_with_caps (C++ 函数), 1628
heap_caps_aligned_alloc (C++ 函数), 1620
heap_caps_aligned_calloc (C++ 函数), 1620
heap_caps_aligned_free (C++ 函数), 1620
heap_caps_calloc (C++ 函数), 1621
heap_caps_calloc_prefer (C++ 函数), 1624
heap_caps_check_integrity (C++ 函数), 1623
heap_caps_check_integrity_addr (C++ 函数), 1623
heap_caps_check_integrity_all (C++ 函数), 1622
heap_caps_dump (C++ 函数), 1624
heap_caps_dump_all (C++ 函数), 1625
heap_caps_enable_nonos_stack_heaps (C++ 函数), 1627
heap_caps_free (C++ 函数), 1619
heap_caps_get_allocated_size (C++ 函数), 1625
heap_caps_get_free_size (C++ 函数), 1621
heap_caps_get_info (C++ 函数), 1622

heap_caps_get_largest_free_block (C++ 函数), 1622
heap_caps_get_minimum_free_size (C++ 函数), 1621
heap_caps_get_total_size (C++ 函数), 1621
heap_caps_init (C++ 函数), 1627
heap_caps_malloc (C++ 函数), 1619
heap_caps_malloc_extmem_enable (C++ 函数), 1623
heap_caps_malloc_prefer (C++ 函数), 1624
heap_caps_print_heap_info (C++ 函数), 1622
heap_caps_realloc (C++ 函数), 1619
heap_caps_realloc_prefer (C++ 函数), 1624
HEAP_TRACE_ALL (C++ 枚举子), 1648
heap_trace_dump (C++ 函数), 1648
heap_trace_get (C++ 函数), 1647
heap_trace_get_count (C++ 函数), 1647
heap_trace_init_standalone (C++ 函数), 1645
heap_trace_init_tohost (C++ 函数), 1646
HEAP_TRACE_LEAKS (C++ 枚举子), 1648
heap_trace_mode_t (C++ 类型), 1648
heap_trace_record_t (C++ 类), 1648
heap_trace_record_t::address (C++ 成员), 1648
heap_trace_record_t::alloced_by (C++ 成员), 1648
heap_trace_record_t::ccount (C++ 成员), 1648
heap_trace_record_t::freed_by (C++ 成员), 1648
heap_trace_record_t::size (C++ 成员), 1648
heap_trace_resume (C++ 函数), 1647
heap_trace_start (C++ 函数), 1646
heap_trace_stop (C++ 函数), 1646
HTTP_AUTH_TYPE_BASIC (C++ 枚举子), 1169
HTTP_AUTH_TYPE_DIGEST (C++ 枚举子), 1169
HTTP_AUTH_TYPE_NONE (C++ 枚举子), 1169
HTTP_EVENT_DISCONNECTED (C++ 枚举子), 1168
HTTP_EVENT_ERROR (C++ 枚举子), 1167
http_event_handle_cb (C++ 类型), 1167
HTTP_EVENT_HEADER_SENT (C++ 枚举子), 1167
HTTP_EVENT_HEADERS_SENT (C++ 枚举子), 1167
HTTP_EVENT_ON_CONNECTED (C++ 枚举子), 1167
HTTP_EVENT_ON_DATA (C++ 枚举子), 1167
HTTP_EVENT_ON_FINISH (C++ 枚举子), 1168
HTTP_EVENT_ON_HEADER (C++ 枚举子), 1167
HTTP_METHOD_DELETE (C++ 枚举子), 1168
HTTP_METHOD_GET (C++ 枚举子), 1168
HTTP_METHOD_HEAD (C++ 枚举子), 1168
HTTP_METHOD_MAX (C++ 枚举子), 1169
HTTP_METHOD_NOTIFY (C++ 枚举子), 1168
HTTP_METHOD_OPTIONS (C++ 枚举子), 1169
HTTP_METHOD_PATCH (C++ 枚举子), 1168
HTTP_METHOD_POST (C++ 枚举子), 1168
HTTP_METHOD_PUT (C++ 枚举子), 1168
HTTP_METHOD_SUBSCRIBE (C++ 枚举子), 1168
HTTP_METHOD_UNSUBSCRIBE (C++ 枚举子), 1168
HTTP_TRANSPORT_OVER_SSL (C++ 枚举子), 1168
HTTP_TRANSPORT_OVER_TCP (C++ 枚举子), 1168
HTTP_TRANSPORT_UNKNOWN (C++ 枚举子), 1168
HTTPD_200 (C 宏), 1204
HTTPD_204 (C 宏), 1204
HTTPD_207 (C 宏), 1204
HTTPD_400 (C 宏), 1204
HTTPD_400_BAD_REQUEST (C++ 枚举子), 1209
HTTPD_404 (C 宏), 1204
HTTPD_404_NOT_FOUND (C++ 枚举子), 1209
HTTPD_405_METHOD_NOT_ALLOWED (C++ 枚举子), 1209
HTTPD_408 (C 宏), 1204
HTTPD_408_REQ_TIMEOUT (C++ 枚举子), 1209
HTTPD_411_LENGTH_REQUIRED (C++ 枚举子), 1209
HTTPD_414_URI_TOO_LONG (C++ 枚举子), 1209
HTTPD_431_REQ_HDR_FIELDS_TOO_LARGE (C++ 枚举子), 1209
HTTPD_500 (C 宏), 1204
HTTPD_500_INTERNAL_SERVER_ERROR (C++ 枚举子), 1209
HTTPD_501_METHOD_NOT_IMPLEMENTED (C++ 枚举子), 1209
HTTPD_505_VERSION_NOT_SUPPORTED (C++ 枚举子), 1209
httpd_close_func_t (C++ 类型), 1208
httpd_config (C++ 类), 1200
httpd_config::backlog_conn (C++ 成员), 1201
httpd_config::close_fn (C++ 成员), 1201

httpd_config::core_id (C++ 成员), 1200
httpd_config::ctrl_port (C++ 成员), 1200
httpd_config::global_transport_ctx (C++ 成员), 1201
httpd_config::global_transport_ctx_free_fn (C++ 成员), 1201
httpd_config::global_user_ctx (C++ 成员), 1201
httpd_config::global_user_ctx_free_fn (C++ 成员), 1201
httpd_config::lru_purge_enable (C++ 成员), 1201
httpd_config::max_open_sockets (C++ 成员), 1200
httpd_config::max_resp_headers (C++ 成员), 1201
httpd_config::max_uri_handlers (C++ 成员), 1200
httpd_config::open_fn (C++ 成员), 1201
httpd_config::recv_wait_timeout (C++ 成员), 1201
httpd_config::send_wait_timeout (C++ 成员), 1201
httpd_config::server_port (C++ 成员), 1200
httpd_config::stack_size (C++ 成员), 1200
httpd_config::task_priority (C++ 成员), 1200
httpd_config::uri_match_fn (C++ 成员), 1202
httpd_config_t (C++ 类型), 1208
HTTPD_DEFAULT_CONFIG (C 宏), 1204
HTTPD_ERR_CODE_MAX (C++ 枚举子), 1209
httpd_err_code_t (C++ 类型), 1209
httpd_err_handler_func_t (C++ 类型), 1207
httpd_free_ctx_fn_t (C++ 类型), 1207
httpd_get_global_transport_ctx (C++ 函数), 1199
httpd_get_global_user_ctx (C++ 函数), 1199
httpd_handle_t (C++ 类型), 1207
HTTPD_MAX_REQ_HDR_LEN (C 宏), 1203
HTTPD_MAX_URI_LEN (C 宏), 1203
httpd_method_t (C++ 类型), 1207
httpd_open_func_t (C++ 类型), 1207
httpd_pending_func_t (C++ 类型), 1206
httpd_query_key_value (C++ 函数), 1187
httpd_queue_work (C++ 函数), 1197
httpd_recv_func_t (C++ 类型), 1206
httpd_register_err_handler (C++ 函数), 1195
httpd_register_uri_handler (C++ 函数), 1180
httpd_req (C++ 类), 1202
httpd_req::aux (C++ 成员), 1202
httpd_req::content_len (C++ 成员), 1202
httpd_req::free_ctx (C++ 成员), 1203
httpd_req::handle (C++ 成员), 1202
httpd_req::ignore_sess_ctx_changes (C++ 成员), 1203
httpd_req::method (C++ 成员), 1202
httpd_req::sess_ctx (C++ 成员), 1202
httpd_req::uri (C++ 成员), 1202
httpd_req::user_ctx (C++ 成员), 1202
httpd_req_get_hdr_value_len (C++ 函数), 1184
httpd_req_get_hdr_value_str (C++ 函数), 1185
httpd_req_get_url_query_len (C++ 函数), 1186
httpd_req_get_url_query_str (C++ 函数), 1186
httpd_req_recv (C++ 函数), 1184
httpd_req_t (C++ 类型), 1205
httpd_req_to_sockfd (C++ 函数), 1183
httpd_resp_send (C++ 函数), 1188
httpd_resp_send_404 (C++ 函数), 1193
httpd_resp_send_408 (C++ 函数), 1193
httpd_resp_send_500 (C++ 函数), 1194
httpd_resp_send_chunk (C++ 函数), 1189
httpd_resp_send_err (C++ 函数), 1192
httpd_resp_sendstr (C++ 函数), 1189
httpd_resp_sendstr_chunk (C++ 函数), 1190
httpd_resp_set_hdr (C++ 函数), 1191
httpd_resp_set_status (C++ 函数), 1190
httpd_resp_set_type (C++ 函数), 1191
HTTPD_RESP_USE_STRLEN (C 宏), 1205
httpd_send (C++ 函数), 1194
httpd_send_func_t (C++ 类型), 1205
httpd_sess_get_ctx (C++ 函数), 1197
httpd_sess_get_transport_ctx (C++ 函数), 1198
httpd_sess_set_ctx (C++ 函数), 1198
httpd_sess_set_pending_override (C++ 函数), 1183

httpd_sess_set_recv_override (C++ 函数), 1182
httpd_sess_set_send_override (C++ 函数), 1182
httpd_sess_set_transport_ctx (C++ 函数), 1198
httpd_sess_trigger_close (C++ 函数), 1199
httpd_sess_update_lru_counter (C++ 函数), 1199
HTTPD_SOCK_ERR_FAIL (C 宏), 1204
HTTPD_SOCK_ERR_INVALID (C 宏), 1204
HTTPD_SOCK_ERR_TIMEOUT (C 宏), 1204
httpd_ssl_config (C++ 类), 1211
httpd_ssl_config::cacert_len (C++ 成员), 1211
httpd_ssl_config::cacert_pem (C++ 成员), 1211
httpd_ssl_config::httpd (C++ 成员), 1211
httpd_ssl_config::port_insecure (C++ 成员), 1211
httpd_ssl_config::port_secure (C++ 成员), 1211
httpd_ssl_config::prvtkey_len (C++ 成员), 1211
httpd_ssl_config::prvtkey_pem (C++ 成员), 1211
httpd_ssl_config::transport_mode (C++ 成员), 1211
HTTPD_SSL_CONFIG_DEFAULT (C 宏), 1212
httpd_ssl_config_t (C++ 类型), 1212
httpd_ssl_start (C++ 函数), 1210
httpd_ssl_stop (C++ 函数), 1211
HTTPD_SSL_TRANSPORT_INSECURE (C++ 枚举子), 1212
httpd_ssl_transport_mode_t (C++ 类型), 1212
HTTPD_SSL_TRANSPORT_SECURE (C++ 枚举子), 1212
httpd_start (C++ 函数), 1195
httpd_stop (C++ 函数), 1196
HTTPD_TYPE_JSON (C 宏), 1204
HTTPD_TYPE_OCTET (C 宏), 1204
HTTPD_TYPE_TEXT (C 宏), 1204
httpd_unregister_uri (C++ 函数), 1182
httpd_unregister_uri_handler (C++ 函数), 1181
httpd_uri (C++ 类), 1203
httpd_uri::handler (C++ 成员), 1203
httpd_uri::method (C++ 成员), 1203
httpd_uri::uri (C++ 成员), 1203
httpd_uri::user_ctx (C++ 成员), 1203
httpd_uri_match_func_t (C++ 类型), 1208
httpd_uri_match_wildcard (C++ 函数), 1187
httpd_uri_t (C++ 类型), 1205
httpd_work_fn_t (C++ 类型), 1208
HttpStatus_Code (C++ 类型), 1169
HttpStatus_Found (C++ 枚举子), 1169
HttpStatus_MovedPermanently (C++ 枚举子), 1169
HttpStatus_TemporaryRedirect (C++ 枚举子), 1169
HttpStatus_Unauthorized (C++ 枚举子), 1169

I

i2c_ack_type_t (C++ 类型), 892
I2C_ADDR_BIT_10 (C++ 枚举子), 892
I2C_ADDR_BIT_7 (C++ 枚举子), 892
I2C_ADDR_BIT_MAX (C++ 枚举子), 892
i2c_addr_mode_t (C++ 类型), 892
I2C_APB_CLK_FREQ (C 宏), 889
I2C_CMD_END (C++ 枚举子), 891
i2c_cmd_handle_t (C++ 类型), 890
i2c_cmd_link_create (C++ 函数), 881
i2c_cmd_link_delete (C++ 函数), 881
I2C_CMD_READ (C++ 枚举子), 891
I2C_CMD_RESTART (C++ 枚举子), 891
I2C_CMD_STOP (C++ 枚举子), 891
I2C_CMD_WRITE (C++ 枚举子), 891
i2c_config_t (C++ 类), 890
i2c_config_t::addr_10bit_en (C++ 成员), 890
i2c_config_t::clk_speed (C++ 成员), 890
i2c_config_t::mode (C++ 成员), 890
i2c_config_t::scl_io_num (C++ 成员), 890
i2c_config_t::scl_pullup_en (C++ 成员), 890
i2c_config_t::sda_io_num (C++ 成员), 890
i2c_config_t::sda_pullup_en (C++ 成员), 890
i2c_config_t::slave_addr (C++ 成员), 890
I2C_DATA_MODE_LSB_FIRST (C++ 枚举子), 892
I2C_DATA_MODE_MAX (C++ 枚举子), 892
I2C_DATA_MODE_MSB_FIRST (C++ 枚举子), 892
i2c_driver_delete (C++ 函数), 879
i2c_driver_install (C++ 函数), 878

i2c_filter_disable (C++ 函数), 886
i2c_filter_enable (C++ 函数), 886
i2c_get_data_mode (C++ 函数), 889
i2c_get_data_timing (C++ 函数), 888
i2c_get_period (C++ 函数), 885
i2c_get_start_timing (C++ 函数), 887
i2c_get_stop_timing (C++ 函数), 887
i2c_get_timeout (C++ 函数), 888
i2c_isr_free (C++ 函数), 880
i2c_isr_register (C++ 函数), 880
I2C_MASTER_ACK (C++ 枚举子), 892
I2C_MASTER_ACK_MAX (C++ 枚举子), 892
i2c_master_cmd_begin (C++ 函数), 884
I2C_MASTER_LAST_NACK (C++ 枚举子), 892
I2C_MASTER_NACK (C++ 枚举子), 892
i2c_master_read (C++ 函数), 883
I2C_MASTER_READ (C++ 枚举子), 891
i2c_master_read_byte (C++ 函数), 882
i2c_master_start (C++ 函数), 881
i2c_master_stop (C++ 函数), 883
i2c_master_write (C++ 函数), 882
I2C_MASTER_WRITE (C++ 枚举子), 891
i2c_master_write_byte (C++ 函数), 881
I2C_MODE_MASTER (C++ 枚举子), 891
I2C_MODE_MAX (C++ 枚举子), 891
I2C_MODE_SLAVE (C++ 枚举子), 891
i2c_mode_t (C++ 类型), 891
I2C_NUM_0 (C 宏), 889
I2C_NUM_1 (C 宏), 890
I2C_NUM_MAX (C 宏), 890
i2c_opmode_t (C++ 类型), 891
i2c_param_config (C++ 函数), 879
i2c_port_t (C++ 类型), 891
i2c_reset_rx_fifo (C++ 函数), 879
i2c_reset_tx_fifo (C++ 函数), 879
i2c_rw_t (C++ 类型), 891
I2C_SCLK_APB (C++ 枚举子), 892
I2C_SCLK_REF_TICK (C++ 枚举子), 892
i2c_sclk_t (C++ 类型), 892
i2c_set_data_mode (C++ 函数), 889
i2c_set_data_timing (C++ 函数), 888
i2c_set_period (C++ 函数), 885
i2c_set_pin (C++ 函数), 880
i2c_set_start_timing (C++ 函数), 886
i2c_set_stop_timing (C++ 函数), 887
i2c_set_timeout (C++ 函数), 888
i2c_slave_read_buffer (C++ 函数), 884
i2c_slave_write_buffer (C++ 函数), 884
i2c_trans_mode_t (C++ 类型), 891
i2s_adc_disable (C++ 函数), 903
i2s_adc_enable (C++ 函数), 903
I2S_BITS_PER_SAMPLE_16BIT (C++ 枚举子), 906
I2S_BITS_PER_SAMPLE_24BIT (C++ 枚举子), 906
I2S_BITS_PER_SAMPLE_32BIT (C++ 枚举子), 906
I2S_BITS_PER_SAMPLE_8BIT (C++ 枚举子), 906
i2s_bits_per_sample_t (C++ 类型), 906
I2S_CHANNEL_FMT_ALL_LEFT (C++ 枚举子), 907
I2S_CHANNEL_FMT_ALL_RIGHT (C++ 枚举子), 907
I2S_CHANNEL_FMT_ONLY_LEFT (C++ 枚举子), 907
I2S_CHANNEL_FMT_ONLY_RIGHT (C++ 枚举子), 907
I2S_CHANNEL_FMT_RIGHT_LEFT (C++ 枚举子), 907
i2s_channel_fmt_t (C++ 类型), 906
I2S_CHANNEL_MONO (C++ 枚举子), 906
I2S_CHANNEL_STEREO (C++ 枚举子), 906
i2s_channel_t (C++ 类型), 906
I2S_CLK_APLL (C++ 枚举子), 908
I2S_CLK_D2CLK (C++ 枚举子), 908
i2s_clock_src_t (C++ 类型), 908
I2S_COMM_FORMAT_I2S (C++ 枚举子), 906
I2S_COMM_FORMAT_I2S_LSB (C++ 枚举子), 906
I2S_COMM_FORMAT_I2S_MSB (C++ 枚举子), 906
I2S_COMM_FORMAT_PCM (C++ 枚举子), 906
I2S_COMM_FORMAT_PCM_LONG (C++ 枚举子), 906
I2S_COMM_FORMAT_PCM_SHORT (C++ 枚举子), 906
i2s_comm_format_t (C++ 类型), 906
i2s_config_t (C++ 类), 904
i2s_config_t::bits_per_sample (C++ 成员), 904
i2s_config_t::channel_format (C++ 成员), 904
i2s_config_t::communication_format (C++ 成员), 904
i2s_config_t::dma_buf_count (C++ 成员), 904
i2s_config_t::dma_buf_len (C++ 成员), 904
i2s_config_t::fixed_mclk (C++ 成员), 904

i2s_config_t::intr_alloc_flags (C++ 成员), 904
i2s_config_t::mode (C++ 成员), 904
i2s_config_t::sample_rate (C++ 成员), 904
i2s_config_t::tx_desc_auto_clear (C++ 成员), 904
i2s_config_t::use_apll (C++ 成员), 904
I2S_DAC_CHANNEL_BOTH_EN (C++ 枚举子), 908
I2S_DAC_CHANNEL_DISABLE (C++ 枚举子), 908
I2S_DAC_CHANNEL_LEFT_EN (C++ 枚举子), 908
I2S_DAC_CHANNEL_MAX (C++ 枚举子), 908
I2S_DAC_CHANNEL_RIGHT_EN (C++ 枚举子), 908
i2s_dac_mode_t (C++ 类型), 908
i2s_driver_install (C++ 函数), 898
i2s_driver_uninstall (C++ 函数), 899
I2S_EVENT_DMA_ERROR (C++ 枚举子), 908
I2S_EVENT_MAX (C++ 枚举子), 908
I2S_EVENT_RX_DONE (C++ 枚举子), 908
i2s_event_t (C++ 类), 905
i2s_event_t::size (C++ 成员), 905
i2s_event_t::type (C++ 成员), 905
I2S_EVENT_TX_DONE (C++ 枚举子), 908
i2s_event_type_t (C++ 类型), 907
i2s_get_clk (C++ 函数), 902
i2s_isr_handle_t (C++ 类型), 904
I2S_MODE_ADC_BUILT_IN (C++ 枚举子), 907
I2S_MODE_DAC_BUILT_IN (C++ 枚举子), 907
I2S_MODE_MASTER (C++ 枚举子), 907
I2S_MODE_RX (C++ 枚举子), 907
I2S_MODE_SLAVE (C++ 枚举子), 907
i2s_mode_t (C++ 类型), 907
I2S_MODE_TX (C++ 枚举子), 907
i2s_pin_config_t (C++ 类), 905
i2s_pin_config_t::bck_io_num (C++ 成员), 905
i2s_pin_config_t::data_in_num (C++ 成员), 905
i2s_pin_config_t::data_out_num (C++ 成员), 905
i2s_pin_config_t::ws_io_num (C++ 成员), 905
I2S_PIN_NO_CHANGE (C 宏), 905
i2s_port_t (C++ 类型), 905
i2s_read (C++ 函数), 900
i2s_set_adc_mode (C++ 函数), 903
i2s_set_clk (C++ 函数), 902
i2s_set_dac_mode (C++ 函数), 898
i2s_set_pin (C++ 函数), 897
i2s_set_sample_rates (C++ 函数), 901
i2s_start (C++ 函数), 901
i2s_stop (C++ 函数), 901
i2s_write (C++ 函数), 899
i2s_write_expand (C++ 函数), 899
i2s_zero_dma_buffer (C++ 函数), 902
I_ADDI (C 宏), 2234
I_ADDR (C 宏), 2233
I_ANDI (C 宏), 2234
I_ANDR (C 宏), 2233
I_BGE (C 宏), 2233
I_BL (C 宏), 2233
I_BXFI (C 宏), 2233
I_BXFR (C 宏), 2233
I_BXI (C 宏), 2233
I_BXR (C 宏), 2233
I_BXZI (C 宏), 2233
I_BXZR (C 宏), 2233
I_DELAY (C 宏), 2232
I_END (C 宏), 2232
I_HALT (C 宏), 2232
I_LD (C 宏), 2232
I_LSHI (C 宏), 2234
I_LSHR (C 宏), 2234
I_MOVI (C 宏), 2234
I_MOVR (C 宏), 2234
I_ORI (C 宏), 2234
I_ORR (C 宏), 2233
I_RD_REG (C 宏), 2232
I_RSHI (C 宏), 2234
I_RSHR (C 宏), 2234
I_ST (C 宏), 2232
I_SUBI (C 宏), 2234
I_SUBR (C 宏), 2233
I_WR_REG (C 宏), 2232
intr_handle_data_t (C++ 类型), 1670
intr_handle_t (C++ 类型), 1670
intr_handler_t (C++ 类型), 1670

L

LEDC_APB_CLK (*C++ 枚举子*), 926
LEDC_APB_CLK_HZ (*C 宏*), 923
LEDC_AUTO_CLK (*C++ 枚举子*), 925
ledc_bind_channel_timer (*C++ 函数*), 919
LEDC_CHANNEL_0 (*C++ 枚举子*), 926
LEDC_CHANNEL_1 (*C++ 枚举子*), 926
LEDC_CHANNEL_2 (*C++ 枚举子*), 926
LEDC_CHANNEL_3 (*C++ 枚举子*), 926
LEDC_CHANNEL_4 (*C++ 枚举子*), 926
LEDC_CHANNEL_5 (*C++ 枚举子*), 927
LEDC_CHANNEL_6 (*C++ 枚举子*), 927
LEDC_CHANNEL_7 (*C++ 枚举子*), 927
ledc_channel_config (*C++ 函数*), 913
ledc_channel_config_t (*C++ 类*), 923
ledc_channel_config_t::channel (*C++ 成员*), 924
ledc_channel_config_t::duty (*C++ 成员*), 924
ledc_channel_config_t::gpio_num (*C++ 成员*), 924
ledc_channel_config_t::hpoint (*C++ 成员*), 924
ledc_channel_config_t::intr_type (*C++ 成员*), 924
ledc_channel_config_t::speed_mode (*C++ 成员*), 924
ledc_channel_config_t::timer_sel (*C++ 成员*), 924
LEDC_CHANNEL_MAX (*C++ 枚举子*), 927
ledc_channel_t (*C++ 类型*), 926
ledc_clk_cfg_t (*C++ 类型*), 925
ledc_clk_src_t (*C++ 类型*), 926
LEDC_DUTY_DIR_DECREASE (*C++ 枚举子*), 925
LEDC_DUTY_DIR_INCREASE (*C++ 枚举子*), 925
LEDC_DUTY_DIR_MAX (*C++ 枚举子*), 925
ledc_duty_direction_t (*C++ 类型*), 925
LEDC_ERR_DUTY (*C 宏*), 923
LEDC_ERR_VAL (*C 宏*), 923
ledc_fade_func_install (*C++ 函数*), 921
ledc_fade_func_uninstall (*C++ 函数*), 921
LEDC_FADE_MAX (*C++ 枚举子*), 928
ledc_fade_mode_t (*C++ 类型*), 928
LEDC_FADE_NO_WAIT (*C++ 枚举子*), 928
ledc_fade_start (*C++ 函数*), 921
LEDC_FADE_WAIT_DONE (*C++ 枚举子*), 928
ledc_get_duty (*C++ 函数*), 916
ledc_get_freq (*C++ 函数*), 915
ledc_get_hpoint (*C++ 函数*), 916
LEDC_INTR_DISABLE (*C++ 枚举子*), 925
LEDC_INTR_FADE_END (*C++ 枚举子*), 925
LEDC_INTR_MAX (*C++ 枚举子*), 925
ledc_intr_type_t (*C++ 类型*), 925
ledc_isr_handle_t (*C++ 类型*), 923
ledc_isr_register (*C++ 函数*), 917
LEDC_LOW_SPEED_MODE (*C++ 枚举子*), 925
ledc_mode_t (*C++ 类型*), 925
LEDC_REF_CLK_HZ (*C 宏*), 923
LEDC_REF_TICK (*C++ 枚举子*), 926
ledc_set_duty (*C++ 函数*), 916
ledc_set_duty_and_update (*C++ 函数*), 921
ledc_set_duty_with_hpoint (*C++ 函数*), 915
ledc_set_fade (*C++ 函数*), 917
ledc_set_fade_step_and_start (*C++ 函数*), 922
ledc_set_fade_time_and_start (*C++ 函数*), 922
ledc_set_fade_with_step (*C++ 函数*), 919
ledc_set_fade_with_time (*C++ 函数*), 920
ledc_set_freq (*C++ 函数*), 915
ledc_set_pin (*C++ 函数*), 914
LEDC_SLOW_CLK_APB (*C++ 枚举子*), 925
LEDC_SLOW_CLK_RTC8M (*C++ 枚举子*), 925
ledc_slow_clk_sel_t (*C++ 类型*), 925
LEDC_SPEED_MODE_MAX (*C++ 枚举子*), 925
ledc_stop (*C++ 函数*), 914
LEDC_TIMER_0 (*C++ 枚举子*), 926
LEDC_TIMER_1 (*C++ 枚举子*), 926
LEDC_TIMER_10_BIT (*C++ 枚举子*), 927
LEDC_TIMER_11_BIT (*C++ 枚举子*), 927
LEDC_TIMER_12_BIT (*C++ 枚举子*), 927
LEDC_TIMER_13_BIT (*C++ 枚举子*), 927
LEDC_TIMER_14_BIT (*C++ 枚举子*), 928
LEDC_TIMER_15_BIT (*C++ 枚举子*), 928
LEDC_TIMER_16_BIT (*C++ 枚举子*), 928
LEDC_TIMER_17_BIT (*C++ 枚举子*), 928
LEDC_TIMER_18_BIT (*C++ 枚举子*), 928

LEDC_TIMER_19_BIT (C++ 枚举子), 928
LEDC_TIMER_1_BIT (C++ 枚举子), 927
LEDC_TIMER_2 (C++ 枚举子), 926
LEDC_TIMER_20_BIT (C++ 枚举子), 928
LEDC_TIMER_2_BIT (C++ 枚举子), 927
LEDC_TIMER_3 (C++ 枚举子), 926
LEDC_TIMER_3_BIT (C++ 枚举子), 927
LEDC_TIMER_4_BIT (C++ 枚举子), 927
LEDC_TIMER_5_BIT (C++ 枚举子), 927
LEDC_TIMER_6_BIT (C++ 枚举子), 927
LEDC_TIMER_7_BIT (C++ 枚举子), 927
LEDC_TIMER_8_BIT (C++ 枚举子), 927
LEDC_TIMER_9_BIT (C++ 枚举子), 927
LEDC_TIMER_BIT_MAX (C++ 枚举子), 928
ledc_timer_bit_t (C++ 类型), 927
ledc_timer_config (C++ 函数), 913
ledc_timer_config_t (C++ 类), 924
ledc_timer_config_t::bit_num (C++ 成员), 924
ledc_timer_config_t::clk_cfg (C++ 成员), 924
ledc_timer_config_t::duty_resolution (C++ 成员), 924
ledc_timer_config_t::freq_hz (C++ 成员), 924
ledc_timer_config_t::speed_mode (C++ 成员), 924
ledc_timer_config_t::timer_num (C++ 成员), 924
LEDC_TIMER_MAX (C++ 枚举子), 926
ledc_timer_pause (C++ 函数), 918
ledc_timer_resume (C++ 函数), 919
ledc_timer_rst (C++ 函数), 918
ledc_timer_set (C++ 函数), 918
ledc_timer_t (C++ 类型), 926
ledc_update_duty (C++ 函数), 914
LEDC_USE_APB_CLK (C++ 枚举子), 926
LEDC_USE_REF_TICK (C++ 枚举子), 926
LEDC_USE_RTC8M_CLK (C++ 枚举子), 926

M

M_BGE (C 宏), 2234
M_BL (C 宏), 2234
M_BX (C 宏), 2235
M_BXF (C 宏), 2235
M_BXZ (C 宏), 2235
M_LABEL (C 宏), 2234
MALLOC_CAP_32BIT (C 宏), 1625
MALLOC_CAP_8BIT (C 宏), 1625
MALLOC_CAP_DEFAULT (C 宏), 1626
MALLOC_CAP_DMA (C 宏), 1626
MALLOC_CAP_EXEC (C 宏), 1625
MALLOC_CAP_INTERNAL (C 宏), 1626
MALLOC_CAP_INVALID (C 宏), 1626
MALLOC_CAP_PID2 (C 宏), 1626
MALLOC_CAP_PID3 (C 宏), 1626
MALLOC_CAP_PID4 (C 宏), 1626
MALLOC_CAP_PID5 (C 宏), 1626
MALLOC_CAP_PID6 (C 宏), 1626
MALLOC_CAP_PID7 (C 宏), 1626
MALLOC_CAP_SPIRAM (C 宏), 1626
MAX_BLE_DEVNAME_LEN (C 宏), 1282
MAX_FDS (C 宏), 1388
mbc_master_destroy (C++ 函数), 1236
mbc_master_get_cid_info (C++ 函数), 1239
mbc_master_get_parameter (C++ 函数), 1240
mbc_master_init (C++ 函数), 1235
mbc_master_send_request (C++ 函数), 1238
mbc_master_set_descriptor (C++ 函数), 1238
mbc_master_set_parameter (C++ 函数), 1240
mbc_master_setup (C++ 函数), 1235
mbc_master_start (C++ 函数), 1236
mbc_slave_check_event (C++ 函数), 1237
mbc_slave_destroy (C++ 函数), 1236
mbc_slave_get_param_info (C++ 函数), 1237
mbc_slave_init (C++ 函数), 1234
mbc_slave_set_descriptor (C++ 函数), 1237
mbc_slave_setup (C++ 函数), 1235
mbc_slave_start (C++ 函数), 1236
MCPWM0A (C++ 枚举子), 952
MCPWM0B (C++ 枚举子), 952
MCPWM1A (C++ 枚举子), 952
MCPWM1B (C++ 枚举子), 952
MCPWM2A (C++ 枚举子), 952
MCPWM2B (C++ 枚举子), 952
MCPWM_ACTION_FORCE_HIGH (C++ 枚举子), 936
MCPWM_ACTION_FORCE_LOW (C++ 枚举子), 936

MCPWM_ACTION_NO_CHANGE (C++ 枚举子), 936
mcpwm_action_on_pwmxa_t (C++ 类型), 952
mcpwm_action_on_pwmxb_t (C++ 类型), 952
MCPWM_ACTION_TOGGLE (C++ 枚举子), 936
MCPWM_ACTIVE_HIGH_COMPLIMENT_MODE (C++ 枚举子), 937
MCPWM_ACTIVE_HIGH_MODE (C++ 枚举子), 937
MCPWM_ACTIVE_LOW_COMPLIMENT_MODE (C++ 枚举子), 937
MCPWM_ACTIVE_LOW_MODE (C++ 枚举子), 937
MCPWM_ACTIVE_RED_FED_FROM_PWMXA (C++ 枚举子), 937
MCPWM_ACTIVE_RED_FED_FROM_PWMXB (C++ 枚举子), 937
MCPWM_BYPASS_FED (C++ 枚举子), 937
MCPWM_BYPASS_RED (C++ 枚举子), 937
MCPWM_CAP_0 (C++ 枚举子), 953
MCPWM_CAP_1 (C++ 枚举子), 953
MCPWM_CAP_2 (C++ 枚举子), 953
mcpwm_capture_disable (C++ 函数), 947
mcpwm_capture_enable (C++ 函数), 947
mcpwm_capture_on_edge_t (C++ 类型), 937
mcpwm_capture_signal_get_edge (C++ 函数), 948
mcpwm_capture_signal_get_value (C++ 函数), 948
mcpwm_capture_signal_t (C++ 类型), 955
mcpwm_carrier_config_t (C++ 类), 951
mcpwm_carrier_config_t::carrier_duty (C++ 成员), 951
mcpwm_carrier_config_t::carrier_ivt_mode (C++ 成员), 951
mcpwm_carrier_config_t::carrier_os_mode (C++ 成员), 951
mcpwm_carrier_config_t::carrier_period (C++ 成员), 951
mcpwm_carrier_config_t::pulse_width_in_os (C++ 成员), 951
mcpwm_carrier_disable (C++ 函数), 943
mcpwm_carrier_enable (C++ 函数), 942
mcpwm_carrier_init (C++ 函数), 942
mcpwm_carrier_oneshot_mode_disable (C++ 函数), 944
mcpwm_carrier_oneshot_mode_enable (C++ 函数), 944
mcpwm_carrier_os_t (C++ 类型), 954
MCPWM_CARRIER_OUT_IVT_DIS (C++ 枚举子), 954
MCPWM_CARRIER_OUT_IVT_EN (C++ 枚举子), 954
mcpwm_carrier_out_ivt_t (C++ 类型), 954
mcpwm_carrier_output_invert (C++ 函数), 944
mcpwm_carrier_set_duty_cycle (C++ 函数), 943
mcpwm_carrier_set_period (C++ 函数), 943
mcpwm_config_t (C++ 类), 950
mcpwm_config_t::cmpr_a (C++ 成员), 950
mcpwm_config_t::cmpr_b (C++ 成员), 950
mcpwm_config_t::counter_mode (C++ 成员), 951
mcpwm_config_t::duty_mode (C++ 成员), 950
mcpwm_config_t::frequency (C++ 成员), 950
MCPWM_COUNTER_MAX (C++ 枚举子), 936
mcpwm_counter_type_t (C++ 类型), 935
MCPWM_DEADTIME_BYPASS (C++ 枚举子), 936
mcpwm_deadtime_disable (C++ 函数), 945
mcpwm_deadtime_enable (C++ 函数), 945
MCPWM_DEADTIME_TYPE_MAX (C++ 枚举子), 937
mcpwm_deadtime_type_t (C++ 类型), 936
MCPWM_DOWN_COUNTER (C++ 枚举子), 935
MCPWM_DUTY_MODE_0 (C++ 枚举子), 936
MCPWM_DUTY_MODE_1 (C++ 枚举子), 936
MCPWM_DUTY_MODE_MAX (C++ 枚举子), 936
mcpwm_duty_type_t (C++ 类型), 936
MCPWM_FAULT_0 (C++ 枚举子), 952
MCPWM_FAULT_1 (C++ 枚举子), 953
MCPWM_FAULT_2 (C++ 枚举子), 953
mcpwm_fault_deinit (C++ 函数), 947
mcpwm_fault_init (C++ 函数), 945
mcpwm_fault_input_level_t (C++ 类型), 954
mcpwm_fault_set_cyc_mode (C++ 函数), 946
mcpwm_fault_set_oneshot_mode (C++ 函数), 946
mcpwm_fault_signal_t (C++ 类型), 954
MCPWM_FORCE_MCPWMXA_HIGH (C 宏), 951
MCPWM_FORCE_MCPWMXA_LOW (C 宏), 951
MCPWM_FORCE_MCPWMXB_HIGH (C 宏), 951
MCPWM_FORCE_MCPWMXB_LOW (C 宏), 951
MCPWM_GEN_A (C++ 枚举子), 953
MCPWM_GEN_B (C++ 枚举子), 954

MCPWM_GEN_MAX (C++ 枚举子), 954
mcpwm_generator_t (C++ 类型), 953
mcpwm_get_duty (C++ 函数), 941
mcpwm_get_frequency (C++ 函数), 940
mcpwm_gpio_init (C++ 函数), 938
MCPWM_HAL_GENERATOR_MODE_FORCE_HIGH (C++ 枚举子), 936
MCPWM_HAL_GENERATOR_MODE_FORCE_LOW (C++ 枚举子), 936
MCPWM_HIGH_LEVEL_TGR (C++ 枚举子), 954
mcpwm_init (C++ 函数), 939
mcpwm_intr_t (C++ 类型), 935
mcpwm_io_signals_t (C++ 类型), 952
mcpwm_isr_register (C++ 函数), 949
MCPWM_LL_INTR_CAP0 (C++ 枚举子), 935
MCPWM_LL_INTR_CAP1 (C++ 枚举子), 935
MCPWM_LL_INTR_CAP2 (C++ 枚举子), 935
MCPWM_LOW_LEVEL_TGR (C++ 枚举子), 954
MCPWM_NEG_EDGE (C++ 枚举子), 937
MCPWM_NO_CHANGE_IN_MCPWMXA (C 宏), 951
MCPWM_NO_CHANGE_IN_MCPWMXB (C 宏), 951
MCPWM_ONESHOT_MODE_DIS (C++ 枚举子), 954
MCPWM_ONESHOT_MODE_EN (C++ 枚举子), 954
mcpwm_operator_t (C++ 类型), 952
MCPWM_OPR_A (C 宏), 951
MCPWM_OPR_B (C 宏), 951
MCPWM_OPR_MAX (C 宏), 951
mcpwm_output_action_t (C++ 类型), 936
mcpwm_pin_config_t (C++ 类), 949
mcpwm_pin_config_t::mcpwm0a_out_num (C++ 成员), 949
mcpwm_pin_config_t::mcpwm0b_out_num (C++ 成员), 949
mcpwm_pin_config_t::mcpwm1a_out_num (C++ 成员), 949
mcpwm_pin_config_t::mcpwm1b_out_num (C++ 成员), 949
mcpwm_pin_config_t::mcpwm2a_out_num (C++ 成员), 950
mcpwm_pin_config_t::mcpwm2b_out_num (C++ 成员), 950
mcpwm_pin_config_t::mcpwm_cap0_in_num (C++ 成员), 950
mcpwm_pin_config_t::mcpwm_cap1_in_num (C++ 成员), 950
mcpwm_pin_config_t::mcpwm_cap2_in_num (C++ 成员), 950
mcpwm_pin_config_t::mcpwm_fault0_in_num (C++ 成员), 950
mcpwm_pin_config_t::mcpwm_fault1_in_num (C++ 成员), 950
mcpwm_pin_config_t::mcpwm_fault2_in_num (C++ 成员), 950
mcpwm_pin_config_t::mcpwm_sync0_in_num (C++ 成员), 950
mcpwm_pin_config_t::mcpwm_sync1_in_num (C++ 成员), 950
mcpwm_pin_config_t::mcpwm_sync2_in_num (C++ 成员), 950
MCPWM_POS_EDGE (C++ 枚举子), 937
MCPWM_SELECT_CAP0 (C++ 枚举子), 955
MCPWM_SELECT_CAP1 (C++ 枚举子), 955
MCPWM_SELECT_CAP2 (C++ 枚举子), 955
MCPWM_SELECT_F0 (C++ 枚举子), 954
MCPWM_SELECT_F1 (C++ 枚举子), 954
MCPWM_SELECT_F2 (C++ 枚举子), 954
MCPWM_SELECT_SYNC0 (C++ 枚举子), 937
MCPWM_SELECT_SYNC1 (C++ 枚举子), 937
MCPWM_SELECT_SYNC2 (C++ 枚举子), 937
mcpwm_set_duty (C++ 函数), 939
mcpwm_set_duty_in_us (C++ 函数), 940
mcpwm_set_duty_type (C++ 函数), 940
mcpwm_set_frequency (C++ 函数), 939
mcpwm_set_pin (C++ 函数), 938
mcpwm_set_signal_high (C++ 函数), 941
mcpwm_set_signal_low (C++ 函数), 941
mcpwm_start (C++ 函数), 941
mcpwm_stop (C++ 函数), 942
MCPWM_SYNC_0 (C++ 枚举子), 952
MCPWM_SYNC_1 (C++ 枚举子), 952
MCPWM_SYNC_2 (C++ 枚举子), 952
mcpwm_sync_disable (C++ 函数), 948
mcpwm_sync_enable (C++ 函数), 948
mcpwm_sync_signal_t (C++ 类型), 937

MCPWM_TIMER_0 (C++ 枚举子), 953
MCPWM_TIMER_1 (C++ 枚举子), 953
MCPWM_TIMER_2 (C++ 枚举子), 953
MCPWM_TIMER_MAX (C++ 枚举子), 953
mcpwm_timer_t (C++ 类型), 953
MCPWM_TOG_MCPWMXA (C 宏), 951
MCPWM_TOG_MCPWMXB (C 宏), 951
MCPWM_UNIT_0 (C++ 枚举子), 953
MCPWM_UNIT_1 (C++ 枚举子), 953
MCPWM_UNIT_MAX (C++ 枚举子), 953
mcpwm_unit_t (C++ 类型), 953
MCPWM_UP_COUNTER (C++ 枚举子), 935
MCPWM_UP_DOWN_COUNTER (C++ 枚举子), 936
mdns_free (C++ 函数), 1130
mdns_handle_system_event (C++ 函数), 1136
mdns_hostname_set (C++ 函数), 1130
MDNS_IF_AP (C++ 枚举子), 1138
MDNS_IF_ETH (C++ 枚举子), 1138
mdns_if_internal (C++ 类型), 1138
MDNS_IF_MAX (C++ 枚举子), 1139
MDNS_IF_STA (C++ 枚举子), 1138
mdns_if_t (C++ 类型), 1138
mdns_init (C++ 函数), 1129
mdns_instance_name_set (C++ 函数), 1130
mdns_ip_addr_s (C++ 类), 1137
mdns_ip_addr_s::addr (C++ 成员), 1137
mdns_ip_addr_s::next (C++ 成员), 1137
mdns_ip_addr_t (C++ 类型), 1138
MDNS_IP_PROTOCOL_MAX (C++ 枚举子), 1138
mdns_ip_protocol_t (C++ 类型), 1138
MDNS_IP_PROTOCOL_V4 (C++ 枚举子), 1138
MDNS_IP_PROTOCOL_V6 (C++ 枚举子), 1138
mdns_query (C++ 函数), 1133
mdns_query_a (C++ 函数), 1135
mdns_query_aaaa (C++ 函数), 1136
mdns_query_ptr (C++ 函数), 1134
mdns_query_results_free (C++ 函数), 1134
mdns_query_srv (C++ 函数), 1134
mdns_query_txt (C++ 函数), 1135
mdns_result_s (C++ 类), 1137
mdns_result_s::addr (C++ 成员), 1137
mdns_result_s::hostname (C++ 成员), 1137
mdns_result_s::instance_name (C++ 成员), 1137
mdns_result_s::ip_protocol (C++ 成员), 1137
mdns_result_s::next (C++ 成员), 1137
mdns_result_s::port (C++ 成员), 1137
mdns_result_s::tcpip_if (C++ 成员), 1137
mdns_result_s::txt (C++ 成员), 1137
mdns_result_s::txt_count (C++ 成员), 1137
mdns_result_t (C++ 类型), 1138
mdns_service_add (C++ 函数), 1130
mdns_service_instance_name_set (C++ 函数), 1131
mdns_service_port_set (C++ 函数), 1131
mdns_service_remove (C++ 函数), 1131
mdns_service_remove_all (C++ 函数), 1133
mdns_service_txt_item_remove (C++ 函数), 1133
mdns_service_txt_item_set (C++ 函数), 1132
mdns_service_txt_set (C++ 函数), 1132
mdns_txt_item_t (C++ 类), 1136
mdns_txt_item_t::key (C++ 成员), 1137
mdns_txt_item_t::value (C++ 成员), 1137
MDNS_TYPE_A (C 宏), 1138
MDNS_TYPE_AAAA (C 宏), 1138
MDNS_TYPE_ANY (C 宏), 1138
MDNS_TYPE_NSEC (C 宏), 1138
MDNS_TYPE_OPT (C 宏), 1138
MDNS_TYPE_PTR (C 宏), 1138
MDNS_TYPE_SRV (C 宏), 1138
MDNS_TYPE_TXT (C 宏), 1138
mesh_addr_t (C++ 类型), 742
mesh_addr_t::addr (C++ 成员), 742
mesh_addr_t::mip (C++ 成员), 742
mesh_ap_cfg_t (C++ 类), 748
mesh_ap_cfg_t::max_connection (C++ 成员), 748
mesh_ap_cfg_t::password (C++ 成员), 748
MESH_ASSOC_FLAG_NETWORK_FREE (C 宏), 752
MESH_ASSOC_FLAG_ROOT_FIXED (C 宏), 752
MESH_ASSOC_FLAG_ROOTS_FOUND (C 宏), 752
MESH_ASSOC_FLAG_VOTE_IN_PROGRESS (C 宏), 752
mesh_cfg_t (C++ 类), 748
mesh_cfg_t::allow_channel_switch (C++ 成员), 748
mesh_cfg_t::channel (C++ 成员), 748

mesh_cfg_t::crypto_funcs (C++ 成员), 748
mesh_cfg_t::mesh_ap (C++ 成员), 748
mesh_cfg_t::mesh_id (C++ 成员), 748
mesh_cfg_t::router (C++ 成员), 748
MESH_DATA_DROP (C 宏), 751
MESH_DATA_ENC (C 宏), 751
MESH_DATA_FROMDS (C 宏), 751
MESH_DATA_GROUP (C 宏), 751
MESH_DATA_NONBLOCK (C 宏), 751
MESH_DATA_P2P (C 宏), 751
mesh_data_t (C++ 类), 747
mesh_data_t::data (C++ 成员), 747
mesh_data_t::proto (C++ 成员), 747
mesh_data_t::size (C++ 成员), 747
mesh_data_t::tos (C++ 成员), 747
MESH_DATA_TODS (C 宏), 751
mesh_disconnect_reason_t (C++ 类型), 755
MESH_EVENT_CHANNEL_SWITCH (C++ 枚举子), 752
mesh_event_channel_switch_t (C++ 类), 744
mesh_event_channel_switch_t::channel (C++ 成员), 744
MESH_EVENT_CHILD_CONNECTED (C++ 枚举子), 753
mesh_event_child_connected_t (C++ 类型), 752
MESH_EVENT_CHILD_DISCONNECTED (C++ 枚举子), 753
mesh_event_child_disconnected_t (C++ 类型), 752
mesh_event_connected_t (C++ 类), 744
mesh_event_connected_t::connected (C++ 成员), 744
mesh_event_connected_t::self_layer (C++ 成员), 744
mesh_event_disconnected_t (C++ 类型), 752
MESH_EVENT_FIND_NETWORK (C++ 枚举子), 754
mesh_event_find_network_t (C++ 类), 745
mesh_event_find_network_t::channel (C++ 成员), 745
mesh_event_find_network_t::router_bssid (C++ 成员), 745
mesh_event_id_t (C++ 类型), 752
mesh_event_info_t (C++ 类型), 742
mesh_event_info_t::channel_switch (C++ 成员), 742
mesh_event_info_t::child_connected (C++ 成员), 742
mesh_event_info_t::child_disconnected (C++ 成员), 742
mesh_event_info_t::connected (C++ 成员), 742
mesh_event_info_t::disconnected (C++ 成员), 742
mesh_event_info_t::find_network (C++ 成员), 743
mesh_event_info_t::layer_change (C++ 成员), 742
mesh_event_info_t::network_state (C++ 成员), 743
mesh_event_info_t::no_parent (C++ 成员), 742
mesh_event_info_t::root_addr (C++ 成员), 743
mesh_event_info_t::root_conflict (C++ 成员), 743
mesh_event_info_t::root_fixed (C++ 成员), 743
mesh_event_info_t::router_switch (C++ 成员), 743
mesh_event_info_t::routing_table (C++ 成员), 742
mesh_event_info_t::scan_done (C++ 成员), 743
mesh_event_info_t::switch_req (C++ 成员), 743
mesh_event_info_t::toDS_state (C++ 成员), 743
mesh_event_info_t::vote_started (C++ 成员), 743
MESH_EVENT_LAYER_CHANGE (C++ 枚举子), 753
mesh_event_layer_change_t (C++ 类), 744
mesh_event_layer_change_t::new_layer (C++ 成员), 745
MESH_EVENT_MAX (C++ 枚举子), 754
MESH_EVENT_NETWORK_STATE (C++ 枚举子), 754
mesh_event_network_state_t (C++ 类), 746
mesh_event_network_state_t::is_rootless (C++ 成员), 746
MESH_EVENT_NO_PARENT_FOUND (C++ 枚举子), 753
mesh_event_no_parent_found_t (C++ 类), 744
mesh_event_no_parent_found_t::scan_times (C++ 成员), 744
MESH_EVENT_PARENT_CONNECTED (C++ 枚举子), 753

MESH_EVENT_PARENT_DISCONNECTED (C++ 枚举子), 753
MESH_EVENT_ROOT_ADDRESS (C++ 枚举子), 753
mesh_event_root_address_t (C++ 类型), 752
MESH_EVENT_ROOT_ASKED_YIELD (C++ 枚举子), 753
mesh_event_root_conflict_t (C++ 类), 745
mesh_event_root_conflict_t::addr (C++ 成员), 746
mesh_event_root_conflict_t::capacity (C++ 成员), 746
mesh_event_root_conflict_t::rssi (C++ 成员), 746
MESH_EVENT_ROOT_FIXED (C++ 枚举子), 753
mesh_event_root_fixed_t (C++ 类), 746
mesh_event_root_fixed_t::is_fixed (C++ 成员), 746
MESH_EVENT_ROOT_SWITCH_ACK (C++ 枚举子), 753
MESH_EVENT_ROOT_SWITCH_REQ (C++ 枚举子), 753
mesh_event_root_switch_req_t (C++ 类), 745
mesh_event_root_switch_req_t::rc_addr (C++ 成员), 745
mesh_event_root_switch_req_t::reason (C++ 成员), 745
MESH_EVENT_ROUTER_SWITCH (C++ 枚举子), 754
mesh_event_router_switch_t (C++ 类型), 752
MESH_EVENT_ROUTING_TABLE_ADD (C++ 枚举子), 753
mesh_event_routing_table_change_t (C++ 类), 746
mesh_event_routing_table_change_t::rt_size_change (C++ 成员), 746
mesh_event_routing_table_change_t::rt_size_new (C++ 成员), 746
MESH_EVENT_ROUTING_TABLE_REMOVE (C++ 枚举子), 753
MESH_EVENT_SCAN_DONE (C++ 枚举子), 754
mesh_event_scan_done_t (C++ 类), 746
mesh_event_scan_done_t::number (C++ 成员), 746
MESH_EVENT_STARTED (C++ 枚举子), 752
MESH_EVENT_STOP_RECONNECTION (C++ 枚举子), 754
MESH_EVENT_STOPPED (C++ 枚举子), 752
MESH_EVENT_TODS_STATE (C++ 枚举子), 753
mesh_event_toDS_state_t (C++ 类型), 756
MESH_EVENT_VOTE_STARTED (C++ 枚举子), 753
mesh_event_vote_started_t (C++ 类), 745
mesh_event_vote_started_t::attempts (C++ 成员), 745
mesh_event_vote_started_t::rc_addr (C++ 成员), 745
mesh_event_vote_started_t::reason (C++ 成员), 745
MESH_EVENT_VOTE_STOPPED (C++ 枚举子), 753
MESH_IDLE (C++ 枚举子), 754
MESH_INIT_CONFIG_DEFAULT (C 宏), 752
MESH_LEAF (C++ 枚举子), 754
MESH_MPS (C 宏), 750
MESH_MTU (C 宏), 750
MESH_NODE (C++ 枚举子), 754
MESH_OPT_RECV_DS_ADDR (C 宏), 752
MESH_OPT_SEND_GROUP (C 宏), 751
mesh_opt_t (C++ 类), 746
mesh_opt_t::len (C++ 成员), 747
mesh_opt_t::type (C++ 成员), 747
mesh_opt_t::val (C++ 成员), 747
MESH_PROTO_BIN (C++ 枚举子), 754
MESH_PROTO_HTTP (C++ 枚举子), 754
MESH_PROTO_JSON (C++ 枚举子), 755
MESH_PROTO_MQTT (C++ 枚举子), 755
mesh_proto_t (C++ 类型), 754
mesh_rc_config_t (C++ 类型), 743
mesh_rc_config_t::attempts (C++ 成员), 743
mesh_rc_config_t::rc_addr (C++ 成员), 743
MESH_REASON_CYCLIC (C++ 枚举子), 755
MESH_REASON_DIFF_ID (C++ 枚举子), 755
MESH_REASON_EMPTY_PASSWORD (C++ 枚举子), 756
MESH_REASON_IE_UNKNOWN (C++ 枚举子), 756
MESH_REASON_LEAF (C++ 枚举子), 755
MESH_REASON_PARENT_IDLE (C++ 枚举子), 755
MESH_REASON_PARENT_STOPPED (C++ 枚举子), 755
MESH_REASON_PARENT_UNENCRYPTED (C++ 枚举子), 756
MESH_REASON_PARENT_WORSE (C++ 枚举子), 756

MESH_REASON_ROOTS (C++ 枚举子), 755
MESH_REASON_SCAN_FAIL (C++ 枚举子), 756
MESH_REASON_WAIVE_ROOT (C++ 枚举子), 756
MESH_ROOT (C++ 枚举子), 754
MESH_ROOT_LAYER (C 宏), 750
mesh_router_t (C++ 类), 747
mesh_router_t::allow_router_switch (C++ 成员), 747
mesh_router_t::bssid (C++ 成员), 747
mesh_router_t::password (C++ 成员), 747
mesh_router_t::ssid (C++ 成员), 747
mesh_router_t::ssid_len (C++ 成员), 747
mesh_rx_pending_t (C++ 类), 749
mesh_rx_pending_t::toDS (C++ 成员), 749
mesh_rx_pending_t::toSelf (C++ 成员), 749
MESH_TODS_REACHABLE (C++ 枚举子), 756
MESH_TODS_UNREACHABLE (C++ 枚举子), 756
MESH_TOS_DEF (C++ 枚举子), 755
MESH_TOS_E2E (C++ 枚举子), 755
MESH_TOS_P2P (C++ 枚举子), 755
mesh_tos_t (C++ 类型), 755
mesh_tx_pending_t (C++ 类), 749
mesh_tx_pending_t::broadcast (C++ 成员), 749
mesh_tx_pending_t::mgmt (C++ 成员), 749
mesh_tx_pending_t::to_child (C++ 成员), 749
mesh_tx_pending_t::to_child_p2p (C++ 成员), 749
mesh_tx_pending_t::to_parent (C++ 成员), 749
mesh_tx_pending_t::to_parent_p2p (C++ 成员), 749
mesh_type_t (C++ 类型), 754
MESH_VOTE_REASON_CHILD_INITIATED (C++ 枚举子), 755
MESH_VOTE_REASON_ROOT_INITIATED (C++ 枚举子), 755
mesh_vote_reason_t (C++ 类型), 755
mesh_vote_t (C++ 类), 748
mesh_vote_t::config (C++ 成员), 749
mesh_vote_t::is_rc_specified (C++ 成员), 749
mesh_vote_t::percentage (C++ 成员), 749
mip_t (C++ 类), 744
mip_t::ip4 (C++ 成员), 744
mip_t::port (C++ 成员), 744
MQTT_CONNECTION_ACCEPTED (C++ 枚举子), 1233
MQTT_CONNECTION_REFUSE_BAD_USERNAME (C++ 枚举子), 1233
MQTT_CONNECTION_REFUSE_ID_REJECTED (C++ 枚举子), 1233
MQTT_CONNECTION_REFUSE_NOT_AUTHORIZED (C++ 枚举子), 1233
MQTT_CONNECTION_REFUSE_PROTOCOL (C++ 枚举子), 1233
MQTT_CONNECTION_REFUSE_SERVER_UNAVAILABLE (C++ 枚举子), 1233
MQTT_ERROR_TYPE_CONNECTION_REFUSED (C++ 枚举子), 1233
MQTT_ERROR_TYPE_ESP_TLS (C++ 枚举子), 1233
MQTT_ERROR_TYPE_NONE (C++ 枚举子), 1233
MQTT_EVENT_ANY (C++ 枚举子), 1232
MQTT_EVENT_BEFORE_CONNECT (C++ 枚举子), 1233
mqtt_event_callback_t (C++ 类型), 1231
MQTT_EVENT_CONNECTED (C++ 枚举子), 1232
MQTT_EVENT_DATA (C++ 枚举子), 1232
MQTT_EVENT_DISCONNECTED (C++ 枚举子), 1232
MQTT_EVENT_ERROR (C++ 枚举子), 1232
MQTT_EVENT_PUBLISHED (C++ 枚举子), 1232
MQTT_EVENT_SUBSCRIBED (C++ 枚举子), 1232
MQTT_EVENT_UNSUBSCRIBED (C++ 枚举子), 1232
MQTT_PROTOCOL_UNDEFINED (C++ 枚举子), 1234
MQTT_PROTOCOL_V_3_1 (C++ 枚举子), 1234
MQTT_PROTOCOL_V_3_1_1 (C++ 枚举子), 1234
MQTT_TRANSPORT_OVER_SSL (C++ 枚举子), 1233
MQTT_TRANSPORT_OVER_TCP (C++ 枚举子), 1233
MQTT_TRANSPORT_OVER_WS (C++ 枚举子), 1233
MQTT_TRANSPORT_OVER_WSS (C++ 枚举子), 1233
MQTT_TRANSPORT_UNKNOWN (C++ 枚举子), 1233
multi_heap_aligned_alloc (C++ 函数), 1629
multi_heap_aligned_free (C++ 函数), 1629
multi_heap_check (C++ 函数), 1631
multi_heap_dump (C++ 函数), 1631
multi_heap_free (C++ 函数), 1629
multi_heap_free_size (C++ 函数), 1631
multi_heap_get_allocated_size (C++ 函数), 1630

multi_heap_get_info (C++ 函数), 1632
multi_heap_handle_t (C++ 类型), 1633
multi_heap_info_t (C++ 类), 1632
multi_heap_info_t::allocated_blocks (C++ 成员), 1633
multi_heap_info_t::free_blocks (C++ 成员), 1633
multi_heap_info_t::largest_free_block (C++ 成员), 1632
multi_heap_info_t::minimum_free_bytes (C++ 成员), 1633
multi_heap_info_t::total_allocated_bytes (C++ 成员), 1632
multi_heap_info_t::total_blocks (C++ 成员), 1633
multi_heap_info_t::total_free_bytes (C++ 成员), 1632
multi_heap_malloc (C++ 函数), 1629
multi_heap_minimum_free_size (C++ 函数), 1632
multi_heap_realloc (C++ 函数), 1630
multi_heap_register (C++ 函数), 1630
multi_heap_set_lock (C++ 函数), 1631

N

name_uuid (C++ 类), 1282
name_uuid::name (C++ 成员), 1282
name_uuid::uuid (C++ 成员), 1282
nvs_close (C++ 函数), 1365
nvs_commit (C++ 函数), 1364
NVS_DEFAULT_PART_NAME (C 宏), 1370
nvs_entry_find (C++ 函数), 1367
nvs_entry_info (C++ 函数), 1368
nvs_entry_info_t (C++ 类), 1368
nvs_entry_info_t::key (C++ 成员), 1368
nvs_entry_info_t::namespace_name (C++ 成员), 1368
nvs_entry_info_t::type (C++ 成员), 1368
nvs_entry_next (C++ 函数), 1367
nvs_erase_all (C++ 函数), 1364
nvs_erase_key (C++ 函数), 1364
nvs_flash_deinit (C++ 函数), 1356
nvs_flash_deinit_partition (C++ 函数), 1356
nvs_flash_erase (C++ 函数), 1356
nvs_flash_erase_partition (C++ 函数), 1356
nvs_flash_generate_keys (C++ 函数), 1357
nvs_flash_init (C++ 函数), 1355
nvs_flash_init_partition (C++ 函数), 1355
nvs_flash_read_security_cfg (C++ 函数), 1358
nvs_flash_secure_init (C++ 函数), 1357
nvs_flash_secure_init_partition (C++ 函数), 1357
nvs_get_blob (C++ 函数), 1362
nvs_get_i16 (C++ 函数), 1360
nvs_get_i32 (C++ 函数), 1360
nvs_get_i64 (C++ 函数), 1360
nvs_get_i8 (C++ 函数), 1359
nvs_get_stats (C++ 函数), 1365
nvs_get_str (C++ 函数), 1360
nvs_get_u16 (C++ 函数), 1360
nvs_get_u32 (C++ 函数), 1360
nvs_get_u64 (C++ 函数), 1360
nvs_get_u8 (C++ 函数), 1360
nvs_get_used_entry_count (C++ 函数), 1366
nvs_handle (C++ 类型), 1370
nvs_handle_t (C++ 类型), 1370
nvs_iterator_t (C++ 类型), 1371
NVS_KEY_SIZE (C 宏), 1358
nvs_open (C++ 函数), 1362
nvs_open_from_partition (C++ 函数), 1362
nvs_open_mode (C++ 类型), 1371
nvs_open_mode_t (C++ 类型), 1371
NVS_PART_NAME_MAX_SIZE (C 宏), 1370
NVS_READONLY (C++ 枚举子), 1371
NVS_READWRITE (C++ 枚举子), 1371
nvs_release_iterator (C++ 函数), 1368
nvs_sec_cfg_t (C++ 类), 1358
nvs_sec_cfg_t::eky (C++ 成员), 1358
nvs_sec_cfg_t::tky (C++ 成员), 1358
nvs_set_blob (C++ 函数), 1363
nvs_set_i16 (C++ 函数), 1359
nvs_set_i32 (C++ 函数), 1359
nvs_set_i64 (C++ 函数), 1359
nvs_set_i8 (C++ 函数), 1359
nvs_set_str (C++ 函数), 1359

nvs_set_u16 (C++ 函数), 1359
nvs_set_u32 (C++ 函数), 1359
nvs_set_u64 (C++ 函数), 1359
nvs_set_u8 (C++ 函数), 1359
nvs_stats_t (C++ 类), 1368
nvs_stats_t::free_entries (C++ 成员), 1368
nvs_stats_t::namespace_count (C++ 成员), 1369
nvs_stats_t::total_entries (C++ 成员), 1369
nvs_stats_t::used_entries (C++ 成员), 1368
NVS_TYPE_ANY (C++ 枚举子), 1372
NVS_TYPE_BLOB (C++ 枚举子), 1371
NVS_TYPE_I16 (C++ 枚举子), 1371
NVS_TYPE_I32 (C++ 枚举子), 1371
NVS_TYPE_I64 (C++ 枚举子), 1371
NVS_TYPE_I8 (C++ 枚举子), 1371
NVS_TYPE_STR (C++ 枚举子), 1371
nvs_type_t (C++ 类型), 1371
NVS_TYPE_U16 (C++ 枚举子), 1371
NVS_TYPE_U32 (C++ 枚举子), 1371
NVS_TYPE_U64 (C++ 枚举子), 1371
NVS_TYPE_U8 (C++ 枚举子), 1371

O

OTA_SIZE_UNKNOWN (C 宏), 1702

P

PCNT_CHANNEL_0 (C++ 枚举子), 967
PCNT_CHANNEL_1 (C++ 枚举子), 967
PCNT_CHANNEL_MAX (C++ 枚举子), 967
pcnt_channel_t (C++ 类型), 966
pcnt_config_t (C++ 类), 965
pcnt_config_t::channel (C++ 成员), 965
pcnt_config_t::counter_h_lim (C++ 成员), 965
pcnt_config_t::counter_l_lim (C++ 成员), 965
pcnt_config_t::ctrl_gpio_num (C++ 成员), 965
pcnt_config_t::hctrl_mode (C++ 成员), 965
pcnt_config_t::lctrl_mode (C++ 成员), 965
pcnt_config_t::neg_mode (C++ 成员), 965
pcnt_config_t::pos_mode (C++ 成员), 965
pcnt_config_t::pulse_gpio_num (C++ 成员), 965
pcnt_config_t::unit (C++ 成员), 965
PCNT_COUNT_DEC (C++ 枚举子), 966
PCNT_COUNT_DIS (C++ 枚举子), 966
PCNT_COUNT_INC (C++ 枚举子), 966
PCNT_COUNT_MAX (C++ 枚举子), 966
pcnt_count_mode_t (C++ 类型), 966
pcnt_counter_clear (C++ 函数), 958
pcnt_counter_pause (C++ 函数), 958
pcnt_counter_resume (C++ 函数), 958
pcnt_ctrl_mode_t (C++ 类型), 966
pcnt_event_disable (C++ 函数), 960
pcnt_event_enable (C++ 函数), 959
PCNT_EVT_H_LIM (C++ 枚举子), 967
PCNT_EVT_L_LIM (C++ 枚举子), 967
PCNT_EVT_MAX (C++ 枚举子), 967
PCNT_EVT_THRES_0 (C++ 枚举子), 967
PCNT_EVT_THRES_1 (C++ 枚举子), 967
pcnt_evt_type_t (C++ 类型), 967
PCNT_EVT_ZERO (C++ 枚举子), 967
pcnt_filter_disable (C++ 函数), 962
pcnt_filter_enable (C++ 函数), 962
pcnt_get_counter_value (C++ 函数), 957
pcnt_get_event_value (C++ 函数), 960
pcnt_get_filter_value (C++ 函数), 962
pcnt_intr_disable (C++ 函数), 959
pcnt_intr_enable (C++ 函数), 959
pcnt_isr_handle_t (C++ 类型), 965
pcnt_isr_handler_add (C++ 函数), 963
pcnt_isr_handler_remove (C++ 函数), 964
pcnt_isr_register (C++ 函数), 961
pcnt_isr_service_install (C++ 函数), 964
pcnt_isr_service_uninstall (C++ 函数), 964
PCNT_MODE_DISABLE (C++ 枚举子), 966
PCNT_MODE_KEEP (C++ 枚举子), 966
PCNT_MODE_MAX (C++ 枚举子), 966
PCNT_MODE_REVERSE (C++ 枚举子), 966
pcnt_port_t (C++ 类型), 966
pcnt_set_event_value (C++ 函数), 960
pcnt_set_filter_value (C++ 函数), 962
pcnt_set_mode (C++ 函数), 963
pcnt_set_pin (C++ 函数), 961
pcnt_unit_config (C++ 函数), 957
pcnt_unit_t (C++ 类型), 966
pcQueueGetName (C++ 函数), 1514

pcTaskGetTaskName (C++ 函数), 1487
pcTimerGetTimerName (C++ 函数), 1564
PDM_PCM_CONV_DISABLE (C++ 枚举子), 907
PDM_PCM_CONV_ENABLE (C++ 枚举子), 907
pdm_pcm_conv_t (C++ 类型), 907
PDM_SAMPLE_RATE_RATIO_128 (C++ 枚举子), 907
PDM_SAMPLE_RATE_RATIO_64 (C++ 枚举子), 907
pdm_sample_rate_ratio_t (C++ 类型), 907
PendedFunction_t (C++ 类型), 1578
protocomm_add_endpoint (C++ 函数), 1272
protocomm_ble_config (C++ 类), 1282
protocomm_ble_config::device_name (C++ 成员), 1282
protocomm_ble_config::nu_lookup (C++ 成员), 1282
protocomm_ble_config::nu_lookup_count (C++ 成员), 1282
protocomm_ble_config::service_uuid (C++ 成员), 1282
protocomm_ble_config_t (C++ 类型), 1282
protocomm_ble_name_uuid_t (C++ 类型), 1282
protocomm_ble_start (C++ 函数), 1281
protocomm_ble_stop (C++ 函数), 1281
protocomm_close_session (C++ 函数), 1274
protocomm_delete (C++ 函数), 1272
protocomm_http_server_config_t (C++ 类), 1280
protocomm_http_server_config_t::port (C++ 成员), 1280
protocomm_http_server_config_t::stack_size (C++ 成员), 1280
protocomm_http_server_config_t::task_priority (C++ 成员), 1280
protocomm_httpd_config_data_t (C++ 类型), 1280
protocomm_httpd_config_data_t::config (C++ 成员), 1280
protocomm_httpd_config_data_t::handle (C++ 成员), 1280
protocomm_httpd_config_t (C++ 类), 1280
protocomm_httpd_config_t::data (C++ 成员), 1280
protocomm_httpd_config_t::ext_handle_provided (C++ 成员), 1280
PROTOCOMM_HTTPD_DEFAULT_CONFIG (C 宏), 1280
protocomm_httpd_start (C++ 函数), 1279
protocomm_httpd_stop (C++ 函数), 1279
protocomm_new (C++ 函数), 1272
protocomm_open_session (C++ 函数), 1273
protocomm_remove_endpoint (C++ 函数), 1273
protocomm_req_handle (C++ 函数), 1274
protocomm_req_handler_t (C++ 类型), 1277
protocomm_security (C++ 类), 1277
protocomm_security::cleanup (C++ 成员), 1277
protocomm_security::close_transport_session (C++ 成员), 1278
protocomm_security::decrypt (C++ 成员), 1278
protocomm_security::encrypt (C++ 成员), 1278
protocomm_security::init (C++ 成员), 1277
protocomm_security::new_transport_session (C++ 成员), 1278
protocomm_security::security_req_handler (C++ 成员), 1278
protocomm_security::ver (C++ 成员), 1277
protocomm_security_handle_t (C++ 类型), 1278
protocomm_security_pop (C++ 类), 1277
protocomm_security_pop::data (C++ 成员), 1277
protocomm_security_pop::len (C++ 成员), 1277
protocomm_security_pop_t (C++ 类型), 1278
protocomm_security_t (C++ 类型), 1278
protocomm_set_security (C++ 函数), 1275
protocomm_set_version (C++ 函数), 1276
protocomm_t (C++ 类型), 1277
protocomm_unset_security (C++ 函数), 1275
protocomm_unset_version (C++ 函数), 1276
PROV_DATA_FLAGS_FLAG (C 宏), 416
PROV_DATA_IV_INDEX_FLAG (C 宏), 416
PROV_DATA_NET_IDX_FLAG (C 宏), 416
PROXY_FILTER_BLACKLIST (C++ 枚举子), 434
PROXY_FILTER_WHITELIST (C++ 枚举子), 434
psk_hint_key_t (C++ 类型), 1150
psk_key_hint (C++ 类), 1146
psk_key_hint::hint (C++ 成员), 1146
psk_key_hint::key (C++ 成员), 1146
psk_key_hint::key_size (C++ 成员), 1146

PTHREAD_STACK_MIN (C 宏), 1452
pvTaskGetThreadLocalStoragePointer (C++ 函数), 1489
pvTimerGetTimerID (C++ 函数), 1560
pxTaskGetStackStart (C++ 函数), 1487

Q

QueueHandle_t (C++ 类型), 1535
QueueSetHandle_t (C++ 类型), 1535
QueueSetMemberHandle_t (C++ 类型), 1535

R

R0 (C 宏), 2231
R1 (C 宏), 2231
R2 (C 宏), 2231
R3 (C 宏), 2231
RINGBUF_TYPE_ALLOWSPLIT (C++ 枚举子), 1612
RINGBUF_TYPE_BYTEBUF (C++ 枚举子), 1612
RINGBUF_TYPE_MAX (C++ 枚举子), 1613
RINGBUF_TYPE_NOSPLIT (C++ 枚举子), 1612
RingbufferType_t (C++ 类型), 1612
RingbufHandle_t (C++ 类型), 1612
RMT_BASECLK_APB (C++ 枚举子), 991
RMT_BASECLK_MAX (C++ 枚举子), 991
RMT_BASECLK_REF (C++ 枚举子), 991
RMT_CARRIER_LEVEL_HIGH (C++ 枚举子), 992
RMT_CARRIER_LEVEL_LOW (C++ 枚举子), 992
RMT_CARRIER_LEVEL_MAX (C++ 枚举子), 992
rmt_carrier_level_t (C++ 类型), 992
RMT_CHANNEL_BUSY (C++ 枚举子), 992
RMT_CHANNEL_IDLE (C++ 枚举子), 992
rmt_channel_status_result_t (C++ 类), 990
rmt_channel_status_result_t::status (C++ 成员), 990
rmt_channel_status_t (C++ 类型), 992
rmt_channel_t (C++ 类型), 991
RMT_CHANNEL_UNINIT (C++ 枚举子), 992
rmt_clr_intr_enable_mask (C++ 函数), 981
rmt_config (C++ 函数), 983
rmt_config_t (C++ 类), 988
rmt_config_t::channel (C++ 成员), 989
rmt_config_t::clk_div (C++ 成员), 989
rmt_config_t::gpio_num (C++ 成员), 989
rmt_config_t::mem_block_num (C++ 成员), 989
rmt_config_t::rmt_mode (C++ 成员), 989
rmt_config_t::rx_config (C++ 成员), 989
rmt_config_t::tx_config (C++ 成员), 989
RMT_DATA_MODE_FIFO (C++ 枚举子), 991
RMT_DATA_MODE_MAX (C++ 枚举子), 991
RMT_DATA_MODE_MEM (C++ 枚举子), 991
rmt_data_mode_t (C++ 类型), 991
RMT_DEFAULT_CONFIG_RX (C 宏), 989
RMT_DEFAULT_CONFIG_TX (C 宏), 989
rmt_driver_install (C++ 函数), 984
rmt_driver_uninstall (C++ 函数), 984
rmt_fill_tx_items (C++ 函数), 984
rmt_get_channel_status (C++ 函数), 985
rmt_get_clk_div (C++ 函数), 973
rmt_get_counter_clock (C++ 函数), 985
rmt_get_idle_level (C++ 函数), 980
rmt_get_mem_block_num (C++ 函数), 975
rmt_get_mem_pd (C++ 函数), 976
rmt_get_memory_owner (C++ 函数), 978
rmt_get_ringbuf_handle (C++ 函数), 986
rmt_get_rx_idle_thresh (C++ 函数), 974
rmt_get_source_clk (C++ 函数), 979
rmt_get_status (C++ 函数), 980
rmt_get_tx_loop_mode (C++ 函数), 979
RMT_IDLE_LEVEL_HIGH (C++ 枚举子), 992
RMT_IDLE_LEVEL_LOW (C++ 枚举子), 992
RMT_IDLE_LEVEL_MAX (C++ 枚举子), 992
rmt_idle_level_t (C++ 类型), 992
rmt_isr_deregister (C++ 函数), 983
rmt_isr_handle_t (C++ 类型), 990
rmt_isr_register (C++ 函数), 983
RMT_MEM_ITEM_NUM (C 宏), 989
RMT_MEM_OWNER_MAX (C++ 枚举子), 991
RMT_MEM_OWNER_RX (C++ 枚举子), 991
rmt_mem_owner_t (C++ 类型), 991
RMT_MEM_OWNER_TX (C++ 枚举子), 991
rmt_memory_rw_rst (C++ 函数), 977
RMT_MODE_MAX (C++ 枚举子), 992
RMT_MODE_RX (C++ 枚举子), 992
rmt_mode_t (C++ 类型), 991

RMT_MODE_TX (C++ 枚举子), 991
rmt_register_tx_end_callback (C++ 函数), 987
rmt_rx_config_t (C++ 类), 988
rmt_rx_config_t::filter_en (C++ 成员), 988
rmt_rx_config_t::filter_ticks_thresh (C++ 成员), 988
rmt_rx_config_t::idle_threshold (C++ 成员), 988
rmt_rx_start (C++ 函数), 977
rmt_rx_stop (C++ 函数), 977
rmt_set_clk_div (C++ 函数), 973
rmt_set_err_intr_en (C++ 函数), 981
rmt_set_idle_level (C++ 函数), 980
rmt_set_intr_enable_mask (C++ 函数), 981
rmt_set_mem_block_num (C++ 函数), 974
rmt_set_mem_pd (C++ 函数), 976
rmt_set_memory_owner (C++ 函数), 978
rmt_set_pin (C++ 函数), 982
rmt_set_rx_filter (C++ 函数), 979
rmt_set_rx_idle_thresh (C++ 函数), 974
rmt_set_rx_intr_en (C++ 函数), 981
rmt_set_source_clk (C++ 函数), 979
rmt_set_tx_carrier (C++ 函数), 975
rmt_set_tx_intr_en (C++ 函数), 982
rmt_set_tx_loop_mode (C++ 函数), 978
rmt_set_tx_thr_intr_en (C++ 函数), 982
rmt_source_clk_t (C++ 类型), 991
rmt_translator_init (C++ 函数), 986
rmt_tx_config_t (C++ 类), 988
rmt_tx_config_t::carrier_duty_percent (C++ 成员), 988
rmt_tx_config_t::carrier_en (C++ 成员), 988
rmt_tx_config_t::carrier_freq_hz (C++ 成员), 988
rmt_tx_config_t::carrier_level (C++ 成员), 988
rmt_tx_config_t::idle_level (C++ 成员), 988
rmt_tx_config_t::idle_output_en (C++ 成员), 988
rmt_tx_config_t::loop_en (C++ 成员), 988
rmt_tx_end_callback_t (C++ 类), 989
rmt_tx_end_callback_t::arg (C++ 成员), 989
rmt_tx_end_callback_t::function (C++ 成员), 989
rmt_tx_end_fn_t (C++ 类型), 990
rmt_tx_start (C++ 函数), 976
rmt_tx_stop (C++ 函数), 977
rmt_wait_tx_done (C++ 函数), 986
rmt_write_items (C++ 函数), 985
rmt_write_sample (C++ 函数), 987
ROLE_FAST_PROV (C++ 枚举子), 434
ROLE_NODE (C++ 枚举子), 434
ROLE_PROVISIONER (C++ 枚举子), 434
rtc_gpio_deinit (C++ 函数), 866
rtc_gpio_force_hold_all (C++ 函数), 869
rtc_gpio_force_hold_dis_all (C++ 函数), 870
rtc_gpio_get_drive_capability (C++ 函数), 870
rtc_gpio_get_level (C++ 函数), 866
rtc_gpio_hold_dis (C++ 函数), 869
rtc_gpio_hold_en (C++ 函数), 868
rtc_gpio_init (C++ 函数), 865
RTC_GPIO_IS_VALID_GPIO (C 宏), 871
rtc_gpio_is_valid_gpio (C++ 函数), 865
rtc_gpio_isolate (C++ 函数), 869
RTC_GPIO_MODE_DISABLED (C++ 枚举子), 871
RTC_GPIO_MODE_INPUT_ONLY (C++ 枚举子), 871
RTC_GPIO_MODE_INPUT_OUTPUT (C++ 枚举子), 871
RTC_GPIO_MODE_INPUT_OUTPUT_OD (C++ 枚举子), 871
RTC_GPIO_MODE_OUTPUT_OD (C++ 枚举子), 871
RTC_GPIO_MODE_OUTPUT_ONLY (C++ 枚举子), 871
rtc_gpio_mode_t (C++ 类型), 871
rtc_gpio_pulldown_dis (C++ 函数), 868
rtc_gpio_pulldown_en (C++ 函数), 868
rtc_gpio_pullup_dis (C++ 函数), 868
rtc_gpio_pullup_en (C++ 函数), 867
rtc_gpio_set_direction (C++ 函数), 867
rtc_gpio_set_direction_in_sleep (C++ 函数), 867
rtc_gpio_set_drive_capability (C++ 函数), 870
rtc_gpio_set_level (C++ 函数), 866
rtc_gpio_wakeup_disable (C++ 函数), 871
rtc_gpio_wakeup_enable (C++ 函数), 870
rtc_io_number_get (C++ 函数), 865

RTC_SLOW_MEM (*C 宏*), 2235

S

sample_to_rmt_t (*C++ 类型*), 990
SC_EVENT_FOUND_CHANNEL (*C++ 枚举子*), 703
SC_EVENT_GOT_SSID_PSWD (*C++ 枚举子*), 704
SC_EVENT_SCAN_DONE (*C++ 枚举子*), 703
SC_EVENT_SEND_ACK_DONE (*C++ 枚举子*), 704
SC_TYPE_AIRKISS (*C++ 枚举子*), 703
SC_TYPE_ESPTOUCH (*C++ 枚举子*), 703
SC_TYPE_ESPTOUCH_AIRKISS (*C++ 枚举子*), 703
sdio_event_cb_t (*C++ 类型*), 1026
sdio_slave_buf_handle_t (*C++ 类型*), 1026
sdio_slave_clear_host_int (*C++ 函数*), 1024
sdio_slave_config_t (*C++ 类*), 1025
sdio_slave_config_t::event_cb (*C++ 成员*), 1025
sdio_slave_config_t::flags (*C++ 成员*), 1025
sdio_slave_config_t::recv_buffer_size (*C++ 成员*), 1025
sdio_slave_config_t::send_queue_size (*C++ 成员*), 1025
sdio_slave_config_t::sending_mode (*C++ 成员*), 1025
sdio_slave_config_t::timing (*C++ 成员*), 1025
sdio_slave_deinit (*C++ 函数*), 1020
SDIO_SLAVE_FLAG_DAT2_DISABLED (*C 宏*), 1026
SDIO_SLAVE_FLAG_HOST_INTR_DISABLED (*C 宏*), 1026
SDIO_SLAVE_FLAG_INTERNAL_PULLUP (*C 宏*), 1026
sdio_slave_get_host_intena (*C++ 函数*), 1024
SDIO_SLAVE_HOSTINT_BIT0 (*C++ 枚举子*), 1019
SDIO_SLAVE_HOSTINT_BIT1 (*C++ 枚举子*), 1019
SDIO_SLAVE_HOSTINT_BIT2 (*C++ 枚举子*), 1019
SDIO_SLAVE_HOSTINT_BIT3 (*C++ 枚举子*), 1019
SDIO_SLAVE_HOSTINT_BIT4 (*C++ 枚举子*), 1019
SDIO_SLAVE_HOSTINT_BIT5 (*C++ 枚举子*), 1019
SDIO_SLAVE_HOSTINT_BIT6 (*C++ 枚举子*), 1019
SDIO_SLAVE_HOSTINT_BIT7 (*C++ 枚举子*), 1019
SDIO_SLAVE_HOSTINT_SEND_NEW_PACKET (*C++ 枚举子*), 1019
sdio_slave_hostint_t (*C++ 类型*), 1019
sdio_slave_initialize (*C++ 函数*), 1020
sdio_slave_read_reg (*C++ 函数*), 1023
sdio_slave_recv (*C++ 函数*), 1022
sdio_slave_recv_get_buf (*C++ 函数*), 1022
sdio_slave_recv_load_buf (*C++ 函数*), 1021
SDIO_SLAVE_RECV_MAX_BUFFER (*C 宏*), 1026
sdio_slave_recv_register_buf (*C++ 函数*), 1021
sdio_slave_recv_unregister_buf (*C++ 函数*), 1021
sdio_slave_reset (*C++ 函数*), 1021
sdio_slave_send_get_finished (*C++ 函数*), 1023
sdio_slave_send_host_int (*C++ 函数*), 1024
SDIO_SLAVE_SEND_PACKET (*C++ 枚举子*), 1020
sdio_slave_send_queue (*C++ 函数*), 1022
SDIO_SLAVE_SEND_STREAM (*C++ 枚举子*), 1020
sdio_slave_sending_mode_t (*C++ 类型*), 1020
sdio_slave_set_host_intena (*C++ 函数*), 1024
sdio_slave_start (*C++ 函数*), 1021
sdio_slave_stop (*C++ 函数*), 1021
SDIO_SLAVE_TIMING_NSEND_NSAMPLE (*C++ 枚举子*), 1020
SDIO_SLAVE_TIMING_NSEND_PSAMPLE (*C++ 枚举子*), 1020
SDIO_SLAVE_TIMING_PSEND_NSAMPLE (*C++ 枚举子*), 1020
SDIO_SLAVE_TIMING_PSEND_PSAMPLE (*C++ 枚举子*), 1019
sdio_slave_timing_t (*C++ 类型*), 1019
sdio_slave_transmit (*C++ 函数*), 1023
sdio_slave_wait_int (*C++ 函数*), 1025
sdio_slave_write_reg (*C++ 函数*), 1024
sdmmc_card_init (*C++ 函数*), 1336
sdmmc_card_print_info (*C++ 函数*), 1336
sdmmc_card_t (*C++ 类*), 1345
sdmmc_card_t::cid (*C++ 成员*), 1345
sdmmc_card_t::csd (*C++ 成员*), 1345
sdmmc_card_t::ext_csd (*C++ 成员*), 1346
sdmmc_card_t::host (*C++ 成员*), 1345
sdmmc_card_t::is_ddr (*C++ 成员*), 1346
sdmmc_card_t::is_mem (*C++ 成员*), 1346
sdmmc_card_t::is_mmc (*C++ 成员*), 1346
sdmmc_card_t::is_sdio (*C++ 成员*), 1346

sdmmc_card_t::log_bus_width (C++ 成员), 1346
sdmmc_card_t::max_freq_khz (C++ 成员), 1346
sdmmc_card_t::num_io_functions (C++ 成员), 1346
sdmmc_card_t::ocr (C++ 成员), 1345
sdmmc_card_t::raw_cid (C++ 成员), 1345
sdmmc_card_t::rca (C++ 成员), 1346
sdmmc_card_t::reserved (C++ 成员), 1346
sdmmc_card_t::scr (C++ 成员), 1345
sdmmc_cid_t (C++ 类), 1342
sdmmc_cid_t::date (C++ 成员), 1343
sdmmc_cid_t::mfg_id (C++ 成员), 1342
sdmmc_cid_t::name (C++ 成员), 1343
sdmmc_cid_t::oem_id (C++ 成员), 1342
sdmmc_cid_t::revision (C++ 成员), 1343
sdmmc_cid_t::serial (C++ 成员), 1343
sdmmc_command_t (C++ 类), 1343
sdmmc_command_t::arg (C++ 成员), 1344
sdmmc_command_t::blklen (C++ 成员), 1344
sdmmc_command_t::data (C++ 成员), 1344
sdmmc_command_t::datalen (C++ 成员), 1344
sdmmc_command_t::error (C++ 成员), 1344
sdmmc_command_t::flags (C++ 成员), 1344
sdmmc_command_t::opcode (C++ 成员), 1344
sdmmc_command_t::response (C++ 成员), 1344
sdmmc_command_t::timeout_ms (C++ 成员), 1344
sdmmc_csd_t (C++ 类), 1342
sdmmc_csd_t::capacity (C++ 成员), 1342
sdmmc_csd_t::card_command_class (C++ 成员), 1342
sdmmc_csd_t::csd_ver (C++ 成员), 1342
sdmmc_csd_t::mmc_ver (C++ 成员), 1342
sdmmc_csd_t::read_block_len (C++ 成员), 1342
sdmmc_csd_t::sector_size (C++ 成员), 1342
sdmmc_csd_t::tr_speed (C++ 成员), 1342
sdmmc_ext_csd_t (C++ 类), 1343
sdmmc_ext_csd_t::power_class (C++ 成员), 1343
SDMMC_FREQ_26M (C 宏), 1347
SDMMC_FREQ_52M (C 宏), 1347
SDMMC_FREQ_DEFAULT (C 宏), 1346
SDMMC_FREQ_HIGHSPEED (C 宏), 1347
SDMMC_FREQ_PROBING (C 宏), 1347
SDMMC_HOST_DEFAULT (C 宏), 999
sdmmc_host_deinit (C++ 函数), 998
sdmmc_host_do_transaction (C++ 函数), 997
SDMMC_HOST_FLAG_1BIT (C 宏), 1346
SDMMC_HOST_FLAG_4BIT (C 宏), 1346
SDMMC_HOST_FLAG_8BIT (C 宏), 1346
SDMMC_HOST_FLAG_DDR (C 宏), 1346
SDMMC_HOST_FLAG_SPI (C 宏), 1346
sdmmc_host_get_slot_width (C++ 函数), 996
sdmmc_host_init (C++ 函数), 995
sdmmc_host_init_slot (C++ 函数), 995
sdmmc_host_io_int_enable (C++ 函数), 997
sdmmc_host_io_int_wait (C++ 函数), 997
sdmmc_host_pullup_en (C++ 函数), 998
sdmmc_host_set_bus_ddr_mode (C++ 函数), 996
sdmmc_host_set_bus_width (C++ 函数), 995
sdmmc_host_set_card_clk (C++ 函数), 996
SDMMC_HOST_SLOT_0 (C 宏), 999
SDMMC_HOST_SLOT_1 (C 宏), 999
sdmmc_host_t (C++ 类), 1344
sdmmc_host_t::command_timeout_ms (C++ 成员), 1345
sdmmc_host_t::deinit (C++ 成员), 1345
sdmmc_host_t::do_transaction (C++ 成员), 1345
sdmmc_host_t::flags (C++ 成员), 1344
sdmmc_host_t::get_bus_width (C++ 成员), 1345
sdmmc_host_t::init (C++ 成员), 1344
sdmmc_host_t::io_int_enable (C++ 成员), 1345
sdmmc_host_t::io_int_wait (C++ 成员), 1345
sdmmc_host_t::io_voltage (C++ 成员), 1344
sdmmc_host_t::max_freq_khz (C++ 成员), 1344
sdmmc_host_t::set_bus_ddr_mode (C++ 成员), 1345
sdmmc_host_t::set_bus_width (C++ 成员), 1345
sdmmc_host_t::set_card_clk (C++ 成员), 1345
sdmmc_host_t::slot (C++ 成员), 1344
sdmmc_io_enable_int (C++ 函数), 1340
sdmmc_io_get_cis_data (C++ 函数), 1340
sdmmc_io_print_cis_info (C++ 函数), 1341
sdmmc_io_read_blocks (C++ 函数), 1339
sdmmc_io_read_byte (C++ 函数), 1337
sdmmc_io_read_bytes (C++ 函数), 1338

sdmmc_io_wait_int (C++ 函数), 1340
sdmmc_io_write_blocks (C++ 函数), 1339
sdmmc_io_write_byte (C++ 函数), 1337
sdmmc_io_write_bytes (C++ 函数), 1338
sdmmc_read_sectors (C++ 函数), 1337
sdmmc_response_t (C++ 类型), 1347
sdmmc_scr_t (C++ 类), 1343
sdmmc_scr_t::bus_width (C++ 成员), 1343
sdmmc_scr_t::sd_spec (C++ 成员), 1343
SDMMC_SLOT_CONFIG_DEFAULT (C 宏), 999
sdmmc_slot_config_t (C++ 类), 998
sdmmc_slot_config_t::flags (C++ 成员), 999
sdmmc_slot_config_t::gpio_cd (C++ 成员), 998
sdmmc_slot_config_t::gpio_wp (C++ 成员), 998
sdmmc_slot_config_t::width (C++ 成员), 998
SDMMC_SLOT_FLAG_INTERNAL_PULLUP (C 宏), 999
SDMMC_SLOT_NO_CD (C 宏), 999
SDMMC_SLOT_NO_WP (C 宏), 999
SDMMC_SLOT_WIDTH_DEFAULT (C 宏), 999
sdmmc_switch_func_rsp_t (C++ 类), 1343
sdmmc_switch_func_rsp_t::data (C++ 成员), 1343
sdmmc_write_sectors (C++ 函数), 1336
SDSPI_HOST_DEFAULT (C 宏), 1003
sdspi_host_deinit (C++ 函数), 1002
sdspi_host_do_transaction (C++ 函数), 1001
sdspi_host_init (C++ 函数), 1000
sdspi_host_init_slot (C++ 函数), 1001
sdspi_host_io_int_enable (C++ 函数), 1002
sdspi_host_io_int_wait (C++ 函数), 1002
sdspi_host_set_card_clk (C++ 函数), 1002
SDSPI_SLOT_CONFIG_DEFAULT (C 宏), 1003
sdspi_slot_config_t (C++ 类), 1003
sdspi_slot_config_t::dma_channel (C++ 成员), 1003
sdspi_slot_config_t::gpio_cd (C++ 成员), 1003
sdspi_slot_config_t::gpio_cs (C++ 成员), 1003
sdspi_slot_config_t::gpio_int (C++ 成员), 1003
sdspi_slot_config_t::gpio_miso (C++ 成员), 1003
sdspi_slot_config_t::gpio_mosi (C++ 成员), 1003
sdspi_slot_config_t::gpio_sck (C++ 成员), 1003
sdspi_slot_config_t::gpio_wp (C++ 成员), 1003
SDSPI_SLOT_NO_CD (C 宏), 1003
SDSPI_SLOT_NO_INT (C 宏), 1003
SDSPI_SLOT_NO_WP (C 宏), 1003
SemaphoreHandle_t (C++ 类型), 1554
semBINARY_SEMAPHORE_QUEUE_LENGTH (C 宏), 1535
semGIVE_BLOCK_TIME (C 宏), 1535
semSEMAPHORE_QUEUE_ITEM_LENGTH (C 宏), 1535
shutdown_handler_t (C++ 类型), 1685
sigmadelta_channel_t (C++ 类型), 1029
sigmadelta_config (C++ 函数), 1027
sigmadelta_config_t (C++ 类), 1029
sigmadelta_config_t::channel (C++ 成员), 1029
sigmadelta_config_t::sigmadelta_duty (C++ 成员), 1029
sigmadelta_config_t::sigmadelta_gpio (C++ 成员), 1029
sigmadelta_config_t::sigmadelta_prescale (C++ 成员), 1029
sigmadelta_port_t (C++ 类型), 1029
sigmadelta_set_duty (C++ 函数), 1027
sigmadelta_set_pin (C++ 函数), 1028
sigmadelta_set_prescale (C++ 函数), 1028
slave_transaction_cb_t (C++ 类型), 1066
smartconfig_event_got_ssid_pswd_t (C++ 类), 702
smartconfig_event_got_ssid_pswd_t::bssid (C++ 成员), 703
smartconfig_event_got_ssid_pswd_t::bssid_set (C++ 成员), 702
smartconfig_event_got_ssid_pswd_t::cellphone_ip (C++ 成员), 703
smartconfig_event_got_ssid_pswd_t::password (C++ 成员), 702
smartconfig_event_got_ssid_pswd_t::ssid (C++ 成员), 702
smartconfig_event_got_ssid_pswd_t::token (C++ 成员), 703
smartconfig_event_got_ssid_pswd_t::type

(C++ 成员), 703
smartconfig_event_t (C++ 类型), 703
SMARTCONFIG_START_CONFIG_DEFAULT (C 宏), 703
smartconfig_start_config_t (C++ 类), 703
smartconfig_start_config_t::enable_log (C++ 成员), 703
smartconfig_type_t (C++ 类型), 703
sntp_get_sync_interval (C++ 函数), 1734
sntp_get_sync_mode (C++ 函数), 1734
sntp_get_sync_status (C++ 函数), 1734
sntp_restart (C++ 函数), 1735
sntp_set_sync_interval (C++ 函数), 1734
sntp_set_sync_mode (C++ 函数), 1733
sntp_set_sync_status (C++ 函数), 1734
sntp_set_time_sync_notification_cb (C++ 函数), 1734
SNTP_SYNC_MODE_IMMED (C++ 枚举子), 1735
SNTP_SYNC_MODE_SMOOTH (C++ 枚举子), 1735
sntp_sync_mode_t (C++ 类型), 1735
SNTP_SYNC_STATUS_COMPLETED (C++ 枚举子), 1735
SNTP_SYNC_STATUS_IN_PROGRESS (C++ 枚举子), 1735
SNTP_SYNC_STATUS_RESET (C++ 枚举子), 1735
sntp_sync_status_t (C++ 类型), 1735
sntp_sync_time (C++ 函数), 1733
sntp_sync_time_cb_t (C++ 类型), 1735
SPI1_HOST (C++ 枚举子), 1042
SPI2_HOST (C++ 枚举子), 1042
SPI3_HOST (C++ 枚举子), 1042
spi_bus_add_device (C++ 函数), 1046
spi_bus_add_flash_device (C++ 函数), 1311
spi_bus_config_t (C++ 类), 1044
spi_bus_config_t::flags (C++ 成员), 1044
spi_bus_config_t::intr_flags (C++ 成员), 1044
spi_bus_config_t::max_transfer_sz (C++ 成员), 1044
spi_bus_config_t::miso_io_num (C++ 成员), 1044
spi_bus_config_t::mosi_io_num (C++ 成员), 1044
spi_bus_config_t::quadhd_io_num (C++ 成员), 1044
spi_bus_config_t::quadwp_io_num (C++ 成员), 1044
spi_bus_config_t::sclk_io_num (C++ 成员), 1044
spi_bus_free (C++ 函数), 1043
spi_bus_initialize (C++ 函数), 1043
spi_bus_remove_device (C++ 函数), 1046
spi_bus_remove_flash_device (C++ 函数), 1311
spi_cal_clock (C++ 函数), 1050
SPI_DEVICE_3WIRE (C 宏), 1054
spi_device_acquire_bus (C++ 函数), 1050
SPI_DEVICE_BIT_LSBFIRST (C 宏), 1054
SPI_DEVICE_CLK_AS_CS (C 宏), 1055
SPI_DEVICE_DDRCLK (C 宏), 1055
spi_device_get_trans_result (C++ 函数), 1047
SPI_DEVICE_HALFDUPLEX (C 宏), 1054
spi_device_handle_t (C++ 类型), 1056
spi_device_interface_config_t (C++ 类), 1051
spi_device_interface_config_t::address_bits (C++ 成员), 1052
spi_device_interface_config_t::clock_speed_hz (C++ 成员), 1052
spi_device_interface_config_t::command_bits (C++ 成员), 1052
spi_device_interface_config_t::cs_ena_posttrans (C++ 成员), 1052
spi_device_interface_config_t::cs_ena_pretrans (C++ 成员), 1052
spi_device_interface_config_t::dummy_bits (C++ 成员), 1052
spi_device_interface_config_t::duty_cycle_pos (C++ 成员), 1052
spi_device_interface_config_t::flags (C++ 成员), 1052
spi_device_interface_config_t::input_delay_ns (C++ 成员), 1052
spi_device_interface_config_t::mode (C++ 成员), 1052
spi_device_interface_config_t::post_cb (C++ 成员), 1053
spi_device_interface_config_t::pre_cb (C++ 成员), 1052

spi_device_interface_config_t::queue_size (C++ 成员), 1052
spi_device_interface_config_t::spics_io_num (C++ 成员), 1052
SPI_DEVICE_NO_DUMMY (C 宏), 1055
spi_device_polling_end (C++ 函数), 1049
spi_device_polling_start (C++ 函数), 1048
spi_device_polling_transmit (C++ 函数), 1049
SPI_DEVICE_POSITIVE_CS (C 宏), 1054
spi_device_queue_trans (C++ 函数), 1047
spi_device_release_bus (C++ 函数), 1050
SPI_DEVICE_RXBIT_LSBFIRST (C 宏), 1054
spi_device_transmit (C++ 函数), 1048
SPI_DEVICE_TXBIT_LSBFIRST (C 宏), 1054
spi_flash_chip_t (C++ 类型), 1319
SPI_FLASH_DIO (C++ 枚举子), 1323
SPI_FLASH_DOUT (C++ 枚举子), 1323
SPI_FLASH_FASTRD (C++ 枚举子), 1323
spi_flash_host_driver_t (C++ 类), 1320
spi_flash_host_driver_t (C++ 类型), 1322
spi_flash_host_driver_t::common_command (C++ 成员), 1320
spi_flash_host_driver_t::configure_host_io_mode (C++ 成员), 1321
spi_flash_host_driver_t::dev_config (C++ 成员), 1320
spi_flash_host_driver_t::driver_data (C++ 成员), 1320
spi_flash_host_driver_t::erase_block (C++ 成员), 1321
spi_flash_host_driver_t::erase_chip (C++ 成员), 1320
spi_flash_host_driver_t::erase_sector (C++ 成员), 1321
spi_flash_host_driver_t::flush_cache (C++ 成员), 1321
spi_flash_host_driver_t::host_idle (C++ 成员), 1321
spi_flash_host_driver_t::max_read_bytes (C++ 成员), 1321
spi_flash_host_driver_t::max_write_bytes (C++ 成员), 1321
spi_flash_host_driver_t::poll_cmd_done (C++ 成员), 1321
spi_flash_host_driver_t::program_page (C++ 成员), 1321
spi_flash_host_driver_t::read (C++ 成员), 1321
spi_flash_host_driver_t::read_id (C++ 成员), 1320
spi_flash_host_driver_t::read_status (C++ 成员), 1321
spi_flash_host_driver_t::set_write_protect (C++ 成员), 1321
spi_flash_host_driver_t::supports_direct_read (C++ 成员), 1321
spi_flash_host_driver_t::supports_direct_write (C++ 成员), 1321
SPI_FLASH_QIO (C++ 枚举子), 1323
SPI_FLASH_QOUT (C++ 枚举子), 1323
SPI_FLASH_READ_MODE_MAX (C++ 枚举子), 1323
SPI_FLASH_READ_MODE_MIN (C 宏), 1322
SPI_FLASH_SLOWRD (C++ 枚举子), 1322
spi_flash_trans_t (C++ 类), 1320
spi_flash_trans_t::address (C++ 成员), 1320
spi_flash_trans_t::address_bitlen (C++ 成员), 1320
spi_flash_trans_t::command (C++ 成员), 1320
spi_flash_trans_t::miso_data (C++ 成员), 1320
spi_flash_trans_t::miso_len (C++ 成员), 1320
spi_flash_trans_t::mosi_data (C++ 成员), 1320
spi_flash_trans_t::mosi_len (C++ 成员), 1320
spi_get_actual_clock (C++ 函数), 1050
spi_get_freq_limit (C++ 函数), 1051
spi_get_timing (C++ 函数), 1051
spi_host_device_t (C++ 类型), 1042
SPI_MAX_DMA_LEN (C 宏), 1044
SPI_SLAVE_BIT_LSBFIRST (C 宏), 1066
spi_slave_free (C++ 函数), 1062
spi_slave_get_trans_result (C++ 函数), 1063
spi_slave_initialize (C++ 函数), 1062
spi_slave_interface_config_t (C++ 类), 1064
spi_slave_interface_config_t::flags (C++ 成员), 1064

spi_slave_interface_config_t::mode (C++ 成员), 1065
spi_slave_interface_config_t::post_setup_cb (C++ 成员), 1065
spi_slave_interface_config_t::post_trans_cb (C++ 成员), 1065
spi_slave_interface_config_t::queue_size (C++ 成员), 1064
spi_slave_interface_config_t::spics_io_num (C++ 成员), 1064
spi_slave_queue_trans (C++ 函数), 1063
SPI_SLAVE_RXBIT_LSBFIRST (C 宏), 1065
spi_slave_transaction_t (C++ 类), 1065
spi_slave_transaction_t (C++ 类型), 1066
spi_slave_transaction_t::length (C++ 成员), 1065
spi_slave_transaction_t::rx_buffer (C++ 成员), 1065
spi_slave_transaction_t::trans_len (C++ 成员), 1065
spi_slave_transaction_t::tx_buffer (C++ 成员), 1065
spi_slave_transaction_t::user (C++ 成员), 1065
spi_slave_transmit (C++ 函数), 1064
SPI_SLAVE_TXBIT_LSBFIRST (C 宏), 1065
SPI_SWAP_DATA_RX (C 宏), 1045
SPI_SWAP_DATA_TX (C 宏), 1044
SPI_TRANS_MODE_DIO (C 宏), 1055
SPI_TRANS_MODE_DIOQIO_ADDR (C 宏), 1055
SPI_TRANS_MODE_QIO (C 宏), 1055
SPI_TRANS_SET_CD (C 宏), 1055
SPI_TRANS_USE_RXDATA (C 宏), 1055
SPI_TRANS_USE_TXDATA (C 宏), 1055
SPI_TRANS_VARIABLE_ADDR (C 宏), 1055
SPI_TRANS_VARIABLE_CMD (C 宏), 1055
SPI_TRANS_VARIABLE_DUMMY (C 宏), 1055
spi_transaction_ext_t (C++ 类), 1054
spi_transaction_ext_t::address_bits (C++ 成员), 1054
spi_transaction_ext_t::base (C++ 成员), 1054
spi_transaction_ext_t::command_bits (C++ 成员), 1054
spi_transaction_ext_t::dummy_bits (C++ 成员), 1054
spi_transaction_t (C++ 类), 1053
spi_transaction_t (C++ 类型), 1056
spi_transaction_t::addr (C++ 成员), 1053
spi_transaction_t::cmd (C++ 成员), 1053
spi_transaction_t::flags (C++ 成员), 1053
spi_transaction_t::length (C++ 成员), 1053
spi_transaction_t::rx_buffer (C++ 成员), 1054
spi_transaction_t::rx_data (C++ 成员), 1054
spi_transaction_t::rxlength (C++ 成员), 1053
spi_transaction_t::tx_buffer (C++ 成员), 1053
spi_transaction_t::tx_data (C++ 成员), 1054
spi_transaction_t::user (C++ 成员), 1053
SPICOMMON_BUSFLAG_DUAL (C 宏), 1045
SPICOMMON_BUSFLAG_IOMUX_PINS (C 宏), 1045
SPICOMMON_BUSFLAG_MASTER (C 宏), 1045
SPICOMMON_BUSFLAG_MISO (C 宏), 1045
SPICOMMON_BUSFLAG_MOSI (C 宏), 1045
SPICOMMON_BUSFLAG_NATIVE_PINS (C 宏), 1046
SPICOMMON_BUSFLAG_QUAD (C 宏), 1046
SPICOMMON_BUSFLAG_SCLK (C 宏), 1045
SPICOMMON_BUSFLAG_SLAVE (C 宏), 1045
SPICOMMON_BUSFLAG_WPHD (C 宏), 1046
StaticRingbuffer_t (C++ 类型), 1612
system_event_ap_probe_req_rx_t (C++ 类型), 1469
SYSTEM_EVENT_AP_PROBEREQRECVED (C++ 枚举子), 1471
SYSTEM_EVENT_AP_STA_GOT_IP6 (C 宏), 1468
SYSTEM_EVENT_AP_STACONNECTED (C++ 枚举子), 1471
system_event_ap_staconnected_t (C++ 类型), 1469
SYSTEM_EVENT_AP_STADISCONNECTED (C++ 枚举子), 1471
system_event_ap_stadisconnected_t (C++ 类型), 1469
SYSTEM_EVENT_AP_STAIPASSIGNED (C++ 枚举子), 1471
system_event_ap_staipassigned_t (C++ 类型),

1469
SYSTEM_EVENT_AP_START (C++ 枚举子), 1470
SYSTEM_EVENT_AP_STOP (C++ 枚举子), 1470
system_event_cb_t (C++ 类型), 1469
SYSTEM_EVENT_ETH_CONNECTED (C++ 枚举子), 1471
SYSTEM_EVENT_ETH_DISCONNECTED (C++ 枚举子), 1471
SYSTEM_EVENT_ETH_GOT_IP (C++ 枚举子), 1471
SYSTEM_EVENT_ETH_START (C++ 枚举子), 1471
SYSTEM_EVENT_ETH_STOP (C++ 枚举子), 1471
SYSTEM_EVENT_GOT_IP6 (C++ 枚举子), 1471
system_event_got_ip6_t (C++ 类型), 1469
system_event_handler_t (C++ 类型), 1469
system_event_id_t (C++ 类型), 1470
system_event_info_t (C++ 类型), 1467
system_event_info_t::ap_probereqrecved (C++ 成员), 1468
system_event_info_t::ap_staipassigned (C++ 成员), 1468
system_event_info_t::auth_change (C++ 成员), 1467
system_event_info_t::connected (C++ 成员), 1467
system_event_info_t::disconnected (C++ 成员), 1467
system_event_info_t::got_ip (C++ 成员), 1467
system_event_info_t::got_ip6 (C++ 成员), 1468
system_event_info_t::scan_done (C++ 成员), 1467
system_event_info_t::sta_connected (C++ 成员), 1468
system_event_info_t::sta_disconnected (C++ 成员), 1468
system_event_info_t::sta_er_fail_reason (C++ 成员), 1468
system_event_info_t::sta_er_pin (C++ 成员), 1467
SYSTEM_EVENT_MAX (C++ 枚举子), 1471
SYSTEM_EVENT_SCAN_DONE (C++ 枚举子), 1470
SYSTEM_EVENT_STA_AUTHMODE_CHANGE (C++ 枚举子), 1470
system_event_sta_authmode_change_t (C++ 类型), 1469
SYSTEM_EVENT_STA_CONNECTED (C++ 枚举子), 1470
system_event_sta_connected_t (C++ 类型), 1468
SYSTEM_EVENT_STA_DISCONNECTED (C++ 枚举子), 1470
system_event_sta_disconnected_t (C++ 类型), 1469
SYSTEM_EVENT_STA_GOT_IP (C++ 枚举子), 1470
system_event_sta_got_ip_t (C++ 类型), 1469
SYSTEM_EVENT_STA_LOST_IP (C++ 枚举子), 1470
system_event_sta_scan_done_t (C++ 类型), 1468
SYSTEM_EVENT_STA_START (C++ 枚举子), 1470
SYSTEM_EVENT_STA_STOP (C++ 枚举子), 1470
SYSTEM_EVENT_STA_WPS_ER_FAILED (C++ 枚举子), 1470
SYSTEM_EVENT_STA_WPS_ER_PBC_OVERLAP (C++ 枚举子), 1470
SYSTEM_EVENT_STA_WPS_ER_PIN (C++ 枚举子), 1470
system_event_sta_wps_er_pin_t (C++ 类型), 1469
SYSTEM_EVENT_STA_WPS_ER_SUCCESS (C++ 枚举子), 1470
SYSTEM_EVENT_STA_WPS_ER_TIMEOUT (C++ 枚举子), 1470
system_event_sta_wps_fail_reason_t (C++ 类型), 1468
system_event_t (C++ 类), 1468
system_event_t::event_id (C++ 成员), 1468
system_event_t::event_info (C++ 成员), 1468
SYSTEM_EVENT_WIFI_READY (C++ 枚举子), 1470

T

taskDISABLE_INTERRUPTS (C 宏), 1502
taskENABLE_INTERRUPTS (C 宏), 1502
taskENTER_CRITICAL (C 宏), 1502
taskENTER_CRITICAL_ISR (C 宏), 1502
taskEXIT_CRITICAL (C 宏), 1502
taskEXIT_CRITICAL_ISR (C 宏), 1502
TaskHandle_t (C++ 类型), 1503
TaskHookFunction_t (C++ 类型), 1503
taskSCHEDULER_NOT_STARTED (C 宏), 1502

taskSCHEDULER_RUNNING (C 宏), 1502
taskSCHEDULER_SUSPENDED (C 宏), 1502
TaskSnapshot_t (C++ 类型), 1504
TaskStatus_t (C++ 类型), 1503
taskYIELD (C 宏), 1502
temp_sensor_config_t (C++ 类), 1068
temp_sensor_config_t::clk_div (C++ 成员), 1068
temp_sensor_config_t::dac_offset (C++ 成员), 1068
temp_sensor_dac_offset_t (C++ 类型), 1068
temp_sensor_get_config (C++ 函数), 1067
temp_sensor_read_celsius (C++ 函数), 1067
temp_sensor_read_raw (C++ 函数), 1067
temp_sensor_set_config (C++ 函数), 1066
temp_sensor_start (C++ 函数), 1067
temp_sensor_stop (C++ 函数), 1067
TIMER_0 (C++ 枚举子), 1083
TIMER_1 (C++ 枚举子), 1083
TIMER_ALARM_DIS (C++ 枚举子), 1084
TIMER_ALARM_EN (C++ 枚举子), 1084
TIMER_ALARM_MAX (C++ 枚举子), 1085
timer_alarm_t (C++ 类型), 1084
TIMER_AUTORELOAD_DIS (C++ 枚举子), 1085
TIMER_AUTORELOAD_EN (C++ 枚举子), 1085
TIMER_AUTORELOAD_MAX (C++ 枚举子), 1085
timer_autoreload_t (C++ 类型), 1085
TIMER_BASE_CLK (C 宏), 1082
timer_config_t (C++ 类), 1082
timer_config_t::alarm_en (C++ 成员), 1082
timer_config_t::auto_reload (C++ 成员), 1083
timer_config_t::counter_dir (C++ 成员), 1083
timer_config_t::counter_en (C++ 成员), 1082
timer_config_t::divider (C++ 成员), 1083
timer_config_t::intr_type (C++ 成员), 1082
timer_count_dir_t (C++ 类型), 1083
TIMER_COUNT_DOWN (C++ 枚举子), 1083
TIMER_COUNT_MAX (C++ 枚举子), 1083
TIMER_COUNT_UP (C++ 枚举子), 1083
timer_deinit (C++ 函数), 1077
timer_disable_intr (C++ 函数), 1078
timer_enable_intr (C++ 函数), 1078
timer_get_alarm_value (C++ 函数), 1074
timer_get_config (C++ 函数), 1077
timer_get_counter_time_sec (C++ 函数), 1072
timer_get_counter_value (C++ 函数), 1071
TIMER_GROUP_0 (C++ 枚举子), 1083
TIMER_GROUP_1 (C++ 枚举子), 1083
timer_group_clr_intr_sta_in_isr (C++ 函数), 1081
timer_group_clr_intr_status_in_isr (C++ 函数), 1079
timer_group_enable_alarm_in_isr (C++ 函数), 1079
timer_group_get_auto_reload_in_isr (C++ 函数), 1081
timer_group_get_counter_value_in_isr (C++ 函数), 1079
timer_group_get_intr_status_in_isr (C++ 函数), 1080
timer_group_intr_clr_in_isr (C++ 函数), 1079
timer_group_intr_disable (C++ 函数), 1078
timer_group_intr_enable (C++ 函数), 1077
timer_group_intr_get_in_isr (C++ 函数), 1080
TIMER_GROUP_MAX (C++ 枚举子), 1083
timer_group_set_alarm_value_in_isr (C++ 函数), 1080
timer_group_set_counter_enable_in_isr (C++ 函数), 1080
timer_group_t (C++ 类型), 1083
timer_idx_t (C++ 类型), 1083
timer_init (C++ 函数), 1076
TIMER_INTR_LEVEL (C++ 枚举子), 1085
TIMER_INTR_MAX (C++ 枚举子), 1085
timer_intr_mode_t (C++ 类型), 1085
TIMER_INTR_NONE (C++ 枚举子), 1084
timer_intr_t (C++ 类型), 1084
TIMER_INTR_T0 (C++ 枚举子), 1084
TIMER_INTR_T1 (C++ 枚举子), 1084
TIMER_INTR_WDT (C++ 枚举子), 1084
timer_isr_callback_add (C++ 函数), 1075
timer_isr_callback_remove (C++ 函数), 1075
timer_isr_handle_t (C++ 类型), 1082
timer_isr_register (C++ 函数), 1076

timer_isr_t (C++ 类型), 1082
TIMER_MAX (C++ 枚举子), 1083
timer_pause (C++ 函数), 1073
TIMER_PAUSE (C++ 枚举子), 1084
timer_set_alarm (C++ 函数), 1074
timer_set_alarm_value (C++ 函数), 1074
timer_set_auto_reload (C++ 函数), 1073
timer_set_counter_mode (C++ 函数), 1073
timer_set_counter_value (C++ 函数), 1072
timer_set_divider (C++ 函数), 1073
timer_spinlock_give (C++ 函数), 1081
timer_spinlock_take (C++ 函数), 1081
timer_start (C++ 函数), 1072
TIMER_START (C++ 枚举子), 1084
timer_start_t (C++ 类型), 1083
timer_wdt_behavior_t (C++ 类型), 1084
TIMER_WDT_INT (C++ 枚举子), 1084
TIMER_WDT_OFF (C++ 枚举子), 1084
TIMER_WDT_RESET_CPU (C++ 枚举子), 1084
TIMER_WDT_RESET_SYSTEM (C++ 枚举子), 1084
TimerCallbackFunction_t (C++ 类型), 1578
TimerHandle_t (C++ 类型), 1578
TlsDeleteCallbackFunction_t (C++ 类型), 1504
tmrCOMMAND_CHANGE_PERIOD (C 宏), 1565
tmrCOMMAND_CHANGE_PERIOD_FROM_ISR (C 宏), 1565
tmrCOMMAND_DELETE (C 宏), 1565
tmrCOMMAND_EXECUTE_CALLBACK (C 宏), 1564
tmrCOMMAND_EXECUTE_CALLBACK_FROM_ISR (C 宏), 1564
tmrCOMMAND_RESET (C 宏), 1564
tmrCOMMAND_RESET_FROM_ISR (C 宏), 1565
tmrCOMMAND_START (C 宏), 1564
tmrCOMMAND_START_DONT_TRACE (C 宏), 1564
tmrCOMMAND_START_FROM_ISR (C 宏), 1565
tmrCOMMAND_STOP (C 宏), 1565
tmrCOMMAND_STOP_FROM_ISR (C 宏), 1565
tmrFIRST_FROM_ISR_COMMAND (C 宏), 1565
TOUCH_PAD_GPIO0_CHANNEL (C 宏), 1089
TOUCH_PAD_GPIO12_CHANNEL (C 宏), 1090
TOUCH_PAD_GPIO13_CHANNEL (C 宏), 1090
TOUCH_PAD_GPIO14_CHANNEL (C 宏), 1090
TOUCH_PAD_GPIO15_CHANNEL (C 宏), 1090
TOUCH_PAD_GPIO27_CHANNEL (C 宏), 1090
TOUCH_PAD_GPIO2_CHANNEL (C 宏), 1089
TOUCH_PAD_GPIO32_CHANNEL (C 宏), 1090
TOUCH_PAD_GPIO33_CHANNEL (C 宏), 1090
TOUCH_PAD_GPIO4_CHANNEL (C 宏), 1089
TOUCH_PAD_NUM0_GPIO_NUM (C 宏), 1089
TOUCH_PAD_NUM1_GPIO_NUM (C 宏), 1089
TOUCH_PAD_NUM2_GPIO_NUM (C 宏), 1089
TOUCH_PAD_NUM3_GPIO_NUM (C 宏), 1090
TOUCH_PAD_NUM4_GPIO_NUM (C 宏), 1090
TOUCH_PAD_NUM5_GPIO_NUM (C 宏), 1090
TOUCH_PAD_NUM6_GPIO_NUM (C 宏), 1090
TOUCH_PAD_NUM7_GPIO_NUM (C 宏), 1090
TOUCH_PAD_NUM8_GPIO_NUM (C 宏), 1090
TOUCH_PAD_NUM9_GPIO_NUM (C 宏), 1090
transaction_cb_t (C++ 类型), 1056
TSENS_CONFIG_DEFAULT (C 宏), 1068
TSENS_DAC_DEFAULT (C++ 枚举子), 1069
TSENS_DAC_L0 (C++ 枚举子), 1068
TSENS_DAC_L1 (C++ 枚举子), 1068
TSENS_DAC_L2 (C++ 枚举子), 1068
TSENS_DAC_L3 (C++ 枚举子), 1068
TSENS_DAC_L4 (C++ 枚举子), 1068
TSENS_DAC_MAX (C++ 枚举子), 1069
tskIDLE_PRIORITY (C 宏), 1502
tskKERNEL_VERSION_BUILD (C 宏), 1501
tskKERNEL_VERSION_MAJOR (C 宏), 1501
tskKERNEL_VERSION_MINOR (C 宏), 1501
tskKERNEL_VERSION_NUMBER (C 宏), 1501
tskNO_AFFINITY (C 宏), 1502

U

uart_at_cmd_t (C++ 类), 1119
uart_at_cmd_t::char_num (C++ 成员), 1119
uart_at_cmd_t::cmd_char (C++ 成员), 1119
uart_at_cmd_t::gap_tout (C++ 成员), 1119
uart_at_cmd_t::post_idle (C++ 成员), 1119
uart_at_cmd_t::pre_idle (C++ 成员), 1119
UART_BREAK (C++ 枚举子), 1118
UART_BUFFER_FULL (C++ 枚举子), 1118
uart_clear_intr_status (C++ 函数), 1104
uart_config_t (C++ 类), 1119

uart_config_t::baud_rate (C++ 成员), 1120
uart_config_t::data_bits (C++ 成员), 1120
uart_config_t::flow_ctrl (C++ 成员), 1120
uart_config_t::parity (C++ 成员), 1120
uart_config_t::rx_flow_ctrl_thresh (C++ 成员), 1120
uart_config_t::source_clk (C++ 成员), 1120
uart_config_t::stop_bits (C++ 成员), 1120
UART_CTS_GPIO19_DIRECT_CHANNEL (C 宏), 1124
UART_CTS_GPIO6_DIRECT_CHANNEL (C 宏), 1124
UART_CTS_GPIO8_DIRECT_CHANNEL (C 宏), 1124
UART_DATA (C++ 枚举子), 1118
UART_DATA_5_BITS (C++ 枚举子), 1121
UART_DATA_6_BITS (C++ 枚举子), 1121
UART_DATA_7_BITS (C++ 枚举子), 1121
UART_DATA_8_BITS (C++ 枚举子), 1121
UART_DATA_BITS_MAX (C++ 枚举子), 1121
UART_DATA_BREAK (C++ 枚举子), 1118
uart_disable_intr_mask (C++ 函数), 1105
uart_disable_pattern_det_intr (C++ 函数), 1111
uart_disable_rx_intr (C++ 函数), 1105
uart_disable_tx_intr (C++ 函数), 1105
uart_driver_delete (C++ 函数), 1100
uart_driver_install (C++ 函数), 1099
uart_enable_intr_mask (C++ 函数), 1104
uart_enable_pattern_det_baud_intr (C++ 函数), 1112
uart_enable_rx_intr (C++ 函数), 1105
uart_enable_tx_intr (C++ 函数), 1106
UART_EVENT_MAX (C++ 枚举子), 1118
uart_event_t (C++ 类), 1117
uart_event_t::size (C++ 成员), 1117
uart_event_t::type (C++ 成员), 1117
uart_event_type_t (C++ 类型), 1118
UART_FIFO_OVF (C++ 枚举子), 1118
uart_flush (C++ 函数), 1110
uart_flush_input (C++ 函数), 1111
UART_FRAME_ERR (C++ 枚举子), 1118
uart_get_baudrate (C++ 函数), 1102
uart_get_buffered_data_len (C++ 函数), 1111
uart_get_collision_flag (C++ 函数), 1115
uart_get_hw_flow_ctrl (C++ 函数), 1104
uart_get_parity (C++ 函数), 1102
uart_get_stop_bits (C++ 函数), 1101
uart_get_wakeup_threshold (C++ 函数), 1116
uart_get_word_length (C++ 函数), 1101
UART_GPIO10_DIRECT_CHANNEL (C 宏), 1124
UART_GPIO11_DIRECT_CHANNEL (C 宏), 1124
UART_GPIO16_DIRECT_CHANNEL (C 宏), 1124
UART_GPIO17_DIRECT_CHANNEL (C 宏), 1124
UART_GPIO19_DIRECT_CHANNEL (C 宏), 1123
UART_GPIO1_DIRECT_CHANNEL (C 宏), 1123
UART_GPIO22_DIRECT_CHANNEL (C 宏), 1124
UART_GPIO3_DIRECT_CHANNEL (C 宏), 1123
UART_GPIO6_DIRECT_CHANNEL (C 宏), 1124
UART_GPIO7_DIRECT_CHANNEL (C 宏), 1124
UART_GPIO8_DIRECT_CHANNEL (C 宏), 1124
UART_GPIO9_DIRECT_CHANNEL (C 宏), 1124
uart_hw_flowcontrol_t (C++ 类型), 1122
UART_HW_FLOWCTRL_CTS (C++ 枚举子), 1122
UART_HW_FLOWCTRL_CTS_RTS (C++ 枚举子), 1122
UART_HW_FLOWCTRL_DISABLE (C++ 枚举子), 1122
UART_HW_FLOWCTRL_MAX (C++ 枚举子), 1122
UART_HW_FLOWCTRL_RTS (C++ 枚举子), 1122
uart_intr_config (C++ 函数), 1108
uart_intr_config_t (C++ 类), 1117
uart_intr_config_t::intr_enable_mask (C++ 成员), 1117
uart_intr_config_t::rx_timeout_thresh (C++ 成员), 1117
uart_intr_config_t::rxfifo_full_thresh (C++ 成员), 1117
uart_intr_config_t::txfifo_empty_intr_thresh (C++ 成员), 1117
uart_is_driver_installed (C++ 函数), 1100
uart_isr_free (C++ 函数), 1106
uart_isr_handle_t (C++ 类型), 1118
uart_isr_register (C++ 函数), 1106
UART_MODE_IRDA (C++ 枚举子), 1120
UART_MODE_RS485_APP_CTRL (C++ 枚举子), 1121
UART_MODE_RS485_COLLISION_DETECT (C++ 枚举子), 1120
UART_MODE_RS485_HALF_DUPLEX (C++ 枚举子),

1120
uart_mode_t (C++ 类型), 1120
UART_MODE_UART (C++ 枚举子), 1120
UART_NUM_0 (C 宏), 1117
UART_NUM_0_CTS_DIRECT_GPIO_NUM (C 宏), 1124
UART_NUM_0_RTS_DIRECT_GPIO_NUM (C 宏), 1124
UART_NUM_0_RXD_DIRECT_GPIO_NUM (C 宏), 1123
UART_NUM_0_TXD_DIRECT_GPIO_NUM (C 宏), 1123
UART_NUM_1 (C 宏), 1117
UART_NUM_1_CTS_DIRECT_GPIO_NUM (C 宏), 1124
UART_NUM_1_RTS_DIRECT_GPIO_NUM (C 宏), 1124
UART_NUM_1_RXD_DIRECT_GPIO_NUM (C 宏), 1124
UART_NUM_1_TXD_DIRECT_GPIO_NUM (C 宏), 1124
UART_NUM_2_CTS_DIRECT_GPIO_NUM (C 宏), 1124
UART_NUM_2_RTS_DIRECT_GPIO_NUM (C 宏), 1124
UART_NUM_2_RXD_DIRECT_GPIO_NUM (C 宏), 1124
UART_NUM_2_TXD_DIRECT_GPIO_NUM (C 宏), 1124
UART_NUM_MAX (C 宏), 1118
uart_param_config (C++ 函数), 1108
UART_PARITY_DISABLE (C++ 枚举子), 1121
UART_PARITY_ERR (C++ 枚举子), 1118
UART_PARITY_EVEN (C++ 枚举子), 1121
UART_PARITY_ODD (C++ 枚举子), 1121
uart_parity_t (C++ 类型), 1121
UART_PATTERN_DET (C++ 枚举子), 1118
uart_pattern_get_pos (C++ 函数), 1113
uart_pattern_pop_pos (C++ 函数), 1112
uart_pattern_queue_reset (C++ 函数), 1113
UART_PIN_NO_CHANGE (C 宏), 1118
uart_port_t (C++ 类型), 1120
uart_read_bytes (C++ 函数), 1110
UART_RTS_GPIO11_DIRECT_CHANNEL (C 宏), 1124
UART_RTS_GPIO22_DIRECT_CHANNEL (C 宏), 1124
UART_RTS_GPIO7_DIRECT_CHANNEL (C 宏), 1125
UART_RXD_GPIO16_DIRECT_CHANNEL (C 宏), 1124
UART_RXD_GPIO3_DIRECT_CHANNEL (C 宏), 1124
UART_RXD_GPIO9_DIRECT_CHANNEL (C 宏), 1124
UART_SCLK_APB (C++ 枚举子), 1123
UART_SCLK_REF_TICK (C++ 枚举子), 1123
uart_sclk_t (C++ 类型), 1122
uart_set_baudrate (C++ 函数), 1102
uart_set_dtr (C++ 函数), 1107
uart_set_hw_flow_ctrl (C++ 函数), 1103
uart_set_line_inverse (C++ 函数), 1103
uart_set_loop_back (C++ 函数), 1116
uart_set_mode (C++ 函数), 1113
uart_set_parity (C++ 函数), 1102
uart_set_pin (C++ 函数), 1107
uart_set_rts (C++ 函数), 1107
uart_set_rx_full_threshold (C++ 函数), 1114
uart_set_rx_timeout (C++ 函数), 1114
uart_set_stop_bits (C++ 函数), 1101
uart_set_sw_flow_ctrl (C++ 函数), 1103
uart_set_tx_empty_threshold (C++ 函数), 1114
uart_set_tx_idle_num (C++ 函数), 1108
uart_set_wakeup_threshold (C++ 函数), 1115
uart_set_word_length (C++ 函数), 1100
UART_SIGNAL_CTS_INV (C++ 枚举子), 1122
UART_SIGNAL_DSR_INV (C++ 枚举子), 1122
UART_SIGNAL_DTR_INV (C++ 枚举子), 1122
uart_signal_inv_t (C++ 类型), 1122
UART_SIGNAL_IRDA_RX_INV (C++ 枚举子), 1122
UART_SIGNAL_IRDA_TX_INV (C++ 枚举子), 1122
UART_SIGNAL_RTS_INV (C++ 枚举子), 1122
UART_SIGNAL_RXD_INV (C++ 枚举子), 1122
UART_SIGNAL_TXD_INV (C++ 枚举子), 1122
UART_STOP_BITS_1 (C++ 枚举子), 1121
UART_STOP_BITS_1_5 (C++ 枚举子), 1121
UART_STOP_BITS_2 (C++ 枚举子), 1121
UART_STOP_BITS_MAX (C++ 枚举子), 1121
uart_stop_bits_t (C++ 类型), 1121
uart_sw_flowctrl_t (C++ 类), 1119
uart_sw_flowctrl_t::xoff_char (C++ 成员), 1119
uart_sw_flowctrl_t::xoff_thrd (C++ 成员), 1119
uart_sw_flowctrl_t::xon_char (C++ 成员), 1119
uart_sw_flowctrl_t::xon_thrd (C++ 成员), 1119
uart_tx_chars (C++ 函数), 1109
UART_TXD_GPIO10_DIRECT_CHANNEL (C 宏), 1124
UART_TXD_GPIO17_DIRECT_CHANNEL (C 宏), 1124
UART_TXD_GPIO1_DIRECT_CHANNEL (C 宏), 1124
uart_wait_tx_done (C++ 函数), 1109
uart_wait_tx_idle_polling (C++ 函数), 1116

uart_word_length_t (C++ 类型), 1121
uart_write_bytes (C++ 函数), 1109
uart_write_bytes_with_break (C++ 函数), 1110
UINT16 (C++ 类型), 430
UINT32 (C++ 类型), 430
UINT64 (C++ 类型), 430
UINT8 (C++ 类型), 430
ulp_load_binary (C++ 函数), 2238, 2243
ulp_process_macros_and_load (C++ 函数), 2230
ulp_run (C++ 函数), 2231, 2239, 2244
ulp_set_wakeup_period (C++ 函数), 2239, 2245
ulTaskNotifyTake (C++ 函数), 1499
uxQueueMessagesWaiting (C++ 函数), 1511
uxQueueMessagesWaitingFromISR (C++ 函数), 1507
uxQueueSpacesAvailable (C++ 函数), 1511
uxSemaphoreGetCount (C 宏), 1554
uxTaskGetNumberOfTasks (C++ 函数), 1487
uxTaskGetStackHighWaterMark (C++ 函数), 1487
uxTaskGetSystemState (C++ 函数), 1490
uxTaskPriorityGet (C++ 函数), 1480
uxTaskPriorityGetFromISR (C++ 函数), 1481

V

vendor_ie_data_t (C++ 类), 683
vendor_ie_data_t::element_id (C++ 成员), 683
vendor_ie_data_t::length (C++ 成员), 683
vendor_ie_data_t::payload (C++ 成员), 683
vendor_ie_data_t::vendor_oui (C++ 成员), 683
vendor_ie_data_t::vendor_oui_type (C++ 成员), 683
vEventGroupDelete (C++ 函数), 1587
vprintf_like_t (C++ 类型), 1677
vQueueAddToRegistry (C++ 函数), 1513
vQueueDelete (C++ 函数), 1511
vQueueUnregisterQueue (C++ 函数), 1513
vRingbufferDelete (C++ 函数), 1609
vRingbufferGetInfo (C++ 函数), 1611
vRingbufferReturnItem (C++ 函数), 1608
vRingbufferReturnItemFromISR (C++ 函数), 1609
vSemaphoreDelete (C 宏), 1553
vTaskDelay (C++ 函数), 1478
vTaskDelayUntil (C++ 函数), 1479
vTaskDelete (C++ 函数), 1477
vTaskGetRunTimeStats (C++ 函数), 1493
vTaskList (C++ 函数), 1493
vTaskNotifyGiveFromISR (C++ 函数), 1498
vTaskPrioritySet (C++ 函数), 1481
vTaskResume (C++ 函数), 1483
vTaskSetApplicationTaskTag (C++ 函数), 1488
vTaskSetThreadLocalStoragePointer (C++ 函数), 1488
vTaskSetThreadLocalStoragePointerAndDelCallback (C++ 函数), 1489
vTaskSuspend (C++ 函数), 1482
vTaskSuspendAll (C++ 函数), 1485
vTimerSetTimerID (C++ 函数), 1560

W

WEBSOCKET_EVENT_ANY (C++ 枚举子), 1175
WEBSOCKET_EVENT_CONNECTED (C++ 枚举子), 1176
WEBSOCKET_EVENT_DATA (C++ 枚举子), 1176
WEBSOCKET_EVENT_DISCONNECTED (C++ 枚举子), 1176
WEBSOCKET_EVENT_ERROR (C++ 枚举子), 1176
WEBSOCKET_EVENT_MAX (C++ 枚举子), 1176
WEBSOCKET_TRANSPORT_OVER_SSL (C++ 枚举子), 1176
WEBSOCKET_TRANSPORT_OVER_TCP (C++ 枚举子), 1176
WEBSOCKET_TRANSPORT_UNKNOWN (C++ 枚举子), 1176
wifi_active_scan_time_t (C++ 类), 678
wifi_active_scan_time_t::max (C++ 成员), 678
wifi_active_scan_time_t::min (C++ 成员), 678
WIFI_ALL_CHANNEL_SCAN (C++ 枚举子), 695
WIFI_AMPDU_RX_ENABLED (C 宏), 676
WIFI_AMPDU_TX_ENABLED (C 宏), 676
WIFI_ANT_ANT0 (C++ 枚举子), 694
WIFI_ANT_ANT1 (C++ 枚举子), 695
wifi_ant_config_t (C++ 类), 687
wifi_ant_config_t::enabled_ant0 (C++ 成员), 687

wifi_ant_config_t::enabled_ant1 (C++ 成员), 687
wifi_ant_config_t::rx_ant_default (C++ 成员), 687
wifi_ant_config_t::rx_ant_mode (C++ 成员), 687
wifi_ant_config_t::tx_ant_mode (C++ 成员), 687
wifi_ant_gpio_config_t (C++ 类), 687
wifi_ant_gpio_config_t::gpio_cfg (C++ 成员), 687
wifi_ant_gpio_t (C++ 类), 687
wifi_ant_gpio_t::gpio_num (C++ 成员), 687
wifi_ant_gpio_t::gpio_select (C++ 成员), 687
WIFI_ANT_MAX (C++ 枚举子), 695
WIFI_ANT_MODE_ANT0 (C++ 枚举子), 697
WIFI_ANT_MODE_ANT1 (C++ 枚举子), 697
WIFI_ANT_MODE_AUTO (C++ 枚举子), 697
WIFI_ANT_MODE_MAX (C++ 枚举子), 697
wifi_ant_mode_t (C++ 类型), 696
wifi_ant_t (C++ 类型), 694
wifi_ap_config_t (C++ 类), 681
wifi_ap_config_t::authmode (C++ 成员), 681
wifi_ap_config_t::beacon_interval (C++ 成员), 681
wifi_ap_config_t::channel (C++ 成员), 681
wifi_ap_config_t::max_connection (C++ 成员), 681
wifi_ap_config_t::password (C++ 成员), 681
wifi_ap_config_t::ssid (C++ 成员), 681
wifi_ap_config_t::ssid_hidden (C++ 成员), 681
wifi_ap_config_t::ssid_len (C++ 成员), 681
wifi_ap_record_t (C++ 类), 679
wifi_ap_record_t::ant (C++ 成员), 680
wifi_ap_record_t::authmode (C++ 成员), 680
wifi_ap_record_t::bssid (C++ 成员), 679
wifi_ap_record_t::country (C++ 成员), 680
wifi_ap_record_t::group_cipher (C++ 成员), 680
wifi_ap_record_t::pairwise_cipher (C++ 成员), 680
wifi_ap_record_t::phy_11b (C++ 成员), 680
wifi_ap_record_t::phy_11g (C++ 成员), 680
wifi_ap_record_t::phy_11n (C++ 成员), 680
wifi_ap_record_t::phy_lr (C++ 成员), 680
wifi_ap_record_t::primary (C++ 成员), 679
wifi_ap_record_t::reserved (C++ 成员), 680
wifi_ap_record_t::rssi (C++ 成员), 679
wifi_ap_record_t::second (C++ 成员), 679
wifi_ap_record_t::ssid (C++ 成员), 679
wifi_ap_record_t::wps (C++ 成员), 680
WIFI_AUTH_MAX (C++ 枚举子), 692
wifi_auth_mode_t (C++ 类型), 692
WIFI_AUTH_OPEN (C++ 枚举子), 692
WIFI_AUTH_WEP (C++ 枚举子), 692
WIFI_AUTH_WPA2_ENTERPRISE (C++ 枚举子), 692
WIFI_AUTH_WPA2_PSK (C++ 枚举子), 692
WIFI_AUTH_WPA2_WPA3_PSK (C++ 枚举子), 692
WIFI_AUTH_WPA3_PSK (C++ 枚举子), 692
WIFI_AUTH_WPA_PSK (C++ 枚举子), 692
WIFI_AUTH_WPA_WPA2_PSK (C++ 枚举子), 692
wifi_bandwidth_t (C++ 类型), 695
WIFI_BW_HT20 (C++ 枚举子), 695
WIFI_BW_HT40 (C++ 枚举子), 695
WIFI_CIPHER_TYPE_AES_CMAC128 (C++ 枚举子), 694
WIFI_CIPHER_TYPE_CCMP (C++ 枚举子), 694
WIFI_CIPHER_TYPE_NONE (C++ 枚举子), 694
wifi_cipher_type_t (C++ 类型), 694
WIFI_CIPHER_TYPE_TKIP (C++ 枚举子), 694
WIFI_CIPHER_TYPE_TKIP_CCMP (C++ 枚举子), 694
WIFI_CIPHER_TYPE_UNKNOWN (C++ 枚举子), 694
WIFI_CIPHER_TYPE_WEP104 (C++ 枚举子), 694
WIFI_CIPHER_TYPE_WEP40 (C++ 枚举子), 694
wifi_config_t (C++ 类型), 677
wifi_config_t::ap (C++ 成员), 678
wifi_config_t::sta (C++ 成员), 678
WIFI_CONNECT_AP_BY_SECURITY (C++ 枚举子), 695
WIFI_CONNECT_AP_BY_SIGNAL (C++ 枚举子), 695
WIFI_COUNTRY_POLICY_AUTO (C++ 枚举子), 692
WIFI_COUNTRY_POLICY_MANUAL (C++ 枚举子), 692
wifi_country_policy_t (C++ 类型), 692
wifi_country_t (C++ 类), 678
wifi_country_t::cc (C++ 成员), 678

wifi_country_t::max_tx_power (C++ 成员), 678
wifi_country_t::nchan (C++ 成员), 678
wifi_country_t::policy (C++ 成员), 678
wifi_country_t::schan (C++ 成员), 678
wifi_csi_cb_t (C++ 类型), 677
wifi_csi_config_t (C++ 类), 686
wifi_csi_config_t::channel_filter_en (C++ 成员), 686
wifi_csi_config_t::htltf_en (C++ 成员), 686
wifi_csi_config_t::lltf_en (C++ 成员), 686
wifi_csi_config_t::ltf_merge_en (C++ 成员), 686
wifi_csi_config_t::manu_scale (C++ 成员), 686
wifi_csi_config_t::shift (C++ 成员), 686
wifi_csi_config_t::stbc_htltf2_en (C++ 成员), 686
WIFI_CSI_ENABLED (C 宏), 676
wifi_csi_info_t (C++ 类), 686
wifi_csi_info_t::buf (C++ 成员), 686
wifi_csi_info_t::first_word_invalid (C++ 成员), 686
wifi_csi_info_t::len (C++ 成员), 687
wifi_csi_info_t::mac (C++ 成员), 686
wifi_csi_info_t::rx_ctrl (C++ 成员), 686
WIFI_DEFAULT_RX_BA_WIN (C 宏), 676
WIFI_DEFAULT_TX_BA_WIN (C 宏), 676
WIFI_DYNAMIC_TX_BUFFER_NUM (C 宏), 676
wifi_err_reason_t (C++ 类型), 692
wifi_event_ap_probe_req_rx_t (C++ 类), 689
wifi_event_ap_probe_req_rx_t::mac (C++ 成员), 690
wifi_event_ap_probe_req_rx_t::rssi (C++ 成员), 690
WIFI_EVENT_AP_PROBEREQRECVED (C++ 枚举子), 700
WIFI_EVENT_AP_STACONNECTED (C++ 枚举子), 700
wifi_event_ap_staconnected_t (C++ 类), 689
wifi_event_ap_staconnected_t::aid (C++ 成员), 689
wifi_event_ap_staconnected_t::mac (C++ 成员), 689
WIFI_EVENT_AP_STADISCONNECTED (C++ 枚举子), 700
wifi_event_ap_stadisconnected_t (C++ 类), 689
wifi_event_ap_stadisconnected_t::aid (C++ 成员), 689
wifi_event_ap_stadisconnected_t::mac (C++ 成员), 689
WIFI_EVENT_AP_START (C++ 枚举子), 700
WIFI_EVENT_AP_STOP (C++ 枚举子), 700
WIFI_EVENT_MASK_ALL (C 宏), 691
WIFI_EVENT_MASK_AP_PROBEREQRECVED (C 宏), 691
WIFI_EVENT_MASK_NONE (C 宏), 691
WIFI_EVENT_MAX (C++ 枚举子), 700
WIFI_EVENT_SCAN_DONE (C++ 枚举子), 699
WIFI_EVENT_STA_AUTHMODE_CHANGE (C++ 枚举子), 699
wifi_event_sta_authmode_change_t (C++ 类), 689
wifi_event_sta_authmode_change_t::new_mode (C++ 成员), 689
wifi_event_sta_authmode_change_t::old_mode (C++ 成员), 689
WIFI_EVENT_STA_CONNECTED (C++ 枚举子), 699
wifi_event_sta_connected_t (C++ 类), 688
wifi_event_sta_connected_t::authmode (C++ 成员), 688
wifi_event_sta_connected_t::bssid (C++ 成员), 688
wifi_event_sta_connected_t::channel (C++ 成员), 688
wifi_event_sta_connected_t::ssid (C++ 成员), 688
wifi_event_sta_connected_t::ssid_len (C++ 成员), 688
WIFI_EVENT_STA_DISCONNECTED (C++ 枚举子), 699
wifi_event_sta_disconnected_t (C++ 类), 688
wifi_event_sta_disconnected_t::bssid (C++ 成员), 688
wifi_event_sta_disconnected_t::reason (C++ 成员), 688
wifi_event_sta_disconnected_t::ssid (C++ 成员), 688
wifi_event_sta_disconnected_t::ssid_len

(C++ 成员), 688
wifi_event_sta_scan_done_t (C++ 类), 687
wifi_event_sta_scan_done_t::number (C++ 成员), 688
wifi_event_sta_scan_done_t::scan_id (C++ 成员), 688
wifi_event_sta_scan_done_t::status (C++ 成员), 688
WIFI_EVENT_STA_START (C++ 枚举子), 699
WIFI_EVENT_STA_STOP (C++ 枚举子), 699
WIFI_EVENT_STA_WPS_ER_FAILED (C++ 枚举子), 699
WIFI_EVENT_STA_WPS_ER_PBC_OVERLAP (C++ 枚举子), 700
WIFI_EVENT_STA_WPS_ER_PIN (C++ 枚举子), 700
wifi_event_sta_wps_er_pin_t (C++ 类), 689
wifi_event_sta_wps_er_pin_t::pin_code (C++ 成员), 689
WIFI_EVENT_STA_WPS_ER_SUCCESS (C++ 枚举子), 699
WIFI_EVENT_STA_WPS_ER_TIMEOUT (C++ 枚举子), 699
wifi_event_sta_wps_fail_reason_t (C++ 类型), 700
wifi_event_t (C++ 类型), 699
WIFI_EVENT_WIFI_READY (C++ 枚举子), 699
WIFI_FAST_SCAN (C++ 枚举子), 695
WIFI_IF_AP (C 宏), 690
WIFI_IF_STA (C 宏), 690
WIFI_INIT_CONFIG_DEFAULT (C 宏), 676
WIFI_INIT_CONFIG_MAGIC (C 宏), 676
wifi_init_config_t (C++ 类), 673
wifi_init_config_t::ampdu_rx_enable (C++ 成员), 674
wifi_init_config_t::ampdu_tx_enable (C++ 成员), 674
wifi_init_config_t::beacon_max_len (C++ 成员), 674
wifi_init_config_t::csi_enable (C++ 成员), 674
wifi_init_config_t::dynamic_rx_buf_num (C++ 成员), 674
wifi_init_config_t::dynamic_tx_buf_num (C++ 成员), 674
wifi_init_config_t::event_handler (C++ 成员), 674
wifi_init_config_t::feature_caps (C++ 成员), 675
wifi_init_config_t::magic (C++ 成员), 675
wifi_init_config_t::mgmt_sbuf_num (C++ 成员), 675
wifi_init_config_t::nano_enable (C++ 成员), 674
wifi_init_config_t::nvs_enable (C++ 成员), 674
wifi_init_config_t::osi_funcs (C++ 成员), 674
wifi_init_config_t::rx_ba_win (C++ 成员), 674
wifi_init_config_t::static_rx_buf_num (C++ 成员), 674
wifi_init_config_t::static_tx_buf_num (C++ 成员), 674
wifi_init_config_t::tx_ba_win (C++ 成员), 674
wifi_init_config_t::tx_buf_type (C++ 成员), 674
wifi_init_config_t::wifi_task_core_id (C++ 成员), 674
wifi_init_config_t::wpa_crypto_funcs (C++ 成员), 674
wifi_interface_t (C++ 类型), 691
WIFI_MGMT_SBUF_NUM (C 宏), 676
WIFI_MODE_AP (C++ 枚举子), 692
WIFI_MODE_APSTA (C++ 枚举子), 692
WIFI_MODE_MAX (C++ 枚举子), 692
WIFI_MODE_NULL (C++ 枚举子), 691
WIFI_MODE_STA (C++ 枚举子), 691
wifi_mode_t (C++ 类型), 691
WIFI_NANO_FORMAT_ENABLED (C 宏), 676
WIFI_NVS_ENABLED (C 宏), 676
WIFI_PHY_RATE_11M_L (C++ 枚举子), 697
WIFI_PHY_RATE_11M_S (C++ 枚举子), 697
WIFI_PHY_RATE_12M (C++ 枚举子), 697
WIFI_PHY_RATE_18M (C++ 枚举子), 698
WIFI_PHY_RATE_1M_L (C++ 枚举子), 697
WIFI_PHY_RATE_24M (C++ 枚举子), 697

WIFI_PHY_RATE_2M_L (C++ 枚举子), 697
WIFI_PHY_RATE_2M_S (C++ 枚举子), 697
WIFI_PHY_RATE_36M (C++ 枚举子), 698
WIFI_PHY_RATE_48M (C++ 枚举子), 697
WIFI_PHY_RATE_54M (C++ 枚举子), 698
WIFI_PHY_RATE_5M_L (C++ 枚举子), 697
WIFI_PHY_RATE_5M_S (C++ 枚举子), 697
WIFI_PHY_RATE_6M (C++ 枚举子), 697
WIFI_PHY_RATE_9M (C++ 枚举子), 698
WIFI_PHY_RATE_LORA_250K (C++ 枚举子), 699
WIFI_PHY_RATE_LORA_500K (C++ 枚举子), 699
WIFI_PHY_RATE_MAX (C++ 枚举子), 699
WIFI_PHY_RATE_MCS0_LGI (C++ 枚举子), 698
WIFI_PHY_RATE_MCS0_SGI (C++ 枚举子), 698
WIFI_PHY_RATE_MCS1_LGI (C++ 枚举子), 698
WIFI_PHY_RATE_MCS1_SGI (C++ 枚举子), 698
WIFI_PHY_RATE_MCS2_LGI (C++ 枚举子), 698
WIFI_PHY_RATE_MCS2_SGI (C++ 枚举子), 698
WIFI_PHY_RATE_MCS3_LGI (C++ 枚举子), 698
WIFI_PHY_RATE_MCS3_SGI (C++ 枚举子), 698
WIFI_PHY_RATE_MCS4_LGI (C++ 枚举子), 698
WIFI_PHY_RATE_MCS4_SGI (C++ 枚举子), 698
WIFI_PHY_RATE_MCS5_LGI (C++ 枚举子), 698
WIFI_PHY_RATE_MCS5_SGI (C++ 枚举子), 699
WIFI_PHY_RATE_MCS6_LGI (C++ 枚举子), 698
WIFI_PHY_RATE_MCS6_SGI (C++ 枚举子), 699
WIFI_PHY_RATE_MCS7_LGI (C++ 枚举子), 698
WIFI_PHY_RATE_MCS7_SGI (C++ 枚举子), 699
wifi_phy_rate_t (C++ 类型), 697
WIFI_PKT_CTRL (C++ 枚举子), 696
WIFI_PKT_DATA (C++ 枚举子), 696
WIFI_PKT_MGMT (C++ 枚举子), 696
WIFI_PKT_MISC (C++ 枚举子), 696
wifi_pkt_rx_ctrl_t (C++ 类), 683
wifi_pkt_rx_ctrl_t::__pad0__ (C++ 成员), 684
wifi_pkt_rx_ctrl_t::__pad1__ (C++ 成员), 684
wifi_pkt_rx_ctrl_t::__pad2__ (C++ 成员), 684
wifi_pkt_rx_ctrl_t::__pad3__ (C++ 成员), 684
wifi_pkt_rx_ctrl_t::__pad4__ (C++ 成员), 685
wifi_pkt_rx_ctrl_t::__pad5__ (C++ 成员), 685
wifi_pkt_rx_ctrl_t::__pad6__ (C++ 成员), 685
wifi_pkt_rx_ctrl_t::__pad7__ (C++ 成员), 685
wifi_pkt_rx_ctrl_t::aggregation (C++ 成员), 684
wifi_pkt_rx_ctrl_t::ampdu_cnt (C++ 成员), 684
wifi_pkt_rx_ctrl_t::ant (C++ 成员), 685
wifi_pkt_rx_ctrl_t::channel (C++ 成员), 684
wifi_pkt_rx_ctrl_t::cwb (C++ 成员), 684
wifi_pkt_rx_ctrl_t::fec_coding (C++ 成员), 684
wifi_pkt_rx_ctrl_t::mcs (C++ 成员), 684
wifi_pkt_rx_ctrl_t::not_sounding (C++ 成员), 684
wifi_pkt_rx_ctrl_t::rate (C++ 成员), 684
wifi_pkt_rx_ctrl_t::rssi (C++ 成员), 684
wifi_pkt_rx_ctrl_t::rx_state (C++ 成员), 685
wifi_pkt_rx_ctrl_t::secondary_channel (C++ 成员), 685
wifi_pkt_rx_ctrl_t::sgi (C++ 成员), 684
wifi_pkt_rx_ctrl_t::sig_len (C++ 成员), 685
wifi_pkt_rx_ctrl_t::sig_mode (C++ 成员), 684
wifi_pkt_rx_ctrl_t::smoothing (C++ 成员), 684
wifi_pkt_rx_ctrl_t::stbc (C++ 成员), 684
wifi_pkt_rx_ctrl_t::timestamp (C++ 成员), 685
wifi_pmf_config_t (C++ 类), 680
wifi_pmf_config_t::capable (C++ 成员), 681
wifi_pmf_config_t::required (C++ 成员), 681
WIFI_PROMIS_CTRL_FILTER_MASK_ACK (C 宏), 691
WIFI_PROMIS_CTRL_FILTER_MASK_ALL (C 宏), 690
WIFI_PROMIS_CTRL_FILTER_MASK_BA (C 宏), 691
WIFI_PROMIS_CTRL_FILTER_MASK_BAR (C 宏), 691
WIFI_PROMIS_CTRL_FILTER_MASK_CFEND (C 宏), 691
WIFI_PROMIS_CTRL_FILTER_MASK_CFENDACK (C 宏), 691
WIFI_PROMIS_CTRL_FILTER_MASK_CTS (C 宏), 691
WIFI_PROMIS_CTRL_FILTER_MASK_PSPOLL (C 宏), 691
WIFI_PROMIS_CTRL_FILTER_MASK_RTS (C 宏), 691
WIFI_PROMIS_CTRL_FILTER_MASK_WRAPPER (C 宏), 690
WIFI_PROMIS_FILTER_MASK_ALL (C 宏), 690
WIFI_PROMIS_FILTER_MASK_CTRL (C 宏), 690
WIFI_PROMIS_FILTER_MASK_DATA (C 宏), 690
WIFI_PROMIS_FILTER_MASK_DATA_AMPDU (C 宏), 690

WIFI_PROMIS_FILTER_MASK_DATA_MPDU (C 宏), 690
WIFI_PROMIS_FILTER_MASK_MGMT (C 宏), 690
WIFI_PROMIS_FILTER_MASK_MISC (C 宏), 690
wifi_promiscuous_cb_t (C++ 类型), 676
wifi_promiscuous_filter_t (C++ 类), 685
wifi_promiscuous_filter_t::filter_mask (C++ 成员), 686
wifi_promiscuous_pkt_t (C++ 类), 685
wifi_promiscuous_pkt_t::payload (C++ 成员), 685
wifi_promiscuous_pkt_t::rx_ctrl (C++ 成员), 685
wifi_promiscuous_pkt_type_t (C++ 类型), 696
WIFI_PROTOCOL_11B (C 宏), 690
WIFI_PROTOCOL_11G (C 宏), 690
WIFI_PROTOCOL_11N (C 宏), 690
WIFI_PROTOCOL_LR (C 宏), 690
wifi_prov_cb_event_t (C++ 类型), 1300
wifi_prov_cb_func_t (C++ 类型), 1300
wifi_prov_config_data_handler (C++ 函数), 1303
wifi_prov_config_get_data_t (C++ 类), 1303
wifi_prov_config_get_data_t::conn_info (C++ 成员), 1303
wifi_prov_config_get_data_t::fail_reason (C++ 成员), 1303
wifi_prov_config_get_data_t::wifi_state (C++ 成员), 1303
wifi_prov_config_handlers (C++ 类), 1304
wifi_prov_config_handlers::apply_config_handler (C++ 成员), 1304
wifi_prov_config_handlers::ctx (C++ 成员), 1304
wifi_prov_config_handlers::get_status_handler (C++ 成员), 1304
wifi_prov_config_handlers::set_config_handler (C++ 成员), 1304
wifi_prov_config_handlers_t (C++ 类型), 1305
wifi_prov_config_set_data_t (C++ 类), 1303
wifi_prov_config_set_data_t::bssid (C++ 成员), 1304
wifi_prov_config_set_data_t::channel (C++ 成员), 1304
wifi_prov_config_set_data_t::password (C++ 成员), 1304
wifi_prov_config_set_data_t::ssid (C++ 成员), 1304
WIFI_PROV_CRED_FAIL (C++ 枚举子), 1300
WIFI_PROV_CRED_RECV (C++ 枚举子), 1300
WIFI_PROV_CRED_SUCCESS (C++ 枚举子), 1300
wifi_prov_ctx_t (C++ 类型), 1304
WIFI_PROV_DEINIT (C++ 枚举子), 1301
WIFI_PROV_END (C++ 枚举子), 1301
WIFI_PROV_EVENT_HANDLER_NONE (C 宏), 1299
wifi_prov_event_handler_t (C++ 类), 1298
wifi_prov_event_handler_t::event_cb (C++ 成员), 1298
wifi_prov_event_handler_t::user_data (C++ 成员), 1298
WIFI_PROV_INIT (C++ 枚举子), 1300
wifi_prov_mgr_config_t (C++ 类), 1299
wifi_prov_mgr_config_t::app_event_handler (C++ 成员), 1299
wifi_prov_mgr_config_t::scheme (C++ 成员), 1299
wifi_prov_mgr_config_t::scheme_event_handler (C++ 成员), 1299
wifi_prov_mgr_configure_sta (C++ 函数), 1297
wifi_prov_mgr_deinit (C++ 函数), 1291
wifi_prov_mgr_disable_auto_stop (C++ 函数), 1294
wifi_prov_mgr_endpoint_create (C++ 函数), 1295
wifi_prov_mgr_endpoint_register (C++ 函数), 1295
wifi_prov_mgr_endpoint_unregister (C++ 函数), 1296
wifi_prov_mgr_event_handler (C++ 函数), 1296
wifi_prov_mgr_get_wifi_disconnect_reason (C++ 函数), 1297
wifi_prov_mgr_get_wifi_state (C++ 函数), 1297
wifi_prov_mgr_init (C++ 函数), 1291
wifi_prov_mgr_is_provisioned (C++ 函数), 1292
wifi_prov_mgr_set_app_info (C++ 函数), 1294

wifi_prov_mgr_start_provisioning (C++ 函数), 1292
wifi_prov_mgr_stop_provisioning (C++ 函数), 1293
wifi_prov_mgr_wait (C++ 函数), 1294
wifi_prov_scheme (C++ 类), 1298
wifi_prov_scheme::delete_config (C++ 成员), 1298
wifi_prov_scheme::new_config (C++ 成员), 1298
wifi_prov_scheme::prov_start (C++ 成员), 1298
wifi_prov_scheme::prov_stop (C++ 成员), 1298
wifi_prov_scheme::set_config_endpoint (C++ 成员), 1298
wifi_prov_scheme::set_config_service (C++ 成员), 1298
wifi_prov_scheme::wifi_mode (C++ 成员), 1299
wifi_prov_scheme_ble_event_cb_free_ble (C++ 函数), 1301
wifi_prov_scheme_ble_event_cb_free_bt (C++ 函数), 1301
wifi_prov_scheme_ble_event_cb_free_btdm (C++ 函数), 1301
WIFI_PROV_SCHEME_BLE_EVENT_HANDLER_FREE_BLE (C 宏), 1302
WIFI_PROV_SCHEME_BLE_EVENT_HANDLER_FREE_BT (C 宏), 1302
WIFI_PROV_SCHEME_BLE_EVENT_HANDLER_FREE_BTDM (C 宏), 1302
wifi_prov_scheme_ble_set_service_uuid (C++ 函数), 1301
wifi_prov_scheme_softap_set_httpd_handle (C++ 函数), 1302
wifi_prov_scheme_t (C++ 类型), 1300
wifi_prov_security (C++ 类型), 1301
WIFI_PROV_SECURITY_0 (C++ 枚举子), 1301
WIFI_PROV_SECURITY_1 (C++ 枚举子), 1301
wifi_prov_security_t (C++ 类型), 1300
WIFI_PROV_STA_AP_NOT_FOUND (C++ 枚举子), 1305
WIFI_PROV_STA_AUTH_ERROR (C++ 枚举子), 1305
wifi_prov_sta_conn_info_t (C++ 类), 1303
wifi_prov_sta_conn_info_t::auth_mode (C++ 成员), 1303
wifi_prov_sta_conn_info_t::bssid (C++ 成员), 1303
wifi_prov_sta_conn_info_t::channel (C++ 成员), 1303
wifi_prov_sta_conn_info_t::ip_addr (C++ 成员), 1303
wifi_prov_sta_conn_info_t::ssid (C++ 成员), 1303
WIFI_PROV_STA_CONNECTED (C++ 枚举子), 1305
WIFI_PROV_STA_CONNECTING (C++ 枚举子), 1305
WIFI_PROV_STA_DISCONNECTED (C++ 枚举子), 1305
wifi_prov_sta_fail_reason_t (C++ 类型), 1305
wifi_prov_sta_state_t (C++ 类型), 1305
WIFI_PROV_START (C++ 枚举子), 1300
WIFI_PS_MAX_MODEM (C++ 枚举子), 695
WIFI_PS_MIN_MODEM (C++ 枚举子), 695
WIFI_PS_NONE (C++ 枚举子), 695
wifi_ps_type_t (C++ 类型), 695
WIFI_REASON_4WAY_HANDSHAKE_TIMEOUT (C++ 枚举子), 693
WIFI_REASON_802_1X_AUTH_FAILED (C++ 枚举子), 693
WIFI_REASON_AKMP_INVALID (C++ 枚举子), 693
WIFI_REASON_ASSOC_EXPIRE (C++ 枚举子), 693
WIFI_REASON_ASSOC_FAIL (C++ 枚举子), 693
WIFI_REASON_ASSOC_LEAVE (C++ 枚举子), 693
WIFI_REASON_ASSOC_NOT_AUTHED (C++ 枚举子), 693
WIFI_REASON_ASSOC_TOOMANY (C++ 枚举子), 693
WIFI_REASON_AUTH_EXPIRE (C++ 枚举子), 692
WIFI_REASON_AUTH_FAIL (C++ 枚举子), 693
WIFI_REASON_AUTH_LEAVE (C++ 枚举子), 693
WIFI_REASON_BEACON_TIMEOUT (C++ 枚举子), 693
WIFI_REASON_CIPHER_SUITE_REJECTED (C++ 枚举子), 693
WIFI_REASON_CONNECTION_FAIL (C++ 枚举子), 693
WIFI_REASON_DISASSOC_PWRCAP_BAD (C++ 枚举子), 693
WIFI_REASON_DISASSOC_SUPCHAN_BAD (C++ 枚举子), 693
WIFI_REASON_GROUP_CIPHER_INVALID (C++ 枚举子), 693

WIFI_REASON_GROUP_KEY_UPDATE_TIMEOUT (C++ 枚举子), 693
WIFI_REASON_HANDSHAKE_TIMEOUT (C++ 枚举子), 693
WIFI_REASON_IE_IN_4WAY_DIFFERS (C++ 枚举子), 693
WIFI_REASON_IE_INVALID (C++ 枚举子), 693
WIFI_REASON_INVALID_PMKID (C++ 枚举子), 693
WIFI_REASON_INVALID_RSN_IE_CAP (C++ 枚举子), 693
WIFI_REASON_MIC_FAILURE (C++ 枚举子), 693
WIFI_REASON_NO_AP_FOUND (C++ 枚举子), 693
WIFI_REASON_NOT_ASSOCED (C++ 枚举子), 693
WIFI_REASON_NOT_AUTHED (C++ 枚举子), 693
WIFI_REASON_PAIRWISE_CIPHER_INVALID (C++ 枚举子), 693
WIFI_REASON_UNSPECIFIED (C++ 枚举子), 692
WIFI_REASON_UNSUPP_RSN_IE_VERSION (C++ 枚举子), 693
wifi_scan_config_t (C++ 类), 679
wifi_scan_config_t::bssid (C++ 成员), 679
wifi_scan_config_t::channel (C++ 成员), 679
wifi_scan_config_t::scan_time (C++ 成员), 679
wifi_scan_config_t::scan_type (C++ 成员), 679
wifi_scan_config_t::show_hidden (C++ 成员), 679
wifi_scan_config_t::ssid (C++ 成员), 679
wifi_scan_method_t (C++ 类型), 695
wifi_scan_threshold_t (C++ 类), 680
wifi_scan_threshold_t::authmode (C++ 成员), 680
wifi_scan_threshold_t::rssi (C++ 成员), 680
wifi_scan_time_t (C++ 类), 678
wifi_scan_time_t::active (C++ 成员), 679
wifi_scan_time_t::passive (C++ 成员), 679
WIFI_SCAN_TYPE_ACTIVE (C++ 枚举子), 694
WIFI_SCAN_TYPE_PASSIVE (C++ 枚举子), 694
wifi_scan_type_t (C++ 类型), 694
WIFI_SECOND_CHAN_ABOVE (C++ 枚举子), 694
WIFI_SECOND_CHAN_BELOW (C++ 枚举子), 694
WIFI_SECOND_CHAN_NONE (C++ 枚举子), 694
wifi_second_chan_t (C++ 类型), 693
WIFI_SOFTAP_BEACON_MAX_LEN (C 宏), 676
wifi_sort_method_t (C++ 类型), 695
wifi_sta_config_t (C++ 类), 681
wifi_sta_config_t::bssid (C++ 成员), 682
wifi_sta_config_t::bssid_set (C++ 成员), 682
wifi_sta_config_t::channel (C++ 成员), 682
wifi_sta_config_t::listen_interval (C++ 成员), 682
wifi_sta_config_t::password (C++ 成员), 681
wifi_sta_config_t::pmf_cfg (C++ 成员), 682
wifi_sta_config_t::scan_method (C++ 成员), 682
wifi_sta_config_t::sort_method (C++ 成员), 682
wifi_sta_config_t::ssid (C++ 成员), 681
wifi_sta_config_t::threshold (C++ 成员), 682
wifi_sta_info_t (C++ 类), 682
wifi_sta_info_t::mac (C++ 成员), 682
wifi_sta_info_t::phy_11b (C++ 成员), 682
wifi_sta_info_t::phy_11g (C++ 成员), 682
wifi_sta_info_t::phy_11n (C++ 成员), 682
wifi_sta_info_t::phy_lr (C++ 成员), 683
wifi_sta_info_t::reserved (C++ 成员), 683
wifi_sta_info_t::rssi (C++ 成员), 682
wifi_sta_list_t (C++ 类), 683
wifi_sta_list_t::num (C++ 成员), 683
wifi_sta_list_t::sta (C++ 成员), 683
WIFI_STATIC_TX_BUFFER_NUM (C 宏), 676
WIFI_STORAGE_FLASH (C++ 枚举子), 695
WIFI_STORAGE_RAM (C++ 枚举子), 696
wifi_storage_t (C++ 类型), 695
WIFI_TASK_CORE_ID (C 宏), 676
WIFI_VENDOR_IE_ELEMENT_ID (C 宏), 690
wifi_vendor_ie_id_t (C++ 类型), 696
wifi_vendor_ie_type_t (C++ 类型), 696
WIFI_VND_IE_ID_0 (C++ 枚举子), 696
WIFI_VND_IE_ID_1 (C++ 枚举子), 696
WIFI_VND_IE_TYPE_ASSOC_REQ (C++ 枚举子), 696
WIFI_VND_IE_TYPE_ASSOC_RESP (C++ 枚举子), 696
WIFI_VND_IE_TYPE_BEACON (C++ 枚举子), 696
WIFI_VND_IE_TYPE_PROBE_REQ (C++ 枚举子), 696
WIFI_VND_IE_TYPE_PROBE_RESP (C++ 枚举子), 696

wl_erase_range (C++ 函数), 1400
wl_handle_t (C++ 类型), 1402
WL_INVALID_HANDLE (C 宏), 1402
wl_mount (C++ 函数), 1400
wl_read (C++ 函数), 1401
wl_sector_size (C++ 函数), 1402
wl_size (C++ 函数), 1402
wl_unmount (C++ 函数), 1400
wl_write (C++ 函数), 1401
WPS_FAIL_REASON_MAX (C++ 枚举子), 700
WPS_FAIL_REASON_NORMAL (C++ 枚举子), 700
WPS_FAIL_REASON_RECV_M2D (C++ 枚举子), 700

X

xEventGroupClearBits (C++ 函数), 1582
xEventGroupClearBitsFromISR (C 宏), 1588
xEventGroupCreate (C++ 函数), 1578
xEventGroupCreateStatic (C++ 函数), 1579
xEventGroupGetBits (C 宏), 1590
xEventGroupGetBitsFromISR (C++ 函数), 1587
xEventGroupSetBits (C++ 函数), 1583
xEventGroupSetBitsFromISR (C 宏), 1588
xEventGroupSync (C++ 函数), 1585
xEventGroupWaitBits (C++ 函数), 1580
xQueueAddToSet (C++ 函数), 1515
xQueueCreate (C 宏), 1517
xQueueCreateSet (C++ 函数), 1514
xQueueCreateStatic (C 宏), 1518
xQueueGenericCreate (C++ 函数), 1514
xQueueGenericCreateStatic (C++ 函数), 1514
xQueueGenericReceive (C++ 函数), 1509
xQueueGenericSend (C++ 函数), 1507
xQueueGenericSendFromISR (C++ 函数), 1505
xQueueGiveFromISR (C++ 函数), 1506
xQueueIsQueueEmptyFromISR (C++ 函数), 1507
xQueueIsQueueFullFromISR (C++ 函数), 1507
xQueueOverwrite (C 宏), 1524
xQueueOverwriteFromISR (C 宏), 1532
xQueuePeek (C 宏), 1526
xQueuePeekFromISR (C++ 函数), 1508
xQueueReceive (C 宏), 1527
xQueueReceiveFromISR (C++ 函数), 1511
xQueueRemoveFromSet (C++ 函数), 1515
xQueueReset (C 宏), 1534
xQueueSelectFromSet (C++ 函数), 1516
xQueueSelectFromSetFromISR (C++ 函数), 1517
xQueueSend (C 宏), 1522
xQueueSendFromISR (C 宏), 1533
xQueueSendToBack (C 宏), 1521
xQueueSendToBackFromISR (C 宏), 1530
xQueueSendToFront (C 宏), 1519
xQueueSendToFrontFromISR (C 宏), 1529
xRingbufferAddToQueueSetRead (C++ 函数), 1610
xRingbufferCanRead (C++ 函数), 1610
xRingbufferCreate (C++ 函数), 1602
xRingbufferCreateNoSplit (C++ 函数), 1603
xRingbufferCreateStatic (C++ 函数), 1603
xRingbufferGetCurFreeSize (C++ 函数), 1609
xRingbufferGetMaxItemSize (C++ 函数), 1609
xRingbufferPrintInfo (C++ 函数), 1611
xRingbufferReceive (C++ 函数), 1605
xRingbufferReceiveFromISR (C++ 函数), 1606
xRingbufferReceiveSplit (C++ 函数), 1606
xRingbufferReceiveSplitFromISR (C++ 函数), 1607
xRingbufferReceiveUpTo (C++ 函数), 1607
xRingbufferReceiveUpToFromISR (C++ 函数), 1608
xRingbufferRemoveFromQueueSetRead (C++ 函数), 1610
xRingbufferSend (C++ 函数), 1603
xRingbufferSendAcquire (C++ 函数), 1604
xRingbufferSendComplete (C++ 函数), 1605
xRingbufferSendFromISR (C++ 函数), 1604
xSemaphoreCreateBinary (C 宏), 1535
xSemaphoreCreateBinaryStatic (C 宏), 1536
xSemaphoreCreateCounting (C 宏), 1550
xSemaphoreCreateCountingStatic (C 宏), 1552
xSemaphoreCreateMutex (C 宏), 1546
xSemaphoreCreateMutexStatic (C 宏), 1547
xSemaphoreCreateRecursiveMutex (C 宏), 1548
xSemaphoreCreateRecursiveMutexStatic (C 宏), 1549
xSemaphoreGetMutexHolder (C 宏), 1553

xSemaphoreGive (C 宏), 1541
xSemaphoreGiveFromISR (C 宏), 1544
xSemaphoreGiveRecursive (C 宏), 1542
xSemaphoreTake (C 宏), 1537
xSemaphoreTakeFromISR (C 宏), 1546
xSemaphoreTakeRecursive (C 宏), 1539
xSTATIC_RINGBUFFER (C++ 类), 1612
xTASK_SNAPSHOT (C++ 类), 1501
xTASK_SNAPSHOT::pxEndOfStack (C++ 成员), 1501
xTASK_SNAPSHOT::pxTCB (C++ 成员), 1501
xTASK_SNAPSHOT::pxTopOfStack (C++ 成员), 1501
xTASK_STATUS (C++ 类), 1500
xTASK_STATUS::eCurrentState (C++ 成员), 1500
xTASK_STATUS::pcTaskName (C++ 成员), 1500
xTASK_STATUS::pxStackBase (C++ 成员), 1501
xTASK_STATUS::ulRunTimeCounter (C++ 成员), 1501
xTASK_STATUS::usStackHighWaterMark (C++ 成员), 1501
xTASK_STATUS::uxBasePriority (C++ 成员), 1501
xTASK_STATUS::uxCurrentPriority (C++ 成员), 1500
xTASK_STATUS::xCoreID (C++ 成员), 1501
xTASK_STATUS::xHandle (C++ 成员), 1500
xTASK_STATUS::xTaskNumber (C++ 成员), 1500
xTaskCallApplicationTaskHook (C++ 函数), 1489
xTaskCreate (C++ 函数), 1472
xTaskCreatePinnedToCore (C++ 函数), 1472
xTaskCreateStatic (C++ 函数), 1475
xTaskCreateStaticPinnedToCore (C++ 函数), 1474
xTaskGetApplicationTaskTag (C++ 函数), 1488
xTaskGetIdleTaskHandle (C++ 函数), 1490
xTaskGetIdleTaskHandleForCPU (C++ 函数), 1490
xTaskGetTickCount (C++ 函数), 1487
xTaskGetTickCountFromISR (C++ 函数), 1487
xTaskNotify (C++ 函数), 1494
xTaskNotifyFromISR (C++ 函数), 1495
xTaskNotifyGive (C 宏), 1502
xTaskNotifyWait (C++ 函数), 1497
xTaskResumeAll (C++ 函数), 1485
xTaskResumeFromISR (C++ 函数), 1484
xtensa_perfmon_config (C++ 类), 1705
xtensa_perfmon_config::call_function (C++ 成员), 1705
xtensa_perfmon_config::call_params (C++ 成员), 1705
xtensa_perfmon_config::callback (C++ 成员), 1706
xtensa_perfmon_config::callback_params (C++ 成员), 1706
xtensa_perfmon_config::counters_size (C++ 成员), 1706
xtensa_perfmon_config::max_deviation (C++ 成员), 1705
xtensa_perfmon_config::repeat_count (C++ 成员), 1705
xtensa_perfmon_config::select_mask (C++ 成员), 1706
xtensa_perfmon_config::tracelevel (C++ 成员), 1706
xtensa_perfmon_config_t (C++ 类型), 1706
xtensa_perfmon_dump (C++ 函数), 1704
xtensa_perfmon_exec (C++ 函数), 1704
xtensa_perfmon_init (C++ 函数), 1703
xtensa_perfmon_overflow (C++ 函数), 1704
xtensa_perfmon_reset (C++ 函数), 1703
xtensa_perfmon_start (C++ 函数), 1704
xtensa_perfmon_stop (C++ 函数), 1704
xtensa_perfmon_value (C++ 函数), 1704
xtensa_perfmon_view_cb (C++ 函数), 1705
xTimerChangePeriod (C 宏), 1566
xTimerChangePeriodFromISR (C 宏), 1575
xTimerCreate (C++ 函数), 1554
xTimerCreateStatic (C++ 函数), 1557
xTimerDelete (C 宏), 1568
xTimerGetExpiryTime (C++ 函数), 1562
xTimerGetPeriod (C++ 函数), 1561
xTimerGetTimerDaemonTaskHandle (C++ 函数), 1561
xTimerIsTimerActive (C++ 函数), 1560
xTimerPendFunctionCall (C++ 函数), 1564
xTimerPendFunctionCallFromISR (C++ 函数), 1562

xTimerReset (*C 宏*), 1569
xTimerResetFromISR (*C 宏*), 1576
xTimerStart (*C 宏*), 1565
xTimerStartFromISR (*C 宏*), 1571
xTimerStop (*C 宏*), 1566
xTimerStopFromISR (*C 宏*), 1573

环境变量

CONFIG_EFUSE_CUSTOM_TABLE, 1428
CONFIG_EFUSE_MAX_BLK_LEN, 1429
CONFIG_EFUSE_VIRTUAL, 1434
CONFIG_ESPTOOLPY_FLASHSIZE, 1307
CONFIG_LOG_DEFAULT_LEVEL, 1670, 1671
CONFIG_LWIP_SNTP_UPDATE_DELAY, 1733
CONFIG_LWIP_USE_ONLY_LWIP_SELECT, 1380
CONFIG_SPIRAM_BANKSWITCH_ENABLE, 1655
CONFIG_SPIRAM_BANKSWITCH_RESERVE, 1655
EFUSE_CODE_SCHEME_SELECTOR, 1431