




# ESP32

## Bluetooth Architecture



Version 1.1  
Espressif Systems  
Copyright © 2019



# About This Guide

---

This document introduces the ESP32 Bluetooth<sup>®</sup> architecture, namely Bluetooth, Classic Bluetooth and Bluetooth Low Energy. Note that this user guide is only applicable to ESP-IDF V2.1 and earlier.

## Release Notes

Date	Version	Release notes
2018.01	V1.0	First release.
2019.11	V1.1	<ul style="list-style-type: none"><li>• Updated document cover;</li><li>• Updated an example in Section 3.24.</li></ul>

## Documentation Change Notification

Espressif provides email notifications to keep customers updated on changes to technical documentation. Please subscribe at <https://www.espressif.com/en/subscribe>.

## Certification

Download certificates for Espressif products from <https://www.espressif.com/en/certificates>.

# Table of Contents

<b>1. Bluetooth.....</b>	<b>1</b>
1.1. Overview .....	1
1.1.1. Bluetooth Application Structure .....	1
1.1.2. Selection of the HCI Interfaces .....	2
1.1.3. Bluetooth Operating Environment .....	4
1.2. Architecture .....	4
1.2.1. Controller .....	4
1.2.2. BLUEDROID .....	4
1.2.2.1. Overall Architecture .....	4
1.2.2.2. OS-related Adaptation .....	6
1.2.3. Bluetooth Directory Introduction .....	6
<b>2. Classic Bluetooth.....</b>	<b>9</b>
2.1. Overview .....	9
2.1.1. L2CAP .....	10
2.1.2. SDP .....	10
2.1.3. GAP .....	10
2.1.4. A2DP and AVRCP .....	11
<b>3. Bluetooth Low Energy .....</b>	<b>14</b>
3.1. GAP .....	14
3.1.1. Overview .....	14
3.1.2. Status Transitions among GAP Roles .....	15
3.1.3. BLE Broadcast Procedure .....	16
3.1.3.1. Broadcast using a public address .....	16
3.1.3.2. Broadcast using a resolvable address .....	17
3.1.3.3. Broadcast using a static random address .....	18
3.1.4. BLE Modes .....	19
3.1.4.1. Connectable Scannable Undirected Mode .....	19
3.1.4.2. High Duty Cycle Directed Mode and Connectable Low Duty Cycle Directed Mode .....	19
3.1.4.3. Scannable Undirected Mode .....	19
3.1.4.4. Non-connectable Undirected Mode .....	20
3.1.5. BLE Broadcast Filtering Policy .....	20
3.1.6. BLE Scanning Procedure .....	20

3.1.7.	BLE GAP Implementation Mechanism .....	21
3.2.	GATT .....	21
3.2.1.	ATT .....	21
3.2.2.	GATT Profile.....	23
3.2.3.	Add Gatt Services in ESP32 IDF Environment .....	25
3.2.4.	Discover a Peer Device's Services in ESP32 IDF (GATT Client) .....	26
3.3.	SMP .....	27
3.3.1.	Overview .....	27
3.3.2.	Safety Management Controller.....	28
3.3.2.1.	BLE Encryption .....	28
3.3.2.2.	BLE Bonding .....	30
3.3.3.	The Implementation of SMP .....	30

---



# 1. Bluetooth

This chapter describes the basic Bluetooth architecture of ESP32.

## 1.1. Overview

### 1.1.1. Bluetooth Application Structure

蓝牙是一种用于短距离交换数据的无线技术标准，具有鲁棒性、低功耗和低成本等优点。

Bluetooth is a wireless technology standard for exchanging data over short distances, with advantages including robustness, low power consumption and low cost. The Bluetooth system can be divided into two different categories: Classic Bluetooth and Bluetooth Low Energy (BLE). ESP32 supports dual-mode Bluetooth, meaning that both Classic Bluetooth and BLE are supported by ESP32.

基本上，蓝牙协议栈分为两部分：“控制器栈”和“主机栈”。

Basically, the Bluetooth protocol stack is split into two parts: a “controller stack” and a “host stack”. The controller stack contains the PHY, Baseband, Link Controller, Link Manager, Device Manager, HCI and other modules, and is used for the hardware interface management and link management. The host stack contains L2CAP, SMP, SDP, ATT, GATT, GAP and various profiles, and functions as an interface to the application layer, thus facilitating the application layer to access the Bluetooth system. The Bluetooth Host can be implemented on the same device as the Controller, or on different devices. Both approaches are supported by ESP32. Figure 1-1 describes some typical application structures:

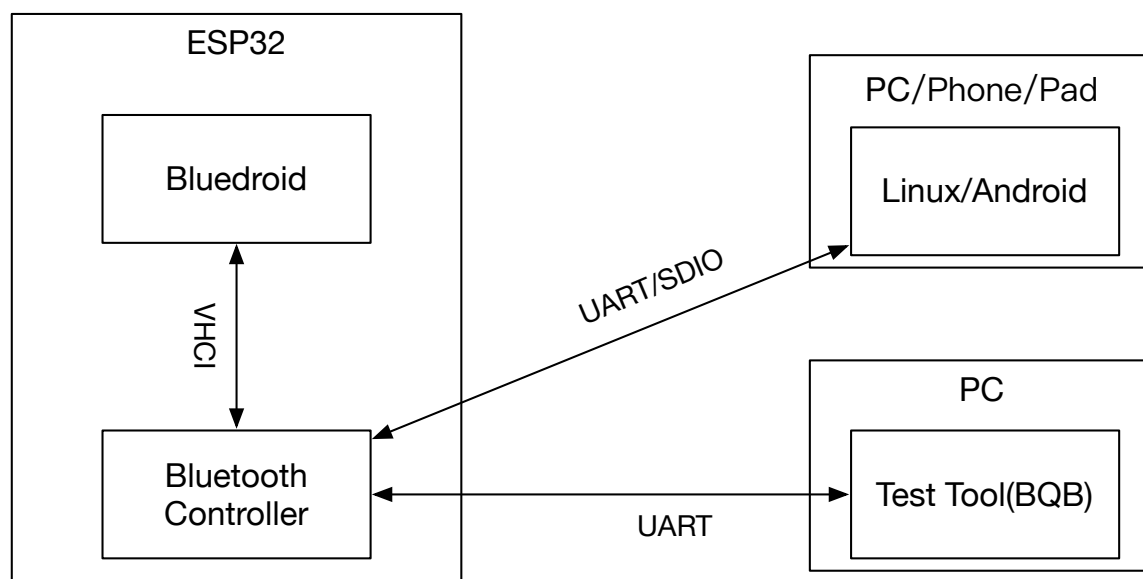


Figure 1-1. The architecture of Bluetooth host and controller in ESP-IDF

- Scenario 1 (Default ESP-IDF setting): BLUEDROID is selected as the Bluetooth Host, and VHCI (software-implemented virtual HCI interface) is used for the communication between Bluetooth Host and Controller. In this scenario, both the BLUEDROID and

场景一（ESP-IDF 默认设置）：蓝牙 Host 选择 BLUEDROID，蓝牙 Host 和 Controller 使用 VHCI（软件实现的虚拟 HCI 接口）进行通信。控制器在同一设备（即 ESP32 芯片）上实现，无需额外的 PC 或其他运行蓝牙主机的主机设备。



the Controller are implemented on the same device (i.e. the ESP32 chip), eliminating the need for an extra PC or other host devices running the Bluetooth Host.

- Scenario 2: the ESP32 system is used only as a Bluetooth Controller, and an extra device running the Bluetooth Host is required (such as a Linux PC running BlueZ or an Android device running BLUEDROID, etc). In this scenario, Controller and Host are implemented on different devices, which is quite similar to the case of mobile phones, PADs or PCs.
- Scenario 3: This scenario is similar to Scenario 2. The difference lies in that, in the BQB controller test (or other certifications), ESP32 can be tested by connecting it to the test tools, with the UART being enabled as the IO interface.

场景二：ESP32 系统仅作为蓝牙控制器使用，需要额外的运行蓝牙主机的设备（如运行 BlueZ 的 Linux PC 或运行 BLUEDROID 的 Android 设备等）。在这种场景下，Controller 和 Host 实现在不同的设备上，这与手机、PAD 或 PC 的情况非常相似。

场景三：该场景与场景二类似。不同之处在于，在BQB控制器测试（或其他认证）中，ESP32可以通过连接测试工具进行测试，并启用UART作为IO接口。

### 1.1.2. Selection of the HCI Interfaces

在 ESP32 系统中，HCI 一次只能使用一个 IO 接口，这意味着如果启用 UART，则禁用其他接口，例如 VHCI 和 SDIO。在 ESP-IDF (V2.1 及更高版本) 中，您可以在 menuconfig 界面将蓝牙 HCI IO 接口配置为 VHCI 或 UART，如下图：

In the ESP32 system, only one IO interface at a time can be used by HCI, meaning that if UART is enabled, other interfaces such as the VHCI and SDIO are disabled. In the ESP-IDF (V2.1 and later versions), you can configure the Bluetooth HCI IO interface as VHCI or UART in the *menuconfig* screen, as shown below:

```
--- Bluetooth
[ ] Bluedroid Bluetooth stack enabled --->
[ ] HCI use UART as IO (NEW) ----
```

Figure 1-2.Configuration of the HCI IO interface

当选择Bluedroid蓝牙堆栈启用选项时，将启用VHCI作为IO接口，并且HCI使用UART作为IO(新)选项将从菜单中消失。当选择了HCI使用UART作为IO(新)选项时，将启用UART作为IO接口。ESP-IDF目前不支持其他IO。如果要使用其他IO，如SPI，则需要SPI-VHCI网桥。

When the *Bluedroid Bluetooth stack enabled* option is selected, VHCI is enabled as the IO interface and the *HCI use UART as IO (NEW)* option will disappear from the menu.

When the *HCI use UART as IO (NEW)* option is selected, UART is enabled as the IO interface. Currently, other IOs are not supported in ESP-IDF. If you want to use other IOs, such as the SPI, a SPI-VHCI bridge is required.

#### Option 1:

When the *Bluedroid Bluetooth stack enabled* option is selected, the following screen is displayed:

当选择Bluedroid蓝牙堆栈启用选项时，将显示以下屏幕：



```
-- Bluetooth stack enabled
(3072) Bluetooth event (callback to application) task stack size
[ ] Blueroid memory debug
[ ] Classic Bluetooth
[ ] Release DRAM from Classic BT controller
[*] Include GATT server module(GATTS)
[*] Include GATT client module(GATTC)
[*] Include BLE security module(SMP)
[ ] Close the blueroid bt stack log print
(4) BT/BLE MAX ACL CONNECTIONS(1~7)
```

Figure 1-3.VHCI configuration

Here, users can configure the following items: [在这里，用户可以配置以下项目：](#)

- **Bluetooth event (callback to application) task stack size:** sets the size of the BTC Task; [设置BTC任务的大小](#)；
- **Blueroid memory debug:** debugs the BLUEDROID memory; [调试BLUEDROID内存](#)
- **Classic Bluetooth:** enables the Classic Bluetooth; [启用经典蓝牙](#)
- **Release DRAM from Classic BT Controller:** releases the DRAM from the Classic Bluetooth Controller; [从经典蓝牙控制器释放DRAM](#)
- **Include GATT server module (GATTS):** includes the GATTS module;
- **Include GATT client module (GATTC):** includes the GATTC module;
- **Include BLE security module (SMP):** includes the SMP module;
- **Close the blueroid bt stack log print:** closes the BLUEDROID printing;
- **BT/BLE MAX ACL CONNECTIONS (1~7):** sets the maximum number of ACL connections. [设置ACL连接的最大数量。](#)

**Option 2:** [当选择HCI使用UART作为IO选项时，将显示以下屏幕](#)

When the **HCI use UART as IO** option is selected, the following screen is displayed:

```
-- HCI use UART as IO
(1)  UART Number for HCI (NEW)
(921600) UART Baudrate for HCI (NEW)
```

Figure 1-4.UART configuration

Users can configure the "**UART Number for HCI (NEW)**" (UART port number) and the "**UART Baudrate for HCI (NEW)**" (the baud rate of UART) here. It should also be mentioned here that CTS/RTS must be supported, in order to enable the UART as the HCI IO interface.

[用户可以在这里配置“UART Number for HCI \(New\)” \(UART端口号\)和“UART Baudrate for HCI \(New\)” \(UART的波特率\)。这里还应该提到，必须支持CTS/RTS，才能启用UART作为HCI IO接口。](#)



## 蓝牙操作环境

### 1.1.3. Bluetooth Operating Environment

ESP-IDF的默认运行环境为双核FreeRTOS。ESP32蓝牙可以为基于功能的任务分配不同的优先级。具有最高优先级的任务是运行控制器的任务。在FreeRTOS系统中，除了IPC任务主要用于双核CPU之间的进程间通信外，对实时性要求较高的控制器任务具有最高的优先级。BLUEDROID(默认ESP-IDF蓝牙主机)总共包含四个任务，分别运行BTC、BTU、HCI UPWARD 和 HCI DOWNWARD。

The default operating environment of ESP-IDF is dual-core FreeRTOS. ESP32 Bluetooth can assign function-based tasks with different priorities. The tasks with the highest priority are the ones running the Controller. The Controller tasks, which have higher requirements of real time, have the highest priority in the FreeRTOS system except for the IPC tasks, which are mainly for the interprocess communication between the dual-core CPUs. BLUEDROID (the default ESP-IDF Bluetooth Host) contains four tasks in total, which run the BTC, BTU, HCI UPWARD, and HCI DOWNWARD.

## 1.2. Architecture

### 1.2.1. Controller

ESP32的蓝牙控制器支持Classic BT和BLE(v4.2)。该控制器集成了多种功能，包括H4协议、人机界面、链路管理器、链路控制器、设备管理器和硬件接口。这些函数以库的形式提供给开发人员，同时还提供了一些可以访问控制器的API。有关详细信息，请参阅ReadtheDocs。

The Bluetooth Controller of ESP32 supports both the Classic BT and BLE (V4.2). The Controller has integrated a variety of functions, including H4 protocol, HCI, Link Manager, Link Controller, Device Manager, and HW Interface. These functions are provided to the developers in the form of libraries, while some APIs that can access the Controller are also provided. For details, see [readthedocs](#).

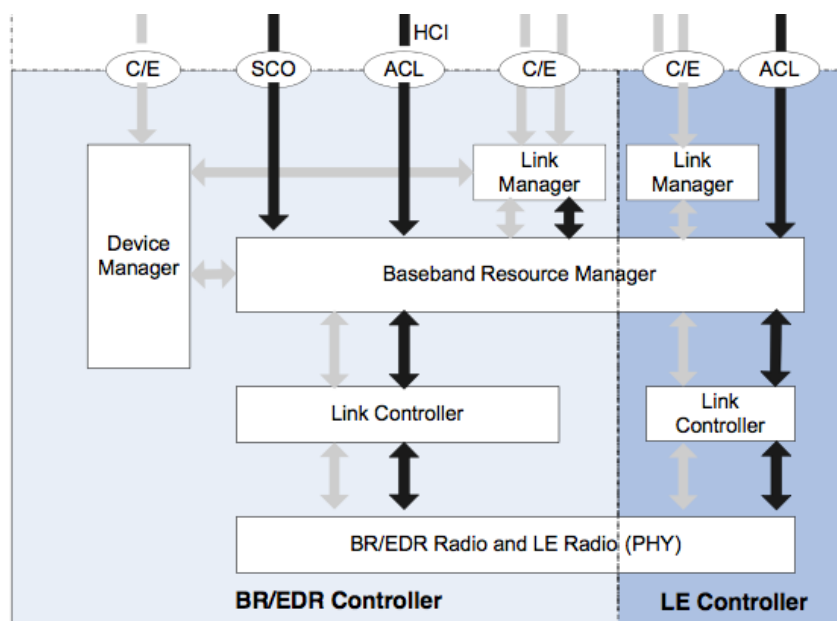


Figure 1-5. Architecture of the Classic BT & BLE controller (from the SIG BT CORE 4.2)

### 1.2.2. BLUEDROID

#### 1.2.2.1. Overall Architecture

在ESP-IDF中，显著修改的BLUEDROID被用作蓝牙主机(经典BT+BLE)。BLUEDROID拥有一套完整的功能，支持大多数常用的标准和架构设计，相对复杂。

In ESP-IDF, the significantly modified BLUEDROID is used as the Bluetooth Host (Classic BT + BLE). The BLUEDROID, which has a complete set of functions and supports most of the commonly-used standards and architectural designs, is relatively complicated. However, the modified BLUEDROID retains most of the codes below the BTA layer and





然而，修改后的BLUEDROID保留了BTA层以下的大部分代码，并使用更精简的BTC层作为内置规范和Misc控制层，几乎完全删除了BTIF层代码。

almost completely deletes the BTIF layer code, using a leaner BTC layer as the built-in specification and Misc control layer. The architecture of the modified BLUEDROID and its relationship with the Controller are shown in the figure below:

修改后的BLUEDROID的架构及其与控制器的关系如下图所示：

Espressif Bluetooth  
Coarse Architecture  
V1.1

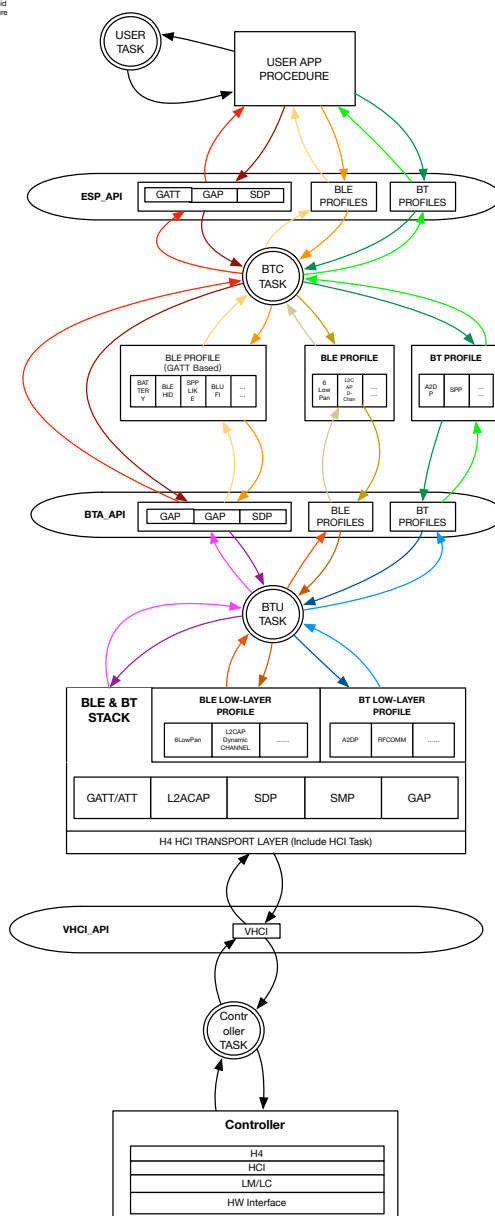


Figure 1-6.ESP32 BLUEDROID diagram

#### Note:

此图主要描述BLUEDROID BLUEDROID的层次结构，而不是详细信息，如HCI任务。关于每一层的详细信息可以在下面看到。  
This diagram mainly describes the hierarchy of the BLUEDROID BLUEDROID, rather than details, such as the HCI TASK. The detailed information about each layer can be seen below.

如上图所示，BLUEDROID主要分为两层，即BTU层和BTC层(除HCI外)。每一层都负责其相应的任务。

As shown in the figure above, the BLUEDROID can be divided into two layers mainly, which are the BTU layer and the BTC layer (except for HCI). Each layer is responsible for its corresponding tasks. More specifically, the BTU layer is mainly responsible for processing the bottom layer protocol stacks of the Bluetooth Host, including L2CAP, GATT/ATT, SMP, 更具体地说，BTU层主要负责处理蓝牙主机的底层协议栈，包括L2CAP、GATT/ATT、SMP、GAP等Profile。



BTU层提供以“bta”为前缀的接口。BTC层主要负责向应用层提供前缀为“esp”的支持接口，处理基于GATT的配置文件和处理各种任务。所有API都位于ESP\_API层。开发者应该使用前缀为“esp”的蓝牙API。

GAP, and other profiles. The BTU layer provides an interface prefixed with "bta". The BTC layer is mainly responsible for providing a supported interface, prefixed with "esp", to the application layer, processing GATT-based profiles and handling miscellaneous tasks. All the APIs are located in the ESP\_API layer. Developers should use the Bluetooth APIs prefixed with "esp".

上图没有详细描述HCI层。事实上，HCI层有两个任务来处理下行和上行数据(在ESP-IDF V2.1和更早版本的设计中)。

The figure above does not describe the HCI layer in detail. In fact, the HCI layer has two tasks which process the Downward and Upward data (in the designs of ESP-IDF V2.1 and older versions).

该架构背后的设计逻辑是通过将蓝牙相关任务移交给BTC层来最小化用户任务的负载并简化蓝牙结构。

The design logic behind this architecture is to minimize the load on the User Tasks and streamline the Bluetooth structure by handing over the Bluetooth-related tasks to the BTC layer.

Due to legacy reasons and actual demand, some of the Classic Bluetooth profiles, such as RFCOMM and A2DP, as well as other lower layer protocols are implemented in the BTU layer, while some of the protocols that are related to procedural controls, or require the ESP-API, are implemented in the BTC layer.

在BTU层实现蓝牙低能量协议的一些规范和低层功能，如6LoWPAN或动态L2CAP通道，从而通过BTC为应用层提供ESP-API。

Some of the profiles and lower layer functions of the Bluetooth Low Energy, such as the 6LowPan or Dynamic L2CAP Channel, will be implemented in the BTU layer, thus providing the application layer with the ESP-API through the BTC.

#### 1.2.2.2. OS-related Adaptation 与操作系统相关的适应

BLUEDROID中与系统相关的一些接口需要OSI适配。涉及的功能包括计时器(Alarm)、任务(Thread)、未来等待/就绪(Semaphore)和分配器/GKI (Malloc/Free)。

Some interfaces that are related with the system in BLUEDROID require OSI adaptation.

The functions involved include Timer (Alarm), Task (Thread), Future Await/Ready (Semaphore), and Allocator/GKI (malloc/free).

BLUEDROID中的FreeRTOS定时器被封装为警报，用于启动定时器，定时器触发某些任务。

The FreeRTOS Timer in BLUEDROID has been packaged as an Alarm, and is used to start the timer which triggers certain tasks.

在BLUEDROID中，POSIX线程被替换为自由实时操作系统任务，并使用自由实时操作系统队列来触发任务(即唤醒)。

In BLUEDROID, the POSIX Thread has been replaced with the FreeRTOS tasks, and uses the FreeRTOS Queue to trigger tasks (i.e. wake up).

In BLUEDROID, the Future Await/Ready function is used to achieve Blocking. Future Lock packages the xSemaphoreTake of FreeRTOS as the future\_await function, and packages the xSemaphoreGive as the future\_ready function. It is worth noting that the future\_await and future\_ready functions cannot be called in the same task context.

In BLUEDROID, malloc/free in the standard library is packaged as the Allocator function that reserves (mallocs) or frees memory. Besides, the GKI function also uses malloc/free as the core function of GKI\_getbuf/GKI\_freebuf.

#### 1.2.3. Bluetooth Directory Introduction 蓝牙目录简介

在ESP-IDF的组件/bt目录中，您可以看到以下子文件夹和子文件：

In the **component/bt** screen of the **ESP-IDF**, you can see the following sub-folders and sub-files:

由于遗留的原因和实际需求，一些经典的蓝牙规范，如RFCOMM和A2DP，以及其他较低层的协议，在BTU层实现，而一些与程序控制相关的协议，或需要ESP-API的协议，在BTC层实现。

在BLUEDROID中，使用Future Await/Ready函数来实现阻塞。Future Lock将FreeRTOS的xSemaphoreTake打包为Future\_await函数，并将xSemaphoreGive打包为Future\_ready函数。值得注意的是，不能在同一任务上下文中调用FUTURE\_AWAIT和FUTURE\_READY函数。

在BLUEDROID中，标准库中的Malloc/Free被打包为保留(Mallocs)或释放内存的Allocator函数。此外，GKI函数还使用Malloc/Free作为gki\_getbuf/gki\_freebuf的核心函数。



```
├─ Kconfig
├─ bluebird
├─ |
├─ |   └─ api
├─ |   └─ bta
├─ |   └─ btc
├─ |   └─ btcore
├─ |   └─ btif
├─ |   └─ device
├─ |   └─ external
├─ |   └─ gki
├─ |   └─ hci
├─ |   └─ include
├─ |   └─ main
├─ |   └─ osi
├─ |   └─ stack
├─ |   └─ utils
├─ |   └─ bt.c
├─ |   └─ component.mk
├─ |   └─ include
├─ |   └─ |
├─ |   └─ |   └─ bt.h
├─ |   └─ lib
├─ |   └─ |
├─ |   └─ |   └─ LICENSE
├─ |   └─ |   └─ README.rst
├─ |   └─ |   └─ libbtm_app.a
```

Figure 1-7. Component/bt in ESP-IDF

The detailed description of each sub-folder and sub-file can be found in the table below:  
[各子文件夹和子文件的详细说明可在下表中找到：](#)

Table 1-1. Description of component/bt in ESP-IDF

Dictionary	Description	Remarks
├─ <i>Kconfig</i>	Menuconfig files	–
├─ <i>bluebird</i>	BLUEDROID home entry	–
└─ <i>api</i>	The API directory, which includes all the APIs (except for those that are related to the Controller)	–
└─ <i>bta</i>	The Bluetooth adaptation layer, which is suitable for the interface of some bottom layer protocols in the host.	–
└─ <i>btc</i>	The Bluetooth control layer, which controls the upper-layer protocols (including profiles) and miscellaneous items in the host.	–
└─ <i>btcore</i>	Some of the original <i>feature/bdaddr</i> conversion functions	To be abandoned
└─ <i>btif</i>	Some call out functions used by the BTA layer	To be abandoned
└─ <i>device</i>	Related to the device control of the Controller, e.g. the basic set of HCI CMD controller processes	–



Dictionary	Description	Remarks
└── <b>external</b>	Codes that are not directly related to the Bluetooth, but are still usable, e.g. the SBC codec software programs	–
└── <b>gki</b>	The management codes that are commonly used by the BLUEDROID memory, e.g. the buffer and queue.	–
└── <b>hci</b>	HCI layer protocols	–
└── <b>include</b>	The top-layer BLUEDROID directory	–
└── <b>main</b>	Main program (mainly to start or halt the process)	–
└── <b>osi</b>	OS interfaces (including semaphore/timer/thread, etc.)	–
└── <b>stack</b>	The bottom layer protocol stacks in the Host (GAP/ATT/GATT/SDP/SMP, etc.)	–
└── <b>utils</b>	Practical utilities	–
└── <b>bt.c</b>	Controller-related processing files	–
└── <b>component.mk</b>	makefile	–
└── <b>include</b>	Controller-related header file directory	–
└── <b>bt.h</b>	Header files that contain the controller-related APIs	–
└── <b>lib</b>	Controller library directory	–
└── <b>LICENSE</b>	License	–
└── <b>README.rst</b>	Readme files	–
└── <b>libbtdm_app.a</b>	Controller library	–



## 2. Classic Bluetooth

This chapter introduces the Classic Bluetooth in ESP-IDF.

### 2.1. Overview

ESP-IDF中的蓝牙主机协议栈起源于BLUEDROID，并已适应于嵌入式应用。在较低层，蓝牙主机堆栈通过虚拟HCI接口与蓝牙双模控制器通信。在上层，蓝牙主机堆栈向用户应用程序提供用于堆栈管理的配置文件和API。

The Bluetooth Host Stack in ESP-IDF originates from BLUEDROID and has been adapted to embedded applications. At the lower layer, the Bluetooth Host Stack communicates with the Bluetooth dual-mode Controller via the virtual HCI interface. At the upper layer, the Bluetooth Host stack provides the profiles and APIs for stack management to the user applications.

协议定义了旨在完成特定功能的消息格式和过程，例如数据传输、链路控制、安全服务和交换。另一方面，蓝牙配置文件定义了蓝牙系统中从PHY到L2CAP的每一层所需的功能和特性，以及核心规范之外的任何其他协议。

Protocols define the message formats and the procedures aimed to accomplish specific functions, e.g. data transportation, link control, security service and service information exchange. Bluetooth profiles, on the other hand, define the functions and features required of each layer in the Bluetooth system, from PHY to L2CAP, and any other protocols outside the core specification.

以下是主机堆栈中当前支持的经典BT配置文件和协议。

Below are the Classic BT profiles and protocols currently supported in the Host Stack.

- Profiles: GAP, A2DP(SNK), AVRCP(CT)
- Protocols: L2CAP, SDP, AVDTP, AVCTP

The protocol model is depicted in Figure 2-1. 协议模型如图2-1所示。

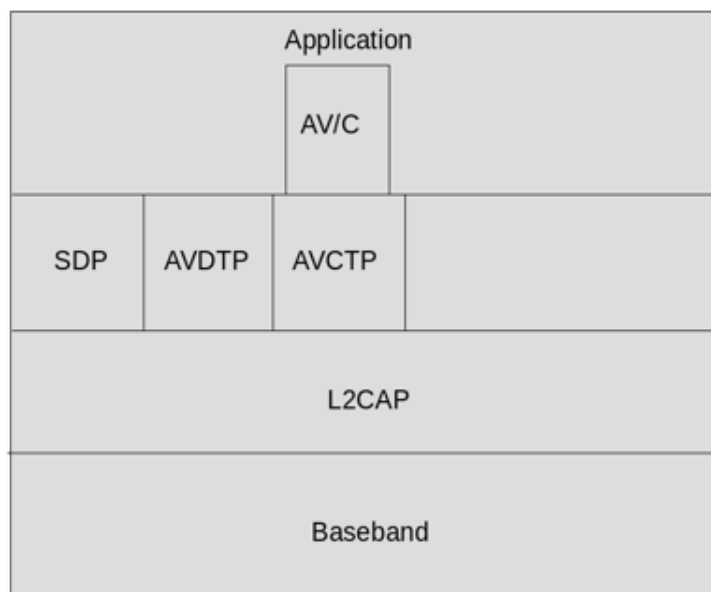


Figure 2-1. Profile Dependencies



在图2-1中，L2CAP和SDP在经典蓝牙的最小主机堆栈中是必需的。AVDTP、AV/C和AVCTP不在核心规范范围内，由特定配置文件使用。

In Figure 2-1, L2CAP and SDP are necessary in a minimal Host Stack for Classic Bluetooth. AVDTP, AV/C and AVCTP are outside the core specification and are used by specific profiles.

### 2.1.1. L2CAP

作为OSI第二层蓝牙协议，蓝牙逻辑链路控制和适配协议(L2CAP)支持更高层的协议复用、分组分割和重组，以及服务信息质量的传递。L2CAP使不同的应用程序可以共享一条ACL-U逻辑链路。应用程序和服务协议使用面向通道的接口与L2CAP对接，以创建到其他设备上的对等实体的连接。

As an OSI layer 2 Bluetooth protocol, the Bluetooth Logical Link Control and Adaptation Protocol (L2CAP) supports higher level protocol multiplexing, packet segmentation and reassembly, as well as the delivery of service information quality. L2CAP makes it possible for different applications to share an ACL-U logical link. Applications and service protocols interface with L2CAP, using a channel-oriented interface, to create connections to equivalent entities on other devices.

L2CAP信道可以通过L2CAP信道配置过程选择的六种模式中的一种模式下操作。操作模式与它们能够提供的服务质量不同，并且在不同的应用条件中使用。这些模式是：

L2CAP channels may operate in one of the six modes selected through the L2CAP channel configuration procedure. The operation modes are distinguished from the QoS that they can provide, and are utilized in different application conditions. These modes are:

- Basic L2CAP Mode
- Flow Control Mode
- Retransmission Mode
- Enhanced Retransmission Mode
- Streaming Mode
- LE Credit-Based Flow Control Mode

对于ACL-U逻辑链路，支持的操作模式为基本L2CAP模式、增强重传模式和流模式。对于其他功能，L2CAP信令信道是支持的固定信道，而帧校验序列(FCS)也是支持的选项。

For ACL-U logical links, the supported operation modes are the Basic L2CAP Mode, Enhanced Retransmission Mode and Streaming Mode. For other features, the L2CAP Signaling channel is the supported fixed channel, while the Frame Check Sequence (FCS) is also a supported option.

### 2.1.2. SDP

服务发现协议(SDP)为应用程序提供了一种发现由对等蓝牙设备提供的服务以及确定可用服务的特征的方法。SDP涉及SDP服务器和SDP客户端之间的通信。服务器维护描述与服务器相关联的服务的特征的服务记录的列表。客户端可以通过发出SDP请求来检索此信息。

The Service Discovery Protocol (SDP) provides a means for applications to discover services offered by a peer Bluetooth device, as well as to determine the characteristics of the available services. The SDP involves communication between an SDP server and an SDP client. A server maintains a list of service records that describe the characteristics of services associated with the server. A client can retrieve this information by issuing an SDP request.

SDP客户端和服务器都实现在主机堆栈中，该模块仅供A2DP、AVRCP等Profile使用，暂时不提供用户应用的API。

Both SDP client and server are implemented in the Host stack, and this module is only used by profiles, such as A2DP and AVRCP, and does not provide APIs for user applications at the moment.

### 2.1.3. GAP

通用访问配置文件(GAP)描述了设备可发现性、连接和安全性方面的模式和程序。

The Generic Access Profile (GAP) provides a description of the modes and procedures in device discoverability, connection and security.



目前，经典蓝牙主机堆栈仅提供有限数量的GAP API。应用程序可以使用这些API，就像它们是可以被对等蓝牙设备发现并连接到的“被动设备”一样。然而，用于发起查询过程的API目前未提供给客户（用户应用程序）。

至于安全方面，IO能力被硬编码为“无输入，无输出”。因此，仅支持安全简单配对中的“Just Works”关联模型。链接密钥的存储在主机中自动完成。

接下来将推出更多针对经典蓝牙的GAP API。更强大、更支持其他关联模式的安全API将在不久的将来提供。用于设备发现和链路策略设置的API也将在稍后阶段提供。

For the time being, only a limited number of GAP APIs are provided to the Classic Bluetooth Host Stack. An application can make use of these APIs, as if they were a "passive device" which could be discovered by and connected to peer Bluetooth devices. However, APIs used for initiating the inquiry procedure are not currently provided to customers (user applications).

As for the security aspect, the IO capability is hard-coded as "No Input, No Output". Therefore, only the "Just Works" association model in the Secure Simple Pairing is supported. The storage of the link key is done automatically in the Host.

More GAP APIs for Classic Bluetooth are coming up next. Security APIs that are more powerful and supportive of other association models will be provided in the near future. APIs for device discovery and link policy settings will also be given at a later stage.

#### 2.1.4. A2DP and AVRCP

高级音频分发配置文件(A2DP)定义了实现在ACL通道上以单声道或立体声形式分发高质量音频内容的协议和程序。A2DP处理音频流，通常与音频/视频远程控制配置文件(AVRCP)一起使用，AVRCP包括音频/视频控制功能。

The Advanced Audio Distribution Profile (A2DP) defines the protocols and procedures that realize the distribution of high-quality audio content in mono or stereo on ACL channels. A2DP handles audio streaming and is often used together with the Audio/Video Remote Control Profile (AVRCP), which includes the audio/video control functions. Figure 2-2 depicts the structure and dependencies of the profiles[1]:

图2-2描述了剖面的结构和相关性[1]：

**Note:**

[1]: Advanced Audio Distribution Profile Specification, Revision 1.3.1.

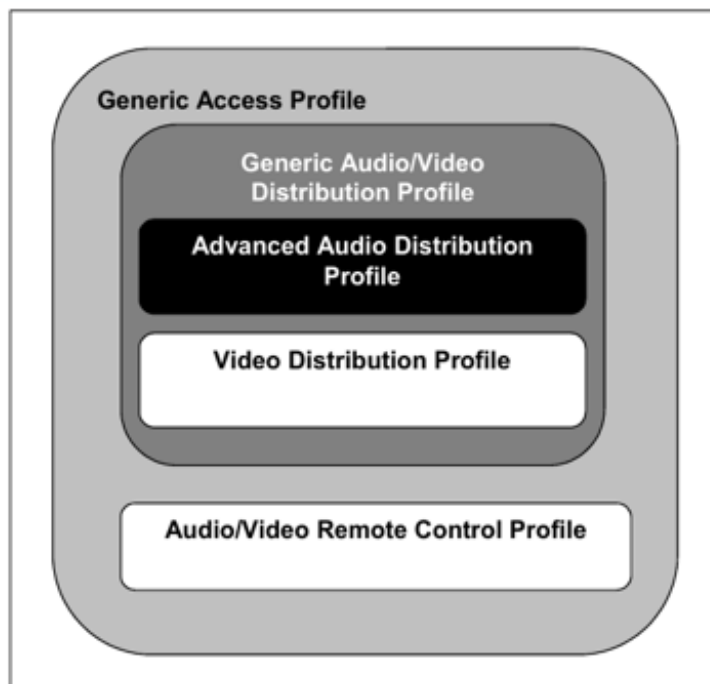


Figure 2-2. Profile Dependencies

As indicated in Figure 2-2, the A2DP is dependent upon the GAP, as well as the Generic Audio/Video Distribution Profile (GAVDP), which defines procedures required to set up an audio/video streaming.



Two roles are defined in A2DP: Source (SRC) and Sink (SNK). SRC functions as a source of a digital audio stream and SNK functions as a sink of a digital audio stream delivered from the SRC.

Two roles defined in AVRCP are Controller (CT) and Target (TG). CT is a device that initiates a transaction by sending a command frame to a target. Examples of CT include personal computers, PDAs and mobile phones. TG is a device that receives a command frame and accordingly generates a response frame. Audio players or headphones are examples of TG. For the time being, A2DP (SRC) and AVRCP (CT) are supported and the device can work as a loudspeaker which can also send remote control messages to the audio source.

In the current A2DP solution, the only audio codec supported is SBC, which is mandated in the A2DP specification. A2DP Version 1.2 and AVDTP Version 1.2 are implemented.

Audio/Video Distribution Transport Protocol (AVDTP) defines the binary transactions between Bluetooth devices for a streaming set-up, and media streaming for audio and video using L2CAP. As the basic transport protocol for A2DP, AVDTP is built upon the L2CAP layer and consists of a signaling entity for negotiating streaming parameters and a transport entity that handles the streaming itself.

The basic service of AVDTP transport capabilities is mandated by the A2DP specification. According to the configuration of current service capabilities, Media Transport and Media Codec in the basic service capability are provided.

AVRCP defines the requirements necessary for the support of the Audio/Video remote control use case.

The commands used in AVRCP fall into three main categories. The first one is the AV/C Digital Interface Command Set, which is applied only on certain occasions and is transported with the Audio/Video Control Transport Protocol (AVCTP). Browsing commands are included in the second category, which provides browsing functionality over another transport channel called the AVCTP browsing channel. The third category, Cover Art Commands, is used to transmit images associated with media items, and is provided through the protocol defined in the Bluetooth Basic Imaging Profile (BIP) with the OBEX protocol.

Two sets of AV/C commands are used in AVRCP. The first one includes the PASS THROUGH command, UNIT INFO command and SUBUNIT INFO command, which are defined in the AV/C specification. The second set includes AVRCP-specific AV/C commands which are defined as a Bluetooth SIG Vendor-Dependent extension. AV/C commands are sent over the AVCTP control channel. A PASS THROUGH command is used to transfer a user operation via a button from the Controller to the panel subunit, which provides a simple and common mechanism to control the target. For example, the operation IDs in PASS THROUGH include common instructions such as Play, Pause, Stop, Volume Up and Volume down.

AVRCP arranges the A/V functions in four categories to ensure interoperability:

- Player/Recorder
- Monitor/Amplifier





- Tuner
- Menu

In the current implementation, AVRCP Version 1.3 and AVCTP Version 1.4 are provided. The default configuration for AVRCP-supported features is Category 2: Monitor/Amplifier. Also, APIs for sending PASS THROUGH commands are provided.

A2DP and AVRCP are often used together. In the current solution, the lower Host stack implements AVDTP and the AVCTP logic, while providing interfaces for A2DP and AVRCP independently. In the upper layer of the stack, however, the two profiles combined make up the "AV" module. The BTA layer, for example, provides a unified "AV" interface, and in BTC layer there is a state machine that handles the events for both profiles. The APIs, however, are provided separately for A2DP and AVRCP.



# 3. Bluetooth Low Energy

This chapter describes the architecture of the Bluetooth Low Energy in ESP32.

## 3.1. GAP

### 3.1.1. Overview

本节主要介绍ESP32中BLE GAP API的实现和使用。GAP（通用访问配置文件）定义了发现过程、设备管理和BLE设备之间设备连接的建立。

BLE GAP以API调用和事件返回的形式实现。协议栈中API调用的处理结果由Events返回。当对等设备发起请求时，该对等设备的状态也由事件返回。

This section mainly introduces the implementation and use of the BLE GAP APIs in ESP32. The GAP (the Generic Access Profile) defines the discovery process, device management and the establishment of device connection between BLE devices.

The BLE GAP is implemented in the form of API calls and Event returns. The processing result of API calls in the protocol stack is returned by Events. When a peer device initiates a request, the status of that peer device is also returned by an Event.

为BLE设备定义了四个GAP角色：

There are four GAP roles defined for a BLE device:

- 广播员：**广播员是一种发送广告包的设备，因此可以被观察者发现。此设备只能播发，但无法连接。  
• Broadcaster: A broadcaster is a device that sends advertising packets, so it can be discovered by the observers. This device can only advertise, but cannot be connected.
- 观察者：**观察者是一种扫描广播者并将这些信息报告给应用程序的设备。此设备只能发送扫描请求，但无法连接。  
• Observer: An observer is a device that scans for broadcasters and reports this information to an application. This device can only send scan requests, but cannot be connected.
- 外围设备：**外围设备是一种通过使用可连接的广告包进行广告的设备，一旦连接就成为从设备。  
• Peripheral: A peripheral is a device that advertises by using connectable advertising packets and becomes a slave once it gets connected.
- 中央：**中央是一种设备，它启动与外围设备的连接，并在建立物理链路后成为主设备。  
• Central: A central is a device that initiates connections to peripherals and becomes a master once a physical link is established.



### 3.1.2. Status Transitions among GAP Roles

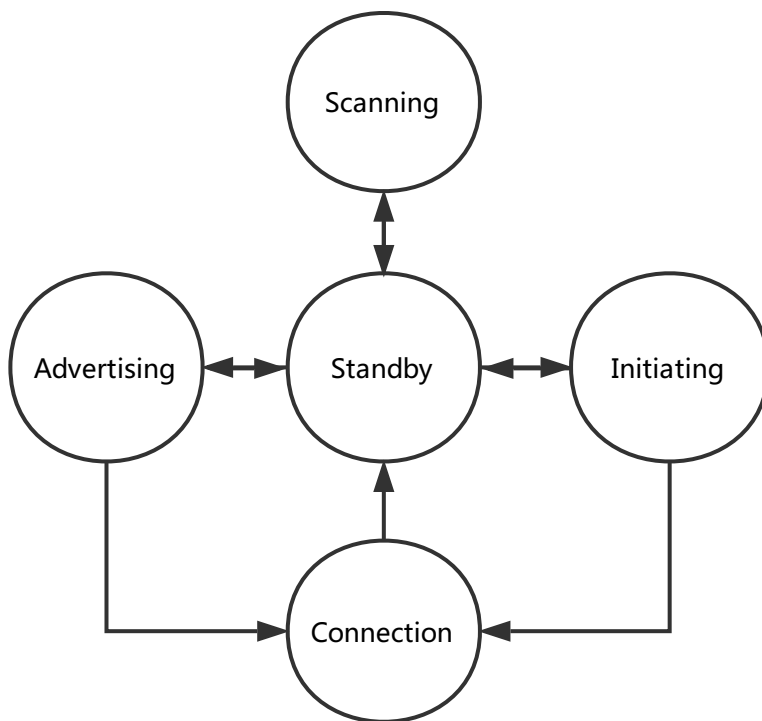


Figure 3-1.The status transitions among GAP roles



### 3.1.3. BLE Broadcast Procedure

使用公共广播进行广播

#### 3.1.3.1. Broadcast using a public address

当公共地址用于广播时，esp\_ble\_adv\_params\_t的own\_addr\_type必须设置为ble\_addr\_type\_public。广播流程图如下：

When a public address is used for broadcasting, the own\_addr\_type of

esp\_ble\_adv\_params\_t must be set to BLE\_ADDR\_TYPE\_PUBLIC. The broadcast flowchart is as follows:

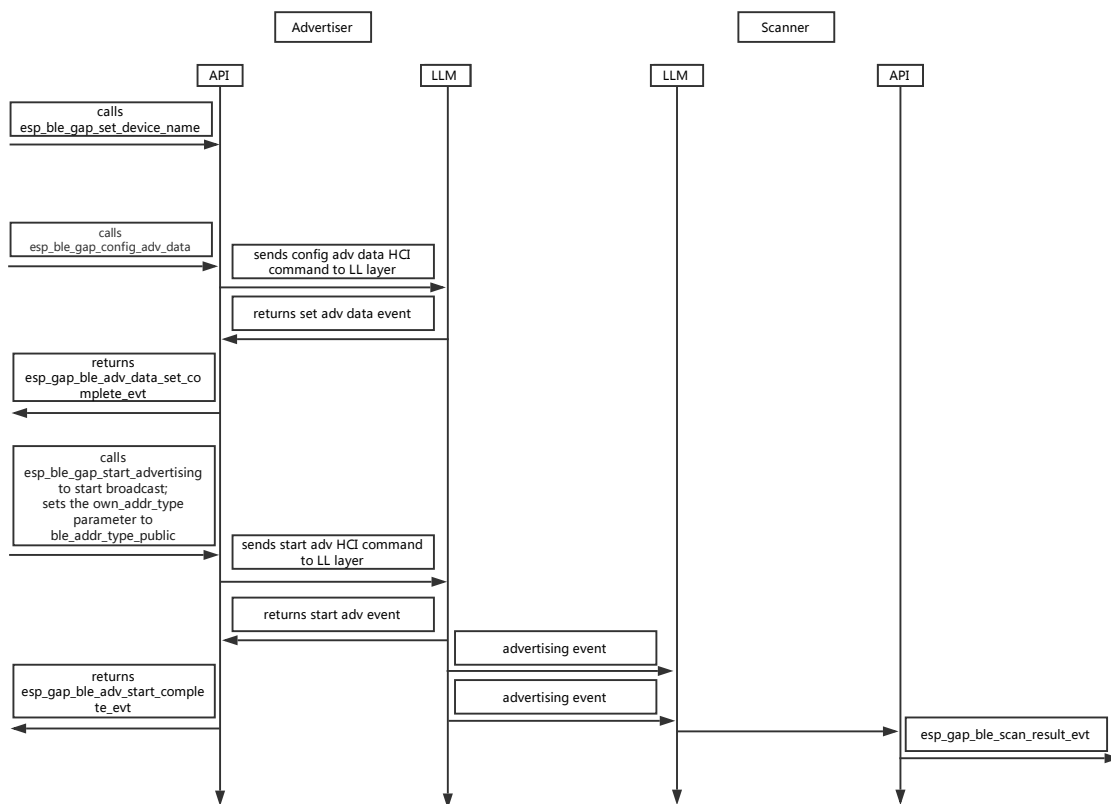


Figure 3-2.Broadcast using a public address



使用可解析地址广播

### 3.1.3.2. Broadcast using a resolvable address

当可解析地址用于广播时，底层协议栈每15分钟更新一次广播地址，并且esp\_ble\_adv\_params\_t的own\_addr\_type必须设置为ble\_addr\_type\_RANDOM。广播流程图如下：

When a resolvable address is used for broadcasting, the underlying protocol stack updates the broadcast address every 15 minutes, and the own\_addr\_type of esp\_ble\_adv\_params\_t must be set to BLE\_ADDR\_TYPE\_RANDOM. The broadcast flowchart is as follows:

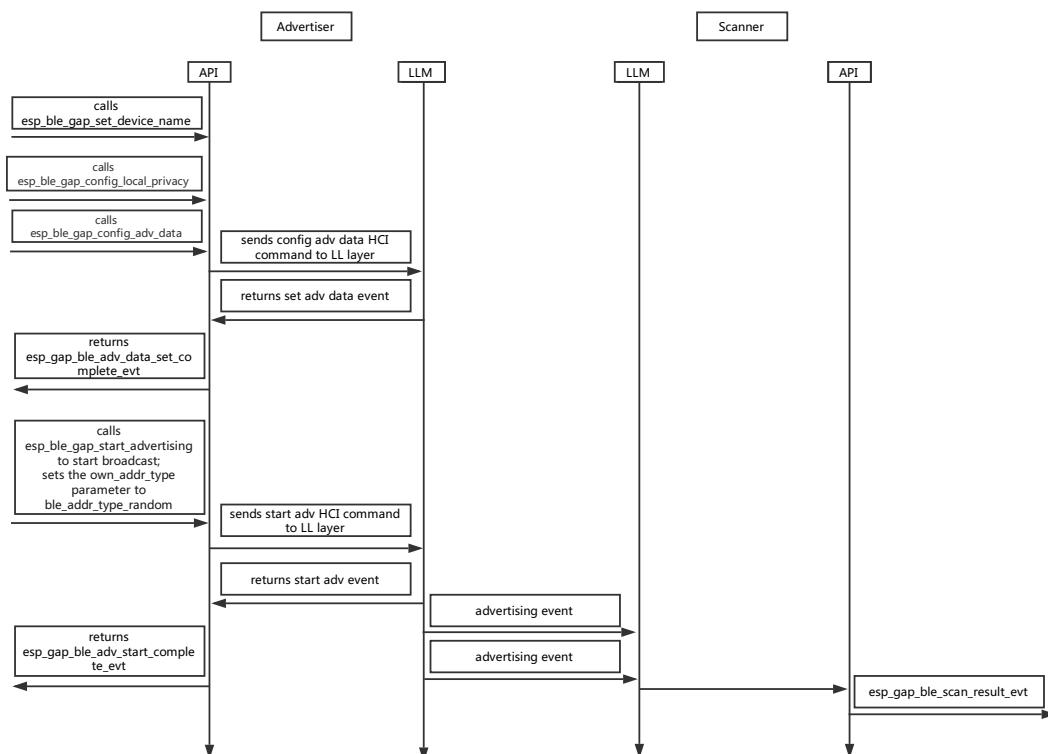


Figure 3-3.Broadcast using a resolvable address

**Note:**

When a resolvable address is used for broadcasting, the broadcast only starts after the esp\_ble\_gap\_config\_local\_privacy event is returned, and the own\_addr\_type type, a broadcast parameter, must be set to BLE\_ADDR\_TYPE\_RANDOM.

当可解析地址用于广播时，广播仅在返回esp\_ble\_gap\_config\_local\_privacy事件后开始，并且own\_addr\_type类型（广播参数）必须设置为ble\_addr\_type\_RANDOM。



## 使用静态随机地址广播

## 3.1.3.3. Broadcast using a static random address

When a static random address is used for broadcasting, the `own_addr_type` of the `esp_ble_adv_params_t` must be set to `BLE_ADDR_TYPE_RANDOM`, which is similar to the case of broadcasting using a resolvable address. The broadcast flowchart is as follows:

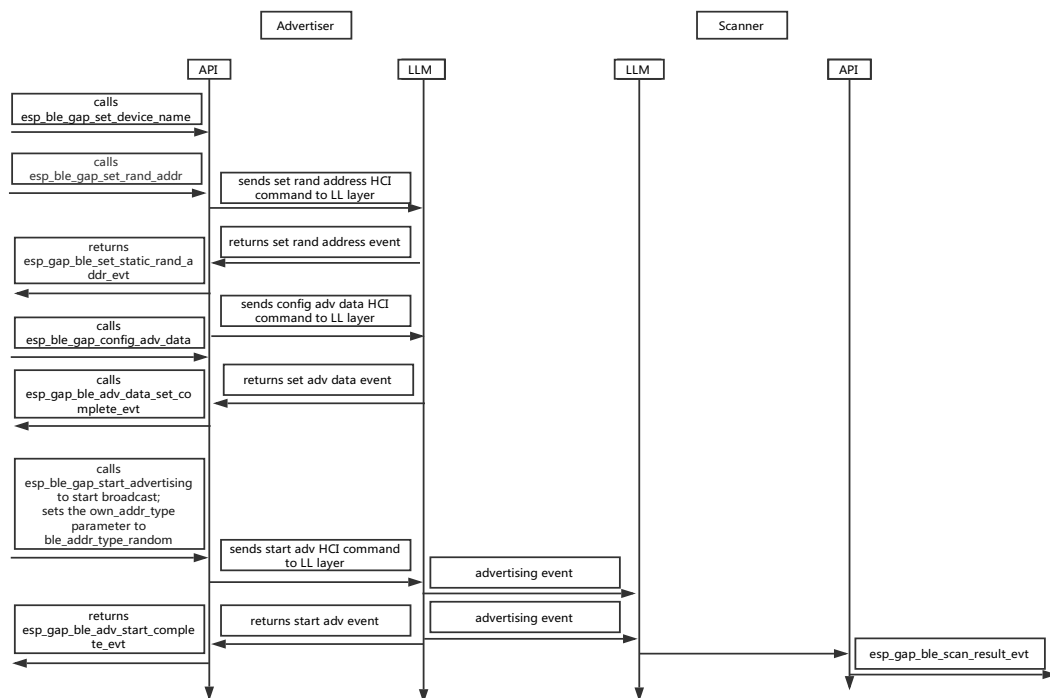


Figure 3-4. Broadcast using a static random address



3.1.4. BLE Modes

BLE广播定义了五种模式：可连接可扫描无方向模式、高占空比定向模式、可扫描无定向模式、不可连接无方向模式和可连接低占空比定向模式。  
Five modes are defined for the BLE broadcasts: Connectable Scannable Undirected mode, High Duty Cycle Directed mode, Scannable Undirected mode, Non-connectable Undirected mode, and Connectable Low Duty Cycle Directed mode.

3.1.4.1. Connectable Scannable Undirected Mode 可连接可扫描无方向模式

Table 3-1.Packet structure

Payload	
AdvA (6 octets)	AdvData (0~31 octets)

处于可连接可扫描无方向模式的设备可以被任何设备发现并连接到任何设备。可扫描性表示当对等设备发送扫描请求时，本地设备需要回复扫描响应。

A device in the Connectable Scannable Undirected mode can be discovered by and connected to any device. Scannability indicates that the local device needs to reply with a scan response, when a peer device sends a scan request.

如上表所示，可连接可扫描无方向广播包包括6字节的广播地址和0~31字节的广播包数据。当静态随机地址用于广播时，通过调用 esp\_ble\_gap\_set\_rand\_addr 指定广播地址。当公共地址或可解析地址用于广播时，广播地址由协议栈自动生成。

As shown in the table above, a Connectable Scannable Undirected broadcast packet includes 6 bytes of the broadcast address and 0 ~ 31 bytes of the broadcast packet data. When a static random address is used for broadcasting, the broadcast address is specified by calling esp\_ble\_gap\_set\_rand\_addr. When a public address or a resolvable address is used for broadcasting, the broadcast address is generated automatically by the protocol stack.

3.1.4.2. High Duty Cycle Directed Mode and Connectable Low Duty Cycle Directed Mode

高占空比定向模式和可连接低占空比定向模式

Table 3-2.Packet structure

Payload	
AdvA (6 octets)	InitA (6 octets)

IP定向广播只能由指定设备发现并连接到指定设备。  
The IP directed broadcasts can only be discovered by and connected to the designated devices.

如上表所示，高占空比定向广播分组包括广播设备地址的6个字节和接收设备地址的六个字节。在此模式下，广播参数adv\_int\_min和adv\_int\_max被忽略。

As shown in the table above, the High Duty Cycle Directed Broadcast packet includes 6 bytes of the broadcasting device's address and 6 bytes of the receiving device's address. In this mode, the broadcast parameters adv\_int\_min and adv\_int\_max are ignored.

在可连接低占空比定向模式下，广播参数adv\_int\_min和adv\_int\_max必须大于100 ms (0xA0)。  
In the Connectable Low Duty Cycle Directed mode, the broadcast parameters adv\_int\_min and adv\_int\_max must be greater than 100 ms (0xA0).

Note:

IP directed broadcasts do not carry Adv Data. IP定向广播不携带高级数据。

3.1.4.3. Scannable Undirected Mode 可扫描无方向模式

In the Scannable Undirected mode, a device can be discovered by any other device, but it cannot get connected to it.  
在“可扫描无方向”模式下，任何其他设备都可以发现设备，但无法连接到该设备。



Table 3-3.Packet structure

Payload	
AdvA (6 octets)	AdvData (0~31 octets)

如上表所示，可扫描未定向数据包包括6字节的广播地址和0~31字节的广播数据包数据，这与可连接可扫描未方向数据包的结构相同。此模式下的设备只能由任何设备扫描，但无法连接到该设备。

As shown in the table above, a Scannable Undirected packet includes 6 bytes of a broadcast address and 0~31 bytes of the broadcast packet data, which is the same structure as in the Connectable Scannable Undirected packet. A device in this mode can only be scanned by any device, but it cannot be connected to it.

#### 3.1.4.4. Non-connectable Undirected Mode 不可连接无方向模式

在不可连接无方向模式下，任何设备都可以发现设备，但它既不能被其他设备扫描，也不能连接到其他设备。当对等设备发送扫描请求时，不可扫描的设备不会回复扫描响应。可断开连接的设备是不能连接到任何设备的设备。

In the Non-connectable Undirected mode, a device can be discovered by any device, but it can neither be scanned by, nor connected to any other devices. An unscannable device is one that will not respond to any other device's scan request. A disconnectable device is one that cannot be connected to any device.

在不可连接无方向模式下，任何设备都可以发现设备，但它既不能被其他设备扫描，也不能连接到其他设备。当对等设备发送扫描请求时，不可扫描的设备不会回复扫描响应。可断开连接的设备是不能连接到任何设备的设备。

Table 3-4.Packet structure

Payload	
AdvA (6 octets)	AdvData (0~31 octets)

如上表所示，不可连接的无方向广播分组还包括6字节的广播地址和0~31字节的广播分组数据。在此模式下，设备可以被发现，但不能被扫描，也不能被其他设备连接。

As shown in the table above, a Non-connectable Undirected broadcast packet also includes 6 bytes of broadcast address and 0~31 bytes of broadcast packet data. In this mode, a device can be discovered but cannot be scanned nor be connected by other devices.

#### 3.1.5. BLE Broadcast Filtering Policy BLE广播过滤策略

在ESP32的BLE架构中，广播过滤策略通过设置adv\_filter\_policy枚举类型来实现，该枚举类型具有以下四个值：

In ESP32's BLE architecture, the broadcast filtering policy is implemented by setting the adv\_filter\_policy enumeration type, which has the following four values:

- ADV\_FILTER\_ALLOW\_SCAN\_ANY\_CON\_ANY
- ADV\_FILTER\_ALLOW\_SCAN\_WLST\_CON\_ANY
- ADV\_FILTER\_ALLOW\_SCAN\_ANY\_CON\_WLST
- ADV\_FILTER\_ALLOW\_SCAN\_WLST\_CON\_WLST

The four values correspond to 4 cases, respectively:

- 可以扫描并连接到任何设备（无白名单）  
Can be scanned and connected to any device (no white list)
- 处理所有连接请求，仅处理白名单中的扫描请求  
Handles all of the connection requests, and only the scan requests in the whitelist
- 处理所有扫描请求，仅处理白名单中的连接请求  
Handles all of the scan requests, and only the connection requests in the whitelist
- 处理白名单中的连接请求和扫描请求  
Handles the connection requests and scan requests in the whitelist

#### 3.1.6. BLE Scanning Procedure BLE扫描程序

在ESP32中，扫描设备主要通过调用esp\_ble\_gap\_set\_scan\_params来设置扫描参数，然后通过调用esp\_ble\_gap\_start\_scanning

In ESP32, the scanning device sets the parameters of the scan mainly by calling esp\_ble\_gap\_set\_scan\_params, and then starts the scan by calling





扫描设备的信息将由ESP\_GAP\_BLE\_SCAN\_RESULT\_EVT事件返回，或在持续时间超时由ESP\_GAP\_SEARCH\_INQ\_CMPL\_EVT事件返回。

esp\_ble\_gap\_start\_scanning. The information of the scanned device will be returned by the ESP\_GAP\_BLE\_SCAN\_RESULT\_EVT event, or by the ESP\_GAP\_SEARCH\_INQ\_CMPL\_EVT event when the duration times out.

**Notice:**

当持续时间的值为0时，设备将被永久扫描而不会超时。

When the value of the duration is 0, the device will be scanned permanently without timeout.

### 3.1.7. BLE GAP Implementation Mechanism BLE GAP实施机制

ESP32调用BLE\_GAP API，注册BLE\_GAP回调，并通过事件的返回值获得当前设备的状态。

ESP32 calls the BLE GAP APIs, registers BLE GAP Callback and obtains the status of the current device by the returned value of the Event.

## 3.2. GATT

### 3.2.1. ATT

BLE架构内的数据以属性的形式存在，由四个基本元素组成：

The data inside the BLE architecture exists in the form of Attributes that consist of four basic elements:

**属性句柄：**属性句柄可以帮助我们定位任何属性，这类似于使用地址定位内存中的数据。例如，第一个属性的句柄为0x0001，第二个属性的Handle为0x0002，以此类推，直到0xFFFF。

- Attribute Handle: an Attribute Handle can help us locate any Attribute, which is similar to using an address to locate data in the memory. For example, the Handle of the first attribute is 0x0001 and the second one is 0x0002, and so on, up to 0xFFFF.

**属性类型：**每个数据集都会公开特定类型的信息，如温度、发射功率、电池电量等。公开的数据类型称为属性类型。鉴于可以公开的数据类型不同，使用16位或128位数字（也称为UUID）来标识属性的类型。例如，UUID 0x2A09表示电池电量，UUID 0x2A6E表示温度。

- Attribute Type: Each data set exposes a certain type of information, such as temperature, transmit power, battery level and so on. The type of the data that is exposed is called *attribute type*. Given the different possible types of data that can be exposed, a 16-bit or 128-bit number, also known as UUID, is used to identify the type of the attribute. For example, UUID 0x2A09 is for Battery Level and UUID 0x2A6E is for Temperature.

**属性值：**属性值是每个属性的关键信息，同时添加了其他三个元素（句柄、类型和权限），以便更好地理解属性值。不同属性类型的属性值的长度可以是固定的，也可以是可变的。例如，“电池级别”属性值的长度仅为1字节，足以覆盖“电池级别属性”的所有可能值，即0-100，而启用BLE的直通模块的属性长度是可变的。

- Attribute value: the attribute value is the key information of each attribute, while the other three elements (handle, type and permission) are added so the attribute value can be better understood. The length of the attribute value for different attribute types can be fixed or variable. For example, the length of the attribute value in Battery Level is only 1 byte, which is enough to cover all the possible values of a Battery Level attribute, i.e. 0-100, while the attribute length of a BLE-enabled pass-through module is variable.

**属性权限：**每个属性都包含只能读取或写入的信息。为了方便这些访问限制，每个属性都有自己的属性权限。拥有数据的一方可以通过属性权限控制其本地数据的读/写访问。

- Attribute Permission: Each attribute contains information that can only be read or written. To facilitate these restrictions upon access, each and every attribute has its own attribute permissions. The party that owns the data can control the read/write access of its local data through the attribute permissions.

Table 3-5. Attribute Structure Table

Attribute Handle	Attribute Type	Attribute Value	Attribute Permission
0x0001	-	-	-
0x0002	-	-	-



Attribute Handle	Attribute Type	Attribute Value	Attribute Permission
.....	-	-	-
0xFFFE	-	-	-
0xFFFF	-	-	-

保存数据（即属性）的设备被定义为服务器，而从服务器获取数据的设备则被定义为客户端。服务器和客户端之间的常见操作如下所示：

The device that holds the data (i.e. the attributes) is defined as a server, and the device that obtains the data from the server is defined as a client. The common operations between a server and a client are listed below:

客户端通过将数据写入服务器来向服务器发送数据。写入请求和写入命令都可以用于写入属性值。但是，只有在使用写入请求时才会提示写入响应。

- **A client sends data to a server** by writing data into the server. Both the Write Request and Write Command can be used to write an attribute value. However, a Write Response is only prompted when a Write Request is used.

服务器通过向客户端发送指示或通知来向客户端发送数据。指示和通知之间的唯一区别是，只有在使用指示时才会提示确认。这类似于写请求和写命令之间的区别。

- **A server sends data to a client** by sending an Indication or Notification to the client. The only difference between an Indication and a Notification is that a Confirmation is prompted only when an Indication is used. This is similar to the difference between a Write Request and a Write Command.
- **A client can also obtain data from the server by initiating a Read Request.**  
客户端还可以通过发起读取请求来从服务器获取数据。

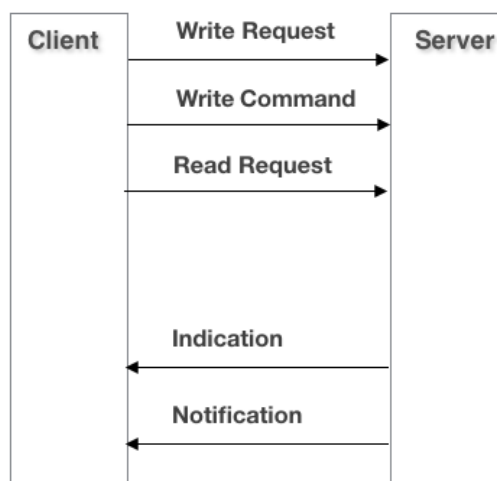


Figure 3-5. Common operations between a server and a client

**Notice:**

有关服务器和客户端之间常见操作的详细信息，请参阅Core\_V5.0，第3卷，F部分，第3.4章“属性协议PDU”。  
For detailed information regarding the common operations between a server and a client, please refer to Core\_V5.0, Vol3, Part F, Chapter 3.4 “Attribute Protocol PDUs”.

The common operations between a server and a client are achieved by using ATT PDU. Each device can specify its supported MTU, which is the maximum length of an ATT message. In ESP32 IDF, the MTU can be 23 - 517 bytes, and the total length of an attribute value is not limited.

服务器和客户端之间的通用操作通过使用ATT PDU来实现。每个设备都可以指定其支持的MTU，即ATT消息的最大长度。在ESP32 IDF中，MTU可以是23-517字节，属性值的总长度不受限制。



如果用户数据包的长度大于 (MTU-3)，则需要准备写入请求来写入数据。类似地，如果分组的长度大于 (MTU-1)，则需要读取Blob请求来读取剩余数据。

If the length of the user's packet is greater than (MTU-3), a Prepare Write Request is required to write data. Similarly, if the length of the packet is greater than (MTU-1), a Read Blob Request is required to read the remaining data.

此处应突出显示单个物理分组的MTU和LE分组长度之间的差异。MTU (在本例中为主机ATT层) 确定是否所有“要发送的数据”都可以装入单个ATT请求，以及是否需要准备写入请求来发送数据。同时，关于PHY层的LE分组长度确定分组是否需要在几个分组中进行划分和传输。例如，如果 (MTU+4) 大于LE分组长度，则需要将该ATT分组分成多个物理分组进行传输。相反，如果 (MTU+4) 短于LE分组长度，则只需要一个物理分组来发送整个ATT分组。我们向MTU添加4字节的原因是因为在传输期间将添加4字节L2CAP头。

#### \*Notice:

The difference between the MTU and the LE Packet Length of a single physical packet should be highlighted here. The MTU, which is about the Host ATT layer in this case, determines whether all the “data to be sent” can fit into a single ATT Request and also whether a Prepare Write Request is required to send the data. Meanwhile, the LE Packet Length, which is about the PHY layer, determines whether the packet needs to be divided and transmitted in several packages. For example, if (MTU + 4) is greater than the LE Packet Length, this ATT packet needs to be divided and transmitted in more than one physical packets. On the contrary, if (MTU+4) is shorter than the LE Packet Length, only one physical packet is required to send the whole ATT packet. The reason we add 4 bytes to the MTU is because a 4-byte L2CAP header will be added during the transmission.

### 3.2.2. GATT Profile

ATT指定了BLE架构中的最小数据存储单元，而GATT定义了如何使用属性值和描述符表示数据集，如何将类似数据聚合到服务中，以及如何找出对等设备拥有的服务和数据。

The ATT specifies the minimum data storage unit in the BLE architecture, while the GATT defines how to represent a data set using attribute values and descriptors, how to aggregate similar data into a service, and how to find out what services and data a peer device owns.

《关贸总协定》引入了特征的概念，该概念涉及的信息并非纯粹的数字信息，如下文所述：

The GATT introduces the concept of Characteristics, which are about information that is not purely numerical, as in the cases outlined below:

给定值的单位，例如，重量以千克 (kg) 为单位，温度以摄氏度 ( ) 为单位等等。

- The unit of a given value, for example, weight measured in kilograms (kg), temperature measured in Celsius (°C), and so on.

给定值的名称。例如，对于具有相同UUID的特性，例如。温度属性，该值的名称通知对等设备该值表示“主卧室的温度”，而另一个值表示“客厅的温度”。

- The name of a given value. For example, for characteristics with the same UUID, e.g. temperature attribute, the name of the value informs the peer device that this value indicates “the temperature in the master bedroom”, while the other one indicates “the temperature in the living room”.

过量数据的指数，如230000和460000。鉴于指数已指定为 $10^4$ ，仅传输“23”和“46”就足以表示230000和46万。

- The exponent of excessive data numbers, such as 230,000 and 460,000. Given that the exponent is already specified as  $10^4$ , transmitting only “23” and “46” is enough to represent 230,000 and 460,000.

这些只是在实际应用中准确描述数据的许多现有需求的几个例子。为了提供更细微的信息，应该保留大量的数据空间来存储每个特征中的附加信息。然而，在许多情况下，保留的大部分额外空间将不会被使用。因此，这样的设计将不符合BLE的先决条件，即协议尽可能简洁。在这种情况下，GATT规范引入了描述符的概念，以概述这些附加信息。必须注意的是，每条数据和描述符并没有一一对应的关系，也就是说，复杂的数据可以有多个描述符，而简单的数据可以完全没有描述符。

These are just a few examples of the many existing requirements for describing data accurately in actual applications. In order to provide more nuanced information, a large piece of data space should be reserved to store this additional information in each characteristic. However, in many cases, most of the extra space reserved will not be used. Such a design, then, will not comply with BLE's prerequisite to have as concise as possible protocols. In cases like this, the GATT specification introduces the concept of descriptors to outline this additional information. It must be noted that each piece of data and descriptor do not have a one-to-one correspondence, that is, complex data can have multiple descriptors, while simple data can have no descriptors at all.

特性由三个基本元素组成：

A characteristic is composed of three basic elements:

- Characteristic Declaration: a declaration is the start of a characteristic, informing a peer device that the content following the declaration is the characteristic value. All

特征声明：声明是特征的开始，通知对等设备声明后的内容是特征值。两个声明之间的所有句柄构成了一个完整的特性。写入和读取属性也包含在声明中。



the handles between two declarations compose a complete characteristic. The write and read properties are also included in a declaration.

特征值：特征值是特征的主要部分，它承载了特征的最重要信息。

- Characteristic Value: a characteristic value is the main part of a characteristic, which carries the most important information of a characteristic.  
描述符：描述符可以进一步描述特征（例如提供配置信息），一个特征可以有多个或没有描述符。
- Descriptor: Descriptors can further describe characteristics (e.g. providing configuration information) and a characteristic can have multiple or no descriptors.

在BLE中，GAP以服务的形式将类似的功能分组在一起。例如，与电池相关的所有特性和行为都可以定义为电池服务；与心率相关的所有特征和行为可以被定义为心率服务；并且与体重秤相关的所有特征和行为都可以被定义为体重秤服务。

In BLE, the GAP groups similar functions together in the form of Services. For example, all of the characteristics and behaviors related to the battery can be defined as a Battery Service; all of the characteristics and behaviors related to the heart rate can be defined as a Heart Rate Service; and all of the characteristics and behaviors related to the weight scale can be defined as a Weight Scale Service.

服务通常包括一个或多个特征，每个特征包括零或多个描述符。用户可以根据自己的应用程序要求选择所需的服务，并形成最终应用程序。A Service typically includes one or more characteristics and each characteristic includes zero or many descriptors. Users can select the required services based on their own application requirements, and form the final application.

A completed service definition table is shown below: 完整的服务定义表如下所示：

Table 3-6. The definition table of services

Attribute Handle	Attribute Type
0x0001	Service 1
0x0002	Characteristic Declaration 1
0x0003	Characteristic Value 1
0x0004	Descriptor 1
0x0005	Characteristic Declaration 2
0x0006	Characteristic Value 2
0x0007	Descriptor 2
0x0008	Descriptor 3
0x0009	Service 2
.....	.....



**\*Notices:**

For other definitions of services, characteristics and descriptors, please refer to:

- Chapter 3 “Service Interoperability Requirements”, Part G, Vol3., Core\_V5.0;
- <https://www.bluetooth.com/zh-cn/specifications/gatt>

**3.2.3. Add Gatt Services in ESP32 IDF Environment** [在ESP32 IDF环境中添加Gatt服务](#)

在ESP32 IDF 1.0版中，用户可以通过事件一个接一个地手动添加服务和特性。所有读写操作都将通过事件到达应用程序层，用户可以使用包对其进行响应。对于不熟悉BLE协议的用户，这种方法很容易出错，尤其是在大型GATT数据库中添加服务时。因此，不建议逐个手动添加服务。



**\*Notice:**

ESP IDF中仍保留手动添加服务和特性的界面和示例。有关更多信息，请参阅gatt\_service示例。  
The interface and examples of adding services and characteristics manually are still reserved in ESP IDF.  
For more information, please refer to the gatt\_service example.

在这种情况下，ESP32 IDF 2.0版引入了使用属性表添加服务和特性。用户只需在属性表中输入新的服务和特性，然后调用esp\_ble\_gatts\_create\_attr\_tab，就可以添加新的服务或特性。此外，在这种情况下还支持较低层的响应，这意味着较低层能够响应某些请求并自动识别错误，因此用户可以专注于接收和发送数据。

In this context, adding services and characteristics with an attribute table is introduced in ESP32 IDF Release 2.0. Users can add new services and characteristics by simply entering them in an attribute table and, then, calling esp\_ble\_gatts\_create\_attr\_tab. Additionally, a lower layer response is also supported in this case, meaning the lower layer is able to respond to some requests and identify errors automatically, so the users can focus on receiving and sending data.

这样，用户可以轻松地将配置文件从其他平台移植到ESP32平台，而无需重新实现BLE规范。  
In this way, users can port profiles to the ESP32 platform from other platforms easily, without the need to implement the BLE specifications all over again.



**\*Notice:**

We recommend that users add services and characteristics through the attribute table, which is much easier, less error-prone, and supports low-layer responses. For details, please refer to the gatt\_server\_service\_table examples.

属性表的结构定义了需要初始化才能通过esp\_gatts\_attr\_db\_t描述属性的所有参数：  
The structure of an attribute table defines all of the parameters that require initialization to describe an attribute through esp\_gatts\_attr\_db\_t:

Table 3-7. The structure parameters of ESP32 IDF

Parameter	Description
uint8_t attr_control	Defines some responses, such as the write_response, which are given by the lower layer automatically or passed to the application layer, so that users can respond manually. The ESP_GATT_AUTO_RSP automatic response mode is recommended.
uint16_t uuid_length	Indicates that the length of UUID is 16 bits, 32 bits or 128 bits. Since the attribute UUID is transmitted by pointers, the length of the UUID has to be specified.



Parameter	Description
uint8_t *uuid_p	Indicates the pointer of the UUID of the current attribute. Users can read a certain length of the UUID value, based on the length information specified in the uuid_lenght parameter.
uint16_t perm	Indicates the write and read permissions of the current attribute. This parameter is bitwise-operated. Each bit represents a specific write and read permission. Operating a certain bit can change the write and read permission of the corresponding attribute. For example, PERM_READ   PERM_WRITE means an attribute that can be read and written.
uint16_t max_length	Indicates the maximum length of the current attribute value. The protocol stack allocates memory to the attribute, based on this parameter. If the length of the attribute value that a peer device has written exceeds the maximum length defined in this parameter, a write error is returned, indicating that the error is due to the length of the write operation exceeding the maximum length of the data.
uint16_t length	Indicates the actual length of the current attribute. For example, in case the maximum length of the attribute is 512 bits and a peer device wants to write "0x1122" into this attribute, we will set the current length of the attribute to 2. When a peer device, then, reads this attribute, we can obtain the actual length of this attribute from the memory, sending only the part with the actual values, instead of the whole 512 bits.
uint8_t *value	Indicates the initialized values of the current attribute value. Since the format of this parameter is a pointer, the actual length of this parameter should be obtained first by the length parameter, in order to get the correct value from the pointer.

在ESP32 IDF ( GATT客户端 ) 中发现对等设备的服务

### 3.2.4. Discover a Peer Device's Services in ESP32 IDF (GATT Client)

发现服务可以帮助GATT客户端发现对等设备的服务和特性。不同设备的发现过程可能不同。本文介绍了ESP32 IDF的发现过程，以及如何发现对等设备的GATT服务的示例。

The Discovering Service can help a GATT Client to discover a peer device's services and characteristics. The discovery procedure can be different for different devices. The discovery procedure of ESP32 IDF is introduced here, along with an example of how to discover a peer device's GATT service.

首先，发现所有对等设备的服务信息，包括服务UUID和属性句柄的范围。

- Firstly, discover all of the peer devices' services information, including the service UUID and the range of the attribute handle.

- GATT Service, UUID 0x1801, Handles 0x0001~0x0005

- GAP Service, UUID 0x1800, Handles 0x0014~0x001C

然后，发现GATT服务句柄范围内的所有对等设备的特性 ( 0x0001~0x0005 )。

- Then, discover all of the peer devices' characteristics within the handle range of a GATT service (0x0001~0x0005).

查找“服务更改特征”，

- Find “Service Change Characteristic”, Handles 0x0002~0x0003

0x0002表示特性声明

- 0x0002 represents the characteristic declaration

0x0003表示特征值

- 0x0003 represents the characteristic value

因此，每个特征都具有至少两个句柄的属性。

- So each characteristic has the attributes of at leasts two handles.





例如，假设GATT服务的句柄在0x0001~0x0005的范围内，则0x0003之后应跟随相应的描述符。那么，所有的描述符都应该是soug

- Given that the handles of a GATT Service lie in the range of 0x0001~0x0005, 0x0003, for example, should be followed by the corresponding descriptor. All descriptors, then, should be sought from 0x0004 onwards.
  - 0x0004表示客户端特征配置的描述符
  - 0x0004 represents the descriptor of the Client Characteristic Configuration
  - 0x0005目前没有任何信息，可能是为此服务保留的句柄。
  - 0x0005 does not have any information at the moment, and could be a handle reserved for this service.
- 至此，GATT服务的发现过程完成。
- At this point, the discovery procedure of GATT Service is complete.

### 3.3. SMP

本章主要介绍ESP32 BLE SMP（安全管理协议）的实现和使用。

This chapter mainly introduces the implementation and use of the ESP32 BLE SMP (Security Management Protocol).

#### 3.3.1. Overview

SMP相关API已打包在ESP32 BLE的GAP模块中。

The SMP-related APIs have been packaged in ESP32 BLE's GAP module.

SMP生成加密密钥和身份密钥，定义用于配对和密钥分发的方便协议，并允许协议栈中的其他层安全地与其他设备连接和交换数据。在此过程中需要数据链路层中的连接和某些安全标准。GAP SMP允许两个设备通过设置蓝牙核心规范版本的SMP章节中的安全级别，在数据链路层中加密其连接。在介绍GAP SMP的实施之前，我们应该澄清以下概念：

The SMP generates encryption keys and identity keys, defines a convenient protocol for pairing and key distribution, and allows the other layers in the protocol stack to connect and exchange data with other devices safely. A connection in the data link layer and certain security standards are required in this process. The GAP SMP allows two devices encrypting their connection in the data link layer by setting such security levels as those in the SMP chapter of the Bluetooth Core Specification version. Before introducing the implementation of the GAP SMP, we should clarify the following concepts:

配对 表示两个设备已同意建立具有特定安全级别的连接。

- Pairing** indicates that two devices have agreed to establish a connection with certain security levels.
- Bonding** indicates that at least one device has sent some kind of indication or security information, which could be an LTK, CSRK or IRK, to another device for future connections. If these two devices can bond with each other, the key distribution occurs after the pairing, otherwise no bonding information will be exchanged. Bonding is not a prerequisite for pairing. However, during pairing the two devices exchange their characteristics to determine whether the peer device is open for bonding. If neither of these two devices is open for bonding, no security information of the peer devices should be stored.
- Authentication** indicates the security of a link. However, a deauthentication link does not necessarily mean this link is not secure at all. When the key for the link encryption has the security attributes that have been confirmed by both devices, these two devices are considered authenticated. When STK is used for the authentication, a keyword is generated during the pairing. For devices with input/output and OOB functions, all the keys generated and exchanged have the MITM attributes (PIN/larger OOB keys are used, which enforces security). If Just Works is used, all the keys generated and exchanged have the No MITM attributes.
- Authorization** is defined as the assignment of permission to perform an operation from the application layer. Some applications may require authorization, in which case

绑定表示至少一个设备已经向另一个设备发送了某种指示或安全信息，可以是LTK、CSRK或IRK，用于将来的连接。如果这两个设备可以彼此绑定，则在配对后进行密钥分发，否则将不会交换绑定信息。结合不是配对的先决条件。然而，在配对期间，两个设备交换它们的特性，以确定对等设备是否开放用于绑定。如果这两个设备都没有打开进行绑定，则不应存储对等设备的安全信息。

身份验证指示链接的安全性。然而，取消身份验证链接并不一定意味着该链接根本不安全。当链接加密的密钥具有两个设备都已确认的安全属性时，这两个设备被视为已验证。当STK用于认证时，在配对过程中会生成关键字。对于具有输入/输出和OOB功能的设备，生成和交换的所有密钥都具有MITM属性（使用PIN/更大的OOB密钥，这加强了安全性）。如果使用Just Works，则生成和交换的所有密钥都具有No MITM属性。



the application must be granted permission before being used. If no permission is given, the whole process will fail.

### 3.3.2. Safety Management Controller

#### 3.3.2.1. BLE Encryption

The encryption of a BLE device can be achieved with two basic methods:

- When no bonding is established between two BLE devices, these devices are encrypted through the pairing procedure, while bonding (or not bonding) is determined according to the specific pairing information of these BLE devices.
- Two bonded devices: initiate the encryption through the bonding procedure. When two devices have already bonded, encryption is initiated with one device resorting to the original bonding process.

The way in which a master initiates an encryption request in Just Works mode can be seen in the flow chart below:

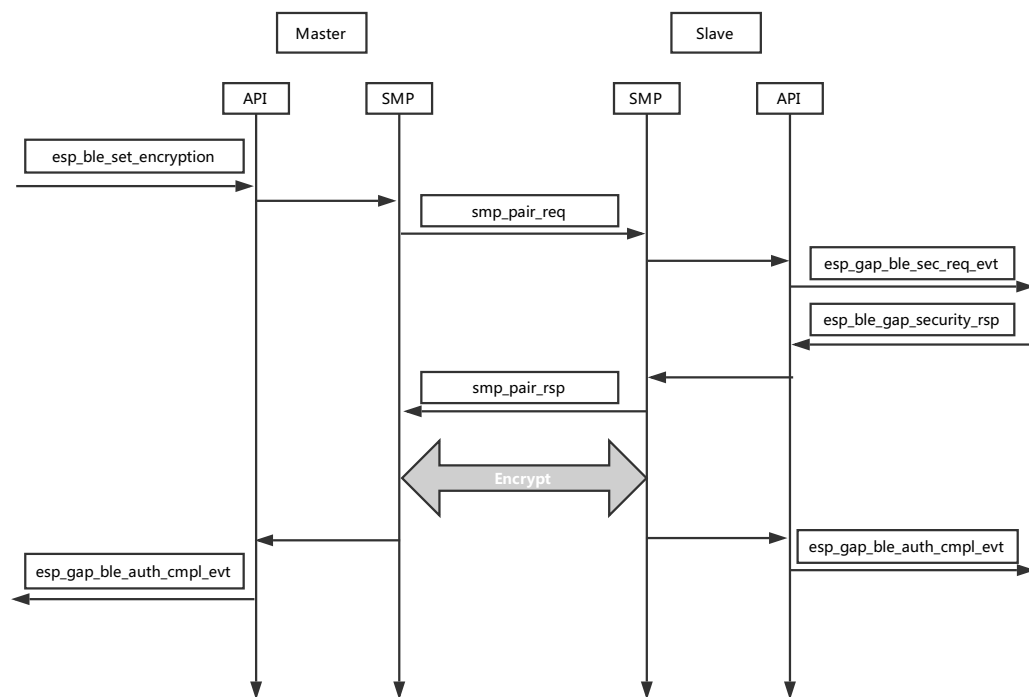


Figure 3-6. The flow chart of encryption in Just Works mode





The way in which a master initiates an encryption request in Passkey Notify mode can be seen in the flow chart below:

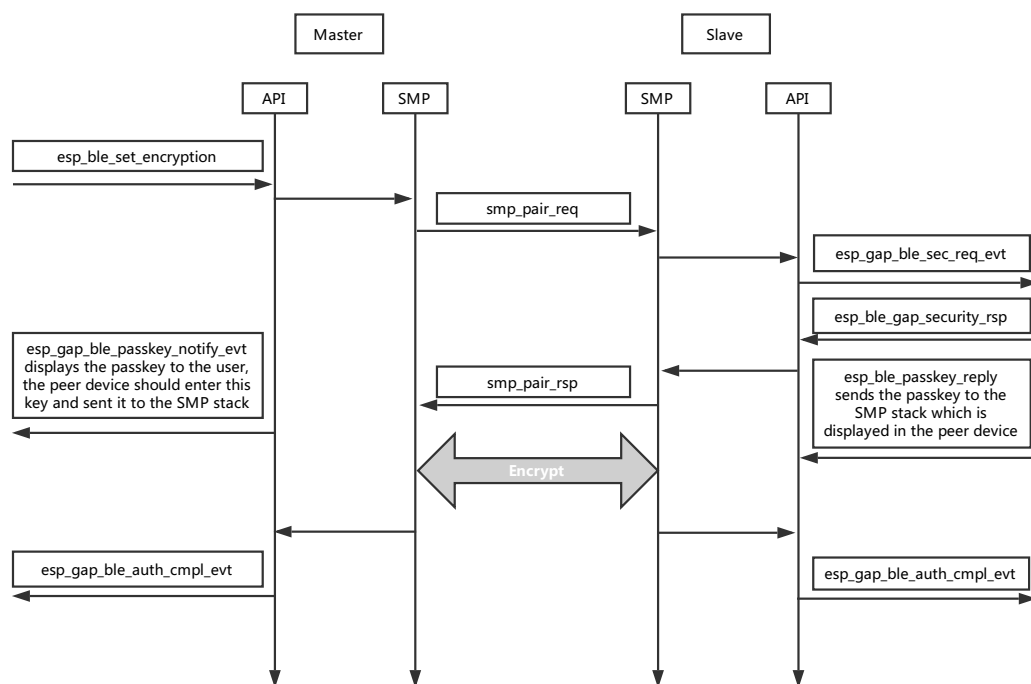


Figure 3-7. The flow chart of encryption in Passkey Notify mode



### 3.3.2.2. BLE Bonding

The bonding between two BLE devices is achieved by calling a GAP API. According to the description in the Bluetooth Core Specification, the purpose of bonding is that two BLE devices, which have been encrypted by SMP, are able to use the same keys to encrypt a link when they reconnect with each other, thus simplifying the reconnection process. These two BLE devices exchange encryption keys during their pairing, and store them for long-term use. The bonding process can be seen in the flow chart below:

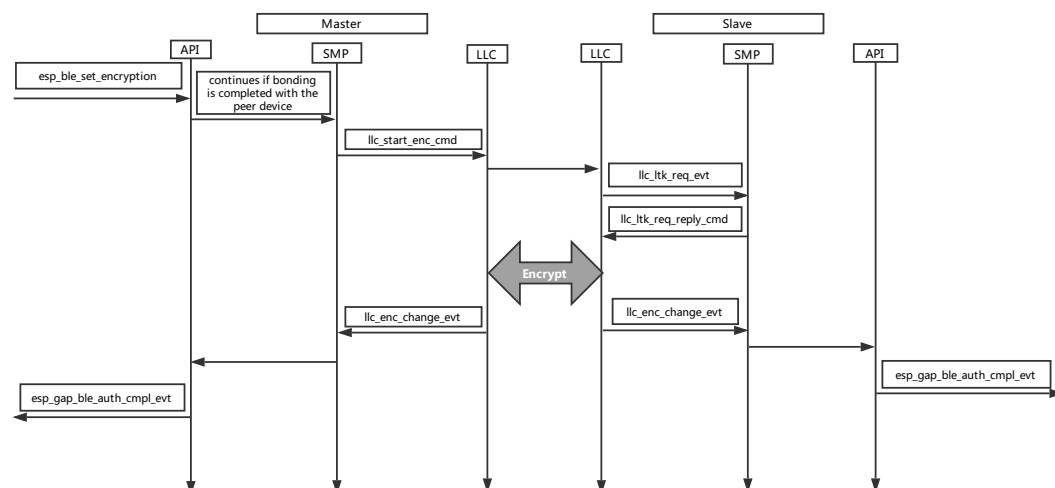


Figure 3-8. The flow chart of BLE bonding process

**! Notice:**

The bonding process must be initiated by a master device during the connection.

### 3.3.3. The Implementation of SMP

The BLE SMP calls encryption APIs in BLE GAP, registers the BLE GAP callbacks, and obtains the current encryption status through the return values of events.



Espressif IoT Team  
[www.espressif.com](http://www.espressif.com)

#### **Disclaimer and Copyright Notice**

Information in this document, including URL references, is subject to change without notice.

THIS DOCUMENT IS PROVIDED AS IS WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.

All liability, including liability for infringement of any proprietary rights, relating to use of information in this document is disclaimed. No licenses express or implied, by estoppel or otherwise, to any intellectual property rights are granted herein.

The Wi-Fi Alliance Member logo is a trademark of the Wi-Fi Alliance. The Bluetooth logo is a registered trademark of Bluetooth SIG.

All trade names, trademarks and registered trademarks mentioned in this document are property of their respective owners, and are hereby acknowledged.

**Copyright © 2019 Espressif Inc. All rights reserved.**