低功耗蓝牙协议栈 v3.x 编程指南

引言

本文档旨在为开发人员提供相关参考编程指南，用于说明如何使用 BLE 协议栈 v3.x 系列 API 和相关事件回调开发低功耗蓝牙（BLE）应用。

本文档介绍了允许访问意法半导体低功耗蓝牙设备片上系统所提供的低功耗蓝牙功能的 BLE 协议栈 v3.x 库框架、API 接口和事件回调。

以下低功耗蓝牙设备支持 BLE 协议栈 v3.x 系列：

- BlueNRG-LP 设备
- BlueNRG-LPS 设备（BLE 协议栈 v3.1 或更高版本）

该文档还关注 API 和回调接口、BLE 协议栈初始化（相对于 BLE 协议栈 v2.x 系列）的关键变化。

本编程手册还提供一些与低功耗蓝牙技术有关的基本概念，以便将 BLE 协议栈 v3.x API、参数及相关事件回调与 BLE 协议栈特性联系起来。希望用户在阅读完本文档后能对 BLE 技术及其主要特性有一个基本的了解。

有关支持的设备和低功耗蓝牙规范的更多信息，请参考本文档结尾处的第 5 节“参考文件”。

手册结构如下：

- 低功耗蓝牙技术的基本原理
- BLE 协议栈 v3.x 库 API 和事件回调概述
- 如何使用 BLE 协议栈 v3.x 库 API 和事件回调设计应用。

注意：　本文档内容适用于所有指定的低功耗蓝牙设备。必要时，会着重标明具体的区别。

1 低功耗蓝牙技术

低功耗蓝牙（BLE）无线技术由蓝牙技术联盟（SIG）开发，目的是使设备能够以极低功耗标准使用纽扣电池工作数年。

传统蓝牙作为一种无线技术标准，可以取代连接便携式和/或固定式电子设备的线缆，但是由于采取了快速跳频、以连接为导向的行为方式和相对复杂的连接流程，无法采用电池供电的方式。

低功耗蓝牙设备的功耗仅为标准蓝牙产品的一小部分，让使用纽扣电池的设备能够无线连接到启用了标准蓝牙的设备。

图 1. 支持低功耗蓝牙技术的以钮扣电池供电的设备

GAMSEC201411251047

低功耗蓝牙技术广泛应用于传输少量数据的传感器应用中：

- 汽车
- 运动与健身
- 医疗
- 娱乐
- 家庭自动化
- 安全和接近感测

1.1 BLE 协议栈架构

低功耗蓝牙技术已被蓝牙核心规范 4.0 正式采纳（参见第 5 节 参考文件）。该版本的蓝牙标准支持两种无线技术系统：

- 基础速率（BR）
- 超低功耗蓝牙

低功耗蓝牙技术工作在工业、科学和医疗（ISM）频段 2.4~2.485GHz，可以在全球许多国家使用而无需官方授权。它使用扩频、跳频、全双工信号。低功耗蓝牙技术的关键特性：

- 稳健性
- 性能
- 可靠性
- 互操作性
- 低速率

- 低功耗。

另外，低功耗蓝牙技术的目的是为了实现在传输极小数据包的同时，其功耗显著低于基础速率（BR）、增强数据率（EBR）以及高速设备（HS）。

低功耗蓝牙协议栈由两部分组成：

- 控制器
- 主机

控制器包含物理层和链路层。

主机包括逻辑链路控制和适配协议（L2CAP）、安全管理器（SM）、属性协议（ATT）、通用属性配置文件（GATT）和通用访问配置文件（GAP）。两个组成部分之间的接口被称为主机控制器接口（HCI）。

此外，已发布的蓝牙规范 v4.1、v4.2 和 v5.x 具备以下新特性：关于这些新特性的更多信息，请参考相关规范文档。

图 2. 低功耗蓝牙协议栈架构

GAMSEC201411251124

1.2 物理层

LE 1M 物理层用来接收和发射 1Mbps 自适应跳频射频信号，采用高斯频移键控（GFSK）技术。它在无需执照的 2.4 GHz ISM 波段中以 2400-2483.5 MHz 的频率工作。很多其他标准也使用此波段：IEEE 802.11, IEEE 802.15.

低功耗蓝牙系统使用 40 射频通道（0-39），2 MHz 间隔。这些射频通道的频率中心为：

$$2402 + k * 2MHz，其中\ k = 0, ..., 39 \quad (1)$$

RF 通道可以分为两组：

1. 主广播通道，使用三个固定的射频通道（37、38 和 39）用于：
   a. 传统广播和连接
   b. 扩展广播的初始化包

2. 使用所有其他 RF 通道的通用通道。

表 1. 低功耗蓝牙 RF 通道类型和频率

通道索引	射频中心频率	通道类型
37	2402 MHz	主广播
0	2404 MHz	通用
1	2406 MHz	通用
….	….	通用
10	2424 MHz	通用
38	2426 MHz	主广播
11	2428 MHz	通用
12	2430 MHz	通用
….	….	通用
36	2478 MHz	通用
39	2480 MHz	主广播

低功耗蓝牙是一种自适应跳频（AFH）技术，只能使用所有可用频率的一个子集，以避开其他非自适应技术使用的所有频率。通过使用特定的跳频算法确定下一个要使用的良好通道，可以从不良通道转移到已知良好通道。

1.2.1 LE 2M 和“LE Coded”物理层

低功耗蓝牙规范 v5.x 在低功耗蓝牙规范 v4.x 提供的 PHY 规范（LE 1M）基础上增加了另外两个 PHY 变体：

- LE 2M
- LE Coded

在低功耗蓝牙规范 v5.x 中定义了标准 HCI API，分别为通过 LE 传输的所有后续连接的发送器 PHY 和接收器 PHY 设置 PHY 首选值（LE 1M、LE 2M、LE Coded），或为特定连接设置发送器 PHY 和接收器 PHY。

注意： *低功耗蓝牙规范 v5.x 中仍然强制支持 LE 1M。*

1.2.2 LE 2M PHY

主要特征：

- 2 Msym/s 调制
- 未编码

有几个应用用例需要更高的吞吐量：

- 无线（OTA）固件升级程序
- 运动、健身、医疗应用用例需要以更高精度收集大量数据，并通过某些医疗设备更频繁地发送数据。

LE 2M PHY 允许物理层以 2 Mbps 的速度工作，因此，PHY 可以实现比 LE 1M 更高的数据速率。它使用自适应跳频射频信号，采用高斯频移键控（GFSK）技术。LE 2M PHY 使用的频率偏差至少 370 kHz。

1.2.3 LE Coded PHY

主要特征：

- 1 Msym/s 调制
  - 与 LE 1M 相同
- 有效负载可以用两种不同的速率编码：
  - 500 kb/s（S = 2）
  - 125 kb/s（S = 8）

一些应用场景要求更大的距离。距离增加之后，信噪比（SNR）开始下降，因此，解码错误概率上升：误码率（BER）升高。

LE Coded 的 PHY 使用前向纠错（FEC）对接收的报文进行纠错。这允许以较低的信噪比（SNR）值正确解码接收到的报文，因此，它增加了发送器距离，而无需提高发送器功率水平（距离可达低功耗蓝牙 v4.x 规范允许的 4 倍）。

FEC 方法在传输的报文中添加一些特定的位，从而使 FEC 能够确定错误位应该具有的正确值。FEC 方法为比特流处理增加了两个步骤：

1. FEC 编码，为每个比特生成 2 个额外的比特
2. 模式映射，它根据两种编码方案转换 P 符号中前一步的每个比特：

- S= 2：没有变化。该情况下，距离（大约）翻倍
- S= 8：每个比特映射为 4 个比特。该情况下，距离（大约）翻两倍

由于 FEC 方法在整个报文中增加了一定数量的比特和要传输的数据量：因此数据通信速率降低了。

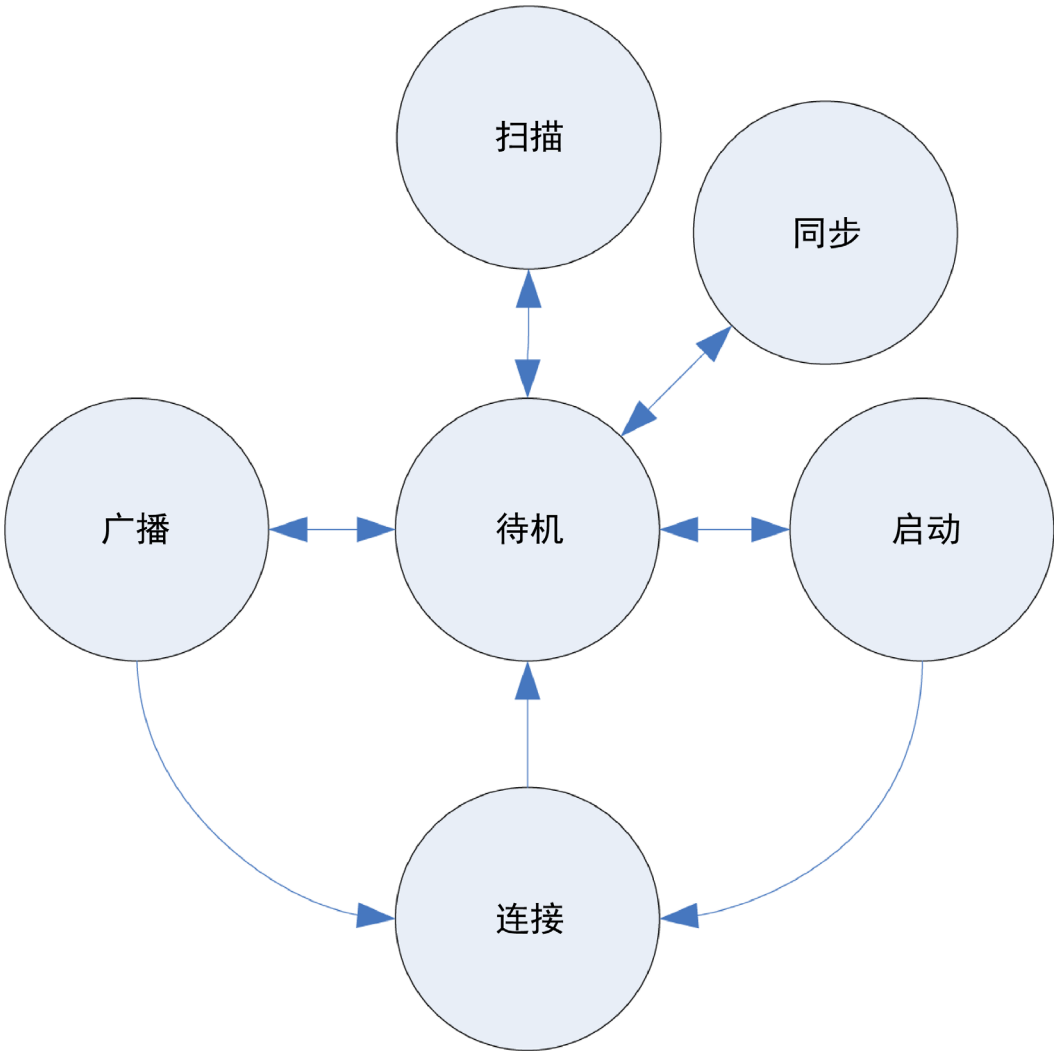
表 2. LE PHY 关键参数

	LE 1M	LE 2M	Le Coded（S=2）	Le Coded（S=8）
符号率	1 Ms/s	2 Ms/s	1 Ms/s	1 Ms/s
数据率	1 Mbps	2 Mbps	500 Kbps	125 Kbps
错误检测	3 bytes CRC	3 bytes CRC	3 bytes CRC	3 bytes CRC
误差校正	无	无	FEC	FEC
距离增加	1	0.8	2	4
低功耗蓝牙规范 5.x 要求类型	强制	可选	可选	可选

1.3 链路层（LL）

链路层（LL）定义了两个设备使用射频信号在彼此之间传递信息的方式。链路层定义了具有五种状态的状态机：

图 3. LL 状态机



- 待机：设备不发送或接收数据包
- 广播：设备在广播通道中发送广播（称为广播设备）
- 扫描：设备寻找广播设备（称为扫描设备）
- 发起：设备发起与广播设备的连接
- 连接：如果从发起状态进入该状态，设备处于主设备位置。如果从广播状态进入连接状态，设备处于从设备位置。主设备与从设备进行通讯并定义传输的时序。
- 同步：设备收听周期性的广播。

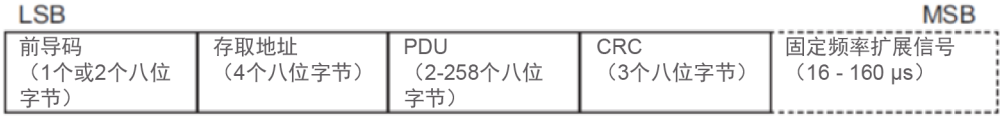
1.3.1 低功耗蓝牙报文

uncoded PHY 和 coded PHY 的低功耗蓝牙报文格式不同。

面向 uncoded PHY（LE 1M 和 LE 2M）的格式如图 4. 2MB 中所示。

低功耗蓝牙数据包的结构如下。

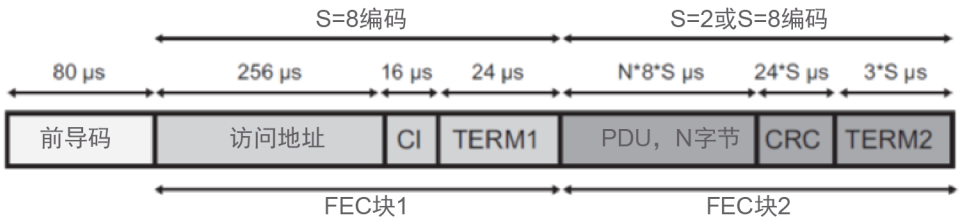
图 4. **2MB**



- 前导码：数据包的开始。1 个八位字节用于 LE 1M PHY，2 个八位字节用于 LE 2M PHY。
- 访问地址：32 位。对于每个连接是不一样的。
- PDU：它可以是广播物理信道 PDU 或数据物理信道 PDU。
- CRC：通过 PDU 计算的 24 位控制代码。
- 固定频率扩展信号：可选。由一串已调制，未白化的 1 组成。

面向 LE Coded 的 PHY 的数据包格式如图 5. Coded PHY 中所示。

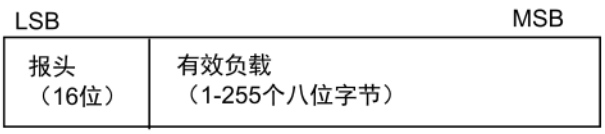
图 5. **Coded PHY**



- 前导码：80 个符号，未编码
- 访问地址：32 位。对于每个连接是不一样的
- CI：编码指示。在两种编码配置中进行选择：S=2 或 S=8
- PDU：可以是广播物理信道 PDU 或数据物理信道 PDU
- CRC：通过 PDU 计算的 24 位控制代码
- TERM1 和 TERM2：FEC 块的终止序列，使编码器进入原始状态。

PDU 的内容取决于数据包的类型：广播物理信道 PDU 或数据物理信道 PDU。
广播物理信道 PDU 的格式如图 6. 广播物理通道 PDU 中所示。

图 6. **广播物理通道 PDU**



- 报头是 16 位，包含关于 PDU 类型和有效负载长度的信息
- 有效负载字段特定于 PDU 类型。

广播报头中包含的 PDU 类型可以是表 3. PDU 广播报头中描述的类型之一。

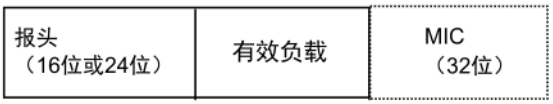
表 3. PDU 广播报头

数据包类型	数据包名称	说明	广播物理信道	允许的 PHY			
				LE 1M	LE 2M	LE Coded	注释
0000b	ADV_IND	可连接非定向广播	初级	X			由广播方在需要另一台设备与之连接时使用。该设备可以被扫描设备扫描，也可以在收到连接请求后作为从属设备进入连接状态。
0001b	ADV_DIRECT_IND	可连接定向广播	初级	X			由广播方在需要连接到特定设备时使用。ADV_DIRECT_IND 数据包仅包含广播方地址和发起方地址。
0010b	ADV_NONCONN_IND	不可连接非定向广播	初级	X			由广播方在需要向所有设备提供某些信息，但是不希望其他设备向其请求更多信息或与之连接时使用。 设备只在相关通道上发送广播数据包，但不希望任何其他设备与之连接或扫描。
0011b	SCAN_REQ	扫描请求	初级	X			由处于扫描状态的设备使用，用于向位于主通道（使用传统广播包 PDU）的广播方请求更多信息。
	AUX_SCAN_REQ	辅助扫描请求	次级	X	X	X	由扫描设备使用，用于请求关于辅助广播物理通道的附加信息。
0100b	SCAN_RSP	扫描响应	初级	X			由广播设备使用，用于在收到 SCAN_REQ 后向扫描设备提供更多信息。
0101b	CONNECT_IND	连接请求	初级	X			由发起设备发送给主广播通道上处于可连接/可发现模式的设备。
	AUX_CONNECT_REQ	辅助连接请求	次级	X	X	X	由发起设备发送给辅助广播通道上处于可连接/可发现模式的设备。
0110b	ADV_SCAN_IND	可扫描非定向广播	初级	X			由希望允许扫描设备请求更多信息的广播方使用。设备不能连接，但是对于广播数据和扫描响应数据而言可发现。
0111b	ADV_EXT_IND	扩展广播 PDU	初级	X		X	扩展广播报文，通常指辅助广播通道上的 AUX_ADV_IND 报文。
	AUX_ADV_IND	辅助广播	次级	X	X	X	包含辅助广播通道广播信息的报文。
	AUX_SCAN_RSP	辅助扫描响应	次级	X	X	X	由广播设备在收到 AUX_SCAN_REQ 之后发送。

数据包类型	数据包名称	说明	广播物理信道	允许的 PHY			
				LE 1M	LE 2M	LE Coded	注释
0111b	AUX_SYNC_IND	辅助同步 PDU	定期	X	X	X	在周期性广播中使用的报文。
	AUX_CHAIN_IND	辅助链接 PDU	辅助和周期性	X	X	X	用于添加额外数据的 PDU。
1000b	AUX_CONNECT_RSP	辅助连接响应	次级	X	X	X	用于响应 AUX_CONNECT_REQ 的 PDU

数据物理信道 PDU 的格式如图 7. 数据物理信道 PDU 中所示。

图 7. 数据物理通道 PDU



- 报头包含 PDU 类型、有效负载长度和序列号等其他信息。其长度为 16 位，但可增至 24 位，前提是包含 CTE 信息（即使用固定频率扩展信号进行测向）
- 有效负载最高可达 251 个八位字节
- 如果有效负载长度不为零，则 MIC 包含在加密链路层连接中。

1.3.2 扩展广播

根据低功耗蓝牙规范 v4.x，广播报文的最大有效负载为 31 字节。每个广播报文的有效负载通过三个特定通道（37、38、39）发送，每次发送一个报文。

低功耗蓝牙 v5.x 广播扩展能力允许：

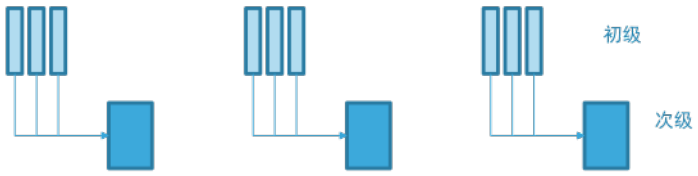
- 在无连接的场景（如信标）下扩展数据长度
- 发送多组广播数据
- 以确定的方式发送广播

新扩展的广播报文可以使用低功耗蓝牙 4.x 连接通道（0-36），用于将广播有效负载扩展到 255 字节。初始广播和遗留 PDU 通过 3 个 RF 通道（37、38、39）传输，这些通道称为“主广播物理通道”。报头字段还包括一个新的数据 AuxPtr，包含通道编号（0-36），报文（包括广播负载）在其中传输（即辅助通道）。

大部分通信是在 37 个 RF 通道即“辅助广播物理通道”上进行的。

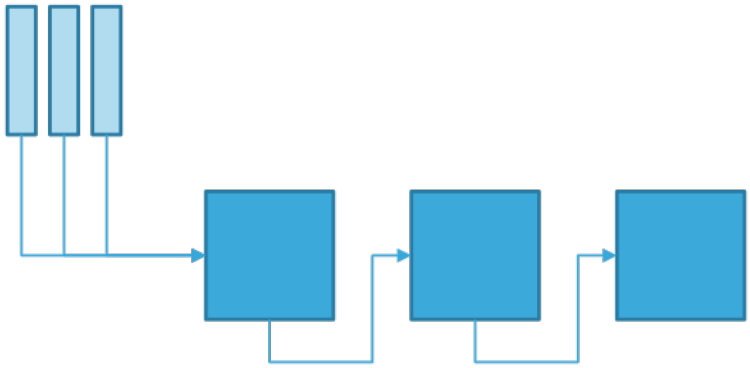
使用新的 ADV_EXT_IND 报文，可在主广播物理通道上发送。如果需要，该报文包含一个指向辅助广播物理通道上辅助报文的指针：大部分信息都在辅助报文上。

图 8. 低功耗蓝牙 5.x 扩展广播



另外，还可以在辅助通道上创建广播数据包链，以传输更多的广播有效负载数据（大于 255 字节）。辅助通道上的每个广播数据包都包括其 AuxPtr，即用于链条上的下一个广播数据包的下一个辅助通道编号。因此，广播数据包链中的每个链条都可以在不同的辅助通道上发送。

图 9. 广播数据包链



新扩展广播数据包的一个直接优点是能够在主通道（37、38、39）上发送更少的数据，从而减少主通道上的数据量。此外，通过降低总体占空比，广播有效负载仅在辅助通道上发送，不再在全部三个主广播通道上发送。

允许的通道存在一些限制：

- 传统广播 PDU 和新的 ADV_EXT_IND 只允许在主广播物理通道上使用
- 新广播 PDU（除了面向 ADV_EXT_IND 的）仅允许在辅助广播物理通道上使用。

允许的 PHY 同样存在一些限制：

- 传统广播 PDU 仅可使用 LE 1M PHY
- 新的广播 PDU 可以使用三个 PHY（LE 1M、LE 2M、LE Coded）中的一个，但是主广播通道上不允许使用 LE 2M。所以，ADV_EXT_IND 仅可在 LE 1M 和“LE Coded”上发送。

在主通道和辅助通道之间，任何 PHY 组合都是可能的。然而，有些组合可能没有现实用例。

常见用例有：LE 1M -> LE 1M，LE 1M -> LE 2M，LE Coded -> LE Coded

注意： *对于非可连接广播，最短广播间隔已从 100 毫秒减少到 20 毫秒。*

1.3.3 广播集

低功耗蓝牙 v4.x 在广播期间不改变广播有效负载，以便让不同的广播数据包拥有不同的数据。

它并未提供足够的灵活性，使不同的广播数据与节能机制相结合。

低功耗蓝牙 v5.x 定义具有 ID 的广播集，用于指示每个数据包具体属于哪个集。

每个集拥有一个 ID（SID，Set ID），用于指示每个数据包属于哪个集。

- SID 是通过无线发送的广播集的标识符
- 由于一台广播设备可以有多个广播集，该数字可以被扫描仪器用于区分来自同一广播设备的不同广播集
- 该 SID（加上 DID，即“Data ID”）用于重复过滤

每个集都有其特定的广播参数（广播间隔、PDU 类型），可以使用主通道或辅助通道。

链路层可以调度和传输由主机层定义的广播集。

1.3.4 广播状态

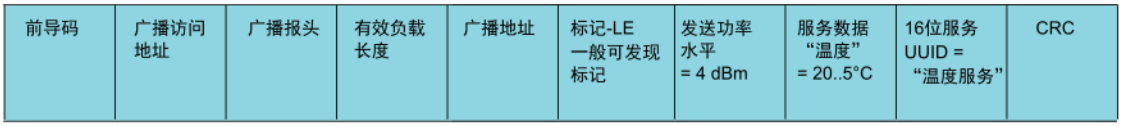
广播状态允许链路层发送广播数据包，并对正在执行扫描的设备发出的扫描请求作出扫描响应。

可通过停止广播使广播设备进入待机状态。

设备每次进行广播时，通过三个广播通道发送相同数据包。此三个数据包序列称为一个广播事件。两个广播事件之间的时间被称为广播间隔，长度从 20 毫秒到 10.28 秒不等。

广播数据包的示例列出了设备实现的服务 UUID（一般可发现标记，发送功率 = 4 dBm，服务数据 = 温度服务和 16 位服务 UUID）。

图 10. 具有 AD 类型标记的广播数据包



GAMSEC201411251139

AD 类型标记字节（Flags）包含以下标记位：

- 有限可发现模式（第 0 位）
- 一般可发现模式（第 1 位）
- 不支持 BR/EDR（第 2 位，在低功耗蓝牙上为 1）
- 可同时以 LE 和 BR/EDR 连接到同一设备（控制器）（第 3 位）
- 可同时以 LE 和 BR/EDR 连接到同一设备（主机）（第 4 位）

如果任意位为非零，则在广播数据中包含 AD 类型标记（不包括在扫描响应中）。

在启用广播之前，可设置下列广播参数：

- 广播间隔
- 广播地址类型
- 广播设备地址
- 广播通道映射：应使用三个广播通道中的哪一个

- 广播过滤策略：
  - 处理来自白名单中的设备的扫描/连接请求
  - 处理所有扫描/连接请求（默认的广播方过滤策略）
  - 处理来自所有设备的连接请求，但仅处理来自白名单中的设备的扫描请求
  - 处理来自所有设备的扫描请求，但仅处理来自白名单中的设备的连接请求。

白名单是设备控制器用于过滤设备的已保存设备地址列表。当白名单正在使用时，不能修改其内容。如果设备处于广播状态且正在使用白名单过滤设备（扫描请求或连接请求），必须禁用广播模式才能修改其白名单。

1.3.5 扫描状态

有两种类型的扫描：

- 被动扫描：允许接收来自广播设备的广播数据
- 主动扫描：在接收到广播数据包时，设备可以发回扫描请求数据包，以便得到广播方的扫描响应。这使扫描设备能够获取来自广播设备的额外信息。

可以设置下列扫描参数：

- 扫描类型（被动或主动）
- 扫描间隔：控制器的扫描频率
- 扫描窗口：对于每个扫描间隔而言，它定义了设备扫描的持续时间
- 扫描过滤策略：它可以接受所有广播数据包（默认策略）或仅接受来自白名单设备的广播数据包。

在设置扫描参数后，可以启用设备扫描。扫描设备的控制器向上层发送任何在广播报告事件中接收的广播数据包。该事件包含该广播数据包的广播方地址、广播方数据和接收信号强度指示（RSSI）。可将 RSSI 与广播数据包中包含的发送功率水平信息一起使用，以确定信号的路径损耗和设备距离：

路径损耗 = 发送功率 – RSSI

1.3.6 连接状态

当待发送数据的复杂度超过广播数据允许的水平，或者需要在两个设备之间建立双向可靠通信时，建立连接。

当发起设备接收到它要连接的广播设备发出的广播数据包时，它可以向广播设备发送连接请求数据包。该数据包包含建立和处理两个设备之间的连接所需的所有必要信息：

- 用于识别已建立连接的物理链路层通讯的访问地址
- CRC 初始值
- 发送窗口大小（第一个数据包的时序窗口）
- 发送窗口偏移（发送窗口起点）
- 连接间隔（两个连接事件之间的时间）
- 从设备延迟（从设备在被强制监听前可以忽略的连接事件次数）
- 监控超时（在链路被视为丢失之前两次正确接收到数据包之间的最长时间）
- 通道映射：37 位（1= 良好；0 = 不佳）
- 跳频值（5 至 16 之间的随机数）
- 睡眠时钟精度范围（用于确定连接事件中从设备的不确定性窗口）。

关于连接请求数据包的详细描述，请参考蓝牙规范[第 6 卷]。

表 1. 低功耗蓝牙 RF 通道类型和频率中总结了允许的时序范围：

表 4. 连接请求时序间隔

参数	最小值	最大值	注释
发送窗口长度	1.25 毫秒	10 毫秒	

参数	最小值	最大值	注释
发送窗口偏移	0	连接间隔	1.25 毫秒的倍数
连接间隔	7.5 毫秒	4 秒	1.25 毫秒的倍数
监控超时	100 毫秒	32 秒	10 毫秒的倍数

发送窗口在连接请求数据包末尾加上发送窗口偏移加上 1.25 ms 的强制延迟后启动。发送窗口启动时，从设备进入接收器模式并等待来自主设备的数据包。如果这段时间内没有接收到数据包，从设备将退出接收器模式，并且在下一个连接间隔重新尝试接收来自主设备的数据。在连接建立后，主设备必须就每个连接事件向从设备发送数据包，以允许从设备向主设备发送数据包。或者，从设备可以跳过给定数量的连接事件（从设备延迟）。

连接事件是上一个连接事件的起点与下一个连接事件的起点之间的时间。

一个 BLE 从设备只能连接一个 BLE 主设备，但是一个 BLE 主设备可以连接多个 BLE 从设备。蓝牙 SIG 对一个主设备可以连接的从设备数量并无限制（只受限于使用的特定 BLE 技术或协议栈）。

1.3.6.1 扩展扫描

由于扩展广播采用新数据包和新 PHY，所以在扫描流程中会体现出这些变化。

使用 LE 1M 可在主通道上扫描，以查找：

- 遗留事件
- 扩展广播事件，可切换到辅助广播物理通道上的其他 PHY

使用 LE Coded 可在主通道上扫描，以查找：

- 扩展广播事件，可切换到辅助广播物理通道上的其他 PHY。

1.3.7 周期性广播和周期性广播同步传输

蓝牙规范 v5.0 定义了周期性广播，它使用确定性调度来允许一台设备将其扫描与另一台设备的广播同步。

通用访问配置文件定义了一种新的同步模式，该模式允许执行周期性广播同步建立流程并与广播同步。

周期性广播使用新的链路层 PDU（称为 AUX_SYNC_IND）。与周期性广播数据包同步所需的信息（定时和定时偏移）被发送到 AUX_ADV_IND PDU 中的 SyncInfo 字段。

周期性广播同步流程有功耗方面的开销，并且一些设备不具备执行此流程的条件。

当前定义了一个新的周期性广播同步传输（PAST）流程，以允许设备 A（从设备 B 接收周期性广播报文）将获取的同步信息传递给与设备 A 相连的另一台设备 C。因此，设备 C 能够直接从设备 B 接收周期性广播报文，而不需要扫描 AUX_ADV_IND PDU，以免消耗过多能量。

1.3.8 随机广播

低功耗蓝牙使用三个主广播通道。发送到这些通道的 PDU 构成一个广播事件。

为了减少可能的报文冲突，两个连续广播事件之间的时间必须有一个从 0 到 10ms 的随机延迟。

在蓝牙核心规范 v5.0 中，广播事件按照严格顺序（即 37、38、39）使用三个主广播通道；而从蓝牙 v5.1 开始，不再需要遵守该顺序。这意味着 PDU 可能不会被发送到固定顺序，现在甚至可以将使用的主广播通道顺序随机化。实践证明，随机选择广播通道进一步降低了报文冲突概率，在拥挤的网络上具有明显优势。

1.3.9 BLE 功率控制

BLE 功率控制功能允许动态更新特定连接上的传输功率水平。

该功能通过动态改变当前连接内的传输功率水平来降低发送设备的总体功耗。

此外，LE 功率控制功能允许接收设备动态监测连接内的信号强度，并请求更改传输设备的功率水平，以便使所需的信号强度保持在请求的范围内并优化功耗。

将传输设备功率水平调整到最佳水平的能力也提高了与所有其他 2.4 GHz 无线网络的共存能力。

1.4 主机控制器接口（HCI）

主机控制器接口（HCI）层使主机和控制器之间可以通过软件 API 或者通过 SPI、UART 或 USB 等硬件接口：实现通信。它来自标准蓝牙规范，包含用于低功耗特定功能的附加新指令。

1.5 逻辑链路控制和适配层协议（L2CAP）

逻辑链路控制和适配层协议（L2CAP）支持更高层协议复用、数据包分割和重组操作以及服务信息质量的通知。

1.5.1 LE L2CAP 以连接为导向的信道

L2CAP 以连接为导向的通道支持高效的大量数据传输，同时降低成本。服务数据单元（SDU）借助流控制进行可靠传输。大 SDU 的分割和重组由 L2CAP 实体自动完成。多路复用支持同时执行多个服务。

1.6 属性配置文件（ATT）

属性配置文件（ATT）允许设备向另一设备公开某些数据，即属性。定义属性的设备被称为服务器，而使用它们的对端设备被称为客户端。

属性是一种包含下列组成部分的数据：

- 属性句柄：16 位值，用于标识服务器端相应属性数据。客户端在读写请求中引用该句柄
- 属性类型：通过通用唯一标识符（UUID）定义，决定了值的含义。蓝牙 SIG 定义了标准 16 位属性 UUID
- 属性值：长度为一个（0 ~ 512）八位组
- 属性权限：由使用该属性的更高层协议定义。它们指定读和/或写访问以及通知和/或指示需要的安全级别。使用属性协议不能发现权限。几种不同的许可类型如下：
  - 访问权限：决定了可以对属性执行的请求类型（可读、可写、可读且可写）
  - 验证权限：决定了属性是否需要验证。如果发生了验证错误，客户端可以尝试使用安全管理器进行验证并发回请求
  - 授权权限（无授权、授权）：这是服务器的属性，决定了是否授权客户端访问一组属性（客户端不能解决授权错误）。

表 5. 属性示例

句柄属性	属性类型	属性值	属性权限
0x0008	“温度 UUID”	“温度值”	“只读，无授权，无验证”

- “温度 UUID”由“温度特征”规范定义，为有符号 16 位整数。

属性的集合被称为数据库，始终包含在属性服务器中。

属性协议定义了一组方法协议，用于在对端设备上发现、读取和写入属性。它在如下属性服务器和属性客户端之间实现端到端的客户端-服务器协议：

- 服务器角色
  - 包含所有属性（属性数据库）
  - 接收请求、执行、响应指令
  - 在数据变化时可以指示、通知属性值
- 客户端角色
  - 与服务器通信
  - 发送请求，等待响应（可以访问（读取）和更新（写入）数据）
  - 确认指示。

服务器公开的属性可以被客户端发现、读取和写入，并且可通过服务器指示和通知，如表 2. LE PHY 关键参数所示：

表 6. 属性协议消息

协议数据单元（PDU 消息）	发送者	说明
请求	客户端	客户询问服务器（总会导致响应）
响应	服务器	服务器发送对客户端请求的响应
指令	客户端	客户端命令服务器做某事（无响应）
通知	服务器	服务器将新值通知客户端（无确认）
指示	服务器	服务器向客户端指示新值（总是导致确认）
确认	客户端	对指示的确认

1.7 安全管理器（SM）

低功耗蓝牙链路层支持使用采用 CBC-MAC（加密块链-消息验证码）算法的计数器模式和 128 位 AES 分组密码（AES-CCM）进行加密和验证。当在连接中使用加密和认证时，为数据通道 PDU 的有效负载添加 4 字节的消息完整性检查（MIC）。

加密应用于 PDU 有效负载和 MIC 字段。

当两个设备要在连接期间进行通信加密时，安全管理器使用配对流程。此流程允许通过交换身份信息来验证两个设备，交换信息是为了创建可作为受信任关系或（单个）安全连接基础的安全密钥。有一些方法用于执行配对过程。其中一些方法提供了：

- 中间人（MITM）攻击防护：设备能够监控和修改两个设备之间的通信通道或向其添加新消息。典型场景为设备能够连接每个设备，并通过与每个设备通信来充当其他设备
- 被动窃听攻击：通过嗅探设备监听其他设备的通信

低功耗蓝牙规范 v4.0 或 v4.1 的配对也被称为 LE 传统配对，该配对支持基于设备 IO 功能的以下方法：直接工作法、输入密钥法和带外（OOB）法。

在低功耗蓝牙规范 v4.2 中，已定义 LE 安全连接配对模型。新安全模型的主要特性为：

1. 密钥交换过程使用了椭圆曲线（ECDH）算法：该算法允许通过不安全的通道交换密钥，并可以防止被动窃听攻击（通过嗅探设备秘密监听其他设备的通信）
2. 在 3 种可用的 LE 传统配对方法的基础上，增加了一种名为“数字比较”的新方法

根据设备 IO 功能选择配对流程。

有三种输入功能：

- 无输入
- 能够选择是/否
- 能够使用键盘输入数字。

有两种输出功能：

- 无输出
- 数字输出：能够显示六位数字

下表显示了可能的 IO 功能组合：

表 7. 低功耗蓝牙设备上输入/输出功能的组合

	无输出	显示
无输入	无输入，无输出	仅显示
是/否	无输入，无输出	显示是/否
键盘	仅键盘	键盘，显示

LE 传统配对

LE 传统配对算法使用并生成 2 个密钥：

- 临时密钥（TK）：用于生成短期密钥（STK）的 128 位临时密钥
- 短期密钥（STK）：用于在配对后加密连接的 128 位临时密钥

配对流程分为三个阶段。

第 1 阶段：配对特征交换

两个已建立连接的设备通过配对请求消息交换其输入/输出能力。此消息还包含一个指出带外数据是否可用的位，以及验证要求。在第 1 阶段交换的信息将用于选择第 2 阶段使用的 STK 生成配对方法。

第 2 阶段：短期密钥（STK）生成

配对设备首先使用下列密钥生成方法之一定义临时密钥（TK）：

1. 带外（OOB）法将带外通信（如 NFC）用于 TK 协议。它提供了身份认证（MITM 保护）。仅在两个设备上都设置了带外位时才选择此方法，否则必须使用设备的 IO 功能来确定可以使用其他方法（输入密钥法或直接工作法）
2. 输入密钥法：用户输入六位数字作为设备之间的 TK。它提供了身份认证（MITM 保护）
3. 直接工作法：该方法不提供认证，也不提供中间人（MITM）攻击防护

根据下表所定义的 IO 功能在输入密钥法和直接工作法之间进行选择。

表 8. 计算临时密钥（TK）的方法

	仅显示	显示是/否	仅键盘	无输入，无输出	键盘，显示
仅显示	直接工作	直接工作	输入密钥	直接工作	输入密钥
显示是/否	直接工作	直接工作	输入密钥	直接工作	输入密钥
仅键盘	输入密钥	输入密钥	输入密钥	直接工作	输入密钥
无输入，无输出	直接工作	直接工作	直接工作	直接工作	直接工作
键盘，显示	输入密钥	输入密钥	输入密钥	直接工作	输入密钥

第 3 阶段：用于计算临时密钥（TK）的传输特定密钥分配方法

一旦第 2 阶段完成，可通过使用 STK 密钥加密的消息分配最多三个 128 位密钥：

1. 长期密钥（LTK）：用于生成链路层加密和验证使用的 128 位密钥
2. 连接签名解析密钥（CSRK）：用于在 ATT 层上执行数据签名和验证的 128 位密钥
3. 身份解析密钥（IRK）：用于生成和解析随机地址的 128 位密钥

LE 安全连接

LE 安全连接配对方法使用并生成一个密钥：

- 长期密钥（LTK）：用于在配对和后续连接后加密连接的 128 位密钥

配对流程分为三个阶段：

第 1 阶段：配对特征交换

两个已建立连接的设备通过配对请求消息交换其输入/输出能力。此消息还包含一个指出带外数据是否可用的位，以及验证要求。在第 1 阶段交换的信息将用于选择第 2 阶段使用的配对方法。

第 2 阶段：长期密钥（LTK）生成

配对流程由发起设备启动，该设备将公钥发送至接收设备。接收设备将回复其公钥。对于所有配对方法（除了 OOB 以外），公钥交换阶段已完成。每个设备均产生自己的椭圆 Diffie-Hellman (ECDH)公钥 - 私钥对。每个密钥对均包含一个私钥（密钥）和一个公钥。密钥对应仅在每个设备上生成一次，并可以在执行配对之前计算。

支持以下配对密钥生成方法：

1. 使用带外通信来建立公钥的带外（OOB）法。如果带外位在配对请求/响应中至少由一个设备设置，则选择该方法，否则必须使用设备的 IO 功能来确定可以使用其他方法（输入密钥法、直接工作法或数字比较法）
2. Just Works：该方法无需验证，并且不提供针对中间人（MITM）攻击的任何保护
3. 输入密钥法：该方法需要认证。用户输入六位数字。该六位数字是设备认证的基础
4. 数字比较：该方法需要认证。两种设备都将 IO 能力设为是/否或键盘显示。两个设备都计算在两个设备上向用户显示的六位数字确定值：用户需要输入是或否来确认是否匹配。如果在两个设备上都选择是，则配对执行成功。当多个设备具有相同名字的情况下，此方法可以确保用户设备与正确的设备连接

根据下表在可能的方法中选择。

表 9. 将 IO 功能映射到可能的密钥生成方法

发起方/响应方	仅显示	显示是/否	仅键盘	无输入，无输出	键盘，显示
仅显示	直接工作	直接工作	输入密钥	直接工作	输入密钥
显示是/否	直接工作	直接工作 （LE 传统配对） 数字比较（LE 安全连接）	输入密钥	直接工作	输入密钥（LE 传统配对） 数字比较（LE 安全连接）
仅键盘	输入密钥	输入密钥	输入密钥	直接工作	输入密钥
无输入，无输出	直接工作	直接工作	直接工作	直接工作	直接工作
键盘，显示	输入密钥	输入密钥 （LE 传统配对） 数字比较（LE 安全连接）	输入密钥	直接工作	输入密钥（LE 传统配对） 数字比较（LE 安全连接）

注意： *如果可能的密钥生成方法未提供与安全属性匹配的密钥（经过身份验证 - MITM 保护或未经过身份验证 - 无 MITM 保护），则设备将发送配对失败指令，并显示“验证要求”的错误代码。*

第 3 阶段：传输特定密钥分配

主设备和从设备之间交换以下密钥：

- 用于验证未加密数据的连接签名解析密钥（CSRK）
- 用于设备身份和隐私的身份解析密钥（IRK）

当设备保存确定的加密密钥以用于未来验证时，设备被绑定。

数据签名

还可以使用 CSRK 密钥，通过未加密的链路层连接传输已验证数据：12 字节签名位于数据有效负载之后。签名算法还使用可防止重放攻击的计数器（一种外部设备，可以获取某些数据包然后发送。接收设备检查数据包计数器并丢弃，因为其帧计数器少于最近接收的良好数据包）。

1.8 隐私

总是用相同地址信息（公共地址或静态随机地址）的广播设备，很容易被其他扫描设备追踪。然而，为了防止这种情况发生，可以在广播设备上启用隐私功能。在启用了隐私的设备上，将使用私有地址。有两种私有地址：

- 不可解析私有地址
- 可解析私有地址

不可解析私有地址是完全随机的（两个最高有效位除外），不能进行解析。因此，使用不可解析私有地址的设备不能被没有预先配对的设备识别。可解析私有地址由 24 位随机部分和哈希部分组成。哈希部分来源于随机数和 IRK（身份解析密钥）。因此，只有知晓该 IRK 的设备能够解析地址并识别该设备。在配对过程中分发 IRK。

两种类型的地址都经常改变，从而增强了设备身份机密性。在 GAP 发现模式和过程期间不使用私有功能，仅在 GAP 连接模式和过程期间使用隐私功能。

在 v4.1 及以下的低功耗蓝牙协议栈中，通过主机解析和产生私有地址。在蓝牙 4.2 中，隐私功能已从版本 1.1 更新到版本 1.2。在低功耗蓝牙协议栈 v4.2 中，控制器可以使用主机提供的设备身份信息来解析和生成私有地址。

外设

启用了隐私功能的处于不可连接模式的外设将使用不可解析或可解析私有地址。

如需连接中央设备，外设设备中如启用主机隐私，则必须且只能使用非定向可连接模式。如启用控制器隐私，则外设设备还可使用定向可连接模式。在可连接模式中，外设设备使用可解析私有地址。

无论使用的是不可解析还是可解析私有地址，均以 15 分钟的间隔时间自动重新生成。设备不向广播数据发送设备名称。

中央设备

启用了隐私功能的执行主动扫描的中央设备只使用不可解析或可解析私有地址。为了连接外设，如果启用了主机隐私，应使用通用连接建立流程。对于基于控制器的隐私，可使用任何连接流程。中央设备将使用可解析私有地址作为发起方的设备地址。每间隔 15 分钟后生成可解析或不可解析私有地址。

广播设备

启用了隐私的广播设备使用不可解析或可解析私有地址。每间隔 15 分钟后将自动生成新地址。广播设备不应将设备名称或唯一数据发送到广播数据。

观察者

启用了隐私的观察者使用不可解析或可解析私有地址。每间隔 15 分钟后将自动生成新地址。

1.8.1 设备过滤

BLE 通过减少设备的响应次数来降低功耗，以减少控制器与上层之间的传输和交互。设备过滤通过白名单来实现。启用白名单后，链路层将忽略不在白名单中的设备。

在蓝牙 4.2 之前，当远程设备使用隐私功能后，无法实现设备过滤。由于引入了链路层隐私特性，可以在检查远程设备身份地址是否在白名单中之前解析远程设备身份地址。

1.9 通用属性配置文件（GATT）

通用属性配置文件（GATT）定义了使用 ATT 协议的框架，它被用于服务、特征、描述符发现、特征读取、写入、指示和通知。

就 GATT 而言，当两个设备已经连接时，有两种设备角色：

- GATT 客户端：通过读取、写入、通知或指示操作访问远程 GATT 服务器上的数据的设备
- GATT 服务器：在本地保存数据并向远程 GATT 客户端提供数据访问方法的设备

一个设备可以既是 GATT 服务器又是 GATT 客户端。

设备的 GATT 角色在逻辑上独立于主、从角色。主、从角色定义了低功耗蓝牙无线连接的管理方式，而 GATT 客户端/服务器角色由数据存储和数据流动来决定。

从（外围）设备是 GATT 服务器以及主（中央）设备是 GATT 客户端，这是最常见的，但不是必需的。

ATT 传输的属性封装在下列基础类型中：

1. 特征（具有相关描述符）
2. 服务（主要、次要和包含服务）

1.9.1 特征属性类型

特征是一种包含单个值和任意数量描述符的属性类型，描述符描述的特征值使用户能够理解该特征。

特征揭示了值代表的数据类型、值是否能够读取或写入以及如何配置要指示或通知的值，它还描述了值的含义。

特征具有以下组成部分：

1. 特征声明
2. 特征值
3. 特征描述符

图 11. 特征定义示例

特征声明是一种属性，其定义如下：

表 10. 特征声明

句柄属性	属性类型	属性值	属性权限
0xNNNN	0x2803 （特征属性类型的 UUID）	特征值属性（读取、广播、写入、写入但不响应、通知、指示等）。确定如何能够使用特征值或如何能够访问特征描述符	“只读，无验证，无授权”
		特征值属性句柄	
		特征值 UUID（16 或 128 位）	

特征声明包含特征值。该值是特征声明后的第一个属性：

表 11. 特征值

句柄属性	属性类型	属性值	属性权限
0xNNNN	0xuuuu – 16 位或 128 位（特征 UUID）	特征值	取决于更高层配置文件或应用

1.9.2 特征描述符类型

特征描述符用于描述特征值，以便为特征添加特定“含义”，使客户能够理解特征。有以下特征描述符可供使用：

1. 特征扩展属性：允许为特征添加扩展属性
2. 特征用户描述：使设备能够将文本字符串关联到特征
3. 客户端特征配置：如果特征可以通知或指示，为强制要求。客户端应用必须写入该特征描述符以使特征通知或指示成为可能（前提是特征属性允许通知或指示）
4. 服务器特征配置：可选描述符
5. 特征表达格式：它允许通过一些字段（例如，格式、指数、单位命名空间、描述）定义特征值表达格式，以便显示相关值（例如，°C 格式的温度测量值）
6. 特征聚合格式：可聚合多种特征表达格式

关于特征描述符的详细描述，请参考蓝牙规范。

1.9.3 服务属性类型

服务是特征的集合，共同为一个可实现的应用配置文件提供通用服务类型。例如，健康体温计服务包含温度测量值特征和每次测量的间隔时间特征。服务或主要服务可以引用被称为次要服务的其他服务。

服务的定义如下：

表 12. 服务声明

句柄属性	属性类型	属性值	属性权限
0xNNNN	0x2800 – “主要服务”的 UUID，或 0x2801 – “次要服务”的 UUID	0xuuuu – -16 位或 128 位（服务 UUID）	“只读， 无验证， 无授权”

服务包含服务声明，其他服务包含定义（可选）和特征定义。服务包含声明位于服务声明之后。

表 13. 包含声明

句柄属性	属性类型	属性值			属性权限
0xNNNN	0x2802（包含属性类型的 UUID）	包含服务属性句柄	结束组句柄	服务 UUID	“只读， 无验证，无授权”

“包含服务属性句柄”是所包含次要服务的属性句柄，“结束组句柄”是所包含次要服务中最后一个属性的句柄。

1.9.4 GATT 流程

通用属性配置文件（GATT）定义了一组发现服务、特征和相关描述符的标准流程以及它们的使用方法。

有以下流程可供使用：

- 发现流程（表 14. 发现流程和相关响应事件）
- 客户端发起的流程（表 15. 客户端发起的流程和相关响应事件）
- 服务器发起的流程（表 16. 服务器发起的流程和相关响应事件）

表 14. 发现流程和相关响应事件

步骤	响应事件
发现所有主要服务	按组读取响应
通过服务 UUID 发现主要服务	按类型值响应查找
查找包含的服务	按类型响应事件读取
发现服务的所有特征	按类型响应读取
通过 UUID 发现特征	按类型响应读取
发现所有特征描述符	查找信息响应

表 15. 客户端发起的流程和相关响应事件

步骤	响应事件
读取特征值	读取响应事件
通过 UUID 读取特征值	读取响应事件
读取长特征值	读取 BLOB 响应事件
读取多个特征值	读取响应事件
写入特征值，无响应	不生成事件
签名写入，无响应	不生成事件
写入特征值	写入响应事件
写入长特征值	准备写入响应 执行写入响应
可靠写入	准备写入响应 执行写入响应

表 16. 服务器发起的流程和相关响应事件

步骤	响应事件
通知	不生成事件
指示	确认事件

关于 GATT 流程和相关响应事件的详细描述，请参考第 5 节 参考文件中的蓝牙规范。

1.9.5 GATT 缓存

低功耗蓝牙服务和特征发现流程允许 GATT 客户端了解存储在服务器属性数据库表上的所有 GATT 服务器服务和特征，并通过 ATT 流程（读、写等）对其加以使用。不过这些流程相当地耗时费力。

有些设备通常不根据新的服务和特征更改其属性表，而只更改特征和描述符值。

其他一些设备可能会在其寿命期间添加新的服务/特征。

将 GATT 服务器属性数据库表上可能发生的更改通知 GATT 客户端的唯一方法是服务更改指示特征，该特征允许 GATT 服务器向匹配连接的 GATT 客户端发送特定指示，而该客户端又可以执行服务/特征发现以获得更新后的表。

该机制不提供客户端和服务器之间的任何同步，可能导致这样一种情况：GATT 客户端在接收到来自服务器的服务更改指示之前，在服务器属性数据库上启动 ATT 过程。

对于已连接但未绑定的设备，与 GATT 服务器属性表可能发生的变化保持一致的唯一安全机制，是在每个连接上执行费时费力的服务发现过程。

蓝牙规范 v5.1 定义两个新特征：

1. 数据库哈希
2. 客户端支持特性

这些特征允许客户端了解服务器 GATT 属性表是否更改了某些内容，即使两台设备之间没有绑定。

GATT 客户端更新服务器上客户端支持特性特征上的一个标志，以向服务器通知其支持数据库哈希特征。

数据库哈希特征存储一个 128 位值，这是一个从服务器属性表计算出来的 AES-CMAC 哈希值。服务器 GATT 属性数据库结构中的任何更改都会导致不同的哈希值。服务器有责任保持数据库哈希值始终最新。

数据库哈希特征允许客户端询问服务器其属性数据库是否发生了更改：每次 GATT 客户端连接到服务器时，都会立即执行读取该特征，以便确定服务器属性表是否发生了更改。

GATT 客户端可以缓存读取的数据库哈希值，以验证自上次连接以来数据库结构是否发生了更改。如此一来，仅在上次发现之后发生更改的情况下，客户端才执行服务和特征发现过程，从而在时间和节能方面增强了用户体验。

除了这两个特征，还定义了稳健缓存的概念，以便同步客户端和服务器的属性表视图。基本上，当客户端在收到服务更改指示之前尝试读取 GATT 服务器属性表时，如果 GATT 服务器属性表内容发生更改，则客户端可能得到不一致的数据。届时，服务器可以发送一个新的 ATT 错误代码（数据库不同步）向客户端告知不一致的问题。

当启用稳健缓存后，从服务器的角度来看，客户端可以处于两种状态：

1. 变更知晓状态
2. 变更不知晓状态

连接之后，没有信任关系的客户端的状态是变更知晓状态。相反，如果客户端已绑定，则状态与前一个连接相同（除非数据库发生了更改）。在这种情况下，客户端被认为是不知晓变更的。

服务器忽略来自不知晓变更的客户端的所有 ATT 命令；如果它收到 ATT 请求，则发送 ATT 错误响应，并将错误代码设置为数据库不同步。

一些事件可以将客户端的状态改为变更知晓状态：

1. 服务器接收到一个 ATT 确认，确认它先前发送的服务变更
2. 服务器在收到客户端的 ATT 请求之后发送“数据库不同”ATT 错误响应，然后从客户端接收另一个 ATT 请求。

1.10 通用访问配置文件（GAP）

蓝牙系统为所有的蓝牙设备定义了一个基础配置文件，叫做通用访问配置文件（GAP）。此配置文件定义了一个蓝牙设备所需具备的基本要求。

下表中描述了四种 GAP 配置文件的角色：

表 17. GAP 角色

角色(1)	说明	发送器	接收器	典型示例
广播设备	发送广播事件	M	O	发送温度值的温度传感器
观察者	接收广播事件	O	M	只接收和显示温度值的温度显示装置
外设	始终为从设备。 处于可连接广播模式。 支持所有 LL 控制流程；可选择加密或不加密	M	M	手表
中央设备	始终为主设备。 从不广播。 支持主动或被动扫描。支持所有 LL 控制流程；可选择加密或不加密	M	M	移动电话

1. *1.M = 强制；O = 可选*

在 GAP 中，定义了两个基本概念：

- GAP 模式：配置设备，使之以特定方式长时间工作。GAP 模式有四种类型：广播、可发现、可连接和可绑定类型
- GAP 规程：配置设备，使之在特定的时间段或有限时间内执行某一动作。有四种类型的 GAP 规程：监听、发现、连接和绑定流程

不同类型的可发现和可连接模式可以同时使用。定义的 GAP 模式如下：

表 18. GAP 广播器模式

模式	说明	注释	GAP 角色
广播模式	设备仅使用链路层广播通道和数据包广播数据（不在 AD 类型标记上设置任何位）	设备可使用监听规程检测广播数据	广播设备

表 19. GAP 可发现模式

模式	说明	注释	GAP 角色
非可发现模式	不能在 AD 类型标记上设置有限和一般可发现位	不能通过执行通用和有限发现规程的设备对其进行发现	外设
有限可发现模式	在 AD 类型标记上设置有限可发现位	允许约 30 s。由用户最近交互的设备使用。例如用户按下设备上的按钮	外设
一般可发现模式	在 AD 类型标记上设置一般可发现位	在设备希望成为可发现设备时使用。对可发现时间无限制	外设

表 20. GAP 可连接模式

模式	说明	注释	GAP 角色
不可连接模式	只能使用 ADV_NONCONN_IND 或 ADV_SCAN_IND 广播数据包	在广播时不能使用可连接广播数据包	外设

模式	说明	注释	GAP 角色
直接可连接模式	使用 ADV_DIRECT 广播数据包	由要快速连接中央设备的外设使用。只能使用 1.28 秒，并且需要外设和中央设备地址	外设
非定向可连接模式	使用 ADV_IND 广播数据包	在设备希望成为可连接设备时使用。由于 ADV_IND 广播数据包可以包含 AD 类型标记，因此设备可同时处于可发现和非定向可连接模式。 当设备进入连接模式或不可连接模式时，可连接模式终止	外设

表 21. GAP 可绑定模式

模式	说明	注释	GAP 角色
不可绑定模式	不允许与对端设备建立绑定	设备不保存密钥	外设
可绑定模式	设备接受来自中央设备的绑定请求		外设

第 1.3.1 节中定义了以下 GAP 规程：

表 22. GAP 监听流程

步骤	说明	注释	角色
监听规程	它允许设备搜索广播设备数据		观察者

表 23. GAP 发现流程

步骤	说明	注释	角色
有限可发现规程	用于在有限发现模式下发现外设	根据 AD 类型标记信息应用设备过滤	中央设备
通用可发现规程	用于在通用和有限发现模式下发现外设	根据 AD 类型标记信息应用设备过滤	中央设备
名称发现规程	用于从可连接设备检索“蓝牙设备名称”的流程		中央设备

表 24. GAP 连接流程

步骤	说明	注释	角色
自动连接建立规程	允许连接一个或多个处于定向可连接模式或非定向可连接模式的设备	使用白名单	中央设备

步骤	说明	注释	角色
通用连接建立规程	允许连接一组处于定向可连接模式或非定向可连接模式的已知对端设备	当在被动扫描期间检测到具有私有地址的设备时，它使用直接连接建立流程支持私有地址	中央设备
选择性连接建立规程	使用主机的选定连接配置参数与白名单中的一组设备建立连接	使用白名单，并按照该白名单进行扫描	中央设备
直接连接建立规程	使用一组连接间隔参数与特定设备建立连接	由通用和选择性规程使用	中央设备
连接参数更新规程	在连接期间更新使用的连接参数		中央设备
终止规程	终止 GAP 规程		中央设备

表 25. GAP 绑定流程

步骤	说明	注释	角色
绑定规程	在配对请求上设置了绑定位的情况下，启动配对过程		中央设备

关于 GAP 规程的详细描述，请参考蓝牙规范。

1.11 测向

使用低功耗蓝牙（LE）的经典位置应用（接近、信标...）基于接收信号强度（RSSI）测量这样一个简单概念。这可以确定两台设备是否位于彼此的范围内，并估算相关距离。

蓝牙规范 v5.1 定义了一项新功能，允许以高精度识别接收到的低功耗蓝牙数据包的方向。

有两种方法：

1. 到达角（AoA）
2. 离开角（AoD）。

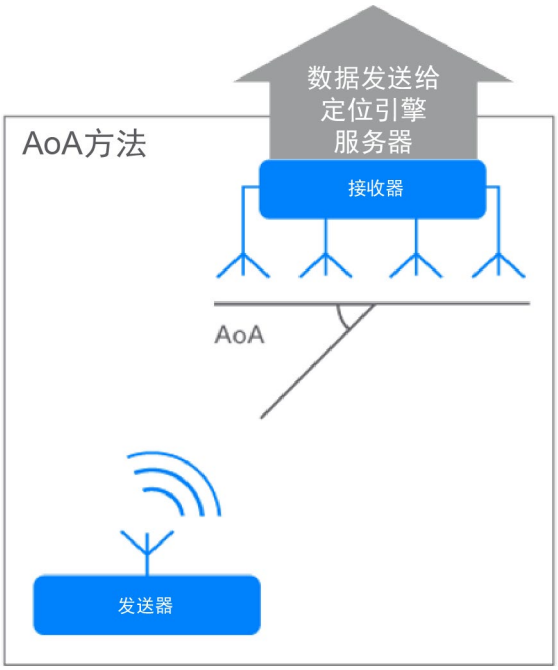
这两种方法都要求两台相互通信设备中的一台具有多天线阵列（AoA 是接收端，AoD 是发送端）。

1.11.1 通过到达角（AoA）进行测向

在 AoA 方法中，正在确定方向的发射（标签）设备使用单一天线发送测向信号。接收设备（定位器）具有多阵列天线，通过阵列天线检测信号相位差（由阵列中每个天线到发射天线的距离不同而引起）。接收设备在阵列中的有源天线之间切换，同时获取信号的 IQ 样本数据。接收设备能够对 IQ 样本数据应用特定算法，从而计算相对信号方向。

该功能允许将多个应用场景作为实时资产跟踪应用。

图 12. 到达角（AoA）



在接收（定位器）设备上，需要天线阵列，而且多个标签的角度计算需要持续处理能力。发射（标签）设备需要单一天线，必须只支持发送执行 IQ 数据采样所需信息的能力。

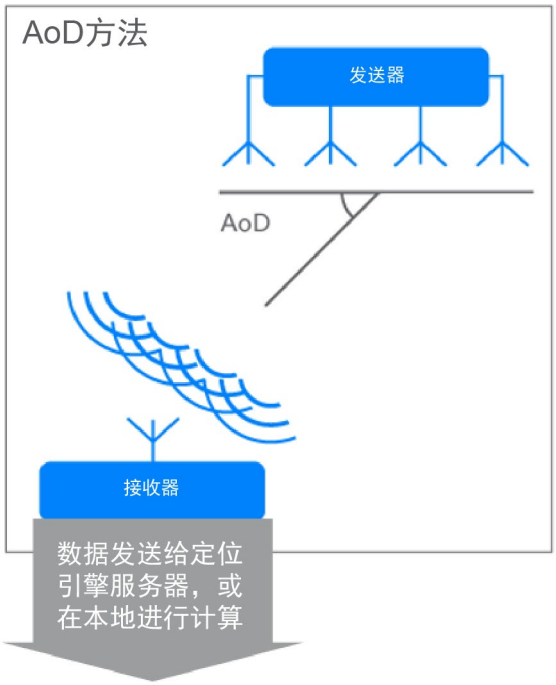
1.11.2 通过离开角（AoD）进行测向

在 AoA 方法中，正在确定方向的发射设备（通常是锚定设备）使用天线阵列发送特殊的测向信号。接收设备（例如，移动电话）配有单一天线。

当发送设备通过天线阵列发送多个信号到接收设备时，接收设备获得 IQ 样本。接收设备能够对 IQ 样本应用特定算法，从而计算相对信号方向。

该功能可为多个应用场景提供室内定位系统解决方案。

图 13. 离开角（AoD）



发射设备需要使用天线阵列发送多天线定位信号。它必须支持发送执行 IQ 数据采样所需信息的能力。

接收设备需要单个天线，而接收信号的角度计算可能需要 FPU（浮点处理单元）和一些存储资源。

1.11.3 同相（I）和正交（Q）

测向使用同相和正交（IQ）采样法来测量在特定时间入射到天线上的无线电波的相位。

采用 AoA 方法时，采样过程是在阵列中的每个天线上完成的（每次一个），并根据阵列的设计按特定顺序进行。

然后通过主机控制器接口（HCI）将采样数据提供给主机。IQ 样本应该与一些特定算法一起用于计算入射波的角度。这些算法没有在低功耗蓝牙核心规范中定义。

IQ 采样过程涉及链路层（LL）和主机控制器接口（HCI）层面的一些变更。在链路层层面，LL 数据包中新增了一个名为“constant tone extension”的字段。该字段提供一个恒定频率信号，可以对其执行 IQ 采样。该字段包含一个由多个 1 组成的序列，不进行白化处理，不用于 CRC 计算。

“Constant tone extension”可以在无连接和面向连接的场景中使用：

1. 在无连接场景中使用：需要周期性广播功能（因为采样过程中的确定性定时很重要），和在 AUX_SYNC_IND PDU 后面附加 CTE。
2. 在面向连接的场景中使用：定义了新的 PDU LL_CTE_REQ 和 LL_CTE_RSP。

1.12 BLE 配置文件和应用

服务集合一组特征并公开这些特征的行为（设备的操作内容，而不是设备如何使用特征）。服务不定义特征用例。用例决定了需要的服务（如何在设备上使用服务）。这是通过配置文件实现的，它定义了特定用例需要的服务：

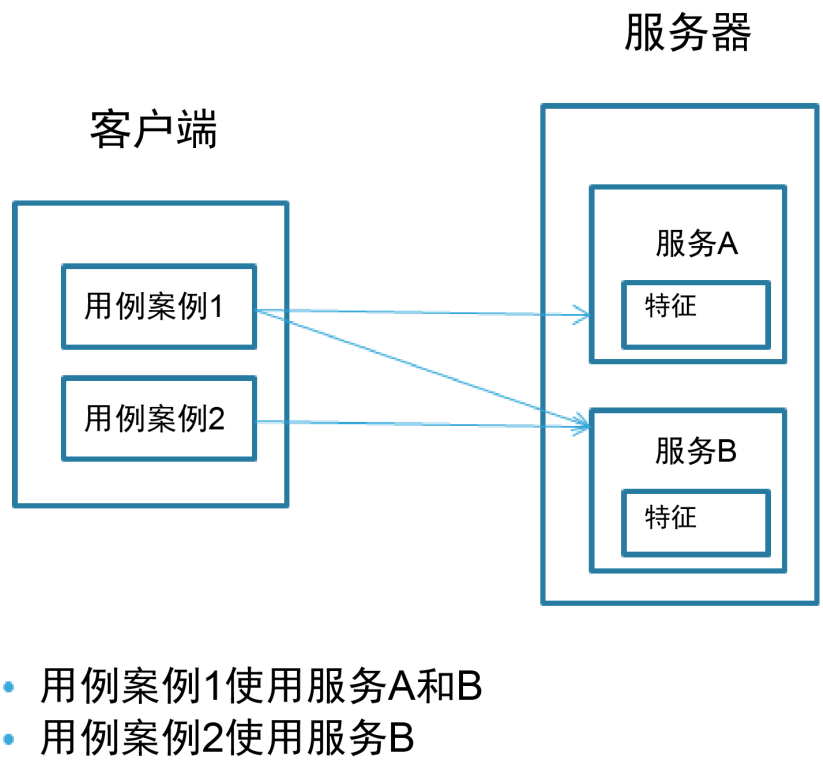
- 配置文件客户端实现用例
- 配置文件服务器实现服务

可以使用标准配置文件或专有配置文件。在使用非标准配置文件时，需要 128 位 UUID，并且必须是随机生成的。

目前，任何标准蓝牙 SIG 配置文件（服务和特征）均使用 16 位 UUID。可从以下 SIG 网页下载服务和特征规范及 UUID 分配：

- https://developer.bluetooth.org/gatt/services/Pages/ServicesHome.aspx
- https://developer.bluetooth.org/gatt/characteristics/Pages/CharacteristicsHome.aspx

图 14. 客户端和服务器配置文件



1.12.1 接近感测示例

本节将介绍接近感测的目标、工作原理和需要的服务：

目标

- 当一个设备由近及远，到一定距离时：
  - 触发警报

工作原理

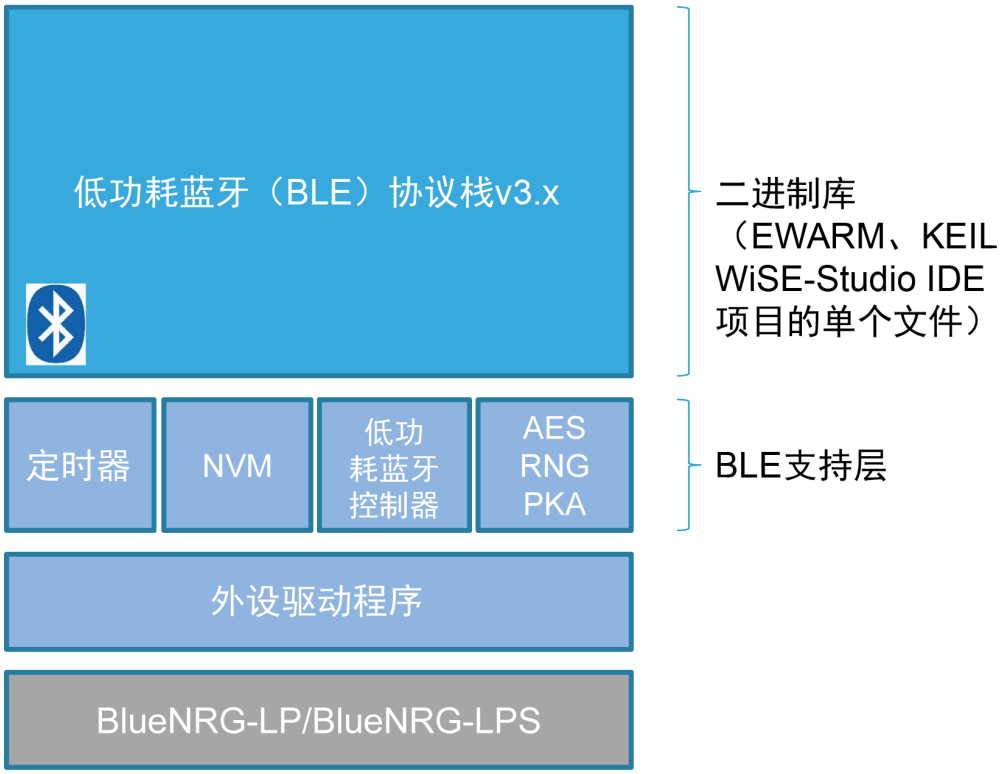
- 如果设备断开，则会导致警报
- 链路丢失警报：«链路丢失» 服务
  - 如果设备距离太远
  - 导致路径丢失警报：«即时警报»和«发送功率»服务
- «链路丢失» 服务
  - «警报级别» 特征
  - 行为：链路丢失时，按照设定的值（《警报级别》）进行警报。

- «即时警报» 服务
  - «警报级别» 特征
  - 行为：当写入时，即刻按照《警报级别》的值进行警报
- «发送功率» 服务
  - «发送功率» 特征
  - 行为：读取时，报告连接的当前发送功率。

2 低功耗蓝牙（BLE）协议栈 v3.x

低功耗蓝牙（BLE）协议栈 v3.x 提供了一种新架构，比以前的低功耗蓝牙（BLE）协议栈 v2.x 系列具有更多的优势。

图 15. 低功耗蓝牙（BLE）协议栈 v3.x 架构



新架构提供的以下新特性具有相关优势：

1. 代码更加模块化，可在隔离状态下测试：
   – 测试覆盖率提高了
2. 硬件相关的部分以源码形式提供：
   – 低功耗蓝牙协议栈外部的新睡眠定时器模块（Init API 和 tick API 将在用户主程序上调用）
   – 低功耗蓝牙协议栈外部的新 NVM 模块（Init API 和 tick API 将在用户主程序上调用）。

注意： *它可以轻松定制或修复错误*

3. 认证仅针对协议部分：
   – 它减少了协议栈版本的数量，因为硬件相关的问题大多被隔离在其他模块中
   – 它减少了认证的数量
4. 它实现更加灵活和稳健的无线电活动调度程序
   – 它使协议栈在一些阻延中断的操作上有更高的鲁棒性（比如，Flash 写操作，和/或用户禁用中断的操作）
5. 它降低了对实时性要求（中断处理程序中的代码更少）
   – 系统给应用程序更多时间

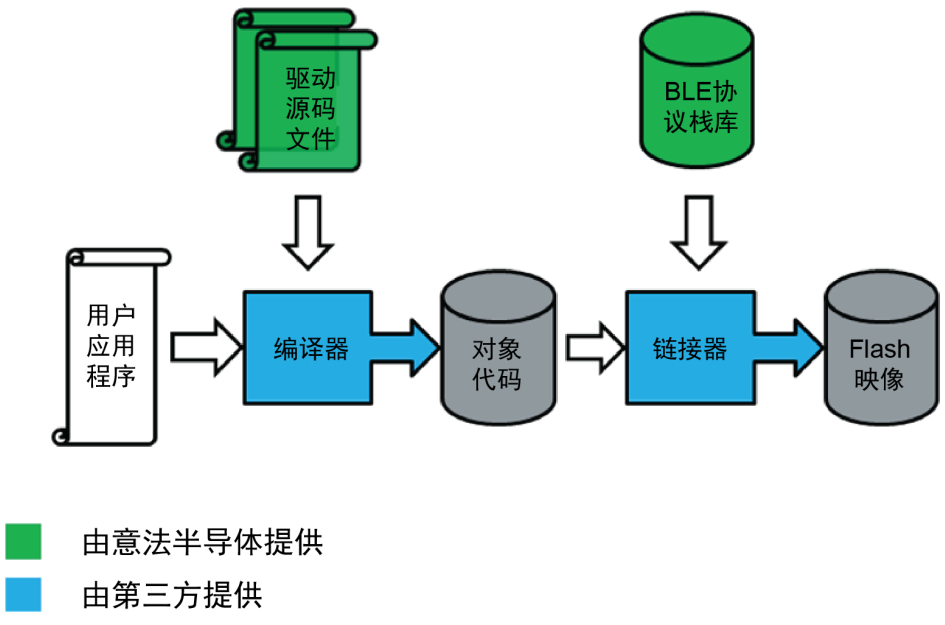
低功耗蓝牙（BLE）协议栈 v3.x 是采用二进制格式的标准 C 库，并提供用于控制意法半导体器件低功耗蓝牙功能的高级接口。低功耗蓝牙二进制库提供以下功能：

- 协议栈 API
  - 低功耗蓝牙（BLE）协议栈初始化
  - BLE 协议栈应用命令接口（以 hci_为前缀的 HCI 指令和以 aci_为前缀的厂商特定指令）
  - 睡眠定时器访问
  - BLE 协议栈状态机处理
- 协议栈事件回调
  - 向用户应用通知 BLE 协议栈事件
  - 睡眠定时器事件
- 面向无线电 IP 的中断处理程序

为能够访问 BLE 协议栈功能，只需用户应用程序：

- 调用相关 API
- 通过提供的协议栈回调处理预期事件
- 将 BLE 协议栈二进制库链接到用户应用程序，如图 16. BLE 协议栈参考应用中所述。

图 16. BLE 协议栈参考应用



注意：

1. *API 是由 BLE 协议栈库定义并由用户应用调用的 C 函数。*
2. *回调是由 BLE 协议栈库调用并由用户应用定义的 C 函数。*
3. *驱动源码文件是一组驱动程序（头文件和源文件），用于处理所有 BLE 设备外设（ADC、I2C、SPI、定时器、看门狗、GPIO、RTC、UART…）。*
4. *与低功耗蓝牙协议栈 v2.x 系列相比，修改了 BLE 无线电初始化结构和相关应用程序配置文件*
5. *与低功耗蓝牙协议栈 v2.x 系列相比，修改了以 BLE stack init 和 BLE stack tick 命名的低功耗蓝牙协议栈 APIx*
6. *与低功耗蓝牙协议栈 v2.x 系列相比，添加了用于 GAP 初始化、扫描、连接和广播 API /事件的新 GAP API 接口*
7. *与低功耗蓝牙协议栈 v2.x 系列相比，定义了新 GATT API/事件框架和接口*

2.1 BLE 协议栈库框架

BLE 协议栈库框架允许将指令发送至 BLE 协议栈，并且还提供了 BLE 事件回调的定义。

BLE 协议栈 API 利用并扩展了在蓝牙规范中定义的标准 HCI 数据格式。

提供的一组 API 支持以下命令：

- 蓝牙规范定义的控制器标准 HCI 指令
- 供应商特定的（VS）控制器 HCI 指令
- 供应商特定的（VS）主机 HCI 指令（L2CAP、ATT、SM、GATT、GAP）

参考低功耗蓝牙 API 接口框架位于意法半导体低功耗蓝牙设备 DK 软件包（针对相关 DK 平台）（请参阅第 5 节 参考文件）。

BlueNRG-LP/BlueNRG-LPS 设备的 BLE 协议栈库框架接口由以下头文件定义：

表 26. BLE 协议栈库框架接口

文件	说明	位置	注释
ble_status.h	BLE 协议栈错误代码的头文件	Middlewares\ST\Bluetooth_LE\inc	
bluenrg_lp_api.h	BLE 协议栈 API 的头文件	“	
bluenrg_lp_events.h	BLE 协议栈事件回调的头文件	“	
bluenrg_lp_gatt.h	BlueNRG-LP/ BlueNRG-LPS BLE GATT 的头文件	“	它提供新的 GATT 架构定义
stack_user_cfg.h	BLE 协议栈配置头文件	“	它为 BLE 协议栈 v3.x 提供了可用的配置选项
stack_user_cfg.c	BLE 协议栈配置文件	Middlewares\ST\Bluetooth_LE\inc	要包含在用户应用 IDE 项目中的源文件，用于支持 BLE 协议栈 v3.x 提供的 BLE 模块化方法

注意： *BLE 协议栈 v3.x 或更高版本提供了相关功能，用于在编译时根据用户特定应用场景启用/禁用以下 BLE 协议栈特性：*

1. 启用/禁用控制器隐私
2. 启用/禁用 LE 安全连接
3. 启用/禁用主设备的能力
4. 启用/禁用数据长度扩展（仅对支持数据长度扩展特性的设备有效）
5. 启用/禁用 LE 2M 和 LE CODED PHY 特性
6. 启用/禁用扩展广播和扫描特性
7. 启用/禁用 L2CAP 面向连接的数据服务特性（L2CAP-COS）
8. 启用/禁用“Constant tone extension”（适用时；从 BLE v3.1 或更高版本起）
9. 启用/禁用 LE 功率控制和路径损耗监测（从 BLE v3.1 或更高版本起）
10. 启用/禁用连接功能（从 BLE 协议栈 v3.1a 或更高版本起）。它配置了连接支持：
    - 如果禁用连接选项，则不支持连接；仅当主设备功能选项被禁用时，设备才是广播设备；仅当主设备功能选项被启用时，设备才是广播设备和观察者。
    - 如果启用连接选项，则支持连接；仅当主设备功能选项被禁用时，设备才作为广播设备或外设；如果主设备功能选项被启用，设备可以充当任意角色（广播设备、观察者、外设，以及中央设备）。

从低功耗蓝牙栈 v3.1a 或后续版本起，主设备功能选项链接到新的连接选项，如下所示：

1. 如果连接选项被禁用，观察者被禁用（主设备功能选项被禁用）或启用（主设备功能选项被启用）
2. 如果连接选项被启用，则观察者和中央设备被禁用（主设备功能选项被禁用）或启用（主设备功能选项被启用）。

模块化配置选项允许用户从可用的 BLE 协议栈的二进制库中去除某些功能，以减少片内 Flash 存储器使用空间。

下表提供了要添加或不添加的模块化选项列表，以满足一些典型的低功耗蓝牙配置需求：

1. 仅广播设备：发送广播 PDU 和扫描响应 PDU；接收和处理扫描请求 PDU。
2. 仅广播设备 + 观察者：发送广播 PDU、扫描请求 PDU 和扫描响应 PDU；接收和处理广播 PDU、扫描请求 PDU 和扫描响应 PDU。
3. 基本：所有模块化选项“OFF”希望能够作为外围设备连接（连接功能已启用）。
4. 基本 + DLE：所有模块化选项“OFF”希望具有连接和数据长度扩展功能。
5. 完全：所有模块化选项“ON”。

表 27. 模块化配置选项组合示例

-	仅广播设备[1]	仅广播设备和观察者[1] [2]	基本	基本 + DLE	满
控制器隐私	否	否	否	否	是
安全连接	否	否	否	否	是
中央/观察者功能	否	是	否	否	是
控制器数据长度扩展	否	否	否	是	是
控制器 2M，coded PHY	否	否	否	否	是
控制器扩展广播	否	否	否	否	是
L2CP Cos	否	否	否	否	是
控制器周期性广播	否	否	否	否	是
控制器 CTE（测向）	否	否	否	否	是

-	仅广播设备[1]	仅广播设备和观察者[1] [2]	基本	基本 + DLE	满
控制器 LE 功率控制	否	否	否	否	是
连接[3][4]	否	否	是	是	是

1. “仅广播设备”或“仅广播设备+观察者”角色还可以扩展其他模块化选项和相关功能，但以下功能除外：
   - *安全连接*
   - *L2CAP COS*
   - *功率控制*
2. 不支持“仅观察者”角色。
3. BLE 协议栈 v3.1a 或更高版本上支持该角色。这意味着只有 BLE 协议栈 v3.1a 或更高版本才支持“仅广播设备”（连接功能= OFF）或“仅广播设备+观察者”（连接功能= OFF，主设备功能= ON），以及可能的扩展。
4. ACI_GAP_INIT()，角色参数应根据连接支持和主角色模块化选项进行设置（使用可用的参数值或此类值“广播设备、外围设备、观察者、中央设备”的组合）。

通过选择一些预处理器选项，可以在软件开发框架上使用一些预定义的配置：

- BLE_STACK_FULL_CONF：所有描述的特性都已启用
- BLE_STACK_BASIC_CONF：所有描述的特性（除了连接选项）都被禁用
- BLE_STACK_SLAVE_DLE_CONF：只启用连接和数据长度扩展特性

在文件 stack_user_cfg.h 中定义的以下 BLE 协议栈预处理器配置选项用于激活/取消激活（1：启用；0：禁用）每个模块配置选项：

- CONTROLLER_MASTER_ENABLED: 中央/观察者 (1) 或仅观察者角色 (0) 取决于 CONNECTION_ENABLED 值(1/0)
- CONTROLLER_PRIVACY_ENABLED:控制器隐私
- SECURE_CONNECTIONS_ENABLED:LE 安全连接特性
- CONTROLLER_DATA_LENGTH_EXTENSION_ENABLED:数据长度扩展特性
- CONTROLLER_2M_CODED_PHY_ENABLED:LE 2M + Coded PHY
- CONTROLLER_EXT_ADV_SCAN_ENABLED:扩展广播和扫描特性
- L2CAP_COS_ENABLED:L2CAP COS
- CONTROLLER_PERIODIC_ADV_ENABLED:周期性广播和同步
- CONTROLLER_CTE_ENABLED:固定频率扩展信号
- CONTROLLER_POWER_CONTROL_ENABLED:LE 功率控制
- CONNECTION_ENABLED：连接功能

注意： *从 BLE 协议栈 v3.1 或更高版本开始，如果用户需要定义一组特定的配置选项，可以遵循以下步骤：*

1. *在一个名为 custom_ble_stack_conf.h 的头文件中定义模块化配置选项，并将其放置在应用程序 Inc 文件夹中*
2. *在应用项目上添加 BLE_STACK_CUSTOM_CONF 预处理器选项。*

下面是一组特定配置选项示例，这些选项可以通过 custom_ble_stack_conf.h 定义。

“仅广播设备”配置

```
#define CONTROLLER_MASTER_ENABLED (0U)
#define CONTROLLER_PRIVACY_ENABLED (0U)
#define SECURE_CONNECTIONS_ENABLED (0U)
#define CONTROLLER_DATA_LENGTH_EXTENSION_ENABLED (0U)
#define CONTROLLER_2M_CODED_PHY_ENABLED (0U)
#define CONTROLLER_EXT_ADV_SCAN_ENABLED (0U)
#define L2CAP_COS_ENABLED (0U)
#define CONTROLLER_PERIODIC_ADV_ENABLED (0U)
```

#define CONTROLLER_CTE_ENABLED (0U)
#define CONTROLLER_POWER_CONTROL_ENABLED (0U)
#define CONNECTION_ENABLED (0U)

注意： *始终支持广播设备功能，且不能关闭。*

“仅广播设备 + 观察者”配置

#define CONTROLLER_MASTER_ENABLED (1U)
#define CONTROLLER_PRIVACY_ENABLED (0U)
#define SECURE_CONNECTIONS_ENABLED (0U)
#define CONTROLLER_DATA_LENGTH_EXTENSION_ENABLED (0U)
#define CONTROLLER_2M_CODED_PHY_ENABLED (0U)
#define CONTROLLER_EXT_ADV_SCAN_ENABLED (0U)
#define L2CAP_COS_ENABLED (0U)
#define CONTROLLER_PERIODIC_ADV_ENABLED (0U)
#define CONTROLLER_CTE_ENABLED (0U)
#define CONTROLLER_POWER_CONTROL_ENABLED (0U)
#define CONNECTION_ENABLED (0U)

具有传统安全性的外围设备（即：基本配置已经通过 BLE_STACK_BASIC_CONF 预处理器选项可用）

#define CONTROLLER_MASTER_ENABLED (0U)
#define CONTROLLER_PRIVACY_ENABLED (0U)
#define SECURE_CONNECTIONS_ENABLED (0U)
#define CONTROLLER_DATA_LENGTH_EXTENSION_ENABLED (0U)
#define CONTROLLER_2M_CODED_PHY_ENABLED (0U)
#define CONTROLLER_EXT_ADV_SCAN_ENABLED (0U)
#define L2CAP_COS_ENABLED (0U)
#define CONTROLLER_PERIODIC_ADV_ENABLED (0U)
#define CONTROLLER_CTE_ENABLED (0U)
#define CONTROLLER_POWER_CONTROL_ENABLED (0U)
#define CONNECTION_ENABLED (1U)

注意： *如果 CONNECTION_ENABLED = 0（不支持连接功能），用户应当对其用户应用程序进行以下修改：*

- *在用户 IDE 项目中，定义了 BLE_STACK_CUSTOM_CONF 预处理器选项，并使用相关的 custom_ble_stack_config.h 头文件（CONNECTION_ENABLED = 0）*
- *在用户 IDE 项目中，gap_profile.c、gatt_profile.c 文件被排除在应用程序构建之外*
- *aci_gatt_srv_init() 不再调用（因为没有添加任何特征）*
- *aci_gap_init()，角色参数被定义为 GAP_BROADCASTER_ROLE（仅广播设备）或 GAP_BROADCASTER_ROLE|GAP_OBSERVER_ROLE（仅广播设备，观察者），具体取决值(0/1)*
- *Gap_profile_set_dev_name()不再调用*
- *aci_gap_set_advertising_configuration()，Discoverable_Mode 参数被定义为 GAP_MODE_BROADCAST*
- *ace_gap_set_advertising_data()上的 Advertising_Data 参数不设置通用可发现模式标志，因为它在广播设备模式中不受支持*

表 28. BLE 应用协议栈库框架接口

文件	说明	位置	注释
ble_const.h	它包括所需的 BLE 协议栈头文件	Middlewares\ST\BLE_Application\layers_inc	将包含在用户主应用中

文件	说明	位置	注释
bluenrg_lp_gap.h	GAP 协议层常量的头文件	“”	它包含在 ble_const.h 头文件中
bluenrg_lp_gatt_server.h	GATT 服务器常量的头文件	“”	它包含在 ble_const.h 头文件中
bluenrg_lp_hal.h	头文件，带 HAL	“”	它包含在 ble_const.h 头文件中
bluenrg_lp_l2cap.h	面向 l2cap 的头文件	“”	它包含在 ble_const.h 头文件中
bluenrg_lp_stack.h	BLE 协议栈初始化、时间片和睡眠定时器 API 的头文件	“”	将包含在用户主应用中
hci_const.h	包含 HCI 层的常量		
link_layer.h	链路层常量的头文件	“”	它包含在 ble_const.h 头文件中
sm.h	安全管理器常量的头文件	“”	它包含在 ble_const.h 头文件中
gap_profile.[ch]	通用访问配置文件服务（GAP）库的头文件和源文件	Middlewares\BLE_Application\Profiles	
gatt_profile.[ch]	通用属性配置文件服务（GATT）库的头文件和源文件	Middlewares\BLE_Application\Profiles	
att_pwrq.[ch]	ATT 的头文件和源文件准备写入队列实现库	Middlewares\BLE_Application\Queued_Write	

注意： *BLE 协议栈所需的 AES CMAC 加密功能在一个新的独立二进制库中提供：cryptolib\\cryptolib.a。该库也必须包含在用户应用的 IDE 项目中。*

2.2 BLE 协议栈事件回调

BLE 协议栈库框架提供一组事件与相关回调，用于向用户应用程序通知要处理的特定事件。

BLE 事件回调原型在头文件 bluenrg_lp_events.h 中定义。默认情况下，通过弱定义来定义所有回调（不对用户定义的事件回调名称进行检查，所以用户应仔细检查每个定义的回调是否与预期的函数名称一致）。

用户应用必须使用与特定应用场景一致的应用代码定义所使用的事件回调。

2.3 BLE 协议栈 init 和 tick API

必须对 BLE 协议栈 v3.x 进行初始化，以便适当配置某些与特定应用场景一致的参数。

在使用任何其他 BLE 协议栈 v3.x 功能以前，必须先调用以下 API：

```
BLE_STACK_Init(&BLE_STACK_InitParams);
```

BLE_STACK_InitParams 是包含设备内存和底层硬件配置数据的变量，并使用以下结构定义：

```
typedef struct {
      uint8_t* BLEStartRamAddress ;
      uint32_t TotalBufferSize ;
      uint16_t NumAttrRecord ;
      uint8_t MaxNumOfClientProcs;
      uint8_t NumOfLinks;
      uint16_t NumBlockCount ;
      uint16_t ATT_MTU;
      uint32_t MaxConnEventLength;
      uint16_t SleepClockAccuracy;
      uint8_t NumOfAdvDataSet;
      uint8_t NumOfAuxScanSlots;
      uint8_t NumOfSyncSlots;
      uint8_t WhiteListSizeLog2;
      uint16_t L2CAP_MPS;
      uint8_t L2CAP_NumChannels;
      uint8_t CTE_MaxNumAntennaIDs;
      uint8_t CTE_MaxNumIQSamples;
      uint16_t isr0_fifo_size;
      uint16_t isr1_fifo_size;
      uint16_t user_fifo_size;
      }
```

表 29. BLE 协议栈 v3.x 初始化参数

名称	说明	值
BLEStartRamAddress	根据 BLE_STACK_TOTAL_BUFFER_SIZE 宏为 GATT 数据库分配 RAM 缓冲区的起始地址	32 位对齐 RAM 区域
TotalBufferSize	BLE_STACK_TOTAL_BUFFER_SIZE 宏返回值，用于检查宏的正确性。 它根据无线电任务的数量、属性的数量、并发 GATT 客户端程序的数量、分配给数据包的内存块的数量、广播集的数量、辅助扫描槽位的数量、白名单的大小、支持的 L2CAP 连接导向通道的数量，来定义用于管理所有数据协议栈的总 RAM。	参照 \Middlewares\ST\Bluetooth_LE\Inc\bluenrg_lp_stack.h 文件
NumAttrRecord	GATT 数据库中可以存储的最大属性数（即特征声明数+特征值数+描述符数+服务数）	每个特征的最小属性数量是 2 个。
MaxNumOfClientProcs	最大并发客户端程序数	该值小于或等于 NumOfLinks
NumOfLinks	设备可以支持的最大同时无线任务数量。无线任务是一个内部链路层状态机，它处理特定的无线活动（连接、广播、扫描）。	无线控制器支持多达 128 个同时无线任务，但实际可用的最大值取决于可用的设备 RAM（NUM_LINKS 用于计算 BLE_STACK_TOTAL_BUFFER_SIZE）

名称	说明	值
NumBlockCount	为 BLE 协议栈数据包分配而分配的内存块数。	使用 bluenrg_lp_stack.h file: BLE_STACK_MBLOCKS_CALC()上提供的特定宏计算最小需求值
ATT_MTU	支持的最大数值 ATT_MTU size	支持的值范围为 23 到 1020 字节
MaxConnEventLength	设备处于从模式时的连接事件最长持续时间，以 625/256 µs (~2.44 µs)为单位	<= 4000（ms）
SleepClockAccuracy	睡眠时钟精度	ppm value
NumOfAdvDataSet	广播数据集的最大数量，仅当广播扩展功能启用后有效	请参见“表 69. 低功耗蓝牙协议栈活动和无线任务”
NumOfAuxScanSlots	辅助广播通道上用于扫描的最大槽位数，仅当广播扩展功能启用后有效	请参见“表 69. 低功耗蓝牙协议栈活动和无线任务”
NumOfSyncSlots	可同步的最大槽位数，仅当“周期性广播与同步特性”启用后有效。	0 - 如果“周期性广播”被禁用。 [1 - NumOfLinks-1]如果“周期性广播”被启用
WhiteListSizeLog2	白名单/解析列表大小，2 的对数	
L2CAP_MPS	L2CAP 层实体可以接受的有效负载数据的最大字节数	支持的值范围为 0 到 1024 字节
L2CAP_NumChannels	基于 LE credit 的流量控制模式下的最大通道数	支持的值范围为 0 到 255 字节
CTE_MaxNumAntennaIDs	在 CTE 连接导向模式下使用的天线方向图中天线 ID 的最大数。	[0x02-0x4B]如果设备启用并支持测向功能。 0：如果设备不支持测向功能。
CTE_MaxNumIQSamples	在 CTE 连接导向模式下使用的缓冲区中 IQ 样本的最大数量。	[0x09-0x52]如果设备启用并支持测向功能。 0：如果设备支持测向功能
isr0_fifo_size	用于由 ISR 产生的关键控制器事件（例如，rx 数据包）的内部 FIFO 的大小	默认值：256
isr1_fifo_size	用于由 ISR 产生的非关键控制器事件（例如，广播或 IQ 采样报告）的内部 FIFO 的大小	默认值：768
user_fifo_size	用于 ISR 之外产生的控制器和主机事件的内部 FIFO 的大小	默认值：1024

注意： *BlueNRG-LP、BlueNRG-LPS 设备软件开发工具包（STSW_BNRGLP_DK）上提供了一款名为“Radio Init Wizard”的特定工具。“Radio Init Wizard”允许根据应用程序需求配置所描述的参数值，并获得与 BLE 协议栈所需的 RAM 相关的概述。用户还可以评估每个参数对总体 RAM 内存占用的影响。这将根据可用的设备 RAM 内存调整每个无线电初始化参数（即 NumOfLinks、ATT_MTU…）的值。*

2.4 BLE 协议栈 v3.x 应用配置

在设备初始化阶段，当意法半导体低功耗蓝牙设备上电后，必须在低功耗蓝牙设备控制器寄存器上定义一些特定参数，以便定义以下配置：

- 低速晶振源：外部 32 kHz 振荡器，内部 RO
- SMPS：开启或关闭（如果开启：2.2 µH、1.5 µH 或 10 µH SMPS 感应器）

BlueNRG-LP/BlueNRG-LPS 应用配置参数在文件 system_bluenrg_lp.c 上通过以下配置表定义：

表 30. 应用配置预处理器选项

预处理器选项	说明
CONFIG_HW_LS_XTAL	低速晶振源：外部 32 kHz 振荡器
CONFIG_HW_LS_RO	低速晶振源：内部 RO
CONFIG_HW_SMPS_10uH	使用 10 uH 电感使能 SMPS
CONFIG_HW_SMPS_2_2uH	使用 2.2 uH 电感使能 SMPS
CONFIG_HW_SMPS_1_5uH	使用 1.5 uH 电感使能 SMPS
CONFIG_HW_SMPS_NONE	禁用 SMPS
CONFIG_HW_HSE_TUNE	HSE 电容器配置：[0-63]作为值范围

2.5 BLE 协议栈 tick 功能

BLE 协议栈 v3.x 提供了一个特殊 API BTLE_StackTick()，当 BLE 协议栈活动正在进行时（通常在主应用程序的 while 循环中），必须对其进行调用才能处理内部 BLE 协议栈状态机。

BLE_STACK_Tick()函数执行所有主机协议栈层的处理，必须定期执行，以处理传入链路层数据包和主机层程序。通过此函数来调用所有协议栈回调。

如果使用低速环形振荡器来替代 LS 晶振，则此函数也执行 LS RO 校准，因此必须在每次系统唤醒时至少调用一次，以保持 500 ppm 的精度（如果作为主设备，则必须至少保持 500 ppm 的精度）。

注意： *当 BTLE_StackTick()正在运行时，不能调用其他任何 BLE 协议栈函数。例如，如果在中断程序中可能调用 BLE 协议栈函数，则必须在执行 BLE_STACK_Tick()期间禁用该中断。*

示例：如果可能在 UART ISR 中调用协议栈函数，则应使用以下代码：

```
NVIC_DisableIRQ(UART_IRQn);
BLE_STACK_Tick();
NVIC_EnableIRQ(UART_IRQn);
```

3 使用 BLE 协议栈 v3.x 设计应用

本节提供关于如何使用 BLE 协议栈 v3.x 二进制库设计和实现低功耗蓝牙应用的信息和代码示例。

用户在实现 BLE 协议栈 v3.x 应用时，必须执行一些基础且常用的步骤：

1. 初始化阶段和应用程序主循环
2. BLE 协议栈事件回调设置
3. 服务和特征配置（在 GATT 服务器上）
4. 创建连接：可发现、可连接模式与流程
5. 安全（配对和绑定）
6. 服务和特征发现
7. 特征通知/指示、写入、读取
8. 基本/典型错误条件描述

注意： *在后面的章节中，将使用一些用户应用“定义”，以便识别低功耗蓝牙设备的角色（中央设备、外设、客户端，以及服务器）。此外，BlueNRG-LP 设备用作运行 BLE 协议栈 v3.x 应用的参考设备。必要时，会突出标明任何依赖于具体设备的部分。*

表 31. 低功耗蓝牙设备角色的用户应用定义

定义	说明
GATT_CLIENT	GATT 客户端角色
GATT_SERVER	GATT 服务器角色

3.1 初始化阶段和应用程序主循环

要正确配置运行 BLE 协议栈 v3.x 的 BLE 设备，需要执行以下主要步骤：

1. 初始化 BLE 设备向量表、中断优先级、时钟：SystemInit() API
2. 初始化 IO 口为省电模式：BSP_IO_Init(); API 。以及初始化用于 I/O 通信的串行通信通道，作为调试和实用信息输出：BSP_COM_Init() API。
3. 初始化 BLE 控制器和定时器模块：BLECNTR_InitGlobal()和 HAL_VTIMER_Init(&VTIMER_InitStruct) API；
4. 初始化 NVM、PKA、RNG 和 AES 模块：BLEPLAT_Init()、PKAMGR_Init()、RNGMGR_Init() 和 AESMGR_Init() API
5. 初始化 BLE 协议栈：BLE_STACK_init(&BLE_STACK_InitParams) API
6. 配置 BLE 设备公共地址（前提是使用公共地址）：aci_hal_write_config_data() API
7. 初始化 BLE GATT 层：aci_gatt_srv_init() API
8. 根据所选设备角色初始化低功耗蓝牙 GAP 层：aci_gap_init(“role”) API
9. 设置合适的安全 I/O 能力和验证要求（前提是使用低功耗蓝牙安全）
10. 如果设备是 GATT 服务器，定义需要的服务、特征和特征描述符
11. 添加一个 while(1)循环，调用定时器模块 tick API HAL_VTIMER_Tick()、BLE 协议栈 tick API BTLE_StackTick()、NVM 模块 tick API NVMDB_Tick()和一个特定的用户 tick 处理程序（处理用户操作/事件）。此外还需调用 HAL_PWR_MNGR_Request() API，以启用设备睡眠模式并保留低功耗蓝牙无线电操作模式。

以下伪代码示例描述了需要的初始化步骤：

```
int main(void)
{
	WakeupSourceConfig_TypeDef wakeupIO;
	PowerSaveLevels stopLevel;
	BLE_STACK_InitTypeDef BLE_STACK_InitParams = BLE_STACK_INIT_PARAMETERS;

	/* System initialization function */
	if (SystemInit(SYSCLK_64M, BLE_SYSCLK_32M) != SUCCESS)
	{
		* Error during system clock configuration take appropriate action */
		while(1);
	}
	/* Configure IOs for power save modes */
	BSP_IO_Init();
	/* Configure I/O communication channel */
	BSP_COM_Init(BSP_COM_RxDataUserCb);
	/* Enable clock for PKA and RNG IPs used by BLE radio */
	LL_AHB_EnableClock(LL_AHB_PERIPH_PKA|LL_AHB_PERIPH_RNG);

	BLECNTR_InitGlobal();

	/* Init Virtual Timer module */
	HAL_VTIMER_InitType VTIMER_InitStruct = {HS_STARTUP_TIME, INITIAL_CALIBRATION,
			CALIBRATION_INTERVAL};
	HAL_VTIMER_Init(&VTIMER_InitStruct);
	/* Init NVM module */
	BLEPLAT_Init();
	if (PKAMGR_Init() == PKAMGR_ERROR)
	{
		while(1);
	}
	if (RNGMGR_Init() != RNGMGR_SUCCESS)
	{
		while(1);
	}
	/* Init the AES block */

	AESMGR_Init();

	ret = BLE_STACK_Init(&BLE_STACK_InitParams);
	if (ret != BLE_STATUS_SUCCESS) {
		printf("Error in BLE_STACK_Init() 0x%02x\r\n", ret);
		while(1);
	}
	/* Device Initialization: GATT and GAP Init APIs.
	It could add services and characteristics (if it is a GATT server)
	It could also initialize its state machine and other specific drivers (
	i.e. leds, buttons, sensors, …) */
	ret = DeviceInit();
	if (ret != BLE_STATUS_SUCCESS) {
		while(1);
	}
	/* Set wakeup sources if needed through the wakeupIO. Variable */
	/* No wakeup sources: */
	wakeupIO.IO_Mask_High_polarity = 0;
	wakeupIO.IO_Mask_Low_polarity = 0;
	wakeupIO.RTC_enable = 0;
	wakeupIO.LPU_enable = 0;

	while(1)
	{
		/* Timer tick */
		HAL_VTIMER_Tick();
		/* BLE stack tick */
		BLE_STACK_Tick();
		/* NVM manager tick */
		NVMDB_Tick();
		/* Application Tick: user application where application
		state machine is handled*/
		APP_Tick();
		/* Power Save management*/
		/* It enables power save modes with wakeup on radio operating timings
		(adverting, connections intervals) */
		HAL_PWR_MNGR_Request(POWER_SAVE_LEVEL_STOP_NOTIMER,wakeupIO, &stopLevel);
	}/* while (1) */

} /* end main() */
```

注意：

1. *BlueNRG_Stack_Init_params 变量定义了 BLE 协议栈事件回调中所述的栈初始化参数*
2. *要处理 BLE 协议栈事件，必须调用 BLE_STACK_Tick()。*
3. *APP_Tick()只是一个依赖于应用的函数，该函数根据应用工作场景来处理用户应用程序状态机。*
4. *HAL_PWR_MNGR_Request（POWER_SAVE_LEVEL_STOP_NOTIMER、wakeupIO、&stopLevel）使能设备硬件断电模式，不带定时器：设备处于深度睡眠状态。所有外设和时钟源均关闭。仅可以从 GPIO 进行唤醒（请参阅参考部分的 BlueNRG-LP、BlueNRG-LPS 数据手册）。值得注意的是，必须在应用程序的 main while 循环中调用具有指定参数的此 API，以便 BLE 设备能够进入在 BLE 协议栈广播或连接间隔上具有唤醒源的睡眠模式。如果不调用，则低功耗蓝牙设备始终处于节能运行模式（除非调用这一指定 API，否则 BLE 协议栈不会自动进入睡眠模式）。用户应用可使用 HAL_PWR_MNGR_Request() API 选择其中一种受支持的 BLE 设备硬件节电模式，并设置相关的唤醒源和睡眠超时（如果适用）。HAL_PWR_MNGR_Request() API 将来自应用的低功耗请求与无线操作模式相结合，选择适用于当前情况的最佳低功耗模式。*

   在无线模块与应用请求之间进行这种协商的目的是避免丢失无线通信数据。
5. *有关 HAL_PWR_MNGR_Request() API 和 BLE 设备低功耗模式的更多信息，请参考本文档末尾的第 5 节 参考文件中的相关应用笔记。*
6. *当未在 aci_gap_init() API 上启用隐私或基于主机的隐私时，为标准 GAP 服务保留的最后一个属性句柄为 0x000F，若在 aci_gap_init() API 上启用基于控制器的隐私，则为 0x0011。*

表 32. GATT，GAP 服务处理

服务	开始句柄	结束句柄	服务 UUID
属性配置文件服务	0x0001	0x0008	0x1801
通用访问配置文件（GAP）服务	0x0009	0x000F	0x1800

表 33. GATT，GAP 特征处理

默认服务	特征	句柄属性	特征属性	特征值句柄	特征 UUID	特征值长度（字节）
属性配置文件服务	-	0x0001	-	-	-	-
-	服务更改	0x0002	指示	0x0003	0x2A05	4
-	客户端支持特性	0x0005	读/写	0x0006	0x2B29	1
-	数据库哈希 GATT	0x0007	读	0x0008	0x2B2A	16
通用访问配置文件（GAP）服务	-	0x0009	-	-	-	-
-	设备名称	0x000A	读取|写入，无响应|写入|验证签名写入	0x000B	0x2A00	8
-	外观	0x000C	读取|写入，无响应|写入|验证签名写入	0x000D	0x2A01	2

默认服务	特征	句柄属性	特征属性	特征值句柄	特征 UUID	特征值长度（字节）
-	外设首选的连接参数	0x000E	读/写	0x000F	0x2A04	8
-	中央设备地址解析(1)	0x0010	无需身份验证或授权即可读取。 不可写	0x00011	0x2AA6	1

1. *仅在 aci_gap_init() API 上启用基于控制器的隐私（0x02）时添加。*

aci_gap_init()角色参数值如下：

表 34. aci_gap_init()角色参数值

参数	角色参数值	注释
角色	0x01：外设 0x02：广播设备 0x04：中央设备 0x08：广播者	角色参数可以是支持的任意值的按位或（同时支持多个角色）
Privacy_Type	0x00 用于禁用隐私； 0x01 用于启用隐私； 0x02 用于启用基于控制器的隐私	指定是否启用隐私以及启用了哪一隐私
device_name_char_len		它允许指定设备名称特征的长度
Identity_Address_Type	0x00：公共地址 0x01：静态随机地址	指定用作标识地址的地址

关于该 API 和相关参数的完整描述，请参考第 5 节 参考文件中的 BLE 协议栈 API 和事件文档。

3.1.1 BLE 地址

BLE 协议栈支持以下设备地址：

- 公共地址
- 随机地址
- 私有地址

公共 MAC 地址（6 字节- 48 位地址）唯一标识了低功耗蓝牙设备，它们由电气和电子工程师协会（IEEE）定义。

公共地址的前 3 个字节标识了发布标识符的公司，称为组织唯一标识符（OUI）。组织唯一标识符（OUI）是一种从 IEEE 购买的 24 位数字。该标识符唯一标识了某个公司，并可以为公司独家使用特定 OUI 预留一组可能的公共地址（最多 2^24 个，来自公共地址的其余 3 个字节）。

当之前分配的地址组的至少 95%已被使用时，任何组织/公司均可以请求一组新的 6 字节地址（最多 2^24 个，通过特定 OUI 提供可能的地址）。

如果用户想要在程序中使用其自定义的 MAC 地址，则必须将其存储在特定设备上仅用于存储 MAC 地址的 Flash 位置。然后，在设备上电时，必须通过调用特定协议栈 API 将该 MAC 地址写入射频模块中。

用于设置 MAC 地址的低功耗蓝牙 API 指令为 aci_hal_write_config_data()

在开始任何 BLE 操作之前（在协议栈初始化 API BLE_STACK_Init()之后），应将 aci_hal_write_config_data()指令发送到 BLE 设备。

以下伪代码示例描述了如何设置公共地址：

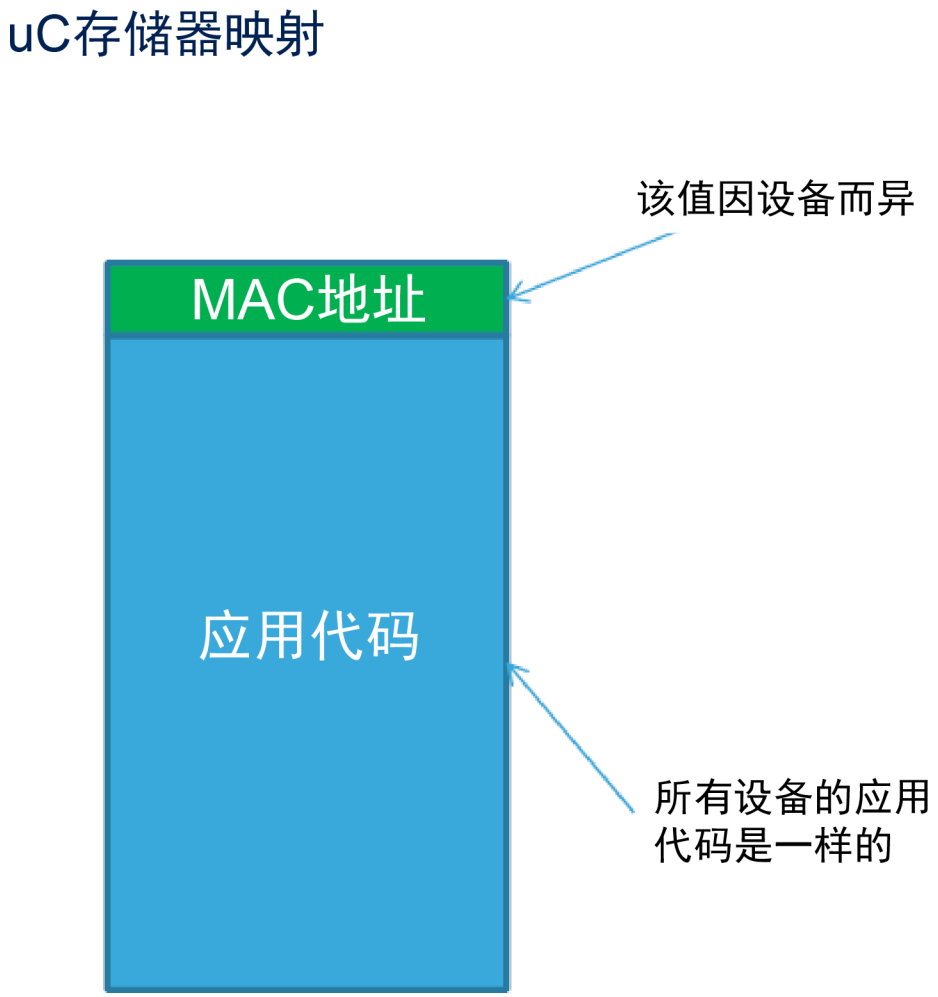
```
uint8_t bdaddr[] = {0x12, 0x34, 0x00, 0xE1, 0x80, 0x02};
ret=aci_hal_write_config_data(CONFIG_DATA_PUBADDR_OFFSET,CONFIG_DATA_PUBAD DR_LEN, bdaddr);
if(ret) PRINTF("Setting address failed.\n")}
```

在 BLE 产品生产时，需要将设备 MAC 地址存储在内部 flash 的指定位置。

用户在写入其应用时，可以假设 MAC 地址位于设备的已知特定 MAC Flash 位置。在生产期间，可通过 SWD 使用客户 Flash 映像为微控制器编程。

第二步可能包括生成唯一的 MAC 地址（即从数据库读取），并将 MAC 地址存储在已知 MAC Flash 位置。

图 17. MAC 地址存储



BlueNRG-LP/BlueNRG_LPS 设备没有有效的预分配 MAC 地址，但有唯一的序列号（对用户为只读）。该唯一序列号是一个 8 字节值，存储在地址 0x10001EF0：它作为两个字存储在地址 0x10001EF0 和 0x10001EF4。

注意： *以下实用工具 API 可以访问 BlueNRG-LP、BlueNRG-LPS 唯一序列号：*

```
uint32_t UID_word0;
uint32_t UID_word1;


/* Get first serial ID 4 bytes at serial id base address (0x10001EF0) */
UID_word0 = LL_GetUID_Word0();


/* Get last serial ID 4 bytes at serial id base address + 4 */
UID_word1 = LL_GetUID_Word1();
```

静态随机地址是在设备第一次启动时生成并被写入 Flash 的指定位置。Flash 上的值是设备使用的实际值：用户每次复位设备时，协议栈都会检查专用 Flash 区域是否存在有效数据并使用该数据（FLASH 上的特殊有效标记用于标识是否存在有效数据）。如果用户执行了批量擦除，则会删除存储值（包括标记），下一次设备重启时又会生成新的 MAC 地址，并存储到 Flash 的指定位置上。

当启用了隐私时，按照低功耗蓝牙规范使用私有地址。欲了解私有地址的更多信息，请参考“安全管理器（SM）”。

3.1.2 设置发送功率

在初始化阶段，用户还可以使用以下 API 选择发送功率：

aci_hal_set_tx_power_level(En_High_Power,PA_Level);

下面是将无线发送功率设置为 0 dBm 输出功率的伪代码示例：

ret= aci_hal_set_tx_power_level (0, 25);

En_High_Power 参数允许从两种可能的电源配置中选择一种。在每种配置中，设置了不同的 SMPS 输出电压水平。PA_Level 参数用于选择功率放大器的输出电平，粒度约为 1dB。该输出电平取决于 SMPS 输出电压。

表 35. 发射功率等级，En_High_Power = 0（SMPS 电压@ 1.4 V）和表 36. 发射功率等级，En_High_Power = 1（SMPS 电压@ 1.9 V）针对设备上两种配置中的每个电平提供预期的 TX 输出功率。

表 35. 发射功率等级，En_High_Power = 0（SMPS 电压@ 1.4 V）

PA 级别	预期的发送功率（dBm）
0	OFF (1)
1	-21
2	-20
3	-19
4	-17
5	-16
6	-15
7	-14
8	-13
9	-12
10	-11
11	-10
12	-9
13	-8
14	-7
15	-6
16	-6

PA 级别	预期的发送功率（dBm）
17	-4
18	-3
19	-3
20	-2
21	-2
22	-1
23	-1
24	0
25	0
26	1
27	2
28	3
29	4
30	5
31	6

1. 如果 PA 电平设为 0，PA 被关闭。然而，在射频引脚上仍然可以测得非常低的输出功率。

表 36. 发射功率等级，En_High_Power = 1（SMPS 电压 @ 1.9 V）

PA 级别	预期的发送功率（dBm）
0	OFF (1)
1	-19
2	-18
3	-17
4	-16
5	-15
6	-14
7	-13
8	-12
9	-11
10	-10
11	-9
12	-8
13	-7
14	-6
15	-5
16	-4
17	-3
18	-3
19	-2
20	-1

PA 级别	预期的发送功率（dBm）
21	0
22	1
23	2
24	3
25	8
26	8
27	8
28	8
29	8
30	8
31	8

1. 如果 PA 电平设为 0，PA 被关闭。然而，在射频引脚上仍然可以测得非常低的输出功率。

注意：当使用 SMPS OFF 配置时，输出功率水平取决于施加在 VFBSD 引脚上的电压。在这种情况下，用户必须用实际测量值填充 miscutil.c 文件（在 Middlewares\ST\hal\Src 目录下）中的查找表。用正确的值填充查找表非常重要，因为蓝牙协议栈使用该值正确地将 dBm 转换为 PA 电平，反之亦然。

3.2 BLE 协议栈 v3.x GATT 接口

3.2.1 BLE 协议栈 v3.x vs BLE 协议栈 v2.x

新的 ATT/GATT 组件旨在优化内存占用和使用情况。为了达成此结果，GATT 组件避免分配静态缓冲区。用户应用程序分配这些资源，并根据需要将其提供给协议栈库。BLE 堆栈不是在 BLE 协议栈缓冲区内为属性值分配空间，而是要求应用程序对属性值进行读和写操作。然后由应用程序决定是否需要存储这些值以及如何对其进行存储和检索。例如，与传感器的实时数据有关联的特征可能不需要静态缓冲区来存储属性值，因为数据是根据需要从传感器读取的。类似地，控制外部组件（例如 LED）的控制点属性可能不需要为属性值提供静态缓冲区。

BLE GATT 栈 v2.x 在 RAM 内存中分配属性，而 BLE 协议栈 v3.x ATT/GATT 组件尽可能地避免这种内存分配，提供一种基于 C 语言结构的配置文件注册机制，可以在需要时存储在 Flash 存储器中。这些结构还有助于减少内存分配，并且比 v2.x 更轻量。需要存储在易失性存储器中的组件被收集到一个在 RAM 中占用 8 字节的结构中。

BLE 协议栈 3.x ATT/GATT 组件还支持蓝牙稳健缓存功能。这是在协议栈内部处理的，不需要应用程序的支持。

3.2.2 GATT 服务器

GATT 服务器负责存储属性，这些属性构成 GATT 数据库中的一个配置文件。

GATT 配置文件供应用程序或其他配置文件使用，并定义如何使用包含的属性来获取某些信息。

3.2.2.1 *服务*

服务是完成某个功能或特性的数据和相关行为的集合。服务定义可以涵盖所包含的服务和特征。

有两种类型的服务：

- **主要服务**是公开设备主要可用功能的服务
- **次要服务**是仅打算包含在主要服务或其他次要服务之外的服务

使用下面的 ble_gatt_srv_def_t C 语言结构定义服务：

```
typedef struct ble_gatt_srv_def_s {
    ble_uuid_t uuid;
    uint8_t type;
    uint16_t group_size;
      struct {
          uint8_t incl_srv_count;
          struct ble_gatt_srv_def_s **included_srv_pp;
      } included_srv;
      struct {
          uint8_t chr_count;
          ble_gatt_chr_def_t *chrs_p;
      } chrs;
} ble_gatt_srv_def_t
```

该结构代表了一个带有以下属性的服务：

- uuid：服务的 16 位蓝牙 UUID 或 128 位 UUID，称为服务 UUID
- 类型：表示该服务是主要服务（BLE_GATT_SRV_PRIMARY_SRV_TYPE，值为 0x01）还是次要服务（BLE_GATT_SRV_SECONDARY_SRV_TYPE，值为 0x02）
- group_size：可选字段，表示该服务组可以添加多少个属性。如果将其设置为 0，则意味着不定义大小，并且不限制添加到服务中的属性数量。保留相同数量的句柄，以便稍后可以向服务添加新属性
- included_srv：可选字段，是包含的服务列表
- chrs：可选字段，是包含的特征列表。如果存在一个或多个特征，则在服务注册时注册这些特征及其描述符（如果有）

例如，考虑一个由主要服务组成的 GAP 配置文件，其 16 位 UUID 等于 0x1800，定义其 C 结构是

```
static ble_gatt_srv_def_t gap_srvc = {
    .type = BLE_GATT_SRV_PRIMARY_SRV_TYPE,
    .uuid = BLE_UUID_INIT_16(0x1800),
    .chrs = {
          .chrs_p = gap_chrs,
          .chr_count = 2U,
    },
};
```

注意： *如果没有显式初始化，则静态变量和全局变量（及其字段，前提是存在结构）将初始化为 0。*

要在 GATT 数据库中注册一个服务，使用 aci_gatt_srv_add_service 函数：

aci_gatt_srv_add_service(&gap_srvc);

在检索分配的属性句柄时，使用 ace_gatt_srv_get_service_handle

函数：uint16_t gap_h = aci_gatt_srv_get_service_handle（&gap_srvc）；

如果需要，还可以在运行时使用 aci_gatt_srv_rm_service 函数删除已注册的服务：aci_gatt_srv_rm_service(gap_h)；

注意： *用于服务定义结构的内存在此类服务保持活跃的所有时间内都保持有效。*

3.2.2.2 *特征*

特征用于公开设备值：例如，公开温度值。

使用下面的 ble_gatt_chr_def_t C 语言结构定义特征：

```
typedef struct ble_gatt_chr_def_s {
    uint8_t properties;
    uint8_t min_key_size;
    uint8_t permissions;
    ble_uuid_t uuid;
    struct {
        uint8_t descr_count;
        ble_gatt_descr_def_t *descrs_p;
    } descrs;
    ble_gatt_val_buffer_def_t *val_buffer_p;
} ble_gatt_chr_def_t;
```

该结构代表一个带有以下属性的特征：

- 属性：位字段，用于决定如何使用特征值或如何访问特征描述符。
- min_key_size：表示访问特征值所需的最小加密密钥大小。此参数仅在请求对该属性进行加密时使用（请参阅“permission”字段），在这种情况下，其值必须大于或等于 7 且小于或等于 16。
- 权限：这是一个位字段，指示如何访问该特征：
  - BLE_GATT_SRV_PERM_NONE (0x00)：表示不需要权限即可访问特征值
  - BLE_GATT_SRV_PERM_AUTHEN_READ (0x01)：表示读取特征值需要经过认证配对（即启用了 MITM 保护）
  - BLE_GATT_SRV_PERM_AUTHOR_READ (0x02)：指示应用程序必须授权访问链路上的设备服务，然后才能授予特征值读取权限
  - BLE_GATT_SRV_PERM_ENCRY_READ (0x04)：表示读取特征值需要加密链路。访问该特征值的最小加密密钥大小必须通过 min_key_size 字段指定
  - BLE_GATT_SRV_PERM_AUTHEN_WRITE (0x08)：表示写入特征值需要经过认证配对（即启用了 MITM 保护）
  - BLE_GATT_SRV_PERM_AUTHOR_WRITE (0x10)：指示应用程序必须授权访问链路上的设备服务，然后才能授予特征值写入权限
  - BLE_GATT_SRV_PERM_ENCRY_WRITE (0x20)：表示写入特征值需要加密链路。访问该特征值的最小加密密钥大小必须通过 min_key_size 字段指定
- uuid：该字段是一个 16 位蓝牙 UUID 或 128 位 UUID，用来描述特征值的类型
- descrs：该可选字段是特征描述符的列表。如果存在一个或多个描述符，则在特征注册时注册这些描述符
- val_buffer_p：可选字段。如果设置该字段，则指向存储特征值的已分配缓冲区。如未设置（例如设为 NULL），则协议栈会发出一个事件，请求对特征值进行读或写操作（参见 aci_gatt_srv_read_event()和 aci_gatt_srv_write_event()）。

例如，考虑一下 GAP 配置文件的设备名称特征，它具有读/写访问权限，无特定安全权限，16 位 UUID 等于 0x2A00（因为不需要加密，所以未设置 min_key_size）：

```
static ble_gatt_chr_def_t gap_device_name_chr = {
.properties = BLE_GATT_SRV_CHAR_PROP_READ,
.permissions = BLE_GATT_SRV_PERM_NONE,
.uuid = BLE_UUID_INIT_16(0x2A00),
.val_buffer_p=(ble_gatt_val_buffer_def_t *)\ &gap_device_name_val_buff,
};
```

为了注册定义的特征，可以使用 aci_gatt_srv_add_char 函数：

Aci_gatt_srv_add_char（&gap_device_name_chr, gap_p）；

作为替代，可以在注册服务的同时将特征添加到服务中。在本例中，特征列表通过 ble_gatt_srv_def_t 结构的 chrs 字段传递给协议栈。

作为服务和特征，有一个函数用于检索分配的属性句柄（aci_gatt_srv_get_char_decl_handle）和在运行时删除注册的特征（aci_gatt_srv_rm_char）。

注意： *用于特征定义结构的内存在此类特征保持活跃的所有时间内都保持有效。*

3.2.2.3 *描述符*

特征描述符用于包含与特征值相关的信息。定义了一组标准的特征描述符，供应用程序使用。应用程序还可以将附加的特征描述符定义为特定于配置文件。

使用下面的 ble_gatt_descr_def_t C 语言结构定义特征描述符：

```
typedef struct ble_gatt_descr_def_s {
    uint8_t properties;
    uint8_t min_key_size;
    uint8_t permissions;
    ble_uuid_t uuid;
    ble_gatt_val_buffer_def_t *val_buffer_p;
} ble_gatt_descr_def_t;
```

该结构代表一个带有以下属性的特征描述符：

- 属性：位字段，用于决定如何访问该特征描述符。BLE_GATT_SRV_DESCR_PROP_READ（0x01）位使能描述符读取，而 BLE_GATT_SRV_DESCR_PROP_WRITE（0x02）位使能描述符写入
- min_key_size：表示访问特征描述符所需的最小加密密钥大小。此参数仅在请求对该属性进行加密时使用（请参阅“permission”字段），在这种情况下，其值必须大于或等于 7 且小于或等于 16
- 权限：位字段，用于指示如何访问该特征描述符：
  - BLE_GATT_SRV_PERM_NONE，（0x00）表示不需要权限即可访问特征描述符
  - BLE_GATT_SRV_PERM_AUTHEN_READ，（0x01）表示读取特征描述符需要经过认证配对（即启用了 MITM 保护）
  - BLE_GATT_SRV_PERM_AUTHOR_READ，（0x02）表示必须先对链路授权，然后才能读取特征描述符
  - BLE_GATT_SRV_PERM_ENCRY_READ，（0x04）表示读取特征描述符需要先经过加密进行配对
  - BLE_GATT_SRV_PERM_AUTHEN_WRITE，（0x08）表示对特征描述符进行写操作需要先经过认证配对（即启用了 MITM 保护）
  - BLE_GATT_SRV_PERM_AUTHOR_WRITE，（0x10）表示必须先对链路授权，然后才能对特征描述符进行写操作
  - BLE_GATT_SRV_PERM_ENCRY_WRITE,，（0x20）表示对特征描述符进行写操作需要先经过加密进行配对
- uuid：16 位蓝牙 UUID 或 128 位 UUID，用于描述特征描述符的类型
- val_buffer_p：可选字段。如果设置该字段，则指向存储特征描述符值的已分配缓冲区。如未设置（例如，设为 NULL），则协议栈会发出一个事件，请求对特征描述符值进行读或写操作（参见 aci_gatt_srv_read_event()和 aci_gatt_srv_write_event()）。

例如，为了定义一个具有读访问权限的描述符，16 位 UUID 值设置为 0xAABB，无安全权限，使用以下结构：

```
static ble_gatt_descr_def_t chr_descr = {
.uuid = BLE_UUID_INIT_16(0xAABB),
    .properties = BLE_GATT_SRV_DESCR_PROP_READ,
    .permissions = BLE_GATT_SRV_PERM_NONE,
};
```

为了在数据库中注册定义的描述符，使用 aci_gatt_srv_add_char_desc 函数：

aci_gatt_srv_add_char_desc（&chr_descr, chr_handle）；

其中 chr_handle 是包含此描述符的特征的属性句柄。此外，对于描述符，还有一个函数可以检索其属性句柄（aci_gatt_srv_get_descriptor_handle）和删除描述符（aci_gatt_srv_rm_char_desc）本身。

通过指定 ble_gatt_chr_def_t 的 descrs 字段，还可以将描述符与相关特征一起添加。

由于客户端特征配置描述符（CCCD）在配置文件中非常常见，因此提供了一些辅助宏对其进行定义：

- BLE_GATT_SRV_CCCD_DECLARE：声明 CCCD 值缓冲区和描述符字段。通常在一个特征只有 CCCD 作为唯一描述符时使用
- BLE_GATT_SRV_CCCD_DEF_STR_FIELDS：填充 CCCD 的描述符结构字段。当一个特征有多个 CCC 描述符来填充描述符定义数组的字段时，可对其进行使用

注意： *用于特征描述符定义结构的内存在此类描述符保持活跃的所有时间内都保持有效。*

3.2.2.4 *值缓冲区*

值缓冲区是一个保存特征值和特征描述符值的结构。该结构存储缓冲区信息，在包含结构的整个生命周期内保持有效。值缓冲区结构由 ble_gatt_val_buffer_def_t type 定义：

```
typedef struct ble_gatt_val_buffer_def_s {
    uint8_t op_flags;
    uint16_t val_len;
    uint16_t buffer_len;
    uint8_t *buffer_p;
} ble_gatt_val_buffer_def_t;
```

该结构具有以下字段：

- op_flags：位字段，用于在写入值之后启用特定行为
  - BLE_GATT_SRV_OP_MODIFIED_EVT_ENABLE_FLAG（0x01）：使能在客户端改变该值时生成 aci_gatt_attribute_modified_event 事件
  - BLE_GATT_SRV_OP_VALUE_VAR_LENGTH_FLAG（0x02）：表示该值为可变长度
- val_len，存储值长度。如果 op_flags 字段中未设置 BLE_GATT_SRV_OP_VALUE_VAR_LENGTH_FLAG 位，则忽略
- buffer_len，buffer_p 指针所指向的缓冲区长度。对于长度固定的特征，这是特征值的长度
- buffer_p，指向值缓冲区的指针。

例如，要定义一个最大为 10 字节的可变长度值缓冲区，使用以下代码：

```
uint8_t buffer[10];
ble_gatt_val_buffer_def_t val_buffer = {
    op_flags = BLE_GATT_SRV_OP_VALUE_VAR_LENGTH_FLAG,
    buffer_len = 10,
    buffer_p = buffer,
};
```

如果应用程序需要以 2 字节的值填充值缓冲区（例如，0x0101），则可直接通过缓冲区为其值赋予地址并设置实际长度：

```
memset(val_buffer.buffer_p, 0x01, 2);
val_buffer.val_len = 2;
```

在远程设备发送读取请求以检索其值之前，协议栈不会意识到这样的值更新。

如果特征具有固定长度，则 ble_gatt_val_buffer_def_t 结构可以定义为常量。

```
const ble_gatt_val_buffer_def_t val_buffer = {
      buffer_len = 2,
      buffer_p = buffer,
};
```

此外，如果该值不能更改（即，只读访问），则 buffer_p 所指向的缓冲区也可以声明为常量。

const uint8_t buffer[2] = {0x01, 0x01};

如果值是动态计算的（例如，温度），则不需要值缓冲区：如果特征或描述符 C 结构的 val_buffer_p 字段未设置（即，设置为 NULL），则生成一些事件来访问该值：

- 当远程设备需要读取特征值或描述符时，生成 aci_gatt_srv_read_event
- 当远程设备需要写入特征值或描述符时，生成 aci_gatt_srv_write_event

注意： *用于值缓冲区定义的内存在该缓冲区处于活跃状态的所有时间内都保持有效。*

3.2.2.5 *GATT 服务器数据库 API*

下一段包含可用于操作 GATT 服务器数据库的函数列表。

表 37. GATT 服务器数据库 API

API	说明
aci_gatt_srv_add_service	该函数将提供的服务添加到数据库中
aci_gatt_srv_rm_service	该函数将提供的服务从数据库中移除
aci_gatt_srv_get_service_handle	该函数检索分配给使用提供的定义结构注册的服务的属性句柄
aci_gatt_srv_add_include_service	该函数添加提供的 include 服务
aci_gatt_srv_rm_include_service	该函数将提供的 include 服务从数据库中移除
aci_gatt_srv_get_include_service_handle	该函数检索分配给 include 服务的属性句柄
aci_gatt_srv_add_char	该函数将提供的特征添加到数据库中
aci_gatt_srv_rm_char	该函数将提供的特征从数据库中移除。所有包含的属性（特征值和描述符）都被相应地删除
aci_gatt_srv_get_char_decl_handle	该函数检索分配给使用提供的定义结构注册的特征的属性句柄
aci_gatt_srv_add_char_desc	该函数将提供的描述符添加到数据库中
aci_gatt_srv_rm_char_desc	该函数将提供的描述符从数据库中移除
aci_gatt_srv_get_descriptor_handle	该函数检索分配给特征描述符的属性句柄，其中、特征描述符使用提供的结构体定义进注册

3.2.2.6 *示例*

下面的示例旨在讲解如何定义配置文件。

GATT 配置文件

该示例说明如何实现和注册 GATT 配置文件。

注意： *完整的 GATT 服务已经在 gatt_profile.c 中实现。这里描述一个更简单的实现。*

该配置文件由一个主要服务组成，该服务具有一个 16 位 UUID（设置为 0x1801）和服务更改特征（设置了指示属性位）。为了支持特征指示，客户端拥有特征配置描述符。为了声明该描述符，可以使用 BLE_GATT_SRV_CCCD_DECLARE 宏。该宏具有以下参数：

BLE_GATT_SRV_CCCD_DECLARE(NAME, NUM_CONN, PERM, OP_FLAGS)

其中：

- NAME 分配用于识别该 CCCD 的名称
- NUM_CONN 是所支持的目标应用程序并发连接数
- PERM，描述符权限的位字段
- OP_FLAGS 是描述符值缓冲区的位字段。

然后，（例如）声明可以如下所示：

```
BLE_GATT_SRV_CCCD_DECLARE(gatt_chr_srv_changed,
                                        GATT_SRV_MAX_CONN,
                                        BLE_GATT_SRV_PERM_NONE,
                                        BLE_GATT_SRV_OP_MODIFIED_EVT_ENABLE_FLAG);
```

现在声明特征值缓冲区，因为它必须作为 ble_gatt_chr_def_t 结构的 val_buffer_p 进行提供：

```
uint8_t srv_chgd_buff[4];
const ble_gatt_val_buffer_def_t srv_chgd_val_buff = {
      .buffer_len = 4U,
      .buffer_p = gatt_chr_srv_changed_buff,
};
```

声明了描述符和特征值缓冲区之后，即可声明服务更改特征：

```
static ble_gatt_chr_def_t gatt_chr = {
.properties = BLE_GATT_SRV_CHAR_PROP_INDICATE,
.permissions = BLE_GATT_SRV_PERM_NONE,
.uuid = BLE_UUID_INIT_16(0x2A05),
.val_buffer_p = &srv_chgd_val_buff,
.descrs = {
.descrs_p =
&BLE_GATT_SRV_CCCD_DEF_NAME(gatt_chr_srv_changed),
.descr_count = 1U,
},
};
```

如上所述，该特征在属性位字段中设置了指示位，无安全权限，UUID 设置为 0x2A05，还有一个描述符。

现在可以声明 GATT 服务：

```
static ble_gatt_srv_def_t gatt_srvc = {
      .type = BLE_GATT_SRV_PRIMARY_SRV_TYPE,
      .uuid = BLE_UUID_INIT_16(0x1801),
      .chrs = {
            .chrs_p = &gatt_chr,
            .chr_count = 1,
      },
};
```

要注册整个配置文件，只注册服务：所有包含的特征及其描述符都是自动注册的。使用 aci_gatt_srv_add_service 注册服务：

aci_gatt_srv_add_service(&gatt_srvc);

血糖

考虑以下数据库：

表 38. 示例数据库

句柄属性	属性类型	UUID	属性	注释
0x0001	主要服务	0x1808		Glucose 服务
	包含的服务			包含的电池服务
	特征	0x2A18	读取，指示，扩展属性	血糖测量特征
	描述符			CCCD
	描述符			扩展属性描述符
0x1000	次要服务	0x180F		电池服务
	特征	0x2A19	读	电池电量特征

开始定义电池服务及其特征：

```
uint8_t battery_level_value;
const ble_gatt_val_buffer_def_t battery_level_val_buff = {
      .buffer_len = 1,
      .buffer_p = &battery_level_value,
};

ble_gatt_chr_def_t batt_level_chr = {
.properties = BLE_GATT_SRV_CHAR_PROP_READ,
.permissions = BLE_GATT_SRV_PERM_NONE,
.uuid = BLE_UUID_INIT_16(0x2A19),
.val_buffer_p = &battery_level_val_buff,
};

ble_gatt_srv_def_t battery_level_srvc = {
      .type = BLE_GATT_SRV_SECONDARY_SRV_TYPE,
      .uuid = BLE_UUID_INIT_16(0x180F),
      .chrs = {
            .chrs_p = &batt_level_chr,
            .chr_count = 1,
      },
};
```

定义血糖配置文件：

```
uint8_t ext_prop_descr_buff[2];
ble_gatt_val_buffer_def_t ext_prop_descr_val_buff = {
.buffer_len = 2,
.buffer_p = ext_prop_descr_buff,
};

BLE_GATT_SRV_CCCD_BUFFER_DECLARE(glucose_mes, MAX_CONN, 0);

ble_gatt_descr_def_t glucose_chr_descrs[] = {
{
      BLE_GATT_SRV_CCCD_DEF_STR_FIELDS(glucose_mes, MAX_CONN,
                                                    BLE_GATT_SRV_CCCD_PERM_DEFAULT),
},
};
ble_gatt_chr_def_t glucose_mes_chr = {
.properties = BLE_GATT_SRV_CHAR_PROP_READ | BLE_GATT_SRV_CHAR_PROP_INDICATE |
BLE_GATT_SRV_CHAR_PROP_EXTENDED_PROP,
.permissions = BLE_GATT_SRV_PERM_NONE,
.uuid = BLE_UUID_INIT_16(0x2A18),
.descrs = {
.descrs_p = &BLE_GATT_SRV_CCCD_DEF_NAME(glucose_chr_descrs),
.descr_count = 2U,
},
};

static ble_gatt_srv_def_t *incl_srv_pa[1] = {&battery_level_srvc};
ble_gatt_srv_def_t glucose_srvc = {
.type=BLE_GATT_SRV_PRIMARY_SRV_TYPE,
.uuid = BLE_UUID_INIT_16(0x1808),
.group_size = 0x1000,
};
```

如图所示，glucose_srvc 设置了 group_size 字段：这就允许在 0x1000 属性句柄处注册电池服务。此服务不包括任何特征或包含的服务。本例仅展示如何逐一注册要素。

首先，注册血糖服务：

aci_gatt_srv_add_service(&glucose_srvc);

这是一个主要服务，其 16 位 UUID 设为 0x1808。

注册电池服务及其特征：

aci_gatt_srv_add_service（&battery_level_srvc）；

这是一个次要服务，其 16 位 UUID 设为 0x180F（包含一个特征）。服务和特征均已注册。

现在，在血糖服务中包含电池服务并注册其特征和描述符。为此，需要用分配的属性句柄来获取血糖和电池服务：

```
uint16_t glucose_srvc_handle =
aci_gatt_srv_get_service_handle(&glucose_srvc);
uint16_t battery_srvc_handle =
aci_gatt_srv_get_service_handle(&battery_level_srvc);
```

将电池包含到血糖服务中：

```
aci_gatt_srv_include_service(glucose_srvc_handle,
battery_srvc_handle);
```

注意： *在向服务注册任何特征之前，将包含的服务添加到服务中。此步骤必须完成，因为 include 服务属性放在服务声明属性之后，任意特征声明属性之前。*

要注册血糖测量特征及其描述符，使用 aci_gatt_srv_add_char()函数，如下所示：

aci_gatt_srv_add_char(&glucose_mes_chr, glucose_srvc_handle);

血糖测量特征和所包含的描述符已经添加完毕。

3.2.2.7 *服务器发起的流程*

服务器发起的流程用于将数据发送到已订阅并可接收数据的远程客户端。服务器发起的流程有两种。

- **通知：**此流程的适用场景 - 服务器已经配置为将特征值通知给客户端，却不期望返回任何通知已被成功接收的确认
- **指示：**此流程的适用场景 - 服务器已经配置为向客户端指示一个特征值，期望返回指示已被成功接收的确认

为这两种流程提供一个 API：aci_gatt_srv_notify()。

表 39. aci_gatt_srv_notify 参数

类型	参数	说明
uint16_t	Connection_Handle	连接句柄，已经为其请求了通知
uint16_t	Attr_Handle	要通知的属性句柄
uint8_t	标志	通知标志： • 0x00：发送通知 • 0x01：发送 flushable 通知 • 0x02：发送指示
uint16_t	Val_Length	所提供值缓冲区的长度
uint8_t *	Val_p	指向包含要通知的值的缓冲区的指针

Flags 参数指示发送的消息类型。例如，要将句柄为 0x13 的属性值通知给连接句柄为 0x0801 的客户端，使用以下代码。

```
uint8_t value = 1;
ret = aci_gatt_srv_notify(0x0801, 0x13, 0, 1, &value);
```

如果客户端将通知位设置为特征的 CCCD，则发送通知，否则返回 BLE_STATUS_NOT_ALLOWED 错误。

3.2.2.8 *属性值读/写*

如前一节中所述，协议栈可以通过各自的值缓冲区访问特征值和描述符。如果在定义结构中没有设置这样的缓冲区（val_buffer_p = NULL），则协议栈将生成一个事件，要求应用程序对相关的值缓冲区进行操作。有两个事件：aci_gatt_srv_read_event()和 aci_gatt_srv_write_event()。对于“queued write”，协议栈并无队列对其进行存储：如果应用程序想要支持“queued write”，则其必须实现队列来存储每个“prepare write”。因此，协议栈为每个接收到的“prepare write”请求生成 aci_att_srv_prepare_write_req_event()事件，以便应用程序可以对其进行存储。当收到“execute write”请求后，生成 aci_att_srv_exec_write_req_event()事件。

aci_gatt_srv_read_event()

如果服务器接收到一个属性值的读取操作，且协议栈没有直接访问该值，则生成该事件。此事件的后面必须是 aci_gatt_srv_resp()，用于传递属性的值。

表 40. aci_gatt_srv_read_event 参数

类型	参数	说明
uint16_t	Connection_Handle	连接句柄，接收读取请求
uint16_t	Attr_Handle	要读取的属性句柄
uint16_t	Data_Offset	开始读取值的偏移量

aci_gatt_srv_write_event()

如果服务器接收到一个属性值的写入操作，且不能访问属性值缓冲区，则生成该事件。如果 Resp_Needed 为 1，此事件的后面必须是 aci_gatt_srv_resp()。

表 41. aci_gatt_srv_write_event 参数

类型	参数	说明
uint16_t	Connection_Handle	连接句柄，接收写入请求
uint8_t	Resp_Needed	如果值为 1，需要调用 aci_gatt_serv_resp()。收到 ATT 请求时，会发生这种情况。ATT 命令不需要响应（该参数设为 0）
uint16_t	Attribute_Handle	要写入的属性句柄
uint16_t	Data_Length	要写入的数据长度
uint8_t *	数据	要写入的数据

aci_att_srv_prepare_write_req_event()

当服务器收到“prepare write”请求时，会生成此事件。它携带存储在应用层的已接收数据。此事件的后面必须是 aci_gatt_srv_resp()，将应用程序写入的值传回协议栈。

表 42. aci_att_srv_prepare_write_req_event 参数

类型	参数	说明
uint16_t	Connection_Handle	标识连接的连接句柄
uint16_t	Attribute_Handle	属性句柄，为其接收写入请求
uint16_t	Data_Offset	开始写入值的偏移量
uint16_t	Data_Length	数据字段长度
uint8_t *	数据	要写入的数据

aci_att_srv_exec_write_req_event()

当服务器收到“execute write”请求时，会生成此事件。应用程序必须处理该事件，以写入或刷新所有存储的“prepare write”请求，具体取决于 Flag 值。此事件的后面必须是 aci_gatt_srv_resp()。

表 43. aci_att_srv_exec_write_req_event 参数

类型	参数	说明
uint16_t	Connection_Handle	标识连接的连接句柄
uint8_t	标志	• 0x00 - 取消所有已准备写入 • 0x01 - 立即写入所有挂起的准备值

aci_gatt_srv_resp()

当协议栈生成先前的事件时，应用程序必须决定允许（并执行）或拒绝所请求的操作。为了将应用程序的选择结果通知协议栈并传递所请求的数据（如果有“read”请求或“prepare write”请求），提供了一个函数：aci_gatt_srv_resp()。该函数用于关闭远程客户端启动的读或写事务。

注意： *此函数在收到事件后 30 秒内执行，否则会发生 GATT 超时。*

表 44. aci_gatt_srv_resp 参数

类型	参数	说明
uint16_t	Connection_Handle	标识连接的连接句柄
uint16_t	Attribute_Handle	属性句柄，为其发出响应命令
uint8_t	Error_Code	请求引起错误响应的原因（使用 ATT 错误代码之一， 参见[1]第 3 卷 F 部分表 3.4）
uint16_t	Val_Length	值字段的长度
uint8_t *	Val	下列情况下的响应数据： • “read”请求 • “prepare write”请求 对于其他请求，该参数可以为 NULL

注意： *函数返回时不再需要 Val 所指向的数据，可以将其释放。*

aci_gatt_srv_read_event() 示例

下面的示例代码展示如何实现 aci_gatt_srv_read_event()句柄。

```
void aci_gatt_srv_read_event(uint16_t Connection_Handle,
                             uint16_t Attribute_Handle,
                             uint16_t Data_Offset)
{
    uint16_t val_len;
    uint8_t attr_error_code;
    uint8_t val_buff[4];
    attr_error_code = BLE_ATT_ERR_NONE;
    if (Attribute_Handle == 0x16)
    {
        /** Attribute is mapped on a GPIO value */
        val_len = 1;
        gpio_get(GPIO_01, val_buffer[0]);
    }
        else if (Attribute_Handle == 0x19)
    {
        /** Fill buffer with some custom data */
        val_len = 4;
        memset(val_buffer, 2, 4);
    }
    else
    {
        val_len = 0;
        attr_error_code = BLE_ATT_ERR_UNLIKELY;
    }
    aci_gatt_srv_resp(Connection_Handle, Attribute_Handleg is that , attr_error_code, val_len,
val_buff);
}
```

代码管理属性句柄 0x16 和 0x19。句柄 0x16 被映射到 GPIO 值上，而句柄 0x19 返回 4 字节。如果 Error_Code 设为 0，aci_gatt_srv_resp()将使用给定的数据生成读取响应，否则它将发送错误响应。

aci_gatt_srv_write_event()示例

下面的示例展示如何管理 aci_gatt_srv_write_event()事件。在本例中，对属性句柄 0x16 进行写操作会改变 GPIO 状态。

```
void aci_gatt_srv_write_event(uint16_t Connection_Handle,
                              uint8_t Resp_Needed,
                              uint16_t Attribute_Handle,
                              uint16_t Data_Length,
                              uint8_t Data[])
{
    uint16_t buffer_len;
    uint8_t *buffer, att_err;

    buffer_len = 0;
    buffer = NULL;
    if (Data_Length != 1)
    {
        att_err = BLE_ATT_ERR_INVAL_ATTR_VALUE_LEN;
    }
    else if (Attribute_Handle != 0x16)
    {
        att_err = BLE_ATT_ERR_UNLIKELY;
    }
    else if ((Data[0] != 0) && (Data[0] != 1))
    {
        att_err = BLE_ATT_ERR_VALUE_NOT_ALLOWED;
    }
    else
    {
        /** Set GPIO value */
        att_err = BLE_ATT_ERR_NONE;
        gpio_set(GPIO_01, Data[0];
    }
    if (Resp_Needed == 1U)
    {
        aci_gatt_srv_resp(Connection_Handle, Attribute_Handle,
    att_err, buffer_len, buffer);
    }
}
```

aci_att_srv_prepare_write_req_event()示例

下面的示例展示如何实现 aci_att_srv_prepare_write_req_event()句柄。该函数使用 ATT_pwrq 模块存储接收到的“prepare write”，但是可以使用任何能够处理此类用例的队列。

```
void aci_att_srv_prepare_write_req_event(uint16_t Connection_Handle,
                                         uint16_t Attribute_Handle,
                                         uint16_t Data_Offset,
                                         uint16_t Data_Length,
                                         uint8_t Data[])
{
    uint8_t att_err;
    tBleStatus ret;
    ret = ATT_pwrq_push(Connection_Handle, Attribute_Handle,
    Data_Offset, Data_Length, Data);
    if (ret != BLE_STATUS_SUCCESS)
    {
        att_err = BLE_ATT_ERR_PREP_QUEUE_FULL;
    }
    else
    {
        att_err = BLE_ATT_ERR_NONE;
    }
    /* Send response */
    aci_gatt_srv_resp(Connection_Handle, Attribute_Handle,
                      att_err, Data_Length, Data);
}
```

注意： *调用 aci_gatt_srv_resp()函数，传递接收到的数据。需要将该数据重定向回协议栈，以生成“prepare write”响应数据包。它包含相关数据，并向客户端确认接收到的数据的正确性。*

aci_gatt_srv_exec_write_resp()示例

下面的示例代码展示如何实现 aci_att_srv_exec_write_req_event()句柄。所有排队的“prepare write”请求都由 write_queued_data()函数写入，如下所示。

aci_gatt_srv_resp() 函数根据 att_error 值生成一个“exec write”响应或一个错误。

```
void aci_gatt_srv_exec_write_resp(uint16_t Conn_Handle,
void aci_gatt_srv_exec_write_resp(uint16_t Conn_Handle, uint8_t Exec)
{
    uint8_t att_error;
    att_error = BLE_ATT_ERR_NONE;
    if (Exec == 1U)
    {
        att_error = write_queued_data(Conn_Handle);
    }
    ATT_pwrq_flush(Conn_Handle);
    aci_gatt_srv_resp(Conn_Handle, 0U, att_error, 0U, NULL);
}
```

3.2.2.9 *GAP 和 GATT 配置文件组件*

BlueNRG 协议栈的 ATT/GATT 组件不会自动分配 GAP 和 GATT 服务，而是将这些服务的实现委托给应用程序：使应用程序能够根据其需要自定义这些配置文件。提供了两个组件作为注册这些配置文件的参考：详见 gatt_profile.c 和 gap_profile.c。这些组件实现协议栈调用的初始化函数，用于注册 GATT 和 GAP 配置文件：Gatt_profile_init()和 Gap_profile_init()。

gatt_profile.c 提供一个默认的 GATT 配置文件，但可进行自定义：例如，如果客户端支持的功能和数据库哈希特征未在 GATT 服务中注册，那么稳健缓存功能将被自动禁用，因为这些特征对于此类功能是必需的。如果删除服务更改特征，则数据库将处于静态，且不能发送任何服务更改指示。

注意： *数据库哈希特征将以一种特殊的方式处理：应用程序不分配值缓冲区，因为哈希是由协议栈内部生成和分配的。gap_profile.c 通过其特征实现默认的 GAP 配置文件。这些组件还提供一些常用功能来设置特征值，如设备名称。*

3.2.2.10 *ATT_PWRQ 组件*

该组件的作用是提供一种机制，在 FIFO 中存储接收到的“prepare write”请求。这是 att_pwrq.c 中作为参考提供的可选组件。

ATT_pwrq_init()

该函数初始化 ATT_PWRQ 组件。

表 45. **ATT_pwrq_init 参数**

类型	参数	说明
uint16_t	queue_length	队列缓冲区大小
uint8_t *	queue_buffer_p	指向用于存储 FIFO 的缓冲区的指针

ATT_pwrq_flush()

此函数删除与连接句柄相关的所有排队“writes”操作。

表 46. **ATT_pwrq_flush 参数**

类型	参数	说明
uint16_t	conn_handle	队列缓冲区大小

ATT_pwrq_read()

读取 FIFO 元素。此类元素是根据连接句柄进行筛选，并按参数进行索引的。

表 47. ATT_pwrq_read 参数

类型	参数	说明
uint16_t	conn_handle	要筛选的连接句柄
uint16_t	idx	要读取的条目索引
ble_gatt_clt_write_ops_t *	wr_ops_p	返回一个指针，该指针指向保存“prepared write”信息的结构

ATT_pwrq_pop()

从 FIFO 提取一个条目。条目是根据连接和属性句柄进行筛选的。

表 48. ATT_pwrq_pop 参数

类型	参数	说明
uint16_t	conn_handle	要筛选的连接句柄
uint16_t	attr_handle	要筛选的属性句柄
ble_gatt_clt_write_ops_t *	wr_ops_p	返回一个指针，该指针指向保存“prepared write”信息的结构

ATT_pwrq_push()

将“prepare write”的数据推入 FIFO。

表 49. ATT_pwrq_push 参数

类型	参数	说明
uint16_t	conn_handle	连接句柄，从这里接收连接句柄
uint16_t	attr_handle	要写入的属性句柄
uint16_t	data_offset	开始写入数据的偏移量
uint16_t	data_length	要写入的数据长度
uint8_t *	数据	指针指向要写入的数据

3.2.3 SoC vs.网络协处理器

在应用程序处理器模式下，应用程序驻留在 BlueNRG-LP/BlueNRG-LPS 内存上；在网络协处理器模式下，应用程序在外部运行。BLE 设备内部的应用层通过串行接口导出 BLE 协议栈功能。对于该范围，网络协处理器需要分配能够保存数据库定义结构和值的缓冲区。

3.2.3.1 DTM

为了将设备用作网络协处理器，设备内部需要一个适配层。DTM（直接测试模式）示例应用程序是在网络处理器模式下使用设备的一种方式。DTM 应用程序公开 ACI（应用程序控制器接口）命令和事件，以便外部设备上的应用程序可以通过串行接口使用 BLE 协议栈。

DTM 应用程序公开的到 GATT 层的接口不同于原生 GATT API。aci_gatt_nwk.c 文件实现网络协处理器 API 和原生 API 之间的适配层。该模块定义一些缓冲区，用于分配定义服务、特征和描述符所需的内存空间。它还分配一个内存，用于存储驻留在 DTM 内存空间中的属性值。它使用动态内存分配器来分配所请求的内存。当客户端请求读/写操作时，将分配的结构插入到链表中以搜索相关的值缓冲区。

一些客户端过程（例如“write”和“write long characteristic”过程）需要在缓冲区中临时存储数据。该组件还分配这些过程所需的缓冲区。这些缓冲区一直保持被分配状态，直至过程结束。

3.2.4 GATT 客户端

设备（作为客户端）向服务器发起命令和请求，并可以接收服务器发送的响应、指示和通知。该角色负责以下操作：

- 交换配置
- 发现服务器上的服务和特征
- 读取属性值
- 写入属性值
- 接收服务器的通知和指示
- 对接收到的指示发送确认。

GATT 使用属性协议(ATT)在客户端和服务器之间以命令、请求、指示、通知和确认的形式传输数据。一些 GATT 客户端流程只生成一个 ATT 请求，并等待来自服务器的响应。接收到响应（或出现超时）后，流程终止。其他流程由多个请求-响应 ATT 数据包的交换组成。

通过接收到 aci_gatt_proc_complete_event 事件表示流程结束。如果一个流程正在进行中，不能在同一服务器上启动其流程。

在下一节中，将显示面向 GATT 客户端的可用函数列表。

表 50. GATT 客户端 API

API	说明
aci_gatt_clt_exchange_config	该流程可将 ATT MTU 设置为两台设备都能支持的最大值。此流程可以在连接期间启动一次
aci_gatt_clt_disc_all_primary_services	该函数用于发现服务器上的所有主要服务
aci_gatt_clt_disc_primary_service_by_uuid	该函数用于在仅知道服务 UUID 的情况下发现服务器上的特定主要服务
aci_gatt_clt_find_included_services	该函数用于在服务器上的服务定义中查找 include 服务声明。此服务由服务句柄范围标识
aci_gatt_clt_disc_all_char_of_service	当仅知道服务句柄范围时，可使用此函数在服务器上的服务定义中查找所有特征声明
aci_gatt_clt_disc_char_by_uuid	如果仅知道服务句柄范围和特征 UUID 时，可使用此函数在服务器上发现服务特征
aci_gatt_clt_disc_all_char_desc	如果仅特征句柄范围已知，可使用此函数查找特征定义中的所有特征描述符属性句柄和属性类型。指定的特征由特征句柄范围标识
aci_gatt_clt_read	如果客户端已知属性句柄，则该函数用于从服务器读取一个属性值
aci_gatt_clt_read_long	如果客户端已知属性句柄，则该函数用于从服务器读取一个长属性值
aci_gatt_clt_read_multiple_char_value	当客户端已知特征值句柄时，该函数用于从服务器读取多个特征值
aci_gatt_clt_read_using_char_uuid	如果客户端仅知特征 UUID，却不知特征的句柄，则该函数用于从服务器读取特征值

API	说明
aci_gatt_clt_write_without_resp	如果客户端知道属性句柄，但是客户端不需要确认写入操作已成功执行，则使用此函数将属性值写入服务器。该函数只写属性值的头 ATT_MTU-3 个字节的数据。
aci_gatt_clt_signed_write_without_resp	如果客户端知道属性句柄，且 ATT bearer 加密，则使用此函数将属性值写入服务器。当启用了“attribute properties authenticated”位，且客户端和服务器设备共享一个 bond 时，才可以使用此函数
aci_gatt_clt_write	如果客户端已知属性句柄，则该函数用于将属性值写入服务器。该函数只写属性值的头 ATT_MTU-3 个字节的数据。
aci_gatt_clt_write_long	如果客户端知道属性句柄，但是属性值的长度超过使用 aci_gatt_clt_write()函数能够发送的长度，则使用此函数将属性值写入服务器
aci_gatt_clt_write_char_reliable	如果客户端知道特征值句柄，则使用该函数将特征值写入服务器；在执行写入操作之前，需要在两个方向上传递要写入的特征值，以确保要写入的特征值是正确的。如果必须在单次操作中按顺序写入多个值，也可以使用此函数。
aci_gatt_clt_notification_event	如果服务器已经配置为将特征值通知给客户端，却不期望返回任何通知已被成功接收的属性协议层确认，则生成该事件
aci_gatt_clt_indication_event	如果服务器已经配置为将特征值通知给客户端，并且期望返回任何指示已被成功接收的属性协议层确认，则生成该事件。如要进行确认，则使用 aci_gatt_clt_confirm_indication 函数
aci_gatt_clt_confirm_indication	此函数用于向服务器生成句柄值确认，以表示已接收到句柄值指示。
aci_gatt_clt_prepare_write_req	该函数用于请求服务器准备写入属性的值。应用程序可以向服务器发送多个“prepare write”请求，服务器对每个句柄值对进行排队并发送响应
aci_gatt_clt_execute_write_req	该函数用于请求服务器写入或取消写入当前保存在该客户端准备队列中的所有准备值

3.2.5 服务和特征配置

为了添加服务和相关特征，用户应用必须定义要处理的特定配置文件：

1. 标准配置文件由蓝牙 SIG 组织定义。用户必须遵循配置文件规范和服务、特征规范文档，以便使用定义的相关配置文件、服务和特征 16 位 UUID 对其进行实现（请参考蓝牙 SIG 网页：www.bluetooth.org/en-%20us/specification/adopted-specifications）。
2. 专有的非标准配置文件。用户必须定义其自己的服务和特征。在本例中，需要 128 位 UUID，并且由用户自己生成（请参考 UUID 生成器网页：www.famkruithof.net/uuid/uuidgen）。

下面的虚拟代码描述如何通过以下 UUID（128 位）定义一个具有两个特征（TX（通知属性）和 RX（“write without response”属性））的服务：

服务 UUID：D973F2E0-B19E-11E2-9E96-0800200C9A66
TX_Char UUID:D973F2E1-B19E-11E2-9E96-0800200C9A66
RX_Char UUID:D973F2E2-B19E-11E2-9E96-0800200C9A66

```
/* Service and Characteristic UUIDs */
#define SRVC_UUID
0x66,0x9a,0x0c,0x20,0x00,0x08,0x96,0x9e,0xe2,0x11,0x9e,0xb1,0xe0,0xf2,0x73,0xd9
#define TX_CHR_UUID
0x66,0x9a,0x0c,0x20,0x00,0x08,0x96,0x9e,0xe2,0x11,0x9e,0xb1,0xe1,0xf2,0x73,0xd9
#define RX_CHR_UUID
0x66,0x9a,0x0c,0x20,0x00,0x08,0x96,0x9e,0xe2,0x11,0x9e,0xb1,0xe2,0xf2,0x73,0xd9
#define RX_BUFFER_SIZE (20)
/* Define the client configuration characteristic descriptor */
BLE_GATT_SRV_CCCD_DECLARE(tx, NUM_LINKS, BLE_GATT_SRV_CCCD_PERM_DEFAULT,
                                    BLE_GATT_SRV_OP_MODIFIED_EVT_ENABLE_FLAG);

/* TX (notification), RX(write without response) characteristics definition */
static const ble_gatt_chr_def_t user_chars[] = {
      {     /* TX characteristic with CCCD */
            .properties = BLE_GATT_SRV_CHAR_PROP_NOTIFY,
            .permissions = BLE_GATT_SRV_PERM_NONE,
            .min_key_size = BLE_GATT_SRV_MAX_ENCRY_KEY_SIZE,
            .uuid = BLE_UUID_INIT_128(TX_CHR_UUID),
            .descrs = {
                  .descrs_p = &BLE_GATT_SRV_CCCD_DEF_NAME(tx),
                  .descr_count = 1U,
            },
      },
      {     /* RX characteristic */
            .properties = BLE_GATT_SRV_CHAR_PROP_WRITE | BLE_GATT_SRV_CHAR_PROP_WRITE_NO_RESP,
            .permissions = BLE_GATT_SRV_PERM_NONE,
            .min_key_size = BLE_GATT_SRV_MAX_ENCRY_KEY_SIZE,
            .uuid = BLE_UUID_INIT_128(RX_CHR_UUID),
      },
};

/* Chat Service definition */
static const ble_gatt_srv_def_t user_service = {
      .type = BLE_GATT_SRV_PRIMARY_SRV_TYPE,
      .uuid = BLE_UUID_INIT_128(SRVC_UUID),
      .chrs = {
            .chrs_p = (ble_gatt_chr_def_t *) user _chars,
            .chr_count = 2U,
      },
};

uint16_t TXCharHandle, RXCharHandle;
```

可使用以下指令添加服务及其特征：

```
ret= aci_gatt_srv_add_service((ble_gatt_srv_def_t *)&user_service);
```

添加了具有相关特征（TX、RX）的服务之后，用户可以使用以下命令获取相关的 TX 和 RX 特征句柄：

```
TXCharHandle=aci_gatt_srv_get_char_decl_handle((ble_gatt_chr_def_t *)&chat_chars[0]);
```

```
RXCharHandle=aci_gatt_srv_get_char_decl_handle((ble_gatt_chr_def_t *)&chat_chars[1]);
```

如需 aci_gatt_srv_add_service () API 参数的详细描述，请参阅头文件 bluenrg_lp_events.h.

3.3 GAP API 接口

BLE 协议栈 v3.x 重新定义了 GAP API 接口，允许启用广播模式和扫描过程，或者在 BLE GAP 中心(主)设备和 BLE GAP 外围(从)设备之间建立连接。

GAP 外设模式 API

aci_gap_set_advertising_configuration() API 允许为传统广播或给定的扩展广播集配置广播参数。特别是，它定义了可发现模式和要使用的广播类型。

表 51. **aci_gap_set_advertising_configuration() API：可发现模式和广播类型选择**

API	Discoverable_Mode 参数	Advertising_Event_Properties 参数
aci_gap_set_advertising_configuration()	0x00：不可发现 0x01：受限的可发现 0x02：一般可发现	0x0001：可连接 0x0002：可扫描 0x0004：定向 0x0008：高占空比定向可连接 0x0010：传统 0x0020：匿名 0X0040：包含 TX 功率

aci_gap_set_advertising_data() API 允许在广播 PDU 中设置数据。特别是，要求用户指定广播数据的长度（Advertising_Data_Length 参数），并提供与 BLE 规范中定义的广播格式一致的广播数据（Advertising_Data 参数）。

包含广播/扫描响应数据的缓冲区，其内容由协议栈直接访问，因此当设备广播时，内容应保持不变。

接收一个 aci_hal_adv_scan_resp_data_update_event，以通知应用程序缓冲区不再被协议栈使用。这可以在通过 aci_gap_set_advertising_data 向协议栈提供新的广播缓冲区之后发生，也可以在广播终止之后发生。仅当变更是原子性的（例如单个字节或词被修改）时，才可能改变广播缓冲区的内容，而不会产生任何不必要的后果（例如，无线数据损坏）。

一旦定义了广播配置和数据，用户可以使用 aci_gap_set_advertising_enable () API（启用参数）启用/禁用广播。

GAP 发现规程

aci_gap_set_scan_configuration () API 允许为给定的 PHY 配置扫描参数。一旦定义了扫描参数，用户可以通过 aci_gap_start_procedure () API、Procedure_Code 参数启动一个特定的发现规程。

表 52. **aci_gap_start_procedure() API**

API	Procedure_Code 参数
aci_gap_start_procedure()	0x00: LIMITED_DISCOVERY 0x01: GENERAL_DISCOVERY 0x02: AUTO_CONNECTION 0x03: GENERAL_CONNECTION 0x04: SELECTIVE_CONNECTION 0x05: OBSERVATION

可以使用 aci_gap_terminate_proc () API 终止 GAP 发现规程。

表 53. aci_gap_terminate_proc() API

API	Procedure_Code 参数
aci_gap_terminate_proc()	0x00: LIMITED_DISCOVERY 0x01: GENERAL_DISCOVERY 0x02: AUTO_CONNECTION 0x03: GENERAL_CONNECTION 0x04: SELECTIVE_CONNECTION 0x05: OBSERVATION

aci_gap_set_connection_configuration () API 允许为给定的 PHY 配置连接配置参数，以便与对端设备建立连接。

可以通过 aci_gap_create_connection()建立与对端设备的直接连接。该 API 指定仅用于连接的 PHY、对端设备类型（Peer_Address_Type 参数），以及对端地址（Peer_Address 参数）。

aci_gap_terminate() API 允许通过选择相关句柄（Connection_Handle 参数）和结束连接的原因（reason 参数）来终止已建立的连接。

3.3.1 设置可发现模式并使用直接连接建立规程

下列伪代码示例仅描述了使 GAP 外设进入一般可发现模式，以及通过直接连接建立流程将 GAP 中央设备直接连接到该 GAP 外设要执行的特定步骤。

注意： *假设在初始化阶段设置了设备公共地址，如下所示：*

```
uint8_t bdaddr[] = {0x12, 0x34, 0x00, 0xE1, 0x80, 0x02};
ret=aci_hal_write_config_data(CONFIG_DATA_PUBADDR_OFFSET,CONFIG_DATA_PUBADDR_LEN, bdaddr);
if(ret != BLE_STATUS_SUCCESS)PRINTF("Failure.\n");

/*GAP Peripheral: configure general discoverable mode and set advertising data
*/

void GAP_Peripheral_Configure_Advertising(void )
{

      tBleStatus ret;

      /* Define advertising data content: set AD type flags and complete local name */
      static uint8_t adv_data[] = {0x02,AD_TYPE_FLAGS,
FLAG_BIT_LE_GENERAL_DISCOVERABLE_MODE|FLAG_BIT_BR_EDR_NOT_SUPPORTED,13,
AD_TYPE_COMPLETE_LOCAL_NAME,   'B','l','u','e','N','R','G','X','T','e','s',' t'};

      /* Configure the General Discoverable mode, connectable and scannable (legacy advertising):
      Advertising_Handle: 0
      Discoverable_Mode: GAP_MODE_GENERAL_DISCOVERABLE (general discovery mode)
      Advertising_Event_Properties     ADV_PROP_CONNECTABLE|ADV_PROP_SCANNABLE|ADV_PROP_LEGACY
(connectable, scannable and legacy)
      Primary_Advertising_Interval_Min: 100;
      Primary_Advertising_Interval_Max: 100
      Primary_Advertising_Channel_Map: ADV_CH_ALL (channels 37,38,39)
      Peer_Address_Type: 0;
      Peer_Address[6]: NULL;
      Advertising_Filter_Policy: ADV_NO_WHITE_LIST_USE (no whitelist);
      Advertising_Tx_Power: 0;
      Primary_Advertising_PHY: 0;
      Secondary_Advertising_Max_Skip: 0;
      Secondary_Advertising_PHY: 0;
      Advertising_SID: 0;
      Scan_Request_Notification_Enable: 0.
      */

      ret = aci_gap_set_advertising_configuration(0,
                                                  GAP_MODE_GENERAL_DISCOVERABLE,
                                                  ADV_PROP_CONNECTABLE
                                                  ADV_PROP_SCANNABLE|ADV_PROP_LEGACY,
                                                  100, 100,
                                                  ADV_CH_ALL,
                                                  0,
                                                  NULL,
                                                  ADV_NO_WHITE_LIST_USE,
                                                  0, 1, 0, 1, 0,0);
      if (ret != BLE_STATUS_SUCCESS) PRINTF("Failure.\n");


      /* Set the advertising data */
      ret = aci_gap_set_advertising_data(0, ADV_COMPLETE_DATA, sizeof(adv_data), adv_data);
      if (ret != BLE_STATUS_SUCCESS) PRINTF("Failure.\n");

} /* end GAP_Peripheral_Configure_Advertising() */
```

一旦广播模式和数据已经配置，GAP 外设可以使用 aci_gap_set_advertising_enable () API 启用广播：

```
static Advertising_Set_Parameters_t Advertising_Set_Parameters[1];

/*GAP Peripheral: enable advertising(and no scan response is sent)
*/
void GAP_Peripheral_Enable_Advertising(void)
{
      /* Enable advertising:
           Enable: ENABLE (enable advertsing);
           Advertising_Set_Parameters: Advertising_Set_Parameters */
      Advertising_Set_Parameters[0].Advertising_Handle = 0;
      Advertising_Set_Parameters[0].Duration = 0;
      Advertising_Set_Parameters[0].Max_Extended_Advertising_Events = 0;
      //enable advertising
      ret = aci_gap_set_advertising_enable(ENABLE, 1, Advertising_Set_Parameters);
      if (ret != BLE_STATUS_SUCCESS) PRINTF("Failure.\n");
}
```

在以可发现模式连接到 GAP 外围设备之前，GAP 中心设备必须配置扫描和连接参数。

```
/*GAP Central: configure scanning and connection parameters
*/

void GAP_Central_Configure_Connection(void)

{
/* Configure the scanning parameters:
   Filter_Duplicates: DUPLICATE_FILTER_ENABLED (Duplicate filtering enabled)
   Scanning_Filter_Policy: SCAN_ACCEPT_ALL (accept all scan requests)
   Scanning_PHY: LE_1M_PHY (1Mbps PHY)
   Scan_Type: PASSIVE_SCAN
   Scan_Interval: 0x4000;
   Scan_Window: 0x4000;
*/
ret=aci_gap_set_scan_configuration(DUPLICATE_FILTER_ENABLED,
      SCAN_ACCEPT_ALL,
      LE_1M_PHY,
      PASSIVE_SCAN,
      0x4000,
      0x4000);
if(ret != BLE_STATUS_SUCCESS) PRINTF("Failure.\n");

/* Configure the connection parameters:
Initiating_PHY: LE_1M_PHY (1Mbps PHY)
Conn_Interval_Min: 40 (Minimum value for the connection event interval);
Conn_Interval_Max: 40 (Maximum value for the connection event interval);
Conn_Latency: 0 (Slave latency for the connection in a number of connection events);
Supervision_Timeout: 60 (Supervision timeout for the LE Link); Minimum_CE_Length: 2000 (Minimum length
of connection needed for the LE connection);
Maximum_CE_Length: 2000 (Maximum length of connection needed for the LE connection).
*/

ret = aci_gap_set_connection_configuration(LE_1M_PHY, 40, 40, 0, 60, 2000, 2000);
if(ret != BLE_STATUS_SUCCESS) PRINTF("Failure.\n");

}/* GAP_Central_Configure_Connection( )*/
```

一旦配置了扫描和连接参数，GAP 中心设备就可以使用 aci_gap_create_connection () API 来直接连接 GAP 外围设备。

```
/*GAP Central: direct connection establishment procedure to connect to the
GAP Peripheral in discoverable mode
*/

void GAP_Central_Make_Connection(void)

{
    /*Start the direct connection establishment procedure to the GAP peripheral device:
    Initiating_PHY:LE_1M_PHY(1 Mbps PHY)
    Peer_Address_Type: PUBLIC_ADDR (public address);
    Peer_Address: {0xaa, 0x00, 0x00, 0xE1, 0x80, 0x02};
    */

    tBDAddr GAP_Peripheral_address = {0xaa, 0x00, 0x00, 0xE1, 0x80, 0x02};

    /* direct connection to GAP Peripheral device */
    ret = aci_gap_create_connection(LE_1M_PHY,
                                    PUBLIC_ADDR, GAP_Peripheral_address);
    if(ret != BLE_STATUS_SUCCESS) PRINTF("Failure.\n");

}/* GAP_Central_Make_Connection(void )*/
```

注意：

1. *如果在 GAP 规程终止时返回了 ret = BLE_STATUS_SUCCESS，则调用 hci_le_enhanced_connection_complete_event() 事件回调，以指示已与 GAP_Peripheral_address 建立连接（在 GAP 外设上返回相同事件）*
2. *通过使用适当的 Procedure_Code 参数值启动 API aci_gap_terminate_proc()，可以明确地终止连接流程*
3. *aci_gap_set_connection_configuration() 的最后两个参数 Minimum_CE_Length 和 Maximum_CE_Length 是低功耗蓝牙连接所需连接事件的长度。这些参数允许用户指定主设备必须为单个从设备分配的时间量，因此必须精心选择。具体来说，当主设备连接更多从设备时，每个从设备的连接间隔必须等于其他连接间隔或是它们的倍数，并且用户不得为每个从设备设置过长的连接事件长度。*

3.3.2 设置可发现模式并使用一般发现规程（主动扫描）

下列伪代码示例仅描述了使 GAP 外设进入一般可发现模式，以及 GAP 中央设备启动一般发现流程（以便发现无线传输范围内的设备）要执行的特定步骤。

注意：假设在初始化阶段设置了设备公共地址，如下所示：

```
uint8_t bdaddr[] = {0x12, 0x34, 0x00, 0xE1, 0x80, 0x02};
ret =aci_hal_write_config_data(CONFIG_DATA_PUBADDR_OFFSET,CONFIG_DATA_PUBADDR_LEN, bdaddr)
if (ret != BLE_STATUS_SUCCESS)PRINTF("Failure.\n");
```

此外，GAP 外设已经配置了第 3.3.1 节“设置可发现模式并使用直接连接建立流程”中所述的广播和相关数据。GAP 外围设备可以启用扫描响应数据，然后使用 aci_gap_set_advertising_enable () API 进行广播：

```
static Advertising_Set_Parameters_t Advertising_Set_Parameters[1];

/* GAP Peripheral:general discoverable mode (scan responses are sent):
*/
void GAP_Peripheral_Make_Discoverable(void)
{
      tBleStatus ret;
      const char local_name[] = {AD_TYPE_COMPLETE_LOCAL_NAME,'B','l','u','e',
        'N','R','G' };
      /* As scan response data, a proprietary 128bits Service UUID is used.
      This 128bits data cannot be inserted within the advertising packet
      (ADV_IND) due its length constraints (31 bytes). AD Type description:
      0x11: length
      0x06: 128 bits Service UUID type
      0x8a,0x97,0xf7,0xc0,0x85,0x06,0x11,0xe3,0xba,0xa7,0x08,0x00,0x20,0x0c,
      0x9a,0x66: 128 bits Service UUID
      */
      static uint8_t ServiceUUID_Scan[18]=
      {0x11,0x06,0x8a,0x97,0xf7,0xc0,0x85,
      0x06,0x11,0xe3,0xba,0xa7,0x08,0x00,0x2,0x0c,0x9a,0x66};
      /* Enable scan response to be sent when GAP peripheral receives scan requests
      from GAP Central performing general
      discovery procedure(active scan)
      */
      aci_gap_set_scan_response_data(18,ServiceUUID_Scan);

      /* Enable advertising:
      Enable: ENABLE (enable advertsing);
      Number_of_Sets: 1;
      Advertising_Set_Parameters: Advertising_Set_Parameters */
      Advertising_Set_Parameters[0].Advertising_Handle = 0;
      Advertising_Set_Parameters[0].Duration = 0;
      Advertising_Set_Parameters[0].Max_Extended_Advertising_Events = 0;

    //enable advertising
    ret = aci_gap_set_advertising_enable(ENABLE, 1,Advertising_Set_Parameters);
    if (ret != BLE_STATUS_SUCCESS) PRINTF("Failure.\n");

} /* end GAP_Peripheral_Make_Discoverable() */
```

在启动 GAP 通用发现流程之前，GAP 中心设备必须配置扫描参数。

```
/*GAP Central: configure scanning parameters for general discovery procedure*/
void GAP_Central_Configure_General_Discovery_Procedure(void)
{
tBleStatus ret;

/* Configure the scanning parameters (active scan):
  Filter_Duplicates: DUPLICATE_FILTER_DISABLED (Duplicate filtering disabled)
  Scanning_Filter_Policy: SCAN_ACCEPT_ALL (accept all scan requests)
  Scanning_PHY: LE_1M_PHY (1Mbps PHY)
  Scan_Type: ACTIVE_SCAN
  Scan_Interval: 0x4000;
  Scan_Window: 0x4000;
*/
ret=aci_gap_set_scan_configuration(DUPLICATE_FILTER_DISABLED,
       SCAN_ACCEPT_ALL,
       LE_1M_PHY,
       ACTIVE_SCAN,
       0x4000,
       0x4000);
if(ret != BLE_STATUS_SUCCESS) PRINTF("Failure.\n");

}/* end GAP_Central_Configure_General_Discovery_Procedure() */

/*GAP Central: start general discovery procedure to discover the GAP peripheral device in dis
coverable mode */
void GAP_Central_General_Discovery_Procedure(void)
{
tBleStatus ret;

/* Start the general discovery procedure using the following parameters:
   Procedure_Code: 0x1 (general discovery procedure)
   PHYs: LE_1M_PHY (1Mbps PHY)*/
ret =aci_gap_start_procedure(0x01,LE_1M_PHY,0,0);
if (ret != BLE_STATUS_SUCCESS)PRINTF("Failure.\n");
}
```

通过 hci_le_extended_advertising_report_event() 事件回调提供流程响应。流程的结束由 aci_gap_proc_complete_event() 事件回调指示，其中 Procdure_code 参数等于 GAP_GENERAL_DISCOVERY_PROC (0x1)。

```
/* This callback is called when an advertising report is received */
void hci_le_extended_advertising_report_event(uint8_t Num_Reports,
                                              Extended_Advertising_Report_t Advertising_Report[])
{
      /* Advertising_Report contains all the expected parameters.
      User application should add code for decoding the received
      Advertising_Report event databased on the specific evt_type (ADV_IND, SCAN_RSP, ..)
      */

      /* Example: store the received Advertising_Report fields */
      uint8_t bdaddr[6];

      /* type of the peer address (PUBLIC_ADDR,RANDOM_ADDR) */
      uint8_t bdaddr_type = Advertising_Report[0].Address_Type;

      /* event type (advertising packets types) */
      uint8_t evt_type = Advertising_Report[0].Event_Type ;

      /* RSSI value */
      int8_t RSSI = Advertising_Report[0].RSSI;

      /* address of the peer device found during discovery procedure
      */ Osal_MemCpy(bdaddr, Advertising_Report[0].Address,6);

    /* length of advertising or scan response data */
    uint8_t data_length = Advertising_Report[0].Length_Data;

    /* data_length octets of advertising or scan response data
    formatted are on Advertising_Report[0].
    Data field: to be stored/filtered based on specific user application scenario*/

} /* hci_le_extended_advertising_report_event() */
```

具体来说，在这一特定背景下，由于 GAP 外设处于可发现模式并启用了扫描响应，GAP 中央设备上的 hci_le_extended_advertising_report_event()回调将发起下列事件：

1. 具有广播数据包类型（evt_type =ADV_IND - 0x0013）的广播报告事件
2. 具有扫描响应数据包类型（evt_type =SCAN_RSP - 0x001B）的广播报告事件

表 54. ADV_IND 事件类型：主字段

事件类型	地址类型	地址	广播数据	RSSI
0x0013 (ADV_IND)	0x00（公共地址）	0x0280E1003412	0x02,0x01,0x06,0x08,0x09,0x42, 0x6C,0x75,0x65,0x4E,0x52,0x4 7,0x02,0x 0A,0xFE	0xCE

广播数据如下所示（请参考“第 5 节 参考文件”中的蓝牙规范版本）：

表 55. ADV_IND 广播数据：主字段

AD 类型标记字段	本地名称字段
0x02：字段的长度 0x01：AD 类型标记 0x06：0x110（2 位：BR/EDR 不支持；第 1 位：一般可发现模式）	0x08：字段的长度 0x09：完整的本地名称类型 0x42,0x6C,0x75,0x65,0x4E0x 52,0x47：BlueNRG

表 56. SCAN_RSP 事件类型

事件类型	地址类型	地址	扫描响应数据	RSSI
0x001B （SCAN_RS P）	0x01（随机地址）	0x0280E1003412	0x12,0x66,0x9A,0x0C, 0x20,0x00,0x08,0xA7,0 xBA,0xE3,0x11,0x06,0x 85,0xC0,0xF7,0x97,0x8A,0x06,0x11	0xDA

扫描响应数据可以解释如下：（参照蓝牙规范）：

表 57. 扫描响应数据

扫描响应数据
0x12：数据长度 0x11：服务 UUID 广播数据的长度；0x06：128 位服务 UUID 类型； 0x66,0x9A,0x0C,0x20,0x00,0x08,0xA7,0xBA,0xE3,0x11,0x06,0x85,0xC0,0xF7,0x97,0x8A: 128 位服务 UUID

3.4 BLE 协议栈事件和事件回调

每次有需要处理的 BLE 协议栈事件时，BLE 协议栈库通过特定事件回调将该事件通知用户应用。事件回调是由用户应用程序定义并被 BLE 协议栈调用的函数，而 API 则是由栈定义而被用户应用程序调用的函数。BLE 协议栈事件回调原型在文件 bluenrg_lp_events.h 中定义。所有事件回调均可以采用弱定义，以便让每个事件回调都有一个定义。因此，用户根据其自身应用场景确定要调用的所需设备事件回调，以及要执行的相关应用特定操作。

在实现低功耗蓝牙应用时，常见且使用广泛的 BLE 协议栈事件是与发现、连接、终止流程、服务、特征、特征描述符发现流程、GATT 客户端上的属性通知/指示事件和 GATT 服务器上的属性修改事件相关的事件。

表 58. BLE 协议栈：主要事件回调

事件回调	说明	其中：
hci_disconnection_complete_event()	连接终止	GAP 中央设备/外设
hci_le_enhanced_connection_complete_event()	向构成连接的两个设备指示新的连接已建立。当 BLE 协议栈支持扩展的广播/扫描特性时引发此事件	GAP 中心/外围（BLE 协议栈 v3.x）
aci_gatt_clt_notification_event()	由 GATT 客户端在服务器通知客户端上的任何属性时生成	GATT 客户端
aci_gatt_clt_indication_event()	由 GATT 客户端在服务器指示客户端上的任何属性时生成	GATT 客户端
aci_gap_pass_key_req_event()	当需要密钥用于配对时，由安全管理器向应用生成。 在接收到该事件时，应用必须以 aci_gap_pass_key_resp() API 响应	GAP 中央设备/外设
aci_gap_pairing_complete_event()	在配对过程成功完成或发生配对过程超时或配对失败时生成	GAP 中央设备/外设
aci_gap_bond_lost_event()	在配对请求发出时生成的事件，响应来自主设备（之前已与从设备绑定）的从设备安全请求。在接收到该事件时，上层必须发出指令 aci_gap_allow_rebond()，以允许从设备继续完成与主设备的配对过程	GAP 外设
aci_gatt_clt_read_by_group_type_resp_event()	发送按组读取型响应，以应答接收到的按组读取型请求，包含已读取的属性的句柄和值	GATT 客户端
aci_gatt_clt_read_by_type_resp_event()	发送读取型响应，以应答接收到的读取型请求，包含已读取的属性的句柄和值	GATT 客户端
aci_gatt_clt_proc_complete_event()	GATT 流程已完成	GATT 客户端
hci_le_extended_advertising_report_event()	在上层启动 GAP 规程之一后，在扫描期间发现设备时 GAP 层向上层生成的事件。当 BLE 协议栈支持扩展的广播/扫描特性时引发此事件	GAP 中心（BLE 协议栈 v3.x）

有关低功耗蓝牙事件和相关格式的详细说明，请参考第 5 节“参考文件”中的 BLE 协议栈 API 和事件文档。

以下伪代码提供了处理一些描述的 BLE 协议栈事件（断开完成事件、连接完成事件、GATT 属性修改事件、GATT 通知事件）的事件回调示例：

```
/* This event callback indicates the disconnection from a peer device.
It is called in the Bluetooth LE radio interrupt context.
*/
void hci_disconnection_complete_event(uint8_t Status,
uint16_t Connection_Handle, uint8_t Reason)
{
      /* Add user code for handling Bluetooth LE disconnection complete event based on application scenario.
      */
}/* end hci_disconnection_complete_event() */

/* This event callback indicates the end of a connection procedure.
* NOTE: If the Bluetooth LE v3.x stack includes the extended advertising/scanning features, the
hci_le_enhanced_connection_complete_event() is raised.
*/
void hci_le_enhanced_connection_complete_event (uint8_t Status,
                                                uint16_t Connection_Handle,
                                                uint8_t Role,
                                                uint8_t Peer_Address_Type,
                                                uint8_t Peer_Address[6],
                                                uint8_t Local_Resolvable_Private_Address[6],
                                                uint8_t Peer_Resolvable_Private_Address[6],
                                                uint16_t Conn_Interval,
                                                uint16_t Conn_Latency,
                                                uint16_t Supervision_Timeout,
                                                uint8_t Master_Clock_Accuracy);
{
      /* Add user code for handling BLE connection complete
      event based on application scenario.
      NOTE: Refer to header file Library\Bluetooth_LE\inc\bluenrg1_events.h
      for a complete description of the event callback
      parameters.
      */

      /* Store connection handle */
      connection_handle = Connection_Handle;
      ...
}/* end hci_le_enhanced_connection_complete_event() */
```

```
#if GATT_SERVER

/* This event callback indicates that an attribute has been written from a peer device.
*/
void aci_gatt_srv_write_event(uint16_t Connection_Handle,
                              uint8_t Resp_Needed,
                              uint16_t Attribute_Handle,
                              uint16_t Data_Length,
                              uint8_t Data[]);{
/* Add user code for handling attribute modification event based on application scenario.
NOTE: Refer to header file bluenrglp_events.h for a complete description of the event callback parameters.
*/
...
} /* end aci_gatt_srv_write_event () */

#endif /* GATT_SERVER */

#if GATT_CLIENT

/* This event callback indicates that an attribute notification has been received from a peer device.
*/
void aci_gatt_notification_event(uint16_t Connection_Handle,uint16_t Attribute_Handle,
                                 uint8_t Attribute_Value_Length,uint8_t Attribute_Value[])
{
      /* Add user code for handling attribute notification event based on application scenario.
      NOTE: Refer to header bluenrglp_events.h for a complete description of the event callback parameters*/
      ...
} /* end aci_gatt_notification_event() */
#endif /* GATT_CLIENT */
```

3.5 安全（配对和绑定）

本节介绍在两个设备之间建立配对要使用的主要功能（验证设备身份，加密链路，以及分发下一次重新连接时要使用的密钥）。

为了与设备成功配对，必须根据所选设备的可用 IO 能力正确配置 IO 能力。

aci_gap_set_io_capability(io_capability)应与下列 io_capability 值之一一起使用：

```
0x00: ‘IO_CAP_DISPLAY_ONLY’
0x01: ‘IO_CAP_DISPLAY_YES_NO’,
0x02: ‘KEYBOARD_ONLY’
0x03: ‘IO_CAP_NO_INPUT_NO_OUTPUT’
0x04: ‘IO_CAP_KEYBOARD_DISPLAY’
```

采用 2 个 BLE 设备时的输入密钥法示例：Device_1, Device_2

下列伪代码示例仅描述了使用输入密钥法将两个设备配对时要执行的特定步骤。

如表 2. LE PHY 关键参数所述，为了选择输入密钥作为安全方法，必须为 Device_1、Device_2 设置 IO 能力。

在本例中，对 Device_1 选择“仅显示”，对 Device_2 选择“仅键盘”，如下所示：

```
/*Device_1: */
tBleStatus ret;
ret= aci_gap_set_io_capability(IO_CAP_DISPLAY_ONLY);
if (ret != BLE_STATUS_SUCCESS) PRINTF("Failure.\n");

/*Device_2: */
tBleStatus ret;
ret= aci_gap_set_io_capability(IO_CAP_KEYBOARD_ONLY);
if (ret != BLE_STATUS_SUCCESS) PRINTF("Failure.\n");
```

在定义了 IO 能力后，应使用 aci_gap_set_authentication_requirement()设置设备需要的所有安全验证要求（MITM 模式（链路是否经过验证），是否有 OOB 数据，是否使用固定 PIN 码，以及是否启用绑定）。

下列伪代码示例仅描述了使用输入密钥法为具有以下配置的设备设置授权要求应遵循的特定步骤：“MITM 保护，无 OOB 数据，请勿使用固定 PIN 码”：此配置用于在配对过程中授权链路并使用非固定 PIN 码。

```
ret=aci_gap_set_authentication_requirement(BONDING,/*bonding is
                                                    enabled */
                                           MITM_PROTECTION_REQUIRED,
                                           SC_IS_SUPPORTED,/*Secure connection
                                                            supported
                                                            but optional */
                                           KEYPRESS_IS_NOT_SUPPORTED,
                                           7, /* Min encryption key size */
                                           16, /* Max encryption
                                                  key size */
                                           0x01, /* fixed pin is not used*/
                                           0x123456, /* fixed pin */
                                           0x00 /* Public Identity address type */);
if (ret != BLE_STATUS_SUCCESS) PRINTF("Failure.\n");
```

在定义了安全 IO 能力和验证要求后，应用可通过以下方式启动配对流程：

1. 在 GAP 外围（从）设备上使用 aci_gap_slave_security_req()（它向主设备发送从设备安全请求）：

```
tBleStatus ret;
ret= aci_gap_slave_security_req(conn_handle,
if (ret != BLE_STATUS_SUCCESS) PRINTF("Failure.\n");
```

- 或在 GAP 中央（主）设备上使用 aci_gap_send_pairing_req()。

由于未设置固定 PIN 码，一旦两个设备之一启动了配对流程，低功耗蓝牙设备调用 aci_gap_pass_key_req_event()事件回调（具有相关连接句柄），以要求用户应用提供用于建立加密密钥的密码。低功耗蓝牙应用必须通过使用 aci_gap_pass_key_resp(conn_handle,passkey)API 提供正确密码。

当在 Device_1 上调用 aci_gap_pass_key_req_event()回调时，它应该生成一个随机 PIN 码，并通过 aci_gap_pass_key_resp() API 对其进行设置，如下所示：

```
void aci_gap_pass_key_req_event(uint16_t Connection_Handle)
{
    tBleStatus ret;
    uint32_t pin;
    /*Generate a random pin with an user specific function */
    pin = generate_random_pin();
    ret= aci_gap_pass_key_resp(Connection_Handle,pin);
    if (ret != BLE_STATUS_SUCCESS) PRINTF("Failure.\n");
}
```

由于 Device_1 的 I/O 能力被设置为“仅显示”，它应在设备显示装置上显示生成的 PIN 码。由于 Device_2 的 I/O 能力被设置为“仅键盘”，用户可以使用键盘，并通过相同的 aci_gap_pass_key_resp() API 为 Device_2 提供 Device_1 上显示的 PIN 码。

或者，如果用户想要设置具有固定 PIN 码 0x123456 的验证要求（无需输入密钥事件），可以使用下列伪代码：

```
tBleStatus ret;

ret= aci_gap_set_auth_requirement(BONDING, /* bonding is
                                  enabled */
                                  MITM_PROTECTION_REQUIRED,
                                  SC_IS_SUPPORTED, /* Secure
                                  connection supported
                                  but optional */
                                  KEYPRESS_IS_NOT_SUPPORTED,
                                  7, /* Min encryption
                                  key size */
                                  16, /* Max encryption
                                  key size */
                                  0x00, /* fixed pin is used*/
                                  0x123456, /* fixed pin */
                                  0x00 /* Public Identity address type */);
if (ret != BLE_STATUS_SUCCESS) PRINTF("Failure.\n");
```

注意：

1. *如果通过调用所述 API（aci_gap_slave_security_req()或 aci_gap_send_pairing_req()）启动配对流程并返回值 ret= BLE_STATUS_SUCCESS，则在流程结束时，将调用 aci_gap_pairing_complete_event()事件回调，以便在回调状态参数上指示配对状态：*
   - *0x00：配对成功*
   - *0x01：配对超时*
   - *0x02：配对失败*

   如果失败，则原因参数提供配对失败原因代码（如果状态参数返回成功或超时，则为 0）。
2. *如果 2 个设备成功配对，将在首次连接时自动加密链路。如果还启用了绑定（存储了密钥以备后用），在重新连接 2 个设备时，可以直接加密链路（无需再次执行配对流程）。用户应用可以直接使用相同的 API，不执行配对流程而直接加密链路：*
   - *aci_gap_slave_security_req)（对于 GAP 外围（从）设备）或*
   - *aci_gap_send_pairing_req()（对于 GAP 中央（主）设备）。*
3. *如果从设备已与主设备绑定，可以向主设备发送从设备安全请求以加密链路。当收到从设备安全请求时，主设备可以加密链路、启动配对流程或拒绝请求。通常情况下，主设备只加密链路，不执行配对流程。相反地，如果主设备启动配对流程，这意味着主设备因某些原因丢失了绑定信息，因此必须重新启动配对流程。因此，从设备调用 aci_gap_bond_lost_event()事件回调，以通知用户应用其已不再与之前绑定的主设备绑定。然后，从设备应用可以决定允许安全管理器完成配对流程，并通过调用指令 aci_gap_allow_rebond()与主设备重新绑定，或者只是关闭连接并将安全问题通知用户。*

3.6 服务和特征发现

本节介绍在两个设备建立连接后，允许 GAP 中央设备发现 GAP 外设服务和特征的主要函数。在下面的伪代码示例中，使用了传感器感测演示服务和特征及相关句柄作为参考服务和特征。此外，假设 GAP 中央设备连接了运行传感器演示配置文件应用的 GAP 外设。GAP 中央设备使用服务和发现流程寻找 GAP 外设传感器感测演示服务和特征。

表 59. 低功耗蓝牙传感器感测演示服务和特征句柄

服务	特征	服务/特征句柄	特征值句柄	特征客户端描述符配置句柄	特征格式句柄
加速服务	NA	0x0010	NA	NA	NA
	自由落体特征	0x0011	0x0012	0x0013	NA
	加速特征	0x0014	0x0015	0x0016	NA
环境服务	NA	0x0017	NA	NA	NA
	温度特征	0x0018	0xx0019	NA	0x001A
	压力特征	0x001B	0xx001C	NA	0x001D

有关传感器配置文件演示的详细信息，请参阅 SDK 用户手册和 SDK 软件包中提供的传感器演示源代码（请参阅第 5 节“参考文件”）。

下面是包含相关描述的服务发现 API 列表：

表 60. 服务发现流程 API

发现服务 API	说明
aci_gatt_clt_disc_all_primary_services()	此 API 启动 GATT 客户端流程以发现 GATT 服务器上的所有主要服务。在 GATT 客户端连接了设备，并想要寻找设备上提供的所有主要服务以确定其功能时使用
aci_gatt_clt_disc_primary_service_by_uuid()	此 API 启动 GATT 客户端流程，以通过使用其 UUID 发现 GATT 服务器上的主要服务。 在 GATT 客户端连接了设备并想要寻找特定服务而无需获取任何其他服务时使用
aci_gatt_clt_find_included_services()	此 API 启动寻找所有已包含服务的流程。在 GATT 客户端已发现主要服务并想要发现次要服务时使用

以下伪代码示例描述了 aci_gatt_clt_disc_all_primary_services() API：

```
/*GAP Central starts a discovery all services procedure: conn_handle is the connection handle returned
on hci_le_extended_advertising_report_event() event callback
*/
if (aci_gatt_clt_disc_all_primary_services(conn_handle) !=BLE_STATUS_SUCCESS)
{
PRINTF("Failure.\n");

}
```

通过 aci_gatt_clt_read_by_group_type_resp_event() 事件回调提供流程响应。通过 aci_gatt_clt_proc_complete_event()事件回调()调用指示流程结束。

```
/* This event is generated in response to a Read By Group Type
   Request: refer to aci_gatt_clt_disc_all_primary_services() */
void aci_gatt_clt_read_by_group_type_resp_event(uint16_t Conn_Handle, uint8_t
                                                Attr_Data_Length,uint8_t Data_Length,
                                                uint8_t Att_Data_List[]);
{
      /*      Conn_Handle: connection handle related to the response;
              Attr_Data_Length: the size of each attribute data;
              Data_Length: length of Attribute_Data_List in octets;
              Att_Data_List: Attribute Data List as defined in Bluetooth LE Specifications.
                             A sequence of attribute handle, end group handle, attribute value tuples:
                             [2 octets for Attribute Handle, 2 octets End Group Handle,
                             (Attribute_Data_Length - 4 octets) for
                             Attribute Value].
*/

/* Add user code for decoding the Att_Data_List field and
getting the services attribute handle, end group handle and service uuid
*/
}/* aci_gatt_clt_read_by_group_type_resp_event() */
```

就传感器感测演示而言，GAP 中央设备应用应获取三个按组读取型响应事件（通过相关的 aci_gatt_clt_read_by_group_type_resp_event()事件回调），并具有以下回调参数值。

第一个按组类型读取的响应事件回调参数：

```
Connection_Handle:0x0801（连接句柄）；
Attr_Data_Length:0x06（每个已发现服务数据（服务
句柄、结束组句柄、服务 UUID）的长度)；
Data_Length:0x0C（Attribute_Data_List 的八位组长度）
Att_Data_List:0x0C 字节如下：
```

表 61. 第一个按组类型读取的响应事件回调参数

句柄属性	结束组句柄	服务 UUID	注释
0x0001	0x0008	0x1801	属性配置文件服务。 标准 16 位服务 UUID
0x0009	0x000F	0x1800	GAP 配置文件服务。 标准 16 位服务 UUID。

第二个按组类型读取的响应事件回调参数：

```
Conn_Handle:0x0801（连接句柄）；
Attr_Data_Length:0x14（每个已发现服务数据：
服务句柄、结束组句柄、服务 UUID）的长度)；
Data_Length:0x14（Attribute_Data_List 的八位组长度）；
Att_Data_List:0x14 字节如下：
```

表 62. 第二个按组类型读取的响应事件回调参数

句柄属性	结束组句柄	服务 UUID	注释
0x0010	0x0016	0x02366E80CF3A11E19AB4 0002A5D5C51B	128 位服务专有 UUID 的加速服务

第三个按组类型读取的响应事件回调参数：

```
Connection_Handle:0x0801（连接句柄）；
Attr_Data_Length:0x14（每个已发现服务数据：
服务句柄、结束组句柄和服务 uuid）；
Data_Length:0x14（Attribute_Data_List 的八位组长度）；
Att_Data_List:0x14 字节如下：
```

表 63. 第三个按组类型读取的响应事件回调参数

句柄属性	结束组句柄	服务 UUID	注释
0x0017	0x001D	0x42821A40E47711E282D00 002A5D5C51B	128 位服务专有 UUID 的环境服务

就传感器感测演示而言，当发现所有主要服务流程完成时，在 GAP 中央设备应用上调用 aci_gatt_clt_proc_complete_event()事件回调，并具有以下参数

```
Conn_Handle:0x0801 (connection handle;
Error_Code:0x00
```

3.7 特征发现流程与相关 GATT 事件

下面是包含相关描述的特征发现 API 列表：

表 64. 特征发现流程 API

发现服务 API	说明
aci_gatt_ctl_disc_all_char_of_service ()	此 API 启动 GATT 流程以发现给定服务的所有特征
aci_gatt_ctl_disc_char_by_uuid ()	此 API 启动 GATT 流程以发现通过 UUID 指定的所有特征
aci_gatt_ctl_disc_all_char_desc ()	此 API 启动 GATT 服务器上发现所有特征描述符的流程

就 BLE 传感器感测演示而言，下面是描述 GAP 中央设备应用如何发现所有加速服务特征的简单虚拟代码（请参考表 1. 低功耗蓝牙 RF 通道类型和频率（第二个按组类型读取的响应事件回调参数）：

```
uint16_t service_handle= 0x0010;
uint16_t end_group_handle = 0x0016;
```

```
/*GAP Central starts a discovery all the characteristics of a service
procedure: conn_handle is the connection handle returned on
hci_le_advertising_report_event()eventcallback */
if(aci_gatt_disc_all_char_of_service(conn_handle,
                                     service_handle,/* Service handle */
                                     end_group_handle/* End group handle
                                     */
                                    );) != BLE_STATUS_SUCCESS)
{
    PRINTF("Failure.\n");
}
```

通过 aci_att_read_by_type_resp_event()事件回调提供流程响应。通过 aci_gatt_proc_complete_event()事件回调调用指示流程结束。

```
void aci_att_read_by_type_resp_event(uint16_t Connection_Handle,
                                     uint8_t Handle_Value_Pair_Length,
                                     uint8_t Data_Length,
                                     uint8_t Handle_Value_Pair_Data[])
{
    /*
    Connection_Handle: connection handle related to the response;
    Handle_Value_Pair_Length: size of each attribute handle-value
    Pair;
    Data_Length: length of Handle_Value_Pair_Data in octets.
    Handle_Value_Pair_Data: Attribute Data List as defined in
    Bluetooth Core specifications. A sequence of handle-value pairs: [2
    octets for Attribute Handle, (Handle_Value_Pair_Length - 2 octets)
                                    for Attribute Value].
    */
    /* Add user code for decoding the Handle_Value_Pair_Data field and
    get the characteristic handle, properties, characteristic value handle,
    characteristic UUID*/
    */
}/* aci_att_read_by_type_resp_event() */
```

就低功耗蓝牙传感器感测演示而言，GAP 中央设备应用应获取两个读取类型响应事件（通过相关 aci_att_read_by_type_resp_event()事件回调），并具有以下回调参数值。

按类型响应事件回调参数的第一次读取：

```
conn_handle: 0x0801（连接句柄）；
Handle_Value_Pair_Length: 0x15（每个已发现
特征数据（特征句柄、属性、
特征值句柄、特征 UUID）的长度)；
Data_Length: 0x16（事件数据的长度）；
Handle_Value_Pair_Data:0x15 字节如下：
```

表 65. 按类型响应事件回调参数的第一次读取

特征句柄	特征属性	特征值句柄	特征 UUID	注释
0x0011	0x10（通知）	0x0012	0xE23E78A0CF4A11E18FFC0002A5D5C51B	128 位特征专有 UUID 的自由落体特征

按类型响应事件回调参数的第二次读取：

```
conn_handle: 0x0801（连接句柄）；
Handle_Value_Pair_Length: 0x15（每个已发现
特征数据（特征句柄、属性、
特征值句柄、特征 UUID）的长度)；
Data_Length:0x16（事件数据的长度）；
Handle_Value_Pair_Data:0x15 字节如下：
```

表 66. 按类型响应事件回调参数的第二次读取

特征句柄	特征属性	特征值句柄	特征 UUID	注释
0x0014	0x12（通知并读取）	0x0015	0x340A1B80CF4B11E1AC360002A5D5C51B	128 位特征专有 UUID 的加速特征

就传感器感测演示而言，当发现所有主要服务流程完成时，在 GAP 中央设备应用上调用 aci_gatt_proc_complete_event()事件回调，并具有以下参数：

```
Connection_Handle:0x0801（连接句柄）；
Error_Code: 0x00.
```

可以执行类似步骤，以便发现环境服务的所有特征（表 1. 低功耗蓝牙 RF 通道类型和频率）。

3.8 特征通知/指示、写入、读取

本节介绍用于访问低功耗蓝牙设备特征的主要函数。

表 67. 特征更新、读取、写入 API

发现服务 API	说明	其中：
aci_gatt_srv_notify	如果服务器端对特征启用了通知（或指示），该 API 向客户端发送通知（或指示）。	GATT 服务器
aci_gatt_clt_read	启动读取属性值的流程。	GATT 客户端
aci_gatt_clt_write	启动写入属性值的流程（当流程完成时，生成 GATT 流程完成事件）。	GATT 客户端
aci_gatt_clt_write_without_resp()	启动写入特征值的流程，不等待服务器的任何响应。	GATT 客户端
aci_gatt_clt_confirm_indication()	确认指示。当应用接收到特征指示时，必须发送该指令。	GATT 客户端

就传感器感测演示而言，GAP 中央设备可以使用下面的伪代码，去设置服务器端自由落体和加速度特征中的叫做“客户端特征配置描述符”的特征描述符以启用服务器端相关特征的通知功能：

```
tBleStatus ret;
uint16_t handle_value = 0x0013;
/*启用自由落体特征客户端描述符配置*/
ret = aci_gatt_clt_write(conn_handle,
                         handle_value /* 自由落体客户端描述符配置的句柄 */
                         0x02,/* 属性值长度 */
                         0x0001, /* 属性值：1 表示通知 */
                         if (ret != BLE_STATUS_SUCCESS) PRINTF("Failure.\n");
handle_value = 0x0016;
/* 启用加速特征客户端描述符配置用于通知 */
ret= aci_gatt_clt_write (conn_handle,handle_value/* 加速客户端描述符
                         配置的句柄 *0x02, /*属性值长度 */
                         0x0001, /* 属性值:1 代表通知 */);
if (ret != BLE_STATUS_SUCCESS) PRINTF("Failure.\n");
```

一旦从 GAP 中央设备启用了特征通知，GAP 外设就可以通知自由落体和加速特征的新值，如下所示：

```
tBleStatus ret;
uint8_t val = 0x01;
uint16_t charac_handle = 0x0017;

/*GAP peripheral notifies free fall characteristic to GAP central*/
ret= aci_gatt_srv_notify (connection_handle,/*connection handle*/
                          charac_handle,/* free fall
                          characteristic handle*/
                          0,/*updated type: notification*/
                          0x01,/* characteristic value length */
                          &val /* characteristic value */)

if(ret != BLE_STATUS_SUCCESS) PRINTF("Failure.\n");

tBleStatus ret;
uint8_t buff[6];
uint16_t charac_handle = 0x004;

/*Set the mems acceleration values on three axis x,y,z on buff array */
/*GAP peripheral notifies acceleration characteristic to GAP Central*/
ret= aci_gatt_srv_notify (connection_handle, /* connection handle */
                          charac_handle, /* acceleration characteristic handle*/
                          0, /* updated type: notification */
                          0x06, /* characteristic value length */
                          buff /* characteristic value */)
                         );
if(ret != BLE_STATUS_SUCCESS) PRINTF("Failure.\n");
```

在 GAP 中央设备上，在从 GAP 外设收到特征通知（加速或自由落体）时发起 aci_gatt_clt_notification_event()事件回调。下面是 aci_gatt_clt_notification_event()回调的虚拟代码：

```
void aci_gatt_clt_notification_event(uint16_t Connection_Handle,
                                     uint16_t Attribute_Handle,
                                     uint8_t Attribute_Value_Length,
                                     uint8_t Attribute_Value[])
{
/* aci_gatt_clt_notification_event event callback parameters:
   Connection_Handle: connection handle related to the response;
   Attribute_Handle: the handle of the notified characteristic;
   Attribute_Value_Length: length of Attribute_Value in octets;
   Attribute_Value: the current value of the notified characteristic.
*/
/* Add user code for handling the received notification based on the application scenario.
*/

}/* aci_gatt_clt_notification_event */
```

3.9 基本/典型错误条件描述

在 BLE 协议栈 v3.x API 框架上，定义了 tBleStatus 类型，以返回 BLE 协议栈错误条件。头文件“ble_status.h”中定义了错误代码。

在调用协议栈 API 时，为了追踪潜在错误状态，建议获取 API 返回状态并对其进行监控。

当 API 成功执行时，返回 BLE_STATUS_SUCCESS (0x00)。有关与每个 ACI API 有关的错误条件列表，请参考第 5 节“参考文档”中的 BLE 协议栈 API 和事件文档。

3.10 同时作为主、从设备的场景

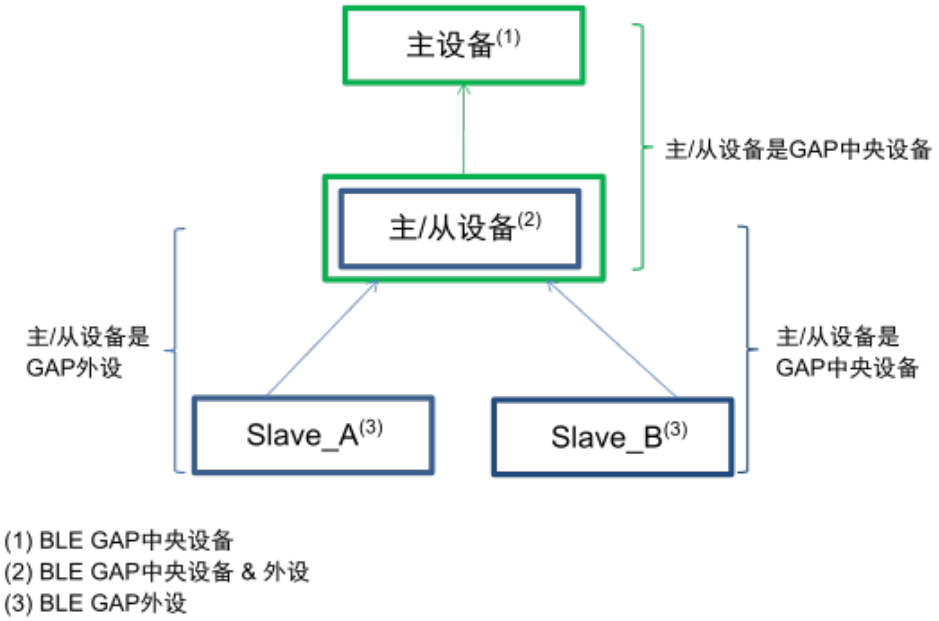
BLE 协议栈 v3.x 支持同时扮演多个角色。因此，同一设备既可作为一个或多个连接上的主设备（最多支持八个连接），也可作为另一个连接上的从设备。

下面的虚拟代码描述了如何初始化 BLE 协议栈设备，以便同时支持中央设备和外设角色：

```
uint8_t role= GAP_PERIPHERAL_ROLE | GAP_CENTRAL_ROLE;
ret= aci_gap_init(role, 0, 0x07,0 &service_handle,
&dev_name_char_handle, &appearance_char_handle);
```

同时作为主、从设备的测试场景如下：

图 18. BLE 同时作为主/从设备的场景



第 1 步. 通过在 GAP_Init() API 上将角色设置为 GAP_PERIPHERAL_ROLE |GAP_CENTRAL_ROLE，将一个低功耗蓝牙设备（称为“主/从设备”）配置为中央设备及外设。假设该设备也定义了相关的服务和特征。

第 2 步. 通过在 GAP_Init() API 上将角色设置为 GAP_PERIPHERAL_ROLE，将两个设备（称为 Slave_A、Slave_B）配置为外设。Slave_A 和 Slave_B 定义与主/从设备相同的服务和特征。

第 3 步. 通过在 GAP_Init() API 上将角色设置为 GAP_CENTRAL_ROLE，将一个设备（称为主设备）配置为中央设备。

第 4 步. Slave_A 和 Slave_B 设备均可使用以下 API 进入发现模式：

– aci_gap_set_advertising_configuration().该 API 通过以下主要参数定义广播配置：
  - Discoverable_mode = GAP_MODE_GENERAL_DISCOVERABLE
  - Advertising_Event_Properties = ADV_PROP_CONNECTABLE|ADV_PROP_SCANNABLE|ADV_PROP_LEGACY,
  - Primary_Advertising_Interval_Min = 0x20
  - Primary_Advertising_Interval_Max = 0x100

– aci_gap_set_advertising_enable().该 API 允许广播按如下方式启用：

```
static Advertising_Set_Parameters_t Advertising_Set_Parameters[1];
Advertising_Set_Parameters[0].Advertising_Handle = 0;
Advertising_Set_Parameters[0].Duration = 0;
Advertising_Set_Parameters[0].Max_Extended_Advertising_Events = 0;
ret = aci_gap_set_advertising_enable(ENABLE, 1,Advertising_Set_Parameters);
```

– aci_gap_set_advertising_data().该 API 通过以下参数定义广播数据：
  - Advertising_Handle = 0;
  - Operation = ADV_COMPLETE_DATA’
  - Advertising_Data_Length = sizeof(adv_data);
  - Advertising_Data_Length = adv_data
  - 其中 adv_data 定义如下：

```
static uint8_t adv_data[] = {0x02,AD_TYPE_FLAGS,
        FLAG_BIT_LE_GENERAL_DISCOVERABLE_MODE|
FLAG_BIT_BR_EDR_NOT_SUPPORTED,6,AD_TYPE_COMPLETE_LOCAL_NAME,
0x08,0x74,0x65,0x73,0x74};
```

第 5 步. 通过使用以下 API，主/从设备在执行发现流程之前配置扫描和连接参数：

– aci_gap_set_scan_configuration().该 API 定义扫描参数：
  - Filter_Duplicates:0x0;
  - Scanning_Filter_Policy:0x0（全部接受）;
  - Scanning_PHY:LE_1M_PHY (1 Mbps PHY);
  - Scan_Type:PASSIVE_SCAN;
  - Scan_Interval:0x10;Scan_Window:0x10

– aci_gap_set_connection_configuration().该 API 定义连接参数：
  - Initiating_PHY = LE_1M_PHY (1 Mbps PHY);
  - Conn_Interval_Min = 0x6c;
  - Conn_Interval_Max= 0x6c;
  - Conn_Latency = 0;
  - Supervision_Timeout = 0xc80;
  - Minimum_CE_Length = 0x000c;
  - Maximum_CE_Length = 0x000c

第 6 步. 主/从设备执行发现流程，以便发现外设 Slave_A 和 Slave_B：

通用发现流程是通过调用 API 启动的

aci_gap_start_procedure()，具有以下参数：

– Procedure_Code = 0x01 /* GENERAL_DISCOVERY */

– PHYs=LE_1M_PHY /* 1 Mbps PHY */

通过使用 hci_le_extendede_advertising_report_event()事件回调通知的广播报告事件发现这两个设备。

在发现两个设备后，主/从设备启动两个连接流程（作为中央设备），以便分别连接到 Slave_A 和 Slave_B 设备：

```
/*
连接到 Slave_A：在发现过程中通过广播报告事件
找到了 Slave_A 的地址类型和地址。
*/
ret = aci_gap_create_connection(LE_1M_PHY, ”Slave_A address type”, ”Slave_A address”);

/* 连接到 Slave_B：在发现过程中通过广播报告事件
找到了 Slave_B 的地址类型和地址。
*/
ret = aci_gap_create_connection(LE_1M_PHY, ”Slave_B address type”, ”Slave_B address”);
```

第 7 步. 一旦连接，主/从设备使用 aci_gatt_clt_write API 在主设备和从设备上启用特征通知。Slave_A 和 Slave_B 设备使用 aci_gatt_srv_notify API 启动特征通知。

第 8 步. 在该阶段，主/从设备进入发现模式（作为外设）。在初始化序列中，用本地名称“Test” = [0x08,0x74,0x65,0x73,0x74]定义广播配置数据和广播数据。主/从设备通过以下方式启用广播，从而进入发现模式：

```
aci_gap_set_advertising_enable(ENABLE, 1, Advertising_Set_Parameters);
```

由于主/从设备也是中央设备，它接收与分别从 Slave_A 和 Slave_B 设备通知的特征值相关的通知事件。

第 9 步. 一旦主/从设备进入发现模式，它还等待来自被配置为 GAP 中央设备的其他低功耗蓝牙设备（称为主设备）的连接请求。使用 aci_gap_set_scan_configuration ()API 配置扫描参数后，主设备开始发现流程以发现主/从设备。

通用发现流程的启动方式如下：

```
ret = aci_gap_start_procedure(Procedure_Code = 0x01,PHYs = 0x01, Duration = 0; Period=0);
```

第 10 步. 一旦发现主/从设备，主设备就会启动一个连接流程进行连接（在使用 aci_gap_set_scan_configuration() API 配置扫描参数之后）。

```
/* 主设备连接主/从设备：在发现流程中通过广播报告事件
发现了主&从设备地址类型和
地址 */
ret= aci_gap_create_connection(Initiating_PHY = 0x01,
Peer_Address_Type= ”主/从设备地址类型”,Peer_Address=” 主&从设备地址");
```

通过使用 hci_le_extended_advertising_report_event()事件回调函数可得知主/从设备被发现。

第 11 步. 一旦连接，主设备使用 aci_gatt_clt_write API 在主/从设备上启用特征通知。

第 12 步. 在该阶段，由于是 GAP 中央设备，主/从设备接收来自从设备 A、从设备 B 的特征通知，而作为 GAP 外设，它还能将这些特征值通知主设备。

3.11 低能耗蓝牙隐私 1.2

BLE 协议栈 v3.x 支持低功耗蓝牙私有 1.2。

私有功能通过频繁修改相关低功耗蓝牙地址来降低跟踪特定低功耗蓝牙的能力。频繁修改的地址被称为私有地址，可通过可信设备进行解析。

为使用该功能，通信中涉及的设备需要先经过配对：使用在先前的配对/绑定过程期间交换的设备 IRK 创建私有地址。

隐私功能有两种变体：

1. 基于主机的隐私私有地址通过主机解析和生成
2. 基于控制器的隐私私有地址通过控制器解析和生成，在主机提供控制器设备身份信息后，不再涉及主机。

当支持控制器隐私时，可进行设备过滤，因为在控制器中执行地址解析（可在检查对端设备是否处于白名单中之前解析对端设备的身份地址）。

3.11.1 基于控制器的私有与设备过滤方案

在 BLE 协议栈 v2.x 上，aci_gap_init() API 支持用于

privacy_type 参数：

- 0x00：禁用隐私
- 0x01：主机隐私启用
- 0x02：控制器隐私启用。

当从设备想要对可解析的私有地址进行解析，并希望能够过滤私有地址以重新连接已绑定和可信设备时，必须执行以下步骤：

1. 在 aci_gap_init()上启用隐私控制器：将 0x02 用作 privacy_typeparameter 参数。
2. 使用允许的安全方法之一连接、配对和绑定候选可信设备：使用设备 IRK 创建私有地址。
3. 使用 aci_gap_get_bonded_devices() API 获取绑定的设备身份地址和类型。
4. 使用 aci_gap_configure_white_and_resolving_list(0x01|0x02)API 将绑定设备的标识地址和类型添加到 BLE 设备控制器白名单中，以及添加到用于解析控制器中的可解析私有地址的地址转换列表中。
5. 设备通过调用 aci_gap_set_advertising_configuration() API 来配置非定向可连接模式，Advertising_Filter_Policy = ADV_WHITE_LIST_FOR_ALL（只允许扫描请求来自白名单，只允许连接请求来自白名单）。
6. 当绑定的主设备执行用于重新连接到从设备的连接过程时，从设备能够解析并过滤主设备的地址并与之连接。

注意： *BlueNRG GUI 软件包中提供了一组测试脚本，用于执行所述的隐私控制器和设备过滤方案（参见第 5 节 参考文件）。这些脚本可以使用 BlueNRG GUI 运行，并可作为使用隐私控制器和设备过滤功能实现固件应用的参考。*

3.11.2 解析地址

在与绑定设备重新连接之后，不必通过严格解析对端设备的地址来加密链路。事实上，BLE 协议栈将自动寻找正确的 LTK 以加密链路。

但是，在某些情况下必须解析对端设备的地址。当设备收到可解析私有地址时，可通过主机或控制器来解析该地址（即链路层）。

基于主机的隐私

如果启用控制器隐私，则可以使用 aci_gap_resolve_private_addr()来解析可解析的私有地址。如果可以在绑定设备存储的 IRK 中找到相应的 IRK，则解析该地址。可通过以下方式接收可解析的私有地址，当低功耗蓝牙设备正在扫描时，通过 hci_le_extended_advertising_report_event()/ hci_le_advertising_report_event()，或当连接已建立时，通过 hci_le_extended_connection_complete_event()/hci_le_connection_complete_event()。.

基于控制器的隐私

如果在链路层启用了地址解析，则在收到可解析的私有地址时使用解析列表。要将绑定设备添加到解析列表，必须调用 aci_gap_configure_white_and_resolving_list();。此函数搜索相应的 IRK 并将其添加到解析列表中。

当启用隐私时，如果已将设备添加到解析列表，则其地址通过链路层自动解析并报告给应用，而无需明确调用任何其他功能。在与设备连接后，返回 hci_le_enhanced_connection_complete_event()。如果已被成功解析，该事件将报告设备的标识地址。

在扫描时，如果该设备使用可解析的私有地址并且其地址已正确解析，则 hci_le_extended_advertising_report_event()包含广播中的设备身份地址。在这种情况下，报告的地址类型为 0x02 或 0x03。如果找不到可解析该地址的 IRK，则报告可解析的私有地址。在广播设备使用了定向广播的情况下，如果已取消屏蔽，并将扫描过滤策略设为 0x02 或 0x03，则通过 hci_le_extended_advertising_report_event()或通过 hci_le_direct_advertising_report_event()报告已解析的私有地址。

3.12 ATT_MTU 和交换 MTU API 事件

ATT_MTU 定义为在客户端和服务器之间发送的任何数据包的最大大小：

- 默认 ATT_MTU 值：23 字节

这决定了用于执行特征操作时的当前最大属性值的大小（通知/写入最大值为 ATT_MTU-3）。

客户端和服务器可以交换使用交换 MTU 请求和响应消息所能接收的最大数据包大小。两个设备均使用这些交换值的最小值来进行所有的进一步通信：

```
tBleStatus aci_gatt_clt_exchange_config(uint16_t Connection_Handle);
```

为响应交换 MTU 请求，在两个设备上触发 aci_att_exchange_mtu_resp_event()回调：

```
void aci_att_exchange_mtu_resp_event(uint16_t Connection_Handle, uint16_t
                                     Server_RX_MTU);
```

Server_RX_MTU 指定了服务器和客户端之间所商定的 ATT_MTU 值。

3.13 LE 数据包长度扩展 API 和事件

在低功耗蓝牙规范 v4.2 中，包数据单元（PDU）的大小已从 27 字节增加到 251 字节。这允许通过降低数据包所需的开销（报头，MIC）来提高数据率。因此，由于降低了开销，可实现更快的 OTA 固件升级操作和更高的效率。

BLE 协议栈 v3.x 支持 LE 数据包长度扩展功能和相关 API、事件：

- HCI LE API（bluenrg_lp_api.h 上的 API 原型）
  - hci_le_set_data_length()
  - hci_le_read_suggested_default_data_length()
  - hci_le_write_suggested_default_data_length()
  - hci_le_read_maximum_data_length()
- HCI LE 事件（bluenrg_lp_events.h 上的事件回调原型）
  - hci_le_data_length_change_event()

hci_le_set_data_length() API 使用户应用程序能够建议用于给定连接的最大传输数据包大小(TxOctets)和最大数据包(TxTime)传输时间：

```
tBleStatus hci_le_set_data_length(uint16_t Connection_Handle,
                                  uint16_t TxOctets,
                                  uint16_t TxTime);
```

支持的 TxOctets 值在[27-251]的范围内，TxTime 按以下方式提供：(TxOctets +14)*8.

在设备连接后，一旦执行 hci_le_set_data_length() API，如果连接的对端设备支持 LE 数据包长度扩展，则会在两个设备上引发以下事件：

```
hci_le_data_length_change_event(uint16_t Connection_Handle,
                                uint16_t MaxTxOctets,
                                uint16_t MaxTxTime,
                                uint16_t MaxRxOctets,
                                uint16_t MaxRxTime)
```

该事件通知主机更改最大链路层有效负载长度或链路层数据通道 PDU 在任一方向（TX 和 RX）的最长时间。所报告的值(MaxTxOctets, MaxTxTime, MaxRxOctets, MaxRxTime)是在更改后实际用于连接的最大值。

3.14 无数据包重试功能

BLE 协议栈 v3.x 提供了禁用链路层重传机制的功能，该机制属于 Bluetooth LE 特征通知的规范，禁用该功能后，对端设备链路层不会对特征通知进行响应。仅当通知的数据长度在链路层数据包允许的范围内，协议栈才支持该功能。

当使用标准低功耗蓝牙协议时，不会丢失数据包，因为协议采用了无限重试次数。在具有大量错误和重试的弱链路中，发送一定数量的数据包所花费的时间可能随错误数量增加。如果应用了“无数据包重试功能”，则协议不会重试损坏的数据包，因此，传递所选数量的数据包所花费的时间相同，但丢失数据包的数量从 0 到某个数字所花费的时间与错误率成比例。可通过将 aci_gatt_srv_notify(); API 的 Flags 参数设置为 0x01 来启用数据包无重传功能。

```
tBleStatus aci_gatt_srv_notify(uint16_tConn_Handle,
                               uint16_t Attr_Handle,
                               uint8_t Flags,
                               uint16_t Value_Length,
                               uint8_t Value[]);
```

有关 API 使用及其参数值的详细信息，请参考 aci_gatt_srv_notify() API 描述。

3.15 BLE 无线电活动和 Flash 操作

在 Flash 擦除或写入操作期间，CPU 从 Flash 取指令的操作可能会停滞，因此无线电中断这样的关键活动可能会延迟。由此可能导致连接丢失和/或不正确的无线电行为。值得注意的是，BLE v3.x 实现一个更灵活和稳健的无线电活动调度程序，增强了中断例程延迟情况下的整体稳健性。

为了防止这种可能的延迟和对无线电活动造成影响，Flash 擦除和写入操作可以通过 aci_hal_end_of_radio_activity_event()回调与预定的 BLE 无线电活动同步。

当设备完成无线电活动时会调用 aci_hal_end_of_radio_activity_event()回调，并会在执行新的无线电活动时提供信息。提供的信息包括无线电活动的类型，以及安排新的无线电活动时系统节拍中的绝对时间。应用可使用此信息来安排与所选无线电活动同步的用户活动。

让我们假设一个 BLE 应用程序启动广播，它还在 Flash 上执行写操作。aci_hal_end_of_radio_activity_event()回调用于在对下一个广播事件进行编程时注册 Next_Advertising_SysTime()时间：

```
void aci_hal_end_of_radio_activity_event(uint8_t Last_State,
                                         uint8_t Next_State,
                                         uint32_t Next_State_SysTime)
{
    if (Next_State == 0x01) /* 0x01: Advertising */
    {
        /* Store time of next programmed advertising */Next_Advertising_SysTime = Next_State_SysTime;
    }
}
```

只有在下一次预定的无线电活动之前有足够的时间执行 Flash 写入操作，FlashRoutine()才会执行。

3.16 BLE 2 Mbps 和 Coded PHY

以下 API 允许主机指定为所有通过 LE 传输的后续连接使用的发送器 PHY 和接收器 PHY 的首选值。

```
BleStatus hci_le_set_default_phy(uint8_t ALL_PHYS,uint8_t TX_PHYS,uint8_t RX_PHYS);
```

下面的 API 允许为 Connection_Handle 标识的连接设置 PHY 首选项。

```
tBleStatus hci_le_set_phy(uint16_t Connection_Handle,
                          uint8_t ALL_PHYS,
                          uint8_t TX_PHYS,
                          uint8_t RX_PHYS,
                          uint16_t PHY_options);
```

以下 API 允许读取当前 PHY：

```
tBleStatus hci_le_read_phy(uint16_t Connection_Handle,
                           uint8_t *TX_PHY,
                           uint8_t *RX_PHY);
```

有关 API 和相关参数的详细描述，请参阅 API html 文档。

3.17 BLE 扩展广播/扫描

BLE 协议栈 v3.x 支持一组用于扩展广播/扫描的 HCI LE 标准 API。参考相关的 html 文档以了解 API 和参数说明。

此外，用于配置广播模式和扫描流程的新 GAP API 允许支持扩展广播/扫描功能。参考“第 3.3 节 GAP API 接口”获取更多详情和示例。

关于如何设置广播配置以启用扩展广播的示例如下：

```
/* Advertising_Handle 用于识别广播集 */
#define ADVERTISING_HANDLE 0
/* 正在配置的广播事件类型：
0x0001:可连接
0x0002:可扫描
0x0004:直接
0x0008:高占空比定向可连接
0x0010:传统（如果没有设置该位，则使用扩展广播）
0x0020:匿名
0x0040:包含 TX 功率
*/
#define ADVERTISING_EVENT_PROPERTIES 0x01 /* Connectable adverting event */
/* PHY on which the advertising packets are transmitted */
#define ADV_PHY LE_1M_PHY
/* Advertising data: ADType flags + manufacturing data */
static uint8_t adv_data[] = {
                                    0x02,0x01,0x06, # ADType Flags for discoverability
                                    0x08,# Length of next AD data
                                    0xFF,/* Manufacturing data */
                                    0x53, 0x54,0x4d, 0x69, 0x63, 0x72,0x6f /* STMicro */
                                  }
/* Set advertising configuration for extended advertising. */

ret = aci_gap_set_advertising_configuration(ADVERTISING_HANDLE
GAP_MODE_GENERAL_DISCOVERABLE,ADVERTISING_EVENT_PROPERTIES,
                                                                    160, 160,
                                                                    ADV_CH_ALL,
                                                                    0,NULL, /* No peer address */
                                                                    ADV_NO_WHITE_LIST_USE,
                                                                    0,/* 0 dBm */
                                                                    ADV_PHY,/* Primary advertising PHY */
                                                                    0, /* 0 skips */
                                                                    ADV_PHY,/* Secondary advertising PHY */
                                                                    0, /* SID */
                                                                    0 /* No scan request notifications */)

/* Set the advertising data */
ret = aci_gap_set_advertising_data(ADVERTISING_HANDLE, ADV_COMPLETE_DATA, sizeof(adv_data), adv_data);

/* Define advertising set (at least one advertising must be set) */
Advertising_Set_Parameters_t Advertising_Set_Parameters[1];
Advertising_Set_Parameters[0].Advertising_Handle = 0;
Advertising_Set_Parameters[0].Duration = 0;
Advertising_Set_Parameters[0].Max_Extended_Advertising_Events = 0;

/* Enable advertising */
ret = aci_gap_set_advertising_enable(ENABLE, 1, Advertising_Set_Parameters);
```

3.17.1 扩展广播&扫描事件

以下事件现在可用：

- hci_le_extended_advertising_report_event():
  此事件表明一个或多个蓝牙设备响应了主动扫描或在被动扫描期间接收了广播广播。
  如果支持扩展广播和扫描，则会生成此事件。否则，会生成 hci_le_advertising_report_event()。
- hci_le_enhanced_connection_complete_event():
  此事件表明已经创建了一个新连接。
  它已经可用于支持控制器隐私（如果已启用）。在 BLE 协议栈 v3.x 上，如果支持扩展广播和扫描，也会生成此事件。否则，会生成 hci_le_connection_complete_event()。

3.18 周期性广播模式

本节介绍支持的周期性广播模式和相关命令/事件。

3.18.1 周期性广播模式

周期性广播模式能令设备以周期性和确定性间隔发送广播数据。此模式仅适用于广播方角色。

处于周期性广播模式的设备会定期使用周期性广播同步信息中指定的跳频序列，发送周期性广播事件。

进入周期性广播模式的设备还应进入周期性广播同步模式，进入的时间至少足以完成一个扩展广播事件。

HCI commands

hci_le_set_periodic_advertising_parameters();

hci_le_set_periodic_advertising_data();

hci_le_set_periodic_advertising_enable().

注意： *如需了解详细的命令说明，请参阅 BLE API html 文档。*

GAP 指令

在广播方角色中用于周期性广播的 GAP 指令只是对 HCI 指令的正式包装。使用了一种机制，该机制类似于面向广播数据的已存在机制。

在片上系统（SoC）模式下，缓冲区指针由应用程序提供，并通过 ll_set_periodic_advertising_data_ptr 传送给控制器，然后由 aci_hal_adv_scan_resp_data_update_event 释放。

在网络协处理器模式下，对于标准和 GAP 级指令，需要在 DTM 层面对缓冲区进行管理：

aci_gap_set_periodic_advertising_configuration();

aci_gap_set_periodic_advertising_enable();

aci_gap_set_periodic_advertising_data();

aci_gap_set_periodic_advertising_data_nwk().

注意： *如需了解详细的命令说明，请参阅 BLE API html 文档。*

3.18.2 周期性广播同步模式

周期性广播同步模式能令设备使用广播方式发送关于“periodic advertising train”的同步信息。此模式仅适用于广播方角色。

处于周期性广播同步模式的设备在不可连接且不可扫描的扩展广播事件中为“periodic advertising train”发送同步信息。所使用的广播间隔与周期性广播事件之间的间隔无关。

除非设备处于周期性广播模式，否则它也不会处于周期性广播同步模式。它可以离开或者重新进入周期性广播同步模式，同时保持在周期性广播模式中。

通过明确地启用/禁用（常规）广播和周期性广播，可以在周期性广播模式和周期性广播同步模式之间进行选择。常规广播和周期性广播均已启用，设备处于同步模式。如果禁用了常规广播，则设备将退出周期性广播同步模式。

3.18.3 周期性广播同步建立流程

周期性广播同步建立流程能令设备接收周期性广播同步信息并同步到“periodic advertising train”。此流程适用于观测者、外围和中心角色

执行周期性广播同步建立流程的设备扫描包含关于“periodic advertising train”的同步信息的不可连接和不可扫描广播事件，或者参与[第 6 卷]B 部分第 5.1.13 节中定义的链路层周期性广播同步传输流程，从而通过现有连接接受周期性广播同步信息。

当设备接收“periodic advertising train”的同步信息时，它可以定期使用周期性广播同步信息中指定的跳频序列侦听周期性广播事件。

HCI 指令

hci_le_periodic_advertising_create_sync();

hci_le_periodic_advertising_create_sync_cancel();
hci_le_periodic_advertising_terminate_sync();
hci_le_set_periodic_advertising_receive_enable();
hci_le_add_device_to_periodic_advertiser_list();
hci_le_remove_device_from_periodic_advertiser_list();
hci_le_clear_periodic_advertiser_list();
hci_le_read_periodic_advertiser_list_size();
hci_le_set_periodic_advertising_sync_transfer_parameters();
hci_le_set_default_periodic_advertising_sync_transfer_parameters()

注意： *如需了解详细的命令说明，请参阅 BLE API html 文档。*

GAP 指令

用于同步的 GAP 指令只是对 HCI 指令的正式包装。

aci_gap_periodic_advertising_create_sync();
aci_gap_periodic_advertising_create_sync_cancel();
aci_gap_periodic_advertising_terminate_sync();
aci_gap_set_periodic_advertising_receive_enable();
aci_gap_add_device_to_periodic_advertiser_list();
aci_gap_remove_device_from_periodic_advertiser_list();
aci_gap_clear_periodic_advertiser_list();
aci_gap_read_periodic_advertiser_list_size();
aci_gap_set_periodic_advertising_sync_transfer_parameters();
aci_gap_set_default_periodic_advertising_sync_transfer_parameters().

注意： *如需了解详细的命令说明，请参阅 BLE API html 文档。*

活动

该模式涉及的事件如下：

hci_le_periodic_advertising_sync_established_event()
hci_le_periodic_advertising_report_event()
hci_le_periodic_advertising_sync_lost_event()
hci_le_periodic_advertising_sync_transfer_received_event().

3.18.4 周期性广播同步传输流程

周期性广播同步传输流程能令设备通过现有连接发送“periodic advertising train”的同步信息。此流程仅适用于外围和中心角色。

执行周期性广播同步传输流程的设备应启动[第 6 卷]B 部分第 5.1.13 节中定义的链路层周期性广播同步传输流程。

HCI 指令

hci_le_periodic_advertising_sync_transfer();
hci_le_periodic_advertising_set_info_transfer()。

注意： *如需了解详细的命令说明，请参阅 BLE API html 文档。*

GAP 指令

aci_gap_periodic_advertising_sync_transfer();
aci_gap_periodic_advertising_set_info_transfer().

注意： *如需了解详细的命令说明，请参阅 BLE API html 文档。*

3.18.5 LE 功率控制和路径丢失监测

以下指令和事件可处理低功耗蓝牙（BLE）协议栈 v3.1 或更高版本提供的低功耗功率控制和路径损耗监控特性。

```
/* Read the current and maximum transmit power levels of the local Controller on a connection */
tBleStatus hci_le_enhanced_read_transmit_power_level(uint16_t Connection_Handle,
                                                     uint8_tPHY,
                                                     int8_t*Current_Transmit_Power_Level,
                                                     int8_t *Max_Transmit_Power_Level);
/* Read the transmit power level used by the remote Controller on a connection*/
tBleStatus hci_le_read_remote_transmit_power_level(uint16_t Connection_Handle,
                                                   uint8_t PHY);
/* Set the path loss threshold reporting parameters for a connection */
tBleStatus hci_le_set_path_loss_reporting_parameters(uint16_t Connection_Handle,
                                                     uint8_t High_Threshold,
                                                     uint8_t High_Hysteresis,
                                                     uint8_t Low_Threshold,
                                                     uint8_t Low_Hysteresis,
                                                     uint16_t Min_Time_Spent);
/* Enable or disable path loss reporting for a connection */
tBleStatus hci_le_set_path_loss_reporting_enable(uint16_t Connection_Handle,
                                                 uint8_t Enable);
/* Enable or disable the reporting of transmit power level changes in the
local and remote Controllers for a connection */
tBleStatus hci_le_set_transmit_power_reporting_enable(uint16_t Connection_Handle,
                                                      uint8_t Local_Enable,
                                                      uint8_t Remote_Enable);
/* Enable or disable the LE Power Control feature and procedure for a given PHY on the later established
connections */
tBleStatus aci_hal_set_le_power_control(uint8_t Enable,
                                        uint8_t PHY,
                                        int8_t RSSI_Target,
                                        uint8_t RSSI_Hysteresis,
                                        int8_t Initial_TX_Power,
                                        uint8_t RSSI_Filtering_Coefficient);
/* Report a path loss threshold crossing on a connection */
void hci_le_path_loss_threshold_event(uint16_t Connection_Handle,
                                      uint8_t Current_Path_Loss,
                                      uint8_t Zone_Entered);
/* Report the transmit power level on a connection */
void hci_le_transmit_power_reporting_event(uint8_t Status,
                                           uint16_t Connection_Handle,
                                           uint8_t Reason,
                                           uint8_t PHY,
                                           int8_t Transmit_Power_Level,
                                           uint8_t Transmit_Power_Level_Flag,
                                           int8_t Delta);
```

注意： *如需了解详细的命令/事件说明，请参阅 BLE API html 文档。*

3.19 测向指令和事件

BlueNRG-LPS 设备支持用于测向特性的两种方法（到达角和离开角），方法是管理：

- 数据包内的“constant tone extension (CTE) ”
- 用于 AoA 和 AoD 的天线切换机制。

BlueNRG-LPS 可以在链路层数据包中添加一个称为“constant tone extension”的新字段：该字段提供了一个恒定频率的信号，可以根据蓝牙核心规范 v5.1 对该信号进行 IQ 采样。

为了支持某些测向功能，控制器需要支持天线切换。BlueNRG- LPS 能够通过使用名为 ANTENNA_ID 的控制信号控制外部天线开关。该信号是一个 7 位天线标识符（ANTENNA_ID[6:0]），表示在特定时隙内使用的天线编号，由内部定序器实时提供。使用 7 位标识符时，可以控制的最大天线数量为 128。

在数据包的 CTE 字段期间，AoD 发射器或 AoA 接收器中的无线电需要切换天线。为此，可以在一些 I/O 上启用 ANTENNA_ID 信号（在相关的复用功能中对其进行编程）。为了更方便地执行正确的 ANTENNA_ID 信号配置，软件开发工具包中提供一条特定指令 aci_hal_set_antenna_switch_parameters()。

该信号需要提供给外部天线切换电路，其中 ANTENNA_ID[0]为要使用的天线标识符的最低有效位，ANTENNA_ID[6]为最高有效位。

注意： *在 BlueNRG-LPS 设备上，要使用的 I/O 为具有 ANTENNA_ID 复用功能（即从 PB0 到 PB6）。*

蓝牙核心规范 v5.1 定义新的 HCI 指令和事件，用于控制“constant tone extension”和 IQ 采样。特别是，天线切换方向图由以下 HCI 指令的 Antenna_IDs 参数控制：

- hci_le_set_connectionless_cte_transmit_parameters()
- hci_le_set_connectionless_iq_sampling_enable()
- hci_le_set_connection_cte_transmit_parameters()
- hci_le_set_connection_cte_receive_parameters()

方向图中指定的每个天线 ID 对应 BlueNRG- LPS 内部定序器生成的 ANTENNA_ID 编号，通过 PB[6:0]发送出去。

表 68. 测向指令和事件提供 HCI 指令和事件列表，这些指令和事件使用适当的 CTE 信息处理数据包的传输和接收。这些指令允许配置以下几个 CTE 方面：

- CTE 长度
- CTE 类型
- 天线切换方向图长度
- 天线 ID
- 时隙持续时间。

表 68. 测向指令和事件

输出参数	指令/事件和参数
tBleStatus	**aci_hal_set_antenna_switch_parameters** (uint8_t Antenna_IDs, uint8_t Antenna_ID_Shift, uint8_t Default_Antenna_ID, uint8_t RF_Activity_Enable)
tBleStatus	**hci_le_set_connectionless_cte_transmit_parameters** (uint8_t Advertising_Handle, uint8_t CTE_Length, uint8_t CTE_Type, uint8_t CTE_Count, uint8_t Switching_Pattern_Length, uint8_t Antenna_IDs[])
tBleStatus	**hci_le_set_connectionless_cte_transmit_enable** (uint8_t Advertising_Handle, uint8_t CTE_Enable)
tBleStatus	**hci_le_set_connectionless_iq_sampling_enable** (uint16_t Sync_Handle, uint8_t Sampling_Enable, uint8_t Slot_Durations, uint8_t Max_Sampled_CTEs, uint8_t Switching_Pattern_Length, uint8_t Antenna_IDs[])
tBleStatus	**hci_le_set_connection_cte_receive_parameters** (uint16_t Connection_Handle, uint8_t Sampling_Enable, uint8_t Slot_Durations, uint8_t Switching_Pattern_Length, uint8_t Antenna_IDs[])
tBleStatus	**hci_le_set_connection_cte_transmit_parameters** (uint16_t Connection_Handle, uint8_t CTE_Type, uint8_t Switching_Pattern_Length, uint8_t Antenna_IDs[])
tBleStatus	**hci_le_connection_cte_request_enable** (uint16_t Connection_Handle, uint8_t Enable, uint16_t CTE_Request_Interval, uint8_t Requested_CTE_Length, uint8_t Requested_CTE_Type)
tBleStatus	**hci_le_connection_cte_response_enable** (uint16_t Connection_Handle, uint8_t Enable)
tBleStatus	**hci_le_read_antenna_information** (uint8_t *Supported_Switching_Sampling_Rates, uint8_t *Num_Antenna, uint8_t *Max_Switching_Pattern_Length, uint8_t *Max_CTE_Length)
tBleStatus	**hci_le_transmitter_test_v3** (uint8_t TX_Channel, uint8_t Test_Data_Length, uint8_t Packet_Payload, uint8_t PHY, uint8_t CTE_Length, uint8_t CTE_Type, uint8_t Switching_Pattern_Length, uint8_t Antenna_IDs[])

输出参数	指令/事件和参数
tBleStatus	**hci_le_transmitter_test_v4** (uint8_t TX_Channel, uint8_t Test_Data_Length, uint8_t Packet_Payload, uint8_t PHY, uint8_t CTE_Length, uint8_t CTE_Type, uint8_t Switching_Pattern_Length, uint8_t Antenna_IDs[], int8_t Transmit_Power_Level)
void	**hci_le_connectionless_iq_report_event** (uint16_t Sync_Handle, uint8_t Channel_Index, uint16_t RSSI, uint8_t RSSI_Antenna_ID, uint8_t CTE_Type, uint8_t Slot_Durations, uint8_t Packet_Status, uint16_t Periodic_Event_Counter, uint8_t Sample_Count, Samples_t Samples[])
void	**hci_le_connection_iq_report_event** (uint16_t Connection_Handle, uint8_t RX_PHY, uint8_t Data_Channel_Index, uint16_t RSSI, uint8_t RSSI_Antenna_ID, uint8_t CTE_Type, uint8_t Slot_Durations, uint8_t Packet_Status, uint16_t Connection_Event_Counter, uint8_t Sample_Count, Samples_t Samples[])
void	**hci_le_cte_request_failed_event** (uint8_t Status, uint16_t Connection_Handle)

注意：

1. *如需详细说明，请参阅低功耗蓝牙 v3.1a 或更高版本的指令和事件文档。*
2. *测向功能仅受 BlueNRG-LPS 设备支持（不受 BlueNRG-LP 设备支持）。*

蓝牙规范允许在无连接和面向连接的通信中使用启用了测向功能的数据包，如下一节“第 3.19.1 节 无连接场景”和“第 3.19.2 节 面向连接场景”中所述。

3.19.1 无连接场景

在无连接场景中，CTE 字段在“periodic advertising train”的广播包中发送。要发送此类广播报文，主机需要遵循以下步骤：

1. 通过 aci_gap_set_advertising_configuration()配置扩展广播。
2. 在需要时通过 aci_gap_set_advertising_data()设置广播数据。在启用广播的同时也可以更改数据。
3. 使用 aci_gap_set_periodic_advertising_configuration()在相同的广播句柄上配置周期性广播。
4. 需要时，使用 aci_gap_set_periodic_advertising_data()设置周期性广播数据。在启用周期性广播的同时也可以更改数据。
5. 通过 hci_le_set_connectionless_cte_transmit_parameters()配置 CTE 传输参数。必须指定 CTE 的类型、使用 CTE 的数据包数量；如果是 AoD，还必须指定切换方向图。
6. 通过 hci_le_set_connectionless_cte_transmit_enable()启用 CTE 的传输。
7. 使用 aci_gap_set_periodic_advertising_enable()启用周期性广播。周期性广播实际上是在扩展广播启用之后启动的。
8. 通过 aci_gap_set_advertising_enable()启用扩展广播。

在扫描仪侧，为了接收周期性广播并从 CTE 字段中提取 IQ 样本，主机必须执行以下步骤：

1. 通过 aci_gap_set_scan_configuration()配置扩展扫描。
2. 通过 aci_gap_start_procedure()启动扩展扫描流程。
3. 使用 aci_gap_periodic_advertising_create_sync()，实现与“periodic advertising train”同步。
4. 如果不再需要，可以通过 aci_gap_terminate_proc()停止扫描流程。
5. 通过 hci_le_set_connectionless_iq_sampling_enable()启用 IQ 采样。需要为 AoA CTE 指定切换方向图。
6. 使用 hci_le_connectionless_iq_report_event()处理接收到的 IQ 样本。

注意：

- *如要接收扫描仪上的无连接 IQ 报告事件，还必须通过 hci_le_set_event_mask()指令进行启用。*
- *扫描仪的某些步骤中可能需要的其他一些事件在默认情况下处于未启用状态。建议至少启用：*
  - *LE 扩展广播报告事件*
  - *LE 定期广播同步建立事件*
  - *LE 定期广播报告事件*
  - *LE 定期广播同步丢失事件*
- *IQ 样本算法不是由蓝牙核心规范定义的*
- *如果不再需要，可以使用 hci_le_set_connectionless_iq_sampling_enable()禁用接收 IQ 报告。*

3.19.2 面向连接的场景

在面向连接的场景下，外设和中心设备都可以向对端设备发送一个请求（LL_CTE_REQ PDU），要求对端设备发送一个包含 CTE 字段的报文（LL_CTE_RSP PDU）。

假设两台设备之间存在连接，主机需要遵循以下步骤。在希望接收带有 CTE 字段的数据包的设备上：

1. 通过 hci_le_set_connection_cte_receive_parameters()配置 CTE 接收参数。仅当设备希望请求 AoA CTE 类型时，才需要指定 IQ 采样的时隙持续时间和天线切换方向图。
2. 通过 hci_le_connection_cte_request_enable()启用请求的自动发送。使用此函数时，必须指定 CTE 类型（AoA/AoD）。
3. 使用 hci_le_connection_iq_report_event()处理接收到的 IQ 样本

在希望在连接期间支持发送 CTE 字段的设备上：

1. 通过 hci_le_set_connection_cte_transmit_parameters()配置 CTE 传输参数。必须指定 CTE 的类型，以及在 CTE 类型为 AoD 时使用的天线方向图。
2. 通过 hci_le_connection_cte_response_enable()启用 CTE 响应。从此刻起，控制器自动响应对等发来的 CTE 请求。

注意：

- *如要接收连接 IQ 报告事件，必须通过 hci_le_set_event_mask()指令进行启用。*
- *蓝牙核心规范没有定义 IQ 采样算法。*
- *如果不再需要，可以使用 hci_le_connection_cte_request_enable()禁用接收 IQ 报告。*

4 低功耗蓝牙协议栈 v3.x 调度程序

本节旨在为用户提供新版低功耗蓝牙协议栈 v3.x 时间调度程序的基础知识。

其目的是帮助用户在多链路场景中建立连接、扫描和广播流程，以便测得的性能（例如，吞吐率、延迟、发生连接掉线情况时的稳健性）尽可能与预期一致。

此外，本节还介绍了低功耗蓝牙协议栈 v2.x 时间调度程序的优点/缺点。

所有链路层（LL）**无线电任务**（即连接、广播、或扫描）都可以通过时隙序列（即相关事件发生的时间窗口）在时间基准中表示。

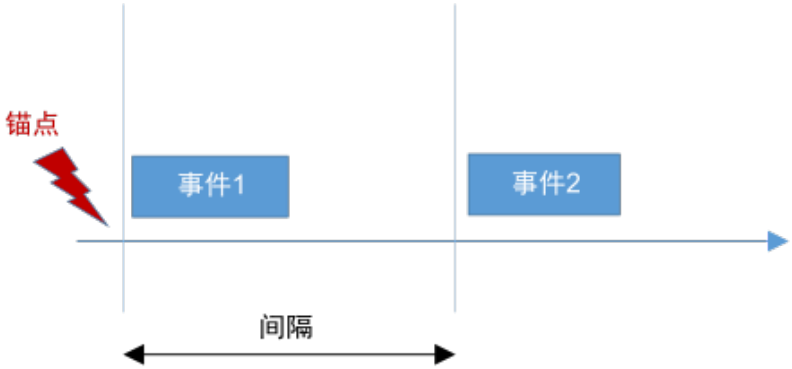
无线电任务占用的每个时隙可以通过以下方式表示：

- **锚点**，即该时隙预期开始的确切时间；
- **长度**，即预期的时隙持续时间。

无线电任务可分为：

- **周期性任务**，时隙以固定的时间间隔重复（例如，连接事件）
- **异步任务**，其中：
  - 时隙不重复（通常称为一次性时隙）
  - 时隙重复，但没有固定的时间间隔（例如，广播事件）

图 19. 周期性无线电任务示例



由于多个无线电任务可以同时在一台设备上执行，因此相关时隙可以在时间上重叠。在下文中，时隙重叠被称为冲突。

如果发生时隙冲突，其中只有一个时隙被授予“access to the air”，并被称为**“预定时隙”**。没有获得“access to the air”的冲突时隙被称为**“被跳过的时隙”**。

时间调度程序是一种软件模块，提供以下功能：

- 当主机启动一个新的无线电任务时，它会计算相关时隙序列的第一个锚点（即，与无线电任务关联的第一个 Rx/Tx 事件预定“grab the air”的确切时间）
- 在一个时隙结束的那一刻，调度器会依据蓝牙标准的特定规则（即，考虑到这么一种情况，在那一刻，蓝牙连接事件的主机端，其 CE_length 长度可能会超出时隙的结束点）：
  1. 它计算相关无线电任务的下一个锚点（如果有）；
  2. 如果设备上当前存在多个无线电任务，则会为每个无线电任务计算相对于当前时间的下一个锚点（如果有）；
  3. 如果设备上当前存在多个无线电任务，则会根据时间顺序选择下一个必须服务的无线电任务；
  4. 如果第一个时隙（按时间顺序）与其他时隙冲突，则将根据优先级机制重新安排其中一个冲突时隙。

无线电任务数量

当一个蓝牙活动（例如，一个新的广播集或连接）启动时，通常需要一个单一无线电任务。但是，有些蓝牙活动需要多个无线电任务。

表 69. 低功耗蓝牙协议栈活动和无线任务

低功耗蓝牙协议栈活动	无线电任务数量
广播集、传统广播	1
广播集、扩展广播	2
扫描、扩展广播和扫描（已禁用）	1
扫描、扩展广播和扫描（已启用）	1 + NumOfAuxScanSlots
连接	1
周期性广播	1
周期性广播同步	1

特别是，如果使用扩展广播 PDU，则每个广播集的无线电任务数量为 2。此外，如果通过模块化配置（CONTROLLER_EXT_ADV_SCAN_ENABLED）启用扩展扫描，则扫描需要的时隙数量等于 1 + NumOfAuxScanSlots（这是传递给 BLE_STACK_Init()函数初始化结构的一个参数）。NumOfAuxScanSlots 等于可同时分配并用于对辅助广播物理通道进行扫描的无线电任务数。一旦接收到主广播物理通道上的数据包（该数据包指向辅助广播物理通道上的数据包），就分配一个无线电任务；如果该任务进入被服务状态，则触发对辅助广播物理通道的扫描。如果多台广播设备使用扩展广播事件，则辅助扫描时隙数量越多，接收扩展广播事件的机会就越高。

可以分配的无线电任务总数通过初始化结构 BLE_STACK_InitTypeDef 的 NumOfLinks 字段指定。除了该约束，同时发生的周期性广播同步时隙的数量还受到 NumOfSyncSlots 字段的限制。

4.1 低功耗蓝牙协议栈 v2.x 时间调度程序的限制

低功耗蓝牙协议栈 v2.x 时间调度程序围绕锚定周期的概念构建。

锚定周期是最小时间间隔，用来表示在一台设备上运行的所有周期性“主”无线电任务。注意，我们用“主”无线电任务来表示蓝牙 4.2 标准的广播、扫描和主连接事件（从连接事件除外）。

根据该定义，时间基准表示为锚定周期的重复，其中所有主无线电任务必须有时间间隔，而间隔是锚定周期的整数倍。

锚定周期在主机启动第一个“主”无线电任务的时间开始，它等于无线电任务间隔。可以减少锚定周期（即，如果启动一个新的主无线电任务，其间隔是当前锚定周期的整数倍），但是在减少之后不能增加。

在下图中，表示一个锚定周期内的两个主无线电任务分配。这里，无线电任务#0 的间隔与锚定周期相同，而无线电任务#1 的间隔是锚定周期的三倍。

图 20. 在一个锚定周期内分配两个无线电任务的示例

下面列出了低功耗蓝牙协议栈 v2.x 时间调度程序的主要优点：

1. 调度程序强制应用程序为新时隙选择合适的参数，以避免与现有时隙冲突。如果新的主时隙无法分配（因其会与其他时隙冲突），则调度程序只返回一个错误。在这种情况下，主时隙之间不会发生冲突，从而保证了最大吞吐量。
2. 只有从连接时隙才会冲突。这里，时间调度程序使用的优先级机制是一种简单的轮询方法

下面列出了低功耗蓝牙协议栈 v2.x 时间调度程序的主要限制：

1. 由于分配的主无线电任务长度的总和必须在锚定周期内，因此并发主无线电任务的数量有限。
2. 由于新的主无线电任务的间隔必须是现有锚定周期的整数倍，因此用户在选择新链路层时隙（即连接、广播，以及扫描间隔）周期时的灵活性非常低。
3. 一旦被减少（由于启动新的无线电任务），锚定周期就不能再增加，从而导致并发无线电任务的数量受到额外限制。
4. 异步无线电任务与锚定周期方法不兼容，因为它们不具备周期性。

蓝牙 5.0 标准引入的几乎所有链路层特性（例如，广播扩展）都以异步事件为特征，这与低功耗蓝牙协议栈 v2.x 时间调度程序的锚定周期方法不兼容；这是导致实现新时间调度程序的主要原因。

多个时隙发生冲突的概率也与并发异步无线电任务的数量成正比，这使得低功耗蓝牙协议栈 v2.x 时间调度程序的简单轮询机制不再适合保证足够的性能。

4.2 新版低功耗蓝牙协议栈 v3.x 时间调度程序

为了支持蓝牙 5.0 标准引入的所有新链路层特性，并允许用户实现具有不同 QoS 要求的应用场景，新版低功耗蓝牙协议栈 v3.x 时间调度程序消除了以前实现的主要约束。

在低功耗蓝牙协议栈 v3.x 时间调度程序中，所有无线电任务都被视为异步任务，意味着它们被排除在锚定周期表示之外。

相对于低功耗蓝牙协议栈 v2.x 时间调度程序的一个显著优势是：由主机启动的 LL 活动永远不会因为其周期性（即连接、广播、或扫描间隔）或事件持续时间（即，连接 CE_Length、扫描窗口等）而被拒绝。

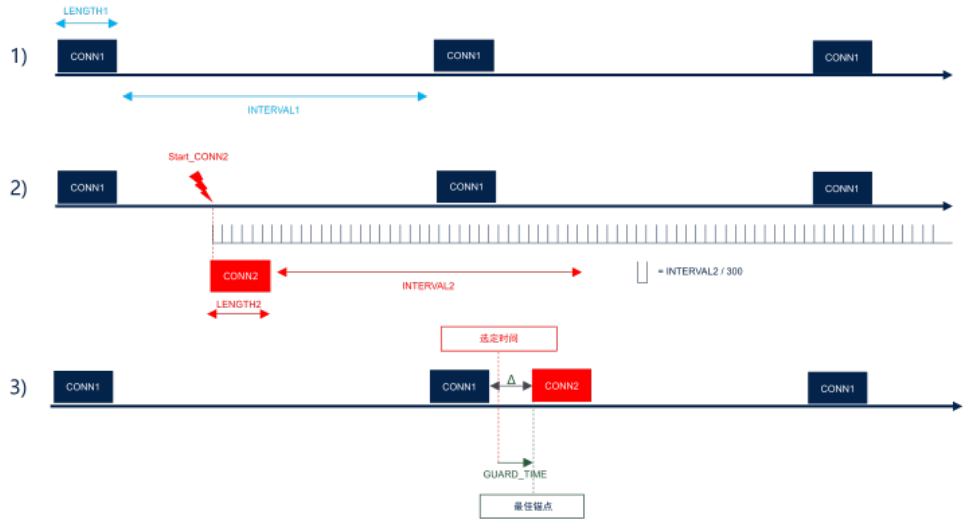
这种新方法的主要缺点是主任务时隙（即，蓝牙连接、广播，以及扫描事件）现在可能会相互冲突。为此，用户偶尔会体验到性能下降（例如，吞吐量下降）。如果遵循某些指导方针，可以提高性能（请参阅“第 4.4 节用户指南”）。

4.3 预调度程序

预调度程序是低功耗蓝牙协议栈 v3.x 时间调度程序的一个软件模块，负责为新启动的主连接选择“最佳锚点”。

最佳锚点是第一个连接时隙的开始时间，将与现有时隙发生冲突的概率降至最低，因为其观察期是新启动连接的连接间隔的 10 倍。

图 21. 预调度程序操作示例



上图提供了一个预调度程序操作示例：

1. 设备已经运行了一个具有自己的时隙长度和间隔的无线电任务（CONN1）
2. 在 Start_CONN2 时间，主机发出指令以启动新连接（即，主角色），并请求预调度程序选择最佳锚点
   a. 创建一个潜在最佳锚点网格，从 Start_CONN2 时间开始，结束于 *（Start_CONN2 + 10 * INTERVAL2）*
      - 网格中的点按时间间隔排列（INTERVAL2/300）
   b. 计算网格中每个点与现有时隙的冲突次数
3. 选取网格中的一个点（图中为 SELECTED_TIME），该点：
   a. 与现有时隙的冲突概率为最低
   b. 尽可能减少与现有时隙的时间间隔（以便更高效地使用带宽）
4. 最佳锚点（图中的 BEST_ANCHOR）等于（SELECTED_TIME + GUARD_TIME），其中 GUARD_TIME 是对时间调度程序在当前预定时隙结束后重新调度新无线电任务（即，对关联的下一个时隙锚点进行编程）所花费的最大时间的估计。

预调度程序选择最佳锚点需要的时间随着当前活跃无线电任务的数量呈线性关系。该时间间隔的范围是 100ms（有 8 个当前活跃的无线电任务）到 1 秒（有 128 个当前活跃的无线电任务）。

预调度程序操作不考虑广播无线电任务，因其通常不是周期性任务；除了周期性广播时隙，预调度程序不考虑这些时隙，原因在于其持续时间差别很大。

预调度程序也不会考虑扫描，因其作为后台操作处理，优先级较低。扫描窗口（即，设备在特定通道上连续接收的时间）被划分为持续时间为 10ms 的扫描块。每当一个扫描块预计会与另一个时隙发生冲突时，就会缩短该扫描块以尽量避免冲突。根据设计，扫描块的持续时间不能减少到低于某个特定的最小值（被硬编码在固件中），因此扫描时隙仍然可能与其他时隙发生冲突。

4.4 用户指南

如前所述，BLE 协议栈 v2.x 预调度程序能够避免主无线电任务之间的冲突（方法是对新时隙的分配添加约束），而 BLE 协议栈 v3.x 预调度程序方法旨在将与现有无线电任务发生冲突的概率降至最低，但不会阻止分配预期会发生冲突的时隙。为此，如未正确选择连接参数，用户可能会体验到较差的性能（相对于 BLE 协议栈 v2.x）。

以下是为用户提供的一些基本建议，可提高 BLE 协议栈 v3.x 时间调度程序在此类场景（设备是多个从设备的主设备）中的性能：

1. 为所有主连接选择相同的连接间隔（Conn_Interval），使其能够包含所有主连接时隙的连接事件长度（CE_Length）加上保护时间的总和，其中：

$$Conn\_Interval > \sum_i \left(\left\lceil \frac{CE\_Length_i}{2} \right\rceil + GUARD \right) \quad (2)$$

其中：*GUARD = max_num_of_radio_tasks/25*

注意：

- *CE_Lengthi 是 i-th 连接的 CE_Length*
- *CE_Length 以 0.625 ms 为时间单位*
- *Conn_Interval 以 1.25 ms 为时间单位*
- *GUARD 以 1.25 ms 为时间单位*
- *CE_Length(i)/2 四舍五入到下一个整数值*
- *GUARD 是大于或等于受支持的最大无线电任务数除以 25 的得数的最小整数。换句话说：*
  - *1 如果 max_num_of_radio_tasks ≤ 25；*
  - *2 如果 25 < max_num_of_radio_tasks ≤ 50，诸如此类。*

为简单起见，做了以下假设：

- *Minimum_CE_Length = Maximum_CE_Length = CE_Length*
- *Conn_Interval_Min = Conn_Interval_Max = Conn_Interval*
- 如果 *Minimum_CE_Length ≠ Maximum_CE_Length*，则预调度程序将仅使用 *Max_CE_Length*。
- 如果 *Conn_Interval_Min ≠ Conn_Interval_Max*，则预调度程序将仅使用 *Conn_Interval_Min*。
- 此外，考虑到协议栈使用的 CE_Length 的实际值可能与应用程序传递的值不同：堆栈使用的最小值（取决于库模块化配置）如下表中所示。

表 70. CE_Lenght 最小值

CONTROLLER_DATA_LENGTH_-EXTENSION_ENABLED	CONTROLLER_2M_CODED_PHY-_ENABLED	CE_Length 最小值
-	-	3
X	-	8
-	X	10
X	X	56

- -：特性未启用
- X：特性已启用

2. 选择允许所需吞吐量的 CE_Length 和 Conn_Interval 的组合。

吞吐量的计算公式如下：

Throughput = num_bits_CE/Conn_Interval_ms

其中，num_bits_CE 为同一连接事件中传输的位数，Conn_Interval_ms 为连接间隔，以 625µs 为单位。

同一连接事件中传输的位数取决于链路层 PDU 的最大长度和每个连接事件的数据包数量，而后者又取决于 CE_length 参数。

如果 Conn_Interval 已给定，则下面的公式给出了为达到期望的吞吐量，每个连接事件需要传输的数据包数量：

$$num\_packets\_CE = [(Throughput \times Conn\_Interval\_ms) / (bytes\_per\_packets \times 8)] \quad (3)$$

其中：

- *吞吐量以 kbps 为单位*
- *Conn_Interval_ms 以 ms 为单位*
- *bytes_per_packets 是 LLn 有效负载中包含的字节数*

在连接过程中，通过数据长度更新流程确定可以发送或接收的 LL PDU 的最大长度。LL 有效负载长度可以通过 hci_le_write_suggsted_default_data_length()指令或 hci_le_set_data_length()指令设置。默认情况下，一个 LL PDU 的有效负载长度为 27 个八位字节，如果支持数据长度扩展特性（在 BLE 协议栈 3.x 中，此步通过 CONTROLLER_DATA_LENGTH_EXTENSION_ENABLED 宏完成），则可以增加到 251 个八位字节。

LL PDU 的持续时间如下表所示，包括消息完整性检查（MIC）字段的传输。Payload_length 不包含 MIC。

表 71. LL PDU 持续时间（包含 MIC）

PHY	Empty LL PDU 持续时间（µs）	Max LL PDU 持续时间（µs）
LE_1M	*80*	*(payload_length + 14) x 8*
LE_2M	*44*	*(payload_length + 15) x 4*
LE_CODED (S=8)	*720*	*976 + payload_length x 64*

连接事件由主从设备之间交换的几个数据包组成。CE 长度的计算公式如下：

$$CE\_Length = [num\_packets\_CE \times (TX\_PDU\_Duration + RX\_PDU\_Duration + 2 \times T\_IFS) / 625] \quad (4)$$

其中：

- *TX_PDU_Duration* 是传输数据 PDU 所需要的以微秒为单位的值（参见表 71. LL PDU 持续时间（包含 MIC））。如未获悉该值，则应使用可能的最大值。
- *RX_PDU_Duration* 是接收数据 PDU 所需要的以微秒为单位的值（参见表 71. LL PDU 持续时间（包含 MIC））。如未获悉该值，则应使用可能的最大值。
- *T_IFS* 等于 150 µs。
- *CE_Length* 以 625µs 为单位，四舍五入到下一个整数值。

如果 CE 长度的结果值大于连接间隔，或者更概括地说，不满足条件（2），则需要增加连接间隔，并且需要重新计算每个连接事件的数据包数量。如果不能增加连接间隔，则无法达到期望的吞吐量。

示例：

要求具有 100 kbps 的单向吞吐量（LL 数据吞吐量），连接间隔为 50 ms。如果启用数据长度扩展（LL 有效负载可达 251 个八位字节），则数据仅在一个方向发送（RX 上的数据包为空），PHY 为 LE_1M：

num_packets_CE = (100 x 50)/(251 x 8) = 3

CE_Length = 3 x (2120 + 80 + 300)/625 = 12 -> CE_Length_ms = 7.5 ms

CE 长度小于连接间隔，不需要增加。

3. 选择的 Conn_Interval 应该小于所有连接所需的最大延迟。

如果主连接可以容忍比 Conn_Interval 更大的延迟，用户可以为该连接选择一个值为 Conn_Interval 的整数倍的连接间隔。

4.5 具有三个主连接的场景示例

如果我们假设应用程序需要启动三个具有不同吞吐量和延迟要求的主连接，例如：

- 应用 1：
  - 吞吐量 = 10 kbps，双向
  - 延迟 = 100 ms
- 应用 2：
  - 吞吐量 = 20 kbps，双向
  - 延迟 = 100 ms

- 应用 3：
  - 吞吐量 = 30 kbps，双向
  - 延迟 = 200 ms

最大连接间隔值，保证所有主连接所需的延迟等于 100 ms：

$$Connection\_Interval \leq 100ms \quad (5)$$

现在，我们必须验证 100 ms 的连接间隔是否也适合包含所有主连接 CE_Lengths。

如果未启用数据长度扩展，则 PHY 为 1 Mbps。

- 应用程序 1 必须在每个连接事件中传输(10 x 100)/(27 x8) = 5 个数据 PDU（每个 27 字节）-> CE_Lenght1 = 5 x ((27+14)x8 + (27+14)x8 + 2x150)/625 = 8，（即，5 毫秒）
- 应用程序 2 必须在每个连接事件中传输(20 x 100)/(27 x8) = 10 个数据 PDU（每个 27 字节）-> CE_Lenght2 = 10 x ((27+14)x8 + (27+14)x8 + 2x150)/625 = 16，（即，10 毫秒）
- 应用程序 3 必须在每个连接事件中传输(30 x 100)/(27 x8) = 14 个数据 PDU（每个 27 字节）-> CE_Lenght3 = 14 x ((27+14)x8 + (27+14)x8 + 2x150)/625 = 22，（即，13.75 毫秒）

适合所有连接 CE_Lengths 的最小连接间隔是：

Conn_Interval_Min_allowed = SUM(Max_CE_Length(i)/2 + GUARD) = (4 + 1) + (8 + 1) + (11 + 1) = 26 --> 32.5 ms

其中，GUARD = 1，假设设备支持的最大无线电任务数不超过 25。

由于 *Conn_Interval_Min_allowed ≤ 100 ms*，因此可以为所有主连接使用 100 ms 的连接间隔，以确保适当的性能。

4.6 优先级机制

在一个时隙结束时，时间调度程序负责计算要服务的下一个时隙（以及下一个无线电任务）。

如果设备上当前有多个无线电任务处于活跃状态，按时间顺序排列的下一个时隙可能与其他时隙冲突。在这种情况下，基于优先级机制选择下一个时隙。

优先级机制包括：

1. 根据以下公式为每个无线电任务分配优先级：

$$priority = \min priority\_max\ ,\ priority\_min + \log 2\ skipped\_slots + 1 \quad (6)$$

其中：

- *priority_max* 和 *priority_min* 是特定于每个无线电任务的常量（就是说，不同于广播、扫描，以及连接无线电任务）
- *skipped_slots* 针对因冲突而被跳过的、与特定无线电任务关联的连续时隙，指示时隙的当前数量
  - 根据该定义，与无线电任务相关联的优先级是一个动态值，在相关联的时隙每次被服务或跳过时进行计算。

2. 从发生冲突的无线电任务中选择具有最高优先级的一个任务。

4.7 与 ISR 稳健性机制的相互作用

ISR 稳健性机制的实现目的是让 FW 在中断服务程序（ISR ）的执行发生延迟时表现稳健。

ISR 延迟通常是由于应用程序禁用时间无线电中断过长，如果不能正确恢复，可能导致协议栈的行为不可预测。

特别是，从 LL 协议的角度来看，关键点在于所谓的背靠背操作，即事件（接收或传输）必须在很短的时间间隔（即 150µs）内触发其他事件。

背靠背操作的一个典型例子是：设备接收到扫描请求，进而触发扫描响应的传输。在这种时间关键型情况下，与扫描请求的接收相关的 ISR 执行延迟可能导致与扫描响应传输相关的 HW 寄存器在传输发生时未被正确设置，从而引起不可预测的行为（例如，传输了畸形数据包）。

通过 ISR 稳健性机制，FW 能够检测到 ISR（相对于相关无线电事件）接受服务过晚的情况。当 FW 检测到这种特定情况时：

1. 相关的背靠背无线电事件被跳过
2. 调用 BLE 协议栈 v3.x 时间调度程序，后者选择要服务的下一个无线电任务
3. HW 错误事件被推送到应用程序，以通知发生了错误。

不期望应用程序以特定的方式处理 HW 错误事件，但是在接收时，用户得到一个指示，表明应用程序的设计在无线电中断的掩蔽时间方面并非 100%正确。

5 参考文件

表 72. 参考文件

名称	标题/描述
蓝牙规范	蓝牙系统规范（v5.x）
DS13282	BlueNRG-LP 数据手册
STSW-BNRGUI	BlueNRG GUI 软件包
DS13819	BlueNRG-LPS 数据手册

6 缩写和缩略语列表

本节列出了文档中使用的标准缩写和缩略语。

表 73. 缩略语列表

术语	意义
ACI	应用命令接口
AoA	到达角（AoA）
AoD	离开角（AoD）
ATT	属性协议
低功耗蓝牙	低功耗蓝牙
BR	基础速率（BR）
CRC	循环冗余校验
CSRK	连接签名解析密钥
CTE	固定频率扩展信号
EDR	增强数据率
DK	开发套件
EXTI	外部中断
GAP	通用访问配置文件
GATT	通用属性参数文件
GFSK	高斯频移键控
HCI	主机控制器接口
IFR	信息寄存器
IRK	身份解析密钥
ISM	工业、科学和医疗
LE	低能耗
L2CAP	逻辑链路控制和适配协议
L2CAP-COS	逻辑链路控制适配层协议 - 面向连接的服务
LTK	长期密钥
MCU	微控制器单元
MITM	中间人
NA	不适用
NESN	下一个序号
OOB	带外
PDU	协议数据单元
射频器件	射频
RSSI	接收的信号强度指示
SIG	技术联盟
SM	安全管理器
SN	序号

术语	意义
USB	通用串行总线
UUID	通用唯一标识符
WPAN	无线个人局域网

版本历史

表 74. 文档版本历史

日期	版本	变更
2020 年 8 月 3 日	1	初始版本。
2021 年 1 月 29 日	2	更新了：第 1.2 节 物理层、第 1.2.2 节 LE 2M PHY、第 1.2.3 节 LE CODED PHY、第 1.3 节 链路层（LL）、第 1.3.1 节 低功耗蓝牙数据包、第 1.3.2 节 扩展广播、第 1.3.3 节 广播集、第 3.1.1 节 低功耗蓝牙地址、第 3.16 节 低功耗蓝牙 2 Mbps 和 Coded PHY、第 4 节 BLE 协议栈 v3.x 调度程序。 新增了：第 1.3.6.1 节 扩展扫描、第 3.17.1 节 扩展广播和扫描事件、第 4.2 节 新的低功耗蓝牙协议栈 v3.x 时间调度程序、第 4.3 节 预调度程序、第 4.4 节 用户指南、第 4.5 节 具有三个主连接的场景示例、第 4.6 章节 优先级机制，以及第 4.7 节 与 ISR 稳健性机制的相互作用。 更新了表 29. BLE 协议栈 v3.x 初始化参数
2021 年 5 月 6 日	3	新增了：第 1.3.7 节 周期性广播和周期性广播同步传输、第 1.3.8 节 随机广播、第 1.9.5 节 GATT 缓存、第 1.3.9 节 低功耗蓝牙功率控制、第 3.18 节 周期性广播模式、第 3.18.1 节 周期性广播模式、第 3.18.2 节 周期性广播同步模式、第 3.18.3 节 周期性广播同步建立流程、第 3.18.5 节 LE 功率控制和路径损耗监测。 更新了图 15. BLE 协议栈 v3.x 架构、第 2.1 节 BLE 协议栈库框架、第 2.3 节 BLE 协议栈 init 和 tick API、以及第 2.3 节 BLE 协议栈 init 和 tick API，以及第 3.3 节 GAP API 接口。 增加了表 33. GATT、GAP 特征句柄、表 35. 发射功率等级，En_High_Power = 0（SMPS 电压@ 1.4 V）和表 36. 发射功率等级，En_High_Power = 1（SMPS 电压 @ 1.9 V） 少量文字改动。
2021 年 6 月 7 日	4	更新了第 3.1 节 初始化阶段和应用程序主循环

日期	版本	变更
2022 年 4 月 6 日	5	**更新了：**章节简介、第 2.1 节 BLE 协议栈库框架、第 2.3 节 BLE 协议栈 init 和 tick API、第 2.4 节 BLE 协议栈 v3.x 应用程序配置、第 3.1 节 初始化阶段和应用程序主循环，以及第 6 章节 缩写和缩略语列表。 **更新了**图 15. 低功耗蓝牙（BLE）协议栈 v3.x 架构。 **新增了：**第 3.19 节 测向指令和事件、第 1.11.1 节 通过到达角（AoA）进行测向、第 1.11.2 节 通过离开角（AoD）进行测向、第 1.11.3 节 同相（I）和正交（Q）、第 3.19 节 测向指令和事件，以及第 5 章节 参考文献。

目录

表格索引

图片目录

重要通知 - 请仔细阅读

意法半导体公司及其子公司（“意法半导体”）保留随时对 ST 产品和/或本文档进行变更、更正、增强、修改和改进的权利，恕不另行通知。买方在订货之前应获取关于意法半导体产品的最新信息。意法半导体产品的销售依照订单确认时的相关意法半导体销售条款。

买方自行负责对意法半导体产品的选择和使用，意法半导体概不承担与应用协助或买方产品设计相关的任何责任。

意法半导体不对任何知识产权进行任何明示或默示的授权或许可。

转售的意法半导体产品如有不同于此处提供的信息的规定，将导致意法半导体针对该产品授予的任何保证失效。

ST 和 ST 标志是意法半导体的商标。关于意法半导体商标的其他信息，请访问 www.st.com/trademarks。其他所有产品或服务名称是其各自所有者的财产。本文档中的信息取代本文档所有早期版本中提供的信息。