life.augmented

PM0237
编程手册

BlueNRG、 BlueNRG-MS 栈编程指南

前言

本文档的主要目的是为开发人员提供一些参考性的关于如何使用 BlueNRG、 BlueNRG-MS 栈 API 开发低功耗蓝牙 （BLE）主机应用的编程指南。

本文档将介绍 BlueNRG 和 BlueNRG-MS 栈架构、 API 接口和回调，通过它们可以访问 BlueNRG 和 BlueNRG-MS 网络协处理器提供的低功耗蓝牙功能。

本编程手册还提供一些与低功耗蓝牙 （BLE）技术有关的基本概念，以便将 BlueNRG 和 BlueNRG-MS API、参数及相关事件与 BLE 协议栈特性联系起来。编者假设用户已具备关于 BLE 技术及其主要特性的基本知识。

关于 BlueNRG、 BlueNRG-MS 设备以及蓝牙规范 v4.0 和 v4.1 的完整信息，请参考本文档末尾的*第 5 节：参考文献*。

BlueNRG 是一种功率极低的低功耗蓝牙 （BLE）单模网络处理器，符合蓝牙规范 v4.0 并支持主设备或从设备角色。

BlueNRG-MS 是一种功率极低的低功耗蓝牙 （BLE）单模网络处理器，符合蓝牙规范 v4.1 并同时支持主设备和从设备角色。

手册结构如下：

- 低功耗蓝牙 （BLE）技术的基本原理
- BlueNRG 和 BlueNRG-MS 栈架构以及应用指令接口 （ACI）概述
- 如何使用 BlueNRG 和 BlueNRG-MS 栈 ACI API 设计应用

注： *本文内容适用于 BlueNRG 和 BlueNRG-MS 设备。必要时，会着重标明具体的区别。*

目录

1 低功耗蓝牙技术 7

1.1 BLE 栈架构 8

1.2 物理层 9

1.3 链路层 （LL） 11

1.3.1 BLE 数据包 12

1.3.2 广告状态 15

1.3.3 扫描状态 16

1.3.4 连接状态 17

1.4 主机控制器接口 （HCI） 18

1.5 逻辑链路控制和适配协议 （L2CAP） 18

1.6 属性协议 （ATT） 18

1.7 安全管理器 （SM） 20

1.8 通用属性配置文件 （GATT） 22

1.8.1 特征属性类型 22

1.8.2 特征描述符类型 24

1.8.3 服务属性类型 24

1.8.4 GATT 流程 25

1.9 通用访问配置文件 （GAP） 26

1.10 BLE 配置文件和应用 31

1.10.1 接近感测示例 32

2 BlueNRG 和 BlueNRG-MS 栈架构以及 ACI 34

2.1 ACI 接口 35

2.2 ACI 接口资源 36

2.3 其他平台资源文件 38

2.3.1 平台配置 38

2.4 如何将 ACI SPI 接口框架移植到选定微控制器 40

3 使用 BlueNRG、 BlueNRG-MS ACI API 设计应用 41

3.1 初始化阶段和应用程序主循环 42

3.1.1 BLE 地址 47

3.1.2 设置发送功率水平 49

3.2 BlueNRG、 BlueNRG-MS 事件和事件回调 50

3.3 服务和特征配置 54
3.4 创建连接：可发现和可连接 API 56
3.4.1 设置可发现模式并使用直接连接建立流程 58
3.4.2 设置可发现模式并使用一般发现流程 （主动扫描） . 60
3.5 安全 （配对和绑定） 65
3.6 私有 69
3.6.1 BlueNRG （蓝牙 v4.0） 69
3.6.2 BlueNRG-MS （蓝牙 v4.1） 70
3.6.3 解析地址 71
3.7 服务和特征发现 71
3.7.1 服务发现流程与相关 GATT 事件 73
3.7.2 特征发现流程与相关 GATT 事件 77
3.8 特征通知 / 指示、写入、读取 80
3.9.1 基本 / 典型错误条件描述 84
3.10 BlueNRG-MS 同时作为主、从设备的场景 84

4 BlueNRG 多连接时序策略 88
4.1 关于低功耗蓝牙时序的基本概念 88
4.1.1 广告时序 88
4.1.2 扫描时序 89
4.1.3 连接时序 89
4.2 BlueNRG 时序和时隙分配概念 89
4.2.1 为第一个主设备连接设置时序 90
4.2.2 为其他主设备连接设置时序 91
4.2.3 广告事件时序 92
4.2.4 扫描时序 93
4.2.5 从设备时序 93
4.3 BlueNRG 多个主设备和从设备连接指南 93

5 参考 95

附录 A 缩写和缩略语列表 96

6 版本历史 98

图片索引

图 1. 支持低功耗蓝牙技术的以钮扣电池供电的设备 7
图 2. 低功耗蓝牙栈架构 9
图 3. 链路层状态机 11
图 4. 数据包结构 12
图 5. 具有 AD 类型标记的广告数据包 15
图 6. 特征定义示例 23
图 7. 客户端和服务器配置文件 32
图 8. BlueNRG、 BlueNRG-MS 栈架构和与外部主机的接口 34
图 9. MAC 地址存储. 48
图 10. BlueNRG-MS 同时作为主、从设备的场景 85
图 11. 广告时序 88
图 12. 三个连接时隙的分配示例 90
图 13. 三个连续连接的时序分配示例 92

表格索引

表 1. BLE RF 通道类型和频率 . . . 10
表 2. 广告数据头文件内容 . . . 12
表 3. 广告数据包类型 . . . 13
表 4. 广告事件类型和允许的响应 . . . 14
表 5. 数据包包头内容 . . . 14
表 6. 数据包长度字段和有效值 . . . 14
表 7. 连接请求时序间隔 . . . 17
表 8. 属性示例 . . . 19
表 9. 属性协议消息 . . . 19
表 10. BLE 设备上输入 / 输出功能的组合 . . . 20
表 11. 计算临时密钥 （TK）的方法 . . . 21
表 12. 特征声明 . . . 23
表 13. 特征值 . . . 24
表 14. 服务声明 . . . 24
表 15. 包含声明 . . . 25
表 16. 发现流程和相关响应事件 . . . 25
表 17. 客户端发起的流程和相关响应事件 . . . 25
表 18. 服务器发起的流程和相关响应事件 . . . 26
表 19. GAP 角色 . . . 26
表 20. GAP 广播器模式 . . . 27
表 21. GAP 可发现模式 . . . 28
表 22. GAP 可连接模式 . . . 28
表 23. GAP 可绑定模式 . . . 28
表 24. GAP 监听流程 . . . 28
表 25. GAP 发现流程 . . . 29
表 26. GAP 连接流程 . . . 30
表 27. GAP 绑定流程 . . . 30
表 28. ACI 接口 . . . 35
表 29. ACI 接口资源文件 . . . 37
表 30. 软件框架平台驱动 . . . 38
表 31. BLE 设备角色的用户应用定义 . . . 41
表 32. BlueNRG GATT、 GAP 默认服务 . . . 44
表 33. BlueNRG GATT、 GAP 默认特征 . . . 44
表 34. BlueNRG-MS GATT、 GAP 默认服务 . . . 45
表 35. BlueNRG-MS GATT、 GAP 默认特征 . . . 45
表 36. GAP_Init （) 角色参数值 . . . 46
表 37. ACI：主要事件、子事件 . . . 50
表 38. ACI：GAP 模式 API . . . 56
表 39. ACI：发现流程 API . . . 57
表 40. ACI：连接流程 API . . . 57
表 41. ADV_IND 事件 . . . 64
表 42. ADV_IND 广告数据 . . . 64
表 43. SCAN_RSP 事件 . . . 64
表 44. 扫描响应数据 . . . 64
表 45. BlueNRG 传感器感测演示服务和特征句柄 . . . 71
表 46. BlueNRG-MS 传感器感测演示服务和特征句柄 . . . 71
表 47. ACI：服务发现流程 API . . . 73
表 48. First evt_att_read_by_group_resp 事件数据 . . . 75

表 49. Second evt_att_read_by_group_resp 事件数据 75
表 50. Third evt_att_read_by_group_resp 事件数据 76
表 51. BlueNRG ACI：特征发现流程 API 77
表 52. 第一个 evt_att_read_by_type_resp 事件数据 79
表 53. 第二个 evt_att_read_by_type_resp 事件数据 80
表 54. 特征更新、读取、写入 API 80
表 55. 时隙算法的时序参数 89
表 56. 参考表 95
表 57. 缩略语列表 96
表 58. 文档版本历史 98
表 59. 中文文档版本历史 98

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。

传统蓝牙技术是按无线标准开发的，可以取代连接便携式和 / 或固定式电子设备的线缆，但是由于快速跳频、以连接为导向的行为和相对复杂的连接流程，不能使电池使用寿命达到最高水平。

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

低功耗蓝牙技术广泛应用于发送少量数据的传感器应用中。

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

1.1 BLE 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：参考文献*）。该版本的蓝牙标准支持两种无线技术系统：

- 基础速率
- 低功耗蓝牙

低功耗蓝牙技术在无需取得执照的工业、科学和医疗 （ISM）波段以 2.4 - 2.485 GHz 的频率工作，可在全球大部分国家 / 地区使用且无需取得执照。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
- 低数据速率
- 低功耗

具体来说，低功耗蓝牙技术是在功耗显著低于基础速率 / 增强数据率 / 高速 （BR/EDR/HS）设备的同时，以每次发送极小数据包为目的而设计的。

低功耗蓝牙技术设计用于两个备选实现工具的寻址：

- 发送方
- 接收方

发送方仅支持 BLE 标准。它适用于低功耗并使用纽扣电池供电的应用 （例如传感器）。

接收方支持 BR/EDR/HS 和 BLE 标准 （通常为移动设备或笔记本电脑）。

低功耗蓝牙栈有两个组成部分：

- 控制器
- 主机

控制器包含物理层和链路层。

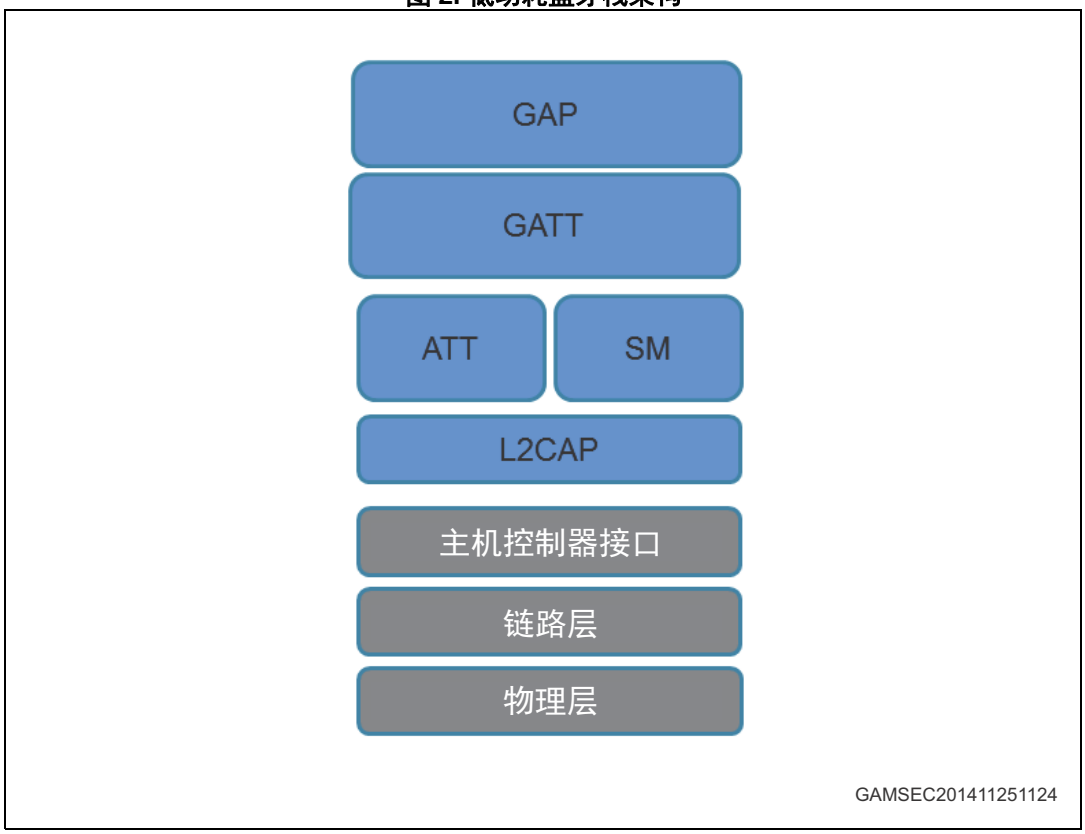
主机包括逻辑链路控制和适配协议 （L2CAP）、安全管理器 （SM）、属性协议 （ATT）、通用属性配置文件 （GATT）和通用访问配置文件 （GAP）。两个组成部分之间的接口被称为主机控制器接口 （HCI）。

此外，已发布的蓝牙规范 v4.1 具备以下新特性：

- 同时支持多个角色
- 支持同时广告并扫描
- 支持同时作为最多两个主设备的从设备
- 私有 V1.1
- 低占空比定向广告
- 连接参数请求流程
- LE Ping
- 32 位 UUID
- L2CAP 以连接为导向的行为

关于这些新特性的更多信息，请参考相关规范文档。

图 2. 低功耗蓝牙栈架构



1.2 物理层

物理层是一种 1 Mbps 自适应跳频高斯频移键控 （GFSK）无线电。它在无需执照的 2.4 GHz ISM 波段中以 2400-2483.5 MHz 的频率工作。许多其他标准也使用该波段，例如：IEEE 802.11， IEEE 802.15。

BLE 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 * 2 MHz，其中 k = 0..39 ；

有两种类型的通道：

1. 广告通道，使用三个固定的射频通道 （37、 38 和 39）用于：
   a) 广告通道数据包
   b) 用于发现 / 连接的数据包
   c) 广播 / 扫描
2. 数据物理通道使用其他 37 个射频通道进行已连接设备的双向通信。

表 1. BLE 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	广告通道
0	2404 MHz	数据通道
1	2406 MHz	数据通道
….	….	数据通道
10	2424 MHz	数据通道
38	2426 MHz	广告通道
11	2428 MHz	数据通道
12	2430 MHz	数据通道
….	….	数据通道
36	2478 MHz	数据通道
39	2480 MHz	广告通道

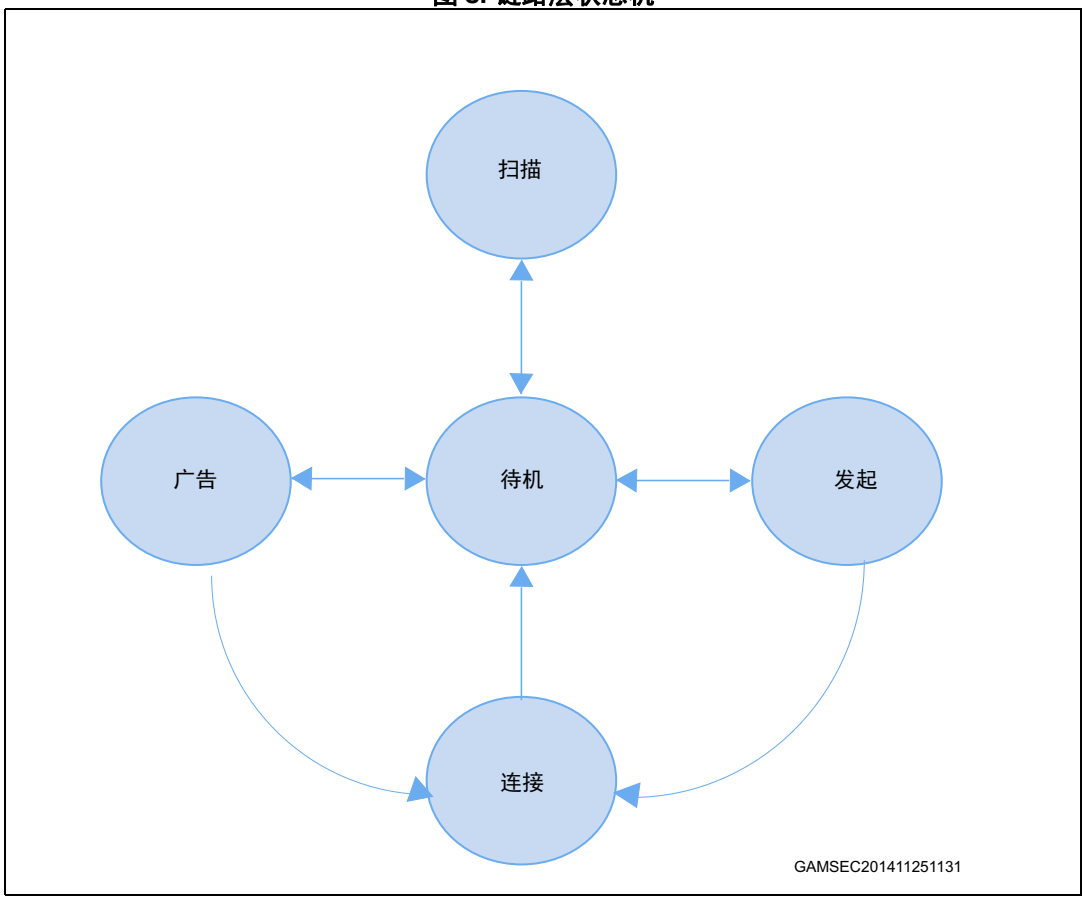
BLE 是一种自适应跳频 （AFH）技术，只能使用所有可用频率的一个子集，以避开其他非自适应技术使用的频率。通过使用特定的跳频算法确定下一个要使用的良好通道，可以从不良通道转移到已知良好通道。

1.3 链路层 （LL）

链路层 （LL）定义了两个设备使用无线技术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. 链路层状态机



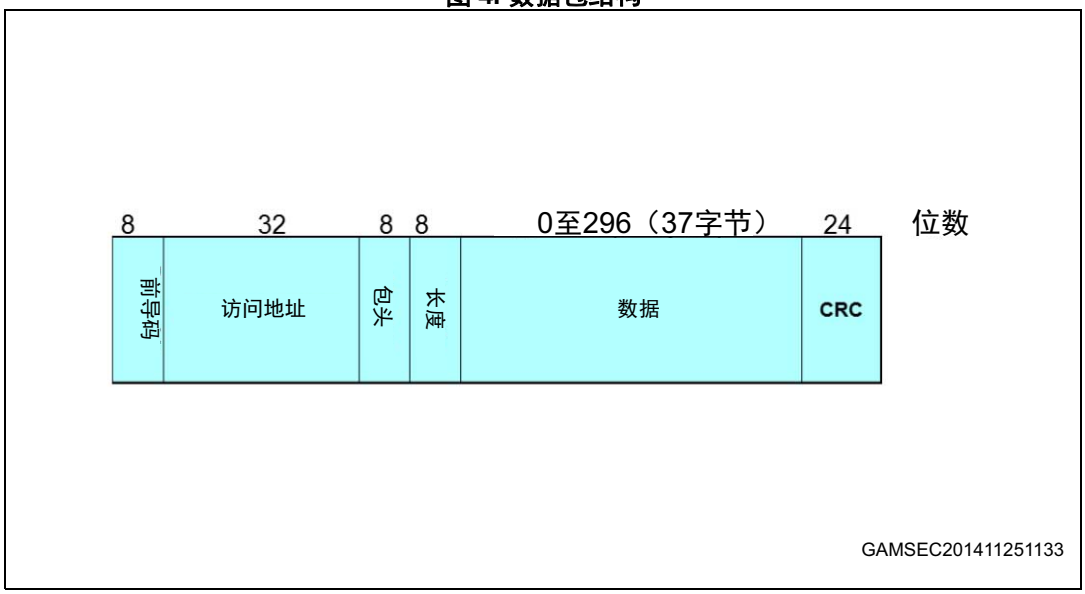
- 待机：设备不发送或接收数据包
- 广告：设备在广告通道中广播广告 （称为广告设备）。
- 扫描：设备寻找广告设备 （称为扫描设备）。
- 发起：设备发起与广告设备的连接
- 连接：发起设备为主设备角色：它与处于从设备角色的设备通信并定义发送时序
- 广告设备处于从设备角色：与处于主设备角色的一个设备通信

1.3.1 BLE 数据包

数据包是带标签的数据，由一个设备发送并由一个或多个其他设备接收。

BLE 数据包的结构如下。

图 4. 数据包结构



- 前导码：射频同步序列
- 访问地址：32 位，广告或数据访问地址 （用于标识物理层通道上的通信数据包）
- 包头：其内容取决于数据包类型 （广告或数据包）

  a) 广告数据包包头：

表 2. 广告数据头文件内容

广告数据包类型	保留	发送地址类型	接收地址类型
（4 位）	（2 位）	（1 位）	（1 位）

b) 广告数据包类型：

表 3. 广告数据包类型

数据包类型	说明	注释
ADV_IND	可连接非定向广告	由广告方在需要另一台设备与之连接时使用。设备可被扫描设备扫描到，或者在收到连接请求时作为从设备建立连接。
ADV_DIRECT_IND	可连接定向广告	由广告方在需要特定设备与之连接时使用。 ADV_DIRECT_IND 数据包仅包含广告方地址和发起方地址。
ADV_NONCONN_IND	不可连接非定向广告	由广告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告数据包，但不希望任何其他设备与之连接或扫描到它。
ADV_SCAN_IND	可扫描非定向广告	由希望允许扫描仪请求更多信息的广告方使用。设备不能连接，但是对于广告数据和扫描响应数据而言可发现。
SCAN_REQ	扫描请求	由处于扫描状态的设备用于向广告方请求更多信息。
SCAN_RSP	扫描响应	由广告设备用于向扫描设备提供更多信息。
CONNECT_REQ	连接请求	由发起设备发送给处于可连接 / 可发现模式的设备。

广告事件类型决定了允许的响应：

表 4. 广告事件类型和允许的响应

广告事件类型	允许的响应	
	SCAN_REQ	CONNECT_REQ
ADV_IND	是	是
ADV_DIRECT_IND	否	是
ADV_NONCONN_IND	否	否
ADV_SCAN_IND	是	否

数据包包头：

表 5. 数据包包头内容

链路层标识符	下一个序号	序号	更多数据	保留
（2 位）	（1 位）	（1 位）	（1 位）	（3 位）

下一个序号 （NESN）位用于执行数据包确认。它将发送设备预期将发送的下一个序号通知接收设备。将重新发送数据包，直至 NESN 不同于所发送数据包中的序号 （SN）值。

更多数据位用于向设备发送信号，表明在当前连接事件期间发送设备有更多数据准备发送。

关于广告和数据头内容及类型的详细描述，请参考*第 5 节：参考文献*中的蓝牙规范 v4.0[第 2 卷]。

- 长度：数据字段中的字节数

表 6. 数据包长度字段和有效值

	长度字段位数
广告数据包	6 位，有效值为 0 至 37 字节
数据包	5 位，有效值为 0 至 31 字节

- 数据或载荷：实际发送的数据 （广告数据、扫描响应数据、连接建立数据或连接期间发送的应用数据）。
- CRC （24 位）：用于保护数据，避免发生位错误。它通过包头、长度和数据字段计算得出。

1.3.2 广告状态

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

可通过停止广告使广告设备进入待机状态。

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

下面是广告数据包的示例，列出了设备实现的服务 UUID （一般可发现标记，发送功率 = 4dbm，服务数据 = 温度服务和 16 位服务 UUID）。

图 5. 具有 AD 类型标记的广告数据包

AD 类型标记字节包含以下标记位：

- 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 BLE 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，应在广告数据中包含 AD 类型标记 （不包括在扫描响应中）。

在启用广告之前，可设置下列广告参数：

- 广告间隔；
- 广告地址类型；
- 广告设备地址；
- 广告通道映射：应使用三个广告通道中的哪一个；
- 广告过滤策略：
  - 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 （默认的广告方过滤策略）
  - 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。如果设备处于广告状态且正在使用白名单过滤设备 （扫描请求或连接请求），必须禁用广告模式才能修改其白名单。

1.3.3 扫描状态

有两种类型的扫描：

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

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：它可以接受所有广告数据包 （默认策略）或仅接受来自白名单设备的广告数据包

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告报告事件中接收的广告数据包。该事件包含该广告数据包的广告方地址、广告方数据和接收信号强度指示 （RSSI）。可将 RSSI 与广告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。

1.3.4 连接状态

当待发送数据的复杂度超过广告数据允许的水平，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告设备发出的广告数据包时，它可以向广告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：

- 访问连接中使用的地址以识别物理链路上的通信
- 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 （发送窗口起点的偏移）
- 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 = 不良）
- 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 V4.0[第 6 卷] 第 2.3.3 节。

表 7 中总结了允许的时序范围。

表 7. 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	
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

在连接请求数据包结束 + 发送窗口偏移 + 1.25 ms 强制延时后，发送窗口开始。当发送窗口开始时，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，并在稍后重新尝试一个连接间隔。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 技术或栈）。

1.4 主机控制器接口 （HCI）

主机控制器接口 （HCI）层通过软件 API 或硬件接口 （例如， SPI、 UART 或 USB）在主机和控制器之间提供了一种通信方式。它来自标准蓝牙规范，提供额外的新指令用于低功耗特定的功能。

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。

1.6 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特定数据片段，即属性。公开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: 标识服务器上的属性的 16 位值，允许客户端引用读取或写入请求中的属性；
- 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特性 （客户端不能解决授权错误）。

属性示例

表 8. 属性示例

属性句柄	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）
  - 可以确认指示

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 9* 所示：

表 9. 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	客户端从服务器请求某些内容 （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
命令	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加密块链 - 消息验证码 （CCM）算法进行加密和验证，以及 128 位 AES 块加密。如果连接中使用了加密和验证，为数据通道 PDU 的载荷添加 4 字节的消息完整性检查 （MIC）。对 PDU 载荷和 MIC 字段应用加密。

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。此流程可以验证两个设备，并创建可作为受信任关系或 （单个）安全连接的基础的共用链路密钥。

配对流程分为三个阶段：

第 1 阶段：配对特征交换

- 两个互连的设备使用配对请求消息进行关于其输入 / 输出功能的通信。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。
- 有三种输入功能：
  a) 无输入；
  b) 能够选择是 / 否；
  c) 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

表 10. BLE 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

在第 1 阶段交换的信息将用于选择第 2 阶段使用的 STK 生成方法。

第 2 阶段：短期密钥 （STK）生成：

- 配对设备首先使用下列方法之一定义临时密钥 （TK）。
  a) 带外 （OOB）法：使用带外通信 （例如， NFC）达成 TK 协议 （如果设置了带外位，则选择该方法）；
  b) 输入密钥法：用户输入六位数字作为设备之间的 TK ；
  c) 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。

根据下表在输入密钥法和 Just Works 法之间进行选择：

表 11. 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：传输特定密钥分配

- 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：
  a) 长期密钥 （LTK）：用于生成链路层加密和验证使用的 128 位密钥；
  b) 连接签名解析密钥 （CSRK）：用于在 ATT 层执行数据签名；
  c) 身份解析密钥 （IRK）：用于在设备公共地址的基础上生成私有地址。

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。

BLE 支持的另一种安全机制是私有地址的使用。通过加密设备公共地址生成私有地址。该私有地址可通过具有相应加密密钥的可信设备进行解析。因此，设备可以使用私有地址进行更安全的通信并频繁更改地址 （仅具有相关 IRK 的设备能够识别该地址）。

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 位签名位于数据载荷之后。

签名算法还使用可提供重放攻击保护的计数器 （一种外部设备，可以只获取某些数据包然后发送，无需对数据包内容有任何理解：接收设备只检查数据包计数器并丢弃，因为它的帧计数器少于最近接收的良好数据包）。

1.8 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 BLE 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。

因此，不要求从 （外围）设备必须是 GATT 服务器以及主 （中央）设备必须是 GATT 客户端。

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）

1.8.1 特征属性类型

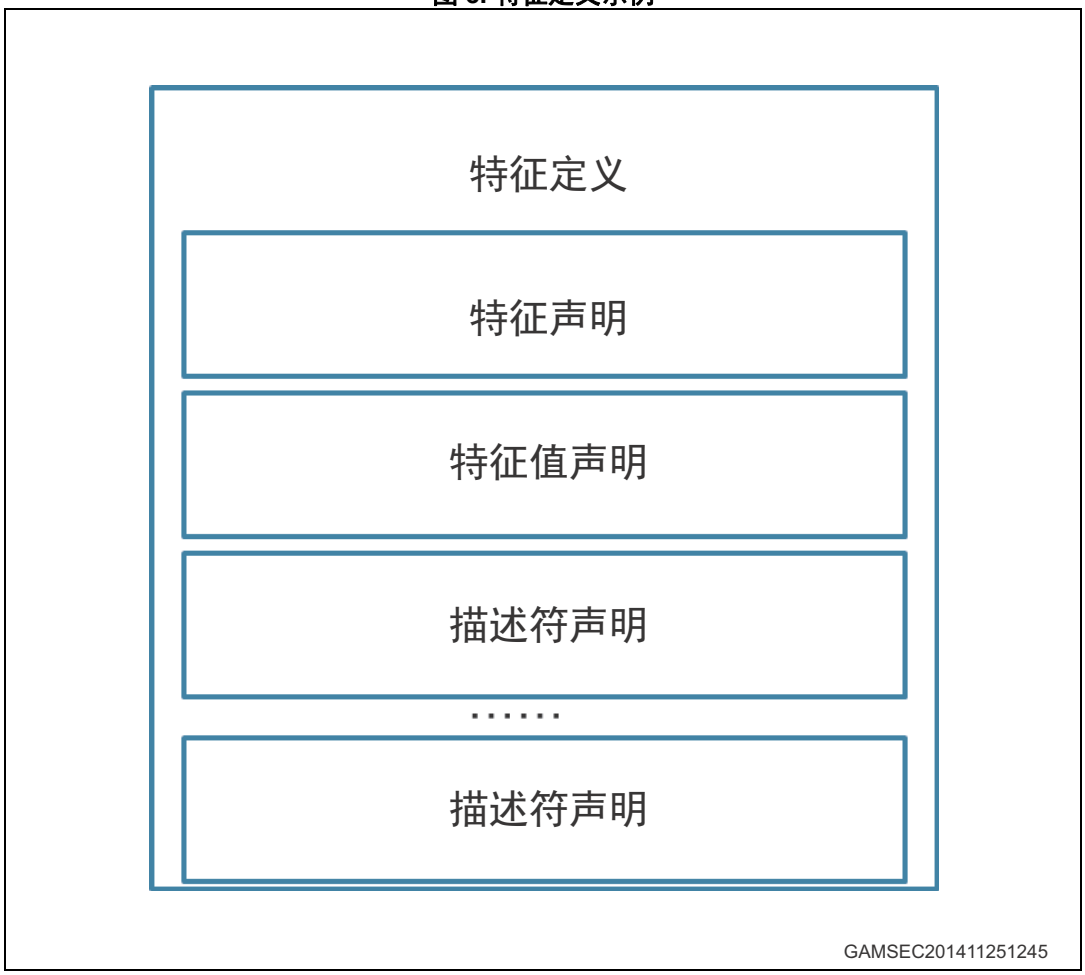
特征是一种包含一个值和任意数量描述符的属性类型，描述符描述的特征值使用户能够理解该特征。

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 6. 特征定义示例



特征声明是一种属性，其定义如下：

表 12. 特征声明

属性句柄	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 13. 特征值

属性句柄	属性类型	属性值	属性权限
0xNNNN	0xuuuu – 16 位或 128 位 （特征 UUID）	特征值	更高层配置文件或取决于实现

1.8.2 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，以便正确显示相关值 （例如，^{O}C 格式的温度测量值）；
6. 特征聚合格式：它可以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 v4.0。

1.8.3 服务属性类型

服务是特征的集合，它们共同为适用配置文件提供通用服务。例如，健康体温计服务包含文档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 14. 服务声明

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

服务应包含服务声明，并可能包含定义和特征定义。服务包含声明位于服务声明和服务器的任何其他属性之后。

表 15. 包含声明

属性句柄	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。

1.8.4 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 16*）
- 客户端发起的流程 （*表 17*）
- 服务器发起的流程 （*表 18*）

表 16. 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 17. 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。

表 17. 客户端发起的流程和相关响应事件 （续）

步骤	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 18. 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：参考文献*中的蓝牙规范 v4.0。

1.9 通用访问配置文件 （GAP）

蓝牙系统定义了所有蓝牙设备应用的基础配置文件，称为通用访问配置文件 （GAP）。此通用配置文件定义了蓝牙设备的基本要求。

下表中描述了四种 GAP 配置文件的角色：

表 19. GAP 角色(1)

角色	说明	发射器	接收器	典型示例
广播方	发送广告事件	M	O	发送温度值的温度传感器
监听方	接收广告事件	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。 处于可连接广告模式。 支持所有 LL 控制流程，可选择加密或不加密。	M	M	观看
中央设备	始终为主设备。 从不广告。 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. M = 强制； O = 可选

以 GAP 为背景，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操作。GAP 模式有四种类型：广播、可发现、可连接和可绑定类型。
- GAP 流程：配置设备，使之在特定的有限时间内执行单一操作。有四种类型的 GAP 流程：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 20. 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告通道和数据包广播数据 （不在 AD 类型标记上设置任何位）。	设备可使用监听流程检测广播数据	广播方

表 21. 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流程的设备发现它	外设
有限可发现模式	在 AD 类型标记上设置有限可发现位。	可持续约 30 秒。它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 22. 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 广告数据包	在广告时不能使用可连接广告数据包	外设
直接可连接模式	使用 ADV_DIRECT 广告数据包	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 广告数据包。	在设备希望成为可连接设备时使用。由于 ADV_IND 广告数据包可以包含 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 23. 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

表 24 中定义了以下 GAP 流程：

表 24. GAP 监听流程

步骤	说明	注释	角色
监听流程	它允许设备搜索广播方设备数据		监听方

表 25. GAP 发现流程

步骤	说明	注释	角色
有限可发现流程	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
通用可发现流程	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
名称发现流程	用于从可连接设备检索 “ 蓝牙设备名称 ” 的流程		中央设备

表 26. GAP 连接流程

步骤	说明	注释	角色
自动连接建立流程	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
通用连接建立流程	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
选择性连接建立流程	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
直接连接建立流程	使用一组连接间隔参数与特定设备建立连接。	由通用和选择性流程使用。	中央设备
连接参数更新流程	在连接期间更新使用的连接参数。		中央设备
终止流程	终止 GAP 流程		中央设备

表 27. GAP 绑定流程

步骤	说明	注释	角色
绑定流程	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 流程的详细描述，请参考蓝牙规范 v4.0。

1.10 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它们）。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

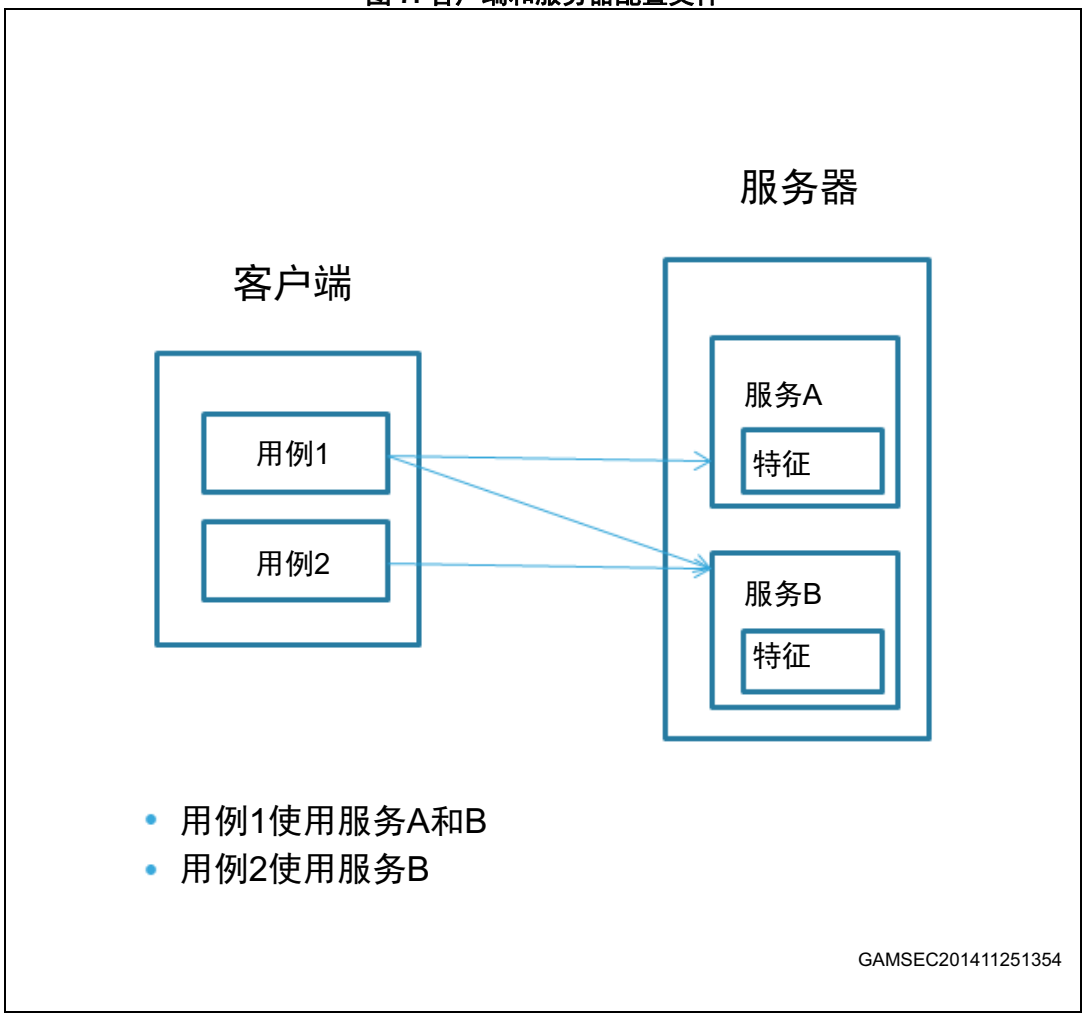
配置文件可以实现一种或多种服务 （以下网站规定并显示了这些服务：http://developer.bluetooth.org）。

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 7. 客户端和服务器配置文件



1.10.1 接近感测示例

本节将简单介绍接近感测的目标、工作原理和需要的服务：

目标

- 当距离非常远的设备接近时：
  – 导致报警

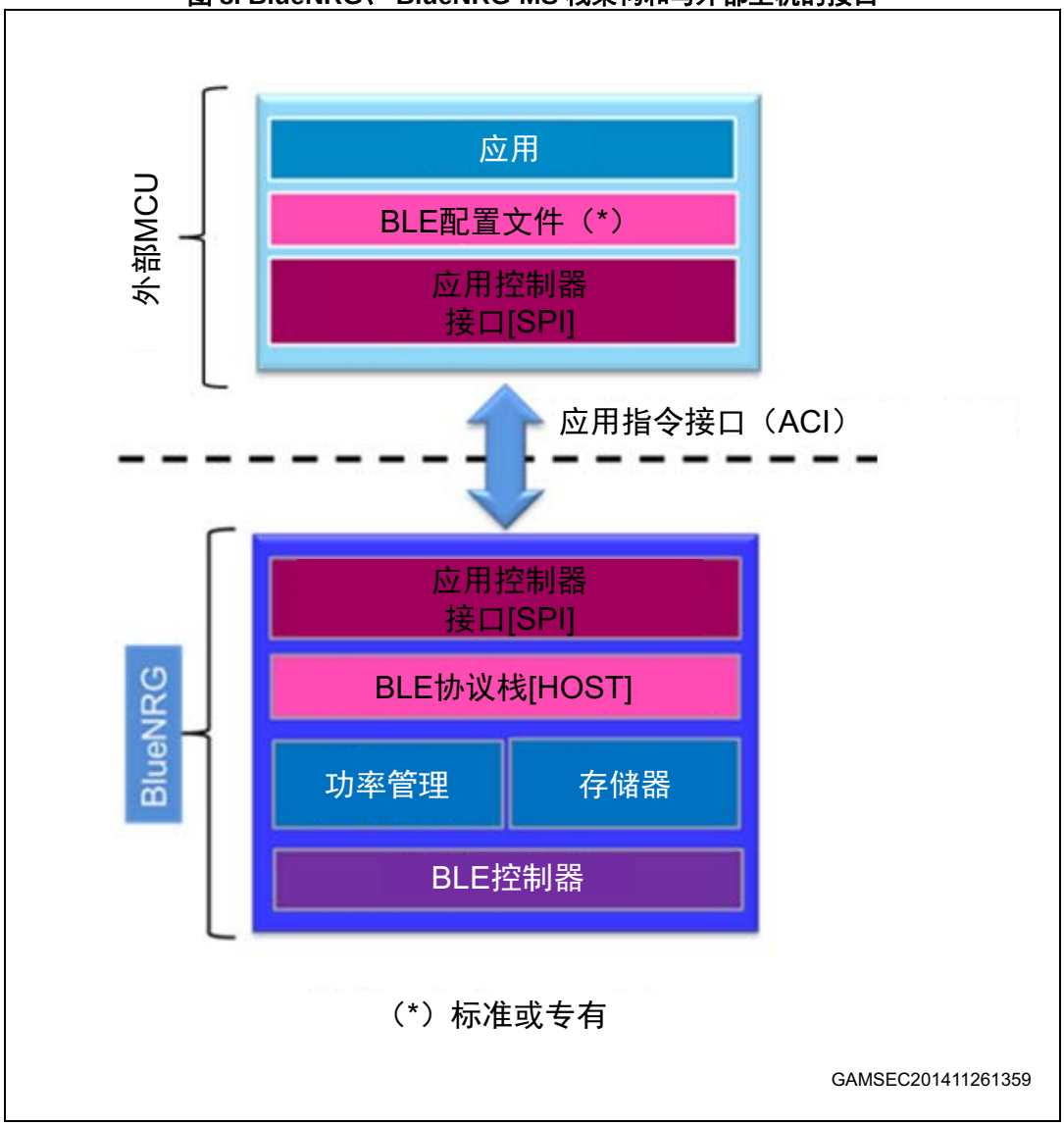
工作原理

- 如果设备断开
- 导致报警
- 链路丢失报警：《链路丢失》服务
  - 如果设备距离太远
  - 导致路径丢失报警：《即时报警》和 《发送功率》服务
- 《链路丢失》服务
  - 《报警级别》特征
  - 行为：链路丢失时，因被枚举导致报警
- 《即时报警》服务
  - 《报警级别》特征
  - 行为：写入时，因被枚举导致报警
- 《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 BlueNRG、 BlueNRG-MS 栈架构和 ACI

BlueNRG、 BlueNRG-MS 设备是网络协处理器，提供高层接口以控制其低功耗蓝牙功能。该接口被称为 ACI （应用指令接口）。

图 8. BlueNRG、 BlueNRG-MS 栈架构和与外部主机的接口



BlueNRG 和 BlueNRG-MS 设备上分别嵌入了蓝牙智能协议栈 v4.0 和 v4.1，因此外部微控制器上不需要 BLE 库，需要与 BlueNRG 或 BlueNRG-MS 设备 SPI 接口通信的配置文件和所有函数除外。 SPI 接口通信协议允许外部微控制器发送 ACI 指令，以便控制 BlueNRG 或 BlueNRG-MS 设备，并接收 BlueNRG 或 BlueNRG-MS 设备网络协处理器生成的 ACI 事件。

2.1 ACI 接口

ACI 指令利用并扩展在蓝牙规范 v4.0 和 v4.1 框架内定义的标准 HCI 数据格式。

ACI 接口支持以下指令：

- 蓝牙规范 （v4.0 和 v4.1）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 和 GAP）

BlueNRG、 BlueNRG-MS 套件软件包针对基于 STM32L1 外部微控制器的 BlueNRG、BlueNRG-MS 套件，提供了参考 ACI 接口框架 （请参考*第 5 节：参考文献*）。

ACI 接口框架包含用于发送 ACI 指令到 BlueNRG 和 BlueNRG-MS 网络处理器的代码。它还提供设备事件的定义。该框架允许以适当方式规定每个 ACI 指令的格式，并发送符合定义的 ACI SPI 通信协议的指令。

用户手册 UM1755“BlueNRG 蓝牙 LE 栈应用指令接口 （ACI） ” 和 UM1865“BlueNRG-MS 蓝牙 LE 栈应用指令接口 （ACI） ”（可在 ST BlueNRG 网页上找到）中描述了 ACI SPI 通信协议。这些用户手册还提供所有相关设备的 ACI 指令格式、名称参数、返回值和生成事件的完整描述。

以下头文件定义了 ACI 框架接口：

表 28. ACI 接口

文件	说明	位置	注释
hci.h	HCI 库函数原型和错误代码定义。	Middlewares\ST\STM32_BlueNRG\SimpleBlueNRG_HCI\includes	将包含在用户主应用中
hci_const.h	包含 HCI 层的常量和函数。参见蓝牙核心规范 v4.0 第 2 卷 E 部分。	“”	
bluenrg_gatt_server.h	GATT 服务器定义的头文件	“”	将包含在用户主应用中
sm.h	BlueNRG 安全管理器的头文件	“”	将包含在用户主应用中
bluenrg_gap.h	BlueNRG GAP 层的头文件	“”	将包含在用户主应用中
bluenrg_aci.h	包含 BlueNRG FW 栈的指令和事件的头文件	“”	将包含在用户主应用中
bluenrg_aci_const.h	包含 BlueNRG FW 栈的 ACI 定义的头文件	“”	包含在 bluenrg_aci.h 中
bluenrg_hal_aci.h	包含 BlueNRG FW 栈的 HCI 指令的头文件	“”	包含在 bluenrg_aci.h 中
bluenrg_l2cap_aci.h	包含 BlueNRG FW 栈的 L2CAP 指令的头文件	“”	包含在 bluenrg_aci.h 中
bluenrg_gatt_aci.h	包含 BlueNRG FW 栈的 GATT 指令的头文件	“”	包含在 bluenrg_aci.h 中

表 28. ACI 接口 （续）

文件	说明	位置	注释
bluenrg_gap_aci.h	包含 BlueNRG FW 栈的 GAP 指令的头文件	“”	包含在 bluenrg_aci.h 中
bluenrg_updater_aci.h	包含 BlueNRG FW 栈的更新器指令的头文件	“”	包含在 bluenrg_aci.h 中

2.2 ACI 接口资源

为了通过 ACI 接口框架与 BlueNRG 或 BlueNRG-MS 网络处理器通信，外部微控制器只需要下列主要资源：

1. SPI 接口
2. 写入 SPI 或从 SPI 读取的取决于平台的代码
3. 用于处理 SPI 超时的定时器

通过文件 SDK_EVAL_SPI_Driver.[ch] 和 hal.[ch] 定义的函数处理 BlueNRG、BlueNRG-MS SPI 接口。这些 API 使外部微控制器能够访问 BlueNRG 或 BlueNRG-MS 设备。 BlueNRG、BlueNRG-MS 设备在有需要读取的数据时使用 SPI IRQ 引脚通知外部微控制器 （SPI 主设备）：这是通过位于文件 stm32l1xx_it.c 的 SPI_IRQ_IRQHandler() 处理器 （标准库框架）和文件 bluenrg_interface.c 的 HAL_GPIO_EXTI_Callback() （Cube 库框架）中的 HCI_Isr() 来处理的。根据 BlueNRG、 BlueNRG-MS SPI 中断线的选定平台 GPIO 线， SPI IRQ 被关联到合适的 EXTI irq 处理器。

BlueNRG、 BlueNRG-MS 套件平台针对 STM32L1xx 微控制器，并使用相关库以访问外设。有两种框架可供使用：

1. STM32L1xx 标准库，位于 platform\STM32L1XX\Libraries\STM32L1xx_StdPeriph_Driver 文件夹；
2. STM32L1xx Cube 库，位于 Drivers\STM32L1xx_HAL_Driver 文件夹

表 29. ACI 接口资源文件

文件	说明	位置	注释
SDK_EVAL_SPI_Driver.[ch]	处理与 BlueNRG、BlueNRG-MS 设备的 SPI 通信的主要 API	platform\STM32L1XX\Libraries\SDK_Eval_STM32L\src （对于 STM32L1 标准库框架）；Drivers\BSP\STM32L1xx_BlueNRG （对于 STM32L1 Cube 库框架）。	这些 API 被映射到处理 SPI 外设的特定微控制器低层驱动
hal.[ch]	处理与 BlueNRG、BlueNRG-MS 设备的通信的其他 API	platform\STM32L1XX （仅限于 STM32L1 标准库框架）	
clock.[ch]	SPI 定时器 API	platform\STM32L1XX （对于 STM32L1 标准库框架）；在特定用户应用文件夹中定义 （对于 STM32L1 Cube 库框架）。	提供处理 SPI 超时的底层 API
stm32l1xx_it.c （标准库）， stm32xx_it.c （Cube 库）	主要中断服务程序	在特定用户应用文件夹中定义	

在使用另一个外部微控制器时，应对这些文件进行移植 / 调整，以便寻址 ACI SPI 通信。

为了正确设置 ACI SPI 接口，仅要求用户在 main() 函数的初始化时执行以下步骤：

1. 调用以下 API 初始化 SPI 接口：
   SdkEvalSpiInit(SPI_MODE_EXTI);
2. 调用以下 API 复位 BlueNRG 模块：
   BlueNRG_RST();

还要求用户将 HCI_Isr() 放入文件 stm32l1xx_it.c 的 SPI_IRQ_IRQHandler() 处理器 （标准库框架）和文件 bluenrg_interface.c 的 HAL_GPIO_EXTI_Callback() （Cube 库框架）中。这使得 BlueNRG、 BlueNRG-MS 设备能够在有数据需要读取时，使用 SPI IRQ 引脚通知外部微控制器 （SPI 主设备）。

2.3 其他平台资源文件

软件框架还提供其他文件，用于处理一些取决于平台的资源，例如 I/O 通信通道 （USB 或 UART）、按钮、 LED 和 EEPROM。

表 30. 软件框架平台驱动

文件	说明	位置	注释
SDK_EVAL_Io.[ch]	处理 I/O 通信的主要 API （USB 虚拟 COM 或 UART）	platform\STM32L1XX\Libraries\SDK_Eval_STM32L （对于 STM32L1 标准库框架）； Drivers\BSP\STM32L1xx_STEVAL_IDB00xV1 （对于 STM32L1 Cube 库框架）。	这些 API 被映射到处理 USB 虚拟 COM 或 UART 的特定微控制器驱动。
SDK_EVAL_Buttons.[ch]	处理平台按钮的 API	platform\STM32L1XX\Libraries\SDK_Eval_STM32L （对于 STM32L1 标准库框架）； Drivers\BSP\STM32L1xx_STEVAL_IDB00xV1 （对于 STM32L1 Cube 库框架）。	这些 API 被映射到处理 GPIO 的特定微控制器驱动。
SDK_EVAL_Leds.[ch]	处理平台 LED 的 API	platform\STM32L1XX\Libraries\SDK_Eval_STM32L （对于 STM32L1 标准库框架）； Drivers\BSP\STM32L1xx_STEVAL_IDB00xV1 （对于 STM32L1 Cube 库框架）。	这些 API 被映射到处理 GPIO 的特定微控制器驱动。
SDK_EVAL_Eeprom.[ch]	处理 EEPROM 的 API	platform\STM32L1XX\Libraries\SDK_Eval_STM32L （对于 STM32L1 标准库框架）； Drivers\BSP\STM32L1xx_STEVAL_IDB00xV1 （对于 STM32L1 Cube 库框架）。	BlueNRG、 BlueNRG-MS 套件提供了外部 EEPROM，用于保存平台生产测试结果。

应对这些文件进行移植 / 调整，以便寻址另一个外部微控制器。

2.3.1 平台配置

为了轻松支持 BlueNRG、 BlueNRG-MS 套件平台，设计了 BlueNRG 软件框架，用于在运行时识别此类平台。仅要求用户在 main() 函数的初始化时调用 SdkEvalIdentification() API。

还可以在编译时，通过对 EWARM 工作区预处理器选项添加以下定义之一来支持 BlueNRG、 BlueNRG-MS 套件平台：

USER_DEFINED_PLATFORM=STEVAL_IDB002V1 （适用于 BlueNRG、 BlueNRG-MS 开发平台）。

USER_DEFINED_PLATFORM=STEVAL_IDB003V1 （适用于 BlueNRG、 BlueNRG-MS USB 电子狗）。

下列定义值允许在编译时选择 platform\STM32L1XX\Libraries\SDK_Eval_STM32L\inc 文件夹中提供的特定平台头文件 （对于 STM32L1xx 标准库）：

```
#if USER_DEFINED_PLATFORM == STEVAL_IDB002V1
#include "USER_Platform_Configuration_STEVAL_IDB002V1.h"
#elif USER_DEFINED_PLATFORM == STEVAL_IDB003V1
#include "USER_Platform_Configuration_STEVAL_IDB003V1.h"
#endif
```

在编译时，只需执行下列步骤即可支持用户平台：

1. 以特定的用户平台配置创建文件 “USER_Platform_Configuration.h”：
   使用 STM32L1xx 标准库时， USER_Platform_Configuration_STEVAL_IDB002V1.h 或 USER_Platform_Configuration_STEVAL_IDB003V1.h 可用作参考 （有待根据可用用户平台资源进行扩展）。
2. 将 “USER_Platform_Configuration.h” 放入 STM32L\platform\STM32L1XX\Libraries\SDK_Eval_STM32L\inc 文件夹 （对于 STM32L1xx 标准库）。
3. 在选择的 EWARM 工作区预处理器选项中，添加此定义：
   USER_DEFINED_PLATFORM=USER_EVAL_PLATFORM。

如果在编译时没有定义用户平台，则通过相关预处理器选项，自动将 USER_DEFINED_PLATFORM 设置为 STEVAL_IDB00xV1。这样可以包含文件 USER_Platform_Configuration_auto.h，该文件中包含从 SdkEvalIdentification() 函数执行的运行时间自动配置流程中使用的 BlueNRG、 BlueNRG-MS 套件平台定义值。用户不得修改此头文件。

注： *对于 STM32L1xx Cube，也可以使用类似方法，例如，引用 Drivers\BSP\STM32L1xx_BlueNRG 文件夹中的文件 USER_Platform_Configuration_bluenrg.h 和 Drivers\BSP\STM32L1xx_Nucleo 文件夹中的文件 USER_Platform_Configuration.h （这些文件用于寻址 STM32L NUCLEO-L152RE + X-NUCLEO_IDB04A1 BlueNRG 平台）。*

2.4 如何将 ACI SPI 接口框架移植到选定微控制器

BlueNRG、 BlueNRG-MS 设备是提供低功耗蓝牙功能的网络协处理器。为了使用其功能，可通过实现之前描述的 ACI SPI 接口框架使用外部微控制器。 BlueNRG、 BlueNRG-MS 开发套件软件包提供了针对该 ACI SPI 接口的参考框架。通过执行以下步骤，可将该框架移植到另一个外部微控制器：

1. 以特定的用户平台 SPI 配置定义特定的 “USER_Platform_Configuration.h” （请参考 *第 2.3.1 节：平台配置*）。
2. 在选择的用户应用预处理器选项中，添加此定义：USER_DEFINED_PLATFORM=USER_EVAL_PLATFORM。
3. 用特定的微控制器底层驱动替换 platform\STM32L1XX\Libraries\STM32L1xx_StdPeriph_Driver 文件夹中的 STM32L1xx 库 （对于 STM32L1xx 标准库）
4. 根据选择的微控制器替换 CMSIS Cortex-M3 文件和启动文件 （文件 startup_stm32l1xx_md.s）
5. 根据选择的微控制器重新调整 / 移植处理 STM32L1xx 系统时钟配置的文件 system_stm32l1xx.c
6. 调整 / 移植这一节中描述的文件，以引用所选外部微控制器底层驱动。

根据选择的微控制器重新调整 / 移植 stm32l1xx_it.c （标准库）、 stm32xx_it.c （Cube 库）文件 （确保在处理 IRQ 线上外部 IRQ 中断的 SPI irq API 中调用了 HCI_Isr()）。

在将 ACI SPI 接口框架移植到选定微控制器后，用户可通过执行应用笔记 AN4494 “ 启动 BlueNRG、 BlueNRG-MS” 的 “SPI 接口 ” 一节中描述的基本测试 （请参考 ST BlueNRG 和 BlueNRG-MS 网页），确认可以从外部微控制器进行 SPI 访问。

注： 对 STM32L1xx Cube 可使用类似方法，使用时请参考 Drivers 文件夹中的特定文件（CMSIS 和 STM32L1xx_HAL_Driver）。

3 使用 BlueNRG、 BlueNRG-MS ACI API 设计应用

本节提供关于如何在选定微控制器上设计和实现低功耗蓝牙应用的信息和代码示例。

用户在选定 MCU 上实现 BLE 主机应用时，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. BlueNRG、 BlueNRG-MS 事件和事件回调设置
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

STM32L1xx 微控制器是以下章节中描述的编程指南使用的参考外部微控制器，因为可用的 BlueNRG、 BlueNRG-MS 套件平台正是基于此类微控制器。

注： *在后面的章节中，将使用一些用户应用“ 定义 ”，以便轻松识别设备低功耗蓝牙的角色 （中央设备、外设、客户端和服务器）。*

此外，对于提供的每段虚拟代码，任何对 BlueNRG 设备的引用也同时适用于 BlueNRG-MS 设备。必要时，会使用 #ifdef BLUENRG_MS 着重标明具体的区别。

表 31. BLE 设备角色的用户应用定义

定义	说明
GAP_CENTRAL	GAP 中央设备角色
GAP_PERIPHERAL	GAP 外设角色
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

为了正确配置选定的外部微控制器和与 BlueNRG 或 BlueNRG-MS 设备的 SPI 通信，需执行下列主要步骤。

1. 初始化 STM32L 设备 （时钟配置、 GPIO 等）
2. 配置选定 BlueNRG 平台
3. 初始化用于 I/O 通信的串行通信通道 （调试和实用工具信息）
4. 初始化就绪表头并释放 HCI 数据包队列
5. 初始化 SPI 接口以允许外部微控制器正常访问 BlueNRG 功能
6. 复位 BlueNRG、 BlueNRG-MS 网络协处理器
7. 配置 BlueNRG、 BlueNRG-MS 公共地址 （如果使用公共地址）
8. 初始化 BLE NRG GATT 层
9. 根据选择的设备角色初始化 BLE NRG GAP 层
10. 设置合适的安全 I/O 能力和验证要求 （如果使用了 BLE NRG 安全）
11. 如果设备是 GATT 服务器，定义需要的服务和特征
12. 添加调用 HCI_Process() API 的 while(1) 循环和处理用户操作 / 事件的特定用户应用函数 （广告、连接、服务和特征发现、通知及相关事件）。

以下虚拟代码示例描述了需要的初始化步骤：

```
int main(void)
{
  int ret;
/* 设备初始化 */
Init_Device();
/* 识别 BlueNRG、 BlueNRG-MS 平台 */
SdkEvalIdentification();
/* 配置 I/O 通信通道：
    它需要处理用户接收到的数据的 void IO_Receive_Data(uint8_t * rx_data, uint16_t data_size) 函数
*/
  SdkEval_IO_Config(processInputData);
  /* 初始化就绪表头并释放 HCI 数据包队列 */
  HCI_Init();
  /* 初始化 SPI 接口 */
  SdkEvalSpiInit(SPI_MODE_EXTI);
  /* 复位 BlueNRG 网络协处理器 */
  BlueNRG_RST();
  /* 将 BlueNRG 地址配置为公共地址 （使用其公共地址）*/
  {
    uint8_t bdaddr[] = {0xaa, 0x00, 0x00, 0xE1, 0x80, 0x02};
    ret = aci_hal_write_config_data(CONFIG_DATA_PUBADDR_OFFSET,
CONFIG_DATA_PUBADDR_LEN,bdaddr);
    if(ret)PRINTF("Setting BD_ADDR failed.\n");
```

```
 }
 /*  初始化 BlueNRG GATT 层 */
 ret = aci_gatt_init();
 if(ret) PRINTF("GATT_Init failed.\n");
 /*  将 BlueNRG GAP 层初始化为外设或中央设备 */
 {
   uint16_t service_handle, dev_name_char_handle, appearance_char_handle;
#if GAP_PERIPHERAL
        uint8_t role = GAP_PERIPHERAL_ROLE;
#else
        uint8_t role = GAP_CENTRAL_ROLE;
#endif
#if BLUENRG_MS
        ret = aci_gap_init(role, 0, 0x07, &service_handle, &dev_name_char_handle,
&appearance_char_handle);
#else
        ret = aci_gap_init(role, &service_handle, &dev_name_char_handle,
&appearance_char_handle);
#endif
if(ret) PRINTF("GAP_Init failed.\n");
 }
 /**** 如果使用安全功能，设置 I/O 能力和验证要求：请参考第节 */
……
 #if  GATT_SERVER
 /* 定义服务和特征的用户应用函数：请参考 “ 服务和特征配置 ” 一节 */
 ret = Add_Server_Services_Characteristics();
 if(ret == BLE_STATUS_SUCCESS)
    PRINTF(" 服务和特征添加成功。 \n");
 else
    PRINTF(" 添加服务和特征时发生错误。 \n");
#endif

 /* 应用程序主循环 */
 while(1)
 {
    /*  处理任何挂起的 HCI 事件读取 */
    HCI_Process()
   /* 处理用户操作和事件的特定用户应用函数 （广告、连接、服务和特征发现及通知）
   */
   User_Process();
 }
```

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

注： *1. Init_Device() 根据选择的框架 （标准或 Cube 库）初始化 STM32L1xx 微控制器。*

2. User_Process() 只是一个取决于应用的函数。在后面的章节中，根据最常用的 BLE 功能描述了一些参考性的特定操作 / 事件。用户开发人员可以对它们进行调整 / 修改 / 替换。

3. 在执行 GATT_Init() 和 GAP_Init() API 时， BlueNRG 和 BlueNRG-MS 栈总是添加两个标准服务：具有服务更改特征的属性配置文件服务 （0x1801）和具有设备名称和外观特征的 GAP 服务 （0x1800）。

4. 为标准 GAP 服务预留的最后一个属性句柄是 0x000F （BlueNRG 栈）和 0x000B （BlueNRG-MS 栈）。

表 32. BlueNRG GATT、 GAP 默认服务

默认服务	开始句柄	结束句柄	服务 UUID
属性配置文件服务	0x0001	0x0004	0x1801
通用访问配置文件 （GAP）服务	0x0005	0x000F	0x1800

表 33. BlueNRG GATT、 GAP 默认特征

默认服务	特征	属性句柄	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						
	服务更改	0x0002	指示	0x0003	0x2A05	4
通用访问配置文件（GAP）服务						
	设备名称	0x0006	读取 | 写入，无响应 | 写入 | 验证签名写入	0x0007	0x2A00	7
	外观	0x0008	读取 | 写入，无响应 | 写入 | 验证签名写入	0x0009	0x2A01	2
	外设私有标记	0x000A	读取 | 写入，无响应 | 写入	0x000B	0x2A02	1
	重新连接地址	0x000C	读取 | 写入，无响应 | 写入	0x000D	0x2A03	6
	外设首选的连接参数	0x000E	读取 | 写入	0x000F	0x2A04	8

表 34. BlueNRG-MS GATT、 GAP 默认服务

默认服务	开始句柄	结束句柄	服务 UUID
属性配置文件服务	0x0001	0x0004	0x1801
通用访问配置文件（GAP）服务	0x0005	0x000B	0x1800

表 35. BlueNRG-MS GATT、 GAP 默认特征

默认服务	特征	属性句柄	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						
	服务更改	0x0002	指示	0x0003	0x2A05	4
通用访问配置文件（GAP）服务						
	设备名称	0x0006	读取 | 写入，无响应 | 写入 | 验证签名写入	0x0007	0x2A00	7
	外观	0x0008	读取 | 写入，无响应 | 写入 | 验证签名写入	0x0009	0x2A01	2
	外设首选的连接参数	0x000A	读取 | 写入	0x000B	0x2A04	8

4. GAP_Init() 角色参数值如下：

表 36. GAP_Init() 角色参数值

器件	角色参数值	注释
BlueNRG	0x01：外设 0x03：中央设备	BlueNRG 设备不支持广播器、监听器
BlueNRG-MS	0x01：外设 0x02：广播方 0x04：中央设备 0x08：监听方	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）

此外，对于 BlueNRG-MS 栈， GAP_Init() API 提供了两个新参数：

– enable_Privacy：0x00，用于禁用私有； 0x01，用于启用私有；

– device_name_char_len：可以指示设备名称特征的长度。

关于该 API 和相关参数的完整描述，请参考*第 5 节：参考文献*中的 UM1755 和 UM1865 用户手册。

3.1.1 BLE 地址

BlueNRG 和 BlueNRG-MS 设备支持下列设备地址：

- 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 BLE 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）。

由于 MAC 地址因厂商而异， BlueNRG 和 BlueNRG-MS 设备没有有效的预分配 MAC 地址。公共地址必须通过外部处理器设置。

设置 MAC 地址的 ACI 指令是 ACI_HAL_WRITE_CONFIG_DATA （操作码：0xFC0C），其指令参数如下：

- 偏移：0x00 （0x00 标识 BTLE 公共地址，即 MAC 地址）
- 长度：0x06 （MAC 地址的长度）
- 值：0xaabbccddeeff （MAC 地址的 48 位数组）

在开始 BLE 操作之前 （BlueNRG 每次启动或复位后），指令 ACI_HAL_WRITE_CONFIG_DATA 应通过 uC 发送至 BlueNRG 和 BlueNRG-MS 设备。

以下虚拟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_PUBADDR_LEN, bdaddr);
if(ret)PRINTF("Setting address failed.\n")}
```

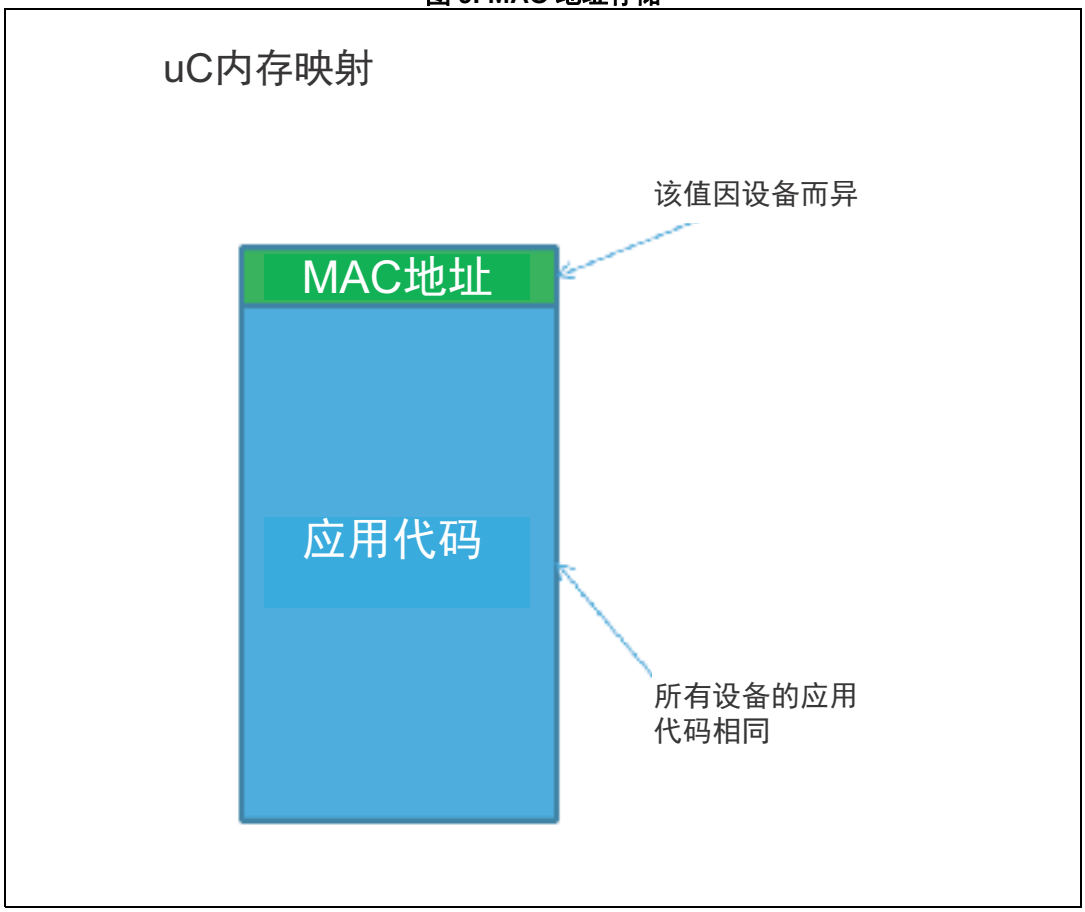
MAC 地址需存储在产品生产期间关联到产品的非易失性存储器上。

用户在写入其应用时，可以假设 MAC 地址位于微控制器的已知 Flash 位置。在生产期间，可通过 JTAG 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 Flash 的空闲 48 位区的已知位置。

当微控制器的应用需要访问 MAC 地址时，只需引用已知的 Flash 存储器位置。

图 9. MAC 地址存储



或者，可以将 MAC 地址存储在 BlueNRG 和 BlueNRG-MS 设备信息寄存器 （IFR）区的空闲区域，但这样做并无益处，原因如下：

- IFR 中 MAC 的编程需要如下几个 SPI 事务：
  - 使设备进入更新器模式 （1 个 SPI 事务）
  - 使用 MAC 地址进行设备 IFR 编程 （1 个 SPI 事务）
  - 使设备退出更新器模式 （1 个 SPI 事务）
- 在设备初始化 （每次启动或复位时）期间访问 MAC 地址需要如下几个 SPI 事务：
  - 使设备进入更新器模式 （1 个 SPI 事务）
  - 从设备 IFR 读取 MAC 地址 （1 个 SPI 事务）
  - 使设备退出更新器模式 （1 个 SPI 事务）

BLE 标准还可以使用用户定义且不遵循公共地址规则的 “ 随机 ” 地址。随机地址由设备自主处理，在每次复位时设置，但是也可能被外部处理器使用 hci_le_set_random_address()API 覆盖。

当启用了私有时，按照低功耗蓝牙规范使用私有地址。关于私有地址的更多信息，请参考 *第 1.7 节：安全管理器 （SM）*。

3.1.2 设置发送功率水平

在初始化阶段，用户还可以使用以下 API 选择发送功率水平：

aci_hal_set_tx_power_level(high or standard, power level)

下面是将无线发送功率设置为高功率和 -2 dBm 输出功率的虚拟代码示例：

```
ret = aci_hal_set_tx_power_level(1,4);
```

关于该 API 和相关参数的完整描述，请参考*第 5 节：参考文献*中的 UM1755 和 UM1865 用户手册。

3.2 BlueNRG、 BlueNRG-MS 事件和事件回调

每次有需要处理的 ACI 事件时， ACI 框架都将通过 HCI_Event_CB() 回调将该事件通知用户应用。在 hci.c 文件的 HCI_Process() 中调用 HCI_Event_CB() 回调。

因此，要求用户应用：

1. 在主应用程序中定义 void HCI_Event_CB(void *pckt) 函数 （pckt 是接收的 ACI 包的指针）
2. 根据具体的应用场景，用户必须标识要检测和处理的必要设备事件，以及发生这些事件后要执行的应用特定的操作。

在实现 BLE 应用时，最常见且使用最广泛的设备事件是与发现、连接、终止流程、服务和特征发现流程、 GATT 服务器上的属性修改事件和 GATT 客户端上的通知 / 指示事件相关的事件。

表 37. ACI：主要事件、子事件

事件 / 子事件	说明	主要事件	位置
EVT_DISCONN_COMPLETE	连接终止	NA	GAP 中央设备 / 外设
EVT_LE_CONN_COMPLETE	向构成连接的两个主机指示新的连接已建立	EVT_LE_META_EVENT	GAP 中央设备 / 外设
EVT_BLUE_GATT_ATTRIBUTE_MODIFIED	如果事件启用，由 GATT 服务器在客户端修改服务器上的任何属性时生成	EVT_VENDOR	GATT 服务器
EVT_BLUE_GATT_NOTIFICATION	由 GATT 客户端在服务器通知客户端上的任何属性时生成	EVT_VENDOR	GATT 客户端
EVT_BLUE_GATT_INDICATION	由 GATT 客户端在服务器指示客户端上的任何属性时生成	EVT_VENDOR	GATT 客户端
EVT_BLUE_GAP_PASS_KEY_REQUEST	当需要密码用于配对时，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_response () API 响应	EVT_VENDOR	GAP 中央设备 / 外设
EVT_BLUE_GAP_PAIRING_CMPLT	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	EVT_VENDOR	GAP 中央设备 / 外设
EVT_BLUE_GAP_BOND_LOST	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	EVT_VENDOR	GAP 外设
EVT_BLUE_ATT_READ_BY_GROUP_RESP	发送按组读取型响应，以回复接收到的按组读取型请求，包含已读取的属性的句柄和值	EVT_VENDOR	GATT 客户端
EVT_BLUE_ATT_READ_BY_TYPE_RESP	发送按类型读取的响应，以回复接收到的按类型读取的请求 请求并包含已读取属性的句柄和值。	EVT_VENDOR	GATT 客户端

表 37. ACI：主要事件、子事件 （续）

事件 / 子事件	说明	主要事件	位置
EVT_BLUE_GAP_DEVICE_FOUND （仅限于 BlueNRG 设备）	在上层启动 GAP 流程之一后，在扫描期间发现设备时 GAP 层向上层生成的事件。	EVT_VENDOR	GAP 中央设备
EVT_BLUE_GATT_PROCEDURE_COMPLETE	GATT 流程已完成	EVT_VENDOR	GATT 客户端
EVT_LE_ADVERTISING_REPORT （仅限于 BlueNRG-MS 设备）	在上层启动 GAP 流程之一后，在扫描期间发现设备时 GAP 层向上层生成的事件。	EVT_LE_META_EVENT	GAP 中央设备

关于 BLE 事件的详细描述和相关格式，请参考用户手册 UM1755 和本文档的*表 38：ACI：GAP 模式 API*。

下列虚拟代码提供了处理一些上述设备事件 （EVT_DISCONN_COMPLETE、EVT_LE_CONN_COMPLETE、 EVT_BLUE_GATT_ATTRIBUTE_MODIFIED 和 EVT_BLUE_GATT_NOTIFICATION）的 HCI_Event_CB() 回调的示例：

```
void HCI_Event_CB(void *pckt)
{
    hci_uart_pckt *hci_pckt = pckt;
    hci_event_pckt *event_pckt = (hci_event_pckt*)hci_pckt->data;
    if(hci_pckt->type != HCI_EVENT_PKT return;
```

```
    switch(event_pckt->evt){
    case EVT_DISCONN_COMPLETE: /* BlueNRG 断开事件 */
    {
        /* 根据应用场景为处理 BLE 断开事件添加用户代码
        */
.........
    }
break;
case EVT_LE_META_EVENT:
{
    /* 获取元事件数据 */
    evt_le_meta_event *evt = (void *)event_pckt->data;
    /* 分析特定的子事件 */
    switch(evt->subevent){
    case EVT_LE_CONN_COMPLETE:/* BlueNRG 连接事件 */
    {
        /* 连接完成事件：获取相关数据 */
        evt_le_connection_complete *cc = (void *)evt->data;
        /* 连接参数：
        cc->status：连接状态 （0x00：连接成功完成）；
        cc->handle：连接期间的通信要使用的连接句柄；
        cc->role：BLE 设备角色 （0x01：主设备； 0x02：从设备）；
        cc->peer_bdaddr_type：连接的设备地址类型 （0x00：公共； 0x01：随机）；
        cc->peer_bdaddr：连接的设备地址；
        cc->interval：连接间隔；
        cc->latency：连接延迟；
        cc->supervision_timeout：连接监控超时；
        cc->master_clock_accuracy：主时钟精度；
    */
        /* 根据应用场景为处理连接事件添加用户代码 */
        conn_handle = cc->handle;
.....
        } /* EVT_LE_CONN_COMPLETE */
        break;
    } /* switch(evt->subevent) */
} /* EVT_LE_META_EVENT */
break;
case EVT_VENDOR:
{
    /* 获取供应商事件数据 */
    evt_blue_aci *blue_evt = (void*)event_pckt->data;
```

```
    switch(blue_evt->ecode){
#if GATT_SERVER
    case EVT_BLUE_GATT_ATTRIBUTE_MODIFIED:
    {
    /* 获取属性修改事件数据 */
        evt_gatt_attr_modified *evt = (evt_gatt_attr_modified*)blue_evt->data;
        evt->conn_handle：修改了属性的连接句柄；
        evt->attr_handle：被修改的属性的句柄；
        evt->data_length：数据长度；
        evt->att_data：新值的指针 （长度为 data_length）。
        /* 根据应用场景为处理属性修改事件添加用户代码 */
....
    }/* EVT_BLUE_GATT_ATTRIBUTE_MODIFIED */
    break;
#endif /* GATT_SERVER */
#if  GATT_CLIENT
        case EVT_BLUE_GATT_NOTIFICATION:
{
    /* 获取属性通知事件数据 */
        evt_gatt_attr_notification *evt = (evt_gatt_attr_notification*)blue_evt->data;
        evt->conn_handle：通知了属性的连接句柄；
        evt->event_data_length：属性值 + 句柄的长度 （2 字节）；
        evt->attr_handle：属性句柄；
        evt->attr_value：属性值的指针 （长度为 event_data_length – 2）。
        /* 根据应用场景为处理属性通知事件添加用户代码 */
.....
    }/* EVT_BLUE_GATT_NOTIFICATION */
    break;
    break;
#endif /* GATT_CLIENT */
        }/* switch(blue_evt->ecode) */
    }/* EVT_VENDOR */
    break;
    }/* switch(evt->subevent)*/
}/* end HCI_Event_CB() */
```

3.3 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寻址的特定配置文件：

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 实现它们 （请参考蓝牙 SIG 网页：https://www.bluetooth.org/en-us/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 UIDS，并且必须由配置文件实现程序生成 （请参考 UUID 生成器网页：http://www.famkruithof.net/uuid/uuidgen）。

可使用以下指令添加服务：

- aci_gatt_add_serv (Service_UUID_Type, Service_UUID_16, Service_Type, Max_Attributes_Records, &ServHandle);

该指令返回服务句柄的指针 （ServHandle），用于标识用户应用中的服务。可使用该指令为该服务添加特征：

- aci_gatt_add_char (ServHandle, Char_UUID_Type, Char_UUID_16, Char_Value_Length, Char_Properties, Security_Permissions, GATT_Evt_Mask, Enc_Key_Size, Is_Variable, &CharHandle);

该指令返回特征句柄的指针 （Char_Handle），用于标识用户应用中的特征。

关于 aci_gatt_add_serv() 和 aci_gatt_add_char() 函数参数的详细描述，请参考用户手册 UM1755 和 UM1865。

以下虚拟代码示例描述了在专有非标准配置文件中添加服务和两个关联的特征所要执行的步骤。

```
tBleStatus Add_Server_Services_Characteristics(void)
{
    tBleStatus ret;
    /*
    以下 128 位 UUID 由随机 UUID 生成器生成：
    D973F2E0-B19E-11E2-9E96-0800200C9A66 --> 服务 128 位 UUID
    D973F2E1-B19E-11E2-9E96-0800200C9A66 --> Characteristic_1 128bits UUID
    D973F2E2-B19E-11E2-9E96-0800200C9A66 --> Characteristic_2 128bits UUID
    */
/* 服务 128 位 UUID */
const uint8_t service_uuid[16] =
{0x66,0x9a,0x0c,0x20,0x00,0x08,0x96,0x9e,0xe2,0x11,0x9e,0xb1,0xe0,0xf2,0x73,0xd9};
/* Characteristic_1 128 位 UUID */
const uint8_t charUuid_1[16] =
{0x66,0x9a,0x0c,0x20,0x00,0x08,0x96,0x9e,0xe2,0x11,0x9e,0xb1,0xe1,0xf2,0x73,0xd9};
/* Characteristic_2 128 位 UUID */
```

```
const uint8_t charUuid_2[16] =
{0x66,0x9a,0x0c,0x20,0x00,0x08,0x96,0x9e,0xe2,0x11,0x9e,0xb1,0xe2,0xf2,0x73,0xd9};
/* 将具有 service_uuid 128 位 UUID 的服务添加到 GATT 服务器数据库。返回服务句柄 ServHandle。
*/
ret = aci_gatt_add_serv(UUID_TYPE_128, service_uuid, PRIMARY_SERVICE, 7, &ServHandle);
if (ret != BLE_STATUS_SUCCESS) PRINTF("Failure.\n")}

/* 将具有 charUuid_1 128 位 UUID 的特征添加到服务 ServHandle。
该特征具有最大长度为 20 位的特征值、通知属性 （CHAR_PROP_NOTIFY）、无安全许可
（ATTR_PERMISSION_NONE）、无 GATT 事件掩码 (0)、加密密钥长度 16 以及长度可变的特征 (1)。
返回特征句柄 （CharHandle_1）。
*/
ret =  aci_gatt_add_char(ServHandle, UUID_TYPE_128, charUuid_1, 20, CHAR_PROP_NOTIFY,
ATTR_PERMISSION_NONE, 0,16, 1, &CharHandle_1);
if (ret != BLE_STATUS_SUCCESS) PRINTF("Failure.\n")}

/* 将具有 charUuid_2 128 位 UUID 的特征添加到服务 ServHandle。该特征具有最大长度为 20 位的特
征值、读取 / 写入 / 写入且无响应属性、无安全许可 （ATTR_PERMISSION_NONE）、在将 GATT 事
件掩码作为属性写入时通知应用 （GATT_NOTIFY_ATTRIBUTE_WRITE）、加密密钥长度 16 以及长
度可变的特征 (1)。返回特征句柄 （CharHandle_2）。
*/
ret =  aci_gatt_add_char(ServHandle, UUID_TYPE_128, charUuid_2, 20,
CHAR_PROP_WRITE|CHAR_PROP_WRITE_WITHOUT_RESP, ATTR_PERMISSION_NONE,
GATT_NOTIFY_ATTRIBUTE_WRITE,16, 1, &CharHandle_2);
if (ret != BLE_STATUS_SUCCESS) PRINTF("Failure.\n")}
    return ret ;
}/* end Add_Server_Services_Characteristics() */
```

3.4 创建连接：可发现和可连接 API

为了在 BlueNRG GAP 中央 （主）设备和 BlueNRG GAP 外围 （从）设备之间建立连接，可以按照*表 38：ACI：GAP 模式 API*、*表 39：ACI：发现流程 API* 和*表 40：ACI：连接流程 API* 以及用户手册 UM1755 和 UM1865 （*第 5 节：参考文献*）中描述的相关 ACI API 使用 GAP 可发现 / 可连接模式和流程。

GAP 外设可发现和可连接模式 API

可按照以下 API 的描述使用不同类型的可发现和可连接模式：

表 38. ACI：GAP 模式 API

API	支持的广告事件类型	说明
aci_gap_set_discoverable()	0x00：可连接非定向广告（默认）	将设备设置为一般可发现模式。设备处于可发现模式，直至主机发布 aci_gap_set_non_discoverable() API.
	0x02：可扫描非定向广告	
	0x03：不可连接非定向广告	
aci_gap_set_limited_discoverable()	0x00：可连接非定向广告（默认）；	将设备设置为有限可发现模式。设备处于可发现模式的最长时间为 TGAP (lim_adv_timeout) = 180 秒。随时可通过调用 aci_gap_set_non_discoverable() API 禁用广告
	0x02：可扫描非定向广告；	
	0x03：不可连接非定向广告。	
aci_gap_set_non_discoverable()	NA	将设备设置为不可发现模式。该指令禁用 LL 广告并将设备设置为待机状态。
aci_gap_set_direct_connectable()	NA	将设备设置为定向可连接模式。设备的定向可连接模式仅持续 1.28 秒。如果在此期间没有建立连接，设备将进入不可发现模式，并且必须明确地重新启用广告功能。
aci_gap_set_non_connectable()	0x02：可扫描非定向广告	使设备进入不可连接模式。
	0x03：不可连接非定向广告	
aci_gap_set_undirect_connectable()	NA	使设备进入非定向可连接模式。

表 39. ACI：发现流程 API

ACI API	说明
aci_gap_start_limited_discovery_proc()	启动有限发现流程。命令控制器开始主动扫描。当该流程启动时，仅处于有限可发现模式的设备返回上层。
aci_gap_start_general_discovery_proc()	启动一般发现流程。命令控制器开始主动扫描。

表 40. ACI：连接流程 API

ACI API	说明
aci_gap_start_auto_conn_establishment()	启动自动连接建立流程。将指定设备添加到控制器的白名单，在发起方过滤策略被设置为 “ 使用白名单确定要连接的广告方 ” 的情况下，GAP 从控制器调用 LE_Create_Connection。
aci_gap_create_connection()	启动直接连接建立流程。在发起方过滤策略被设置为 “ 只对特定设备忽略白名单并处理可连接广告数据包 ” 的情况下， GAP 从控制器调用 LE_Create_Connection。
aci_gap_start_general_conn_establishment()	启动通用连接建立流程。在扫描仪过滤策略设置为 “ 接受所有广告数据包 ” 的情况下，主机在控制器中启用扫描，并使用事件 EVT_BLUE_GAP_DEVICE_FOUND （对于 BlueNRG 设备）和 EVT_LE_ADVERTISING_REPORT （对于 BlueNRG-MS 设备）将扫描结果中的所有设备发送至上层。
aci_gap_start_selective_conn_establishment()	启动选择性连接建立流程。 GAP 将指定设备地址添加到白名单中，并在扫描仪过滤策略设置为 “ 仅接受来自白名单中的设备的数据包 ” 的控制器中启用扫描。通过事件 EVT_BLUE_GAP_DEVICE_FOUND （在 BlueNRG 设备上）和 EVT_LE_ADVERTISING_REPORT （在 BlueNRG-MS 设备上）将找到的所有设备发送至上层。
aci_gap_terminate_gap_procedure()	终止指定的 GAP 流程。

3.4.1 设置可发现模式并使用直接连接建立流程

下列虚拟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。

/* GAP 外设：一般可发现模式 （不发送扫描响应）*/

注： *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")}
*/
void GAP_Peripheral_Make_Discoverable(void )

{

        tBleStatus ret;

        const char local_name[]=
{AD_TYPE_COMPLETE_LOCAL_NAME,'B','l','u','e','N','R','G','_','T','e','s','t'};

/* 禁用扫描响应：被动扫描 */

hci_le_set_scan_resp_data(0,NULL);

/* 使 GAP 外设进入一般可发现模式：

    Advertising_Event_Type：ADV_IND （非定向可扫描且可连接）；

    Adv_Interval_Min: 0;

    Adv_Interval_Max: 0;

    Address_Type: PUBLIC_ADDR ( 公共地址 : 0x00);
```

```
        Adv_Filter_Policy: NO_WHITE_LIST_USE ( 不使用白名单 );
        Local_Name_Length: 13
        Local_Name: BlueNRG_Test;
        Service_Uuid_Length: 0 ( 不广告服务 );
        Service_Uuid_List: NULL;
        Slave_Conn_Interval_Min: 0 ( 从设备连接间隔最小值 );
        Slave_Conn_Interval_Max: 0 ( 从设备连接间隔最大值 ).
       */
ret = aci_gap_set_discoverable(ADV_IND, 0, 0, PUBLIC_ADDR,
                                              NO_WHITE_LIST_USE,
                                              sizeof(local_name),
                                              local_name,
                                              0, NULL, 0, 0);
    if (ret != BLE_STATUS_SUCCESS) PRINTF("Failure.\n")}
} /* 结束 GAP_Peripheral_Make_Discoverable() */
/* GAP 中央设备：启动直接连接建立流程以连接处于可发现模式的 GAP 外设 */
void GAP_Central_Make_Connection(void )
{
        tBleStatus ret;
        tBDAddr GAP_Peripheral_address = {0xaa, 0x00, 0x00, 0xE1, 0x80, 0x02};
        /* 启动直接连接建立流程以连接处于一般可发现模式的 GAP 外设，使用以下连接参数：
        Scan_Interval: 0x4000;
        Scan_Window: 0x4000;
        Peer_Address_Type: PUBLIC_ADDR (GAP 外设地址类型：公共地址 );
        Peer_Address: {0xaa, 0x00, 0x00, 0xE1, 0x80, 0x02};
        Own_Address_Type: PUBLIC_ADDR ( 设备地址类型 );
        Conn_Interval_Min: 40 ( 连接事件间隔最小值 );
        Conn_Interval_Max: 40 ( 连接事件间隔最大值 );
```

Conn_Latency: 0 (大量连接事件中连接的从设备延迟);

Supervision_Timeout: 60 (LE 链路的监控超时);

Conn_Len_Min: 2000 (LE 连接所需的最小连接长度);

Conn_Len_Max: 2000 (LE 连接所需的最大连接长度).

*/

ret = aci_gap_create_connection(0x4000, 0x4000, PUBLIC_ADDR, GAP_Peripheral_address, PUBLIC_ADDR, 40, 40, 0, 60, 2000 , 2000);

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

} /* 结束 GAP_Peripheral_Make_Discoverable() */

注： *1. 如果在 GAP 流程终止时返回了 ret = BLE_STATUS_SUCCESS，将在 HCI_Event_CB() 事件回调时返回 EVT_LE_CONN_COMPLETE 事件，以指示已与 GAP_Peripheral_address 建立连接（在 GAP 外设上返回相同事件）。*

2. 可通过发出指令 aci_gap_terminate_gap_procedure() 明确终止连接流程。

3. aci_gap_create_connection() 的最后两个参数 Conn_Len_Min 和 Conn_Len_Max 是 BLE 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。

3.4.2 设置可发现模式并使用一般发现流程 （主动扫描）

下列虚拟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。

/* GAP 外设：一般可发现模式 （发送扫描响应）：

注： *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")}
*/
void GAP_Peripheral_Make_Discoverable(void )
{
```

```
    tBleStatus ret;
    const char local_name[] =
{AD_TYPE_COMPLETE_LOCAL_NAME,'B','l','u','e','N','R','G' };
 /* 作为扫描响应数据，使用专有 128 位服务 UUID。
    由于长度限制 （31 字节），此 128 位数据不能插入广告数据包 （ADV_IND）。
*/
/*
AD 类型描述：
0x11：长度
0x06：128 位服务 UUID 类型
0x8a,0x97,0xf7,0xc0,0x85,0x06,0x11,0xe3,0xba,0xa7,0x08,0x00,0x20,0x0c,0x9a,0x66: 128 bits
Service UUID

*/
    uint8_t ServiceUUID_Scan[18]=
{0x11,0x06,0x8a,0x97,0xf7,0xc0,0x85,0x06,0x11,0xe3,0xba,0xa7,0x08,0x00,0x20,0x0c,0x9a,0x66};
/* 启用在 GAP 外设接收到
  来自 GAP 中央设备的执行一般
  发现流程 （主动扫描）的扫描请求时发送扫描响应的功能 */
hci_le_set_scan_resp_data(18 , ServiceUUID_Scan);

/* 使 GAP 外设进入一般可发现模式：
    Advertising_Event_Type：ADV_IND （非定向可扫描且可连接）；
    Adv_Interval_Min: 0;
    Adv_Interval_Max: 0;
    Address_Type: PUBLIC_ADDR ( 公共地址 : 0x00);
    Adv_Filter_Policy: NO_WHITE_LIST_USE ( 不使用白名单 );
    Local_Name_Length: 8
    Local_Name: BlueNRG;
    Service_Uuid_Length: 0 ( 不广告服务 );
    Service_Uuid_List: NULL;
    Slave_Conn_Interval_Min: 0 ( 从设备连接间隔最小值 );
    Slave_Conn_Interval_Max: 0 ( 从设备连接间隔最大值 ).
*/
ret = aci_gap_set_discoverable(ADV_IND, 0, 0, PUBLIC_ADDR,
NO_WHITE_LIST_USE,sizeof(local_name), local_name, 0, NULL, 0, 0);
if (ret != BLE_STATUS_SUCCESS) PRINTF("Failure.\n")}
} /* 结束 GAP_Peripheral_Make_Discoverable() */

/* GAP 中央设备：启动一般发现流程以发现处于可发现模式的 GAP 外设 */
void GAP_Central_General_Discovery_Procedure(void )
{
    tBleStatus ret;
```

```
    /* 使用以下参数启动一般发现流程 （主动扫描）：
    Scan_Interval: 0x4000;
    Scan_Window: 0x4000;
    Own_address_type: 0x00 ( 设备公共地址 );
    filterDuplicates: 0x00 ( 重复过滤已禁用 );

    ret = aci_gap_start_general_discovery_proc(0x4000, 0x4000,0x00,0x00);
    if (ret != BLE_STATUS_SUCCESS) PRINTF("Failure.\n")}
}
```

通过 HCI_Event_CB() 回调 （EVT_VENDOR 作为主要事件）发起的 EVT_BLUE_GAP_DEVICE_FOUND （BlueNRG）和 EVT_LE_ADVERTISING_REPORT （BlueNRG-MS）事件提供流程响应。 通过 HCI_Event_CB() 回调时的 EVT_BLUE_GAP_PROCEDURE_COMPLETE 事件指示流程结束 （EVT_VENDOR 作为主要事件）：

```
void HCI_Event_CB(void *pckt)
{
    hci_uart_pckt *hci_pckt = pckt;
    hci_event_pckt *event_pckt = (hci_event_pckt*)hci_pckt->data;
    if(hci_pckt->type != HCI_EVENT_PKT return;
    switch(event_pckt->evt){
    case EVT_VENDOR:
{
/* 获取供应商事件数据 */
evt_blue_aci *blue_evt = (void*)event_pckt->data;
switch(blue_evt->ecode){
case EVT_BLUE_GAP_DEVICE_FOUND:
{
evt_gap_device_found *pr = (void*)blue_evt->data;
/* evt_gap_device_found 参数：
    pr->evt_type：事件类型 （广告数据包类型）；
    pr->bdaddr_type：对端地址的类型 (PUBLIC_ADDR,RANDOM_ADDR) ；
    pr->bdaddr：在扫描期间发现的对端设备地址；
    pr->length：广告或扫描响应数据的长度；
    pr->data_RSSI[]：广告或扫描响应数据 + RSSI 的长度。
RSSI 是最后一个八位组 （有符号整数）。
*/
/* 添加用户代码以便根据特定的 pr->evt_type (ADV_IND, SCAN_RSP, ..) 解码 evt_gap_device_found 事件数据 */
……
}/* EVT_BLUE_GAP_DEVICE_FOUND */
break;
case EVT_BLUE_GAP_PROCEDURE_COMPLETE:
{
    /* 在一般发现流程终止时，
```

```
    返回 EVT_BLUE_GAP_PROCEDURE_COMPLETE 事件，流程代码设置为
GAP_GENERAL_DISCOVERY_PROC (0x02)。
*/
    evt_gap_procedure_complete *pr = (void*)blue_evt->data;
    /* evt_gap_procedure_complete 参数：
    pr->procedure_code：终止的流程代码；
    pr->status：BLE_STATUS_SUCCESS、 BLE_STATUS_FAILED 或 ERR_AUTH_FAILURE ；
    pr->data[VARIABLE_SIZE]：流程特定的数据 （如适用）
    */
/* 如果需要，添加处理事件数据的用户代码 */
.....
              }/* EVT_BLUE_GAP_PROCEDURE_COMPLETE */
              break;
         }/* switch(blue_evt->ecode) */
    }/* EVT_VENDOR */
    break;

case EVT_LE_META_EVENT:
{
  evt_le_meta_event *evt = (void *)event_pckt->data;
  switch(evt->subevent)
  {
    case EVT_LE_ADVERTISING_REPORT: /* BlueNRG-MS 栈 */
    {
      le_advertising_info *pr = (void *)(evt->data+1); /* evt->data[0] 报告的数量 （对于 BlueNRG-MS，
该值始终为 1）*/
      /* le_advertising_info 参数：
           pr->evt_type：事件类型 （广告数据包类型）；
           pr->bdaddr_type：对端地址的类型 (PUBLIC_ADDR,RANDOM_ADDR) ；
           pr->bdaddr：在扫描期间发现的对端设备地址；
           pr->length：广告或扫描响应数据的长度；
           pr->data_RSSI[]：广告或扫描响应数据 + RSSI 的长度。
           RSSI 是最后一个八位组 （有符号整数）。
       */
      /* 添加用户代码以便根据特定的 pr->evt_type (ADV_IND, SCAN_RSP, ..) 解码
le_advertising_info 事件数据 */
      ...
    }/* EVT_LE_ADVERTISING_REPORT */
    break;
  }/* 结束 switch() */
}/* EVT_LE_META_EVENT */
break;
    }/* switch(event_pckt->evt)*/
}/* end HCI_Event_CB() */
```

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_Event_CB() 回调将发起下列事件：

1. EVT_BLUE_GAP_DEVICE_FOUND （BlueNRG） /EVT_LE_ADVERTISING_REPORT （BlueNRG-MS），广告数据包数据类型 （evt_type = ADV_IND）
2. EVT_BLUE_GAP_DEVICE_FOUND （BlueNRG 设备） /EVT_LE_ADVERTISING_REPORT （BlueNRG-MS），扫描响应数据包类型 （evt_type = SCAN_RSP）

表 41. ADV_IND 事件

事件类型	地址类型	地址	广告数据	RSSI
0x00 (ADV_IND)	0x00 (公共地址);	0x0280E1003412	0x02,0x01,0x06,0x08,0x08,0x42,0x6C,0x75,0x65,0x4E,0x52,0x47,0x02,0x0A,0x08	0xDA

广告数据可作如下解释 （请参考*第 5 节：参考文献*中提到的蓝牙规范 4.0 版 [第 3 卷] 和 4.1 版 [第 2 卷]）：

表 42. ADV_IND 广告数据

AD 类型标记字段	本地名称字段	发送功率水平
0x02：字段的长度 0x01：AD 类型标记 0x06：0x110 （第 2 位：不支持 BR/EDR ；第 1 位：一般可发现模式）	0x08：字段的长度 0x08：本地名称类型缩写 0x42,0x6C,0x75,0x65,0x4E0x52,0x47: BlueNRG	0x02：字段的长度 0x0A：发送功率类型 0x08：功率值

表 43. SCAN_RSP 事件

事件类型	地址类型	地址	扫描响应数据	RSSI
0x04 (SCAN_RSP)	0x00 (公共地址);	0x0280E1003412	0x12,0x66,0x9A,0x0C,0x20,0x00,0x08,0xA7,0xBA,0xE3,0x11,0x06,0x85,0xC0,0xF7,0x97,0x8A,0x06,0x11	0xDA

扫描响应数据可作如下解释 （请参考中提到的蓝牙规范 4.0 版 [第 3 卷] 和 4.1 版 [第 2 卷]）：

表 44. 扫描响应数据

扫描响应数据
0x12：数据长度 0x11：服务 UUID 广告数据的长度； 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

3.5 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）。

在使用安全功能时，必须设置一些底层参数 （根密钥），然后才能发出任何其他 ACI 指令：

- 用于得出 CSRK 的 DIV 根密钥
- 用于得出 LTK 和 CSRK 的加密根 （ER）密钥
- 用于得出 IRK 和 CSRK 的身份根 （IR）密钥

外部微控制器 （MCU）负责提供这些参数：

1. 它必须随机生成三个根密钥值 （如果尚未生成），并将它们保存在非易失性存储器中。为了意义明确地建立特定的 BlueNRG、 BlueNRG-MS 设备安全设置，这三个根密钥只能一次生成。
2. 每次 MCU 启动时，必须从非易失性存储器读取根密钥，并在 BlueNRG、 BlueNRG-MS 设备的初始化阶段设置它们。

下列简单的虚拟代码显示了如何设置 BlueNRG、 BlueNRG-MS 设备上随机生成的安全根密钥的读取：

```
uint8_t DIV[2];
uint8_t ER[16];
uint8_t IR[16];

/* 复位 BlueNRG、 BlueNRG-MS 设备 */
BlueNRG_RST();

/* 微控制器特定的实现：
    1) MCU 必须随机生成 DIV、 ER 和 IR，并将它们保存在非易失性存储器中。
    2) 当 MCU 启动时，必须从非易失性存储器读取 DIV、 ER 和 IR 值。
*/
……

/* 配置 BlueNRG、 BlueNRG-MS 设备上的读取根密钥 DIV  */
ret = aci_hal_write_config_data(CONFIG_DATA_DIV_OFFSET,
CONFIG_DATA_DIV_LEN,(uint8_t *) DIV);

/* 配置 BlueNRG、 BlueNRG-MS 设备上的读取根密钥 ER */
ret = aci_hal_write_config_data(CONFIG_DATA_ER_OFFSET,
CONFIG_DATA_ER_LEN,(uint8_t *) ER);

/* 配置 BlueNRG、 BlueNRG-MS 设备上的读取根密钥 IR */
```

```
ret = aci_hal_write_config_data(CONFIG_DATA_IR_OFFSET,
CONFIG_DATA_IR_LEN,(uint8_t *) IR);
```

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：仅显示

0x01：显示是 / 否

0x02：仅键盘

0x03：无输入，无输出

0x04：键盘，显示

2 个 BlueNRG 设备 （Device_1、 Device_2）的输入密钥示例

下列虚拟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 11：计算临时密钥 （TK）的方法*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_auth_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，是否使用固定引脚，以及是否启用绑定）。

下列虚拟代码示例仅描述了为设备设置以下验证要求要执行的特定步骤：“MITM 保护，无 OOB 数据，不使用固定引脚 ”：这一配置用于验证链路和在使用输入密钥法的配对过程中使用非固定引脚。

```
ret = aci_gap_set_auth_requirement(MITM_PROTECTION_REQUIRED,
                                   OOB_AUTH_DATA_ABSENT, /* 无 OOB 数据 */
                                   NULL, /* 无 OOB 数据 */
                                   7,  /* 加密 密钥的最小长度 */
                                   16, /* 加密密钥的最大长度 */
                                  DONOT_USE_FIXED_PIN_FOR_PAIRING,/* 无固定引脚
*/
                                   0, /* 不使用固定引脚 */
```

```
                                                BONDING /* 绑定启用 */);
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_request() （它向主设备发送从设备安全请求）：

```
tBleStatus ret;
ret = aci_gap_slave_security_request(conn_handle,
                                     BONDING,
                                     MITM_PROTECTION_REQUIRED
);
if (ret != BLE_STATUS_SUCCESS) PRINTF("Failure.\n")}
```

或

2. 在 GAP 中央 （主）设备上使用 aci_gap_send_pairing_request()。

由于设置了 DONOT_USE_FIXED_PIN_FOR_PAIRING （无固定引脚），一旦 2 个设备中的 1 个启动了配对流程， BlueNRG、 BlueNRG-MS 将生成 EVT_BLUE_GAP_PASS_KEY_REQUEST 事件 （具有相关连接句柄），以请求主机应用提供设定加密密钥要使用的密码。 BlueNRG、 BlueNRG-MS 应用必须通过使用 aci_gap_pass_key_response(conn_handle,passkey)API 提供正确密码。

下列虚拟代码示例仅描述了当 Device_1 上生成了 EVT_BLUE_GAP_PASS_KEY_REQUEST 事件时，提供配对过程将使用的输入密钥时要执行的特定步骤 （“ 仅显示 ” 功能）：

```
tBleStatus ret;

/* 使用特定用户函数生成随机引脚 */

pin = generate_random_pin();

ret = aci_gap_slave_security_request(conn_handle,
                                     BONDING,
                          MITM_PROTECTION_REQUIRED
                                     );
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引脚。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_response() API 为 Device_2 提供 Device_1 上显示的引脚。

或者，如果用户想要设置具有固定引脚 0x123456 的验证要求 （无需输入密钥事件），可以使用下列虚拟代码：

```
tBleStatus ret;

ret = aci_gap_set_auth_requirement(MITM_PROTECTION_REQUIRED,
                                   OOB_AUTH_DATA_ABSENT,NULL,
                                   7,
                                   16,
                                   USE_FIXED_PIN_FOR_PAIRING,
                                   0x123456,/* 固定引脚 */
                                   BONDING
                                   );
if (ret != BLE_STATUS_SUCCESS) PRINTF("Failure.\n")}
```

注意：

1. 如果通过调用所述 API （aci_gap_slave_security_request() 或 aci_gap_send_pairing_request()）启动配对流程并在流程结束时返回值 ret = BLE_STATUS_SUCCESS，将在 HCI_Event_CB() 事件回调时返回 EVT_BLUE_GAP_PAIRING_CMPLT 事件以指示配对状态：

- 0x00：配对成功；
- 0x01：配对超时；
- 0x02：配对失败。

从与 EVT_BLUE_GAP_PAIRING_CMPLT 事件关联的 evt_gap_pairing_cmplt 数据的状态字段给出配对状态。

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）。主机应用可以直接使用相同 API，不执行配对流程而直接加密链路：

- aci_gap_slave_security_request() （对于 GAP 外围 （从）设备）

或

- aci_gap_send_pairing_request() （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，从设备将接收到 EVT_BLUE_GAP_BOND_LOST 事件，通知主机应用它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。
4. 或者，可以使用 aci_gap_set_auth_requirement() 选择带外法，同时启用 OOB_Enable 字段并在 OOB_Data 中指定 OOB 数据。这意味着两个设备都在使用该方法，并设置了通过带外通信 （例如， NFC）定义的相同 OOB 数据。

3.6 私有

正在广告的设备始终具有相同地址 （公共或静态随机），可通过扫描仪跟踪。

为避免这样做，可以在广告设备上启用私有功能。在启用了私有的设备上，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。

3.6.1 BlueNRG （蓝牙 v4.0）

外设

根据蓝牙规范 4.0，如果外设私有标记特征存在并设置为 1，私有启用。

BlueNRG 栈自动将外设私有标记特征添加到 GAP 服务中。如果绑定数小于或等于 1，外设私有标记特征为可写入状态，因此可通过中央设备修改。

通过在初始化 GAP 层后启用外设私有标记特征的值，可默认启用私有。

当私有启用时，外设仅使用公共或静态地址。如果私有启用，外设将在非定向可连接模式下使用私有，并以可解析私有地址进行广告。如果私有启用且重新连接地址特征存在，外设即使在定向可连接模式下也将使用私有，以重新连接地址进行广告。

在 GAP 初始化期间， BlueNRG 栈自动添加重新连接地址特征 （参见*表 33：BlueNRG GATT、 GAP 默认特征*）。

若需更多信息，请参考蓝牙核心规范 v4.0 第 3 卷 C 部分第 10.7.1 节。10.7.1.

中央设备

在作为中央设备时， BlueNRG 始终启用私有。

当使用通用或自动连接建立流程时， BlueNRG 将使用不可解析私有地址作为自己的地址。具体来说，在通用连接建立流程中创建连接后， BlueNRG 将读取外设私有标记特征并检查重新连接地址是否存在。如果外设上启用了私有并且重新连接地址存在，它会将新的重新连接地址自动写入外设的重新连接地址特征。将生成事件 （EVT_BLUE_GAP_RECONNECTION_ADDRESS），以通知应用新的重新连接地址已生成，可以用于与外设的后续连接。

3.6.2 BlueNRG-MS （蓝牙 v4.1）

在蓝牙 4.1 中，私有功能已从版本 1.0 更新到版本 1.1。此新版本只是进行了简化。重新连接地址的使用不再能够预见。

外设

启用了私有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。为了连接中央设备，只应使用非定向可连接模式。这种情况下，将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。建议在设备广告数据时不要发送设备名称。

中央设备

启用了私有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，应使用通用连接建立流程。该流程将使用可解析私有地址作为发起方的设备地址。

每间隔 15 分钟后将生成可解析或不可解析私有地址。

广播方

启用了私有的广播方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。广播方不应在广告数据中发送设备名称。

监听方

启用了私有的监听方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。

3.6.3 解析地址

当设备想要解析一个可解析私有地址时，可以使用 aci_gap_resolve_private_address() 函数。这在 BlueNRG 主设备想要连接已知设备时十分有用。

当 BlueNRG 从设备连接了绑定设备时，无需解析主设备的私有地址即可加密链路。事实上， BlueNRG 栈将自动寻找正确的 LTK 以加密链路。

3.7 服务和特征发现

本节介绍在两个设备建立连接后，允许 BlueNRG、 BlueNRG-MS GAP 中央设备发现 GAP 外设服务和特征的主要函数。

在下面的虚拟代码示例中，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 45. BlueNRG 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	0x0019	NA	0x001A

表 46. BlueNRG-MS 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	0x000C	NA	NA	NA
	自由落体特征	0x000D	0x000E	0x000F	NA
	加速特征	0x0010	0x0011	0x0012	NA
环境服务	NA	0x0013	NA	NA	NA
	温度特征	0x0014	0xx0015	NA	0x0016

注： *属性值句柄取决于为标准 GAP 服务预留的最后一个属性句柄 （BlueNRG 栈为 0x000F， BlueNRG-MS 栈为 0x000B）。*

关于传感器感测演示的详细信息，请参考用户手册 UM1686 和开发套件软件包中的传感器演示源代码 （参见*参考文献*）。在下面的例子中， BlueNRG GAP 外设传感器感测演示环境服务只定义了温度特征 （没有使用具有压力和湿度传感器的扩展板）。

3.7.1 服务发现流程与相关 GATT 事件

下面是包含相关描述的服务发现 API 列表。

表 47. ACI：服务发现流程 API

发现服务 API	说明
aci_gatt_disc_all_prim_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_disc_ prim_services_by_service_uuid()	此 API 启动 GATT 客户端流程，以使用其 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_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。

以下虚拟代码示例描述了 aci_gatt_disc_all_prim_services() API：

```
/* GAP 中央设备启动发现所有服务的流程：conn_handle 是 HCI_Event_CB() 事件回调，即
EVT_LE_CONN_COMPLETE 事件返回的连接句柄 */

if (aci_gatt_disc_all_prim_services(conn_handle) != BLE_STATUS_SUCCESS)

{

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

}
```

通过 HCI_Event_CB() 回调 （EVT_VENDOR 作为主要事件）发起的 EVT_BLUE_ATT_READ_BY_GROUP_RESP 事件提供流程响应。 通过 HCI_Event_CB() 回调（EVT_VENDOR 作为主要事件）时的 EVT_BLUE_GATT_PROCEDURE_COMPLETE 事件指示流程结束：

```
void HCI_Event_CB(void *pckt)

{
  hci_uart_pckt *hci_pckt = pckt;
  hci_event_pckt *event_pckt = (hci_event_pckt*)hci_pckt->data;
  if(hci_pckt->type != HCI_EVENT_PKT return;
  switch(event_pckt->evt){
    case EVT_VENDOR:
    {
```

```
/* 获取供应商事件数据 */
evt_blue_aci *blue_evt = (void*)event_pckt->data;
switch(blue_evt->ecode){
case EVT_BLUE_ATT_READ_BY_GROUP_RESP:
    {
 evt_att_read_by_group_resp *pr = (void*)blue_evt->data;
 /* evt_att_read_by_group_resp 参数：
       pr->conn_handle：连接句柄；
       pr->event_data_length：事件数据的总长度；
       pr->attribute_data_length：
      attribute_data_list[] 中每个特定数据的长度；
       pr->attribute_data_list[]：事件数据。
*/
/* 添加用户代码，用于解码 pr->attribute_data_list[] 并获取服务句柄、结束组句柄和服务 UUID */
.........
}/* EVT_BLUE_ATT_READ_BY_GROUP_RESP */
break;
case EVT_BLUE_GATT_PROCEDURE_COMPLETE:
{
  evt_gatt_procedure_complete *pr = (void*)blue_evt->data;
     /* evt_gatt_procedure_complete 参数：
     pr->conn_handle：连接句柄；
     pr->attribute_data_length：事件数据的长度；
     pr->data[]：事件数据。
*/
 /* 如果需要，添加使用事件数据的用户代码 */
 .......
}/* EVT_BLUE_GATT_PROCEDURE_COMPLETE */
 break;
}/* switch(blue_evt->ecode) */
}/* EVT_VENDOR */
break;
}/* switch(evt->subevent)*/
}/* end HCI_Event_CB() */
```

就传感器感测演示而言， GAP 中央设备应用应获取三个 EVT_BLUE_ATT_READ_BY_GROUP_RESP 事件，包含下列 evt_att_read_by_group_resp 数据：

第一个 evt_att_read_by_group_resp 事件数据

pr->conn_handle : 0x0801 (连接句柄);

pr->event_data_length: 0x0D (事件数据的长度);

pr->handle_value_pair_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);

pr-> attribute_data_list: 0x0C 字节如下：

表 48. 第一个 evt_att_read_by_group_resp 事件数据

服务句柄	结束组句柄	服务 UUID	注释
0x0001	0x0004	0x1801	属性配置文件服务（由 GATT_Init() 添加）。 标准 16 位服务 UUID。
0x0005	0x000F (BlueNRG), 0x000B (BlueNRG-MS)	0x1800	GAP 配置文件服务 (GAP_Init() adds it). 标准 16 位服务 UUID。

第二个 evt_att_read_by_group_resp 事件数据：

pr->conn_handle : 0x0801 (连接句柄);

pr->event_data_length: 0x15 (事件数据的长度);

pr->attribute_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);

pr-> attribute_data_list content: 0x14 字节如下：

表 49. 第二个 evt_att_read_by_group_resp 事件数据

服务句柄	结束组句柄	服务 UUID	注释
0x0010 (BlueNRG), 0x000C (BlueNRG-MS)	0x0016 (BlueNRG), 0x0012 (BlueNRG-MS)	0x02366E80CF3A11E19AB40002A5D5C51B	加速服务 128 位服务专有 UUID

第三个 evt_att_read_by_group_resp 事件数据：

pr->conn_handle : 0x0801 (连接句柄);

pr->event_data_length: 0x15 (事件数据的长度);

pr->attribute_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);

pr-> attribute_data_list: 0x14 字节如下：

表 50. 第三个 evt_att_read_by_group_resp 事件数据

服务句柄	结束组句柄	服务 UUID	注释
0x0017 (BlueNRG), 0x0013 (BlueNRG-MS)	0x001A (BlueNRG), 0x0016 (BlueNRG-MS)	0x42821A40E47711E282D00002A5D5C51B	环境服务 128 位服务专有 UUID

就传感器感测演示而言，当发现所有主要服务流程完成时， GAP 中央设备应用上生成 EVT_BLUE_GATT_PROCEDURE_COMPLETE，包含下列 evt_gatt_procedure_complete 数据：

pr->conn_handle : 0x0801 (连接句柄);

pr-> data_length: 0x01 (事件数据的长度);

pr->data[]: 0x00 (事件数据).

3.7.2 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 51. BlueNRG ACI：特征发现流程 API

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

就 BlueNRG 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 49：第二个 evt_att_read_by_group_resp 事件数据*）：

```
uint16_t service_handle;
uint16_t end_group_handle;

#ifdef BLUENRG_MS

service_handle = 0x000C;
end_group_handle = 0x0012;

#else /*
service_handle_value  = 0x0010;
charac_handle_value = 0x0016;

#endif


/* BlueNRG GAP 中央设备启动发现所有服务特征的流程：conn_handle 是 HCI_Event_CB() 事件回调，即 EVT_LE_CONN_COMPLETE 事件返回的连接句柄 */
if (aci_gatt_disc_all_charac_of_serv(conn_handle,
                                     service_handle,/*  服务句柄 */
                                     end_group_handle/*  结束组句柄 */
                                     );) != BLE_STATUS_SUCCESS)
{
  if (ret != BLE_STATUS_SUCCESS) PRINTF("Failure.\n")}
}
```

通过 HCI_Event_CB() 回调 （EVT_VENDOR 作为主要事件）发起的 EVT_BLUE_ATT_READ_BY_TYPE_RESP 事件提供流程响应。通过 HCI_Event_CB() 回调（EVT_VENDOR 作为主要事件）时的 EVT_BLUE_GATT_PROCEDURE_COMPLETE 事件指示流程结束：

```
void HCI_Event_CB(void *pckt)

{
```

```
hci_uart_pckt *hci_pckt = pckt;
hci_event_pckt *event_pckt = (hci_event_pckt*)hci_pckt->data;
if(hci_pckt->type != HCI_EVENT_PKT return;
switch(event_pckt->evt){
 case EVT_VENDOR:
 {
  /* 获取供应商事件数据 */
  evt_blue_aci *blue_evt = (void*)event_pckt->data;
  switch(blue_evt->ecode){
  case EVT_BLUE_ATT_READ_BY_TYPE_RESP:
       {
        evt_att_read_by_type_resp *pr = (void*)blue_evt->data;
        /* evt_att_read_by_type_resp 参数：
            pr->conn_handle：连接句柄；
            pr->event_data_length：事件数据的总长度；
            pr->handle_value_pair_length：handle_value_pair[] 中每个特定数据的长度；
            pr->handle_value_pair[]：事件数据。
   */
       /* 添加用户代码，用于解码 pr->handle_value_pair[] 并获取特征句柄、属性、特征值句柄、
       特征 UUID */
   ……
  }/* EVT_BLUE_ATT_READ_BY_TYPE_RESP */
  break;
  case EVT_BLUE_GATT_PROCEDURE_COMPLETE:
       {
        evt_gatt_procedure_complete *pr = (void*)blue_evt->data;
        /* evt_gatt_procedure_complete 参数：
            pr->conn_handle：连接句柄；
            pr->data_length：事件数据的长度；
```

```
                pr->data[]：事件数据。
        */
            /* 如果需要，添加使用事件数据的用户代码 */
        ……
      }/* EVT_BLUE_GATT_PROCEDURE_COMPLETE */
      break;
    }/* switch(blue_evt->ecode) */
  }/* EVT_VENDOR */
  break;
 }/* switch(evt->subevent)*/
}/* end HCI_Event_CB() */
```

就 BlueNRG 传感器感测演示而言， GAP 中央设备应用应获取两个 EVT_BLUE_ATT_READ_BY_TYPE_RESP 事件，包含下列 evt_att_read_by_type_resp 数据：

第一个 evt_att_read_by_type_resp 事件数据

pr->conn_handle : 0x0801 (连接句柄);

pr->event_data_length: 0x16 (事件数据的长度);

pr->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);

pr->handle_value_pair: 0x15 字节如下：

表 52. 第一个 evt_att_read_by_type_resp 事件数据

特征句柄	特征属性	特征值句柄	特征 UUID	注释
0x0011 (BlueNRG), 0x000D (BlueNRG-MS)	0x10 （通知）	0x0012 (BlueNRG), 0x000E (BlueNRG-MS)	0xE23E78A0CF4A11E18FFC0002A5D5C51B	自由落体特征 128 位特征专有 UUID

第二个 evt_att_read_by_type_resp **事件数据**

pr->conn_handle : 0x0801 (连接句柄);

pr->event_data_length: 0x16 (事件数据的长度);

pr->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);

pr->handle_value_pair: 0x15 字节如下：

表 53. 第二个 evt_att_read_by_type_resp 事件数据

特征句柄	特征属性	特征值句柄	特征 UUID	注释
0x0014 (BlueNRG), 0x0010 (BlueNRG-MS)	0x12 （通知并读取）	0x0015 (BlueNRG), 0x0011 (BlueNRG-MS)	0x340A1B80CF4B11E1AC360002A5D5C51B	加速特征 128 位特征专有 UUID

就传感器感测演示而言，当发现所有服务特征流程完成时， GAP 中央设备应用上生成 EVT_BLUE_GATT_PROCEDURE_COMPLETE，包含下列 evt_gatt_procedure_complete 数据：

pr->conn_handle : 0x0801 (连接句柄);

pr-> data_length: 0x01 (事件数据的长度);

pr->data[]: 0x00 (事件数据).

可以执行类似步骤，以便发现环境服务的所有特征 （请参考*表 45：BlueNRG 传感器感测演示服务和特征句柄*和*表 46：BlueNRG-MS 传感器感测演示服务和特征句柄*）。

3.8 特征通知 / 指示、写入、读取

本节介绍访问 BLE 设备特征的主要函数。

表 54. 特征更新、读取、写入 API

发现服务 API	说明	位置
aci_gatt_update_char_value()	如果对特征启用了通知 （或指示），该 API 向客户端发送通知 （或指示）。	GATT 服务器
aci_gatt_read_charac_val()	启动读取属性值的流程。	GATT 客户端
aci_gatt_write_charac_value()	启动写入属性值的流程 （当流程完成时，生成 EVT_BLUE_GATT_PROCEDURE_COMPLETE 事件）。	GATT 客户端
aci_gatt_write_without_response()	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_write_charac_descriptor()	启动写入特征描述符的流程 .	GATT 客户端
aci_gatt_confirm_indication()	确认指示。如果应用在接收到特征指示时接收到事件 EVT_BLUE_GATT_INDICATION，必须发送该指令。	GATT 客户端

就传感器感测演示而言，下面的简单虚拟代码描述了应如何使用 GAP 中央设备应用，以配置自由落体和加速特征客户端描述符配置用于通知：

```
tBleStatus ret;

uint16_t handle_value;

#ifdef BLUENRG_MS

handle_value = 0x000F;

#else /*

handle_value = 0x0013;

#endif

/* 启用自由落体特征客户端描述符配置用于 ret = aci_gatt_write_charac_descriptor(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")}
#ifdef BLUENRG_MS

handle_value = 0x0012;

#else /*

handle_value = 0x0016;
```

```
#endif
/* 启用加速特征客户端描述符配置
用于通知 */
ret = aci_gatt_write_charac_descriptor (conn_handle,
                                        handle_value /* 加速句柄
                                                        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")}
```

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 service_handle_value;
uint16_t charac_handle_value;

#ifdef BLUENRG_MS

service_handle_value = 0x000C;
charac_handle_value = 0x000D;

#else /*
service_handle_value  = 0x0010;
charac_handle_value = 0x0011;

#endif
/* GAP 外设将自由落体特征通知 GAP 中央设备 */
ret = aci_gatt_update_char_value (service_handle_value , /* 加速服务句柄 */
                                  charac_handle_value, /* 自由落体特征
                                                          句柄 */
                                  0, /* 特征
                                        值偏移 */
                                  0x01,/* 特征值
                                  长度 */
                                 &val, /* 特征值 */ );
ret = (accServHandle, freeFallCharHandle, 0, 1, &val);
if (ret != BLE_STATUS_SUCCESS) PRINTF("Failure.\n")}

tBleStatus ret;
```

```
uint8_t buff[6];
#ifdef BLUENRG_MS

charac_handle_value = 0x0010;

#else /*

charac_handle_value = 0x0014;

#endif
/* 设置缓冲队列中 x、 y、 z 轴上的 MEMS 加速值 */
....
/* GAP 外设将加速特征通知 GAP 中央设备 */
ret = aci_gatt_update_char_value (service_handle_value , /* 加速
                                                          服务句柄 */
                                  charac_handle_value, /* 加速
                                                          特征
                                                          句柄 */
                                  0 , /* 特征
                                         值偏移 */
                                  0x06, /* 特征
                                          值长度 */
                                  buff, /* 特征
                                          值 */
                                 );
ret = (accServHandle, freeFallCharHandle, 0, 1, &val);
if (ret != BLE_STATUS_SUCCESS) PRINTF("Failure.\n")}
```

在 GAP 中央设备上， HCI_Event_CB() 回调 （EVT_VENDOR 作为主要事件），在接收到来自 GAP 外设的特征通知 （加速或自由落体）时发起 EVT_BLUE_GATT_NOTIFICATION。下面是 HCI_Event_CB() 回调的虚拟代码：

```
void HCI_Event_CB(void *pckt)
{
 hci_uart_pckt *hci_pckt = pckt;
 hci_event_pckt *event_pckt = (hci_event_pckt*)hci_pckt->data;
 if(hci_pckt->type != HCI_EVENT_PKT return;
 switch(event_pckt->evt){
  case EVT_VENDOR:
  {
   /* 获取供应商事件数据 */
   evt_blue_aci *blue_evt = (void*)event_pckt->data;
   switch(blue_evt->ecode)
   {
    case EVT_BLUE_GATT_NOTIFICATION:
    {
   evt_gatt_attr_notification *evt = (evt_gatt_attr_notification*)blue_evt->data;
```

```
/*
        evt_gatt_attr_notification 数据：
        evt->conn_handle：连接句柄；
        evt->event_data_length：属性值 + 句柄的长度 （2 字节）；
        evt->attr_handle：通知的特征的句柄；
        evt->attr_value[]：特征值。
   根据应用场景为处理接收到的通知添加
   用户代码。
  */
   ........
            }
            break;
        }/* switch(blue_evt->ecode)*/
    }/* switch(evt->subevent)*/
}/* end HCI_Event_CB() */
```

3.9 基本 / 典型错误条件描述

在 BlueNRG、 BlueNRG-MS ACI 框架上，定义了 tBleStatus 类型以便返回 BlueNRG、BlueNRG-MS 栈错误状态。头文件 “ble_status.h” 中定义了状态和错误代码。

在调用栈 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)。关于每个 ACI API 的相关错误状态的详细列表，请参考*第 5 节：参考文献*中的 UM1755 和 UM1865 用户手册。

3.10 BlueNRG-MS 同时作为主、从设备的场景

BlueNRG-MS 设备栈可同时支持多个角色。因此，同一设备既可作为一个或多个连接上的主设备 （栈模式 3 支持最多八个连接），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 BlueNRG-MS 设备以便能够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, &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 10. BlueNRG-MS 同时作为主、从设备的场景

1. 通过在 GAP_Init() API 上将角色设置为 GAP_PERIPHERAL_ROLE | GAP_CENTRAL_ROLE，将一个 BlueNRG-MS 设备 （称为 “ 主设备兼从设备 ”）配置为中央设备及外设。然后，配置栈模式 3，以便能够连接一个以上的外设。假设该设备定义了具有特征的服务。
2. 通过在 GAP_Init() API 上将角色设置为 GAP_PERIPHERAL_ROLE，将两个 BlueNRG-MS 设备 （称为 Slave_A、 Slave_B）配置为外设。 Slave_A 和 Slave_B 均作为 “ 主设备兼从设备 ” 定义了相同的服务和特征。
3. 通过在 GAP_Init() API 上将角色设置为 GAP_CENTRAL_ROLE，将一个 BlueNRG-MS 设备 （称为主设备）配置为中央设备。
4. Slave_A 和 Slave_B 设备均进入发现模式，如下所示：

```
ret =aci_gap_set_discoverable(Advertising_Type= 0x00,
                              Advertising_Interval_Min=0x20,
                              Advertising_Interval_Max=0x100,
                              Local_Name_Length=0x05,
                              Local_Name=[0x08,0x74,0x65,0x73,0x74],
                              Slave_Conn_Interval_Min = 0x0006,
                              Slave_Conn_Interval_Max = 0x0008)
```

5. 主设备兼从设备执行发现流程，以便发现外设 Slave_A 和 Slave_B：

```
ret = aci_gap_start_gen_disc_proc (LE_Scan_Interval=0x10,
                                   LE_Scan_Window=0x10)
```

通过 EVT_LE_ADVERTISING_REPORT 事件发现两个设备。

6. 在发现两个设备后，主设备兼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 Slave_A：Slave_A 的地址类型和地址已在 EVT_LE_ADVERTISING_REPORT 事件的发现流程中找到 */
ret= aci_gap_create_connection(LE_Scan_Interval=0x0010,
                               LE_Scan_Window=0x0010,
                               Peer_Address_Type= ”Slave_A 地址类型 ”,
                               Peer_Address= ”Slave_A 地址 ”,
                               Conn_Interval_Min=0x6c,
                               Conn_Interval_Max=0x6c,
                               Conn_Latency=0x00,
                               Supervision_Timeout=0xc80,
                               Minimum_CE_Length=0x000c,
                               Maximum_CE_Length=0x000c)

/* 连接 Slave_B：Slave_B 的地址类型和地址已在 EVT_LE_ADVERTISING_REPORT 事件的发现流程中找到 */
  ret= aci_gap_create_connection(LE_Scan_Interval=0x0010,
                               LE_Scan_Window=0x0010,
                               Peer_Address_Type= ”Slave_B 地址类型 ”,
                               Peer_Address= ”Slave_B 地址 ”,
                               Conn_Interval_Min=0x6c,
                               Conn_Interval_Max=0x6c,
                               Conn_Latency=0x00,
                               Supervision_Timeout=0xc80,
                               Minimum_CE_Length=0x000c,
                               Maximum_CE_Length=0x000c)
```

7. 一旦连接，主设备兼从设备使用 aci_gatt_write_charac_descriptor() API 对二者启用特征通知。 Slave_A 和 Slave_B 设备使用 aci_gatt_upd_char_val() API 启动特征通知。
8. 在该阶段，主设备兼从设备进入发现模式 （作为外设）：

```
/* 使主设备兼从设备进入可发现模式， Name = 'Test' =
  [0x08,0x74,0x65,0x73,0x74*/
ret = aci_gap_set_discoverable(Advertising_Type= 0x00,
                               Advertising_Interval_Min=0x20,
                               Advertising_Interval_Max=0x100,
                               Local_Name_Length=0x05,
                               Local_Name=[0x08,0x74,0x65,0x73,0x74],
                               Slave_Conn_Interval_Min = 0x0006,
                               Slave_Conn_Interval_Max = 0x0008)
```

由于主设备兼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 EVT_BLUE_GATT_NOTIFICATION 事件。

9. 一旦主设备兼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 BlueNRG-MS 设备 （称为主设备）的连接请求。主设备启动发现流程，以便发现主设备兼从设备：

```
ret = aci_gap_start_gen_disc_proc (LE_Scan_Interval=0x10,
                                   LE_Scan_Window=0x10)
```

通过 EVT_LE_ADVERTISING_REPORT 事件发现主设备兼从设备。

10. 一旦发现主设备兼从设备，主设备启动连接流程以便与之连接：

```
/* 主设备连接主设备兼从设备：主设备兼从设备的地址类型和地址已在
EVT_LE_ADVERTISING_REPORT 事件的发现流程中找到 */
ret= aci_gap_create_connection(LE_Scan_Interval=0x0010,
                               LE_Scan_Window=0x0010,
                               Peer_Address_Type= “ 主设备兼从设备地址类型 ”
                               Peer_Address= “ 主设备兼从设备地址 ”
                               Conn_Interval_Min=0x6c,
                               Conn_Interval_Max=0x6c,
                               Conn_Latency=0x00,
                               Supervision_Timeout=0xc80,
                               Minimum_CE_Length=0x000c,
                               Maximum_CE_Length=0x000c)
```

11. 一旦连接，主设备使用 aci_gatt_write_charac_descriptor （) API 在主设备兼从设备上启用特征通知。

12. 在该阶段，由于是 GAP 中央设备，主设备兼从设备接收来自 Slave_A、 Slave_B 设备的特征通知，而作为 GAP 外设，它还能将这些特征值通知主设备。

注： *BlueNRG DK 软件包 （参见“ 参考文献” 部分）中提供了一组测试脚本，用于练习上述 BlueNRG-MS 同时为主、从设备的场景。这些脚本可以使用 BlueNRG GUI 运行，并可作为使用 BlueNRG-MS 同时作为主、从设备功能实现固件应用的参考。*

4 BlueNRG 多连接时序策略

本节提供当多个主设备和从设备连接处于激活状态时连接时序管理策略的概述。

4.1 关于低功耗蓝牙时序的基本概念

本节介绍与广告、扫描和连接操作相关的低功耗蓝牙时序管理的基本概念。

4.1.1 广告时序

广告状态的时序由 3 个时序参数来定义，它们通过以下公式串联：

T_advEvent = advInterval + advDelay

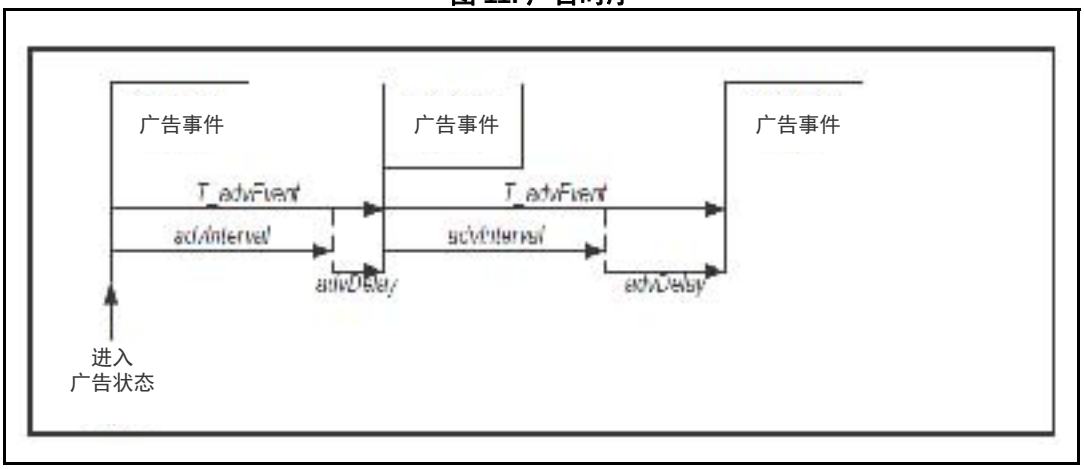
其中：

- T_advEvent：两个连续广告事件的启动时间之间的时间；

  如果广告事件类型为可扫描非定向事件类型或不可连接非定向类型，advInterval 应不小于 100 ms ；

  如果广告事件类型为低占空比模式下使用的可连接非定向事件类型或可连接定向事件类型， advInterval 可大于等于 20 ms。
- advDelay：链路层为每个广告事件生成的虚拟随机值，范围为 0 ms 至 10 ms。

图 11. 广告时序



4.1.2 扫描时序

扫描状态的时序由 2 个时序参数来定义：

- scanInterval：两个连续扫描事件的启动时间之间的间隔；
- scanWindow：链路层在这一时间内监听广告通道索引。

scanWindow 和 scanInterval 参数应小于等于 10.24 s。

scanWindow 应小于等于 scanInterval。

4.1.3 连接时序

连接事件的时序取决于 2 个参数：

- 连接事件间隔 （*connInterval*）：两个连续连接事件的启动时间之间的间隔，绝不应重叠；连接事件启动的时间点被称为*锚点*。

主设备应在锚点开始向从设备发送数据通道 PDU，而从设备应监听其主设备在锚点发送的数据包。

主设备应确保连接事件在下一个连接事件的锚点的至少 T_IFS=150 µs （帧间间隔时间，即相同通道索引上连续数据包之间的时间间隔）前结束。

connInterval 应是 1.25 ms 的倍数，范围为 7.5 ms 至 4.0 s。

- *从设备延迟*（*connSlaveLatency*）：允许从设备使用更少的连接事件。该参数定义了从设备无需监听主设备的连续连接事件数量。

当主机想要创建连接时，它为控制器提供连接间隔 （*Conn_Interval_Min*、*Conn_Interval_Max*）和连接长度 （*Minimum_CE_Length*、 *Maximum_CE_Length*）的最大和最小值，从而在选择实际参数方面给予了控制器一定的灵活性，以便满足其他时序限制，例如有多个连接时。

4.2 BlueNRG 时序和时隙分配概念

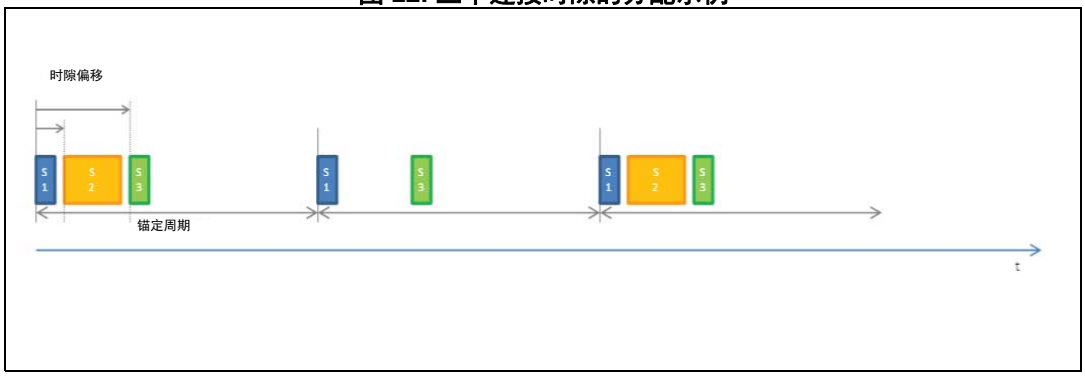
BlueNRG 采用一种时隙机制，以便能够分配同步的主设备和从设备连接。控制时隙机制的基本参数为：

表 55. 时隙算法的时序参数

参数	说明
锚定周期	循环时间间隔，这段时间内可分配最多 8 个连接时隙。 在这 8 个时隙中，一次只有一个是扫描或广告时隙 （互斥）。
时隙持续时间	时间间隔，在此期间发生完整事件 （即广告或扫描及连接）；时隙持续时间是分配给连接时隙的持续时间，与连接事件的最长持续时间有关。
时隙偏移	锚定周期的开始时间与连接时隙的开始时间之间的延时对应的时间值。
时隙延迟	代表连续锚定周期内特定连接时隙的实际利用率的值。 （例如，时隙延迟等于 ‘1’ 表示在每个锚定周期内实际使用了特定连接时隙；时隙延迟等于 n 表示每 n 个锚定周期内只实际使用一次特定连接时隙）

此类时序分配概念允许对多个连接进行明确的时间处理，但同时对控制器可以接受的实际连接参数强加了一些限制。*图 12* 所示为时基参数和连接时隙分配的示例。

图 12. 三个连接时隙的分配示例



时隙 1 就锚定周期而言的偏移为 0，时隙 2 的时隙延迟 = 2，所有时隙相隔 1.25 ms 的保护时间。

4.2.1 为第一个主设备连接设置时序

上述时基机制在创建首个主设备连接时实际启动。此类首个连接的参数决定了锚定周期的初始值，并影响与首个连接同步的任何其他主设备连接可以接受的时序设置。

尤其是：

- 选择的初始锚定周期等于主机请求的最大和最小连接周期的平均值。
- 第一个连接时隙位于锚定周期开始处。
- 第一个连接时隙的持续时间设置为等于请求的连接长度的最大值。

很明显，此类首个连接时隙与锚定周期相比的相对持续时间，限制了为其他主设备连接分配其他连接时隙的可能性。

4.2.2 为其他主设备连接设置时序

在配置并启动了时基（如上文所述）后，时隙分配算法将尝试在一定范围内动态地重新配置时基，以分配其他主机请求。

具体来说，考虑下列三种情况：

1. 当前锚定周期处于为新连接指定的 *Conn_Interval_Min* 至 *Conn_Interval_Max* 的范围内。这种情况下，不能修改时基，将新连接的连接间隔设置为等于当前锚定周期。
2. 当前锚定周期小于新连接所要求的 *Conn_Interval_Min*。这种情况下，算法搜索满足下列条件的整数 *m*：

$$Conn\_Interval\_Min \leq Anchor\_Period \cdot m \leq Conn\_Interval\_Max$$

   如果找到这样的值，则维持当前锚定周期，并将新连接的连接间隔设置为等于 $Anchor\_Period \cdot m$，时隙延迟等于 *m*。
3. 当前锚定周期大于新连接所要求的 *Conn_Interval_Max*。这种情况下，算法搜索满足下列条件的整数 *k*：

$$Conn\_Interval\_Min \leq \frac{Anchor\_Period}{k} \leq Conn\_Interval\_Max$$

   如果找到这样的值，则将当前锚定周期缩短为：

$$\frac{Anchor\_Period}{k}$$

   将新连接的连接间隔设置为：

$$\frac{Anchor\_Period}{k}$$

   将现有连接的时隙延迟乘以因数 *k*。注意，这种情况下还必须满足下列条件：

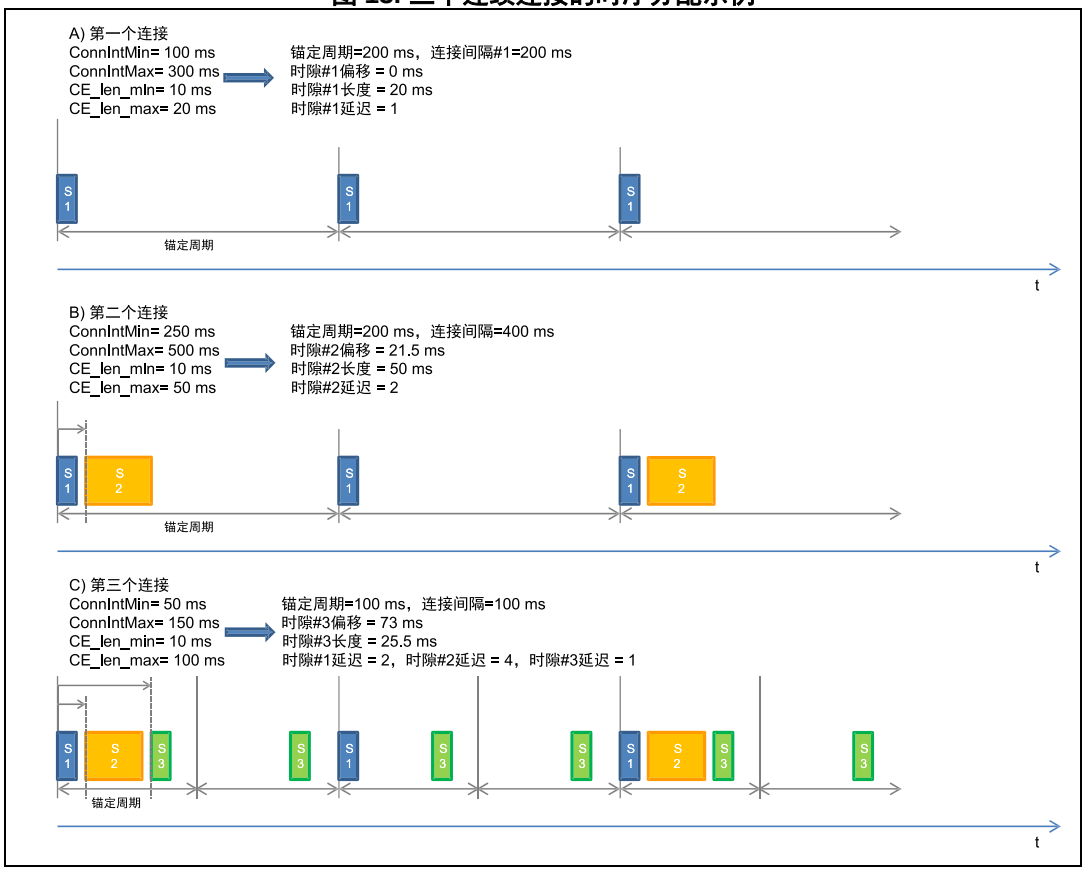
- *Anchor_Period/k* 必须是 1.25 ms 的倍数
- *Anchor_Period/k* 必须足够大，以包含已分配给之前的连接的所有连接时隙

一旦找到符合上述标准的合适锚定周期，在其中为实际连接时隙分配时间间隔。一般而言，如果锚定周期内有足够间隔，算法将分配最大请求连接事件长度，否则将缩短为实际可用间隔。

如果创建了多个连续连接，通常按顺序排列相对连接时隙，中间以很短的保护时间（1.5 ms）隔开；当连接关闭时，通常在两个连接时隙之间产生未使用的间隙。在之后创建新连接时，算法将尝试将新连接时隙安插到现有间隙之一；如果间隙不够宽，则直接将连接时隙排列在最后一个连接时隙之后。

图 13 所示为创建连续连接时如何管理时基参数的示例。

图 13. 三个连续连接的时序分配示例



4.2.3 广告事件时序

广告事件的周期 （由 *advInterval* 控制）是基于下列参数计算的，这些参数由从设备通过主机在 HCI_LE_Set_Advertising_parameters 指令中指定：

- *Advertising_Interval_Min, Advertising_Interval_Max;*
- *Advertising_Type*;

如果将 *Advertising_Type* 设置为高占空比定向广告，则将广告间隔设置为 3.75ms，无论 *Advertising_Interval_Min* 和 *Advertising_Interval_Max* 的值为多少；这种情况下，超时也设置为 1.28 s，即这种情况下广告事件的最长持续时间；

在所有其他情况下，选择的广告间隔等于 （*Advertising_Interval_Min* + 5 ms）和（*Advertising_Interval_Max* + 5ms）的平均值。在前一种情况下，广告没有最长持续时间，只在连接建立时或主机明确请求时停止。

每个广告事件的长度由软件默认设置为等于 14.6ms （即允许的最大广告事件长度），并且不能缩短。

在主设备时隙的相同时基内分配广告时隙 （即扫描和连接时隙）。因此，将由软件在至少一个主设备时隙时接受的广告启用指令处于激活状态，广告间隔必须是实际锚定周期的整数倍。

4.2.4 扫描时序

主设备通过下列参数 （由主机在 HCI_LE_Set_Scan_parameters 指令中指定）请求扫描时序：

– *LE_Scan_Interval*：用于计算扫描时隙的周期。

– *LE_Scan_Window*：用于计算要分配到主设备时基中的扫描时隙的长度。

分配的扫描时隙位于其他激活的主设备时隙 （即连接时隙）和广告时隙 （如果有一个处于激活状态）的相同时基内。

为了让扫描启用指令被软件接受， *LE_Scan_Interval* 必须是实际锚定周期的整数倍。

鉴于扫描参数只是 BT 官方规范提供的建议 （v.4.1 第 2 卷 E 部分第 7.8.10 节），每当 *LE_Scan_Interval* 大于实际锚定周期时，为了维持主机要求的相同占空比，软件自动尝试二次采样 *LE_Scan_Interval* 并缩短分配的扫描时隙长度 （最多达到 *LE_Scan_Window* 的 ¼）。

4.2.5 从设备时序

就上述时基机制而言，从设备时序由主设备在连接创建时定义，因此从设备链路的连接时隙是异步管理的。从设备假设主设备可以使用与连接间隔一样长的连接事件长度。

每当在从设备和主设备时隙之间预测到冲突条件时，时序安排算法采用循环调度仲裁策略。此外，调度器还可以对从设备连接时隙持续时间使用动态限制，以保留主设备和从设备连接。

尤其是：

- 如果主设备连接时隙的末端与从设备连接时隙的起始端重叠，则交替保留 / 取消主设备和从设备连接；
- 如果从设备连接时隙的末端与主设备连接时隙的起始端重叠，则强行限制从设备连接时隙长度以避免此类重叠。如果产生的时间间隔过小而不允许至少两个数据包的交换，则使用循环调度仲裁策略。

4.3 BlueNRG 多个主设备和从设备连接指南

为了使用 BlueNRG 和 BlueNRG-MS 设备正确处理多个主设备和从设备连接，应遵循下列指南：

1. 避免分配过长的连接事件长度：选择尽可能小的 *Minimum_CE_Length* 和 *Maximum_CE_Length* 以严格满足应用需求。这样做有助于分配算法在锚定周期内分配多个连接并缩短锚定周期，如果需要，为连接分配较小的连接间隔。
2. 对于第一个主设备连接：
   a) 如可能，创建具有最短连接间隔的连接作为第一个连接。这样将有助于为其他连接分配为初始锚定周期倍数的连接周期。
   b) 如可能，为 *Conn_Interval_Min* = *Conn_Interval_Max* 选择 10 *ms* 的倍数。这样将有助于为其他连接分配既是初始锚定周期的 2、 4 和 8 （或更多）倍的约数又是 1.25 *ms* 的倍数的连接间隔。
3. 对于其他主设备连接：
   a) 使 *ScanInterval* 等于现有主设备连接之一的连接间隔
   b) 选择 *ScanWin*，使已分配主设备时隙之和 （包括广告 （如激活））小于最短已分配连接间隔
   c) 选择 *Conn_Interval_Min* 和 *Conn_Interval_Max*，使间隔包含：
   – 最短已分配连接间隔的倍数
   – 最短已分配连接间隔的因数也是 1,25 *ms* 的倍数
   d) 选择 *Maximum_CE_Length* = *Minimum_CE_Length*，使已分配主设备时隙之和 （包括广告 （如激活））加上 *Minimum_CE_Length* 后小于最短已分配连接间隔
4. 每次开始广告时：
   a) 如果是定向广告，选择 *Advertising_Interval_Min* = *Advertising_Interval_Max* = 最短已分配连接间隔的整数倍
   b) 如果是定向广告，选择 *Advertising_Interval_Min* = *Advertising_Interval_Max*，使（*Advertising_Interval_Min* + 5ms）等于最短已分配连接间隔的整数倍
5. 每次开始扫描时：
   a) 使 *ScanInterval* 等于现有主设备连接之一的连接间隔
   b) 选择 *ScanWin*，使已分配主设备时隙之和 （包括广告 （如激活））小于最短已分配连接间隔
6. 注意，创建多个连接然后关闭其中一些连接并重新创建新连接的过程，随着时间的推移，会降低时隙分配算法的总体效率。如果在分配新连接时遇到困难，可将时基重置为初始状态 （将关闭所有现有连接）。

5 参考文献

表 56. 参考文献表

名称	标题
AN4494	启动 BlueNRG 应用笔记
BlueNRG 数据手册	Bluetooth® 低能耗无线网络处理器
BlueNRG-MS 数据手册	Bluetooth® 低能耗无线网络处理器
BlueNRG 开发套件软件包	BlueNRG 和 BlueNRG-MS 套件的 BlueNRG 软件包
蓝牙规范 V4.1	蓝牙系统规范 v4.0
蓝牙规范 V4.1	蓝牙系统规范 v4.1
UM1755	BlueNRG 蓝牙 LE 栈应用指令接口 （ACI）用户手册
UM1865	BlueNRG-MS 蓝牙 LE 栈应用指令 接口 （ACI）用户手册
UM1686	BlueNRG 开发套件用户手册
UM1870	BlueNRG-MS 开发套件用户手册

附录 A 缩写和缩略语列表

本附录列出了文档中使用的标准缩写和缩略语。

表 57. 缩略语列表

术语	意义
ACI	应用命令接口
ATT	属性协议
BLE	低功耗蓝牙
BR	基础速率
CRC	循环冗余校验
CSRK	连接签名解析密钥
EDR	增强数据率
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
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
RF	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
USB	通用串行总线

表 57. 缩略语列表 （续）

术语	意义
UUID	通用唯一标识符
WPAN	无线个人局域网

6 版本历史

表 58. 文档版本历史

日期	版本	变更
2015 年 1 月 23 日	1	初始版本。
2015 年 4 月 21 日	2	将本文修改为同时适合 BlueNRG 和 BlueNRG-MS 两种器件。
2015 年 9 月 02 日	3	在*第 2 章：BlueNRG、 BlueNRG-MS 栈架构和 ACI* 中增加了对 STM32L Cube 库的引用 增加了*第 4 章：BlueNRG 多连接时序策略*。
2015 年 12 月 18 日	4	更新了*表 35：BlueNRG-MS GATT、GAP 默认特征*。增加了新章节：*第 3.6 章：私有*。

表 59. 中文文档版本历史

日期	版本	变更
2016 年 12 月 8 日	1	中文初始版本。

重要通知 - 请仔细阅读

意法半导体公司及其子公司 （“ST”）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 ST 产品的最新信息。 ST 产品的销售依照订单确认时的相关 ST 销售条款。

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

ST 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 ST 产品如有不同于此处提供的信息的规定，将导致 ST 针对该产品授予的任何保证失效。

ST 和 ST 徽标是 ST 的商标。所有其他产品或服务名称均为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。