CC3100/CC3200 SimpleLink™ Wi-Fi® 片上互联网解决方案

User’s Guide

Literature Number: ZHCU938C
MAY 2018 – REVISED JANUARY 2021

目录

插图清单

表格清单

章节 1

概述

1.1 文档范围

本文档旨在为使用 Wi-Fi® 子系统的软件编程人员提供有关其网络功能，以及如何通过主机驱动程序使用这些功能所需的全部知识。这包括有关如何编写联网应用的概述、整个器件的 API 联网操作模式和功能的详细说明，以及对每个驱动程序 API 的回顾，并在各部分随附了源代码示例。

1.2 概述

SimpleLink™ Wi-Fi® CC3100 和 CC3200 是新一代的嵌入式 Wi-Fi 解决方案。CC3100 片上互联网™ 解决方案可以将 Wi-Fi 和互联网功能添加至任何微控制器 (MCU)，如 TI 的超低功耗 MSP430™。CC3200 是一款可编程的 Wi-Fi MCU，可实现真正的集成物联网 (IoT) 开发。这两款 SimpleLink Wi-Fi 器件中的 Wi-Fi 网络处理器子系统集成了 Wi-Fi 和互联网的所有协议，大大降低了 MCU 软件要求。借助内置安全协议，SimpleLink Wi-Fi 可提供简单但可靠的安全体验。

1.2.1 特性列表

表 1-1. 特性列表

Wi-Fi	
支持的通道	1-13
支持的协议	WEP、WPA 和 WPA2
个人安全	WPA-2 企业级
企业安全	EAP Fast、EAP PEAPv0 MSCHAPv2、 EAP PEAPv0 TLS、EAP PEAPv1 TLS、EAP TLS EAP TTLS TLS、EAP TTLS MSCHAPv2
配置	SmartConfig™ 技术
	Wi-Fi 保护设置 (WPS2)
	具有内部 HTTP Web 服务器的接入点模式
标准	802.11b/g 接入点和 Wi-Fi Direct 组所有者
客户端	1
个人安全	WEP、WPA 和 WPA2
网络	
IP	IPv4
传输	UDP RAW ICMP
跨层	DHCP ARP DNS

表 1-1. 特性列表 (continued)

Wi-Fi	
应用	mDNS DNS-SD HTTP 1.0 Web 服务器
传输层安全	SSLV3 SSL_RSA_WITH_RC4_128_SHA SSLV3 SSL_RSA_WITH_RC4_128_MD5 TLSV1 TLS_RSA_WITH_RC4_128_SHA TLSV1 TLS_RSA_WITH_RC4_128_MD5 TLSV1 TLS_RSA_WITH_AES_256_CBC_SHA TLSV1 TLS_DHE_RSA_WITH_AES_256_CBC_SHA TLSV1 TLS_ECDHE_RSA_WITH_RC4_128_SHA TLSV1 TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA TLSV1_1 TLS_RSA_WITH_RC4_128_SHA TLSV1_1 TLS_RSA_WITH_RC4_128_MD5 TLSV1_1 TLS_RSA_WITH_AES_256_CBC_SHA TLSV1_1 TLS_DHE_RSA_WITH_AES_256_CBC_SHA TLSV1_1 TLS_ECDHE_RSA_WITH_RC4_128_SHA TLSV1_1 TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA TLSV1_2 TLS_RSA_WITH_RC4_128_SHA TLSV1_2 TLS_RSA_WITH_RC4_128_MD5 TLSV1_2 TLS_RSA_WITH_AES_256_CBC_SHA TLSV1_2 TLS_DHE_RSA_WITH_AES_256_CBC_SHA TLSV1_2 TLS_ECDHE_RSA_WITH_RC4_128_SHA TLSV1_2 TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
用户应用套接字	多达 8 个开放的套接字 最多 2 个安全应用套接字： • 一台服务器（监听套接字和接受套接字）+ 客户端（数据套接字） • 最多两个客户端（数据套接字）
高级特性	
802.11 收发器	发送和接收原始 Wi-Fi 数据包，完全控制有效载荷。Wi-Fi 断开连接模式。
	可用于通用型应用（标签、监听器、RF 测试）
接口	
SPI	量产器件的标准 SPI 高达 20MHz，预量产器件则高达 14MHz
UART	4 线 UART，高达 3MHz
功耗模式	
低功耗模式	使用 802.11 省电和器件深度睡眠功耗，包括三个用户可配置的策略
可配置电源策略	• 正常（默认）– 在流量传递时间和电源性能之间进行了合理权衡 • 低功耗 – 仅用于收发器模式应用（断开连接模式） • 长睡眠间隔 – 在可配置的睡眠间隔后为下一个 DTIM 唤醒，最长 2 秒。此策略仅适用于客户端套接字模式。

1.3 主机驱动程序概述

SimpleLink Wi-Fi 片上互联网器件提供全面的网络功能。为了简化使用 SimpleLink Wi-Fi 器件的网络应用的集成和开发，TI 提供了一个简单的用户友好型主机驱动程序。

SimpleLink 主机驱动程序的功能如下：

- 为用户应用提供一个简单 API
- 处理与器件的通信，包括：
  - 构建和解析命令
  - 处理异步事件
  - 处理数据路径的流控制
  - 并发命令的串行化
- 使用现有的 UART 或 SPI 物理接口驱动程序
- 使用操作系统适配层，可灵活地选择使用或不使用操作系统
- 支持移植到任何平台

主机驱动程序采用严格的 ANSI C89 编写，可与大多数嵌入式平台和开发环境完全兼容。

主机驱动程序的关键架构概念如下：

- 微控制器
  - 可在 8 位、16 位或 32 位微控制器上运行
  - 可在任何时钟速度下运行，没有性能或时间依赖性
  - 支持大端和小端字节序格式

标准接口通信端口：

- SPI – 支持标准 4 线 SPI：
  - 8 位、16 位或 32 位字长
  - 默认模式 0（CPOL=0，CPHA=0）
  - SPI 时钟速率可配置为高达 20Mbps。
  - 需要 CS。
  - 异步操作需要额外的 IRQ 线。
- UART
  - 具有硬件流控制 (RTS/CTS) 的标准 UART，速率高达 3Mbps。
  - 默认波特率为 115200（8 位，无奇偶校验，1 个开始/停止位）。
- 支持系统使用或不使用操作系统：
  - 简单的操作系统包装器，只需两个对象包装器：
    - Sync Obj（事件/二进制信标）
    - Lock Obj（互斥量/二进制信标）
  - 驱动程序的内置逻辑，用于不运行操作系统的系统

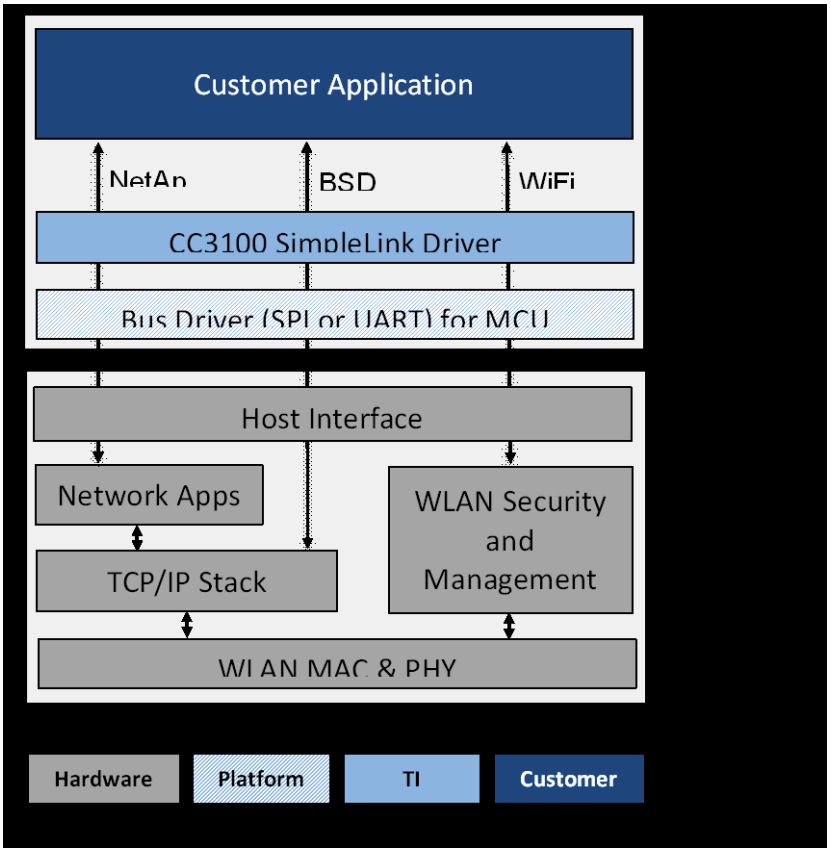


图 1-1. 主机驱动程序架构

SimpleLink 主机驱动程序包括一组六个逻辑模块和一个简单 API 模块：

- 器件 API – 管理硬件相关功能，如启动、停止、设置和获取器件配置。
- WLAN API – 管理 WLAN、802.11 协议相关功能，如器件模式 (工作站、AP 或 P2P) 、设置配置方法、添加连接配置文件和设置连接策略。
- 套接字 API – 用户应用最常见的 API 集，遵循 BSD 套接字 API。
- NetApp API – 启用不同的网络服务，包括超文本传输协议 (HTTP) 服务器服务、DHCP 服务器服务和 MDNS 客户端\服务器服务。
- NetCfg API – 配置不同的网络参数，如设置 MAC 地址、通过 DHCP 获取 IP 地址以及设置静态 IP 地址。
- 文件系统 API – 提供对串行闪存组件的访问，以进行针对网络或用户专有数据的读取和写入操作。

图 1-2 是主机驱动程序解剖图。

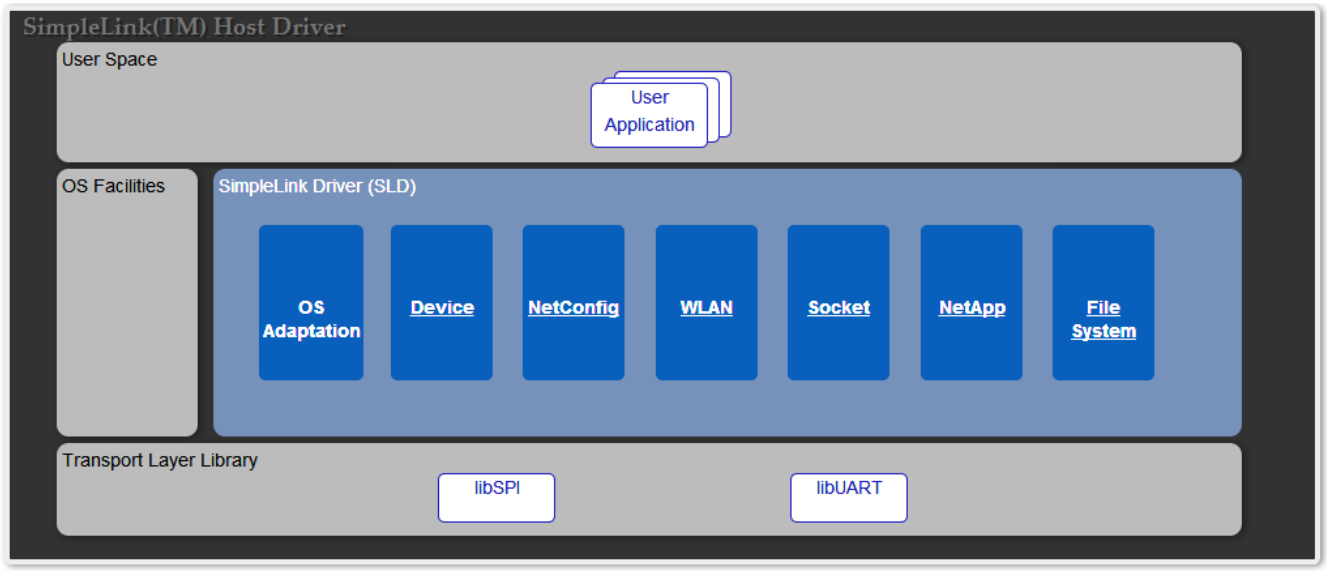


图 1-2. 主机驱动程序解剖图

1.4 可配置的信息元素

表 1-2. 本文档中使用的首字母缩写词

首字母缩写词	说明/备注
AP	WLAN 接入点
DEV	器件角色
FW	固件
LAN	局域网
STA	WLAN 工作站
WLAN	无线 LAN
信息元素	信息元素
IE	信息元素
OUI	组织唯一 ID
NWP	网络处理器
P2P	对等
GO	组所有者

编写一个简单的联网应用程序

2.1 概述

本章将介绍构建联网应用时所需的软件块。此外，本章还将介绍推荐用于大多数应用的流程。提供的信息仅供参考。程序员在如何使用各种软件块方面具有充分的灵活性。使用 SimpleLink 器件的程序由以下软件块组成：

- Wi-Fi 子系统初始化 – 将 Wi-Fi 子系统从休眠状态唤醒。
- 配置 – 指不经常发生的初始化时配置，例如将 Wi-Fi 子系统从 WLAN STA 更改为 WLAN 软 AP、更改 MAC 地址，等等。
- WLAN 连接 – 必须建立物理接口。有很多方法可以实现此目的；较简单的方法是手动连接到作为无线工作站的 AP。
  - DHCP – 虽然不是 WLAN 连接中必不可少的操作，但用户必须等待接收 IP 地址，然后才能继续使用 TCP 和 UDP 套接字执行下一步。
- 套接字连接 – 此时，应用必须设置 TCP/IP 层。此阶段分为以下几个部分：
  - 创建套接字 – 选择 TCP、UDP 或 RAW 套接字，是使用客户端套接字还是服务器套接字，定义阻塞/非阻塞、套接字超时等套接字特征。
  - 查询服务器 IP 地址 – 大多数情况下，在实现客户端通信时，远程服务器端 IP 地址是未知的，但建立套接字连接则需要此信息。为此，可以通过 DNS 协议并使用服务器名称来查询服务器 IP 地址。
  - 创建套接字连接 – 使用 TCP 套接字时，必须在执行数据事务之前建立正确的套接字连接。
- 数据事务 – 建立套接字连接之后，应通过实现应用逻辑在客户端和服务器之间双向传输数据。
- 套接字断开连接 – 完成所需的数据事务后，TI 建议正常关闭套接字通信通道。
- Wi-Fi 子系统休眠 – 如果长时间不使用 Wi-Fi 子系统，TI 建议将其置于休眠模式。

2.1.1 基本示例代码

实施联网应用时，需要考虑不同的应用块、上述主机驱动程序软件概念以及硬件和操作系统等系统。

图 2-1 展示了说明基本软件设计的状态机图。

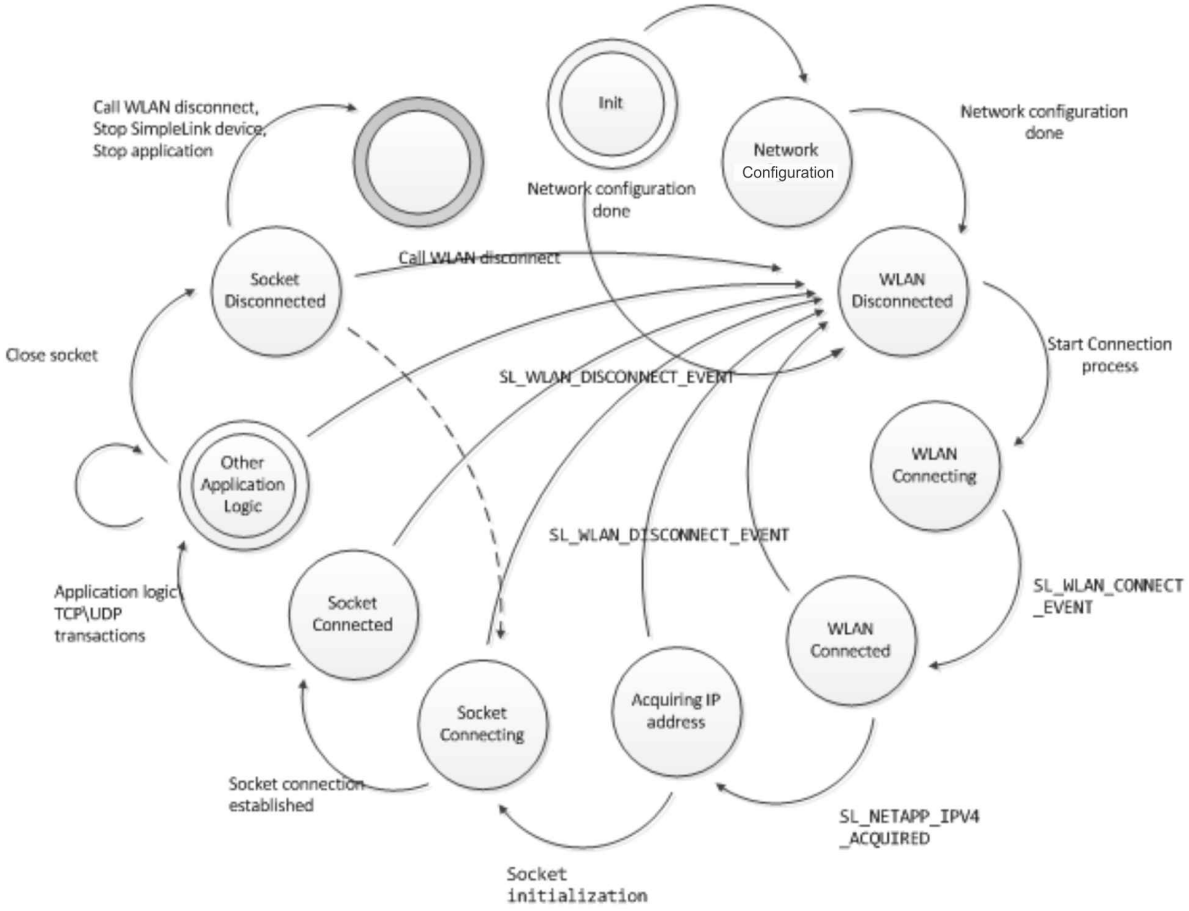


图 2-1. 基本联网应用状态机

图 2-1 展示了本章所述的不同状态、触发代码以在不同状态之间切换的主机驱动程序事件，以及基本错误处理事件。

下面是一个使用代码实现状态机的示例：

- **初始状态** – 下面是一个将 Wi-Fi 子系统初始化为 WLAN 工作站的示例：

```
case INIT:
status = sl_Start(0, 0, 0);
if (status == ROLE_STA)
{
g_State = CONFIG;
}
else
{
g_State = SIMPLELINK_ERR;
}
break;
```

- **WLAN 连接** – 下面是 WLAN 和网络事件处理程序示例，展示了 WLAN 连接、等待成功连接和获取 IP 地址：

```
/* SimpleLink WLAN 事件处理程序 */
void SimpleLinkWlanEventHandler(void *pWlanEvents)
{
SlWlanEvent_t *pWlan = (SlWlanEvent_t *)pWlanEvents;
    switch(pWlan->Event)
    {
        case SL_WLAN_CONNECT_EVENT:
g_Event |= EVENT_CONNECTED;
memcpy(g_AP_Name, pWlan->EventData.STAandP2PModeWlanConnected.ssid_name, pWlan-
>EventData.STAandP2PModeWlanConnected.ssid_len);
        break;
        case SL_WLAN_DISCONNECT_EVENT:
g_DisconnectionCntr++;
      g_Event |= EVENT_DISCONNECTED;
g_DisconnectionReason = pWlan->EventData.STAandP2PModeDisconnected.reason_code;
memcpy(g_AP_Name, pWlan->EventData.STAandP2PModeWlanConnected.ssid_name, pWlan-
>EventData.STAandP2PModeWlanConnected.ssid_len);
        break;
        default:
        break;
    }
}
/* SimpleLink 网络事件处理程序 */
void SimpleLinkNetAppEventHandler(void *pNetAppEvent)
{
    SlNetAppEvent_t *pNetApp = (SlNetAppEvent_t *)pNetAppEvent;
    switch( pNetApp->Event )
    {
        case SL_NETAPP_IPV4_ACQUIRED:
            g_Event |= EVENT_IP_ACQUIRED;
g_Station_Ip = pNetApp->EventData.ipAcquiredV4.ip;
g_GW_Ip =  pNetApp->EventData.ipAcquiredV4.gateway;
g_DNS_Ip = pNetApp->EventData.ipAcquiredV4.dns;
break;
        default:
        break;
    }
}
.
.
.
/* 启动 WLAN 连接 */
case WLAN_CONNECTION:
status = sl_WlanConnect(User.SSID,strlen(User.SSID),0,
&secParams, 0);
if (status == 0)
{
g_State = WLAN_CONNECTING;
}
else
{
g_State = SIMPLELINK_ERR;
}
/* 等待 SL_WLAN_CONNECT_EVENT 通知连接成功 */
case WLAN_CONNECTING:
if (g_Event
&EVENT_CONNECTED)
{
printf("Connected to %s\n", g_AP_Name);
g_State = WLAN_CONNECTED;
}
break;
/* 等待 SL_NETAPP_IPV4_ACQUIRED 通知接收 IP 地址 */
case WLAN_CONNECTED:
if (g_Event
&EVENT_IP_ACQUIRED)
{
printf("Received IP address:%d.%d.%d.%d\n", (g_Station_Ip>>24)&0xFF,(g_Station_Ip>>16)&0xFF,
(g_Station_Ip>>8)&0xFF,(g_Station_Ip&0xFF));
g_State = GET_SERVER_ADDR;
}
break;
```

• **套接字连接** – 下面是使用服务器名称查询远程服务器 IP 地址、创建 TCP 套接字和连接至远程服务器套接字的示例：

```
case GET_SERVER_ADDR:
 status = sl_NetAppDnsGetHostByName(appData.HostName,
                                    strlen(appData.HostName),

&appData.DestinationIP, SL_AF_INET);
if (status == 0)
{
g_State = SOCKET_CONNECTION;
}
else
{
printf("Unable to reach Host\n");
g_State = SIMPLELINK_ERR;
}
break;
case SOCKET_CONNECTION:
Addr.sin_family = SL_AF_INET;
Addr.sin_port = sl_Htons(80);
/* 将 DestinationIP 字节序更改为大端字节序 */
Addr.sin_addr.s_addr = sl_Htonl(appData.DestinationIP);
AddrSize = sizeof(SlSockAddrIn_t);
SockId = sl_Socket(SL_AF_INET,SL_SOCK_STREAM, 0);
if( SockId < 0 )
{
printf("Error creating socket\n\r");
status = SockId;
g_State = SIMPLELINK_ERR;
}
if (SockId >= 0)
{
status = sl_Connect(SockId, ( SlSockAddr_t *)
&Addr, AddrSize);
if( status >= 0 )
{
g_State = SOCKET_CONNECTED;
}
else
{
printf("Error connecting to socket\n\r");
g_State = SIMPLELINK_ERR;
}
}
break;
```

- **数据事务** – 下面是通过开放的套接字发送和接收 TCP 数据的示例：

```
case SOCKET_CONNECTED:
     /* 将数据发送至远程服务器 */
     sl_Send(appData.SockID, appData.SendBuff, strlen(appData.SendBuff), 0);
    /* 接收远程服务器发出的数据 */
    sl_Recv(appData.SockID, &appData.Recvbuff[0], MAX_RECV_BUFF_SIZE, 0);

break;
```

- **套接字断开连接** – 下面是关闭套接字的示例：

```
case SOCKET_DISCONNECT:
    sl_Close(appData.SockID);
    /* 重新打开套接字 */
    g_State = SOCKET_CONNECTION;
    break;
```

- **器件休眠** – 下面是让 Wi-Fi 子系统进入休眠状态的示例：

```
case SIMPLELINK_HIBERNATE:
    sl_Stop();
    g_State = …
    break;
```

章节 3

器件初始化

3.1 概述

调用 *sl_Start()* API 即可启用 Wi-Fi 子系统。在初始化期间，主机驱动程序执行以下关键步骤：

- 启用总线接口（在 CC3200 中为 SPI；在 CC3100 中为 SPI 或 UART）
- 注册异步事件处理程序
- 启用 Wi-Fi 子系统（在 CC3200 中，这是由内部应用微控制器完成的。在 CC3100 中，这是由主机处理器完成的）
- 向 Wi-Fi 子系统发送同步消息并等待 IRQ 响应，在初始化阶段完成时发出信号。

图 3-1 展示了基本的初始化流程：

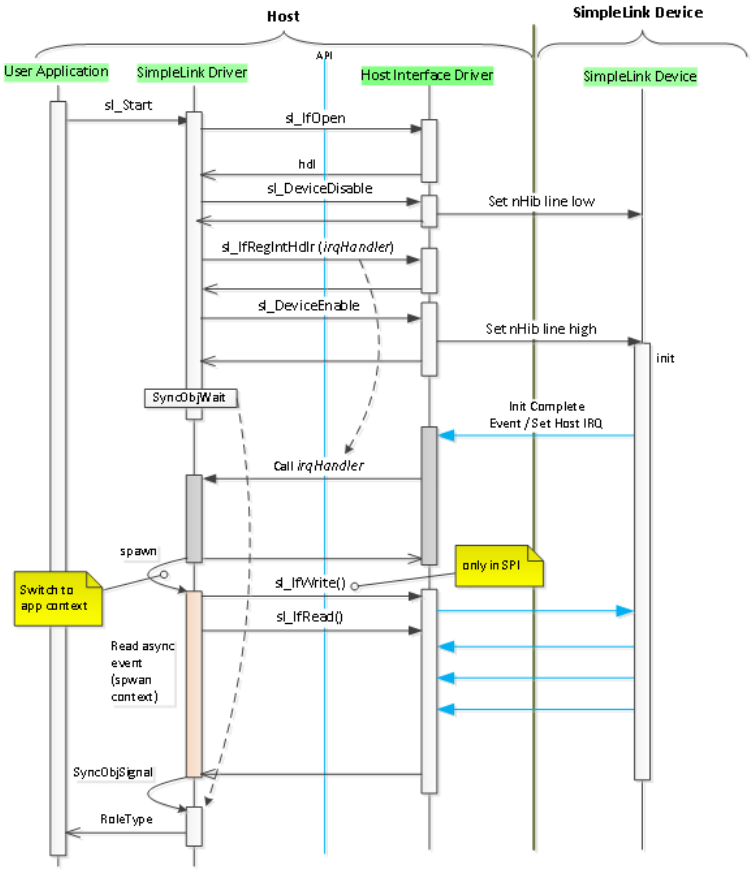


图 3-1. 基本初始化流程

Wi-Fi 子系统的初始化只需数十毫秒即可完成。主机驱动程序支持使用“sl_Start(const void* pIfHdl, char* pDevName and const P_INIT_CALLBACK pInitCallBack)”API 进行初始化的两个主要选项：

- 阻止 – 必须将 pInitCallBack 设置为 NULL。在整个初始化过程完成（在接收到初始化完成的中断信号时）之前会一直对调用应用程序进行阻止。请参阅以下代码示例：
  - ```
    if( sl_Start(NULL, NULL, NULL) == 0)
    {
            LOG("Error opening interface to device\n");
    }
    ```
    ```

- 异步 – 为 pInitCallBack 赋予一个指针，指向在初始化过程完成时调用的函数。在此情况中，对 *sl_Start()* 的调用会立即返回结果。请参阅以下代码示例：
  -
```
Void InitCallBack(UINT32 Status)
{
  Network_IF_SetMCUMachineState(MCU_SLHost_INIT);
}
.
.
Void Network_IF_InitDriver(void)
{
   ..
  sl_Start(NULL,NULL,InitCallBack);
  while(!(g_usMCUstate & MCU_SLHost_INIT));
  ..
}
```

3.2 主机接口

SimpleLink Wi-Fi 器件提供全面的网络功能。为了简化网络应用的集成和开发，我们提供了一个更简单的用户友好型主机驱动程序。SimpleLink Wi-Fi 主机驱动程序的功能如下：

- 为用户应用提供一个简单 API
- 处理与网络处理器的通信
- 构建和解析命令
- 处理异步事件
- 处理数据路径的流控制
- 提供并发命令的串行化
- 使用现有的 UART 或 SPI 物理通信接口驱动程序
- 提供使用或不使用操作系统的能力
- 支持移植

SimpleLink Wi-Fi 主机驱动程序采用严格的 ANSI C89 编写，可与大多数嵌入式平台和开发环境完全兼容。

以下信息与 SimpleLink Wi-Fi CC31xx 无线网络处理器有关，该处理器必须具有与选定 MCU 进行通信的接口。

该器件支持 SPI 和 UART 标准通信接口。通过 user.h 中的以下定义来定义接口函数，将通信接口绑定到主机驱动程序：

- sl_IfOpen
- sl_IfClose
- sl_IfRead
- sl_IfWrite
- sl_IfRegIntHdlr

3.2.1 SPI 接口

SimpleLink Wi-Fi CC3100 器件作为 SPI 从器件运行，并支持 4 线 SPI 接口。表 3-1 列出了所需的 SPI 设置。

表 3-1. SPI 配置

属性	值
时钟速率	高达 20MHz
字长	8 位、16 位、32 位
模式	0（CPOL=0、CPHA=0）
其他	需要 CS，并且无法绑定到活动状态 需要其他 IRQ 线路以指示器件生成的异步事件

在 SPI 中，总线上的所有通信都由 SPI 主器件（在本例中为主机）发起。总线上始终只有一个主器件。为了允许 SimpleLink Wi-Fi 器件向主机触发异步事件，它们之间必须连接一个额外的 I/O (H.IRQ)。此线路将触发主机从器件读取消息。

一个采用 SPI 接口的 CC31xx 无线网络处理器的典型主机设置如图 3-2 所示。

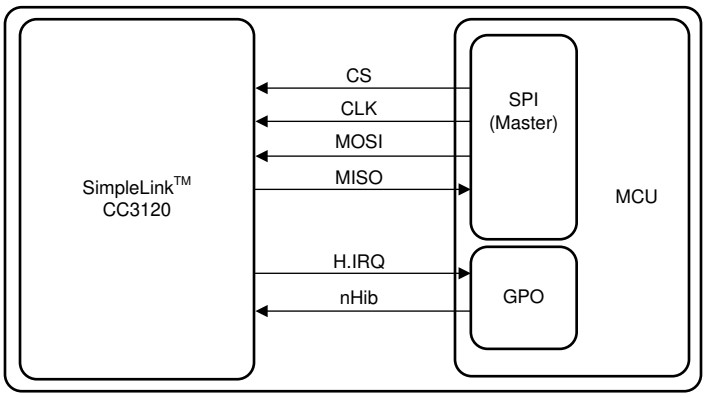


图 3-2. 典型的 CC31xx 设置 (SPI)

有关 CC3100 SPI 接口的更多详细信息，请参阅*适用于 MCU 应用的 CC3100 SimpleLink™ Wi-Fi® 网络处理器、物联网解决方案数据表*。

3.2.2 UART 接口

SimpleLink Wi-Fi CC3100 器件支持具有硬件流控制 (RTS/CTS) 功能的标准 UART 接口。默认波特率为 115,200bps，可增加至 3Mbps。表 3-2 列出了所需的 UART 设置。

表 3-2. UART 设置

属性	值
波特率	115,200bps （可增加至 3Mbps）
流控制	CTS/RTS
奇偶校验	无
数据位数	8
停止位	1

采用 UART 接口的 SimpleLink Wi-Fi 器件的典型主机设置如图 3-3 所示。

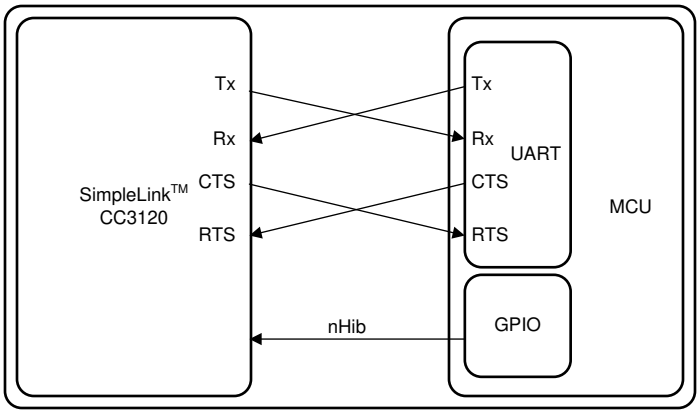


图 3-3. 典型的 CC31xx 设置 (UART)

使用 UART 接口时需要使用硬件流控制功能来避免数据丢失。UART 硬件流控制生效的原因是准备好接受数据的实体保持其 RTS 线路处于有效状态。在传输之前，第二侧的 UART 外设会读取其连接到第一侧 RTS 的 CTS 线路，以验证是否允许通过该线路发送数据。SimpleLink Wi-Fi 器件在某些情况下可能会请求停止传输；因此，必须遵循其 RTS 线路的要求。如果主机速度足够快并且不需要随时停止来自 SimpleLink 器件的传输，则 SimpleLink Wi-Fi 器件的 CTS 线路可能会连接到上拉电阻。

仅针对 UART 模式，应在 user.h 中添加以下定义：#define SL_IF_TYPE_UART

有关 CC3100 UART 接口的更多详细信息，请参阅*适用于 MCU 应用的 CC3100 SimpleLink™ Wi-Fi® 网络处理器、物联网解决方案数据表*。

3.2.2.1 更改 UART 波特率

SimpleLink 器件不支持自动波特率检测；因此，在每次复位后应设置此参数。调用 sl_start 时，作为 API 参数的一部分，必须设置默认波特率 (115,200)。如果需要另一个波特率，主机可以在初始化过程完成之后使用 API sl_DeviceUartSetMode 来设置它。此设置不是持久的，每次调用 sl_Start 时都必须重复设置。

支持的波特率：

- SL_DEVICE_BAUD_9600
- SL_DEVICE_BAUD_14400
- SL_DEVICE_BAUD_19200
- SL_DEVICE_BAUD_38400
- SL_DEVICE_BAUD_57600
- SL_DEVICE_BAUD_115200
- SL_DEVICE_BAUD_230400
- SL_DEVICE_BAUD_460800
- SL_DEVICE_BAUD_921600

示例：

```
_i16 Status;
_i16 Role;
SlDeviceUartIfParams_t params;
#define COMM_PORT_NUM 24 /* uart com port number */
params.BaudRate = SL_DEVICE_BAUD_115200; /*set default baud rate */
params.FlowControlEnable = 1;
params.CommPort = COMM_PORT_NUM;
Role = sl_Start(NULL, (signed char*)&params, NULL)
params.BaudRate = SL_DEVICE_BAUD_921600; /* set default baud rate 921600 */
Status = sl_DeviceUartSetMode((signed char*)&params);
if( Status )
{
    /* error */
}
```

器件配置

4.1 概述

本章涵盖了所有用户可配置的参数，以及用于反映器件和网络状态的只读参数（只读参数在本文档中会进行相应标注）。

Wi-Fi 子系统具有几个可配置的参数用于控制其行为。主机驱动程序使用不同的 API 来配置这些参数。这些参数按照其功能分为不同的组。

本章中介绍的大多数参数都存储在串行闪存中（易失性的以及未存储在串行闪存中的参数在本文档中会进行相应标注）。

此外，每个参数都有默认值。如果未设置参数的值，则互联网 Wi-Fi 子系统会使用默认值。存储在串行闪存中的值始终优先于默认值。

通常，应用程序只能在冷启动后或需要更改特定配置时配置一次参数。

备注

串行闪存中存储的所有参数将在下次器件引导时生效。

4.2 器件参数

时间和日期：配置器件的内部日期和时间。如需了解更多详情，请参阅节 17.1。

固件版本：一个只读参数，返回 Wi-Fi 子系统固件版本。如需了解更多详情，请参阅节 17.1。

器件状态：返回在互联网子系统中保持一致的上次事件的状态。如需了解更多详情，请参阅节 17.1。

异步事件屏蔽：屏蔽的事件不会从 Wi-Fi 子系统生成异步消息 (IRQ)。更多详细信息，请查看节 17.1 中的 *sl_EventMaskGet* 和 *sl_EventMaskSet*。

UART 配置：使用 UART 接口时，应用程序可以设置几个 UART 参数：波特率、流控制和 COM 端口。如需了解更多详情，请参阅节 17.1。

备注

- 部分易失性参数 – 在休眠模式下保留，但在关机模式下复位的参数
- 只读参数

4.3 WLAN 参数

器件模式 – Wi-Fi 子系统能够以多种 WLAN 角色运行。不同的选项包括：

- WLAN 工作站
- WLAN AP
- WLAN P2P

如需了解更多详情，请参阅节 17.3。

AP 模式 – 如果设置为接入点角色，则 Wi-Fi 子系统具有许多可以设置的 AP 模式特定配置：

- SSID – AP 名称
- 国家/地区代码
- 信标间隔
- 运行通道
- 隐藏 SSID – 启用或禁用
- DTIM 周期

- 安全类型 – 可能的选项为：
  - 开放安全
  - WEP 安全
  - WPA 安全
- 安全密码：
  - 对于 WPA：8 至 63 个字符
  - 对于 WEP：5 至 13 个字符 (ASCII)
- WPS 状态

如需了解更多详情，请参阅节 7.4。

P2P – 如果设置为对等角色，则 Wi-Fi 子系统具有许多可以设置的 P2P 模式特定配置：

- 器件名称
- 器件类型
- 运行通道 – 侦听通道和监管等级确定了 p2p 查找侦听阶段的器件侦听通道。
  运行通道和监管等级确定了此器件更适合的运行通道（如果此器件是组所有者，则会使用该运行通道）。通道应该是社会通道之一 (1/6/11)。如果未选择侦听通道或运行通道，则将随机选择 1/6/11。
- 信息元素 – 应用程序可以为每个角色 (AP/P2P GO) 设置 MAX_PRIVATE_INFO_ELEMENTS_SUPPORTED 个信息元素。若要删除信息元素，请使用相关索引和长度 = 0。
  应用程序可以为同一角色设置 MAX_PRIVATE_INFO_ELEMENTS_SUPPORTED。但是，对于 AP，可以为所有信息元素存储不超过 INFO_ELEMENT_MAX_TOTAL_LENGTH_AP 字节。
  对于 P2P GO，可以为所有信息元素存储不超过 INFO_ELEMENT_MAX_TOTAL_LENGTH_P2P_GO 字节。
- 扫描通道 – 更改扫描通道和 RSSI 阈值。

如需了解更多详情，请参阅章节 10。

4.3.1 高级

国家/地区代码 – 设置 Wi-Fi 子系统监管域国家/地区代码。仅与 WLAN 工作站和 P2P 客户端模式相关。支持的国家/地区代码为：US、JP 和 EU。如需了解更多详情，请参阅节 17.3。

TX 功率 – 设置 Wi-Fi 子系统的最大发射功率。如需了解更多详情，请参阅节 17.3。

在图 4-1 中可以看到 TX 功率级别与实际输出功率（以 dBm 为单位）之间的关系。

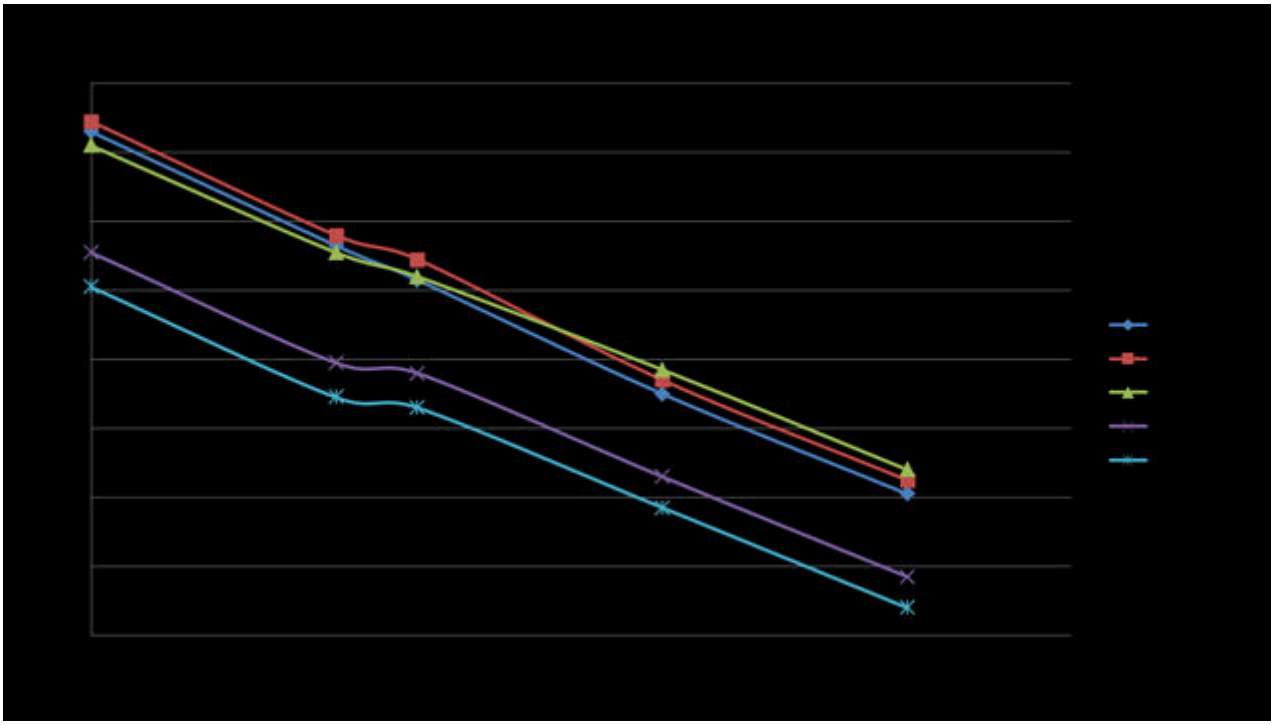


图 4-1. TX 输出功率与 TX 级别

4.4 网络参数

MAC 地址 – 设置器件的 MAC 地址。如需了解更多详情，请参阅节 17.2。

IP 地址 – 将 Wi-Fi 子系统配置为使用 DHCP 或静态 IP 配置。如果是静态配置，用户可以设置 IP 地址、DNS 地址、GW 地址和子网掩码。如需了解更多详情，请参阅节 17.2。

4.5 互联网和网络服务参数

HTTP 服务器：如需了解更多详情，请参阅章节 11。

DHCP 服务器：如需了解更多详情，请参阅节 9.6。

mDNS：如需了解更多详情，请参阅章节 12。

SmartConfig：如需了解更多详情，请参阅节 7.2。

4.6 电源管理参数

4.6.1 概述

电源管理和能量保存是极其麻烦的问题，也是在线系统的主要关注点。有效处理电源工作方式是任何功率感知解决方案的基础。如果整体解决方案的某个组件比系统的其他部分更耗电，例如许多支持嵌入式 Wi-Fi 的系统，这一问题就会变得更具挑战性。

4.6.2 电源策略

从主机应用程序的角度来看，主机只明确选择两种运行模式：休眠或启用。

Wi-Fi 子系统配备了一个策略管理实体，使开发人员（主机应用程序的程序员）能够通过预定义的电源策略来指导电源管理算法的行为。*sl_PolicySet* API 用于配置器件电源管理策略。

可用的策略如下：

- 正常（默认）– 在流量传递时间和电源性能之间实现理想折衷。若要设置正常电源管理策略，请使用：
  - `sl_WlanPolicySet(SL_POLICY_PM , SL_NORMAL_POLICY, NULL,0)`
- 始终开启 - Wi-Fi 子系统始终保持完全活动状态，提供出色的 WLAN 流量性能。此策略以用户为导向；用户可提供目标延迟数字。若要设置始终开启电源管理策略，请使用：
  - `sl_WlanPolicySet(SL_POLICY_PM, SL_ALWAYS_ON_POLICY, NULL,0)`
- 长睡眠间隔 – 这种低功耗模式附带一个所需的最大睡眠时间参数。该参数反映了用于信标接收的两次连续唤醒之间的所需睡眠间隔。Wi-Fi 模块会计算所需时间并唤醒到下一个不超过指定时间的 DTIM。对于所需的最大睡眠时间参数，最大允许值为 2 秒。
  若要设置低延迟电源管理策略，请使用：
  - `sl_WlanPolicySet(SL_POLICY_PM , SL_LOW_LATENCY_POLICY, NULL,0)`

备注

此策略仅在客户端模式下适用。它会自动终止器件上运行的 mDNS 和内部 HTTP 服务器。用户应用程序启动的 TCP 和 UDP 服务器会导致不可预测的系统行为和性能。

- 低功耗 – 器件电源管理算法更具投机性，并抓住机会来降低其功耗模式。权衡时倾向于节能性能（传感器应用）。
  - `sl_WlanPolicySet(SL_POLICY_PM, SL_LOW_LATENCY_POLICY, NULL,0)`

备注

仅当 Wi-Fi 子系统未连接到 AP 时，才支持低功耗模式。因此，它主要与收发器模式相关。

4.7 扫描参数

4.7.1 扫描策略

SL_POLICY_SCAN 定义了没有连接时的系统扫描时间间隔。默认间隔为 10 分钟。设置扫描间隔后，系统会立即激活扫描。下一次扫描基于设置的扫描间隔。

- 例如，使用以下代码将扫描间隔设置为 1 分钟：

```
unsigned long intervalInSeconds = 60;
 #define SL_SCAN_ENABLE  1
 sl_WlanPolicySet (SL_POLICY_SCAN,SL_SCAN_ENABLE,   (unsigned char
 *)&intervalInSeconds,sizeof(intervalInSeconds));
```

- 例如，若要禁用扫描：

```
 #define SL_SCAN_DISABLE  0
  sl_WlanPolicySet(SL_POLICY_SCAN,SL_SCAN_DISABLE,0,0);
```

This page intentionally left blank.

WLAN 连接

连接到 WLAN 网络是启动套接字通信的第一步。Wi-Fi 子系统支持两种建立 WLAN 连接的方式：

1. 手动连接 – 应用程序调用一个触发连接过程的 API。
2. 使用配置文件进行连接 – Wi-Fi 子系统自动连接到预定义的连接配置文件。

5.1 手动连接

5.1.1 STA

对于手动连接，用户应用程序必须执行以下步骤：

1. 调用 *sl_WlanConnect* API。该 API 调用接受 AP SSID、安全类型和密钥（如果适用）。
2. 实现一个回调函数来处理异步连接事件 (SL_WLAN_CONNECT_EVENT)，发出连接过程完成的信号。

有关这些 API 的更多信息，请参阅节 17.3 或 Doxygen API 手册。

5.1.2 P2P

相关详细信息，请参阅章节 10。

5.2 使用配置文件的连接

WLAN 配置文件提供连接到给定 AP 所需的信息。这包括 SSID、安全类型和安全密钥。每个配置文件对应某一个 AP。配置文件存储在串行闪存文件系统（非易失性存储器）中，并且在器件复位时会保留。以下 API 可用于处理配置文件：

- *sl_WlanProfileAdd* – 添加新的配置文件。必须提供 SSID 和安全信息，返回值指向存储的索引（在七个可用选项中）。
- *sl_WlanProfileDel* – 删除某个存储的配置文件，或一次删除所有配置文件。索引应为输入参数。
- *sl_WlanProfileGet* – 从特定的存储配置文件检索信息。索引应为输入参数。

有关这些 API 的更多信息，请参阅节 17.3 或 Doxygen API 手册。

```
/* 删除所有存储的配置文件 (0xFF) */
sl_WlanProfileDel(0xFF);
/* 添加优先级为 0（最低）的不安全 AP 配置文件 */
sl_WlanProfileAdd(SL_SEC_TYPE_OPEN, (unsigned char*)UNSEC_SSID_NAME, strlen(UNSEC_SSID_NAME),
g_BSSID, 0, 0, 0, 0);
/* 添加优先级为 1 的 WPA2 安全 AP 配置文件（0 为最低）
sl_WlanProfileAdd(SL_SEC_TYPE_WPA, (unsigned char*)SEC_SSID_NAME, strlen(SEC_SSID_NAME), g_BSSID,
1, (unsigned char*)SEC_SSID_KEY, strlen(SEC_SSID_KEY), 0);
```

5.3 连接策略

SL_POLICY_CONNECTION 将类型参数传递至 *sl_WlanPolicySet* API，以修改或设置连接策略参数。有关此 API 的更多信息，请参阅 doxygen API 手册。下载新版 SDK 以获取完整的示例代码。

WLAN 连接策略定义了五个用于将 SimpleLink 器件连接至给定 AP 的选项。连接策略的五个选项是：

- *自动* – 器件会根据优先级尝试从存储的配置文件连接到 AP。最多支持七个配置文件。尝试连接之后，器件会选择最高优先级配置文件。如果有几个配置文件具有相同的优先级，则根据安全类型作出决策 (WPA2 → WPA → OPEN)。如果安全类型相同，则根据接收到的信号的强度作出选择。

  使用以下命令来设置此选项：

  ```
  sl_WlanPolicySet(SL_POLICY_CONNECTION,SL_CONNECTION_POLICY(1,0,0,0,0),NULL,0)
  ```

- *快速* – 器件尝试连接到上一次连接的 AP。因为 SSID 和通道都是已知的，所以在此模式中，在身份验证请求之前不传输探测请求。

  使用以下命令来设置此选项：

  ```
  sl_WlanPolicySet(SL_POLICY_CONNECTION,SL_CONNECTION_POLICY(0,1,0,0,0),NULL,0)
  ```

- *anyP2P*（仅与 P2P 模式相关）– CC31xx 器件尝试自动连接至可用的第一个 P2P 器件，仅支持按钮。

  使用以下命令来设置此选项：

  ```
  sl_WlanPolicySet(SL_POLICY_CONNECTION,SL_CONNECTION_POLICY(0,0,0,1,0),NULL,0)
  ```

- *autoSmartConfig* – 针对重新启动后的自动 SmartConfig（来自主机的任何命令都将结束此状态）。

  若要设置该选项，请使用 `sl_WlanPolicySet(SL_POLICY_CONNECTION,SL_CONNECTION_POLICY(0,0,0,0,1),NULL,0)`

使用以下命令来设置长睡眠间隔策略：

```
unsigned short PolicyBuff[4] = {0,0,800,0}; // PolicyBuff[2] is max sleep time in mSec
sl_WlanPolicySet(SL_POLICY_PM , SL_LONG_SLEEP_INTERVAL_POLICY, PolicyBuff,sizeof(PolicyBuff));
```

5.4 与连接相关的异步事件

5.4.1 WLAN 事件

该事件处理异步 WLAN 事件。

sl_WlanEvtHdlr 是用于 WLAN 连接或断连指示的事件处理程序。

可能的事件包括：

- SL_WLAN_CONNECT_EVENT – 指示 WLAN 已连接。
- SL_WLAN_DISCONNECT_EVENT – 指示 WLAN 已断开连接。

5.4.2 网络事件

该事件处理网络事件。

sl_NetAppEvtHdlr 是 IP 地址异步事件的事件处理程序；通常在连接新的 WLAN 后接受。

可能的事件包括：

- SL_NETAPP_IPV4_ACQUIRED – 已获取 IP 地址（DHCP 或静态）。

5.4.3 不同连接场景的事件

- 如果将器件连接至 AP – 主机处理器将收到两个事件：SL_WLAN_CONNECT_EVENT 和 SL_NETAPP_IPV4_ACQUIRED（DHCP 过程完成之后）。
- 如果器件未从配置的配置文件中找到 AP – SimpleLink 器件将继续持续扫描接入点，不会建立连接，也不会向主机处理器发送任何事件
- 如果 AP 存在且器件具有适用于该 AP 的配置文件，但它并未连接 – 如果 SimpleLink 器件的配置文件与 AP 具有相同的 SSID，但具有不同的安全密钥；器件将尝试连接，但以失败告终。主机处理器会收到错误代码为 SL_ERROR_CON_MGMT_STATUS_DISCONNECT_DURING_CONNECT 的一般事件。

5.5 使用 BSSID 进行 WLAN 连接

SimpleLink WiFi 器件无法仅根据其 BSSID（手动或使用配置文件）连接到接入点。BSSID 参数可用于区分使用相同 SSID 的两个接入点。这意味着，如果输入 BSSID，SimpleLink 器件将尝试连接到 SSID+BSSID 组合，而不仅仅是根据 SSID。

This page intentionally left blank.

插座

5.1 概述

SimpleLink 中使用的实际网络 API 标准是 BSD (Berkeley) 套接字；Linux™、POSIX 和 Windows™ 套接字 API 都是基于该标准。我们尝试尽量向 Linux 的版本靠拢。主要区别在于错误代码（直接返回结果，不带 errno）和额外的 *setsockopt()* 选项（这是 BSD 套接字 API 的一个小子集，但添加了一些特定于 TI 的套接字选项）。

- 请参阅 Simplelink 文档和示例。
- 维基百科上的 Berkeley 套接字

本页内容假设用户已基本了解互联网协议套件以及 TCP 与 UDP 连接之间的区别。以下是一些基本概念：

5.1.1 传输控制协议 (TCP)

TCP 在维基百科上定义如下：

传输控制协议 (TCP) 是互联网协议套件 (IP)的核心协议之一，并且使用非常普遍，因此整个套件通常被称为 TCP/IP。TCP 提供可靠、有序和经过错误校验的八位字节流传递，确保在连接到局域网、内联网或公共互联网的计算机上运行的程序之间进行通信。该协议位于传输层。不需要 TCP 连接可靠性的应用程序可以改用无连接的用户数据报协议 (UDP)，该协议强调低运行开销和更低延迟，而不是错误校验和传递验证。

5.1.2 用户数据报协议 (UDP)

UDP 在维基百科上定义如下：

用户数据报协议 (UDP) 是互联网协议套件（用于互联网的网络协议集）的核心成员之一。通过使用 UDP，计算机应用程序可以将消息（在这种情况下称为数据报）发送到互联网协议 (IP) 网络上的其他主机，无需事先通信以建立特殊的传输通道或数据路径。UDP 适合不需要执行错误检查和纠正或在应用程序中执行错误检查和纠正的用途，可避免在网络接口级别进行此类处理的开销。对时间敏感的应用程序通常会使用 UDP，因为更好的做法是丢弃数据包而非等待延迟数据包，但这可能不适合实时系统。如果在网络接口级别需要纠错功能，应用程序可以使用为此目的而设计的传输控制协议 (TCP) 或流控制传输协议 (SCTP)。

5.2 套接字连接流程

服务器和客户端之间的 TCP 或 UDP 连接的一般流程如图 5-1 所示。整个流程与 Linux 实现几乎相同。如果用户以前一直在 Linux 上开发联网的应用程序，那么这一流程应该非常简单。

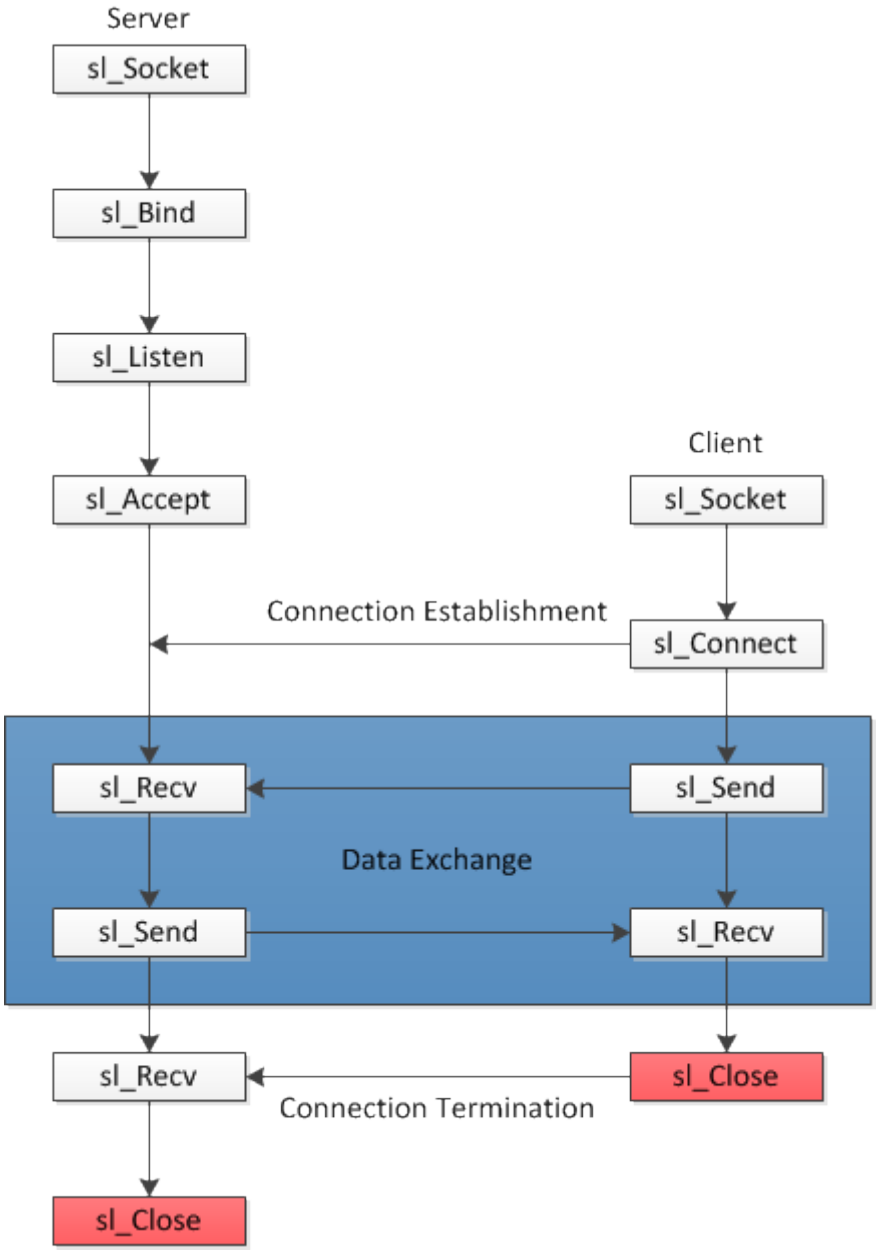


图 5-1. 套接字连接流程

5.3 TCP 连接流程

以下程序结构提供了有关如何使用 SimpleLink API 的一些基本理念。如需完整的示例应用程序代码，请参阅 SDK 示例中的 tcp_socket。

5.3.1 客户端

第一步是创建套接字。返回的套接字处理程序是应用中最重要的元素。如果没有返回的套接字处理程序，网络将无法正常工作。

```
int SockID;
SockID = sl_Socket(SL_AF_INET, SL_SOCK_STREAM, 0);
```

SL_AF_INET 表示使用 IPv4，SL_SOCK_STREAM 表示使用 TCP。socket.h 头文件中对这两个值进行了定义。该示例在第三个参数中设置 0，以便从所选域和类型中选择默认协议。在线文档中提供了更多详细信息。有些结构和常数的定义位于 SDK 内的 socket.h 头文件中。

作为 TCP 客户端，应用会执行 *sl_Connect()* 以连接至服务器。服务器实现方案可以在下面找到。

```
/* IP addressed of server side socket.Should be in long format,
 * E.g: 0xc0a8010a == 192.168.1.10 */
#define IP_ADDR    0xc0a80168
int Status;
int Port = 5001;
SlSockAddrIn_t  Addr;
Addr.sin_family      = SL_AF_INET;
Addr.sin_port        = sl_Htons((UINT16)Port);
Addr.sin_addr.s_addr = sl_Htonl((UINT32)IP_ADDR);
Status = sl_Connect(SockID,  ( SlSockAddr_t *) &Addr, sizeof(SlSockAddrIn_t));
```

Addr 指定目标地址和相关信息。因为结构类型 SlSockAddr 是通用类型，请使用 SlSockAddrIn_t 填充细节并将其转换为 SlSockAddr。成功连接之后，SockID 套接字处理程序已准备好执行数据交换。

sl_Send() 和 *sl_Recv()* 函数可用于数据交换。定义缓冲区大小。

```
#define BUF_SIZE 1400
char SendBuf[BUF_SIZE];
/* Write data to your buffer*/
<write buffer action>
Status = sl_Send(SockID, SendBuf, BUF_SIZE, 0 );
char RecvBuf[BUF_SIZE];
Status = sl_Recv(SockID, RecvBuf, BUF_SIZE, 0);
```

完成后，使用 *sl_Close()* 关闭套接字，以便其他应用能够重复使用资源。

```
sl_Close(SockID);
```

5.3.2 服务器端

与 TCP 客户端不同，TCP 服务器必须在进行通信之前确定几件事情。

- 与客户端的实现方式相似，创建基于 TCP 的 IPv4 套接字。

  ```
  SockID = sl_Socket(SL_AF_INET,SL_SOCK_STREAM, 0);
  ```

- 在 TCP 服务器的实现中，套接字必须执行 Bind 和 Listen 操作。Bind 为服务器套接字提供一个地址。Listen 将套接字置于侦听模式以监视传入的客户端连接。

  ```
  #define PORT_NUM  5001
  SlSockAddrIn_t  LocalAddr;
  LocalAddr.sin_family      = SL_AF_INET;
  LocalAddr.sin_port        = sl_Htons(PORT_NUM);
  LocalAddr.sin_addr.s_addr = 0;
  Status = sl_Bind(SockID,  (SLSockAdd_t *) &LocalAddr, sizeof(SlSockAddrIn_t));
  Status = sl_Listen(SockID, 0);
  ```

- 现在套接字正在侦听，通过 *sl_Accept()* 接受任何传入的连接请求。共有两种方法可以执行此操作：阻塞和非阻塞。此示例在 *sl_SetSockOpt()* 中使用非阻塞机制，并将 *sl_Accept()* 放置在一个循环中，以确保每次失败后都重试连接。有关阻塞和非阻塞机制的详细信息，请参阅节 5.5.1。

  连接成功后，会返回一个新的套接字处理程序 newSockID，然后将该标识用于将来的通信。

  ```
  long nonBlocking = 1;
  int  newSockID;
  Status = sl_SetSockOpt (SockID, SL_SOL_SOCKET, SL_SO_NONBLOCKING, &nonBlocking,
  sizeof(nonBlocking));
  while( newSockID < 0 )
  {
  ```

```
  newSockID = sl_Accept(SockID, ( struct SlSockAddr_t
*) &Addr, (SlSocklen_t*) &AddrSize) ;
    if( newSockID == SL_EAGAIN )
    {
        /* 等待 1 毫秒 */
        Delay(1);
    }
else if( newSockID < 0 )
{
        return -1;
}
}
```

- 服务器端与客户端中采用完全相同的方式来实现数据交换。用户可能需要颠倒顺序；当一端发送时，另一端必须接收。

```
#define BUF_SIZE 1400
char SendBuf[BUF_SIZE];
/* 将数据写入缓冲区*/
<write buffer action>
Status = sl_Send(newSockID, SendBuf, BUF_SIZE, 0 );
char RecvBuf[BUF_SIZE];
Status = sl_Recv(newSockID, RecvBuf, BUF_SIZE, 0);
```

- 最后，使用 *sl_Close()* 来关闭这两个套接字，以便让其他应用程序运行以重复使用资源。

```
sl_Close(newSockID);
sl_Close(SockID);
```

5.4 UDP 连接流程

以下程序结构提供了有关如何使用 SimpleLink API 的一些基本理念。如需完整的示例应用程序代码，请参阅 SDK 示例中的 udp_socket。

5.4.1 客户端

与先前的 TCP 例子类似，创建基于 IPv4 的套接字。但是，将第二个参数更改为 SL_SOCK_DGRAM，这表示将用于 UDP 连接的套接字。

```
SockID = sl_Socket(SL_AF_INET, SL_SOCK_DGRAM, 0);
```

因为 UDP 是无连接协议，所以客户端可以开始将数据发送到指定的目标地址，而不会检查目标是否处于活动状态。

```
#define IP_ADDR          0xc0a80164
#define PORT_NUM         5001
Addr.sin_family      = SL_AF_INET;
Addr.sin_port        = sl_Htons((UINT16)PORT_NUM);
Addr.sin_addr.s_addr = sl_Htonl((UINT32)IP_ADDR);
Status = sl_SendTo(SockID, uBuf.BsdBuf, BUF_SIZE, 0,    (SlSockAddr_t *) &Addr,
sizeof(SlSockAddrIn_t));
```

最后，关闭套接字。

```
sl_Close(SockID);
```

5.4.2 服务器端

套接字的服务器端与客户端相同。

```
SockID = sl_Socket(SL_AF_INET,SL_SOCK_DGRAM, 0);
```

与 TCP 类似，将套接字绑定到本地地址。UDP 无连接，因此不需要侦听。

```
#define PORT_NUM        5001
SlSockAddrIn_t  LocalAddr;
AddrSize = sizeof(SlSockAddrIn_t);
TestBufLen  = BUF_SIZE;
LocalAddr.sin_family      = SL_AF_INET;
LocalAddr.sin_port        = sl_Htons((UINT16) PORT_NUM);
LocalAddr.sin_addr.s_addr = 0;
Status = sl_Bind(SockID,  (SlSockAddr_t *) &LocalAddr, AddrSize);
```

套接字现在尝试接收有关套接字的信息。如果用户未将套接字选项指定为非阻塞，则会阻止该命令，直至接收到 BUF_SIZE 大小的数据量。第五个参数指定了发送数据的源地址。

```
#define BUF_SIZE 1400
SlSockAddrIn_t  Addr;
char            RecvBuf[BUF_SIZE];
Status = sl_RecvFrom(SockID, RecvBuf, BUF_SIZE, 0, (SlSockAddr_t *) &Addr, (SlSocklen_t*)
&AddrSize );
```

通信完成后关闭套接字。

```
sl_Close(SockID);
```

5.5 套接字选项

5.5.1 阻塞与非阻塞

根据具体实现情况，选择在有或没有操作系统的情况下运行应用。通常，在没有操作的情况下运行应用时，使用 *sl_SetSockOpt()* 将套接字选项设置为非阻塞机制，即第三方参数为 SL_SO_NONBLOCKING。但是，基于操作系统的应用可以选择执行多线程，并可处理阻塞函数。

```
Status = sl_SetSockOpt(SockID, SL_SOL_SOCKET, SL_SO_NONBLOCKING,
         &nonBlockingValue, sizeof(nonBlockingValue)) ;
```

如果使用了阻塞机制，则这些函数会阻塞，直至执行完成。

如果使用了非阻塞机制，则这些函数会返回错误代码。错误代码的值取决于所使用的函数。详细信息，请参阅在线文档。

sl_Connect()、*sl_Accept()*、*sl_Aend()*、*sl_Aendto()*、*sl_Recv()* 和 *sl_Recvfrom()* 受此标志影响。如果未设置，则默认使用阻塞机制。

客户端应用中的 *sl_Connect()* 示例：

- 阻塞：*sl_Connect()* 会阻塞，直至它连接至服务器，或发生错误。如果应用是单线程且需要执行其他任务（例如，处理多个套接字以进行读取/写入），请勿使用阻塞机制。但是，如果应用基于操作系统（例如，FreeRTOS），则可以使用阻塞机制。

  ```
  Status = sl_Connect(SockID, ( SlSockAddr_t *) &Addr, AddrSize);
  ```

- 非阻塞：*sl_Connect()* 会立即返回，而不考虑连接。如果连接成功，则返回值 0。如果未成功，则在正常情况下函数返回 SL_EALREADY。TI 建议在循环中调用函数，以便函数一直重新尝试连接，直至用户决定退出。非阻塞机制的优势是可以防止应用始终卡在一个地方。如果应用需要同时执行其他任务（例如，闪烁 LED、读取传感器数据或同时进行其他连接），这将特别有用。

  ```
  while( Status < 0 )
  {
  Status = sl_Connect(SockID, ( SlSockAddr_t *)&Addr, AddrSize);
      if( Status == SL_EALREADY )
      {
          /* 在下一次重试之前等待 1 毫秒 */
          Delay(1);
  ```

```
    }
else if( Status < 0 )
{
        return -1; //Error
}
    /* 在重新尝试连接之前执行其他任务 */
}
```

5.5.2 安全套接字

如需了解更多信息，请参阅 节 8.2。

5.6 SimpleLink 支持的套接字 API

表 5-1 描述了一系列支持的 BSD 套接字和相应的 SimpleLink 实现。

表 5-1. SimpleLink 支持的套接字 API

BSD 套接字	Simplelink 实现	服务器端/客户端	TCP/UDP	说明
socket()	sl_Socket()	两种	两种	创建一个用于通信的终点
bind()	sl_Bind()	服务器端	两种	将套接字分配到地址
listen()	sl_Listen()	服务器端	两种	侦听套接字上的连接
connect()	sl_Connect()	客户端	两种	启动套接字上的连接
accept()	sl_Accept()	服务器端	TCP	接受套接字上的传入连接
send()、recv()	sl_Send()、sl_Recv()	两种	TCP	向 TCP 套接字写入数据和从中读取数据。
write()、read()	不支持			
sendto()、recvfrom()	sl_SendTo()、sl_RecvFrom()	两种	UDP	向 UDP 套接字写入数据和从中读取数据
close()	sl_Close()	两种	两种	使系统释放分配给套接字的资源。对于 TCP，连接将终止。
gethostbyname()、gethostbyaddr()	不支持			
select()	sl_Select()	两种	两种	用于挂起，等待提供的套接字列表中一个或多个套接字准备好读取、准备好写入，或者是因为存在错误
poll()	不支持			
getsockopt()	sl_SockOpt()	两种	两种	检索指定套接字的特定套接字选项的当前值
setsockopt()	sl_SetSockOpt()	两种	两种	为指定的套接字设置特定的套接字选项
htons()、ntohs()	sl_Htons()、sl_Ntohs()	两种	两种	将 16 位无符号值的字节顺序从处理器顺序更改为网络顺序
htonl()、ntohl()	sl_Htonl()、sl_Ntohl()	两种	两种	将 32 位无符号值的字节顺序从处理器顺序更改为网络顺序

5.7 可用套接字的数量

共有 8 个常规（非安全）套接字，其中 2 个套接字会是安全*套接字。

如果所有套接字都是客户端套接字，则总共可以使用 8 个套接字。

- 6 个“常规”套接字和 2 个“安全”套接字
- 7 个“常规”套接字和 1 个“安全”套接字

如果某些套接字是服务器套接字，则可用于通信的套接字数量取决于监听套接字的数量。如果为公共套接字保留的 1 个套接字用于侦听传入的客户端请求，则剩下 7 个私有套接字用于实际的客户端通信。如果保留 2 个服务器套接字用于侦听，则仅剩 6 个私有套接字用于通信，依此类推。

可用于 UDP 连接的服务器套接字的数量仍是 8 个，因为 UDP 是无连接套接字。并不需要套接字一直处于侦听模式，因此所有 8 个套接字均可用于客户端通信。

服务器端安全套接字 (SSL/TLS) 连接数不受影响，因为用户可以使用常规服务器套接字进行侦听。一旦接受了新的客户端连接，就可以切换到安全套接字。

5.8 数据包聚合

默认情况下，SimpleLink 器件对传入并收到的数据包（Rx 端）进行连接。这样做是为了支持更高的 Rx 吞吐量，并且适用于任何类型的套接字。这意味着当调用 *sl_Recv()* 或 *sl_RecvFrom()* 时，SimpleLink 器件会尝试返回这些 API 函数调用所请求的字节数。

当使用 UDP 套接字 (*sl_RecvFrom*) 时，这个特性可能会出现问题，因为 UDP 套接字是无连接的数据套接字，而服务器套接字可以从多个客户端接收数据，所以聚合这些数据包可能是错误的做法。

使用 *sl_NetCfgSet()* API 可以禁用数据包聚合功能。

```
u8 RxAggrEnable = 0;
sl_NetCfgSet(SL_SET_HOST_RX_AGGR, 0, sizeof(RxAggrEnable), (_u8 *) &RxAggrEnable);
```

6.1 概述

在休眠时，器件消耗的功耗最低。在这种状态下不会维护 Wi-Fi 子系统的易失性存储器，只会维护 RTC，以便缩短引导时间并保持系统日期和时间。

在调用 *sl_Stop* API 时，Wi-Fi 子系统会进入休眠状态。此 API 仅接收一个参数，即超时参数。该参数将器件配置为在进入休眠状态之前等待最短的时间。

更多详细信息，请参阅章节 API 概述以及 API Doxygen 应用报告。

This page intentionally left blank.

配置

7.1 概述

Wi-Fi 配置是将新的 Wi-Fi 器件（工作站）连接到 Wi-Fi 网络（接入点）的过程。配置过程涉及向工作站中加载网络名称（通常称为 SSID）及其安全凭据。Wi-Fi 安全标准区分了主要用于家庭和小型组织的个人安全与用于大型办公室和园区的企业安全。为工作站配置企业安全的过程通常涉及安装一些证书，这些证书用于与 IT 部门管理的安全服务器进行交互来验证工作站和网络的完整性。另一方面，个人 Wi-Fi 安全需要由家庭用户进行处理，只需输入预定义的密码即可。为了提供强大的安全性，密码可以长达 64 个字符。

有关配置代码示例和用法的更多详细信息，请参阅 SimpleLink 维基百科网页：http://processors.wiki.ti.com/index.php/CC31xx_%26_CC32xx_Provisioning_Features。

7.2 SmartConfig

7.2.1 一般说明

SmartConfig 技术是 TI 在 2012 年推出的一种专为无外设器件设计的专有配置方法。该方法使用移动应用从智能手机、平板电脑或未配置的 TI Wi-Fi 器件广播网络凭据。当 SmartConfig 在未配置的器件中被触发时，它会进入一个特殊的扫描模式，等待接收由手机应用广播的网络信息。手机需要连接到 Wi-Fi 网络，才能通过无线方式传输 SmartConfig 信号。通常，这与新器件将要接入的网络是同一个家庭网络。

有关 SmartConfig 智能手机应用的更多详细信息，请参阅 SimpleLink 维基百科网页 http://processors.wiki.ti.com/index.php/CC32xx_Provisioning_Smart_Config。

7.2.2 使用方式/API

7.2.2.1 自动激活（开箱即用）

若要启用功能，请启动 SimpleLink 器件。该器件应以 STA 角色身份启动。假设之前没有添加任何配置文件，如果等待几秒后没有命令发出，SmartConfig 应该会启动。

若要在智能手机或 PC 上启动 SmartConfig 应用程序：

1. 将智能手机连接至任何 Wi-Fi 网络。
2. 输入 WLAN 凭据（SSID，安全凭据）。
3. 提供密钥（可选：加密 Wi-Fi 密码）。
4. 按“Start”（开始）按钮。

SmartConfig 操作应该会在几秒内完成，但也可能需要长达两分钟才能完成。如果请求的网络在器件附近，器件会立即连接到网络。

进行自动激活时，以下主题适用：

- 向器件发送任何命令都会终止 SmartConfig 操作。
- 如果在器件串行闪存中存储了密钥，则会对密码加密且必须提供密钥。
- 如果在 SmartConfig 自动启动之前更改了器件配置，则可能会出现问题。在这种情况下，确保满足以下条件：
  - 自动启动策略已设置
  - 自动 SmartConfig 策略已设置
  - 之前没有添加任何配置文件

若要验证配置，请调用：

```
sl_WlanPolicyGet(SL_POLICY_CONNECTION, 0, pVal, pValLen);
```

返回的策略存储在 pVal 所指向的已分配缓冲区中。

如果设置了“自动启动”策略和“自动 SmartConfig”策略，则应设置位 0 和位 4。

如果实际情况并非如此，请通过调用以下项来手动设置这些策略：

```
sl_WlanPolicySet(SL_POLICY_CONNECTION,SL_CONNECTION_POLICY(1,0,0,0,1),NULL,0)
```

为确保没有保存任何配置文件，请通过调用以下项来删除所有已保存的配置文件：

```
sl_WlanProfileDel(255);
```

发送这些命令后，将器件复位，此时 SmartConfig 应该会成功运行。

7.2.2.2 手动激活

若要手动启动 SmartConfig，请发送以下命令：

```
sl_WlanSmartConfigStart(groupIdBitmask,
                        cipher,
                        publicKeyLen,
                        group1KeyLen,
                        group2KeyLen,
                        publicKey,
                        group1Key,
                        group2Key)
```

参数说明：

- groupIdBitmask – 使用 1 作为默认组 ID 位掩码（组 ID 0）。

若要在加密密钥未存储在器件的串行闪存中时对密码进行加密，请使用：

- cipher = 1
- publicKeyLen = 16
- group1KeyLen = 0
- group2KeyLen = 0
- publicKey = 将密钥置于此处（使用 16 字符的字符串）
- group1Key = NULL
- group2Key = NULL

若要在加密密钥存储在器件的串行闪存中时对密码进行加密，请使用：

- cipher = 0
- publicKeyLen = 0
- group1KeyLen = 0
- group2KeyLen = 0
- publicKey = NULL
- group1Key = NULL
- group2Key = NULL

若要避免对密码进行加密，请使用：

- cipher = 1
- publicKeyLen = 0
- group1KeyLen = 0
- group2KeyLen = 0
- publicKey = NULL
- group1Key = NULL
- group2Key = NULL

发送此命令后，SmartConfig 将启动。

手动运行 SmartConfig 时，如果用于对密码进行加密的密钥存储在外部串行闪存中或在命令中提供，则必须在 SmartConfig 应用中提供该密钥。

7.2.2.3 停止智能配置

若要停止 SmartConfig 操作，请调用：

```
sl_WlanSmartConfigStop()
```

备注

- 器件在连接到请求的网络后，应该会从 AP 或路由器接收 IP 地址。
- 如果智能手机所连接的 Wi-Fi 网络使用不适合 SmartConfig 的传输模式和速率，SmartConfig 操作可能无法成功完成。在这种情况下，请使用 AP 配置方法。

7.3 AP 模式

7.3.1 一般说明

接入点 (AP) 模式是当今无外设器件的常见配置方法。在 AP 模式下，未配置的器件最初作为 AP 而唤醒，具有器件制造商定义的 SSID。在首次尝试连接到家居网络之前，未配置的器件会创建自己的网络，使 PC 或智能手机能够直接与其连接，以方便其初始配置。

7.3.2 使用方式/API

1. 首先，以 AP 角色启动 SimpleLink 器件。共有两种启动方法：
   a. 调用 *sl_WlanSetMode(mode)*，其中 mode 应为 ROLE_AP (2)。复位器件。更多详细信息，请参阅章节 9。
   b. 强制将该器件作为 AP 启动。
2. 使用智能手机或任何其他具有 Wi-Fi 连接的器件搜索该器件。该器件应出现在包含所有可用网络的列表中。器件名称应为 mysimplelink-xxyyzz，其中 xxyyzz 是器件 MAC 地址的最后六位数字。如果用户已通过调用 *sl_WlanCfgSet(0, 0, ssid_length, ssid)* 更改了器件 SSID，该 SSID 将显示在列表中。如果用户尚未更改 SSID，但通过调用 *sl_NetAppSet(16, 0, urn_length, urn)* 更改了器件 URN，则器件名称应为 urn-xxyyzz，其中 xxyyzz 是器件 MAC 地址的最后六位数字。
3. 在器件屏幕上，选择器件网络并与其连接。

   备注

   除非用户更改了模式，否则连接应处于不安全状态。在这种情况下，请提供器件发出请求时所输入的密码。
4. 连接器件后打开浏览器，转至：http://www.mysimplelink.net。
5. 转至“Profiles”（配置文件）选项卡。
6. 输入 WLAN 凭据（SSID、安全类型和密钥），选择该配置文件的优先级（0-7 之间的任意值），然后按“Add”（添加）。请参阅图 7-1。

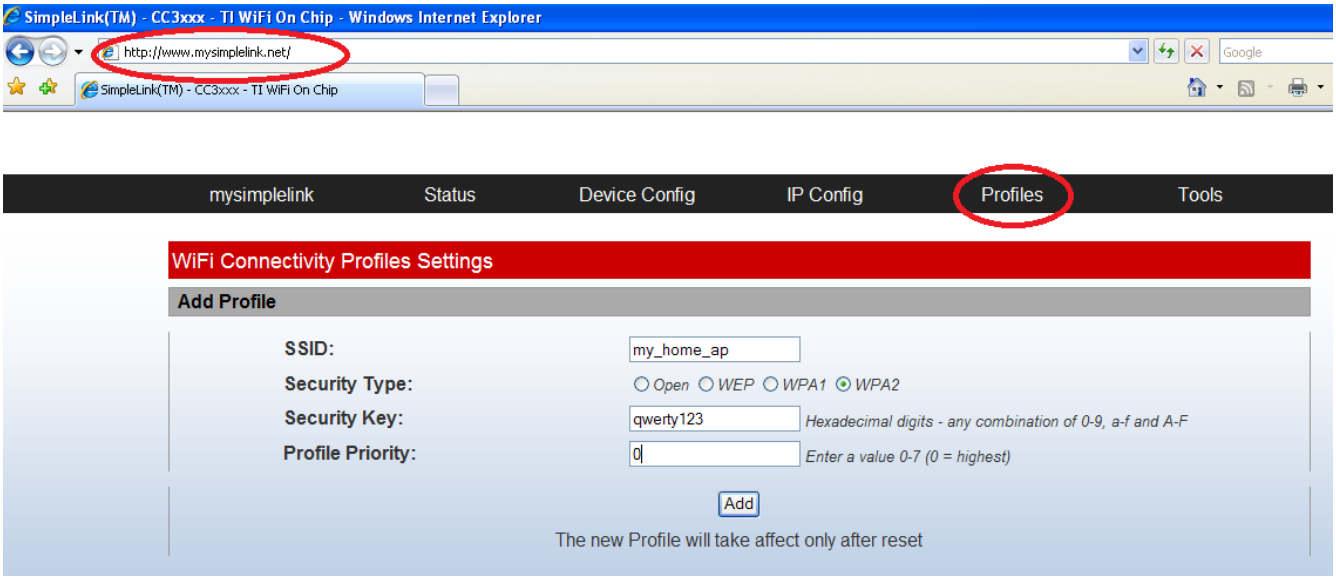


图 7-1. AP 模式连接

7. 滚动到网页底部，并检查器件是否已添加到“Profiles”（配置文件）（不会显示密码）。请参阅图 7-2。

图 7-2. 配置文件

8. 以 STA 角色启动器件。转至“Device Config”（器件配置）选项卡，然后将“Device Mode”（器件模式）设置为“Station”（（工作站）或删除“Force AP”约束）并复位器件。复位器件后，器件会自动连接到请求的网络。请参阅图 7-3。

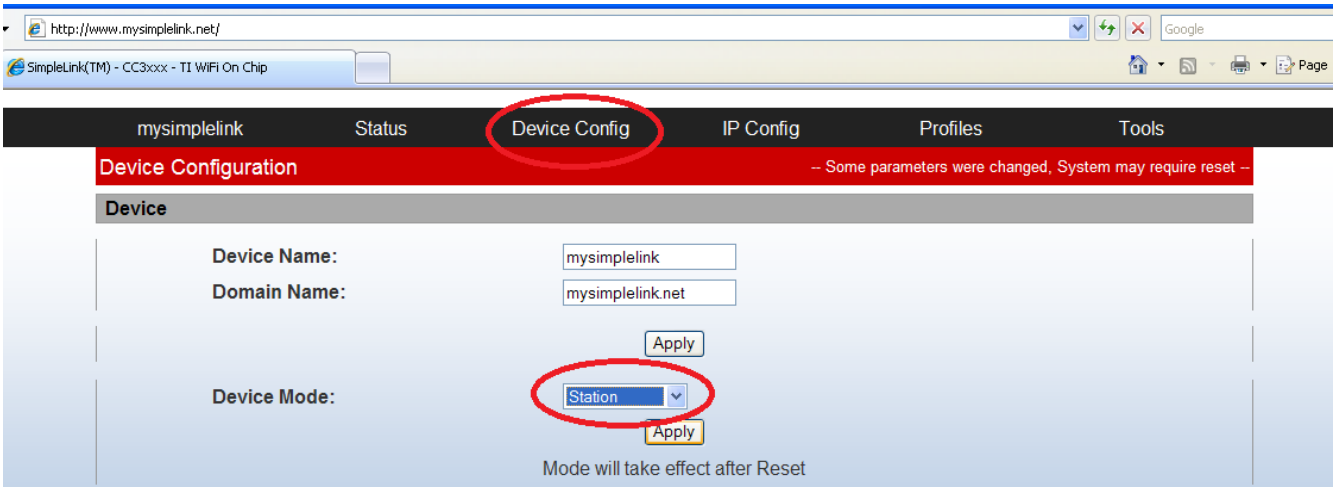


图 7-3. “Device Config”（器件配置）选项卡

7.3.3 配置 AP 配置时的注意事项

当使用智能手机连接到 (AP) 器件时，由于运行 DHCP 服务器，该器件会分配一个 IP 地址。智能手机（或其他配置器件）不应使用静态 IP 地址。

输入 WLAN 凭据后（在“Profiles”选项卡上），进入 STA 模式并重置器件。器件应连接到请求的网络，并且会从 AP 或路由器接收 IP 地址。

7.4 WPS

7.4.1 一般说明

Wi-Fi 保护设置 (WPS) 是目前为数不多的可用于配置无外设器件的工业标准。它是 Wi-Fi Alliance 在 2006 年推出的一种简单而安全的方法，无需知道网络名称且无需输入长密码即可配置器件。该标准为启用 WPS 的接入点 (AP) 定义了两种强制性方法：个人标识号 (PIN) 方法和 PushButton-Connect (PBC) 方法。

按钮式：按下 AP 中的 W+PS 按钮，或如果该按钮不可用，则使用 AP 的 GUI 来启动 WPS 进程。AP 将进入 WPS 配置过程并持续 2 分钟。在此期间，SimpleLink 器件还应通过调用具有 WPS 参数的 *sl_WlanConnect* API 进入 WPS 配置过程（请参阅节 7.4.3）。例如，调用此 API 可以映射到 MCU 中的一个按钮。此过程结束时，系统将自动配置具有网络名称和安全性的无线网络。

基于 PIN：使用 AP 的 GUI 输入由主机生成的 PIN。AP 将进入 WPS 配置过程并持续 2 分钟。在此期间，SimpleLink 器件还应通过调用具有 WPS 参数的 *sl_WlanConnect* API 进入 WPS 配置过程（请参阅节 7.4.3）。此过程结束时，系统将自动配置具有网络名称和安全性的无线网络。

WPS 过程成功完成后，系统将根据 AP 的配置（Open、WEP、WPA 或 WPA2），在正确的安全设置中建立与 AP 的连接。连接参数另存为一个配置文件。使用连接策略 AUTO 会在复位后触发重新连接。

7.4.2 使用方式/API

```
sl_WlanConnect(char* pName, int NameLen, unsigned char *pMacAddr, SlSecParams_t* pSecParams,
SlSecParamsExt_t* pSecExtParams);
```

这个具有正确设置的 API 可以触发两种配置的 WPS 连接：按钮式和基于 PIN。

参数配置：

- **pName** – NULL
- **NameLen** – 0
- **unsigned char *pMacAddr** – NULL
- **SlSecParamsExt_t* pSecExtParams** – 与 WPS 无关，设置为 NULL

- 按钮式：
  - **SlSecParams_t* pSecParams**：类型 – SL_SEC_TYPE_WPS_PBC (3)
    密钥 – NULL
    密钥长度 – 设置为 0
- 基于 PIN：
  - **SlSecParams_t* pSecParams**：类型 – SL_SEC_TYPE_WPS_PIN (4)
    密钥 – WPS PIN 码
    密钥长度 – WPS PIN 码长度

```
int sl_WlanProfileGet(int Index,char* pName, int *pNameLen, unsigned char *pMacAddr,
SlSecParams_t* pSecParams, SlSecParamsExt_t* pEntParams, unsigned long *pPriority)
```

  - 此 API 检索在 WPS 连接过程中保存的配置文件参数。

```
sl_WlanProfileDel(int Index)
```

  - 此 API 用于删除在 WPS 连接过程中保存的配置文件。在索引设置为 255 的情况下调用此 API 会清除存储的所有配置文件。

7.4.3 使用 WPS 的示例

```
// Push Button
void main()
{
     SlSecParams_t WPSsecParams;
     Int status;
     Int role;
     role = sl_Start(NULL, NULL, NULL);
     if( 0 > role )
          {
     printf("failed start cc3100\n");
          }
     if(role == ROLE_STA)
          {
WPSsecParams.Type = SL_SEC_TYPE_WPS_PBC;
     WPSsecParams.Key = NULL;
     WPSsecParams.KeyLen = 0;
status = sl_WlanConnect(0,0,0,&WPSsecParams,0);
     while (SL_IPEQUIRED != g_SlConnState)
          {
     Sleep(20);
          }
          }
}
// PIN-based
void main()
{
     SlSecParams_t WPSsecParams;
     Int status;
     Int role;
     role = sl_Start(NULL, NULL, NULL);
     if( 0 > role )
          {
               printf("failed start cc3100\n");
          }
     if(role == ROLE_STA) {
        WPSsecParams.Type = SL_SEC_TYPE_WPS_PIN;
               WPSsecParams.Key = "34374696"; //example pin code
               WPSsecParams.KeyLen = 8;
               status = sl_WlanConnect("" ==>,0,0,&WPSsecParams,0);
     while (SL_IPEQUIRED != g_SlConnState) {
                    Sleep(20);
               }
          }
}
```

7.4.4 配置选项之间的权衡

表 7-1. 配置方法

配置方法	接入点模式	SmartConfig	WPS
需要的工具	网络浏览器	Android 或 iOS 手机应用	路由器上的按钮
支持的网络	任何网络	不支持使用 MIMO、5GHz、ISO-40MHz 和专有调制方案的网络连接	仅启用 WPS 的路由器
步数	多步	1 步	1 步（按钮）
配置的器件数	配置一个器件	配置多个步骤	配置一个器件
家庭网络连接	手机必须断开与家庭网络的连接	手机仍然连接到家庭网络	不适用
安全	可以是安全的	可以是安全的	未受安全保护
远程应用程序	不需要	需要（受 Android 4.2+ 和 iOS 6+ 支持）	不适用
其他说明	不适用	无法识别使用中文或亚洲字符的 SSID	不适用

- 每种配置方法都有其优点和局限性。
- 由于没有一种配置方法是完美的，一个好的实用方法是在具体产品中支持多种选项。这将确保在最终产品中获得理想可靠的配置。

备注

采用 SmartConfig 的产品还应具有可作为后备配置的 AP 模式或 WPS 。

章节 8

安全性

8.1 WLAN 安全

8.1.1 个人

Wi-Fi 子系统支持 Wi-Fi 安全类型 AES、TKIP 和 WEP。在手动连接 API 和配置文件连接 API 中均通过相同的参数类型 **SlSecParams_t** 来设置个人安全类型和个人安全密钥。此结构包含以下字段：

- 安全类型 – 所用的安全类型。选项包括：
  - SL_SEC_TYPE_OPEN – 无安全性（默认值）。
  - SL_SEC_TYPE_WEP – WEP 安全性。
  - SL_SEC_TYPE_WPA – 用于 WPA\PSK 和 WPA2\PSK 安全类型，或 WPA\WPA2 PSK 安全类型的混合模式（例如，TKIP、AES、混合模式）。
  - SL_SEC_TYPE_WPA_ENT – WPS 安全性。更多信息，请参阅节 7.4。
  - SL_SEC_TYPE_WPS_PBC ENT – 按钮 WPS 安全性。更多信息，请参阅节 7.4。
  - SL_SEC_TYPE_WPS_PIN ENT – 基于引脚的 WPS 安全性。更多信息，请参阅节 7.4。
- 密钥 – 包含预共享密钥 (PSK) 值的字符区域。
- 密钥长度 – 预共享密钥的字符数。

以下是添加 WPA2 安全 AP 配置文件的示例代码：

```
secParams.Type = SL_SEC_TYPE_WPA;
secParams.Key = SEC_SSID_KEY;
secParams.KeyLen = strlen(SEC_SSID_KEY);
sl_WlanProfileAdd((char*)SEC_SSID_NAME, strlen(SEC_SSID_NAME), g_BSSID,  &secParams, 0, 7, 0);
```

8.1.2 企业级

8.1.2.1 一般说明

根据 802.1x 身份验证过程，SimpleLink 器件支持 Wi-Fi 企业连接。企业连接需要 AP 后面的广播服务器对工作站进行身份验证。该器件支持以下身份验证方法：

- EAP-TLS
- 带 MSCHAP 的 EAP-TTLS
- 带 TLS 的 EAP-TTLS
- 带 PSK 的 EAP-TTLS
- 带 TLS 的 EAP-PEAP
- 带 MSCHAP 的 EAP-PEAP
- 带 PSK 的 EAP-PEAP
- EAP-FAST

工作站通过身份验证后，AP 与该工作站协商 WPA(1/2) 安全性。

8.1.2.2 使用方式/API

连接到企业网络时，需要三个文件：

- 私钥 – PEM 格式的工作站（客户端）RSA 私钥文件
- 客户端证书 – 由身份验证网络提供的 PEM 格式的客户端证书（确保公钥与私钥相匹配）
- 服务器根证书颁发机构文件 – 该文件对服务器进行身份验证。该文件必须为 PEM 格式。

这三个文件必须按以下名称进行编程，以便器件使用：

- 证书颁发机构：/cert/ca.pem
- 客户端证书：/cert/client.pem
- 私钥：/cert/private.key

建立连接：

```
sl_WlanConnect(char* pName, int NameLen, unsigned char *pMacAddr, SlSecParams_t* pSecParams,
SlSecParamsExt_t* pSecExtParams)
```

```
sl_WlanProfileAdd(char* pName, int NameLen, unsigned char *pMacAddr, SlSecParams_t* pSecParams,
SlSecParamsExt_t* pSecExtParams, unsigned long  Priority, unsigned long  Options)
```

sl_WlanConnect 和 *sl_WlanProfileAdd* 命令用于不同类型的 Wi-Fi 连接。连接命令用于一次性连接；当自动连接开启时使用添加配置文件命令 (请参阅添加配置文件命令白皮书) 。对于企业连接，使用那些具有额外安全参数的命令 – SlSecParamsExt_t。

下文简要介绍了这些命令的前五个参数。添加配置文件命令白皮书详细介绍了该命令的其他参数。

- SSID 名称 – Wi-Fi 网络的名称
- SSID 长度
- 标志 – 不适用于企业连接
- 指向 SlSecParams_t 的指针 –
  -

```
typedef struct
{
    unsigned char   Type; - type should be SL_SEC_TYPE_WPA_ENT
    char*           Key;  - a key password for the enterprise connection that
                            must have it.MSCHAP, FAST ETC.
    unsigned char   KeyLen;
}SlSecParams_t;
```

- 指向 SlSecParamsExt_t 的指针 –
  -

```
typedef struct
{
    char*           User;  -       the enterprise user name
    unsigned char   UserLen;
    char*           AnonUser;  -   the anonymous user name (optional) for two phase
                                   enterprise connections.
    unsigned char   AnonUserLen;
    unsigned char   CertIndex;  -  not supported
    unsigned long   EapMethod; -
                                   SL_ENT_EAP_METHOD_TLS
                                   SL_ENT_EAP_METHOD_TTLS_TLS
                                   SL_ENT_EAP_METHOD_TTLS_MSCHAPv2
                                   SL_ENT_EAP_METHOD_TTLS_PSK
                                   SL_ENT_EAP_METHOD_PEAP0_TLS
                                   SL_ENT_EAP_METHOD_PEAP0_MSCHAPv2
                                   SL_ENT_EAP_METHOD_PEAP0_PSK
                                   SL_ENT_EAP_METHOD_PEAP1_TLS
                                   SL_ENT_EAP_METHOD_PEAP1_MSCHAPv2
                                   SL_ENT_EAP_METHOD_PEAP1_PSK
                                   SL_ENT_EAP_METHOD_FAST_AUTH_PROVISIONING
                                   SL_ENT_EAP_METHOD_FAST_UNAUTH_PROVISIONING
                                   SL_ENT_EAP_METHOD_FAST_NO_PROVISIONING
}SlSecParamsExt_t;
```

8.1.2.3 示例

图 8-1 显示了连接到企业网络的简单 WLAN 连接命令的示例。

```
void WlanConnect()
{
    SlSecParams_t secParams;
    SlSecParamsExt_t extParams;

    // fill the security parameters
    secParams.Key = "xfdsdnke34wki4de";
    secParams.KeyLen = strlen("xfdsdnke34wki4de");
    secParams.Type = SL_SEC_TYPE_WPA_ENT;

    // fill the enterprise extra parameters
    extParams.User = "myRealUserName";
    extParams.UserLen = strlen("myRealUserName");
    extParams.AnonUser = "myFakePhase1";
    extParams.AnonUserLen = strlen("myFakePhase1");
    extParams.EapMethod = SL_ENT_EAP_METHOD_PEAP0_MSCHAPv2;

    // connect command
    sl_WlanConnect("enterpriseNetwork",strlen("enterpriseNetwork"),0,&secParams,&extParams);
}
```

图 8-1. WLAN 连接命令

8.1.2.4 限制

没有将证书文件绑定到 WLAN 企业连接的命令。必须使用节 8.1.2.2 中指定的名称对网络证书进行编程。

8.2 安全套接字

8.2.1 一般说明

SSL 是基于 TCP 的安全套接字，使用户能够安全地连接到服务器（和互联网站点），或打开安全服务器。

本章介绍了如何将套接字与主机驱动程序一同使用，以及如何为 SSL 生成证书和密钥。

8.2.2 使用方式/API

sl_Socket(SL_AF_INET, SL_SOCK_STREAM, SL_SEC_SOCKET) – 此命令可打开安全套接字。前两个参数是典型的 TCP 套接字参数，最后一个参数用于启用安全性。

使用任何标准 BSD 命令（*sl_Close*、*sl_Listen*、*sl_Accept*、*sl_Bind*、*sl_SetSockOpt* 等）打开客户端、打开服务器、更改套接字参数，等等。

使用 BSD 命令，用户便无需选择 SSL 方法 (SSLv3 TLS1.0/1.1/1.2) 且无需选择连接密码套件（这两项操作在 SSL 握手中协商），即可进行连接。

在客户端套接字中，在根 CA 验证服务器之前，不会验证服务器的证书。

在服务器套接字中，用户必须提供服务器证书和私钥。

使用带有专有选项的 setsockopt 命令来配置套接字的安全参数。

8.2.2.1 选择方法

手动定义 SSL 方法。SimpleLink WiFi 器件支持 SSLv3 TLSv 1.0/1.1/1.2。

SlSockSecureMethod method; method.secureMethod = 选择以下定义之一：

- SL_SO_SEC_METHOD_SSLV3
- SL_SO_SEC_METHOD_TLSV1
- SL_SO_SEC_METHOD_TLSV1_1

- SL_SO_SEC_METHOD_TLSV1_2
- SL_SO_SEC_METHOD_SSLv3_TLSV1_2
- SL_SO_SEC_METHOD_DLSV1

```
Sl_SetSockOpt(sockID, SL_SOL_SOCKET, SL_SO_SECMETHOD, &method, sizeof(SlSockSecureMethod));
```

8.2.2.2 选择密码套件

手动定义 SSL 连接和握手安全算法，又称密码套件。

SlSockSecureMask Mask; mask.secureMask = 在以下项之间选择逻辑或：

- SL_SEC_MASK_SSL_RSA_WITH_RC4_128_SHA
- SL_SEC_MASK_SSL_RSA_WITH_RC4_128_MD5
- SL_SEC_MASK_TLS_RSA_WITH_AES_256_CBC_SHA
- SL_SEC_MASK_TLS_DHE_RSA_WITH_AES_256_CBC_SHA
- SL_SEC_MASK_TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
- SL_SEC_MASK_TLS_ECDHE_RSA_WITH_RC4_128_SHA
- SL_SEC_MASK_TLS_RSA_WITH_AES_128_CBC_SHA256
- SL_SEC_MASK_TLS_RSA_WITH_AES_256_CBC_SHA256
- SL_SEC_MASK_TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
- SL_SEC_MASK_TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256

```
Sl_SetSockOpt(sockID, SL_SOL_SOCKET, SL_SO_SECMETHOD, &mask, sizeof(SlSockSecureMask));
```

8.2.2.3 为套接字选择受保护的文件

定义在连接时套接字将使用的文件。共有四个文件可以连接到套接字：

- 私钥
- 证书
- 根 CA
- DH 文件

这些文件是客户端和服务器套接字中的文件：

客户端

- 私钥、证书 – 如果客户端必须由服务器进行身份验证，则私钥和证书都是必不可少的。使用两个 setsockopt 命令来配置每个文件。
- 根 CA – 这是颁发服务器证书的根证书颁发机构，用于验证服务器是否真实。此文件不是必需的。如果未验证服务器，则虽然会进行连接，但连接命令会返回错误 SL_ESECSNOVERIFY。可以忽略此错误，因为它只是针对未经身份验证的连接提出的警告。
- DH 文件 – 不在客户端中使用

服务器

- 私钥、证书 – 对于服务器而言是必不可少的。
- 根 CA – 颁发给客户端的证书。设置文件后，它要求客户端发送证书以进行客户端身份验证。
- DH 文件 – 用于支持 DH 密码套件 – TLS_DHE_RSA_WITH_AES_256_CBC_SHA。

若要将文件绑定到套接字，请将文件编程到器件。然后，使用 setsockopt 输入文件名。所有受保护的文件必须为 DER 格式。

受保护的文件的 Setsockopt 选项：

- SL_SO_SECURE_FILES_PRIVATE_KEY_FILE_NAME
- SL_SO_SECURE_FILES_CERTIFICATE_FILE_NAME
- SL_SO_SECURE_FILES_CA_FILE_NAME

- SL_SO_SECURE_FILES_DH_KEY_FILE_NAME

例如，使用器件中的文件 rootCA.der：

```
Sl_SetSockOpt(sockID, SL_SOL_SOCKET, SL_SO_SECURE_FILES_CA_FILE_NAME,”rootCA.der”,
strlen(“rootCA.der”));
```

注意 setsockopt 中的 strlen，而不是 sizeof。

8.2.2.4 设置域名以用于验证和 SNI

设置域名以便在 SSL 握手期间验证所需的域。域验证用于帮助抵御“中间人”攻击；在这种攻击中，第三方可能从签署服务器证书的同一个根 CA 购买假证书，并将流量重定向到他们的服务器。除了全链验证外，TI 建议检查域名。此选项仅适用于客户端模式。应在 sl_Connect 或 sl_Listen 之前调用此选项。根据 RFC 6066，设置域名还可以启用客户端 hello 消息的 SNI 扩展。

示例：

```
_i16 status;
status = sl_SetSockOpt(SockID, SL_SOL_SOCKET, SO_SECURE_DOMAIN_NAME_VERIFICATION, "www.google.com",
strlen("www.google.com"));
```

8.2.3 使用 SSL 的示例

```
int CreateConnection(unsigned long DestinationIP)
{
    int Status;
    SlSockAddrIn_t  Addr;
    int AddrSize;
    int SockID = 0;
    SlTimeval_t timeval;
    Addr.sin_family = SL_AF_INET;
    Addr.sin_port = sl_Htons(443);  // secured connection
    Addr.sin_addr.s_addr = sl_Htonl(DestinationIP);
    AddrSize = sizeof(SlSockAddrIn_t);
    SockID = sl_Socket(SL_AF_INET,
                       SL_SOCK_STREAM,
                       SL_SEC_SOCKET);
    if( SockID < 0 )
    {
        // error
        while (1);
    }

      Sl_SetSockOpt(sockID,
                    SL_SOL_SOCKET,
                    SL_SO_SECURE_FILES_CA_FILE_NAME,
                    ”rootCA.der”,
                    strlen(“rootCA.der”));
     Status = sl_Connect(SockID,
                         ( SlSockAddr_t *)&Addr,
                         AddrSize);

     if( Status < 0 && Status != SL_ESECSNOVERIFY )
     {
         // error
         while(1);
     }
     return SockID;
}
```

8.2.4 支持的加密算法

表 8-1. 支持的加密算法

加密算法	标准协议	用途	加密密钥长度
RC4	WEP、TKIP	数据加密	128 位
AES	WPA2	数据加密、身份验证	256 位

表 8-1. 支持的加密算法 (continued)

加密算法	标准协议	用途	加密密钥长度
DES	-	数据加密	56 位
3DES	-	数据加密	56 位
SHA1	EAP-SSL/TLS	认证	160 位
SHA256	EAP-SSL/TLS	认证	256 位
MD5	EAP-SSL/TLS	认证	128 位
RSA	EAP-SSL/TLS	认证	2048 位
DHE	EAP-SSL/TLS	认证	2048 位
ECDHE	EAP-SSL/TLS	认证	160 位

- 支持的 SSL 密钥大小：
  - 客户端模式
    - 客户端模式
    - 密钥交换和质询 – 必须小于或等于 4096
    - 客户端验证 – 必须小于或等于 2048
  - 服务器模式
    - 签名验证校验 – 必须小于或等于 2048
    - 密钥交换和质询 – 必须小于或等于 2048
    - 客户端验证 – 必须小于或等于 2048

8.3 限制

CC31xx 和 CC32xx 可以支持 WEP 十六进制格式，但略微受到限制

8.3.1 主要的已知限制

8.3.1.1 STA 模式

表 8-2. STA 模式

`Sl_connect（主机）或添加配置文件（主机）`	可以支持 WEP 十六进制
	不支持 WPA 十六进制
智能配置	不支持 WPA ASCII，仅支持 WEP 十六进制
	不支持 WPA 十六进制
`HTTP（添加配置文件）`	可以支持 WEP 十六进制
	不支持 WPA 十六进制

8.3.1.2 AP 模式

表 8-3. AP 模式

`ApConfigCmd(Host)`	可以支持 WEP 十六进制
	不支持 WPA 十六进制
`HTTP(AP_config)`	可以支持 WEP 十六进制
	不支持 WPA 十六进制

8.3.1.3 JavaScript 示例

```
<script>

function hex2ascii(hex)
{
        var i;
        var ascii = '';
```

```
        for (i=0 ; i < hex.length/2 ; i++)
        {
                ascii += String.fromCharCode(parseInt(hex.substr(i*2,2),16));
        }
        return ascii;
}

function myFunction()
{
        var HexPassword = "12345ABCDE";
        var AsciiPassword = hex2ascii(HexPassword);
}

</script>
```

8.3.1.4 主机驱动程序示例

```
static INT32 establishConnectionWithAP()
{
        char SSID[MAX_SSID_SIZE];
        char password[MAX_PASSKEY_SIZE];
        SlSecParams_t secParams = {0};
        INT32 retVal = -1;

        /*****************************************************************/
        /**  将 AP 配置为 WEP，采用 40 位十六进制密码："12345ABCDE"  **/
        /*****************************************************************/
        password[0] = 0x12;
        password[1] = 0x34;
        password[2] = 0x5A;
        password[3] = 0xBC;
        password[4] = 0xDE;
        password[5] = 0;

        secParams.Key = password;
        secParams.KeyLen = 5; // [--strlen(password) - Don't use strlen in-order not to bump on zero/
NULL in the password]
        secParams.Type = SL_SEC_TYPE_WEP;
        strcpy(SSID, "myApSsid");

        retVal = sl_WlanConnect(SSID,strlen(SSID),0,&secParams,0);

        ASSERT_ON_ERROR(__LINE__, retVal);

        printf("Connecting to AP %s...\n", SSID);

        /* Wait */
        while((!IS_CONNECTED(g_Status)) || (!IS_IP_AQUIRED(g_Status)));

        return SUCCESS;
}
```

AP 模式

9.1 一般说明

应通过调用允许配置部分 WLAN 参数和部分 IP 参数的 API，对 AP 进行设置和配置。本章介绍了可配置的参数及其配置方法。

9.2 设置 AP 模式 – API

```
sl_WlanSetMode(const unsigned char    mode)
```

其中 mode 应该是 ROLE_AP (2)。此更改在复位后生效。

备注

此更改在复位后生效。

9.3 WLAN 参数配置 – API

```
sl_WlanCfgSet (unsigned short ConfigId,
               unsigned short ConfigOpt,
               unsigned short ConfigLen,
               unsigned char *pValues)
```

此函数用于设置待配置的用户参数。输入参数包括：

- ConfigId – 应根据参数设置为 SL_WLAN_CFG_AP_ID (0) 或 SL_WLAN_CFG_GENERAL_PARAM_ID (1)
- ConfigOpt – 确定待配置的参数
- ConfigLen – 参数大小（以字节为单位）
- pValues – 指向包含参数的存储器的指针

备注

此更改在复位后生效。

表 9-1 说明了如何配置每个参数。

表 9-1. WLAN 参数

参数	说明	ConfigId	ConfigOpt	ConfigLen
SSID	服务集标识符 (SSID)，这是一个 1 到 32 字节的字符串，通常称为“网络名称”。 默认值：由器件 URN 和 MAC 地址的最后 6 位数字组成。例如，mysimplelink-112233。	SL_WLAN_CFG_AP_ID (0)	0	1-32
Beacon interval（信标间隔）	信标传输之间的时间间隔。范围：15 <= 间隔 <= 65,535 毫秒。 默认值：100.	SL_WLAN_CFG_AP_ID (0)	2	2
Channel（通道）	AP 的运行通道。该范围取决于国家/地区代码。 范围： US：1-11 JP：1-14 EU：1-13 默认值：6	SL_WLAN_CFG_AP_ID (0)	3	1
SSID Hidden（SSID 隐藏）	是否不在信标帧内广播 SSID。 默认值：禁用	SL_WLAN_CFG_AP_ID (0)	4	1

表 9-1. WLAN 参数 (continued)

参数	说明	ConfigId	ConfigOpt	ConfigLen
DTIM	表示传递流量指示消息 (DTIM) 的时间间隔。DTIM 字段是一个倒计时字段，通知客户端收听广播和多播消息的下一个窗口期。DTIM 字段位于信标帧中。 范围：DTIM > 0 默认值：2	SL_WLAN_CFG_AP_ID (0)	5	1
安全类型	网络的安全模式。 范围：Open（无安全性）/WEP/WPA 默认值：开路	SL_WLAN_CFG_AP_ID (0)	6	1
密码	在安全类型不是开放的情况下用于网络的密码。密码应该是人类可读的字符串。对于 WEP，密码应该是 5 到 13 个字符。对于 WPA，密码应该是 8 到 63 个字符。	SL_WLAN_CFG_AP_ID (0)	7	5-63
WPS 状态	Wi-Fi Protected Setup（最初是 Wi-Fi Simple Config）是一种网络安全标准，让用户能够轻松保护无线家庭网络。 默认值：禁用	SL_WLAN_CFG_AP_ID (0)	8	1
国家/地区代码	标识 AP 的国家/地区代码。可能的值为 US（美国）、JP（日本）或 EU（欧洲）。 默认值是“US”。	SL_WLAN_CFG_GENERAL_PARAM_ID (1)	9	1
AP TX 功率	AP 传输功率。 范围：0（最大值）– 15（最小值）。 默认值：0（最大 TX 功率）。	SL_WLAN_CFG_GENERAL_PARAM_ID (1)	11	1

9.4 WLAN 参数查询 – API

```
sl_WlanCfgGet(unsigned short ConfigId,
              unsigned short *pConfigOpt,
              unsigned short *pConfigLen,
              unsigned char *pValues)
```

此函数获取用户请求的参数，其中：

- **ConfigId** – 应该像在 SET 函数中一样设置为 SL_WLAN_CFG_AP_ID (0) 或 SL_WLAN_CFG_GENERAL_PARAM_ID (1)
- **ConfigOpt** – 确定要配置的参数。应该是字段的地址。字段的值与 SET 函数中的值相同。
- **ConfigLen** – 输出：指向返回的参数大小的指针（以字节为单位）
- **pValues** – 输出：指向包含参数的存储器的指针

9.5 AP 网络配置

用户必须设置 AP IP 参数，具体来说就是 IP 地址、网关地址、DNS 地址和网络掩码。

若要设置 IP 地址调用：

```
sl_NetCfgSet(unsigned char ConfigId ,
             unsigned char ConfigOpt,
             unsigned char ConfigLen,
             unsigned char *pValues)
```

- **ConfigId** – 应设置为 SL_IPV4_AP_P2P_GO_STATIC_ENABLE (7)
- **ConfigOpt** – 应设置为 1

- **ConfigLen** – 应为参数大小（以字节为单位）
- **pValues** – 指向包含参数的存储器的指针

此示例显示如何将 IP、网关和 DNS 地址配置为 9.8.7.6，将子网设置为 255.255.255.0：

```
_NetCfgIpV4Args_t ipV4;
ipV4.ipV4               = 0x09080706;
ipV4.ipV4Mask           = 0xFFFFFF00;
ipV4.ipV4Gateway        = 0x09080706;
ipV4.ipV4DnsServer      = 0x09080706;
sl_NetCfgSet(SL_IPV4_AP_P2P_GO_STATIC_ENABLE,
             1
             ,sizeof(_NetCfgIpV4Args_t),
             (unsigned char *)&ipV4);
```

备注

更改在复位后生效。

默认值：

ipV4	= 0xc0a80101;	/* 192.168.1.1 */
defaultGatewayV4	= 0xc0a80101;	/* 192.168.1.1 */
ipV4DnsServer	= 0xc0a80101;	/* 192.168.1.1 */
subnetV4	= 0xFFFFFF00;	/* 255.255.255.0 */

9.6 DHCP 服务器配置

用户必须启用 DHCP 服务器并配置 DHCP 服务器参数、DHCP 地址和租用时间。

若要启动 DHCP 服务器调用：

```
sl_NetAppStart(SL_NET_APP_DHCP_SERVER_ID);
```

其中 SL_NET_APP_DHCP_SERVER_ID 是 2。

若要停止 DHCP 服务器调用：

```
sl_NetAppStop(SL_NET_APP_DHCP_SERVER_ID);
```

默认值：DHCP 服务器已启用。

若要配置 DHCP 参数，请使用以下 API：

```
sl_NetAppSet( unsigned char AppId ,
              unsigned char Option,
              unsigned char OptionLen,
              unsigned char *pOptionValue)
```

- **AppId** – 应设置为 SL_NET_APP_DHCP_SERVER_ID (2)
- **Option** – 应设置为 NETAPP_SET_DHCP_SRV_BASIC_OPT (0)
- **OptionLen** – 应为参数大小（以字节为单位）
- **pOptionValue** – 指向包含参数的存储器的指针

此示例显示了如何配置参数（起始 IP 地址 9.8.7.1，结束 IP 地址 9.8.7.5，租用时间 = 1000 秒）：

```
SlNetAppDhcpServerBasicOpt_t dhcpParams;
unsigned char outLen = sizeof(SlNetAppDhcpServerBasicOpt_t);
dhcpParams.lease_time      = 1000;
dhcpParams.ipv4_addr_start = 0x09080701;
```

```
dhcpParams.ipv4_addr_last  = 0x09080705;
sl_NetAppSet(SL_NET_APP_DHCP_SERVER_ID,
             NETAPP_SET_DHCP_SRV_BASIC_OPT,
             outLen,
            (unsigned char*)&dhcpParams);
```

默认值：

lease_time	= 24 * 3600;	/* 24 小时 */
ipv4_addr_start	= 0xc0a80102;	/* 192.168.1.2 */
ipv4_addr_last	= 0xc0a801fe;	/* 192.168.1.254 */

备注

DHCP 服务器地址必须在 AP IP 地址的子网中。更改将在复位后生效。

9.7 设置器件 URN

若要设置器件名称，请调用：

```
sl_NetAppSet (SL_NET_APP_DEVICE_CONFIG_ID,
              NETAPP_SET_GET_DEV_CONF_OPT_DEVICE_URN,
              strlen(device_urn),
             (unsigned char *) device_urn);
Where SL_NET_APP_DEVICE_CONFIG_ID       = 16
NETAPP_SET_GET_DEV_CONF_OPT_DEVICE_URN  = 0
```

默认值：mysimplelink

9.8 发送到主机的异步事件

当工作站重新连接到 AP 或从 AP 断开连接时，系统会向主机发送事件。

事件操作码为：

SL_OPCODE_WLAN_STA_CONNECTED 0x082E

SL_OPCODE_WLAN_STA_DISCONNECTED 0x082F

事件包括表 9-2 中所示的参数。

表 9-2. 事件参数

参数	字节	备注
对等器件名称	32	与 P2P 相关
对等 MAC 地址	6	
对等器件名称长度	1	
WPS 器件密码 ID	1	0 - 不可用
自有 SSID	32	与 P2P 相关
自有 SSID 长度	1	
填充	3	

当向工作站释放 IP 地址时，系统会向主机发送事件。

事件操作码为：

SL_OPCODE_NETAPP_IP_LEASED 0x 182C

事件包括表 9-3 中所示的参数。

表 9-3. 事件参数

参数	字节	备注
IP 地址	4	
租用时间	4	以秒为单位
对等 MAC 地址	6	
填充	2	

当工作站释放 IP 地址时，系统会向主机发送事件。

事件操作码为：

SL_OPCODE_NETAPP_IP_RELEASED 0x 182D

事件包括表 9-4 中所示的参数。

表 9-4. 事件参数

参数	字节	备注
IP 地址	4	
对等 MAC 地址	6	
Reason（原因）	2	0 – 对等器件释放 IP 地址 1 – 对等器件拒绝此 IP 地址 2 – 租用已到期

9.9 示例代码

下面是配置 AP WLAN 参数和网络参数（IP 地址和 DHCP 参数）的示例代码。WLAN 参数也会读回。

```
int main()
{
    int SockID;
    unsigned char outLen = sizeof(SlNetAppDhcpServerBasicOpt_t);
    unsigned char   channel, hidden, dtim, sec_type, wps_state, ssid[32],
         password[65], country[3];
    unsigned short  beacon_int, config_opt, config_len;
    SlNetAppDhcpServerBasicOpt_t dhcpParams;
    _NetCfgIpV4Args_t ipV4;
    sl_Start(NULL, NULL, NULL);
    Sleep(100);
    // Set AP IP params
        ipV4.ipV4 =
SL_IPV4_VAL(192,168,1,1);
        ipV4.ipV4Gateway =
SL_IPV4_VAL(192,168,1,1);
        ipV4.ipV4DnsServer = SL_IPV4_VAL(192,168,1,1);
        ipV4.ipV4Mask = SL_IPV4_VAL(255,255,255,0);
        sl_NetCfgSet( SL_IPV4_AP_P2P_GO_STATIC_ENABLE,
                    1
                      ,sizeof(_NetCfgIpV4Args_t),
                     (unsigned char *)&ipV4);
        //Set AP mode
        sl_WlanSetMode(ROLE_AP);
        //Set AP SSID
        sl_WlanSet(SL_WLAN_CFG_AP_ID, WLAN_AP_OPT_SSID, strlen("cc_ap_test1"),
                    (unsigned char *)"cc_ap_test1");
        //Set AP country code
        sl_WlanSet(SL_WLAN_CFG_GENERAL_PARAM_ID,
              WLAN_GENERAL_PARAM_OPT_COUNTRY_CODE, 2,(unsigned char *)"US");
         //Set AP Beacon interval
         beacon_int = 100;
        sl_WlanSet(SL_WLAN_CFG_AP_ID, WLAN_AP_OPT_BEACON_INT, 2, (unsigned char *)
                    &beacon_int);
        //Set AP channel
        channel = 8;
        sl_WlanSet(SL_WLAN_CFG_AP_ID, WLAN_AP_OPT_CHANNEL, 1, (unsigned char *)
                    &channel);
        //Set AP hidden/broadcast configuraion
```

```
        hidden = 0;
        sl_WlanSet(SL_WLAN_CFG_AP_ID, WLAN_AP_OPT_HIDDEN_SSID, 1, (unsigned char *)
                   &hidden);
        //Set AP DTIM period
        dtim = 2;
        sl_WlanSet(SL_WLAN_CFG_AP_ID, WLAN_AP_OPT_DTIM_PERIOD, 1, (unsigned char *)
                   &dtim);
         //Set AP security to WPA and password
        sec_type = SL_SEC_TYPE_WPA;
        sl_WlanSet(SL_WLAN_CFG_AP_ID, WLAN_AP_OPT_SECURITY_TYPE, 1, (unsigned char *)
                   &sec_type);
        sl_WlanSet(SL_WLAN_CFG_AP_ID, WLAN_AP_OPT_PASSWORD,
                   strlen("password123"), (unsigned char *)"password123");
        sl_Stop(100);
                   sl_Start(NULL, NULL, NULL);
        //Retrive all params to confirm setting
        //Get AP SSID
        sendLog("**********************AP parameters**********************\n");
        config_opt = WLAN_AP_OPT_SSID;
        config_len = MAXIMAL_SSID_LENGTH;
        sl_WlanGet(SL_WLAN_CFG_AP_ID,
                    &config_opt , &config_len, (unsigned char*) ssid);
        sendLog("SSID: %s\n",ssid);
        //Get AP country code
        config_opt = WLAN_GENERAL_PARAM_OPT_COUNTRY_CODE;
        config_len = 3;
        sl_WlanGet(SL_WLAN_CFG_GENERAL_PARAM_ID,
                   &config_opt, &config_len,(unsigned char*) country);
        sendLog("Country code: %s\n",country);
        //Get AP beacon interval
        config_opt = WLAN_AP_OPT_BEACON_INT;
        config_len = 2;
        sl_WlanGet(SL_WLAN_CFG_AP_ID,
                   &config_opt, &config_len, (unsigned char*) &beacon_int);
        sendLog("Beacon interval: %d\n",beacon_int);
         //Get AP channel
        config_opt = WLAN_AP_OPT_CHANNEL;
        config_len = 1;
        sl_WlanGet(SL_WLAN_CFG_AP_ID,
                &config_opt, &config_len, (unsigned char*) &channel);
        sendLog("Channel: %d\n",channel);
        //Get AP hidden configuraion
        config_opt = WLAN_AP_OPT_HIDDEN_SSID;
        config_len = 1;
        sl_WlanGet(SL_WLAN_CFG_AP_ID,
                &config_opt, &config_len, (unsigned char*) &hidden);
        sendLog("Hidden: %d\n",hidden);
        //Get AP DTIM period
        config_opt = WLAN_AP_OPT_DTIM_PERIOD;
        config_len = 1;
          sl_WlanGet(SL_WLAN_CFG_AP_ID,
                   &config_opt, &config_len, (unsigned char*) &dtim);
        sendLog("DTIM period: %d\n",dtim);
        //Get AP security type
        config_opt = WLAN_AP_OPT_SECURITY_TYPE;
        config_len = 1;
        sl_WlanGet(SL_WLAN_CFG_AP_ID,
                 &config_opt, &config_len, (unsigned char*) &sec_type);
        sendLog("Security type: %d\n",sec_type);
        //Get AP password
        config_opt = WLAN_AP_OPT_PASSWORD;
        config_len = 64;
        sl_WlanGet(SL_WLAN_CFG_AP_ID,
                 &config_opt, &config_len, (unsigned char*) password);
        sendLog("Password: %s\n",password);
       //Get AP WPS state
        config_opt = WLAN_AP_OPT_WPS_STATE;
        config_len = 1;
        sl_WlanGet(SL_WLAN_CFG_AP_ID,
                 &config_opt, &config_len, (unsigned char*) &wps_state);
        // Set AP DHCP params
        //configure dhcp addresses to: 192.168.1.10 - 192.168.1.20, lease time
        4096 seconds
        dhcpParams.lease_time = 4096;
        dhcpParams.ipv4_addr_start =  SL_IPV4_VAL(192,168,1,10);
        dhcpParams.ipv4_addr_last = SL_IPV4_VAL(192,168,1,20);
```

```
        outLen = sizeof(SlNetAppDhcpServerBasicOpt_t);
        sl_NetAppStop(SL_NET_APP_DHCP_SERVER_ID);
        sl_NetAppSet(SL_NET_APP_DHCP_SERVER_ID, NETAPP_SET_DHCP_SRV_BASIC_OPT,
                outLen, (unsigned char*)&dhcpParams);
        sl_NetAppStart(SL_NET_APP_DHCP_SERVER_ID);
        // Get AP DHCP params
        sl_NetAppGet(SL_NET_APP_DHCP_SERVER_ID, NETAPP_SET_DHCP_SRV_BASIC_OPT,
                &outLen, (unsigned char*)&dhcpParams);
}
```

章节 10

对等 (P2P) 模式

10.1 一般说明

10.1.1 范围

SimpleLink WiFi 器件中的 P2P 模式/角色是一个强大的功能，使器件无需 AP 而是通过继承整个工作站的 AP 属性即可连接到其他器件，并且这种连接具有更多的功能和优点，如省电。

10.1.2 Wi-Fi Direct 优势

- 无论用户身处何处（家庭、公共场所、旅行途中、工作场所），都可以方便快捷地共享、显示、打印和同步内容。
- 提供新的连接场景，包括用户已经拥有的数以亿计的 Wi-Fi 认证器件。
- 消除对路由器的需求，因为它用于局域网，并且还可能取代应用（不依赖于低功耗）对蓝牙的需求。
- 基于广泛部署的 Wi-Fi 技术提供满足整个行业需求的点对点解决方案。
- 借助对等机会性节能和缺失通知功能，便能够在双方（组所有者和客户端）之间建立完整的低功耗链路，与传统的 AP-STA 链路相比，这是一个很大的优势。

10.1.3 Wi-Fi Direct 支持和功能

- P2P 配置器件名称、器件类型、侦听和操作通道
- P2P 器件发现（完整/社会）
- 全方位的 P2P 协商（0 至 15）
- P2P 协商发起方策略 – 主动/被动/随机退避
- P2P WPS 方法按钮和 PIN 码（键盘和显示屏）
- P2P 用作客户端角色：
  - P2P 器件可以加入现有的 P2P 组。
  - P2P 器件可邀请重新连接持久组（快速连接）。
- P2P 担任组所有者的角色：
  - P2P 组所有者可以接受加入请求。
  - P2P 持久组所有者可以响应邀请请求。
- 删除 P2P 组
- P2P 的“连接-断开-连接”转换，也在不同角色（例如 GO-CL-GO）之间转换
- P2P 客户端旧版 PS 和 NoA 支持
- P2P 角色具有独立的 IP 配置
- 在 P2P-CL/GO 角色之上具有单独的 Net-Apps 配置

10.1.4 限制

- 不支持服务发现
- 不支持 GO-NoA
- P2P 器件条目的智能配置限制为 15 个条目，并且执行动态分配，而不是作为工作站扫描单个条目进行优化。
- 不支持自主式组
- P2P 组所有者模式支持单点（客户端）连接（类似于 AP）。
- 配置文件搜索和连接中的 P2P 查找是无限的，意味着如果找不到远程器件，搜索将无限期地进行下去。

10.2 P2P API 和配置

P2P 配置使用主机驱动程序 API 来控制 SimpleLink WiFi 器件。可以使用特定的 API 访问 P2P 配置的不同选项和模式。通过使用特定的 API，用户可以设置 P2P 功能并使用它们。配置分为几个简单的部分。某些部分的配置会影响其他部分的配置。

配置分为以下几个部分：

- 配置 P2P 全局参数
- 配置 P2P 策略
- 配置 P2P 配置文件策略
- 配置手动连接

- 配置如何搜索并在上述步骤中使用 P2P 器件
- P2P 事件

下一节将说明如何配置 P2P 以及如何使用其选项。注意：

备注

- 可以刷新配置，并且只能进行一次。
- 并不需要使用所有 API。在缺少用户配置时会使用默认参数。
- 所有设置的 API 都有一个 GET 操作。

10.2.1 配置 P2P 全局参数

此节介绍了如何将该器件配置为 P2P 状态并设置一般参数。此节还介绍了 P2P 角色、P2P 网络参数和 P2P 器件名称、器件类型和通道的配置。

10.2.1.1 设置 P2P 角色

使用 API 将器件设置为 P2P 模式：

```
sl_WlanSetMode(ROLE_P2P)
```

此 API 将器件置于 P2P 模式。在进入 P2P 模式之后，所有其他 P2P 配置才会生效。

在进入 P2P 模式之前，所有其他 P2P 配置都不会生效。

10.2.1.2 设置 P2P 网络配置

配置 P2P API 使用的网络配置：

- P2P 客户端（与工作站的 API 相同）：
  - 静态 IP：
    - ```
      sl_NetCfgSet(SL_IPV4_STA_P2P_CL_STATIC_ENABLE,1, sizeof(_NetCfgIpV4Args_t),(unsigned char *)&ipV4)
      ```
  - DHCP 客户端：
    - ```
      sl_NetCfgSet(SL_IPV4_STA_P2P_CL_DHCP_ENABLE,1,1,&val);
      ```
- P2P GO（与 AP 的 API 相同）：
  - GO 自有的静态 IP：
    - ```
      sl_NetCfgSet(SL_IPV4_AP_P2P_GO_STATIC_ENABLE,1, sizeof(_NetCfgIpV4Args_t),(unsigned char *)&ipV4)
      ```

此 API 可设置在设置 P2P 角色时使用的网络配置。

10.2.1.3 设置 P2P 器件名称

以下宏指令将设置 P2P 器件名称。必须为每个单独的器件设置唯一的 P2P 器件名称，因为连接是基于器件名称的。

Default: TI_SIMPLELINK_P2P_xx（xx = 随机的两个字符）

API：

```
sl_NetAppSet (SL_NET_APP_DEVICE_CONFIG_ID,
              NETAPP_SET_GET_DEV_CONF_OPT_DEVICE_URN,
              strlen(device_name),
              (unsigned char *) device_name);
```

10.2.1.4 设置 P2P 器件类型

以下宏指令将设置 P2P 器件类型。器件类型使 P2P 发现参数能够识别器件。
```

Default: 1-0050F204-1

API：

```
sl_WlanSet(SL_WLAN_CFG_P2P_PARAM_ID,
          WLAN_P2P_OPT_DEV_TYPE,
           dev_type_len, dev_type);
```

10.2.1.5 设置 P2P 侦听通道和运行通道

以下宏将设置 P2P 运行通道和侦听通道。侦听通道用于指示发现状态，可以是通道 1、6 或 11。等待 P2P 探针请求时，器件将处于此通道中。运行通道仅供 GO 使用。GO 将在协商阶段后转移到此通道。

默认值： 通道 1、6 或 11 之间的随机值

API：

```
sl_WlanSet(SL_WLAN_CFG_P2P_PARAM_ID, WLAN_P2P_OPT_CHANNEL_N_REGS, 4, channels);
```

备注

2.4G 下的监管域等级应该是 81。

例如：

```
unsigned char  channels [4];
channels [0] = (unsigned char)11; // listen channel
channels [1] = (unsigned char)81; // listen regulatory class
channels [2] = (unsigned char)6;  // oper channel
channels [3] = (unsigned char)81; // oper regulatory class
sl_WlanSet(SL_WLAN_CFG_P2P_PARAM_ID, WLAN_P2P_OPT_CHANNEL_N_REGS,  4, channels);
```

10.2.2 配置 P2P 策略

本节描述了 P2P 策略配置，包括 simplelink 器件给出的另外两个 P2P 工作模式选项。

- P2P 意图值选项 – 器件的 P2P 角色 (客户端、GO 或无关紧要) 。
- 协商引发器选项 – 在 P2P 协商期间使用的值用于指示哪一方将发起协商，哪一方将被动地等待远端发送协商，然后进行响应。

10.2.2.1 配置 P2P 意图值和协商引发器

此配置使用宏 SL_P2P_POLICY，第二个参数被发送到函数 *sl_WlanPolicySet*。

对于意图值，可以使用三个定义选项：

- SL_P2P_ROLE_CLIENT (intent 0): 表示器件强制为 P2P 客户端。
- SL_P2P_ROLE_NEGOTIATE (intent 7): 表示器件可以是 P2P 客户端或 GO，具体取决于 P2P 协商决胜属性。这是系统默认值。
- SL_P2P_ROLE_GROUP_OWNER (intent 15): 表示器件强制为 P2P GO。

对于协商引发器，可以使用三个定义选项：

- SL_P2P_NEG_INITIATOR_ACTIVE：当执行发现操作后找到远程对等器件时，器件会立即向对等器件发送协商请求。
- SL_P2P_NEG_INITIATOR_PASSIVE：当执行发现操作后找到远程对等器件时，器件会被动地等待对等器件启动协商，之后才会响应。
- SL_P2P_NEG_INITIATOR_RAND_BACKOFF：当执行发现操作后找到远程对等器件时，器件会触发随机计时器 (1 至 7 秒) 。在此期间，器件会被动地等待对等器件进行协商。如果计时器在没有协商的情况下到期，器件会立即向对等器件发送协商请求。这是系统默认设置，因为两个 simplelink 器件不需要任何协商同步。

处理两个 simplelink 器件时，使用此配置。用户可能没有用于启动协商的 GUI，因此提供此选项是为了防止在执行发现操作后两个器件同时协商。

API：

```
sl_WlanPolicySet(SL_POLICY_P2P,
                 SL_P2P_POLICY(Intent value, negotiation initiator)//macro,
                 &policyVal,
                 0
                 );
```

例如：

```
sl_WlanPolicySet(SL_POLICY_P2P,
                 SL_P2P_POLICY(SL_P2P_ROLE_NEGOTIATE,
                 SL_P2P_NEG_INITIATOR_RAND_BACKOFF
                              ) //macro,
                 &policyVal,
                 0
                 );
```

10.2.3 配置 P2P 配置文件连接策略

本节讨论了配置文件连接策略。此策略使系统可以在不复位或由远程对等器件执行断开操作的情况下连接至对等器件。

该机制说明了器件如何在与 P2P 自动连接相关的情况下使用这些配置文件。节 10.2.6 中还说明了手动连接。

对等配置文件和对等配置文件配置遵循一种一般机制，本文档未对此进行介绍。节 10.2.8 提供了有关如何添加配置文件的示例。

此配置使用宏 SL_CONNECTION_POLICY，第二个参数被发送到函数 *sl_WlanPolicySet*。

共有四种连接策略选项：

- 自动启动 – 与在 STA 模式中一样，如果器件未连接，它会启动 P2P 查找以搜索在器件上配置的所有 P2P 配置文件。如果找到至少一个备选件，则器件会尝试连接到它。如果找到多个器件，则会根据配置文件参数选择最优的备选件。
- 快速连接 – 在 P2P 角色中，它等效于 P2P 持久组，但在 GO 和 CL 之间具有不同的含义。在复位后进行快速连接时，此选项非常有用，但使用时要注意，因为它取决于上一次连接的状态。因为快速连接选项保存和使用上一次连接的参数，如果器件之前未进行连接，则此选项没有意义。
  - 如果在上一次连接中器件是 P2P 客户端（在复位或远程断开连接操作之前），则在复位后，它会向以前连接的 GO 发送 p2p_invite 以执行快速重新连接。
  - 如果在上一次连接中器件是 P2P GO（在复位或远程断开连接操作之前），则在复位后，它会重新调用 p2p_group_owner 并等待之前连接的对等器件重新连接。
- OpenAP – 与 P2P 模式无关
- AnyP2P 策略 – 与发现期间找到的任何 P2P 对等器件建立连接的策略值。此选项不需要配置文件。仅与带有按钮的协商相关。

每个选项应在此宏中发送或设置为 true 或 false。可以使用多个选项；例如，用户可以将自动启动和快速连接选项设置为 true。

API：

```
sl_WlanPolicySet(SL_POLICY_P2P,
                 SL_CONNECTION_POLICY(auto start, fast connect,openAp,AnyP2p) //macro,
                 &policyVal,
                 0
                 );
```

例如：

```
sl_WlanPolicySet(SL_POLICY_P2P,
                 SL_CONNECTION_POLICY(true, true, false, false) //macro,
                 &policyVal,
                 0
                 );
```

10.2.4 发现远程 P2P 对等器件

此节介绍了如何启动 P2P 搜索和发现，以及如何查看发现的远程 P2P 器件。

发现用于：

- 扫描和查找附近的器件，因为 P2P 连接基于在发现阶段发布的远程器件名称
- 通过主机命令手动连接（不使用现有配置文件）
- 如果邻域未知且用户希望在系统中设置 P2P 配置文件，则发现邻域中的远程对等器件，然后配置意愿配置文件。

10.2.4.1 如何启动 P2P 发现

需要通过一个扫描策略来启动 P2P 查找和发现远程 P2P 对等器件。将扫描策略设置为 P2P 执行完整的 P2P 扫描。

扫描策略的设置应在 P2P 角色下进行。P2P 发现作为任何连接的一部分执行，但也可以使用 SCAN_POLICY 激活。

备注

- 扫描策略的设置应在 P2P 角色下进行。
- P2P 发现也作为任何连接的一部分执行，但也可以使用 SCAN_POLICY 激活。

API：

```
SL_WlanPolicySet(SL_POLICY_SCAN, 1 /*enable scan*/, interval, 0)
```

第二个参数用于启用 P2P 扫描操作，interval 表示 P2P 查找周期之间的等待时间。

10.2.4.2 如何查看/获取 P2P 远程对等器件（网络 P2P 列表）

有以下两种方法可以查看和获取在 P2P 查找和搜索操作期间发现的 P2P 远程器件：

- 侦听事件 SL_WLAN_P2P_DEV_FOUND_EVENT
- 调用 API *sl_WlanGetNetworkList*

SL_WLAN_P2P_DEV_FOUND_EVENT：此事件将发送到所找到的每个远程 P2P。它包含远程器件的 MAC 地址、名称和名称长度。通过侦听此事件，用户可以找到邻域中的每个远程 P2P。

Sl_WlanGetNetworkList：通过调用此 API，用户可以获得一个列表，其中包含找到并保存在器件缓存中的远程对等器件。此 API 也用于工作站模式。

API：

```
sl_WlanGetNetworkList(unsigned char Index, unsigned char Count,
                      Sl_WlanNetworkEntry_t *pEntries)
```

Index – 指示 P2P 器件返回到的列表中的索引。

Count – 显示应返回多少对等器件。

pEntries – 将结果输入到此处，此处由用户分配。

10.2.5 协商方法

接下来的章节将介绍如何通过配置文件手动或自动建立 P2P 连接，以及 WPS 连接之前的协商方法。

如节 10.2.3 所述，根据意图和协商发起方参数启动协商，但需要配置其他参数才能成功完成此步骤。在执行来自主机的手动连接 API 命令期间或在设置自动连接配置文件时提供这些参数，它们会影响协商方法。协商方法由器件完成，无需用户干预。

共有两种 P2P 协商方法来指示协商之后的 WPS 阶段：

- P2P 按钮式连接 – 两端采用 PBC 方法进行协商。定义 SL_SEC_TYPE_P2P_PBC。
- P2P PIN 码连接 – 分为两个选项。PIN_DISPLAY 查找要由其远程 P2P 写入的 PIN。PIN_KEYPAD 向其远程 P2P 发送一个 PIN 码。
  - 定义 SL_SEC_TYPE_P2P_PIN_KEYPAD。
  - 定义 SL_SEC_TYPE_P2P_ PIN_DISPLAY。

如果没有输入 PIN 码，NWP 会使用以下方法从器件 MAC 自动生成 PIN 码：

1. 取器件 MAC 地址中的 7 位 ISB 十进制数字，并将这 7 位数字的校验和添加到 LSB（共 8 位）。例如，如果 MAC 为 03:4A:22:3B:FA:42
2. 转换为十进制：…:059:250:066
3. 七个 ISB 十进制数字是：9250066
4. WPS PIN 校验和位：2
5. 此 MAC 的默认 PIN 码：92500662

共有两个选项可以配置协商方法：

- 通过手动连接命令，在 secParams 结构中设置该值并将其作为参数发送。
  - 对于按钮式：secParams.Type = SL_SEC_TYPE_P2P_PBC
  - 对于 PIN 码键盘：secParams.Type = SL_SEC_TYPE_ PIN_KEYPAD secParams.Key = 12345670
  - 对于 PIN 码显示屏：secParams.Type = SL_SEC_TYPE_ PIN_ DISPLAY secParams.Key = 12345670
- 通过 P2P 配置文件配置，将协商方法定义和密钥作为参数发送。

10.2.6 手动连接 P2P

通过获取事件 SL_WLAN_P2P_DEV_FOUND_EVENT 或调用 **get_networkList** API 找到远程器件后，可以选择使用主机发出的命令来启动连接。该命令执行即时 P2P 发现。一旦找到远程器件，便根据所选的协商发起方策略、方法和意图启动协商阶段。

备注

- 此连接未刷新，因此在断开连接或复位的情况下，只有在启用快速连接策略时才会重新连接。
- 此连接比通过配置文件建立的连接更强，因为系统中已经存在 P2P 连接。当前连接断开，以为手动连接提供便利。

API：

```
sl_WlanConnect(char* pName, int NameLen, unsigned char *pMacAddr,
               SlSecParams_t* pSecParams ,
               SlSecParamsExt_t* pSecExtParams)
```

pName – 在获取事件 SL_WLAN_P2P_DEV_FOUND_EVENT 或调用 **get_networkList** API 之后用户已知的远程器件的名称。

NameLen – pName 的长度

pMacAddr – 根据 BSSID 连接到远程 P2P 的选项。使用 {0,0,0,0,0,0} 根据 MAC 地址来连接。

pSecParams – 请参阅节 10.2.5。

pSecExtParams – 值应为 0。

例如：

```
sl_WlanConnect("my-tv-p2p-device, 17, {0,0,0,0,0,0}
               & pSecParams ,0);
```

10.2.7 手动断开 P2P 连接

手动断开连接选项使用户能够通过主机命令断开与对等器件的连接。该命令执行当前活动角色的 P2P 组移除，无论该角色是 p2p-device、p2p-group-owner 还是 p2p-client。

API：

```
sl_WlanDisconnect();
```

10.2.8 P2P 配置文件

在重置远程对等器件或与远程对等器件断开连接后，配置文件配置可自动建立 P2P 连接。此命令将 P2P 远程器件参数作为新配置文件连同配置文件优先级一起存储在闪存中。这些配置文件类似于工作站配置文件并具有相同的自动连接行为。该连接取决于配置文件策略配置。

如果启用了自启动策略，则会执行 P2P 发现。如果找到一个或多个符合配置文件的远程器件，则会根据所选的协商发起方策略、方法和意图启动协商阶段。此时会选择优先级最高的配置文件。

备注

- 如果启用了快速连接策略并且已经存在连接，则在重置远程对等器件或与远程对等器件断开连接后，将根据上次的连接参数进行快速连接，无需查询 P2P 配置文件列表。
- 如果发送了手动连接，则通过断开连接命令停止配置文件连接，并启动手动连接。

若要使用按钮协商方法来添加配置文件，请调用：

```
sl_WlanProfileAdd(char* pName, int NameLen, unsigned char *pMacAddr,
                  SlSecParams_t* pSecParams , SlSecParamsExt_t*
                  pSecExtParams, unsigned long  Priority,
                  unsigned long  Options)
```

以下是使用按钮协商方法添加配置文件的示例：

```
sl_WlanProfileAdd(SL_SEC_TYPE_P2P_PBC,
                  remote_p2p_device,
                  strlen(remote_p2p_device),
                  bssidEmpty,
                  0/*Priority*/,0,0,0);
```

以下是使用键盘协商方法添加配置文件的示例：

```
sl_WlanProfileAdd(SL_SEC_TYPE_P2P_PIN_ DISPLAY,
                  remote_p2p_device,
                  strlen(remote_p2p_device),
                  bssidEmpty,
                  0/*Priority*/,key,8/*keylen*/,0);
```

10.2.9 删除 P2P 配置文件

若要删除特定配置文件或所有配置文件，请使用以下 API：

```
sl_WlanProfileDel(profile_index/*WLAN_DEL_ALL_PROFILES for all profiles*/)
```

10.3 P2P 连接事件

本节介绍了 P2P 连接事件。这些事件由器件在连接期间发送，并由用户决定如何处理它们。

SL_WLAN_P2P_NEG_REQ_RECEIVED_EVENT – 如果成功完成协商，则会发送此事件。

它包含：

- 器件对等 MAC 地址
- 器件对等名称
- 器件对等名称的长度
- 器件对等 WPS 密码 ID

SL_WLAN_CONNECTION_FAILED_EVENT – 如果连接失败，则会发送此事件，并说明失败原因。

它包含：

- 失效原因

SL_WLAN_CONNECT_EVENT – P2P 客户端和工作站将共享此事件。指示 P2P 连接已成功完成，器件为 P2P 客户端。

它包含：

- 远程器件参数

SL_WLAN_CONNECT_EVENT – P2P GO 和 AP 将共享此事件。指示 P2P 连接已成功完成，器件为 P2P GO。

它包含：

- 远程器件参数
- 有关组所有者参数的信息

10.4 用例和配置

本节介绍了常见的 P2P 用例、它们的含义以及如何配置它们。

10.4.1 案例 1 – 固定式 P2P 客户端低功耗配置文件

该器件是自动连接的 P2P 客户端，每次复位或断开连接后快速连接。该器件可存储上一次连接的参数。

应在系统中配置至少一个配置文件。

在快速连接时，将器件添加到 P2P 请求方以避免查找。

使用网络参数将 P2P-Invite 发送至持久型远程 GO。

如果快速连接失败，退回到配置文件 P2P 查找并执行常规连接方法（加入或完整协商）。

1. 配置 P2P 全局参数。
2. 使用 SL_P2P_ROLE_CLIENT（意图 0）和 SL_P2P_NEG_INITIATOR_RAND_BACKOFF（协商发起方无关）配置 P2P 配置文件策略。SL_P2P_POLICY (SL_P2P_ROLE_CLIENT, SL_P2P_NEG_INITIATOR_RAND_BACKOFF)
3. 将 P2P 配置文件策略连接配置为自动启动和快速连接模式。SL_CONNECTION_POLICY(true, true, false, false)
4. 根据远程器件的意愿参数（例如，名称、MAC 地址等）配置 P2P 配置文件。
5. 如果对等参数未知，请执行发现 P2P 操作，查找 P2P 对等器件，然后对配置文件进行配置。

10.4.2 案例 2 – 移动客户端低功耗配置文件

该器件是自动连接的 P2P 客户端，每次复位或断开连接后不使用快速连接。该器件不存储上次连接的参数。

应在系统中配置至少一个配置文件。

与第一种情况一样，使用存储的网络执行配置文件 P2P 查找，检查结果，然后根据远程器件 GO 能力（组所有者）发送协商请求。

1. 配置 P2P 全局参数。
2. 使用 SL_P2P_ROLE_CLIENT（意图 0）和 SL_P2P_NEG_INITIATOR_RAND_BACKOFF（协商发起方无关）配置 P2P 配置文件策略。SL_P2P_POLICY (SL_P2P_ROLE_CLIENT, SL_P2P_NEG_INITIATOR_RAND_BACKOFF)
3. 将 P2P 配置文件策略连接配置为自动启动和快速连接模式。SL_CONNECTION_POLICY(true, false, false, false)
4. 根据远程器件的意愿参数（例如，名称、MAC 地址等）配置 P2P 配置文件
5. 如果对等参数未知，请执行发现 P2P 操作，查找 P2P 对等器件，然后对配置文件进行配置。

10.4.3 案例 3 – 固定的中心插入式配置文件

该器件是自动连接的 P2P GO，每次复位或断开连接后快速连接。该器件可存储上次连接的参数。

应在系统中配置至少一个配置文件。

- HARD RESET ONLY – 启动组（作为 GO），将以前的持久性网络参数存储在快速连接中，设置 WPS 方法和设置计时器。
- PEER DISCONNECT – 设置 WPS 方法和设置计时器。
- WPS 计时器过期，未连接对等器件 – 拆开组，返回并使用存储的网络执行配置文件 P2P 查找。检查结果并使用策略参数发起协商请求（与意图 = 15 协商，结果为 GO）。

1. 配置 P2P 全局参数。
2. 使用 SL_P2P_ROLE_GROUP_OWNER（意图 15）和 SL_P2P_NEG_INITIATOR_RAND_BACKOFF（协商发起方无关）配置 P2P 配置文件策略。SL_P2P_POLICY(SL_P2P_ROLE_GO, SL_P2P_NEG_INITIATOR_RAND_BACKOFF)
3. 将 P2P 配置文件策略连接配置为自动启动和快速连接模式。SL_CONNECTION_POLICY(true, true, false, false)
4. 根据远程器件的意愿参数（例如，名称、MAC 地址等）配置 P2P 配置文件。
5. 如果对等参数未知，请执行发现 P2P 操作，查找 P2P 对等器件，然后对配置文件进行配置。

10.4.4 案例 4 – 移动中心配置文件

建立组并通过自动连接启动为 GO（非持久），每次复位或断开连接后不进行快速连接。该器件不存储上一次连接的参数。应在系统中配置至少一个配置文件。

- DISCONNECT – 立即拆开组。
- 硬复位或拆开后 – 使用存储的网络执行配置文件 P2P 查找，检查结果并使用策略参数发起协商请求（与意图 = 15 协商，结果为 GO）。

1. 配置 P2P 全局参数。
2. 使用 SL_P2P_ROLE_GROUP_OWNER（意图 15）和 SL_P2P_NEG_INITIATOR_RAND_BACKOFF（协商发起方无关）配置 P2P 配置文件策略。SL_P2P_POLICY(SL_P2P_ROLE_GO, SL_P2P_NEG_INITIATOR_RAND_BACKOFF)
3. 将 P2P 配置文件策略连接配置为自动启动和快速连接模式。SL_CONNECTION_POLICY(true, false, false, false)
4. 根据远程器件的意愿参数（例如，名称、MAC 地址等）配置 P2P 配置文件。
5. 如果对等参数未知，请执行发现 P2P 操作，查找 P2P 对等器件，然后对配置文件进行配置。

10.4.5 案例 5 – 移动通用型配置文件

建立组并启动为 GO（非持久型）或 CL，不存储任何参数。应在系统中配置至少一个配置文件。

- DISCONNECT – 如果为 GO，立即拆开组。
- 硬复位或拆开后 – 使用存储的网络执行配置文件 P2P 查找，检查结果并使用策略参数发起协商请求（与意图 = 7 协商，结果为 GO 或 CL）。

1. 配置 P2P 全局参数。
2. 使用 SL_P2P_ROLE_NEGOTIATE（意图 7）和 SL_P2P_NEG_INITIATOR_RAND_BACKOFF（协商发起方无关）配置 P2P 配置文件策略。SL_P2P_POLICY(SL_P2P_ROLE_GO, SL_P2P_NEG_INITIATOR_RAND_BACKOFF)
3. 将 P2P 配置文件策略连接配置为自动启动和快速连接模式。SL_CONNECTION_POLICY(true, false, false, false)
4. 根据远程器件的意愿参数（例如，名称、MAC 地址等）配置 P2P 配置文件。
5. 如果对等参数未知，请执行发现 P2P 操作，查找 P2P 对等器件，然后对配置文件进行配置。

10.5 示例代码

以下是简单配置文件连接的示例代码，使用按钮方法和具有已知远程 GO 器件名称（不含 MAC）的配置文件连接，将器件配置为充当 P2P 客户端器件。

```
unsigned char val = 1;
unsigned char policyVal;
unsigned char my_p2p_device[33];
unsigned char *remote_p2p_device = "Remote_GO_Device_XX";
unsigned char bssidEmpty[6] = {0,0,0,0,0,0};
sl_Start(NULL, NULL, NULL);
//Set P2P as active role
sl_WlanSetMode(3/*P2P_ROLE*/);
//Set P2P client dhcp enable (assuming remote GO running DHCP server)
sl_NetCfgSet(SL_IPV4_STA_P2P_CL_DHCP_ENABLE,1,1,&val);
//Set Device Name
strcpy(my_p2p_device,"jacky_sl_p2p_device");
sl_NetAppSet (SL_NET_APP_DEVICE_CONFIG_ID,
              NETAPP_SET_GET_DEV_CONF_OPT_DEVICE_URN, strlen(my_p2p_device),
             (unsigned char *) my_p2p_device);
//set connection policy Auto-Connect
sl_WlanPolicySet( SL_POLICY_CONNECTION,
                  SL_CONNECTION_POLICY(1/*Auto*/,0/*Fast*/,
                  0/*OpenAP*/,0/*AnyP2P*/),
                    &policyVal, 0 /*PolicyValLen*/
                  );
//set P2P Policy - intent 0, random backoff
sl_WlanPolicySet( SL_POLICY_P2P,
                     SL_P2P_POLICY(SL_P2P_ROLE_CLIENT/*Intent 0 - Client*/,
                  SL_P2P_NEG_INITIATOR_RAND_BACKOFF/*Negotiation initiator - random backoff*/),
                  &policyVal,0 /*PolicyValLen*/
 );

sl_WlanProfileAdd(
                  SL_SEC_TYPE_P2P_PBC,
                  remote_p2p_device,
                  strlen(remote_p2p_device),
                  bssidEmpty,
                  0, //unsigned long   Priority,
                  0,//unsigned char   *pKey,
                  0,//unsigned long   KeyLen,
                  0//unsigned long   Options)
                  );
sl_Stop(1);
sl_Start(NULL, NULL, NULL);
```

This page intentionally left blank.

HTTP 服务器

11.1 概述

HTTP 网络服务器使最终用户能够使用标准 Web 浏览器与 SimpleLink 器件进行远程通信。

HTTP 网络服务器启用以下功能：

- 器件配置
- 器件状态和诊断
- 应用特定功能

HTTP 是一种客户端/服务器协议，用于将超文本资源（HTML 网页、图像、查询结果等）传送到客户端。HTTP 工作在预定义的 TCP/IP 套接字之上，通常使用端口 80。

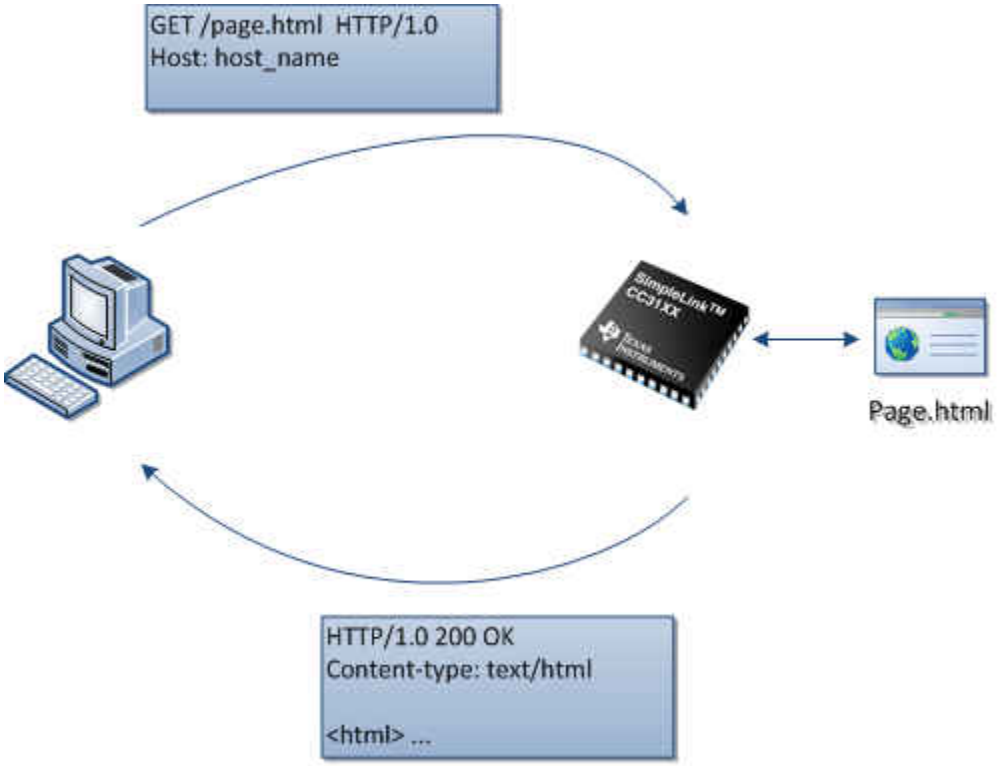


图 11-1. HTTP GET 请求

11.2 支持的功能

- 支持的 HTTP 版本：1.0
- HTTP 请求：GET、POST
- 支持的文件类型：.html、.htm、.css、.xml、.png、.gif
- 使用 POST 方法提交数据 HTML 表单
- 支持的 POST 请求内容类型：“application/x-www-form-urlencoded”
- 可配置 HTTP 端口号 – 默认为 80 端口
- HTTP 网络服务器身份验证
  - 可以启用/禁用（默认禁用）
  - 可配置身份验证名称、密码和领域
- 可配置 Simplelink 域名（AP 模式）
- 内置默认页面，其中提供了器件配置、状态和分析工具（用户端零配置...）
- 可以将器件配置设置为用户提供的页面的一部分
- 出于安全考虑，网络服务器只能访问文件系统上的以下根文件夹：
  - www/

  - www/safe/
- “强制 AP”支持模式 – 在此模式下只能访问文件夹系统上的以下根文件夹
  - 服务器仅允许访问文件系统中的“www/safe/”文件夹。
  - 内部配置网页中特殊的“Clear all Profiles”（清除所有配置文件）按钮。
- 默认值：
  - 域名：“www.mysimplelink.net”或“mysimplelink.net”
  - 身份验证名称：admin
  - 身份验证密码：admin
  - 身份验证领域：SimpleLink CC31xx

11.3 HTTP Web 服务器说明

11.3.1 概述

SimpleLink HTTP 网络服务器是在网络处理器上基于 TCP 协议栈运行的嵌入式网络应用程序。

Simplelink HTTP 网络服务器会侦听预先配置的 TCP/IP 套接字（默认为 80），并等待来自客户端的 HTTP 请求。一旦 HTTP 请求到达，服务器就会对其进行处理并提供正确的 HTTP 响应。

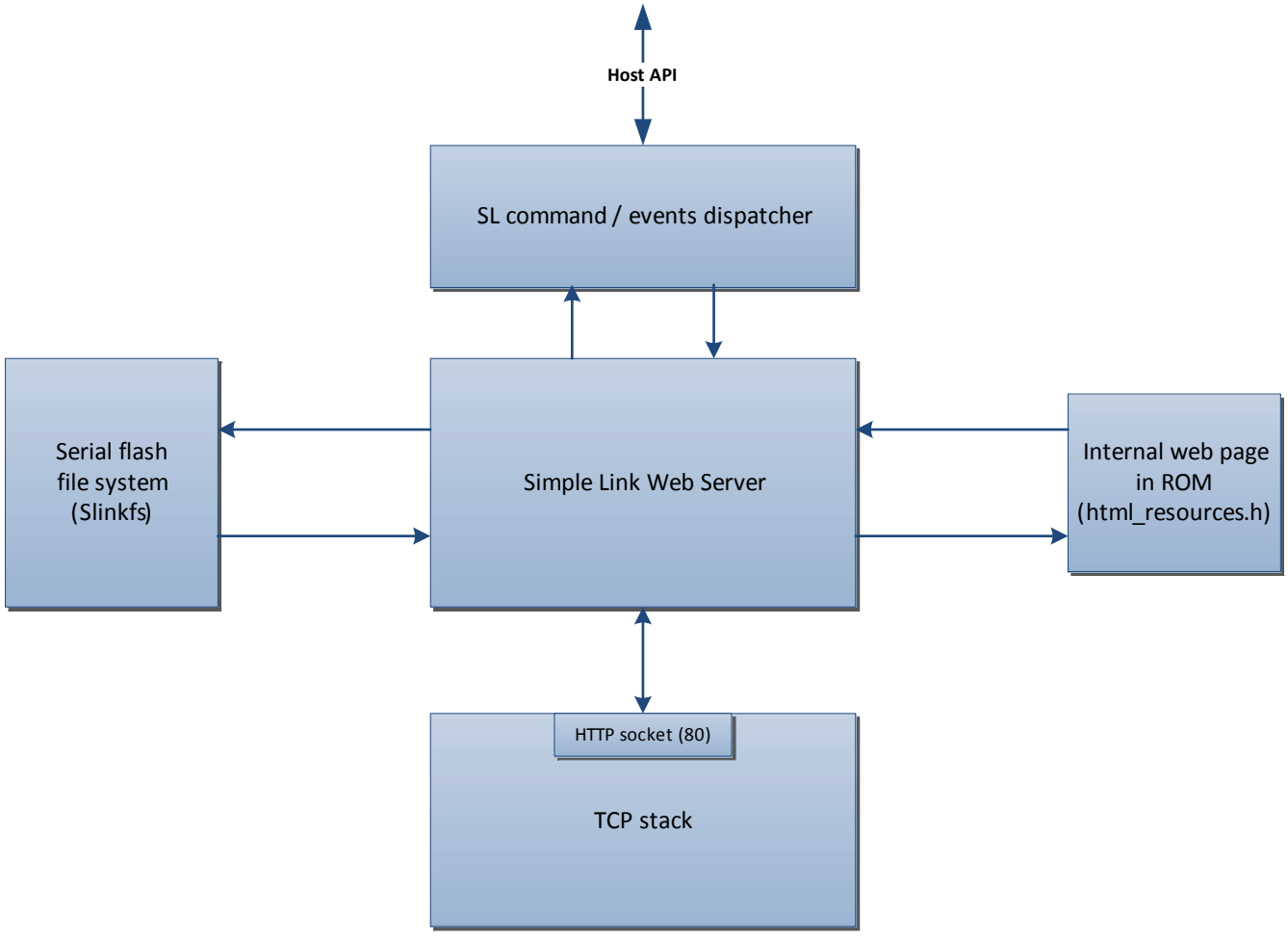


图 11-2. 简要方框图

11.3.1.1 详细信息

SimpleLink 服务器负责处理 HTTP 请求。它会侦听 HTTP 套接字（默认为 80），并根据请求类型（即 HTTP GET 或 HTTP POST）来处理对 URI 资源和内容的请求。然后，它编写适当的 HTTP 响应并发回客户端。

SimpleLink 服务器与托管网页文件的串行闪存文件系统进行通信。这些文件以自己的文件名保存在串行闪存中。文件名可以包括完整路径，以便实现类似行为的目录结构。

出于安全考虑，Web 服务器对文件系统的访问仅限于以下根文件夹：

- www/
- www/safe/

备注

将文件下载到文件系统时，应在文件名前加上以上根文件夹之一作为前缀。在“强制 AP”模式下也使用“www/safe/”文件夹。在文档后面部分可以找到详细信息。

11.4 HTTP GET 处理

11.4.1 概述

当 HTTP 网络服务器收到 HTTP GET 请求时，首先会检查 HTTP 请求的资源 URN 以获取所请求资源的名称。然后，服务器会检查此资源是否存在于串行闪存中。如果资源存在，则服务器会将其作为 HTTP 响应的一部分返回。

如果服务器在闪存中没有找到请求的资源，则服务器会检查该资源是否是 ROM 内部网页的文件之一。如果资源是该网页的文件，则服务器会返回资源；如果不是，则发送 HTTP 错误消息（HTTP/1.0 404 Not Found 响应）。

11.4.2 默认网页

如果 HTTP GET 不包含任何资源名称（例如，/），则 HTTP Web 服务器会按此顺序查找以下文件名：

- Index.html
- Main.html

服务器先在串行闪存中检查它们，然后在内部网页中检查。

11.4.3 SimpleLink GET 令牌

为了在 HTML 页面中显示由服务器动态生成的数据，HTTP 网络服务器支持一组由服务器动态替换的预定义令牌，以及动态生成的内容。

- 令牌具有 10 个字符的固定长度。
- 令牌前缀与所有令牌相同，长度为七个字符。
- 令牌前缀为：**__SL_G_**
- 一个典型的令牌应该是这样的：**__SL_G_XYZ**，其中 XYZ 是预定义的令牌之一。

作为 GET 请求的结果，HTTP 服务器将扫描 HTML 页面以查找“__SL_G_”前缀。如果找到了前缀，则会检查完整的令牌。一旦发现与某个已知令牌相匹配，便会将 HTML 中的令牌替换为与该令牌匹配的相应数据（字符串）（例如，“__SL_G_N.A”将替换为 STA IP 地址，“__SL_G_V.A”将替换为 NWP 版本）。浏览器最终将显示此字符串，而不是显示令牌。

有关预定义令牌的完整列表，请参阅节 11.12。

11.4.4 用户定义的令牌

用户可以定义 HTTP 网络服务器不了解的新令牌。

该令牌应遵循与预定义令牌相同的规则：

- 该令牌具有 10 个字符的固定长度。
- 该令牌的前缀与所有令牌相同，长度为七个字符。
- 该令牌前缀为：**__SL_G_**
- 一个典型的令牌应该是这样的：**__SL_G_XYZ**，其中 XYZ 是用户定义的，可包含任意字符或数字。

如果 HTTP 网络服务器扫描 HTML 文件并找到不在预定义列表中的令牌，则服务器会生成带有令牌名称的 **get_token_value** 异步事件，向主机请求令牌值。

主机应使用 **send_token_value** 命令和令牌值进行响应。HTTP 网络服务器将使用此令牌值并将其返回给客户端。

备注

令牌值的最大长度是 64 字节。

如果主机未响应 **get_token_value** 请求，则服务器会实施两秒的超时。超时过后，使用的令牌值为 Not Available 字符串（并最终显示在浏览器中）。

备注

为防止用户令牌与内部令牌相冲突，请使用具有以下结构的令牌：**__SL_G_UXX**，其中 XX 可以是任意字符或数字。

可以在以下段落中找到关于主机接口的说明：**主机/HTTP 网络服务器 API**。

11.4.5 带有动态 HTML 内容的 HTML 示例代码

在以下代码中，令牌 __SL_G_N.A 将替换为实际 IP 地址：

```
<tr>
  <td  dir=LTR> IP Address: </td>
  <td  dir=LTR>__SL_G_N.A  </td>
</tr>
```

11.5 HTTP POST 处理

11.5.1 概述

客户端使用 HTTP POST 请求来更新服务器中的数据。SimpleLink HTTP 网络服务器支持内容类型为 application/x-www-form-urlencoded 的 HTML 表单。浏览器发送的 POST 信息包括表单操作名称以及一对或多对变量名和变量值。

11.5.2 SimpleLink POST 令牌

在 SimpleLink 中，POST 中的变量名称应遵循与 GET 令牌相同的规则。

- 该令牌具有 10 个字符的固定长度。
- 该令牌的前缀与所有令牌相同，长度为七个字符。
- 该令牌前缀为：**__SL_P_**
- 一个典型的令牌应该是这样的：**__SL_P_XYZ**，其中 XYZ 是用户定义的，可包含任意字符或数字。

当 HTTP 网络服务器接收到 HTTP POST 请求时，服务器首先检查表单操作名称以了解是否应在内部处理此 POST。然后，服务器会检查参数列表，并检查每个变量名称以查看其是否与某个已知的预定义令牌相匹配。如果变量名称与预定义的令牌相匹配，则服务器会处理相应的值。

11.5.3 SimpleLink POST 操作

POST 操作包含两种类型：简单操作和复杂操作。

在简单的 POST 操作中，服务器处理一组 POST 参数并保存新信息（例如设置的域名）。在这种情况下，操作值并不重要，服务器不会识别操作值。

在复杂的 POST 操作中，服务器应收集所有需要的 POST 参数，然后触发特定操作（例如添加配置文件）。在这种情况下，通过 HTTP POST 命令中的操作值来识别相应操作。

11.5.4 用户定义的令牌

用户定义的令牌在 POST 请求中用于向主机发送信息。

如果 HTTP 网络服务器接收到的某个 HTTP POST 请求包含不在预定义列表中的令牌，则服务器会向主机生成一个 **post_token_value** 异步事件，其中包含以下信息：表单操作名称、令牌名称和令牌值。然后，主机可以处理所需的信息。

备注

为防止用户令牌与内部令牌相冲突，请使用具有以下结构的令牌：**__SL_P_UXX**，其中的 XX 可以是任意字符或数字。

11.5.5 发布后重定向

在 SimpleLink 中，用户提交 POST 后在浏览器中重定向到其他网页。

处理完 POST 后，HTTP 网络服务器会检查 POST 请求中收到的 action-URI。如果 action-URI 包含 SimpleLink（串行闪存或 ROM）中的有效网页，则服务器会发出带有 action-URI 值的 HTTP 302 Found 响应以进行重定向。

如果 action-URI 不包含有效网页，则 HTTP 网络服务器会发出 HTTP 204 No content 响应，并且浏览器仍停留在当前网页上。

11.5.6 带有 POST 和动态 HTML 内容的 HTML 示例代码

在以下 POST 示例中，一旦用户点击提交按钮，POST 请求将包含 profiles_add.html（作为操作资源）以及具有用户请求值的变量 __SL_P_P.A 和 __SL_P_P.B。

```
<form method="POST" name="SimpleLink Configuration" action="profiles_add.html">
<tr>
      <td dir=LTR> SSID: </td>
      <td dir=LTR><input type="text" maxlength="32" name="__SL_P_P.A" /> Enter any value of up
                                                                        to 32 characters</td>
</tr>
<tr>
      <td dir=LTR> Security Type: </td>
      <td dir=LTR> <input type="radio" name="__SL_P_P.B" value="0" checked />Open
                   <input type="radio" name="__SL_P_P.B" value="1" />WEP
                   <input type="radio" name="__SL_P_P.B" value="2" />WPA1
                   <input type="radio" name="__SL_P_P.B" value="3" />WPA2</td>
</tr>
<tr>
      <td colspan=2 align=center><input type="submit" value="Add"/></td>
</tr>
</form>
```

当该页面显示在以下示例 (HTTP GET) 中时，__SL_G_N.A 将替换为具有当前 IP 地址值（显示在输入框中）的 HTTP Web 服务器。当用户更改并提交 IP 地址时，新值会连同 __SL_P_N.A 变量一起发送。

```
<form method="POST" name="SimpleLink Configuration action" action="ip_config.html">
<tr>
      <td  dir=LTR> IP Address: </td>
      <td dir=LTR><input type="text"  maxlength="15" name="__SL_P_N.A" value="__SL_G_N.A"/></td>
</tr>
<tr>
      <td colspan=2 align=center><input type="submit" value="Apply"/></td>
</tr>
</form>
```

11.6 内部网页

SimpleLink 器件具有一个已嵌入 ROM 的默认网页。该网页可用于执行以下操作：

- 获取有关该器件的版本信息和一般信息

- IP 配置
- 添加或删除 Wi-Fi 配置文件
- 启用或禁用 ping 测试

通过 API 来配置对内部网页的访问。默认情况下，访问已启用。

该网页由以下文件组成：

- about.html
- image001.png
- ip_config.html
- Logo.gif
- main.html
- ping.html
- profiles_config.html
- simple_link.css
- status.html
- tools.html

11.7 “强制 AP”模式支持

“强制 AP”模式将 SimpleLink 恢复为默认配置。使用特殊的外部 GPIO 可进入该模式。

当 SimpleLink 进入“强制 AP”模式时，HTTP 服务器的行为如下：

- 服务器仅允许访问文件系统中的 *www/safe/* 文件夹。这允许用户在 *www/* 文件夹中放置一组在“强制 AP”模式下无法访问的网页。
- 在此模式下，内部配置网页中会出现一个“清除所有配置文件”按钮，使用户能够将器件中保存的所有配置文件清除。

11.8 访问网页

11.8.1 工作站模式下的 SimpleLink

当 SimpleLink 处于工作站模式时，用户可以使用 IP 地址从浏览器访问网页。HTTP 服务也由 mDNS 服务器发布，因此可以从 mDNS 文档中获取 IP 地址。

11.8.2 AP 模式下的 SimpleLink

在 AP 模式下，需要使用域名访问网页。应使用主机 API 配置域名。

在 AP 模式下，还需要使用 IP 地址访问 SimpleLink。

默认域名为 mysimplelink.net。

可以使用 mysimplelink.net 或 www.mysimplelink.net 访问该网页。

如果在域名中使用 www. 前缀，则当域名不匹配时，服务器会在内部删除此前缀并尝试不带此前缀进行搜索。

备注

请在域名搜索中使用 www. 前缀，因为从 DNS 的角度来看，商业浏览器的表现更好。

11.9 HTTP 身份验证检查

如果启用，SimpleLink 将在客户端首次连接到服务器时执行身份验证检查。可以使用主机 API 来启用或禁用身份验证检查。

身份验证用户名称、密码和领域也可以通过主机 API 进行配置。

默认情况下，禁用身份验证。

默认身份验证值为：

- 身份验证名称：**admin**
- 身份验证密码：**admin**
- 身份验证领域：**SimpleLink CC31xx**

可以在以下段落中找到关于主机接口的说明：章节 11。

11.10 使用 SimpleLink 驱动程序处理主机中的 HTTP 事件

当 HTTP 服务器在 HTML 文件中找到用户令牌时，服务器便会为主机生成 get_token_value（对于 GET 令牌）或 post_token_value（对于 post 令牌）事件，以便用户正确处理它们。

当主机获得具有特定令牌名称的 get_token_value 事件时，服务器会使用 send_token_value 命令返回该令牌名称的令牌值。

如果主机没有任何令牌值要返回，服务器会使用 0 作为令牌值的长度。

当用户获得具有该令牌名称和值的 post_token_value 事件时，必须将这个新的令牌值保存下来。

在 SimpleLink 驱动程序中，当生成上述事件之一时，驱动程序会调用一个名为 *SimpleLinkHttpServerCallback();* 的预定义回调函数

该回调函数定义如下：

```
void SimpleLinkHttpServerCallback(SlHttpServerEvent_t *pHttpServerEvent, SlHttpServerResponse_t
*pHttpServerResponse)
```

其中 serverEvent 和 serverResponse 定义如下：

```
typedef struct
{
   unsigned long                      Event;
   SlHttpServerEventData_u            EventData;
}SlHttpServerEvent_t;
typedef struct
{
   unsigned long                     Response;
   SlHttpServerResponsedata_u        ResponseData;
}SlHttpServerResponse_t;
typedef union
{
  slHttpServerString_t               httpTokenName; /* SL_NETAPP_HTTPGETTOKENVALUE */
  slHttpServerPostData_t             httpPostData;  /* SL_NETAPP_HTTPPOSTTOKENVALUE */
} SlHttpServerEventData_u;
typedef union
{
  slHttpServerString_t               token_value;  /* < 64 bytes*/
} SlHttpServerResponsedata_u;
typedef struct _slHttpServerString_t
{
      UINT8                          len;
      UINT8                          *data;
} slHttpServerString_t;
typedef struct _slHttpServerPostData_t
{
    slHttpServerString_t             action;
    slHttpServerString_t             token_name;
    slHttpServerString_t             token_value;
}slHttpServerPostData_t;
```

以下是用户回调函数示例代码：

```
/*HTTP 服务器回调函数示例 */
void SimpleLinkHttpServerCallback(SlHttpServerEvent_t *pHttpServerEvent,
                                  SlHttpServerResponse_t *pHttpServerResponse)
{
  switch (pHttpServerEvent->Event)
```

```
  {
    /* 处理 Get 令牌值 */
    case SL_NETAPP_HTTPGETTOKENVALUE:
    {
            char * tokenValue;
            tokenValue = GetTokenValue (pHttpServerEvent >EventData.httpTokenName);
            /* 使用驱动程序存储器进行响应 - 将令牌值复制到事件响应
            重要 - 令牌值长度应 < MAX_TOKEN_VALUE_LEN (64 bytes) */
            strcpy (pHttpServerResponse->ResponseData.token_value.data, tokenValue);
            pHttpServerResponse->ResponseData.token_value.len = strlen (tokenValue);
    }
            break;
            /* 处理 Post 令牌 */
            case SL_NETAPP_HTTPPOSTTOKENVALUE:
            {
                 HandleTokenPost (pHttpServerEvent->EventData.httpPostData.action,
                                  pHttpServerEvent->EventData.httpPostData.token_name,
                                  pHttpServerEvent->EventData.httpPostData.token_value);
    }
        break;
    default:
        break;
    }
}
```

备注

若要使 HTTP 回调函数正常工作，应在 *user.h* 中取消注释以下行：

```
#define sl_HttpServerCallback SimpleLinkHttpServerCallback
```

11.11 SimpleLink 驱动程序连接 HTTP 网络服务器

SimpleLink 驱动程序提供了一个 API 来访问和配置 HTTP 服务器。

在 *netapp.h* 中可以找到 API 定义。

11.11.1 启用或禁用 HTTP 服务器

用于启用或禁用 HTTP 服务器的函数。默认情况下，HTTP 服务器已启用。

表 11-1. 启用或禁用 HTTP 服务器

函数名称	说明	参数
void sl_NetAppStart(UINT32 appId)	启动 HTTP 服务器	appId = SL_NET_APP_HTTP_SERVER_ID
void sl_NetAppStop(UINT32 appId)	停止 HTTP 服务器	appId = SL_NET_APP_HTTP_SERVER_ID

11.11.2 配置 HTTP 端口号

用于配置端口号的函数。

表 11-2. 配置 HTTP 端口号

函数名称	说明	参数
long sl_NetAppSet (unsigned char appId , unsigned char Option, unsigned char OptionLen, unsigned char *pOptionValue)	设置 HTTP 服务器将侦听的端口号。端口号为 UINT16。	appId = SL_NET_APP_HTTP_SERVER_ID Option = NETAPP_SET_GET_HTTP_OPT_PORT_NUMBER pOptionLen = 2 (UINT16 的大小) pOptionValue = 指向端口号 (UINT16) 的指针

表 11-2. 配置 HTTP 端口号 (continued)

函数名称	说明	参数
long sl_NetAppGet (unsigned char appId, unsigned char Option, unsigned char *pOptionLen, unsigned char *pOptionValue)	获取当前 HTTP 服务器端口号。 端口号为 UINT16。	appId = SL_NET_APP_HTTP_SERVER_ID Option = NETAPP_SET_GET_HTTP_OPT_PORT_NUMBER pOptionLen = 用户提供的指向返回值长度的指针 pOptionValue = 用户提供的指向返回值的指针

备注

设置新端口号之后，重新启动服务器以进行配置。

11.11.3 启用或禁用身份验证检查

用于启用或禁用身份验证检查的函数。默认情况下，禁用身份验证检查。

表 11-3. 启用或禁用身份验证检查

函数名称	说明	参数
long sl_NetAppSet (unsigned char appId , unsigned char Option, unsigned char OptionLen, unsigned char *pOptionValue)	启用或禁用 HTTP 服务器身份验证检查 Auth_enable 值为 true/false。	appId = SL_NET_APP_HTTP_SERVER_ID Option = NETAPP_SET_GET_HTTP_OPT_AUTH_CHECK pOptionLen = 1（UINT8 的大小） pOptionValue = 指向 auth_enable (true/false) 的指针
long sl_NetAppGet (unsigned char appId, unsigned char Option, unsigned char *pOptionLen, unsigned char *pOptionValue)	获取当前身份验证状态。 返回 auth_enable 值为 true/false。	appId = SL_NET_APP_HTTP_SERVER_ID Option = NETAPP_SET_GET_HTTP_OPT_AUTH_CHECK pOptionLen = 用户提供的指向返回值长度的指针 pOptionValue = 用户提供的指向返回值的指针

11.11.4 设置或获取身份验证名称、密码和领域

用于设置或获取身份验证名称、密码和领域的函数。

表 11-4. 设置或获取身份验证名称

函数名称	说明	参数
long sl_NetAppSet (unsigned char appId , unsigned char Option, unsigned char OptionLen, unsigned char *pOptionValue)	设置身份验证名称。 名称格式为字符串。	appId = SL_NET_APP_HTTP_SERVER_ID Option = NETAPP_SET_GET_HTTP_OPT_AUTH_NAME OptionLen = 身份验证名称长度 pOptionValue = 指向身份验证名称的指针
long sl_NetAppGet (unsigned char appId, unsigned char Option, unsigned char *pOptionLen, unsigned char *pOptionValue)	获取当前身份验证名称。 名称格式为字符串（非 null 结尾）。	appId = SL_NET_APP_HTTP_SERVER_ID Option = NETAPP_SET_GET_HTTP_OPT_AUTH_NAME pOptionLen = 用户提供的指向返回名称长度的指针 pOptionValue = 用户提供的指向返回名称的指针

表 11-5. 设置或获取身份验证密码

函数名称	说明	参数
long sl_NetAppSet (unsigned char appId , unsigned char Option, unsigned char OptionLen, unsigned char *pOptionValue)	设置身份验证密码。 密码格式为字符串。	appId = SL_NET_APP_HTTP_SERVER_ID Option = NETAPP_SET_GET_HTTP_OPT_AUTH_PASSWORD OptionLen = 身份验证密码长度 pOptionValue = 指向身份验证密码的指针
long sl_NetAppGet (unsigned char appId, unsigned char Option, unsigned char *pOptionLen, unsigned char *pOptionValue)	获取当前身份验证密码。 密码格式为字符串（非 null 结尾）。	appId = SL_NET_APP_HTTP_SERVER_ID Option = NETAPP_SET_GET_HTTP_OPT_AUTH_PASSWORD pOptionLen = 用户提供的指向返回密码长度的指针 pOptionValue = 用户提供的指向返回密码的指针

表 11-6. 设置或获取身份验证领域

函数名称	说明	参数
long sl_NetAppSet (unsigned char appId , unsigned char Option, unsigned char OptionLen, unsigned char *pOptionValue)	设置身份验证领域。 领域格式为字符串。	appId = SL_NET_APP_HTTP_SERVER_ID Option = NETAPP_SET_GET_HTTP_OPT_AUTH_REALM OptionLen = 身份验证领域长度 pOptionValue = 指向身份验证领域的指针
long sl_NetAppGet (unsigned char appId, unsigned char Option, unsigned char *pOptionLen, unsigned char *pOptionValue)	获取当前身份验证领域。 领域格式为字符串（非 null 结尾）。	appId = SL_NET_APP_HTTP_SERVER_ID Option = NETAPP_SET_GET_HTTP_OPT_AUTH_REALM pOptionLen = 用户提供的指向返回领域长度的指针 pOptionValue = 用户提供的指向返回领域的指针

11.11.5 设置或获取域名

用于设置或获取域名（用于以 AP 方式访问网络服务器）的函数。

表 11-7. 设置或获取域名

函数名称	说明	参数
long sl_NetAppSet (unsigned char appId , unsigned char Option, unsigned char OptionLen, unsigned char *pOptionValue)	设置域名。 域名格式为字符串。	appId = SL_NET_APP_DEVICE_CONFIG_ID Option = NETAPP_SET_GET_DEV_CONF_OPT_DOMAIN_NAME OptionLen = 域名长度 pOptionValue = 指向域名的指针
long sl_NetAppGet (unsigned char appId, unsigned char Option, unsigned char *pOptionLen, unsigned char *pOptionValue)	获取当前域名。 域名格式为字符串（非 null 结尾）。	appId = SL_NET_APP_DEVICE_CONFIG_ID Option = NETAPP_SET_GET_DEV_CONF_OPT_DOMAIN_NAME pOptionLen = 用户提供的指向返回域名长度的指针 pOptionValue = 用户提供的指向返回域名的指针

11.11.6 设置或获取 URN 名称

用于设置或获取器件唯一 URN 名称的函数。

表 11-8. 设置或获取 URN 名称

函数名称	说明	参数
`long sl_NetAppSet (unsigned char appId , unsigned char Option, unsigned char OptionLen, unsigned char *pOptionValue)`	设置 URN 名称。 URN 名称格式为字符串。	appId = SL_NET_APP_DEVICE_CONFIG_ID Option = NETAPP_SET_GET_DEV_CONF_OPT_DEVICE_URN OptionLen = URN 名称长度 pOptionValue = 指向 URN 名称的指针
`long sl_NetAppGet (unsigned char appId, unsigned char Option, unsigned char *pOptionLen, unsigned char *pOptionValue)`	获取当前 URN 名称。 URN 名称格式为字符串（非 null 结尾）。	appId = SL_NET_APP_DEVICE_CONFIG_ID Option = NETAPP_SET_GET_DEV_CONF_OPT_DEVICE_URN pOptionLen = 用户提供的指向返回 URN 名称长度的指针 pOptionValue = 用户提供的指向返回 URN 名称的指针

11.11.7 启用或禁用 ROM 网页访问

用于启用或禁用 ROM 内部网页访问的函数。默认情况下，Web 访问已启用。

表 11-9. 启用或禁用 ROM 网页访问

函数名称	说明	参数
`long sl_NetAppSet (unsigned char appId , unsigned char Option, unsigned char OptionLen, unsigned char *pOptionValue)`	启用或禁用 HTTP 服务器 ROM 网页访问。 值为 true 或 false。	appId = SL_NET_APP_HTTP_SERVER_ID Option = NETAPP_SET_GET_HTTP_OPT_ROM_PAGES_ACCESS OptionLen = 1（UINT8 的大小） pOptionValue = 指向布尔值 (true/false) 的指针
`long sl_NetAppGet (unsigned char appId, unsigned char Option, unsigned char *pOptionLen, unsigned char *pOptionValue)`	获取当前 ROM 网页访问状态。 返回值为 true 或 false。	appId = SL_NET_APP_HTTP_SERVER_ID Option = NETAPP_SET_GET_HTTP_OPT_ROM_PAGES_ACCESS pOptionLen = 用户提供的指向返回值长度的指针 pOptionValue = 用户提供的指向返回值的指针

11.12 SimpleLink 预定义令牌

本节包含了有关预定义内部令牌的信息，并描述了用于 GET 操作、POST 操作和内部处理的 POST 操作的令牌。

备注

为防止用户令牌与内部令牌相冲突，请使用具有以下结构的令牌：

- 对于 GET 操作：**__SL_G_UXX**，其中 XX 可以是任意字符或数字。
- 对于 POST 操作：**__SL_P_UXX**，其中 XX 可以是任意字符或数字。

11.12.1 GET 值

表 11-10 中显示了 GET 系统信息值。

表 11-10. 系统信息

令牌	名称	值/用法
__SL_G_S.A	系统运行时间	返回自上次复位以来的系统运行时间，格式如下： 000 天 00:00:00
__SL_G_S.B	器件名称 (URN)	返回器件名称
__SL_G_S.C	域名	返回域名
__SL_G_S.D	器件模式（角色）	返回器件角色。 值：工作站、接入点、P2P
__SL_G_S.E	器件角色工作站	下拉菜单 select/not select 如果器件是工作站，则返回“selected”，否则返回“not_selected”。
__SL_G_S.F	器件角色 AP	下拉菜单 select/not select 如果器件是 AP，则返回“selected”，否则返回“not_selected”。
__SL_G_S.G	器件角色 P2P	下拉菜单 select/not select 如果器件处于 P2P 模式，则返回“selected”，否则返回“not_selected”。
__SL_G_S.H	器件名称 URN（截断为 16 个字节）	返回长度不超过 16 字节的 URN 字符串名称。更长的名称会被截断。
__SL_G_S.I	系统需要复位（参数更改后）	如果系统需要复位，则返回值为如下字符串：-- Some parameters were changed, System may require reset --，否则返回空字符串。 处理的每个内部 POST 都会导致此令牌返回 TRUE。
__SL_G_S.J	获取系统时间和日期	返回值为一个字符串，格式如下： 年, 月, 日, 小时, 分钟, 秒
__SL_G_S.K	安全模式状态	如果器件处于安全模式，则返回“Safe Mode”，否则返回空字符串。

表 11-11 中显示了 GET 版本信息。

表 11-11. 版本信息

令牌	名称	值/用法
__SL_G_V.A	NWP 版本	返回带有版本信息的字符串
__SL_G_V.B	MAC 版本	—
__SL_G_V.C	PHY 版本	—
__SL_G_V.D	硬件版本	—

表 11-12 中显示了 GET 网络信息。

表 11-12. 网络信息

令牌	名称	值/用法
工作站（和 P2P 客户端）		
__SL_G_N.A	STA IP 地址	字符串格式：xxx.yyy.zzz.ttt
__SL_G_N.B	STA 子网掩码	—
__SL_G_N.C	STA 默认网关	—
__SL_G_N.D	MAC 地址	字符串格式：00:11:22:33:44:55
__SL_G_N.E	STA DHCP 状态	返回值：Enabled 或 Disabled

表 11-12. 网络信息 (continued)

令牌	名称	值/用法
__SL_G_N.F	STA DHCP 禁用状态	如果 DHCP 禁用，则返回 Checked，否则返回 Not_Checked。 用于 DHCP 单选按钮。
__SL_G_N.G	STA DHCP 启用状态	如果 DHCP 启用，则返回 Checked，否则返回 Not_Checked。 用于 DHCP 单选按钮。
__SL_G_N.H	STA DNS 服务器	字符串格式：xxx.yyy.zzz.ttt
DHCP 服务器		
__SL_G_N.I	DHCP 起始地址	—
__SL_G_N.J	DHCP 最后地址	—
__SL_G_N.K	DHCP 租用时间	租用时间字符串（以秒为单位）
AP（和 P2P Go）		
__SL_G_N.P	AP IP 地址	字符串格式：xxx.yyy.zzz.ttt
__SL_G_W.A	AP 模式下的通道编号	
__SL_G_W.B	SSID	
__SL_G_W.C	安全类型	返回值：Open、WEP、WPA
__SL_G_W.D	安全类型 Open	如果安全类型为 Open，则返回 Checked，否则返回 Not_Checked。 用于安全类型单选按钮 check/not checked。
__SL_G_W.E	安全类型 WEP	如果安全类型为 WEP，则返回 Checked，否则返回 Not_Checked。 用于安全类型单选按钮 check/not checked。
__SL_G_W.F	安全类型 WPA	如果安全类型为 WPA，则返回 Checked，否则返回 Not_Checked。 用于安全类型单选按钮 check/not checked。

表 11-13 中显示了 GET 工具。

表 11-13. 工具

令牌	名称	值/用法
Ping 测试结果		
__SL_G_T.A	IP 地址	字符串格式：xxx.yyy.zzz.ttt
__SL_G_T.B	数据包大小	
__SL_G_T.C	Ping 数	
__SL_G_T.D	Ping 结果 – 总发送数	发送的 Ping 总数
__SL_G_T.E	Ping 结果 – 成功发送数	成功发送的 Ping 数
__SL_G_T.E	Ping 测试持续时间	以秒为单位

表 11-14 中显示了 GET 连接策略状态

表 11-14. 连接策略状态

令牌	名称	值/用法
__SL_G_P.E	自动连接	如果自动连接启用，则返回 Checked，否则返回 Not_Checked。 用于自动连接复选框。
__SL_G_P.F	快速连接	如果快速连接启用，则返回 Checked，否则返回 Not_Checked。 用于快速连接复选框。

表 11-14. 连接策略状态 (continued)

令牌	名称	值/用法
__SL_G_P.G	任何 P2P	如果任何 P2P 启用，则返回 Checked，否则返回 Not_Checked。 用于任何 P2P 复选框。
__SL_G_P.P	自动 SmartConfig	如果自动 SmartConfig 启用，则返回 Checked，否则返回 Not_Checked。 用于自动 SmartConfig 复选框。

表 11-15 中显示了 GET 显示配置文件信息。

表 11-15. 显示配置文件信息

令牌	名称	值/用法
__SL_G_PN1	返回配置文件 1 SSID	SSID 字符串
__SL_G_PN2	返回配置文件 2 SSID	—
__SL_G_PN3	返回配置文件 3 SSID	—
__SL_G_PN4	返回配置文件 4 SSID	—
__SL_G_PN5	返回配置文件 5 SSID	—
__SL_G_PN6	返回配置文件 6 SSID	—
__SL_G_PN7	返回配置文件 7 SSID	—
__SL_G_PS1	返回配置文件 1 安全状态	返回值：Open、WEP、WPA、WPS、ENT、P2P_PBC、P2P_PIN 或 - （用于空配置文件）。
__SL_G_PS2	返回配置文件 2 安全状态	—
__SL_G_PS3	返回配置文件 3 安全状态	—
__SL_G_PS4	返回配置文件 4 安全状态	—
__SL_G_PS5	返回配置文件 5 安全状态	—
__SL_G_PS6	返回配置文件 6 安全状态	—
__SL_G_PS7	返回配置文件 7 安全状态	—
__SL_G_PP1	返回配置文件 1 优先级	配置文件优先级：0-7
__SL_G_PP2	返回配置文件 2 优先级	—
__SL_G_PP3	返回配置文件 3 优先级	—
__SL_G_PP4	返回配置文件 4 优先级	—
__SL_G_PP5	返回配置文件 5 优先级	—
__SL_G_PP6	返回配置文件 6 优先级	—
__SL_G_PP7	返回配置文件 7 优先级	—

表 11-16 中显示了 GET P2P 信息。

表 11-16. P2P 信息

令牌	名称	值/用法
__SL_G_R.A	P2P 器件名称	字符串
__SL_G_R.B	P2P 器件类型	字符串
__SL_G_R.C	P2P 侦听信道	返回侦听信道编号的字符串
__SL_G_R.T	侦听信道 1	如果当前侦听信道为 1，则返回 Selected，否则返回 Not_selected。 用于侦听信道的下拉菜单。

表 11-16. P2P 信息 (continued)

令牌	名称	值/用法
__SL_G_R.U	侦听信道 6	如果当前侦听信道为 6，则返回 Selected，否则返回 Not_selected。 用于侦听信道的下拉菜单。
__SL_G_R.V	侦听信道 11	如果当前侦听信道为 11，则返回 Selected，否则返回 Not_selected。 用于侦听信道的下拉菜单。
__SL_G_R.E	P2P 工作信道	返回工作信道编号的字符串
__SL_G_R.W	工作信道 1	如果当前工作信道为 1，则返回 Selected，否则返回 Not_selected。 用于工作信道的下拉菜单。
__SL_G_R.X	工作信道 6	如果当前工作信道为 6，则返回 Selected，否则返回 Not_selected。 用于工作信道的下拉菜单。
__SL_G_R.Y	工作信道 11	如果当前工作信道为 11，则返回 Selected，否则返回 Not_selected。 用于工作信道的下拉菜单。
__SL_G_R.L	协商意图值	返回值：Group Owner、Negotiate、Client
__SL_G_R.M	组所有者角色	如果意图为组所有者，则返回 Checked，否则返回 Not_Checked。 用于协商意图单选按钮。
__SL_G_R.N	协商角色	如果意图为协商，则返回 Checked，否则返回 Not_Checked。 用于协商意图单选按钮。
__SL_G_R.O	客户端角色	如果意图为客户端，则返回 Checked，否则返回 Not_Checked。 用于协商意图单选按钮。
__SL_G_R.P	协商发起方策略	返回值：Active、Passive、Random Backoff
__SL_G_R.Q	协商发起方主动	如果协商发起方采用主动策略，则返回 Checked，否则返回 Not_Checked。 用于协商发起方策略单选按钮。
__SL_G_R.R	协商发起方被动	如果协商发起方采用被动策略，则返回 Checked，否则返回 Not_Checked。 用于协商发起方策略单选按钮。
__SL_G_R.S	协商发起方随机退避	如果协商发起方采用随机退避策略，则返回 Checked，否则返回 Not_Checked。 用于协商发起方策略单选按钮。

11.12.2 POST 值

表 11-17 中显示了 POST 系统配置。

表 11-17. 系统配置

令牌	名称	值/用法
__SL_P_S.B	器件名称 (URN)	设置器件名称
__SL_P_S.C	域名	设置域名
__SL_P_S.D	器件模式 (角色)	设置器件模式 值：Station、AP、P2P
__SL_P_S.J	发布系统时间和日期	设置系统时间和日期。值是一个字符串，格式如下： 年, 月, 日, 小时, 分钟, 秒

表 11-17. 系统配置 (continued)

令牌	名称	值/用法
__SL_P_S.R	发布后重定向	值应包含有效的网页。如果该页面存在，则网络服务器会发出 HTTP 302 响应以重定向到该网页。 可用于提交表单后的重定向（使用 HTTP POST）。

表 11-18 中显示了 POST 网络配置。

表 11-18. 网络配置

令牌	名称	值/用法
工作站（和 P2P 客户端）		
__SL_P_N.A	STA IP 地址	设置 STA IP 地址。 值格式：xxx.yyy.zzz.ttt
__SL_P_N.B	STA 子网掩码	设置 STA 子网掩码。 值格式：xxx.yyy.zzz.ttt
__SL_P_N.C	STA 默认网关	设置 STA 默认网关。 值格式：xxx.yyy.zzz.ttt
__SL_P_N.D	STA DHCP 状态（必须禁用才能使 IP 设置生效）	启用或禁用 DHCP 状态。 如果值为 Enable，则启用 DHCP，任何其他值都将禁用 DHCP。
__SL_P_N.H	STA DNS 服务器	设置 STA DNS 服务器地址。 值格式：xxx.yyy.zzz.ttt
DHCP 服务器		
__SL_P_N.I	DHCP 起始地址	设置起始地址。 值格式：xxx.yyy.zzz.ttt
__SL_P_N.J	DHCP 最后地址	设置最后地址。 值格式：xxx.yyy.zzz.ttt
__SL_P_N.K	DHCP 租用时间	设置租用时间，以秒为单位
AP（和 P2P Go）		
__SL_P_N.P	AP IP 地址	设置 AP IP 地址。 值格式：xxx.yyy.zzz.ttt
__SL_P_W.A	AP 模式下的通道编号	设置通道编号。 值：1 至 13
__SL_P_W.B	SSID	设置 SSID
__SL_P_W.C	安全类型	设置安全类型： 0（开放）、1 (WEP)、2 (WPA)。
__SL_P_W.G	密码	设置密码

表 11-19 中显示了 POST 连接策略配置。

表 11-19. 连接策略配置

令牌	名称	值/用法
连接策略配置		与连接策略表单（policy_config.html 操作）一同使用
__SL_P_P.E	自动连接	Enable 或 Disable 如果 POST 中存在此参数（具有任何值），则设置此策略。如果 POST 中不存在此参数，则清除此策略标志。

表 11-19. 连接策略配置 (continued)

令牌	名称	值/用法
__SL_P_P.F	快速连接	Enable 或 Disable 如果 POST 中存在此参数（具有任何值），则设置此策略。如果 POST 中不存在此参数，则清除此策略标志。
__SL_P_P.G	任何 P2P	Enable 或 Disable 如果 POST 中存在此参数（具有任何值），则设置此策略。如果 POST 中不存在此参数，则清除此策略标志。
__SL_P_P.P	自动 SmartConfig	Enable 或 Disable 如果 POST 中存在此参数（具有任何值），则设置此策略。如果 POST 中不存在此参数，则清除此策略标志。

表 11-20 中显示了 POST 配置文件配置。

表 11-20. 配置文件配置

令牌	名称	值/用法
添加新配置文件		与添加新配置文件表单（profiles_add.html 操作）一同使用
__SL_P_P.A	SSID	SSID 字符串
__SL_P_P.B	安全类型	安全类型：0（开放）、1 (WEP)、2 (WPA1)、3 (WPA2)
__SL_P_P.C	安全密钥	小于 32 个字符
__SL_P_P.D	配置文件优先级	0 至 7
添加 P2P 配置文件		与添加 P2P 配置文件表单（p2p_profiles_add 操作）一同使用
__SL_P_P.A	P2P 远程器件名称	字符串
__SL_P_P.B	P2P 安全类型	安全类型：6（按钮）、7（PIN 键盘）、8（PIN 显示屏）
__SL_P_P.C	P2P PIN 码	仅限数字
__SL_P_P.D	P2P 配置文件优先级	0 至 7
添加企业配置文件		与添加企业配置文件表单（eap_profiles_add 操作）一同使用
__SL_P_P.H	SSID	字符串
__SL_P_P.I	身份	字符串
__SL_P_P.J	匿名身份	字符串
__SL_P_P.K	密码	字符串
__SL_P_P.L	配置文件优先级	0 至 7
__SL_P_P.M	EAP 方法	值：TLS、TTLS、PEAP0、PEAP1、FAST
__SL_P_P.N	阶段 2 身份验证	值：TLS、MSCHAPV2、PSK
__SL_P_P.O	配置（0、1、2）（仅适用于快速方法）	值：None、0、1、2、3 仅与快速方法相关（值为 0 至 3） 对于其他方法，请使用 None。
配置文件删除		
__SL_P_PRR	删除配置文件	删除所选的配置文件 - 值：1 至 7

表 11-21 中显示了 POST 工具。

表 11-21. 工具

令牌	名称	值/用法
开始 Ping 测试		
__SL_P_T.A	IP 地址	远程器件的 IP 地址
__SL_P_T.B	数据包大小	以字节为单位（32 至 1472）
__SL_P_T.C	Ping 数	0 至无限，1 至 255

表 11-22 中显示了 POST P2P 配置。

表 11-22. P2P 配置

令牌	名称	值/用法
__SL_P_R.E	P2P 通道（运行）	设置 P2P 运行通道。 值：1、6、11
__SL_P_R.L	协商意图值	设置协商意图值。 值：CL（客户端）、NEG（协商）、GO（组所有者）

11.12.3 POST 操作

表 11-23 将内部处理的 POST 操作描述为复杂操作以及每个 POST 操作中使用的相应令牌参数。所有其余的 POST 参数均由服务器进行处理，即更新这些参数各自的值。如需更多信息，请参阅节 11.5.3。

表 11-23. POST 操作

操作名称	说明	POST 令牌参数
sta_ip_config	工作站网络配置	• STA IP 地址 • STA 子网掩码 • STA 默认网关 • STA DHCP 状态 • STA DNS 服务器
ap_ip_config	接入点网络配置	• AP IP 地址 • DHCP 起始地址 • DHCP 最后地址 • DHCP 租用时间
profiles_add.html	添加新配置文件	• SSID • 安全类型 • 安全密钥 • 配置文件优先级
p2p_profiles_add	添加对等配置文件	• P2P 远程器件名称 • P2P 安全类型 • P2P PIN 码 • P2P 配置文件优先级
eap_profiles_add	添加企业配置文件	• SSID • 身份 • 匿名身份 • 密码 • 配置文件优先级 • EAP 方法 • 阶段 2 身份验证 • 配置
remove_all_profiles	删除所有配置文件	不相关 注意：在 HTML 中至少保留一个参数，使 HTTP POST 不会为空。 服务器不会检查参数值。
ping.html	开始 Ping 测试	• IP 地址 • 数据包大小 • Ping 数

表 11-23. POST 操作 (continued)

操作名称	说明	POST 令牌参数
ping_stop	停止 Ping 测试	不相关 注意：在 HTML 中至少有一个参数，使 HTTP POST 不会为空。 服务器不会检查参数值。
policy_config.html	连接策略配置	• 自动连接 • 快速连接 • 任何 P2P • 自动 SmartConfig

11.12.4 HTTP 服务器限制

- HTTPS 当前不受支持
- HTTP Web 服务器只能托管一个域
- 不支持使用 GET 方法提交 HTML 表单
- 配置接入点 IP 地址时，该地址应位于以下子网 – 192.168.1.xxx。

mDNS

12.1 概述

符合 RFC1034 的域名系统 (DNS) 是用于计算机、服务或任何联网资源的分层分布式命名系统。在单播 DNS 系统中，所有域名信息都存储在 DNS 服务器中。客户端只需从服务器获取这些信息即可获知其他的域名。

在不具备本地名称服务器的小型网络中，符合 RFC6762 的多播 DNS (mDNS) 可将主机名解析为互联网协议 (IP) 地址。这是一种零配置服务，使用与单播 DNS 基本相同的编程接口、数据包格式和操作语义。虽然根据设计应独立使用，但也可与单播 DNS 服务器协同工作。此协议使用 IP 多播用户数据报协议 (UDP) 数据包，而不是单播 DNS 使用的传输控制协议 (TCP)。

mDNS 与符合 RFC6763 的 DNS 服务发现 (DNS-SD) 一同构成了 Apple 的零配置网络实现方案，即 Bonjour。TI 的 Simplelink WiFi 目前在广播模式下支持此类实现。查询模式仅支持 PTR，未来将实现对 Bonjour 的完全支持。

12.2 协议详细信息

默认情况下，mDNS 专门解析以 .local 顶级域 (TLD) 结尾的主机名。mDNS 以太网帧是发送到以下位置的多播 UDP 数据包：

- MAC 地址 01:00:5E:00:00:FB（对于 IPv4）或 33:33:00:00:00:FB（对于 IPv6）
- IPv4 地址 224.0.0.251 或 IPv6 地址 FF02::FB
- UDP 端口 5353

以太网帧内容（如 Questions、Answers、Authority、Additional 和 Data 字段，包括它们的格式）与单播 DNS 数据包中的内容相同。有关详细规范，请参阅 RFC1034。

可使用与单播 DNS 相同的资源记录 (RR) 查询来查找服务。RR 存储关于域的大量记录，例如 IP 地址（A 和 AAAA 类型）、指针记录（PTR 类型）、服务定位器（SRV 类型）、文本（TXT 类型）等。完整列表可在 Wikipedia 链接中找到：DNS 记录类型列表。

DNS 区域数据库是资源记录的集合。每条资源记录指定有关特定对象的信息。例如，地址映射用于记录主机名到 IP 地址的对应关系。

需要使用 PTR、SRV、TXT 和 A（如果是 IPv6，则为 AAAA）类型的 RR 查询来发现完整服务的详细信息。

- PTR RR 返回 URN/完整服务名称
- SRV RR 用于表示主机提供的发现服务，并返回服务类型，包括域名。
- TXT RR 获取与域关联的任意文本，或描述相应服务。相关记录将主机名映射到 IPV4 地址。

必须在对查询的单个响应中发送所有答案。例如：

mDNS 客户端在接收到 PTR 记录后必须解析主机名，其中包括应用程序所需的服务：

1. 客户端发送一条 IP 多播查询消息，请求具有该名称的主机来识别它。
2. 然后，该目标机器会多播一条包含其 IP 地址的消息。
3. 然后，该子网中的所有机器都可以使用此信息来更新其 mDNS 缓存。

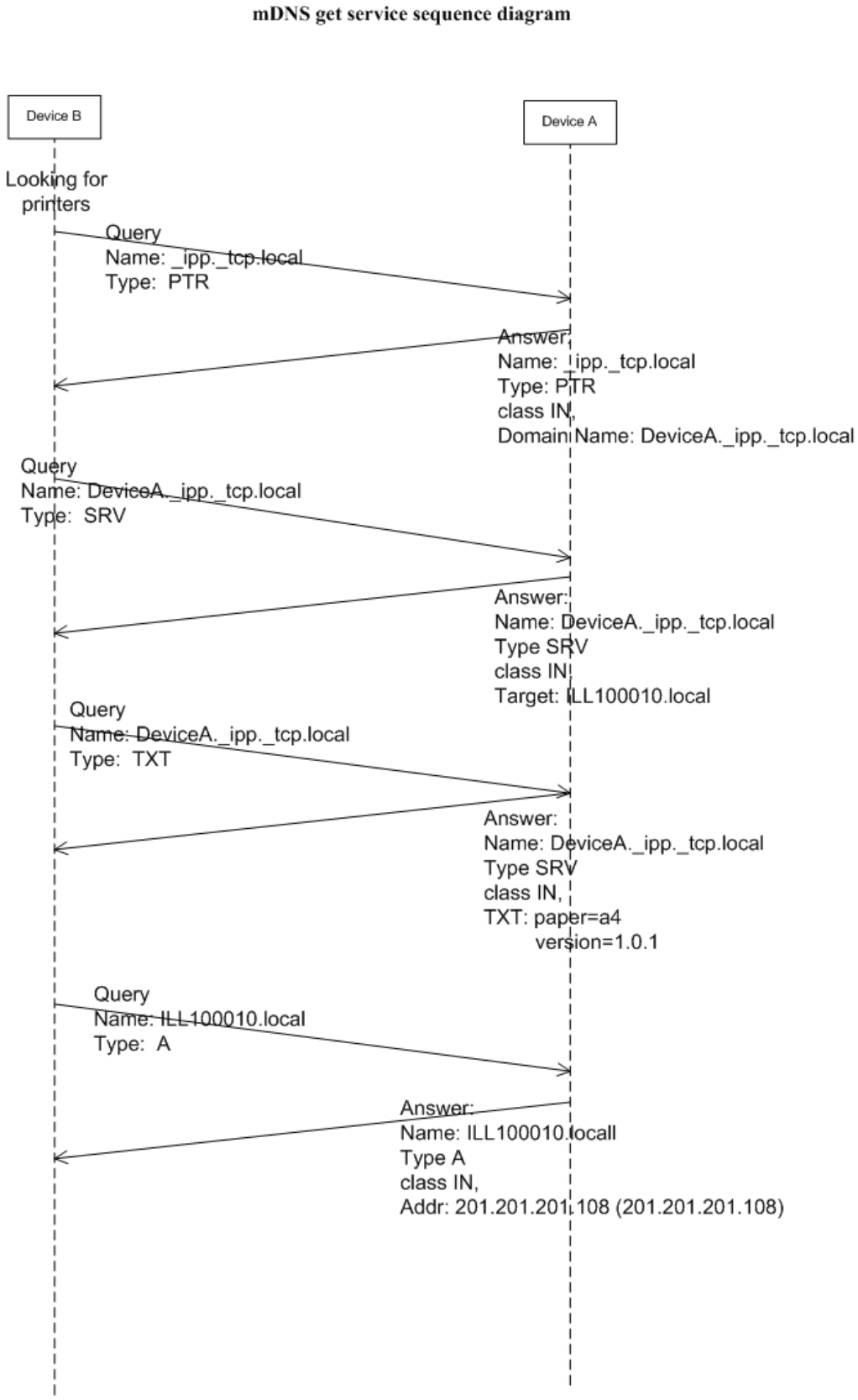


图 12-1. mDNS 获取服务序列

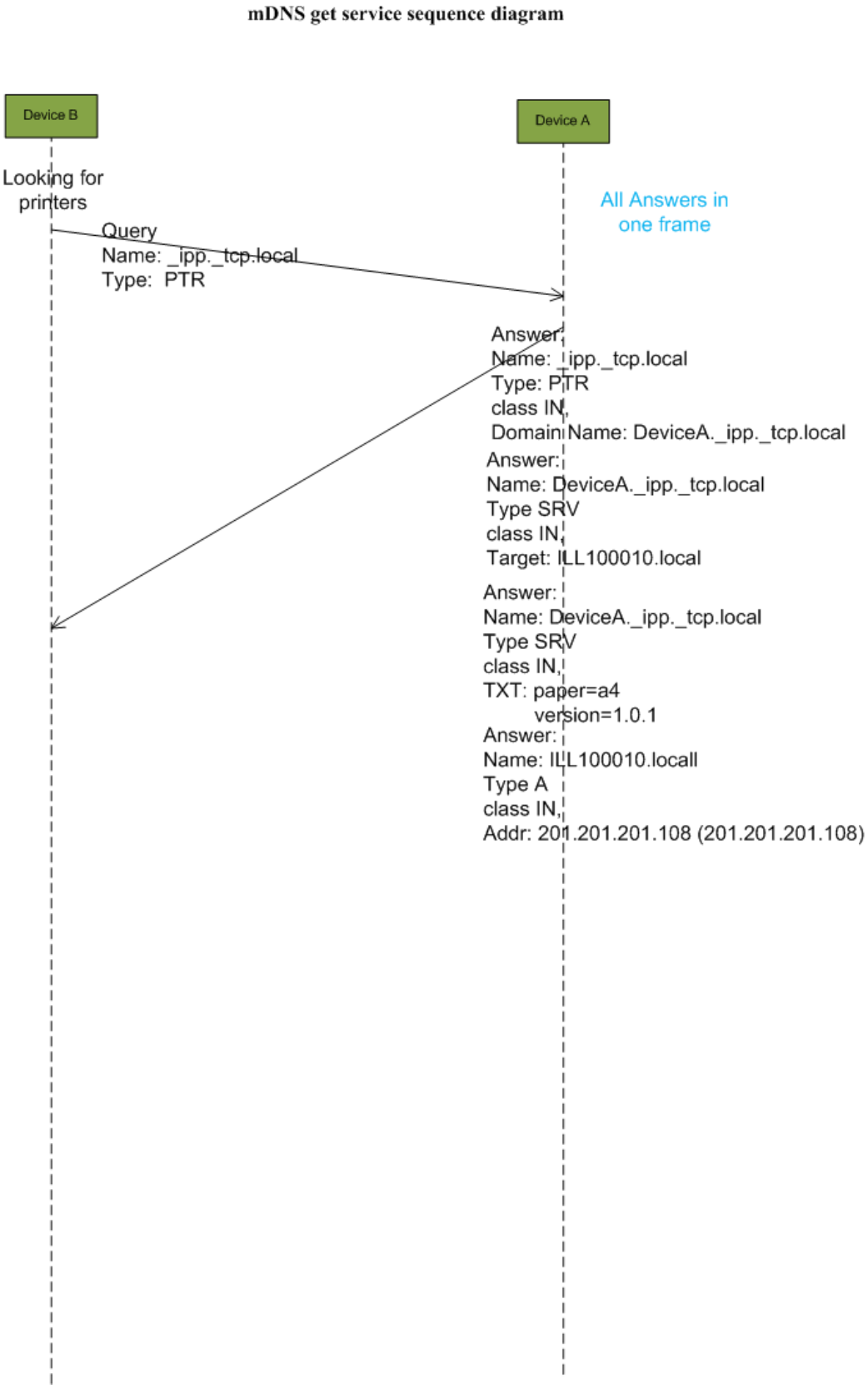


图 12-2. 查询后查找完整服务

12.3 实现

12.3.1 默认实现

默认情况下，使用以下名称启动 Simplelink 实现中的 mDNS 服务：

- 目标名称：<MAC address>-mysimplelink.local

  目标名称是应该位于“A”查询（用于获取 IPv4 的查询）中的名称。该名称由目标的 MAC 地址、“mysimplelink”一词以及本地网络的域组合而成。用户可以通过设置 URN 名称或智能配置操作来更改此名称。
- 内部服务：<MAC address>@mysimplelink._http._local

12.3.2

函数、结构和常量的完整列表可在 Netapp 库中找到。以下各节介绍了顶级特定实现的示例。有关详细信息，请参阅 API。

12.3.3 启动和停止 mDNS

mDNS 是一种可通过以下函数启动或停止的 NWP 服务：

- *sl_NetAppStart(SL_NET_APP_MDNS_ID)*;
- *sl_NetAppStop(SL_NET_APP_MDNS_ID)*;

宏 SL_NET_APP_MDNS_ID 是 mDNS 服务的定义。

仅当存在有效 IP 地址时才会发送 mDNS 帧（广播、对查询的响应），这意味着如果满足以下 WLAN 状态之一，mDNS 将起作用：

- SimpleLink 作为一个工作站连接到 AP
- 正在运行 SimpleLink P2P 模式，无论是作为组所有者 (GO) 还是作为客户端
- 正在运行 SimpleLink AP 模式

请注意 sl_NetAppStart() 和 sl_NetAppStop() 函数：

- 具有持久性 - 状态存储在串行闪存中，即使在经过下电上电后仍将保持先前状态。
- 依赖于角色 - 状态不能在不同角色之间共享。例如，如果 Simplelink 当前处于 AP 模式且 NetApp 已启动，则切换到 STA 模式时，如果先前在 STA 模式期间已禁用 NetApp，现在需要重新启动 NetApp。其他 NetApp 配置，如注册的服务（例如，*sl_NetAppMDNSRegisterService()*），不受角色更改的影响。

12.3.4 mDNS 查询 – 一次性

在一次性查询中，发现结果取决于发现器接收的第一个响应。函数 *sl_NetAppDnsGetHostByService()* 用于执行此类任务。成功发现后，该函数将返回：

- 服务的 IP 地址
- 服务的端口
- 服务的文本

以下代码片段显示了发现“_workstation._tcp.local”服务的一次性查询：

```
#define LISTEN_SERVICE_NAME     "_workstation._tcp.local"
void mDNS_Query_OneShot()
{
    _u32 pAddr;
    _u32 usPort;
    _u16 ulTextLen;
    _i8  cText[200];

    // Set an infinite loop until getting a response
    _i32 iretvalmDNS = 1;
    while(iretvalmDNS)
    {
```

```
        // Search services on the network by specifying the name
        iretvalmDNS = sl_NetAppDnsGetHostByService(
            LISTEN_SERVICE_NAME,
            (_u8) strlen(LISTEN_SERVICE_NAME),
            SL_AF_INET,
            (_u32 *) &pAddr, &usPort, &ulTextLen, &cText[0]);
    }

    // Upon success, 0 is returned and we just need to print out the information
    if(iretvalmDNS == 0)
    {
        cText[ulTextLen] = '\0';
        Report("First Response: Addr: %d.%d.%d.%d, Port: %d, Text: %s, TextLength: %d\n\r",
             SL_IPV4_BYTE(pAddr, 3), SL_IPV4_BYTE(pAddr, 2),
             SL_IPV4_BYTE(pAddr, 1), SL_IPV4_BYTE(pAddr, 0),
             usPort, cText, ulTextLen);
    }
}
```

12.3.5 mDNS 查询 – 连续

在连续 mDNS 查询中，收到一个响应并不一定意味着不会有更多相关响应，并且查询操作将继续进行，直到不再需要响应为止。一次性查询是默认设置，因此需要进行额外设置才能使用连续查询。但是，由于 *sl_NetAppDnsGetHostByService()* 一次只返回一台发现的主机，请使用 *sl_NetAppGetServiceList()* 获取完整的服务列表。下面是一个返回列表格式为 `SlNetAppGetFullServiceWithTextIpv4List_t` 的示例：

```
#define LISTEN_SERVICE_NAME     "_workstation._tcp.local"
#define SERVICE_GET_COUNT_MAX   3   // Specify the max number of services to discover
void mDNS_Query_Continuous()
{
    _u32 pAddr;
    _u32 usPort;
    _u16 ulTextLen;
    _i8  cText[200];
    _i32 iretvalmDNS = -1;
    _i32 i;

    // The structure to be used for retrieving the list
    SlNetAppGetFullServiceWithTextIpv4List_t serviceList[SERVICE_GET_COUNT_MAX];

    // Set to perform Continuous mDNS query
    iretvalmDNS = sl_NetAppSet(SL_NET_APP_MDNS_ID, NETAPP_SET_GET_MDNS_CONT_QUERY_OPT, 0, 0);
    Report("sl_NetAppSet() return: %d\n\r", iretvalmDNS);

    // Set an infinite loop until getting a response
    iretvalmDNS = 1;
    while(iretvalmDNS)
    {
        ulTextLen = 200;
        iretvalmDNS = sl_NetAppDnsGetHostByService(LISTEN_SERVICE_NAME,
                     (unsigned char) strlen(LISTEN_SERVICE_NAME), SL_AF_INET,
                     (_u32 *)&pAddr, &usPort,&ulTextLen,&cText[0]);
    }
    if(iretvalmDNS == 0)
    {
        // Prints the first response, but we want more later on
        cText[ulTextLen] = '\0';
        Report("First Response: Addr: %d.%d.%d.%d, Port: %d, Text: %s, TextLength: %d\n\r",
             SL_IPV4_BYTE(pAddr, 3), SL_IPV4_BYTE(pAddr, 2),
             SL_IPV4_BYTE(pAddr, 1), SL_IPV4_BYTE(pAddr, 0),
             usPort, cText, ulTextLen);
    }

    // Get a list of discovered services
    iretvalmDNS = sl_NetAppGetServiceList(0,
                  SERVICE_GET_COUNT_MAX,
                  SL_NET_APP_FULL_SERVICE_WITH_TEXT_IPV4_TYPE,
                  (_i8*) &serviceList[0],
                  1479);
    // 1479 is the maximum number of characters to return

    // The function returns the total number of discovered services
    // This number will never exceed the SERVICE_GET_COUNT_MAX parameter we passed into the
```

```
    // function
    Report("sl_NetAppSet() return: %d\n\r", iretvalmDNS);
    if(iretvalmDNS > 0)
    {
        // Now just prints all the services
        Report("Complete List:\n\r");
        for(i=0; i<iretvalmDNS; i++)
        {
            Report("Host %d: %s, IP: %d.%d.%d.%d, Name: %s Port: %d, Text: %s\n\r",
                   i,
                   serviceList[i].service_host,
                   SL_IPV4_BYTE(serviceList[i].service_ipv4, 3),
                   SL_IPV4_BYTE(serviceList[i].service_ipv4, 2),
                   SL_IPV4_BYTE(serviceList[i].service_ipv4, 1),
                   SL_IPV4_BYTE(serviceList[i].service_ipv4, 0),
                   serviceList[i].service_name,
                   serviceList[i].service_port,
                   serviceList[i].service_text);
        }
    }
}
```

12.3.6 mDNS 服务注册

除了默认的 HTTP，用户最多可以注册五个 mDNS 进行广播。无论 NWP 是否处于活动状态，注册都是持久的，因为注册信息持久存储在串行闪存中。分别使用 *sl_NetAppMDNSRegisterService()* 和 *sl_NetAppMDNSUnRegisterService()* 来注册和取消注册 mDNS 服务。

以下示例显示了注册过程：

```
#define SERVICE_NAME    "AAA._uart._tcp.local"
#define SERVICE_NAME2   "BBB._http._tcp.local"
void mDNS_Broadcast()
{
    int retVal = 0;

    // Start mDNS - not required since it's started by default.
    // However, it's a good practice just to ensure the mDNS is started.In this
    // case, -6 is returned.0 will be returned if it was stopped previously.
    retVal = sl_NetAppStart(SL_NET_APP_MDNS_ID);
    // Unregister first, then register to clean up previous registration
    // with the same service name.
    sl_NetAppMDNSUnRegisterService(SERVICE_NAME, (unsigned char) strlen(SERVICE_NAME));
    retVal = sl_NetAppMDNSRegisterService(SERVICE_NAME,
        (unsigned char)strlen(SERVICE_NAME),
        "Apple",
        (unsigned char)strlen("Apple"),
         200,
         2000,
         0x00000001);
    Report("MDNS 1 Registered with NAME: %s & ERROR code: %d\n\r",
         SERVICE_NAME,
         retVal);
    //Second registration with a different name
    sl_NetAppMDNSUnRegisterService(SERVICE_NAME2,(unsigned char) strlen(SERVICE_NAME2));
    retVal = sl_NetAppMDNSRegisterService(SERVICE_NAME2,
        (unsigned char)strlen(SERVICE_NAME2),
        "Banana",
        (unsigned char)strlen("Banana"),
        201,
        2000,
        0x00000001);
    Report("MDNS 2 Registered with NAME: %s & ERROR code: %d\n\r", SERVICE_NAME2, retVal);
}
```

若要一次取消注册所有 mDNS 服务，请使用以下调用函数：

```
sl_NetAppMDNSUnRegisterService(0, 0);
```

只需注册一次特定服务。每次复位后都会保存并广播该服务（如果 mDNS 已启动，且具有 IP 的 STA 或 P2P/AP 正常运行）。

- 还会发送对包含服务名称或类型的查询的响应。
- 如果不需要默认参数，则只需设置一次广播计时参数。

12.4 支持的功能

- 支持的功能
  - 完全支持 Bonjour
  - 注册和广播服务
  - 最多可广播五个外部服务
  - 内部服务广播：
    - 默认广播的与 mDNS 数据库无关的服务
    - 产品的内部功能
    - HTTP 服务器 – 对等器件可以在没有 DNS 服务器的情况下找到 IP 并进行浏览
- 响应
  - 对查询的响应
  - 忽略已对服务熟悉的查询
- 使用以下查询在本地网络中搜索服务：
  - 一次性查询
  - 连续查询
- 使用 GET 服务列表 API 收集有关对等服务的已知信息：
  - 带有文本的完整服务 – 名称、主机、IP、端口、文本
  - 不带文本的完整服务 – 名称、主机、IP、端口
  - 短服务 – IP、端口
- 掩蔽在响应或对等广播中收到的对等服务。
  - 设置广播计时。重新配置 mDNS 在发送服务公告时使用的计时参数：服务进行广播的次数和具体时间（与响应查询无关）。

12.5 限制

以下是 mDNS 特性的已知限制：

- 在对等缓存中最多支持或接收八项不同的服务（过滤器仅用于存储所需的服务）。
- 当删除所有服务后，mDNS 机器停止然后再启动；对等缓存被删除。
- 如果用户注册了唯一一项服务，但是网络中已经存在同名的服务，那么服务名称将更改为“名称 (编号)”，例如 PC1 (2)_ipp._tcp.local。在这种情况下，数据库中的名称是原始名称，但广播使用新名称。
- 有限的 Bonjour 支持。目前在广播模式下支持 Bonjour。查询模式仅支持 PTR，未来将实现对 Bonjour 的完全支持。
- 删除由于名称（广播名称和数据库名称）不匹配而更改的唯一名称会导致 mDNS 机器停止然后再启动，并删除对等缓存。
- 如果存在一次性查询但对等缓存已满，则没有地方设置该查询。系统将删除对等缓存，然后发送该查询。
- 存在局部检查，检查注册的服务名称是否合法。用户必须发送合法名称。
- 按服务使用 GET 主机时，只返回一个答案。若要查看所有答案，请等待所有对等答案均发送和接收完毕，然后使用 API GET 服务列表读取答案。
- API GET 服务列表的最大缓冲区列表大小为 1480 字节。对更大列表的请求会返回 SL_ERROR_NETAPP_RX_BUFFER_LENGTH_ERROR 代码。

章节 13

串行闪存文件系统

13.1 概述

CC31XX/CC32XX 系统附带一个多功能的串行闪存。该闪存可保存用于网络子系统、补丁和配置文件的所有相关系统文件。对于 CC32xx，还会保存用户应用程序代码。此外，该产品可以使用剩余的闪存区域作为保存用户数据文件的存储资源。

本段的目的是介绍该文件系统的常用 API 和配置，并说明文件系统的限制。

13.1.1 指令汇总

文件创建

说明

此操作用于创建一个新文件。根据创建请求，器件会检查串行闪存上的可用空间。未能为新文件分配空间会生成错误。此外，创建一个已经存在的文件也会生成错误。

API

```
_i32 sl_FsOpen (_u8 *pFileName,
                _u32 AccessModeAndMaxSize,
                _u32 *pToken,
                _i32 *pFileHandle);
```

参数

表 13-1. 参数

类型	参数	输入/输出	说明
_u8*	pFileName	输入	指向目标文件名的指针。以 NULL 结尾
_u32	AccessModeAndMaxSize	输入	大小和标志如下所述
_u32*	pToken	In	保留以供将来在安全文件系统中使用。应为 NULL
_i32*	pFileHandle	输出	所创建文件的句柄（如果返回 success）

可以使用以下 MACRO 来配置 AccessModeAndMaxSize：

FS_MODE_OPEN_CREATE(maxSizeInBytes,accessModeFlags)

- maxSizeInBytes：每个文件包含 512 字节的元数据。若要在串行闪存上实现理想的字节分配，请在以下公式中使用最大值：

  (((x*4096)- 512) % 粒度) * 粒度，其中 x 是整数，并且支持的粒度为 {256, 1024, 4096, 16384, 65536}
- accessModeFlags：只有 _FS_FILE_OPEN_FLAG_COMMIT 标志适用（失效防护标志）。保留所有其他标志以供将来在安全文件系统中使用

返回值

成功时，返回 0。出错时，返回错误代码。

示例

```
_u8         DeviceFileName[] = "MyFile.txt";
_u32        MaxSize = 63 * 1024;
_i32        DeviceFileHandle = -1;
_i32        RetVal;
RetVal = sl_FsOpen(DeviceFileName,
FS_MODE_OPEN_CREATE(MaxSize , _FS_FILE_OPEN_FLAG_COMMIT ),
                               NULL,
                               &DeviceFileHandle);
RetVal = sl_FsClose(DeviceFileHandle,
NULL,
NULL,
NULL );
```

限制

- 文件大小在创建时分配，以后不能扩展
- 创建带有 COMMIT 标志的文件会导致文件镜像。它仅用于失效防护目的。用户无法选择从哪个实例进行加载。
- 文件最大为 1MB
- 闪存的最小分配字节为 4KB，即一个数据块的大小

文件打开

说明

此操作可以打开一个现有文件进行读取或写入。根据打开请求，器件会检查串行闪存上是否存在该文件，如果不存在，则会生成错误。请注意，没有配置文件大小，因为其仅在创建文件时分配。

API

```
_i32 sl_FsOpen (_u8 *pFileName,
                _u32 AccessModeAndMaxSize,
                _u32 *pToken,
                _i32 *pFileHandle);
```

参数

类型	参数	输入/输出	说明
_u8*	pFileName	输入	指向目标文件名的指针。以 NULL 结尾
_u32	AccessModeAndMaxSize	输入	大小和标志如下所述
_u32*	pToken	输入	保留以供将来在安全文件系统中使用。应为 NULL
_i32*	pFileHandle	输出	所创建文件的句柄（如果返回 success）

可以使用以下 MACRO 来配置 AccessModeAndMaxSize：

- FS_MODE_OPEN_READ：用于文件读取
- FS_MODE_OPEN_WRITE：用于文件写入

返回值

成功时，返回 0。出错时，返回错误代码。

示例

```
_u8         DeviceFileName[] = "MyFile.txt";
_i32        DeviceFileHandle = -1;
_i32        RetVal;
RetVal = sl_FsOpen(DeviceFileName,
             FS_MODE_OPEN_READ,
             NULL,
             &DeviceFileHandle);
RetVal = sl_FsClose(DeviceFileHandle,
             NULL,
             NULL,
             NULL );
RetVal = sl_FsOpen(DeviceFileName,
             FS_MODE_OPEN_WRITE,
             NULL,
             &DeviceFileHandle);
RetVal = sl_FsClose(DeviceFileHandle,
             NULL,
             NULL,
             NULL );
```

限制

- 打开一个文件进行写入访问时，整个文件将被擦除

文件关闭

说明 此操作用于关闭现有文件。

API

```
_i16 sl_FsClose(_i32 pFileHandle,
                _u8* pCeritificateFileName,
                _u8* pSignature,
                _u32 SignatureLen);
```

参数

类型	参数	输入/输出	说明
_i32	pFileName	输入	所创建文件的句柄
_u8*	pCeritificateFileName	输入	保留以供将来在安全文件系统中使用。应为 NULL
_u8*	pSignature	In	保留以供将来在安全文件系统中使用。应为 NULL
_u32	SignatureLen	输入	保留以供将来在安全文件系统中使用。应为 NULL

返回值 成功时，返回 0。出错时，返回错误代码。

示例

```
_u8         DeviceFileName[] = "MyFile.txt";
_i32        DeviceFileHandle = -1;
_i32        RetVal;
RetVal = sl_FsOpen(DeviceFileName,
                   FS_MODE_OPEN_READ,
                   NULL,
                   &DeviceFileHandle);
RetVal = sl_FsClose(DeviceFileHandle,
                    NULL,
                    NULL,
                    NULL);
```

限制

- 无

文件写入

说明

此操作用于写入一个现有文件。设计偏移的主要目的是允许随机访问写入，而不像在“标准”文件系统中那样使用 fseek 方法。使用此方法，用户可以通过单个 API 调用来访问该文件中的任何位置。

出于写入目的，用户可以分块写入该文件，只要在打开该文件时相同的位置不重复两次，就不需要考虑块的顺序。

API

```
_i32 sl_FsWrite(_i32 FileHandle,
                _u32 Offset,
                _u8*  pData,
                _u32 Len);
```

参数

类型	参数	输入/输出	说明
_u32	FileHandle	输入	现有文件的句柄 (在 sl_FsOpen() 上返回)
_u32	偏移	输入	文件中的偏移量
_u32*	pData	输入	指向所写入数据的指针
_i32*	Len	输入	所写入数据的长度

返回值

成功时，返回 0。错误时，返回错误代码。

示例

```
_u8         DeviceFileName[] = "MyFile.txt";
_i32        DeviceFileHandle = -1;
_i32        RetVal;
_u32        Offset;
Offset = 0;
RetVal = sl_FsOpen(DeviceFileName,
                   FS_MODE_OPEN_WRITE,
                   NULL,
                   &DeviceFileHandle);
RetVal = sl_FsWrite( DeviceFileHandle,
                   Offset,
                   (_u8 *)"Hello World.",
                   strlen("Hello World."));
Offset = strlen("HelloWorld.");
RetVal = sl_FsWrite( DeviceFileHandle,
                   Offset,
                   (_u8 *)"This is a test.",
                   strlen("This is a test."));
RetVal = sl_FsClose(DeviceFileHandle,
                   NULL,
                   NULL,
                   NULL );
```

限制

文件附加部分没有嵌入到文件系统中。可以通过主机中的读取-修改-写入操作来进行附加。

文件读取

说明

此操作允许从一个现有文件进行读取。偏轴设计的目的主要是允许随机访问读取，而不像在“标准”文件系统中那样使用 fseek 方法。使用此方法，用户可以通过单个 API 调用来访问该文件中的任何位置。

API

```
_i32 sl_FsRead(_i32 FileHandle,
               _u32 Offset,
               _u8*  pData,
               _u32 Len);
```

参数

类型	参数	输入/输出	说明
_u32	FileHandle	输入	现有文件的句柄 (在 sl_FsOpen() 上返回)
_u32	偏移	输入	文件中的偏移量
_u32*	pData	输入	指向所写入数据的指针
_i32*	Len	输入	所写入数据的长度

返回值

成功时，返回 0。出错时，返回错误代码。

示例

```
_u8         DeviceFileName[] = "MyFile.txt";
_i32        DeviceFileHandle = -1;
_i32        RetVal;
_u8         ReadBuffer[100];
_u32        Offset;
Offset = 0;
RetVal = sl_FsOpen(DeviceFileName,
                   FS_MODE_OPEN_READ,
                   NULL,
                   &DeviceFileHandle);
RetVal = sl_FsRead( DeviceFileHandle,
                   Offset,
                   (_u8 *)ReadBuffer,
                   50);
Offset = 50;
RetVal = sl_FsWrite( DeviceFileHandle,
                   Offset,
                   (_u8 *)&ReadBuffer[50],
                   50);
RetVal = sl_FsClose(DeviceFileHandle,
                   NULL,
                   NULL,
                   NULL );
```

限制

文件附加部分没有嵌入到文件系统中。可以通过主机中的读取-修改-写入操作来进行附加。

文件删除

说明

此操作用于从串行闪存中删除一个现有文件。如果串行闪存上没有该文件，则会生成错误。删除一个文件后，其在串行闪存上占用的空间将被释放，可供系统重新分配。

API

```
_i16 sl_FsDel(_u8 *pFileName,
              _u32 Token);
```

参数

类型	参数	输入/输出	说明
_u8	pFileName	输入	指向目标文件名的指针。以 NULL 结尾
_u32	Token	输入	保留以供将来在安全文件系统中使用。应为 0

返回值

成功时，返回 0。出错时，返回错误代码。

示例

```
_u8     DeviceFileName[] = "MyFile.txt";
_u32    MaxSize = 63 * 1024;
_i32    DeviceFileHandle = -1;
_i32    RetVal;
_u32    Token;
RetVal = sl_FsOpen(DeviceFileName,
                   FS_MODE_OPEN_CREATE(MaxSize , _FS_FILE_OPEN_FLAG_COMMIT ),
                   NULL,
                   &DeviceFileHandle);
RetVal = sl_FsClose(DeviceFileHandle,
                   NULL,
                   NULL,
                   NULL );
Token = 0;
RetVal  = sl_FsDel(DeviceFileName,
                   Token);
```

限制

无

文件信息

说明 此操作可以获取现有文件的信息。如果串行闪存上没有该文件，则会生成错误。

API

```
_i16 sl_FsGetInfo (_u8 *pFileName,
                   _u32 Token,
                   SlFsFileInfo_t* pFsFileInfo);
```

参数

类型	参数	输入/输出	说明
_u8*	pFileName	输入	指向目标文件名的指针。以 NULL 结尾
_u32	Token	输入	保留以供将来在安全文件系统中使用。应为 0
SlFsFileInfo_t *	pFsFileInfo	输出	文件信息：标志、文件大小、分配的大小和令牌。请看下面的结构。

```
typedef struct
{
    _u16 flags;
    _u32  FileLen;
    _u32  AllocatedLen;
    _u32  Token[4];
}SlFsFileInfo_t;
```

- 标志留作将来使用
- FileLen 是文件的实际长度
- AllocatedLen 是文件创建期间请求的分配长度
- 令牌留作将来使用

返回值 成功时，返回 0。出错时，返回错误代码。

示例

```
_u8             DeviceFileName[] = "MyFile.txt";
_u32            MaxSize = 63 * 1024;
_i32            DeviceFileHandle = -1;
_i32            RetVal;
_u32            Token;
SlFsFileInfo_t  FsFileInfo;
RetVal = sl_FsOpen(DeviceFileName,
                   FS_MODE_OPEN_CREATE(MaxSize , _FS_FILE_OPEN_FLAG_COMMIT ),
                   NULL,
                   &DeviceFileHandle);
RetVal = sl_FsClose(DeviceFileHandle,
                   NULL,
                   NULL,
                   NULL );
Token = 0;
RetVal  = sl_FsGetInfo(DeviceFileName,
                   Token,
                   &FsFileInfo);
```

限制 无

文件系统限制

- 文件大小在创建时分配，以后不能扩展
- 文件附加部分没有嵌入到文件系统中。可以通过主机中的读取-修改-写入操作来进行附加。
- 打开一个文件进行写入访问时，整个文件将被擦除
- 创建带有 COMMIT 标志的文件会导致文件镜像。它仅用于失效防护目的。用户无法选择从哪个实例进行加载。
- 文件最大为 1MB。

文件信息 (continued)

- 闪存的最小分配字节为 4KB，即一个数据块的大小
- 碎片不适用。
- 文件系统不是目录结构。尽管如此，强烈建议保留目录前缀结构以提高可读性并便于维护。例如，所有用户文件都应以 /usr/ 前缀开头。

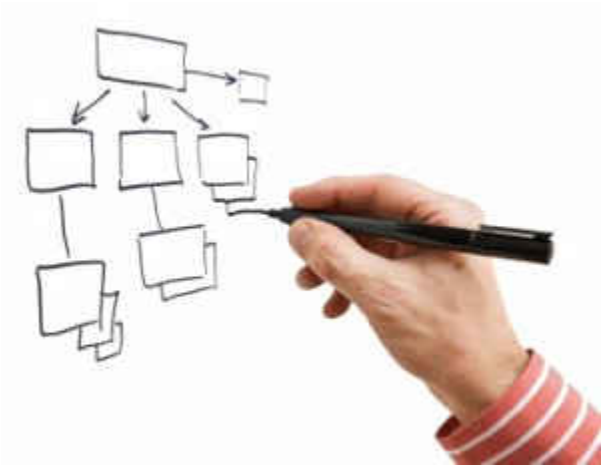
备注

将 www/ 前缀作为根文件夹，Web 服务器即可访问文件系统。

Rx 滤波器

14.1 概述

Rx 滤波器模块允许用户定义一组规则，通过这些规则确定 SimpleLink WiFi 器件将哪些接收到的帧传输到主机，以及丢弃了哪些帧。Rx 滤波器可以在 AP 连接期间以及混杂模式（“断开连接模式”）期间激活。



14.2 详细说明

每个接收到的帧都会遍历一系列用于确定如何处理帧的决策树。

决策树由筛选器节点组成。每个节点都有其筛选规则、操作和触发器。树遍历过程从树的根节点开始：如果根节点的筛选规则和触发为 TRUE，则执行根节点的操作，并且帧继续前往子节点。

筛选规则是特定的协议标头值：

- MAC 层：帧类型、帧子类型、BSSID、源 MAC 地址、目标 MAC 地址和帧长度
- LLC 层：协议类型
- 上层：IP 版本、IP 协议、源 IP 地址、目标 IP 地址、ARP 操作、ARP 目标 IP 地址、源端口号和目标端口号

可能的触发条件：

- 当角色为…（工作站/AP/混杂）时
- 当连接状态为…（已连接/已断开连接）时
- 当计数器达到 X 时

可能的操作：

- 丢弃数据包（不将它传递至主机）
- 增大或减小计数器值

当帧达到其中一个树中的 DROP 操作时，树遍历过程停止。在所有树中逐层完成遍历。

用户可以定义组合筛选节点。此节点有两个父节点（与只有一个父节点的常规节点不同），仅当其两个父节点中的一个或两个（用户定义）均为 TRUE 时才进行检查。例如：if (node_1 OR node_2)。

14.3 示例

示例 1：假设用户的要求如下：

- 仅从两个特定 MAC 地址接收 WLAN 数据广播帧。
- 接收所有 WLAN 单播帧，具有特定 SRC_IP 地址范围的帧除外。
- 如果从 MAC 地址 AA.AA.AA 接收到单播帧，则增大 counter_1。
- 如果从 MAC 地址 BB.BB.BB 接收到单播帧，则增大 counter_2。
- 如果从 MAC 地址 AA.AA.AA 或 BB.BB.BB 接收到单播 UDP 帧，则仅传递来自端口 5001 的数据包。

应创建以下树：（请参阅图 14-1）。

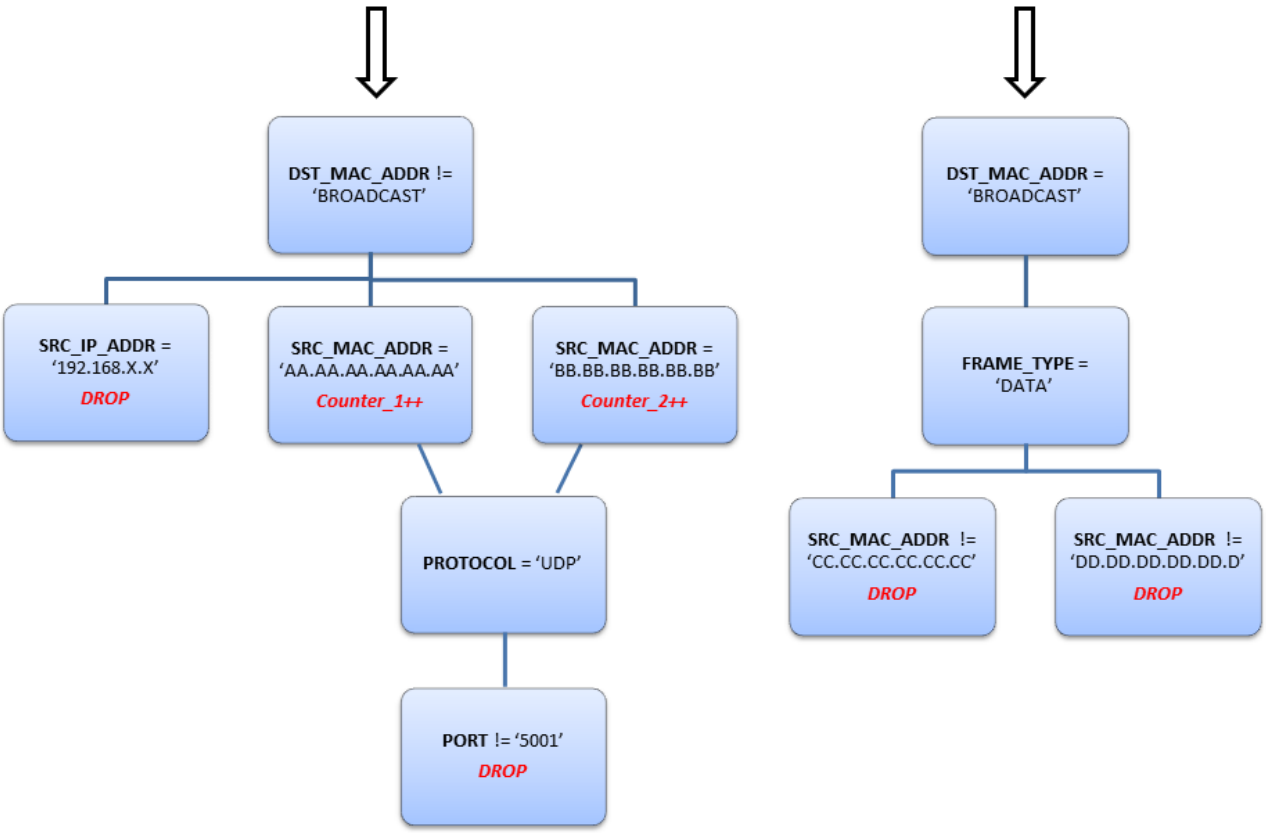


图 14-1. 树示例 1

示例 2：假设用户的要求如下：

- 从所有 MAC 地址仅接收 WLAN 管理信标帧。

应创建以下树：（请参阅图 14-2）。

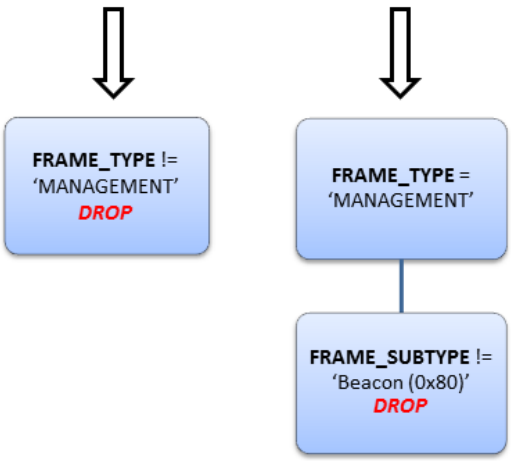


图 14-2. 树示例 2

14.4 创建树

- 树由用户创建。用户单独添加筛选器节点并定义筛选器树层次结构。
- 树也可以由系统在内部创建和应用。
- 筛选器可以创建为持久性筛选器，保存在闪存中并在系统启动时加载。
- 筛选器节点的最大数目为 64。系统使用 14 个筛选器节点；其余 50 个节点供用户使用。
- 可以创建、删除、启用和禁用筛选器。创建筛选器之后，必须启用它们以开始筛选。

14.5 主机 API

- WlanRxFilterAdd – 向系统添加新滤波器
- *sl_WlanRxFilterSet* – 为 Rx 滤波器可能的操作设置参数：
  - SL_ENABLE_DISABLE_RX_FILTER - 启用/禁用滤波器列表中的滤波器
  - SL_ENABLE_DISABLE_RX_FILTER - 启用/禁用滤波器列表中的滤波器
  - SL_STORE_RX_FILTERS - 持久保存滤波器
  - SL_UPDATE_RX_FILTER_ARGS - 更新现有滤波器的参数
  - SL_FILTER_PRE_PREPARED_SET_CREATE_REMOVE_STATE - 更改预先准备好的滤波器的默认创建方式
- *sl_WlanRxFilterGet* – 获取 Rx 过滤器可能操作的参数：
  - SL_FILTER_RETRIEVE_ENABLE_STATE - 检索启用/禁用状态
  - SL_FILTER_PRE_PREPARED_RETRIEVE_CREATE_REMOVE_STATE - 检索预先准备好的滤波器的创建状态

14.5.1 代码示例

```
INT32 AddBeaconFilter()
{
    SlrxFilterRuleType_t   RuleType;
    SlrxFilterID_t         FilterId = 0;
    SlrxFilterFlags_t      FilterFlags;
    SlrxFilterRule_t       Rule;
    SlrxFilterTrigger_t    Trigger;
    SlrxFilterAction_t     Action;
    SlrxFilterIdMask_t     FiltersIdMask
    char                   RetVal = -1;
    UINT8 FrameType;
    UINT8 FrameSubType;
    UINT8 FrameTypeMask;
    memset(FiltersIdMask, 0, sizeof(FiltersIdMask));
```

```
    /*
     * 第一个筛选条件是：
     * if FrameType != Management --> Action == Drop
     */
    /* 构建筛选参数 */
    RuleType = HEADER;
    FilterFlags.IntRepresentation = RX_FILTER_BINARY;
    FrameType     = 0; // 0-Management, 1-Control, 2-Data
    FrameTypeMask = 0xFF;
    Rule.HeaderType.RuleHeaderfield = FRAME_TYPE_FIELD;

memcpy(Rule.HeaderType.RuleHeaderArgsAndMask.RuleHeaderArgs.RxFilterDB1BytesRuleArgs[0],&FrameType,
1);
    memcpy(Rule.HeaderType.RuleHeaderArgsAndMask.RuleHeaderArgsMask,&FrameTypeMask, 1);
    Rule.HeaderType.RuleCompareFunc = COMPARE_FUNC_NOT_EQUAL_TO;
    Trigger.ParentFilterID = 0;
    Trigger.Trigger = NO_TRIGGER;
    Trigger.TriggerArgConnectionState.IntRepresentation =
RX_FILTER_CONNECTION_STATE_STA_NOT_CONNECTED;
    Trigger.TriggerArgRoleStatus.IntRepresentation = RX_FILTER_ROLE_PROMISCUOUS;
    Action.ActionType.IntRepresentation = RX_FILTER_ACTION_DROP;
    /* Add first Filter */
    RetVal = (char)sl_WlanRxFilterAdd(RuleType,FilterFlags,&Rule,&Trigger,&Action,&FilterId);
    if(RetVal == 0)
        printf("Filter created successfully, filter ID is %d\n",FilterId);
    else
{
        printf("Error creating the filter.Error number: %d.\n",RetVal);
        return -1;
   }
    SETBIT8(FiltersIdMask,FilterId);
    /*
     * 第二个筛选条件是：
     * if FrameType == Management --> Action == Do nothing
     */
    Rule.HeaderType.RuleCompareFunc = COMPARE_FUNC_EQUAL;
    Action.ActionType.IntRepresentation = RX_FILTER_ACTION_NULL;
    /* Add 2nd Filter */
    RetVal = (char)sl_WlanRxFilterAdd(RuleType,FilterFlags,&Rule,&Trigger,&Action,&FilterId);
    if(RetVal == 0)
        printf("Filter created successfully, filter ID is %d\n",FilterId);
    else
    {
        printf("Error creating the filter.Error number: %d.\n",RetVal);
        return -1;
    }
    SETBIT8(FiltersIdMask,FilterId);
    /*
     * 第三个筛选条件是：
     * if Frame SubType != Beacon --> Action == Drop
     */
    Trigger.ParentFilterID = FilterId; // The parent Id
    Rule.HeaderType.RuleCompareFunc = COMPARE_FUNC_NOT_EQUAL_TO;
    Action.ActionType.IntRepresentation = RX_FILTER_ACTION_DROP;
    FrameSubType = 0x80; // Beacon Frame SubType
    Rule.HeaderType.RuleHeaderfield = FRAME_SUBTYPE_FIELD;

memcpy(Rule.HeaderType.RuleHeaderArgsAndMask.RuleHeaderArgs.RxFilterDB1BytesRuleArgs[0],&FrameSubTyp
e, 1);
    /* Add 3rd Filter */
    RetVal = (char)sl_WlanRxFilterAdd(RuleType,FilterFlags,&Rule,&Trigger,&Action,&FilterId);
    if(RetVal == 0)
        printf("Filter created successfully, filter ID is %d\n",FilterId);
    else
    {
        printf("Error creating the filter.Error number: %d.\n",RetVal);
        return -1;
    }
    SETBIT8(FiltersIdMask,FilterId);
}
```

14.6 注意事项和限制

使用 Rx 滤波器数据库更新命令（添加/删除/更新/启用/禁用）时，必须关闭主机中的所有套接字。

当 SL 器件暴露于其套接字（标准或 RAW）上的传入接收时，不打算使用 Rx 滤波器数据库更新命令。

章节 15

收发器模式

15.1 一般说明

802.11 收发器模式是一款强大的工具，用于在第 2 层 (MAC) 或第 1 层（物理）发送和接收原始数据。

表 15-1 列出了八个网络层：

表 15-1. 网络层

应用
演示
会话
传输
网络
LLC – 逻辑链路控制
MAC – 介质访问控制
物理

使用从 802.11 报头开始的整个帧的空间来接收和传输数据。

在收发器模式下，没有帧确认或重试。因此，不能保证该帧会到达目的地。在 L1 模式下工作时，可能会与其他帧或能量发生碰撞。

图 15-1 显示了 802.11 帧结构。白色部分可由用户配置，灰色部分不能被覆盖。

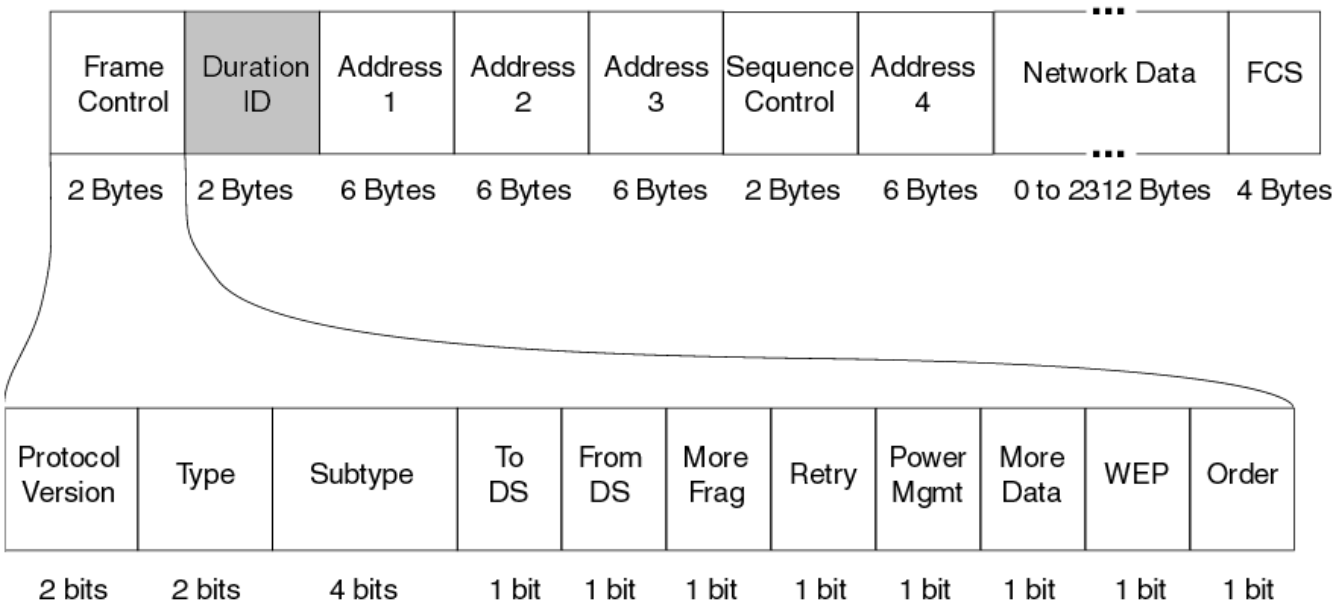


图 15-1. 802.11 帧结构

15.2 使用方式/API

使用控制 SimpleLink WiFi 器件（包括 SimpleLink Studio）的主机驱动程序，用户只需五个命令即可使用 802.11 收发器的功能。因为 SimpleLink WiFi 器件是符合 BSD 套接字实现的网络器件，所以应使用带有以下参数的命令 *sl_Socket* 来启动收发器。

```
soc = sl_Socket(SL_AF_RF ,SL_SOCK_RAW/SL_SOCK_DGRAM,channel);
```

SL_AF_RF – 指示用户可以从网络的哪一级覆盖帧。

SL_SOCK_RAW – 指示 L1 RAW 套接字（不遵守 802.11 介质访问策略 [CCA]）

SL_SOCK_DGRAM – 指示 L2 RAW 套接字（遵守 802.11 介质访问策略）

channel – 配置工作通道，以开始接收或发送流量。使用 0 来保持默认通道。

此命令返回一个套接字 ID，这是一个 2 字节整数，用于引用该套接字。如果套接字有问题，该命令将返回错误代码。

若要关闭套接字，请使用命令 *sl_Close*：

```
sl_Close(soc);
```

15.3 发送和接收

用户可以使用两个命令打开和关闭收发器。若要启动流量传输，请使用 *sl_Send* 命令进行传送，而使用 *sl_Recv* 命令进行接收。

```
sl_Send(soc ,RawData ,len ,flags);
```

soc – 套接字 ID。

RawData – 字符 *array 用于保存要发送的数据（从 802.11 MAC 标头的第一个字节开始）。

len – 数据的大小（以字节为单位）。

flags – 通常用户将此参数设置为 0，但用户可以使用此参数更改任何默认的 channel/rate/tx-power/11b-preamble 参数。使用 SL_RAW_RF_TX_PARAMS 宏来指定提到的参数。

Returns – 已发送的字节数。

例如，若要使用 1MBPS 数据速率以及 Tx 功率设置 1 和短前导码（仅对 11b 有效）以在通道 1 上传输帧，请使用：

```
sl_Send(soc,buf,len,SL_RAW_RF_TX_PARAMS(CHANNEL_1, RATE_1M,1, TI_SHORT_PREAMBLE));
```

```
sl_Recv(soc,buffer,500,0);
```

soc – 套接字 ID。

buffer – 字符 *array 用于包含接收到的数据包。

500 – 接收到的数据包的最大大小。最大大小为 1472：如果数据包的大小超过此值，则会丢弃多余部分。

最后一个参数应始终为 0，表示没有标志。

Return – 已接收的字节数。

15.4 更改套接字属性

命令 *sl_SetSockOpt* 用于更改套接字属性（在开放套接字之后）。

示例：

更改工作通道：

```
sl_SetSockOpt(soc, SL_SOL_SOCKET, SO_CHANGE_CHANNEL, &channel,1);
```

更改默认 PHY 数据速率：

```
sl_SetSockOpt(soc, SL_SOL_PHY_OPT, SO_PHY_RATE, &rate,1);
```

更改默认 Tx 功率：

```
sl_SetSockOpt(soc, SL_SOL_PHY_OPT, SO_PHY_TX_POWER, &power,1);
```

更改待发送的帧数（请参阅节 15.5）：

```
sl_SetSockOpt(soc, SL_SOL_PHY_OPT, SO_PHY_NUM_FRAMES_TO_TX, &numFrames,1);
```

更改 802.11b 前导码：

```
sl_SetSockOpt(soc, SL_SOL_PHY_OPT, SO_PHY_PREAMBLE,&preamble,,1);
```

15.5 内部数据包发生器

出于测试目的，SimpleLink WiFi 器件中带有一个内部数据包发生器，能够重复预定义的数据模式。

若要使用该发生器，请在调用 *sl_Send* 之前，使用 *sl_SetSockOpt* 将帧数设置为 0（无限数量的帧），或设置为需要传输的给定帧数。

然后，使用针对 *sl_Send* API 的单个调用来触发帧传输。

SimpleLink WiFi 器件会一直传输，直到它发送完所有请求的帧，或者直到套接字关闭或另一个套接字属性发生改变（如果使用了无限数量的帧）。

15.6 发送 CW（载波）

若要发送载波信号，请使用具有 NULL 缓冲区和 0（零）长度的 *sl_Send* API。

使用 *sl_Send* API 中的 flags 参数来表示子载波偏移（-25 至 25）。

通过触发另一个 flags = 128（十进制）的 *sl_Send* API 可停止 CW 发送，如下所示：*sl_Send (soc,NULL,0,128);*

15.7 连接策略和收发器模式

若要使用收发器模式，请禁用以前可能会尝试自动连接到 AP 的连接策略。

示例：

```
sl_WlanPolicySet(SL_POLICY_CONNECTION,SL_CONNECTION_POLICY(0,0,0,0,0),NULL,0);
```

```
sl_WlanDisconnect();
```

15.8 关于接收和发送的注意事项

15.8.1 接收

SimpleLink 专有无线电标头附加到正在接收的数据包。该标头具有一些关于数据包的信息。该标头的结构如下。

```
typedef struct
}
     UINT8      rate;              /* 接收的速率格式   */
     UINT8      channel;           /* 接收的通道 */
     INT8       rssi;              /* 当前帧的 RSSI 值(db) */
     UINT8      padding;           /* 填充以对齐到 32 位 */
     UINT32     timestamp;         /* 时间戳（以微秒为单位），  */
}TransceiverRxOverHead_t;
```

rate（速率）是从 0 到 20 的索引，顺序如下：

RATE 1M = 0

RATE 2M = 1

RATE 5.5M = 2

RATE 11M = 3

RATE 6M = 4

RATE 9M = 6

RATE 12M = 7

RATE 18M = 8

RATE 24M = 9

RATE 36M = 10

RATE 48M = 11

RATE 54M = 12

RATE MCS_0 = 13

RATE MCS_1 = 14

RATE MCS_2 = 15

RATE MCS_3 = 16

RATE MCS_4 = 17

RATE MCS_5 = 18

RATE MCS_6 = 19

RATE MCS_7 = 20

通道为 1 至 11。

如果使用 *sl_Recv* 命令使帧进入缓冲区，则应通过将缓冲区的开头转换为 *TransceiverRxOverHead_t* 类型的指针变量来提取标头

```
frameRadioHeader = (TransceiverRxOverHead_t *)buffer;
```

15.9 用例

可以使用此功能开发以下重要应用程序。

15.9.1 嗅探器

收发器可用作嗅探器。打开一个套接字并使用 *sl_Recv* 命令在循环中接收数据包。以下代码描述了如何捕获帧并在 Wireshark 中显示它们：

```
void Sniffer(Channel_e channel)
{
    INT16 soc;
    char buffer[1536];
    int counter = 0;
    int recievedBytes = 0;
    long count = 0;
    long long bytesSent = 0;
    DWORD startTick = 0;
    int fConnected = 0;
    HANDLE hPipe = NULL;
    LPTSTR lpszPipename = TEXT("\\\\.\\pipe\\cc3100");
    DWORD byteWritten;
    DWORD result;
    wireSharkGlobalHeader_t gHeader;
```

图 15-2. 嗅探器

```
    frameRadioHeader_t frame;
    TransceiverRxOverHead_t *frameRadioHeader;

/********************************** Creating Named Pipe for WireShark ********************/
// open a named pipe between this program and wireshark so packets could be sent to it
         hPipe = CreateNamedPipe(lpszPipename,
        PIPE_ACCESS_OUTBOUND,
        PIPE_TYPE_MESSAGE | PIPE_WAIT,
        PIPE_UNLIMITED_INSTANCES,
        65536,
        65536,
        NMPWAIT_USE_DEFAULT_WAIT,
        NULL);


    printf("waiting for connection.... \n(you should add a pipe interface in wireshark name
\\\\.\\pipe\\cc3100) \n");
    fConnected = ConnectNamedPipe(hPipe,NULL);
    printf("connection done...\n");

/********************************** Sending the global header for wire shark **************/
// this is global header for wireshark, to configure the type of packets it going to receive
// for more info check online for pcap format
    gHeader.magic_number = 0xa1b2c3d4;
    gHeader.version_major = 2;
    gHeader.version_minor = 4;
    gHeader.thiszone = 0;
    gHeader.sigfigs = 0;
    gHeader.snaplen = 0x0000FFFF;
    gHeader.network = 127;
    result = WriteFile(hPipe,&gHeader,sizeof(gHeader),&byteWritten,NULL);

/********************************** Receiving frames from the CC3100 and sending it to
wireshark*****/

// open the Transceiver
    soc = sl_Socket(SL_AF_RF,SL_SOCK_RAW,channel);
    while(1)
    {
// start receiving the packets
        recievedBytes = sl_Recv(soc,buffer,1536,0);
// get the receive radio header so we could present its info in wireshark
        frameRadioHeader = (TransceiverRxOverHead_t *)buffer;

// wireshark has its own header to present wifi frames, here we prepare the header, so we could send
// it before the frame
// check online for 80211 radio header for pcap format.
// here you can put the capture time in windows format
        frame.ts_sec = 190285;
        frame.ts_usec = 111284;
// here we put the length of the packet, the 24 is this header length and we decrease the radio
// header which is 8 bytes
        frame.incl_len = 24 + recievedBytes - 8;
        frame.orig_len = 24 + recievedBytes - 8;
        frame.it_version = 0;
        frame.it_pad = 0;
        frame.it_len = 24;
        frame.it_present = 0x0000002d;
// present the timestamp
        frame.tsf_low = frameRadioHeader->timestamp;
        frame.tsf_high = 0;
// present the rate
        frame.rate = RateIndexToRate[frameRadioHeader->rate];
        frame.pad1 = 0x11;
// present the channel
        frame.channel_low = ChannelToFrequency((Channel_e)frameRadioHeader->channel);
        frame.channel_high = 0x0080;
// present the rssi
        frame.antena = frameRadioHeader->rssi;
        frame.pad11 = 0x44;

// send the wireshark frame header
        result = WriteFile(hPipe,&frame,sizeof(frame),&byteWritten,NULL);
// send the frame minus the 8 bytes radio overhead
        result = WriteFile(hPipe,&(buffer[8]),(recievedBytes - 8),&byteWritten,NULL);

    }
    sl_Close(soc);
}
```

图 15-3. 嗅探器

15.10 持续发送

此应用持续发送同一个数据包。此应用使用 Rx 统计信息功能来标记和度量丢失的数据。以下代码展示了如何使用此功能：

```
void TxContinues(Channel_e channel,RateIndex_e rate,UINT32 numberOfPackets,DWORD intervalMiliSec)
{
    INT16 soc;
    UINT32 i;

        //set the rate in the packets header
    RawData_Ping[0] = (char)rate;

        //start the transceiver on the selected channel
    soc = sl_Socket(SL_AF_RF,SL_SOCK_RAW,channel);

        //transmit N packets with the delay chosen
    for(i = 0 ; i < numberOfPackets ; i++)
    {
        sl_Send(soc,RawData_Ping,sizeof(RawData_Ping),0);
        Sleep(intervalMiliSec);
    }

        // close the transceiver
    sl_Close(soc);
}
```

图 15-4. 持续发送

15.11 Ping

为了说明如何在应用程序级别构建 RAW 数据包，下面的过程演示了如何构建采用 ICMP、IP 和 MAC 协议封装的 PING 数据包。

1. 首先，需要通过 WLAN PHY 传输 PING 消息。

"PING data to be sent"

```
0x41, 0x08, 0xBB, 0x8D, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
```

图 15-5. 待发送的 Ping 数据

2. PING 是 ICMP QUERY 消息，因此，应首先用 ICMP 标头进行封装。

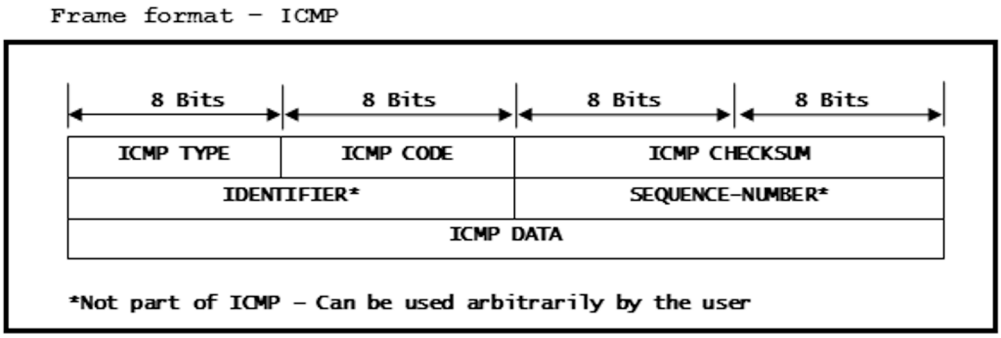


图 15-6. 帧格式 - ICMP

a. 将 ICMP TYPE 设置为 0x08，因为这是一条“Echo-Request”消息。
b. 对于 PING 消息，ICMP CODE 将始终为 0x00。
c. ICMP CHECKSUM 用于标头和数据，对于我们的消息而言是“0xA5, 0x51”。
d. ICMP DATA 是上面定义的“待发送的 PING 数据”。

3. 在将用 ICMP 封装的 PING 命令发送到 MAC 层之前，应将消息封装成 IP 数据报。图 15-7 展示了 IPv4 数据包的帧格式。

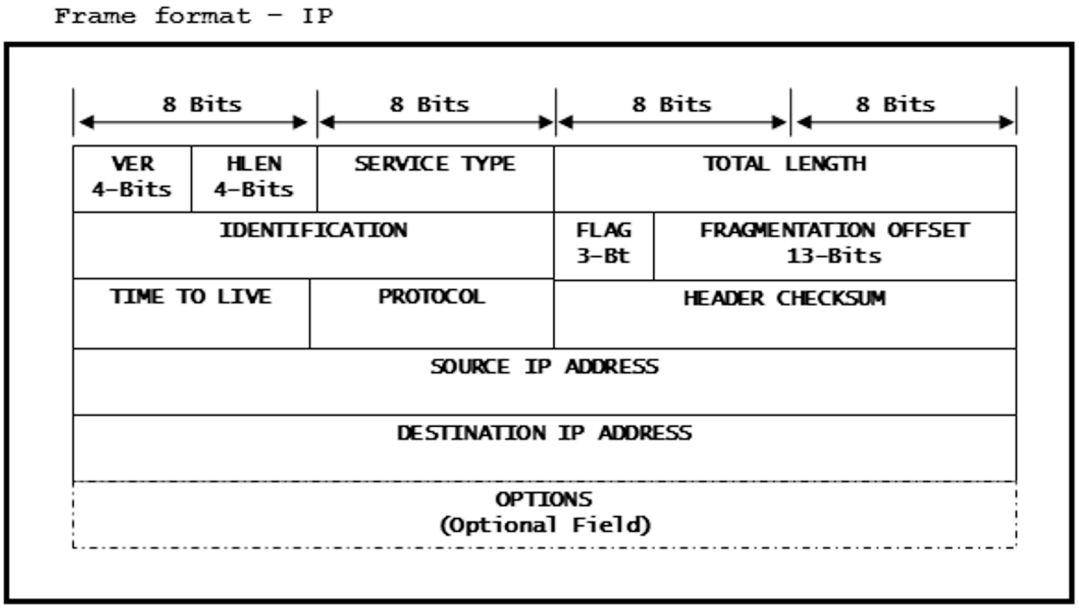


图 15-7. 帧格式 - IP

a. 将 VER 设置为 4，因为它是 IPv4 数据包。
b. HLEN 是以“lwords”表示的标头长度；由于标头长度为 20 字节，此处将该值设置为 5。
c. 将 SERVICE TYPE 设置为 0x00，因为 ICMP 是普通服务。
d. 将 TOTAL LENGTH 设置为“0x00 0x54”字节，其中包括标头和数据长度。
e. 将 VENDOR ID 设置为“0x00, 0x00, 0x00”。
f. IDENTIFICATION 是从该源 IP 发送的所有数据报的唯一身份；针对我们的数据包，将其设置为“0x96, 0xA1”。
g. 将 FLAG 和 FRAGMENTATION OFFSET 设置为“0x00, 0x00”，因为不需要对数据包进行分段。原因：此处发送的数据包小于 WLAN 帧大小。
h. 将 TIME TO LIVE 设置为 0x40，因为数据包将在 64 跳后被丢弃。
i. 将 PROTOCOL 设置为 0x01，因为它是 ICMP 数据包。
j. HEADER CHECKSUM 是 2 字节，在本例中设置为“0x57, 0xFA”。
k. 将 SOURCE IP ADDRESS 设置为“0xc0, 0xa8, 0x01, 0x64”（即 192.168.1.100）。
l. 将 DESTINATION IP ADDRESS 设置为“0xc0, 0xa8, 0x01, 0x65”（即 192.168.1.101）。
m. 将 OPTIONS 字段留空。

4. 接下来，应将 LLC 和 MAC 标头添加到 IP 封装消息中。图 15-8 展示了 LLC+SNAP 标头的帧格式：

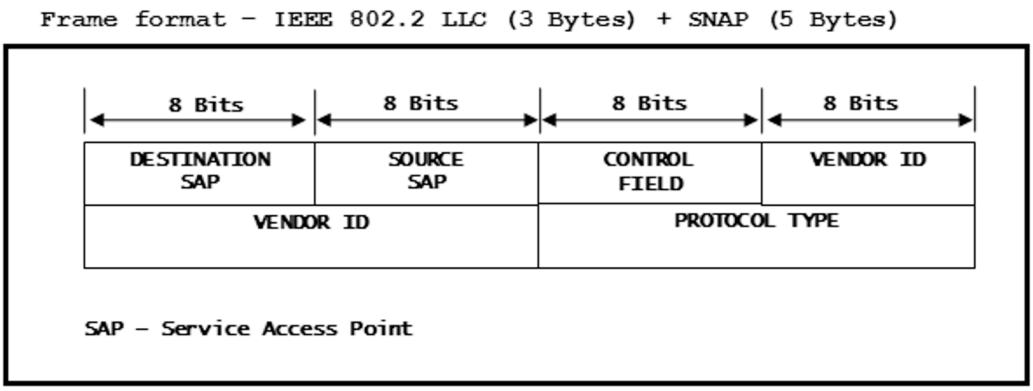


图 15-8. 帧格式 - IEEE 802.2 LLC（3 字节）+ SNAP（5 字节）

a. 将 DSAP 设置为 0xAA 以指定这是一个 SNAP 帧。
b. 将 SSAP 设置为 0xAA 以指定这是一个 SNAP 帧。
c. 对于子网访问协议 (SNAP)，将 CONTROL FIELD 设置为 0x03。
d. 将 VENDOR ID 设置为“0x00, 0x00, 0x00”。
e. 将 PROTOCOL TYPE 设置为“0x08, 0x00”；这表示第 4 层协议是 IP 协议。

5. 消息最终用 MAC 标头进行封装。图 15-9 展示了 802.11 MAC 标头的帧格式。

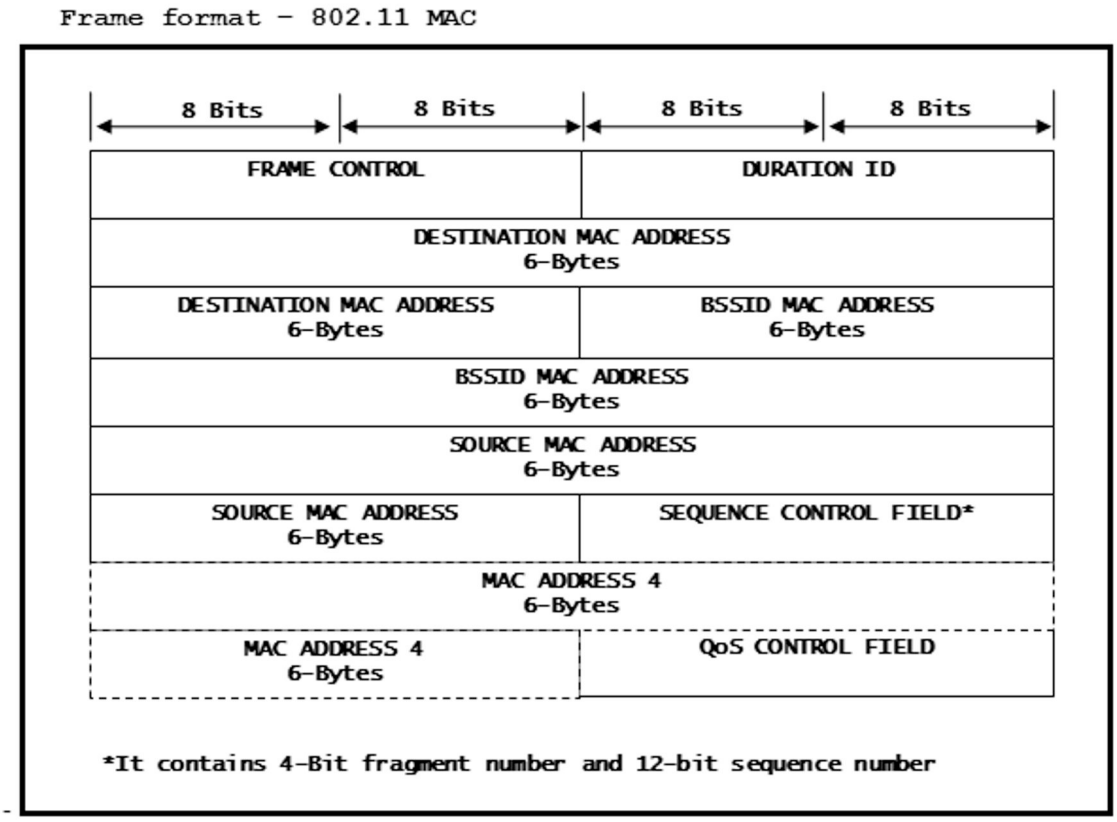


图 15-9. 帧格式 - 802.11 MAC

由于采用以下配置，将 FRAME CONTROL 设置为“0x88, 0x02”：

a. 将用于协议版本的 2 位设置为 00 以表示 802.11 标准。
b. 将用于类型的 2 位设置为 10 以表示“数据”。

c. 将用于子类型的 4 位设置为 1000 以表示“QoS 数据”。
d. Frame Control 字段 0x02 表示该帧来自分布式系统。
e. 将 DURATION ID 设置为 44 µS。
f. 将 DESTINATION MAC ADDRESS 设置为“0x00, 0x23, 0x75, 0x55, 0x55, 0x55”。
g. 将 BSSID MAC ADDRESS 设置为“0x00, 0x22, 0x75, 0x55, 0x55, 0x55”。
h. 将 SOURCE MAC ADDRESS 设置为“0x00, 0x22, 0x75, 0x55, 0x55, 0x55”。
i. 将 SEQUENCE CONTROL 字段设置为“0x42, 0x80”。
   i. 将 Sequence Number 设置为 1064，这是从该源 MAC 发送的所有数据报的唯一身份。
   ii. 将 Fragmentation Number 设置为 0，表示不需要分段。

6. 数据包现在用 ICMP、IP 和 MAC 标头进行封装，并准备好通过 WLAN PHY 进行传输。以下是完整的数据包。

```
RawPingPacket[] = {
                /*---- 802.11 MAC 标头 -----*/
                        0x88,
                        0x02,
                        0x2C, 0x00,
                        0x00, 0x23, 0x75, 0x55,0x55, 0x55,
                        0x00, 0x22, 0x75, 0x55,0x55, 0x55,
                        0x00, 0x22, 0x75, 0x55,0x55, 0x55,
                        0x80, 0x42, 0x00, 0x00,
    /*---- LLC & SNAP Header -----*/
                        0xAA, 0xAA, 0x03, 0x00, 0x00, 0x00, 0x08, 0x00,
                        /*---- IP 标头 -----*/
                        0x45, 0x00, 0x00, 0x54, 0x96, 0xA1, 0x00, 0x00, 0x40, 0x01,
                        0x57, 0xFA,
                        0xc0, 0xa8, 0x01, 0x64,
                        0xc0, 0xa8, 0x01, 0x65,
                        /*---- ICMP Header -----*/
                        0x08, 0x00, 0xA5, 0x51,
                        0x5E, 0x18, 0x00, 0x00,
                /*----有效负载 -----*/
                        0x41, 0x08, 0xBB, 0x8D, 0x00, 0x00,
                        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                        0x00, 0x00, 0x00, 0x00
                        };
```

7. 若要解码数据包，可以对接收到的响应进行解包并注意以下事项：
   a. CC3x00 器件将在接收到的数据包中添加 8 字节的专有无线电标头。标头将包含有关数据包的重要信息。
      i. 1 字节（无符号字符）用于表示速率：这是数据接收速率。
      ii. 1 字节（无符号字符）用于表示通道：这是接收数据的通道号。
      iii. 1 字节（有符号字符）用于表示 RSSI：这是为当前帧计算的 RSSI 值（以 dB 为单位）。
      iv. 1 字节用于表示填充位：这个字节用于实现 4B 对齐。
      v. 4 字节（无符号长整数）用于表示时间戳：这是接收到的数据包的时间戳（以 µS 为单位）。接收到的消息用系统时间标记，默认情况下从 1-1-2000 开始。
   b. 实际数据将位于此 8 字节标头之后，应用程序开发人员应根据相关协议对其进行解析。
   c. 考虑到在接收的数据包中添加了 8 字节专有无线电标头，802.11 MAC 标头从接收的数据包的偏移 8 处开始。
      i. DESTINATION MAC ADDRESS（6 字节信息）位于 802.11 MAC 标头的偏移 4 处，因此可以从接收的数据包的偏移 12 处开始提取。
      ii. SOURCE MAC ADDRESS（6 字节信息）位于 802.11 MAC 标头的偏移 16 处，因此可以从接收的数据包的偏移 24 处开始提取。
   d. 考虑到有 8 字节专有无线电标头、26 字节 MAC 标头以及 8 字节 LLC 和 SNAP 标头，IP 标头从接收的数据包的偏移 42 处开始。

i. SOURCE IP ADDRESS（4 字节信息）位于“IP 标头”的偏移 12 处，因此可以从接收的数据包的偏移 54 处开始提取。
ii. DESTINATION IP ADDRESS（4 字节信息）位于“IP 标头”的偏移 16 处，因此可以从接收的数据包的偏移 58 处开始提取。

e. 数据包的其余部分可以按照下一级协议进行解码，最终的解码代码如下所示：

```
typedef struct {
   UINT8   rate;
   UINT8   channel;
   INT8    rssi;
   UINT8   padding;
   UINT32 timestamp;
}
TransceiverRxOverHead_t;
void TransceiverModeRx (INT8 <channel_number>, INT32 <pkts_to_receive>) {
   TransceiverRxOverHead_t *frameRadioHeader = NULL;
   UINT8 buffer[BUFFER_SIZE] = {'\0'};
   INT32 <socket_hanlde> = -1;
   INT32 recievedBytes = 0;
   <socket_hanlde>= sl_Socket(SL_AF_RF, SL_SOCK_RAW, <channel_number>);
   while(<pkts_to_receive>--)
   {
       memset(&buffer[0], 0, sizeof(buffer));
       recievedBytes = sl_Recv(<socket_hanlde>, buffer, BUFFER_SIZE, 0);
       frameRadioHeader = (TransceiverRxOverHead_t *)buffer;
       PRINT(" ===>>> Timestamp: %iuS, Signal Strength: %idB\n\r", frameRadioHeader->timestamp,
frameRadioHeader->rssi);
       PRINT(" ===>>> Destination MAC Address: %02x:%02x:%02x:%02x:%02x:%02x\n\r", buffer[12],
buffer[13], buffer[14], buffer[15], buffer[16], buffer[17]);
       PRINT(" ===>>> Source MAC Address: %02x:%02x:%02x:%02x:%02x:%02x\n\r", buffer[24],
buffer[25], buffer[26], buffer[27], buffer[28], buffer[29]);
       PRINT(" ===>>> Source IP Address: %d.%d.%d.%d\n\r", buffer[54],  buffer[55], buffer[56],
buffer[57]);
       PRINT(" ===>>> Destination IP Address: %d.%d.%d.%d\n\r", buffer[58],  buffer[59],
buffer[60],  buffer[61]);
   }
   sl_Close(<socket_handle>);
```

15.12 收发器模式限制

- 用户可以在系统中打开一个收发器套接字。
- 如果接收到的数据包长度超过 1472 字节，则会对其进行修整。
- 无法传输超过 1472 字节和低于 14 字节的帧。
- 如果使用 SimpleLink Studio，注意 SPI 的最大吞吐量为 5~6Mbps，所以在拥挤的 WIFI 介质中会有丢包的情况。
- 收发器不会在连接中或已连接模式下打开。请注意，即使未连接，也会将自动连接模式视为已连接模式。

This page intentionally left blank.

Rx 统计信息

16.1 一般说明

Rx 统计信息特性用于确定有关介质和 SimpleLink WiFi 器件 RX 机制的某些重要参数。

Rx 统计信息提供关于以下方面的数据：

- RSSI – 接收功率以 dbm 为单位。
  - RSSI 直方图，从 -40 到 -87dbm，精度为 8dbm
  - 平均 RSSI 分为 DATA + CTRL / MANAGEMENT
- 接收的帧 – 分为有效帧、FCS 错误帧和 PLCP 错误帧
- 速率直方图 – 创建所有 BGN 速率为 1Mbps-MCS7 的直方图

16.2 使用方式/API

共有三个命令可以获取所需的统计信息。第一个命令是 *sl_WlanRxStatStart ()*，开始收集有关所有 Rx 帧的数据。

sl_WlanRxStatStop()，停止收集数据。

sl_WlanRxStatGet()，获取已收集的统计信息并以变量类型 *SlGetRxStatResponse_t* 返回统计信息。该命令的使用方式如下：

```
SlGetRxStatResponse_t rxStatResp;
sl_WlanRxStatGet
(&rxStatResp , 0 );
```

第二个参数已标记，当前未使用。

SlGetRxStatResponse_t 包含以下变量：

ReceivedValidPacketsNumber – 保存接收到的有效数据包的数量

ReceivedFcsErrorPacketsNumber – 保存丢弃的 FCS 错误数据包的数量

ReceivedAddressMismatchPacketsNumber - 保存已接收但被其中一个硬件过滤器滤除的数据包的数量

在连接模式下：

- 只有在调用 *sl_WlanRxStatStart* 后，数据才有效
- 计数器将指示由其中一个硬件过滤器滤除的帧数 – 因为这是连接模式，大多数数据包都将基于不匹配的地址

在收发器模式（断开连接模式）下：

- 只有在调用 *sl_WlanRxStatStart* 后，数据才有效
- 计数器将指示由其中一个硬件过滤器滤除的帧数
- 在 Rx 收发器中配置一个过滤器（MAC 地址、帧类型等），以便查看该计数器的递增情况

avarageDataCtrlRssi – 保存平均数据 + 控制帧 RSSI

avarageMgMntRssi – 保存平均管理帧 RSSI

RateHistogram[NUM_OF_RATE_INDEXES] – 直方图，描述接收到的有效帧的所有速率。速率排序如下：

RATE_1M = 0 …

RATE_2M

RATE_5_5M

RATE_11M

RATE_6M

RATE_9M

RATE_12M

RATE_18M

RATE_24M

RATE_36M

RATE_48M

RATE_54M

RATE_MCS_0

RATE_MCS_1

RATE_MCS_2

RATE_MCS_3

RATE_MCS_4

RATE_MCS_5

RATE_MCS_6

RATE_MCS_7

NUM_OF_RATE_INDEXES 为 21。

RssiHistogram[SIZE_OF_RSSI_HISTOGRAM] – 直方图，保存从 -40dbm 到 -87dbm 的所有接收到的数据包的累积 RSSI（每 8dbm）。

SIZE_OF_RSSI_HISTOGRAM 为 6

StartTimeStamp – 保存开始收集的时间，以微秒为单位

GetTimeStamp – 保存统计信息获取时间，以微秒为单位

16.3 关于接收和发送的注意事项

每个超出 RSSI 直方图上限或下限（-40dbm 或 -87dbm）的数据包分别在高压电芯或低压电芯中累积。

每次调用 *sl_WlanRxStatGet* 都会将统计数据库复位。

当只调用 *sl_WlanRxStatGet* 而不使用启动命令时，只有 RSSI 可用。

16.4 用例

Rx 统计信息功能检查介质的拥塞情况和距离，验证射频硬件，并使用 RSSI 信息将 SimpleLink WiFi 器件定位在理想位置。

示例：

将 SimpleLink 器件连接到 AP，运行从 AP 到器件的 UDP 数据包流，然后在 SimpleLink Studio 中使用以下代码获取 Rx 统计信息。

```
static _i32 RxStatisticsCollect(_i16 channel)
{
    SlGetRxStatResponse_t rxStatResp;
    _u8 buffer[MAX_BUF_RX_STAT] = {'\0'};
    _u8 var[MAX_BUF_SIZE] = {'\0'};
    _i32 idx = -1;
    _i16 sd = -1;
    _i32 retVal = -1;
    memset(&rxStatResp, 0, sizeof(rxStatResp));
    sd = sl_Socket(SL_AF_RF,SL_SOCK_RAW,channel);
    if(sd < 0)
    {
        printf("Error In Creating the Socket\n");
        ASSERT_ON_ERROR(sd);
    }
```

```
    retVal = sl_Recv(sd, buffer, BYTES_TO_RECV, 0);
    ASSERT_ON_ERROR(retVal);
    printf("Press \"Enter\" to start collecting statistics.\n");
    fgets((char *)var, sizeof(var), stdin);

    retVal = sl_WlanRxStatStart();
    ASSERT_ON_ERROR(retVal);
    printf("Press \"Enter\" to get the statistics.\n");
    fgets((char *)var, sizeof(var), stdin);

    retVal = sl_WlanRxStatGet(&rxStatResp, 0);
    ASSERT_ON_ERROR(retVal);
    printf("\n\n**********************Rx Statistics**********************\n\n");
    printf("Received Packets - %d\n",rxStatResp.ReceivedValidPacketsNumber);
    printf("Received FCS - %d\n",rxStatResp.ReceivedFcsErrorPacketsNumber);
    printf("Received Address Mismatch - %d\n",rxStatResp.ReceivedAddressMismatchPacketsNumber);
    printf("Average Rssi for management: %d    Average Rssi for other packets: %d\n",
                    rxStatResp.AvarageMgMntRssi,rxStatResp.AvarageDataCtrlRssi);
    for(idx = 0 ; idx < SIZE_OF_RSSI_HISTOGRAM ; idx++)
    {
        printf("Rssi Histogram cell %d is %d\n", idx, rxStatResp.RssiHistogram[idx]);
    }
    printf("\n");
    for(idx = 0 ; idx < NUM_OF_RATE_INDEXES; idx++)
    {
        printf("Rate Histogram cell %d is %d\n", idx, rxStatResp.RateHistogram[idx]);
    }
    printf("The data was sampled in %u microseconds.\n",
          ((_i16)rxStatResp.GetTimeStamp - rxStatResp.StartTimeStamp));
    printf("\n\n**********************End Rx Statistics**********************\n");
    retVal = sl_WlanRxStatStop();
    ASSERT_ON_ERROR(retVal);
    retVal = sl_Close(sd);
    ASSERT_ON_ERROR(retVal);
    return SUCCESS;
}
```

16.5 Rx 统计信息限制

RSSI 和速率的最大直方图单元容量为 65535。接收到更多数据包时，累积将停止。所有其他统计信息都保存在无符号整数中，当超过最大值时将绕回。

API 概述

本章讨论了不同的 SimpleLink API。本章不涉及参数的数量、参数类型或返回值。如需了解这方面的信息，请参阅 API doxygen 指南。

这些 API 分为六个大组：

- 器件
- NetConfig
- WLAN
- 套接字
- NetApp
- 文件系统

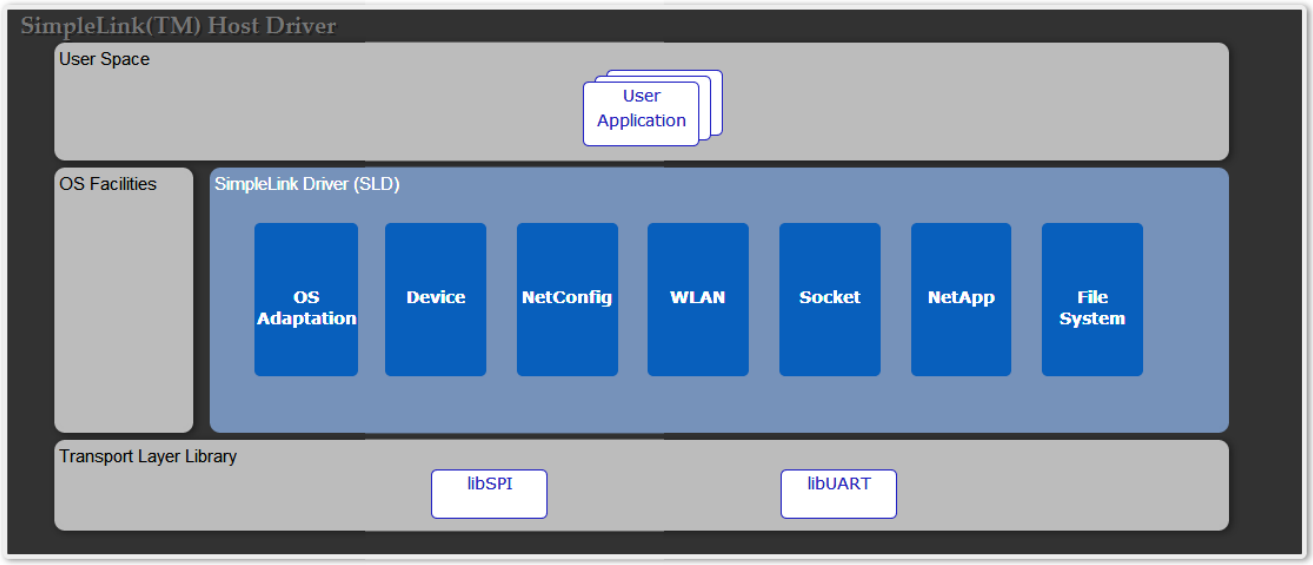


图 17-1. 主机驱动程序 API 孤岛

17.1 器件

器件 API 负责器件的功耗和一般配置。

Sl_Start – 该函数用于初始化通信接口、设置器件的使能引脚以及调用初始化完成回调。如果未提供回调函数，该函数处于阻塞状态，直至器件完成初始化过程。如果成功，器件会返回其职能角色：ROLE_STA、ROLE_AP、ROLE_P2P。否则，在失败时它会返回：ROLE_STA_ERR、ROLE_AP_ERR、ROLE_P2P_ERR

Sl_Stop – 该函数用于清除器件的使能引脚、关闭通信接口并调用停止完成回调（如果存在）。time-out 参数使用户能够控制休眠时序：

- 0 – 立即进入休眠模式
- 0xFFFF – 主机在休眠前等待器件响应，无超时保护。
- 0 < Timeout[msec] < 0xFFFF – 主机在休眠前等待器件响应，已定义超时保护。此超时定义了等待的最长时间。NWP 响应的发送时间可早于此超时。

sl_DevSet – 该函数用于配置不同的器件参数。使用的主参数是 DeviceSetID 和 Option。可能的 DeviceSetID 和 Option 组合为：

- SL_DEVICE_GENERAL_CONFIGURATION – 一般配置选项为：
  - SL_DEVICE_GENERAL_CONFIGURATION_DATE_TIME – 配置器件内部日期和时间。时间参数保留在休眠模式下，但在关机时会复位。

设置器件时间和日期示例：

```
SlDateTime_t dateTime= {0};
dateTime.sl_tm_day =    (unsigned long)23;          // Day of month (DD format) range 1-13
dateTime.sl_tm_mon =    (unsigned long)6;           // Month (MM format) in the range of 1-12
dateTime.sl_tm_year =   (unsigned long)2014;        // Year (YYYY format)
dateTime.sl_tm_hour =   (unsigned long)17;          // Hours in the range of 0-23
dateTime.sl_tm_min =    (unsigned long)55;          // Minutes in the range of 0-59
dateTime.sl_tm_sec =    (unsigned long)22;          // Seconds in the range of  0-59
sl_DevSet(SL_DEVICE_GENERAL_CONFIGURATION,SL_DEVICE_GENERAL_CONFIGURATION_DATE_TIME,sizeof(SlDateTim
e_t),(unsigned char *
) (&dateTime));
```

sl_DevGet – 该函数使用户能够读取不同的器件参数。使用的主参数是 DeviceSetID 和 Option 参数。可能的 DeviceSetID 和 Option 组合为：

- SL_DEVICE_GENERAL_CONFIGURATION
  - SL_DEVICE_GENERAL_CONFIGURATION_DATE_TIME
  - SL_DEVICE_GENERAL_VERSION – 返回器件固件版本
- SL_DEVICE_STATUS – 器件状态选项为：
  - SL_EVENT_CLASS_DEVICE – 可能的值为：
    - EVENT_DROPPED_DEVICE_ASYNC_GENERAL_ERROR – 一般系统错误，检查系统配置。
    - STATUS_DEVICE_SMART_CONFIG_ACTIVE – 器件处于 SmartConfig 模式。
  - SL_EVENT_CLASS_WLAN
    - EVENT_DROPPED_WLAN_WLANASYNCONNECTEDRESPONSE
    - EVENT_DROPPED_WLAN_WLANASYNCDISCONNECTEDRESPONSE
    - EVENT_DROPPED_WLAN_STA_CONNECTED
    - EVENT_DROPPED_WLAN_STA_DISCONNECTED
    - STATUS_WLAN_STA_CONNECTED
- SL_EVENT_CLASS_BSD – 可能的值为：
  - EVENT_DROPPED_SOCKET_TXFAILEDASYNCRESPONSE
- SL_EVENT_CLASS_NETAPP – 可能的值为：
  - EVENT_DROPPED_NETAPP_IPACQUIRED
  - EVENT_DROPPED_NETAPP_IPACQUIRED_V6

  - EVENT_DROPPED_NETAPP_IP_LEASED
  - EVENT_DROPPED_NETAPP_IP_RELEASED
- SL_EVENT_CLASS_NETCFG (当前未使用)
- SL_EVENT_CLASS_NVMEM (当前未使用)

获取版本的示例：

```
SlVersionFull ver;
pConfigOpt = SL_DEVICE_GENERAL_VERSION;
sl_DevGet(SL_DEVICE_GENERAL_CONFIGURATION, &pConfigOpt,&pConfigLen,
          (unsigned char*)(&ver));
          printf("CHIP %d\nMAC 31.%d.%d.%d.%d\nPHY %d.%d.%d.%d\nNWP
          %d.%d.%d.%d\nROM%d\nHOST%d.%d.%d.%d\n",
          ver.ChipFwAndPhyVersion.ChipId,
          ver.ChipFwAndPhyVersion.FwVersion[0],ver.ChipFwAndPhyVersion.FwVersion[1],
          ver.ChipFwAndPhyVersion.FwVersion[2],ver.ChipFwAndPhyVersion.FwVersion[3],
          ver.ChipFwAndPhyVersion.PhyVersion[0],ver.ChipFwAndPhyVersion.PhyVersion[1],
          ver.ChipFwAndPhyVersion.PhyVersion[2],ver.ChipFwAndPhyVersion.PhyVersion[3],
          ver.NwpVersion[0],ver.NwpVersion[1],ver.NwpVersion[2],ver.NwpVersion[3],
          ver.RomVersion, SL_MAJOR_VERSION_NUM, SL_MINOR_VERSION_NUM, SL_VERSION_NUM,
          SL_SUB_VERSION_NUM);
```

sl_EventMaskSet – 屏蔽来自器件的异步事件。屏蔽的事件不会从器件生成异步消息。该函数用于接收 EventClass 和位掩码。事件和掩码选项为：

- SL_EVENT_CLASS_WLAN 用户事件：
  - SL_WLAN_CONNECT_EVENT
  - SL_WLAN_DISCONNECT_EVENT
  - SL_WLAN_STA_CONNECTED_EVENT
  - SL_WLAN_STA_DISCONNECTED_EVENT
  - SL_WLAN_SMART_CONFIG_COMPLETE_EVENT
  - SL_WLAN_SMART_CONFIG_STOP_EVENT
  - SL_WLAN_P2P_DEV_FOUND_EVENT
  - SL_WLAN_P2P_NEG_REQ_RECEIVED_EVENT
  - SL_WLAN_CONNECTION_FAILED_EVENT
- SL_EVENT_CLASS_DEVICE 用户事件：
  - SL_DEVICE_FATAL_ERROR_EVENT
  - SL_DEVICE_ABORT_ERROR_EVENT
- SL_EVENT_CLASS_BSD 用户事件：
  - SL_SOCKET_TX_FAILED_EVENT
  - SL_SOCKET_ASYNC_EVENT
- SL_EVENT_CLASS_NETAPP 用户事件：
  - SL_NETAPP_IPV4_IPACQUIRED_EVENT
  - SL_NETAPP_IPACQUIRED_V6_EVENT
  - SL_NETAPP_IP_LEASED_EVENT
  - SL_NETAPP_IP_RELEASED_EVENT
  - SL_NETAPP_HTTPGETTOKENVALUE_EVENT
  - SL_NETAPP_HTTPPOSTTOKENVALUE_EVENT

从 WLAN 类屏蔽掉连接和断开连接的示例：

```
sl_EventMaskSet(SL_EVENT_CLASS_WLAN, (SL_WLAN_CONNECT_EVENT | SL_WLAN_DISCONNECT_EVENT) );
```

sl_EventMaskGet – 从器件返回事件位掩码。如果已屏蔽该事件，则器件不会将其发送。该函数类似于 *sl_EventMaskSet*。

为 WLAN 类获取事件掩码的示例：

```
sl_EventMaskSet(SL_EVENT_CLASS_WLAN,
(SL_WLAN_CONNECT_EVENT | SL_WLAN_DISCONNECT_EVENT) );
```

sl_EventMaskGet – 从器件返回事件位掩码。如果已屏蔽事件，则器件不会发送事件。该函数类似于 *sl_EventMaskSet*。

为 WLAN 类获取事件掩码的示例：

```
unsigned long maskWlan;
sl_StatusGet(SL_EVENT_CLASS_WLAN,&maskWlan);
```

sl_Task

- **非 OS 平台** – 应从主循环调用
- **多线程平台** – 当用户不采用外部派生函数时，应从分配给 SimpleLink 驱动程序的专用线程调用该函数。在此模式中，该函数从不返回。

sl_UartSetMode – 如果用户所选的主机接口为 UART，应使用此函数。该函数用于设置用户的 UART 配置：

- 波特率
- 流控制
- COM 端口

17.2 NetCfg

sl_NetCfgSet – 管理以下网络功能的配置：

- SL_MAC_ADDRESS_SET – 新 MAC 地址会覆盖默认 MAC 地址并保存在串行闪存文件系统中。
- SL_IPV4_STA_P2P_CL_DHCP_ENABLE – 设置器件在 WLAN STA 模式下或在 P2P 客户端时通过 DHCP 获取 IP 地址。这是系统在连接 WLAN 后获取 IP 地址的默认模式。
- SL_IPV4_STA_P2P_CL_STATIC_ENABLE – 为在 STA 模式下或在 P2P 客户端工作的器件设置静态 IP 地址。IP 地址存储在串行闪存文件系统中。若要禁用静态 IP 并获取从 DHCP 分配的地址，请使用 SL_STA_P2P_CL_IPV4_DHCP_SET。
- SL_SET_HOST_RX_AGGR - 打开/关闭 RX 聚合

  示例：

  ```
  /* 禁用 Rx 聚合 */
  _u8 RxAggrEnable = 0;
  sl_NetCfgSet(SL_SET_HOST_RX_AGGR, 0, sizeof(RxAggrEnable), (_u8 *)&RxAggrEnable);
  ```

- SL_IPV4_AP_P2P_GO_STATIC_ENABLE – 为在 AP 模式下或在 P2P GO 工作的器件设置静态 IP 地址。IP 地址存储在串行闪存文件系统中。

  示例：

  ```
  _NetCfgIpV4Args_t ipV4;
  ipV4.ipV4           = (unsigned long) SL_IPV4_VAL(10,1,1,201);      // unsigned long IP
  address
  ipV4.ipV4Mask       = (unsigned long) SL_IPV4_VAL(255,255,255,0);   // unsigned long //Subnet
  mask for this AP/P2P
  ipV4.ipV4Gateway    = (unsigned long) SL_IPV4_VAL(10,1,1,1);        // unsigned long //
  Default gateway address
  ipV4.ipV4DnsServer  = (unsigned long) SL_IPV4_VAL(8,16,32,64);      // unsigned long DNS
  //server address
  sl_NetCfgSet(SL_IPV4_AP_P2P_GO_STATIC_ENABLE, 1,
  sizeof(_NetCfgIpV4Args_t),(unsigned char *)&ipV4);
  sl_Stop(0);
  sl_Start(NULL,NULL,NULL);
  ```

备注

- AP 模式必须使用静态 IP 设置。
- 所有设置的功能都需要重新启动系统才能使更改生效。

sl_NetCfgGet – 读取网络配置。选项是：

- SL_MAC_ADDRESS_GET
- SL_IPV4_STA_P2P_CL_GET_INFO – 从 WLAN 工作站或 P2P 客户端获取 IP 地址。返回 DHCP 标志，以指示 IP 地址是静态的还是来自 DHCP。
- SL_IPV4_AP_P2P_GO_GET_INFO – 返回 AP 的 IP 地址。

从 WLAN 工作站或 P2P 客户端获取 IP 地址的示例：

```
unsigned char len = sizeof(_NetCfgIpV4Args_t);
unsigned char dhcpIsOn = 0;
_NetCfgIpV4Args_t ipV4 = {0};
sl_NetCfgGet(SL_IPV4_STA_P2P_CL_GET_INFO,&dhcpIsOn,&len,(unsigned char *)&ipV4);
printf("DHCP is %s IP %d.%d.%d.%d MASK %d.%d.%d.%d GW %d.%d.%d.%d DNS %d.%d.%d.%d\n",
(dhcpIsOn > 0) ? "ON":"OFF", SL_IPV4_BYTE(ipV4.ipV4,3), SL_IPV4_BYTE(ipV4.ipV4,2),
SL_IPV4_BYTE(ipV4.ipV4,1), SL_IPV4_BYTE(ipV4.ipV4,0), SL_IPV4_BYTE(ipV4.ipV4Mask,3),
SL_IPV4_BYTE(ipV4.ipV4Mask,2), SL_IPV4_BYTE(ipV4.ipV4Mask,1),
SL_IPV4_BYTE(ipV4.ipV4Mask,0), SL_IPV4_BYTE(ipV4.ipV4Gateway,3),
SL_IPV4_BYTE(ipV4.ipV4Gateway,2), SL_IPV4_BYTE(ipV4.ipV4Gateway,1),
SL_IPV4_BYTE(ipV4.ipV4Gateway,0), SL_IPV4_BYTE(ipV4.ipV4DnsServer,3),
SL_IPV4_BYTE(ipV4.ipV4DnsServer,2), SL_IPV4_BYTE(ipV4.ipV4DnsServer,1),
SL_IPV4_BYTE(ipV4.ipV4DnsServer,0));
```

17.3 WLAN

sl_WlanSetMode – WLAN 器件具有多种 WLAN 运行模式。默认情况下，该器件充当 WLAN 工作站，但也可以充当其他 WLAN 角色。不同的选项包括：

- ROLE_STA – 针对 WLAN 工作站模式
- ROLE_AP – 针对 WLAN AP 模式
- ROLE_P2P – 针对 WLAN P2P 模式

备注

设置的模式功能仅在下一次器件启动时生效。

从任意角色切换到 WLAN AP 角色的示例：

```
sl_WlanSetMode(ROLE_AP);
/*关闭和打开器件以使角色更改生效 */
sl_Stop(0);
sl_Start(NULL,NULL,NULL);
```

sl_WlanSet – 让用户配置不同的 WLAN 相关参数。使用的主参数是 ConfigID 和 ConfigOpt。

可能的 ConfigID 和 ConfigOpt 组合为：

- SL_WLAN_CFG_GENERAL_PARAM_ID – 不同的通用 WLAN 参数如下：
  - WLAN_GENERAL_PARAM_OPT_COUNTRY_CODE
  - WLAN_GENERAL_PARAM_OPT_STA_TX_POWER – 设置 STA 模式 Tx 功率级别（一个 0 至 15 的数字），作为最大功率的 dB 偏移（0 将设置最大功率）。
  - WLAN_GENERAL_PARAM_OPT_AP_TX_POWER – 设置 AP 模式 Tx 功率级别（一个 0 至 15 的数字），作为最大功率的 dB 偏移（0 将设置最大功率）。
- SL_WLAN_CFG_AP_ID – 不同的 AP 配置选项如下：
  - WLAN_AP_OPT_SSID
  - WLAN_AP_OPT_CHANNEL

  - WLAN_AP_OPT_HIDDEN_SSID – 将 AP 设置为隐藏或非隐藏
  - WLAN_AP_OPT_SECURITY_TYPE – 可能的选项为：
    - 开放安全：SL_SEC_TYPE_OPEN
    - WEP 安全：SL_SEC_TYPE_WEP
    - WPA 安全：SL_SEC_TYPE_WPA
  - WLAN_AP_OPT_PASSWORD – 设置 AP 模式的安全密码：
    - 对于 WPA：8 至 63 个字符
    - 对于 WEP：5 至 13 个字符 (ASCII)
- SL_WLAN_CFG_P2P_PARAM_ID
  - WLAN_P2P_OPT_DEV_NAME
  - WLAN_P2P_OPT_DEV_TYPE
  - WLAN_P2P_OPT_CHANNEL_N_REGS – 侦听通道和监管等级确定了 P2P 查找和侦听阶段的器件侦听通道。运行通道和监管等级确定了器件更适合的运行通道（如果器件是组所有者，则会使用该运行通道）。通道应该是社会通道之一（1、6 或 11）。如果未选择侦听通道或运行通道，则将随机选择 1、6 或 11。
  - WLAN_GENERAL_PARAM_OPT_INFO_ELEMENT – 应用程序为每个角色 (AP/P2P GO) 设置 MAX_PRIVATE_INFO_ELEMENTS_SUPPORTED 个信息元素。若要删除信息元素，请使用相关索引和长度 = 0。应用程序可以为同一角色设置不超过 MAX_PRIVATE_INFO_ELEMENTS_SUPPORTED 的值。但是，对于 AP，可以为所有信息元素存储不超过 INFO_ELEMENT_MAX_TOTAL_LENGTH_AP 字节。对于 P2P GO，可以为所有信息元素存储不超过 INFO_ELEMENT_MAX_TOTAL_LENGTH_P2P_GO 字节。
  - WLAN_GENERAL_PARAM_OPT_SCAN_PARAMS – 更改扫描通道和 RSSI 阈值

为 AP 模式设置 SSID 的示例：

```
unsigned char  str[33];
memset(str, 0, 33);
memcpy(str, ssid, len);  // ssid string of 32 characters
sl_WlanSet(SL_WLAN_CFG_AP_ID, WLAN_AP_OPT_SSID, strlen(ssid), str);
```

sl_WlanGet – 让用户能够配置不同的 WLAN 相关参数。使用的主参数是 ConfigID 和 ConfigOpt。*sl_WlanGet* 的用法与 *sl_WlanSet* 类似。

sl_WlanConnect – 手动连接到 WLAN 网络

sl_WlanDisconnect – 断开 WLAN 连接

sl_WlanProfileAdd – 启用自动启动连接策略后，器件会根据配置文件表连接到 AP。最多支持七个配置文件。如果配置了多个配置文件，器件会选择优先级最高的配置文件。在每个优先级组中，器件根据以下参数按优先级降序选择配置文件：安全策略、信号强度。

sl_WlanProfileGet – 从器件读取 WLAN 配置文件

sl_WlanProfileDel – 删除现有配置文件

sl_WlanPolicySet – 管理以下 WLAN 功能的配置：

- SL_POLICY_CONNECTION – SL_POLICY_CONNECTION 类型可定义用于将 CC31xx 器件连接至 AP 的三个选项：
  - 自动连接 – 每次连接失败或者器件重新进行引导后，CC31xx 器件都会尝试自动重新连接到其存储的配置文件之一。若要设置此选项，请使用：
    - `sl_WlanPolicySet(SL_POLICY_CONNECTION,SL_CONNECTION_POLICY(1,0,0,0,0),NULL,0)`
  - 快速连接 – CC31xx 器件尝试快速连接到 AP。若要设置此选项，请使用：
    - `sl_WlanPolicySet(SL_POLICY_CONNECTION,SL_CONNECTION_POLICY(0,1,0,0,0),NULL,0)`
  - P2P 连接 – 如果设置了“任何 P2P”模式，CC31xx 器件会尝试自动连接至可用的第一个 P2P 器件，仅支持使用按钮。若要设置此选项，请使用：

- `sl_WlanPolicySet(SL_POLICY_CONNECTION,SL_CONNECTION_POLICY(0,0,0,1,0),NULL,0)`

– 重新启动后自动 SmartConfig – 器件在 SmartConfig 模式下唤醒。主机发出的任何命令都会使此状态结束。若要设置此选项，请使用：

- `sl_WlanPolicySet(SL_POLICY_CONNECTION,SL_CONNECTION_POLICY(0,0,0,0,1),NULL,0)`

- SL_POLICY_SCAN – 定义没有连接时的系统扫描时间间隔。默认间隔为 10 分钟。设置扫描间隔后，系统会立即激活扫描。下一次扫描基于间隔设置。若要将扫描间隔设置为 1 分钟，请使用以下示例：

```
unsigned long intervalInSeconds = 60;
#define SL_SCAN_ENABLE  1
sl_WlanPolicySet(SL_POLICY_SCAN,SL_SCAN_ENABLE, (unsigned char *)
&intervalInSeconds,sizeof(intervalInSeconds));
```

若要禁用扫描，请使用：

```
#define SL_SCAN_DISABLE  0
sl_WlanPolicySet(SL_POLICY_SCAN,SL_SCAN_DISABLE,0,0);
```

- SL_POLICY_PM – 定义仅用于工作站模式的电源管理策略。共有四个可用的电源策略：
  - SL_NORMAL_POLICY（默认）– 若要设置正常电源管理策略，请使用：
    - `sl_WlanPolicySet(SL_POLICY_PM , SL_NORMAL_POLICY, NULL,0)`
  - SL_ALWAYS_ON_POLICY – 若要设置始终开启电源管理策略，请使用：
    - `sl_WlanPolicySet(SL_POLICY_PM , SL_ALWAYS_ON_POLICY, NULL,0)`
  - SL_LONG_SLEEP_INTERVAL_POLICY – 若要设置长时间睡眠间隔策略，请使用：
    - ```
      unsigned short PolicyBuff[4] = {0,0,800,0}; // 800 is max sleep time in mSec
      sl_WlanPolicySet(SL_POLICY_PM , SL_LONG_SLEEP_INTERVAL_POLICY,
      PolicyBuff,sizeof(PolicyBuff));
      ```
- SL_POLICY_P2P – 为 P2P 角色定义 P2P 协商策略参数。若要设置意图协商值，请设置以下之一：
  - SL_P2P_ROLE_NEGOTIATE – 意图 3
  - SL_P2P_ROLE_GROUP_OWNER – 意图 15
  - SL_P2P_ROLE_CLIENT – 意图 0
- 若要设置协商发起方值（第一个协商操作帧的发起方策略），请设置以下之一：
  - SL_P2P_NEG_INITIATOR_ACTIVE
  - SL_P2P_NEG_INITIATOR_PASSIVE
  - SL_P2P_NEG_INITIATOR_RAND_BACKOFF

例如：

```
set sl_WlanPolicySet(SL_POLICY_P2P,
SL_P2P_POLICY(SL_P2P_ROLE_NEGOTIATE,SL_P2P_NEG_INITIATOR_RAND_BACKOFF),NULL,0);
```

sl_WlanPolicyGet – 读取不同的 WLAN 策略设置。可能的选项包括：

- SL_POLICY_CONNECTION
- SL_POLICY_SCAN
- SL_POLICY_PM
- SL_POLICY_P2P

sl_WlanGetNetworkList – 获取最新的 WLAN 扫描结果

sl_WlanSmartConfigStart – 将器件置于 SmartConfig 状态。SmartConfig 成功结束后，将收到一个异步事件：SL_OPCODE_WLAN_SMART_CONFIG_START_ASYNC_RESPONSE。该事件包含 SSID 以及一个可能也已完成传递的额外字段（例如，器件名称）。
```

sl_WlanSmartConfigStop – 停止 SmartConfig 过程。停止 SmartConfig 后，将收到一个异步事件：SL_OPCODE_WLAN_SMART_CONFIG_STOP_ASYNC_RESPONSE

sl_WlanRxStatStart – 开始收集 WLAN Rx 统计信息（不限时间）

sl_WlanRxStatStop – 停止收集 WLAN Rx 统计信息

sl_WlanRxStatGet – 获取 WLAN Rx 统计信息。调用此命令后，统计信息计数器将被清除。返回的统计信息如下：

- 接收到的有效数据包数 – 正确接收的数据包总数（包括已过滤的数据包）
- 接收到的 FCS 错误数据包 – 由于 FCS 错误而丢弃的数据包总数
- 接收到的 PLCP 错误数据包 – 由于 PLCP 错误而丢弃的数据包总数
- 平均数据 RSSI – 接收到的所有有效数据包的平均 RSSI
- 平均管理 RSSI – 接收到的所有有效管理数据包的平均 RSSI
- 速率直方图 – 接收到的所有有效数据包的速率直方图
- RSSI 直方图 – 从 -40 到 -87 的 RSSI 直方图（低于和高于 RSSI 的所有值都出现在第一个和最后一个单元中）
- 开始时间戳 – 开始收集统计信息的时间戳（以 µSec 为单位）
- 获取时间戳 – 读取统计信息的时间戳（以 µSec 为单位）

17.4 套接字

sl_Socket – 创建某个类型（由整数标识）的新套接字，并为该套接字分配系统资源。支持以下套接字类型：

- SOCK_STREAM（TCP – 面向流的可靠服务或流套接字）
- SOCK_DGRAM（UDP – 数据报服务或数据报套接字）
- SOCK_RAW（网络层之上的原始协议）

sl_Close – 正常关闭套接字。此函数使系统释放分配给套接字的资源。对于 TCP，连接将终止。

sl_Accept – 此函数可与基于连接的套接字类型 (SOCK_STREAM) 搭配使用，用于提取挂起连接队列上的第一个连接请求，创建一个新的连接套接字，并返回一个引用该套接字的新文件描述符。新创建的套接字不处于侦听状态。原始套接字 sd 不受此调用的影响。

sl_Bind – 为套接字指定本地地址 *addr*。*addr* 的长度为 *addrlen* 字节。传统上，在创建套接字时调用此函数，并且此函数存在于命名空间（地址族）中，但未向其分配名称。在 SOCK_STREAM 套接字接收连接之前，必须分配一个本地地址。

sl_Listen – 指定接受传入连接的意愿和传入连接的队列限制。*listen()* 调用仅适用于 SOCK_STREAM 类型的套接字，而 backlog 参数定义了挂起连接队列的最大长度。

sl_Connect – 将套接字描述符 *sd* 引用的套接字连接到 *addr* 指定的地址。*addrlen* 参数指定了 *addr* 的大小。*addr* 中地址的格式由套接字的地址空间决定。如果套接字是 SOCK_DGRAM 类型，则此调用指定与套接字相关联的对等方。数据报应发送至该地址，这是接收数据报的唯一地址。如果套接字是 SOCK_STREAM 类型，则此调用尝试与另一个套接字建立连接。另一个套接字由套接字通信空间中的地址指定。

sl_Select – 允许程序监视多个文件描述符，等待一个或多个文件描述符为某类 I/O 操作做好准备。*sl_Select* 包含可设置文件描述符选项的几个子函数：

- SL_FD_SET – 选择 *SlFdSet_t* SET 函数。在 *SlFdSet_t* 容器上设置当前套接字描述符。
- SL_FD_CLR – 选择 *SlFdSet_t* CLR 函数。在 *SlFdSet_t* 容器上清除当前套接字描述符。
- SL_FD_ISSET – 选择 *SlFdSet_t* ISSET 函数。检查是否设置了当前套接字描述符 (TRUE/FALSE)。
- SL_FD_ZERO – 选择 *SlFdSet_t* ZERO 函数。从 *SlFdSet_t* 容器中清除所有套接字描述符。

sl_SetSockOpt – 操控与套接字相关联的选项。选项涉及多个协议级别，并且始终涉及最高套接字级别。支持以下套接字选项：

- SL_SOL_SOCKET – 套接字选项类别：

  - SL_SO_KEEPALIVE – 通过启用消息的定期传输，让 TCP 连接始终处于活动状态；启用或禁用，定期处于活动状态。*默认值：启用；保持活动超时为 300 秒。*
  - SL_SO_RCVTIMEO – 设置超时值，该值指定输入函数等待完成的最长时间。*默认值：无超时*
  - SL_SO_RCVBUF – 设置 TCP 最大接收窗口期
  - SL_SO_NONBLOCKING – 将套接字设置为非阻塞操作。影响：*sl_Connect*、*sl_Accept*、*sl_Send*、*sl_Sendto*、*sl_Recv* 和 *sl_Recvfrom*。默认值：阻塞。
  - SL_SO_SECMETHOD – 将方法设置为 TCP 安全套接字 (SL_SEC_SOCKET)。默认值：SL_SO_SEC_METHOD_SSLv3_TLSV1_2。
  - SL_SO_SECURE_MASK – 将特定密码设置为 TCP 安全套接字 (SL_SEC_SOCKET)。默认值：适合方法的“理想”密码
  - SL_SO_SECURE_FILES – 将编程的文件映射到安全套接字 (SL_SEC_SOCKET)
  - SL_SO_SECURE_FILES_PRIVATE_KEY_FILE_NAME – 此选项用于配置安全
  - SL_SO_SECURE_FILES_CERTIFICATE_FILE_NAME – 此选项用于配置安全
  - SL_SO_SECURE_FILES_CA_FILE_NAME – 此选项用于配置安全文件
  - SL_SO_SECURE_FILES_DH_KEY_FILE_ NAME – 此选项用于配置安全
  - SL_SO_CHANGE_CHANNEL – 将通道设置为收发器模式
- SL_IPPROTO_IP – IP 选项类别：
  - SL_IP_MULTICAST_IF – 指定传出多播接口。
  - SL_IP_MULTICAST_TTL – 为套接字设置传出多播数据包的存活时间值
  - SL_IP_RAW_RX_NO_HEADER – 原始套接字；从接收的数据中删除 IP 标头。*默认值：数据中包括 IP 标头。*
  - SL_IP_HDRINCL – 仅限 RAW 套接字；除非在套接字上启用 IP_HDRINCL 套接字选项，否则 IPv4 层在发送数据包时会生成 IP 标头。启用套接字选项后，数据包必须包含 IP 标头。*默认值：禁用，网络堆栈生成的 IPv4 标头。*
  - SL_IP_ADD_MEMBERSHIP – UDP 套接字；加入多播组。
  - SL_IP_DROP_MEMBERSHIP – UDP 套接字；离开多播组。
- SL_SOL_PHY_OPT – PHY 选项类别：
  - SL_SO_PHY_RATE – RAW 套接字；设置 WLAN PHY 传输速率。
  - SL_SO_PHY_TX_POWER – RAW 套接字；设置 WLAN PHY Tx 功率。
  - SL_SO_PHY_NUM_FRAMES_TO_TX – RAW 套接字；设置在收发器模式下传输的帧数。
  - SL_SO_PHY_PREAMBLE – RAW 套接字；设置 WLAN PHY 前导码。

sl_GetSockOpt – 操控与套接字相关联的选项。选项可能涉及多个协议级别，并且始终涉及最高套接字级别。这些套接字选项与 *sl_SetSockOpt* 中的相同。

sl_Recv – 从 TCP 套接字读取数据

sl_RecvFrom – 从 UDP 套接字读取数据

sl_Send – 将数据写入 TCP 套接字。向器件发送数据后立即返回。如果 TCP 失败，则会收到异步事件 SL_NETAPP_SOCKET_TX_FAILED。对于 RAW 套接字（收发器模式），应在帧数据缓冲区的末尾为 WLAN FCS 保留额外的 4 个字节。

sl_SendTo – 将数据写入 UDP 套接字。此函数将消息传输到另一个套接字（无连接套接字 SOCK_DGRAM、SOCK_RAW）。向器件发送数据后立即返回。如果传输失败，则会收到异步事件 SL_NETAPP_SOCKET_TX_FAILED。

sl_Htonl – 将 32 位无符号值的字节顺序从处理器顺序更改为网络顺序。

sl_Htons – 将 16 位无符号值的字节顺序从处理器顺序更改为网络顺序。

17.5 NetApp

sl_NetAppStart – 启用或启动不同的网络服务。可以是以下一项，也可以是多项的组合：

- SL_NET_APP_HTTP_SERVER_ID – HTTP 服务器服务

- SL_NET_APP_DHCP_SERVER_ID – DHCP 服务器服务（始终支持 DHCP 客户端）
- SL_NET_APP_MDNS_ID – MDNS 客户端/服务器服务

sl_NetAppStop – 禁用或停止网络服务。与 *sl_NetAppStart* 中的选项类似。

sl_NetAppSet 设置各种网络应用参数

- SL_NET_APP_DHCP_SERVER_ID - DHCP 服务器标识符编号
  - NETAPP_SET_DHCP_SRV_BASIC_OPT - DHCP 服务器基本选项配置标识符（例如 IP 范围、租用时间）
- SL_NET_APP_HTTP_SERVER_ID - HTTP 服务器标识符编号
  - NETAPP_SET_GET_HTTP_OPT_PORT_NUMBER - HTTP 服务器端口号
  - NETAPP_SET_GET_HTTP_OPT_AUTH_CHECK - HTTP 验证校验状态
  - NETAPP_SET_GET_HTTP_OPT_AUTH_NAME - HTTP 验证名称（默认：“admin”）
  - NETAPP_SET_GET_HTTP_OPT_AUTH_PASSWORD - HTTP 验证密码（默认：“admin”）
  - NETAPP_SET_GET_HTTP_OPT_AUTH_REALM - HTTP 验证领域获取（默认：“SimpleLink CC31xx”）
  - NETAPP_SET_GET_HTTP_OPT_ROM_PAGES_ACCESS - HTTP 获取状态，如果允许默认页面（“ROM”）
- SL_NET_APP_MDNS_ID
  - NETAPP_SET_GET_MDNS_CONT_QUERY_OPT - MDNS 设置连续查询
  - NETAPP_SET_GET_MDNS_QEVETN_MASK_OPT - 要忽略的 MDNS 掩码服务类型
  - NETAPP_SET_GET_MDNS_TIMING_PARAMS_OPT - MDNS 重新配置计时参数，如因素、间隔
- SL_NET_APP_DEVICE_CONFIG_ID
  - NETAPP_SET_GET_DEV_CONF_OPT_DEVICE_URN - 设置/获取器件 URN 名称
  - NETAPP_SET_GET_DEV_CONF_OPT_DOMAIN_NAME - 设置/获取器件域名

示例：

```
unsigned char str[32] = "domainname.net";
unsigned char len = strlen((const char *)str);
retVal = sl_NetAppSet(SL_NET_APP_DEVICE_CONFIG_ID, NETAPP_SET_GET_DEV_CONF_OPT_DOMAIN_NAME,
        len, (unsigned char*)str);
```

sl_NetAppGet – 检索各种网络应用参数。

sl_NetAppDnsGetHostByName – 根据机器名称获取网络上某台机器的 IP 地址。

示例：

```
unsigned long DestinationIP;
sl_NetAppDnsGetHostByName("www.ti.com", strlen("www.ti.com"), &DestinationIP, SL_AF_INET);
Addr.sin_family = SL_AF_INET; Addr.sin_port = sl_Htons(80);
Addr.sin_addr.s_addr = sl_Htonl(DestinationIP);
AddrSize = sizeof(SlSockAddrIn_t);
SockID = sl_Socket(SL_AF_INET,SL_SOCK_STREAM, 0);
```

sl_NetAppDnsGetHostByService – 根据服务名称返回服务属性，如 IP 地址、端口和文本。用户设置完整或部分服务名称（请参阅以下示例），并且应获得：

- 服务的 IP
- 服务的端口
- 服务的文本

这类似于 GET host by name（按名称获取主机）方法，实际是对服务名称进行 PTR 类型的一次性查询。完整服务名称的示例如下：

- PC1._ipp._tcp.local
- PC2_server._ftp._tcp.local

部分服务名称的示例如下：

- _ipp._tcp.local
- _ftp._tcp.local

sl_NetAppGetServiceList – 获取对等服务的列表。该列表采用服务结构的形式。用户选择服务结构的类型。支持的结构：

- 带文本的完整服务参数
- 完整服务参数
- 短服务参数（仅限端口和 IP），尤其对于小型主机

备注

不同类型的结构可以节省主机中的内存。

sl_NetAppMDNSRegisterService – 向 mDNS 包和数据库注册新的 mDNS 服务。该注册服务由应用提供。根据 DNS-SD RFC，服务名称应该是完整服务名称，即 SRV 应答中名称字段中的值。

服务名称示例：

- PC1._ipp._tcp.local
- PC2_server._ftp._tcp.local

如果设置了 is_unique 选项，mDNS 会探测服务名称，以确保它是唯一的，然后在网络上公布该服务。

sl_NetAppMDNSUnRegisterService – 从 mDNS 包和数据库中删除 mDNS 服务。

sl_NetAppPingStart – 向网络主机发送 ICMP ECHO_REQUEST（或 Ping）。以下是发送 20 个 Ping 请求并在发送所有请求后将结果报告给回调例程的示例：

```
// callback routine
void pingRes(SlPingReport_t* pReport)
{
             // handle ping results
}
// ping activation
void PingTest()
{
                 SlPingReport_t report;
                 SlPingStartCommand_t pingCommand;

                 pingCommand.Ip = SL_IPV4_VAL(10,1,1,200);     // destination IP address is
10.1.1.200
                 pingCommand.PingSize = 150;                   // size of ping, in bytes
                 pingCommand.PingIntervalTime = 100;           // delay between pings, in
milliseconds
                 pingCommand.PingRequestTimeout = 1000;        // timeout for every ping in
milliseconds
                 pingCommand.TotalNumberOfAttempts = 20;       // max number of ping requests.0 -
forever
                 pingCommand.Flags = 0;                        // report only when finished

                 sl_NetAppPingStart( &pingCommand, SL_AF_INET, &report, pingRes ) ;
}
```

17.6 文件系统

sl_FsOpen – 打开一个文件，从串行闪存存储器 AccessModeAndMaxSize 读取或向其中写入。

可能的输入包括：

- FS_MODE_OPEN_READ – 读取一个文件
- FS_MODE_OPEN_WRITE – 打开一个现有文件进行写入

- FS_MODE_OPEN_CREATE(maxSizeInBytes,accessModeFlags) – 打开以创建一个新文件。最大文件大小以字节为单位。若要获得理想的文件系统大小，请使用 4K 到 512 字节的最大大小（例如，3584,7680）。*SlFileOpenFlags_e* 可以将几种访问模式组合在一起。

  示例：

```
sl_FsOpen("FileName.html",FS_MODE_OPEN_CREATE(3584,_FS_FILE_OPEN_FLAG_COMMIT|
_FS_FILE_PUBLIC_WRITE) ,NULL, &FileHandle);
```

- *sl_FsRead* – 从一个文件中读取数据块
- *sl_FsWrite* – 将数据块写入一个文件
- *sl_FsDel* – 从串行闪存存储器中删除一个特定文件或所有文件（格式）
- *sl_FsGetInfo* – 返回文件信息：标志、文件大小、分配的大小和令牌

异步事件

17.1 概述

SimpleLink 主机驱动程序通过 SPI 或 UART 总线接口向 SimpleLink 器件传输命令，从而与该器件进行交互。某些命令可能会触发需要数百毫秒甚至数秒时间的耗时进程，因此器件和主机驱动程序支持从器件向主机驱动程序发送异步事件的机制。

事件在进程完成时发出通知（例如 SmartConfig 进程完成时），在器件状态更改时发出通知（例如 WLAN 断开事件），或在出错时发出通知（例如致命错误事件）。

这些事件可以分为以下逻辑类别：

- WLAN 事件
- 网络应用事件
- 套接字事件
- 一般器件事件

17.2 WLAN 事件

SL_WLAN_CONNECT_EVENT – 通知该器件已连接到 AP。

事件参数：

- connection_type
- ssid_len
- ssid_name
- go_peer_device_name_len – 仅与 P2P 相关
- go_peer_device_name
- bssid

SL_WLAN_DISCONNECT_EVENT – 通知该器件已从 AP 断开连接。

事件参数：

- connection_type
- ssid_len
- ssid_name
- go_peer_device_name_len – 仅与 P2P 相关
- go_peer_device_name
- bssid
- reason_code – WLAN 断连原因

SL_WLAN_SMART_CONFIG_START_EVENT – 向主机通知 SmartConfig 已结束。事件参数：

- 状态
- ssid_len
- ssid
- private_token_len
- private_token

SL_WLAN_SMART_CONFIG_STOP_EVENT – 向主机通知 SmartConfig 已停止。

事件参数：

- 状态[status]

SL_WLAN_STA_CONNECTED_EVENT – 通知已连接 STA；在 AP 模式或 P2P GO 模式下相关。

事件参数：

- 状态[status]
- go_peer_device_name

- mac
- go_peer_device_name_len
- wps_dev_password_id
- own_ssid
- own_ssid_len

SL_WLAN_STA_DISCONNECTED_EVENT – 通知 STA 已断开连接；在 AP 模式或 P2P GO 模式下相关。

事件参数：

- 状态
- go_peer_device_name
- mac
- go_peer_device_name_len
- wps_dev_password_id
- own_ssid
- own_ssid_len

SL_WLAN_P2P_DEV_FOUND_EVENT – 通知已找到器件；在 P2P 模式下相关。

事件参数：

- go_peer_device_name
- mac
- go_peer_device_name_len
- wps_dev_password_id
- own_ssid
- own_ssid_len

SL_WLAN_P2P_NEG_REQ_RECEIVED_EVENT – 通知协商请求收到了某个事件；在 P2P 模式下相关。

事件参数：

- go_peer_device_name
- mac
- go_peer_device_name_len
- wps_dev_password_id
- own_ssid
- own_ssid_len

SL_WLAN_CONNECTION_FAILED_EVENT – 通知协商失败；在 P2P 模式下相关。

事件参数：

- 状态[status]

17.3 Netapp 事件

SL_NETAPP_IPV4_IPACQUIRED_EVENT – 通知所询问的 IPv4。

事件参数：

- IP
- gateway
- dns

SL_NETAPP_IP_LEASED_EVENT – 通知 STA IP 租用；在 AP 或 P2P GO 模式下相关。

事件参数：

- ip_address

- lease_time
- mac

SL_NETAPP_IP_RELEASED_EVENT – 通知 STA IP 释放；在 AP 或 P2P GO 模式下相关。

事件参数：

- ip_address
- mac
- reason

SL_NETAPP_HTTPGETTOKENVALUE_EVENT – 通知令牌丢失。尝试从主机检索该值。

事件参数：

- httpTokenName
- httpTokenName length

SL_NETAPP_HTTPPOSTTOKENVALUE_EVENT – 通知一个带有所含参数的新 POST。

事件参数：

- action
- action length
- token_name
- token_name length
- token_value
- token_value length

17.4 套接字事件

SL_SOCKET_TX_FAILED_EVENT – 通知 TX 失败。

事件参数：

- 状态
- sd

SL_SOCKET_ASYNC_EVENT – 通知异步事件。

事件参数：

- sd
- type – 事件类型可以是以下其中之一：
  - SSL_ACCEPT – SSL 问题导致未能成功接受（TCP 通过）
  - RX_FRAGMENTATION_TOO_BIG – 无连接模式，RX 数据包分段 > 16K，数据包被释放。
  - OTHER_SIDE_CLOSE_SSL_DATA_NOT_ENCRYPTED – 远端关闭，从安全变为不安全
- 值

17.5 器件事件

- SL_DEVICE_FATAL_ERROR_EVENT – 通知发生了致命错误；必须执行器件复位。事件参数：
  - 状态：来自器件的错误代码指示
  - 发送方：基于 SlErrorSender_e 枚举的发送方发起人
- SL_DEVICE_ABORT_ERROR_EVENT - 表示发生了一个严重错误，器件已停止：
  - AbortType：指示事件类型
  - AbortData：有关数据错误的其他信息

18.1 通用

.本章描述了信息元素的要求、设计和实现。主机应能够以 AP/P2P GO 角色在信标/探针响应中添加/删除信息元素，并且还允许从连接的 AP/P2P GO 或对等 STA/P2P 客户端（专有）获取已发布的 IE。此功能的目的是允许使用 802.11 标准功能发布特定于供应商的信息。

图 18-1 显示了取自 802.11 规范的信息元素声明。

8.4.2.28 Vendor Specific element

The Vendor Specific element is used to carry information not defined in this standard within a single defined format, so that reserved element IDs are not usurped for nonstandard purposes and so that interoperability is more easily achieved in the presence of nonstandard information. The element is in the format shown in Figure 8-189 and requires that the first 3 or more octets of the Information field identify the entity that has defined the content of the particular Vendor Specific element. The length of the Organization Identifier field (see 8.4.1.31) is *j* octets, and the order of this field is described in 8.2.2. The length of the Information field (*n*) is constrained by $j \le n \le 255$. The length of the vendor-specific content is *n*–*j* octets.

	Element ID	Length	Organization Identifier (see 8.4.1.31)	Vendor-specific content
Octets:	1	1	*j*	*n*–*j*

Figure 8-189—Vendor Specific element format

Multiple Vendor Specific elements are optionally present in a single frame. Each Vendor Specific element might have a different Organization Identifier value. The number of Vendor Specific elements that may appear in a frame is limited only by the maximum frame size.

图 18-1. 802.11 规范 - 信息元素

信息元素可以有相同的 ID 和 OUI，只是有效载荷不同。

图 18-2 表明存在四个具有相同 ID (221) 和相同 OUI (00:40:96) 的信息元素

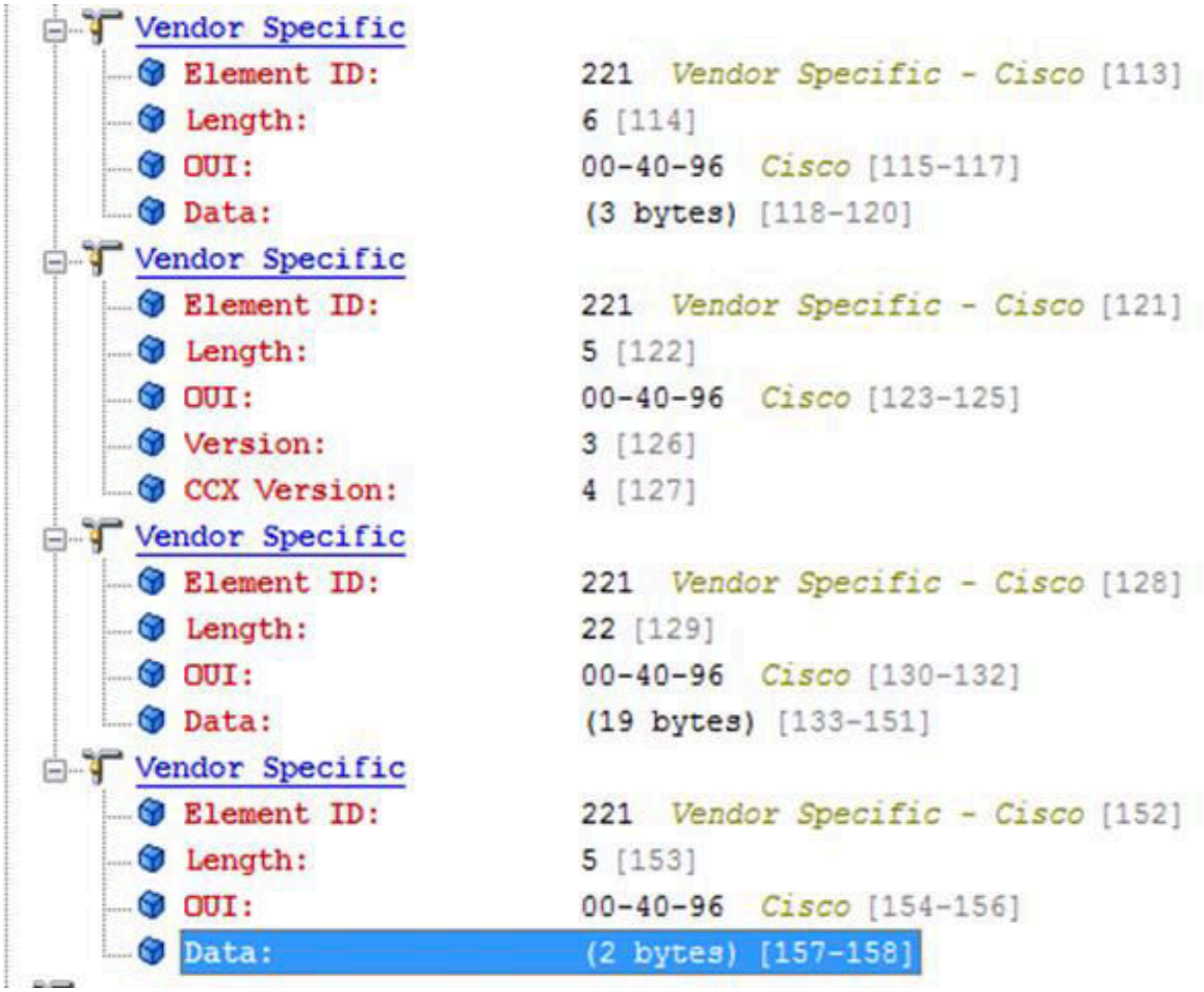


图 18-2. 具有相同 ID 和 OUI 的信息元素

18.2 应用接口

主机具有一个 API，使用 *sl_WlanSet()*，允许添加和删除信息元素。执行命令后，将向主机发送基本响应事件。

示例：

```
sl_protocol_WlanSetInfoElement_t    infoele;
        infoele.index     = Index;                  // Index of the info element. range: 0 -
MAX_PRIVATE_INFO_ELEMENTS_SUPPROTED
        infoele.role      = Role;                   // INFO_ELEMENT_AP_ROLE (0) or
INFO_ELEMENT_P2P_GO_ROLE (1)
        infoele.ie.id     =  Id;                    // Info element ID. if INFO_ELEMENT_DEFAULT_ID
(0) is set, ID will be set to 221.
        // Organization unique ID.If all 3 bytes are zero - it will be replaced with 08,00,28.
        infoele.ie.oui[0] =  Oui0;                  // Organization unique ID first Byte
        infoele.ie.oui[1] =  Oui1;                  // Organization unique ID second Byte
        infoele.ie.oui[2] =  Oui2;                  // Organization unique ID third Byte
        infoele.ie.length = Len;                    // Length of the info element. must be smaller
than 253 bytes
        memset(infoele.ie.data, 0, INFO_ELEMENT_MAX_SIZE);
        if ( Len <= INFO_ELEMENT_MAX_SIZE )
        {
            memcpy(infoele.ie.data, IE, Len);           // Info element. length of the info element
is [0-252]

sl_WlanSet(SL_WLAN_CFG_GENERAL_PARAM_ID,WLAN_GENERAL_PARAM_OPT_INFO_ELEMENT,sizeof(sl_protocol_WlanS
etInfoElement_t),(_u8* ) &infoele);
```

```
        }

sl_WlanSet(SL_WLAN_CFG_GENERAL_PARAM_ID,WLAN_GENERAL_PARAM_OPT_INFO_ELEMENT,sizeof(sl_protocol_WlanS
etInfoElement_t),(_u8* ) &infoele);
```

表 18-1. API 输入

参数	字节数	说明
Index	1	条目的索引
Role	1	AP – 0，P2P GO = 1
Index	1	IE 的索引
Role	1	0 – AP，1 – P2P GO
ID	1	IE 编号（null = 特定于供应商 [221]）
OUI	3	特定供应商 IE 中的组织 ID (null = MAC_ADDR_OUI)
IE Length	2	IE 有效载荷长度。0 表示删除此 IE。值在 [0-252] 范围内。所有配置的信息元素的总长度不应大于 (INFO_ELEMENT_MAX_TOTAL_LENGTH)
IE	IE 长度 (0-252)	信息元素的有效载荷

- 当用户想要添加/删除信息元素时，它将指定待添加或删除的条目的索引。
- 如果用户将 ID 设置为 0，该值将在 NWP 中替换为 221。
- 如果用户将 OUI 设置为 00:00:00，该值将在 NWP 中替换为 08:00:28（德州仪器 (TI) 的 OUI）。

该命令使用以下结构。

```
typedef struct {
    UINT8                       index;
    UINT8                       role;   //bit0: AP = 0, GO = 1
    sl_protocol_InfoElement_t   ie;
} sl_protocol_WlanSetInfoElement_t;
```

其中 *sl_protocol_InfoElement_t* 定义如下：

```
typedef struct {
    UINT8   id;
    UINT8   oui[3];
    UINT16  length;
    UINT8   data[252];
} sl_protocol_InfoElement_t;
```

主机最多可以配置 4 个 IE。

```
#define MAX_PRIVATE_INFO_ELEMENTS_SUPPROTED     4
```

信息元素大小上限为 252 字节

```
#define INFO_ELEMENT_MAX_SIZE                   252
```

AP 模式下所有信息元素的总长度为 300 字节（例如 - 4 个信息元素，每个元素 75 字节）

```
#define INFO_ELEMENT_MAX_TOTAL_LENGTH_AP        300
```

P2P GO 模式下所有信息元素的总长度为 160 字节

```
#define INFO_ELEMENT_MAX_TOTAL_LENGTH_AP        160
```

备注

此限制包括 ID (1 字节) + 长度 (1) + OUI (3)。

Role 定义如下：

```
#define INFO_ELEMENT_AP_ROLE        0
#define INFO_ELEMENT_P2P_GO_ROLE    1
```

18.2.1 API 输出

在命令处理完成时，NWP 会传输基本响应事件。

返回代码将反映操作已成功。

可能的错误代码：

- STATUS_OK (0) – 成功
- ERROR_INVALID_PARAM(-2) – 无效长度 (IE 长度大于 252 个字节，或新 IE 长度加上所有已配置长度的总体长度大于 INFO_ELEMENT_MAX_TOTAL_LENGTH 个字节)
- ERROR_MALLOC_FAILED (-4) – IE 的长度超过 INFO_ELEMENT_MAX_TOTAL_LENGTH 阈值
- ERROR_FS_ACCESS_FAILED (-7) – 无法从存储器获取信息元素

18.3 所有信息元素的总大小上限

MAC 中的每个信标和探测器响应模板帧为 512 字节；不要添加会导致帧大于此限值的私有信息元素。

信标和探测器响应帧包括符合以下条件的信息元素：

- 始终以静态长度存在 (TIM)
- 始终以可变长度存在 (SID)
- 并非始终存在 (WPA)
- 存在于 P2P GO 下但不存在于 AP 下，或反之 (P2P IE)
- 在探测器响应和信标中以不同的长度存在 (探测器响应中的 WPS)

因此，我们可以添加的信息元素在 AP 和 P2P GO 下会有不同的大小限制。

备注

如果探测器响应是由 NWP 传输的，则在 FW 传输相应帧的情况下，大小会受到限制。我们希望私有 IE 在所有帧中都相同，因此在这种情况下不允许使用更大的 IE。

表 18-2 汇总了 SimpleLink WiFi 器件的信标和探测器响应帧中包含的所有字段。

表 18-2. 信标和探测器响应参数

			AP		P2P		
名称	id	大小	信标	探头	信标	探头	更多信息
MAC 标头		24	24	24	24	24	
时间戳		8	8	8	8	8	
信标间隔		2	2	2	2	2	
功能信息		2	2	2	2	2	
SSID	0	32	34	34	34	34	
支持的速率	1	8	10	10	10	10	
直接序列	3	1	3	3	3	3	
TIM	5	4	6	0	6	0	
国家/地区	7	6	8	8	8	8	

表 18-2. 信标和探测器响应参数 (continued)

			AP		P2P		
名称	id	大小	信标	探头	信标	探头	更多信息
ERP	42	1	3	3	3	3	
交换速率	50	4	6	6	6	6	
RSN	48	20	22	22	22	22	P2P 中为 20，AP 中为 24
WPA	221	26	28	28	0	0	在支持 WPS 的 P2P 中不存在 WPA
HT 功能	45	26	0	0	0	0	未设置
HT 操作	61	22	0	0	0	0	未设置
WMM	221	24	0	0	26	26	在自主 P2P GO 下设置（由于认证）
WPS	221	178	0	0	80	180	OUI=0050F2，类型=4。仅在 P2P 中：信标中为 78，探测器响应中为 178（取决于是否存在配置方法以及器件名称[最多 33 个字符]）
P2P（Wi-Fi 直连）	221	61	0	0	20	0	OUI=506F9A，类型=9。信标中为 18，探测器响应中为 61。在探测器响应内部，仅当请求者发送时才包含此参数。它不是探测器响应模板的一部分！！！
总结		449	156	150	254	328	

可以看出，AP 模式下的信标和探测器响应帧占用 160 字节。

在 P2P GO 下，信标占用 238 字节，探测器响应占用 348 字节。

这样就会在 AP 角色中为私有信息元素留下大约 350 字节，在 P2P GO 角色中为私有信息元素留下大约 120 字节。

This page intentionally left blank.

19.1 捕获 NWP 日志

19.1.1 概述

NWP 日志可以帮助 TI 工程师调试各种问题。这些日志可以作为加密的二进制内容从专用的 UART 引脚读取。

如果 TI 工程师要求捕获 NWP（网络处理器）日志，请按照以下说明将日志文件发送给 TI 支持部门。

19.1.2 指令

19.1.2.1 为 CC32xx 配置引脚复用

在引脚初始化代码中（例如，在 SDK 的 pinmux.c 示例中）添加以下行：

```
// If your application already has UART0 configured, no need for this line
MAP_PRCMPeripheralClkEnable(PRCM_UARTA0, PRCM_RUN_MODE_CLK);
 // Mux Pin 62 to mode 1 for outputting NWP logs
MAP_PinTypeUART(PIN_62, PIN_MODE_1);
```

必须包括以下头文件以启用干净编译：

```
#include "hw_types.h"
#include "rom_map.h"
#include "pin.h"
#include "prcm.h"
```

确保不会与引脚 62 产生冲突。

提取 P1.62 并将它连接至串行至 USB 转换器。如果有 CC31XXEMUBOOT，请将信号连接至引脚 P4.7。

如果正在使用 CC3100 BoosterPack 模块且它已安装在 CC31XXEMUBOOT 上，则在此步骤中不需要执行任何操作。

19.1.2.2 终端设置

打开诸如 TeraTerm 或 Putty 之类的串行连接应用程序（可以使用任何其他终端仿真器程序），并配置以下设置：

表 19-1. 终端设置

参数	值
波特率	921600
数据位	8
停止位	1
奇偶校验	无
流控	无

如果使用的是 CC31XXEMUBOOST，请在 Device Manager 中连接第四个端口。

配置终端仿真以便在二进制模式（而不是文本/ASCII 模式）下运行并记录日志。以下是一些示例：

Tera Term

Setup → Serial Port…

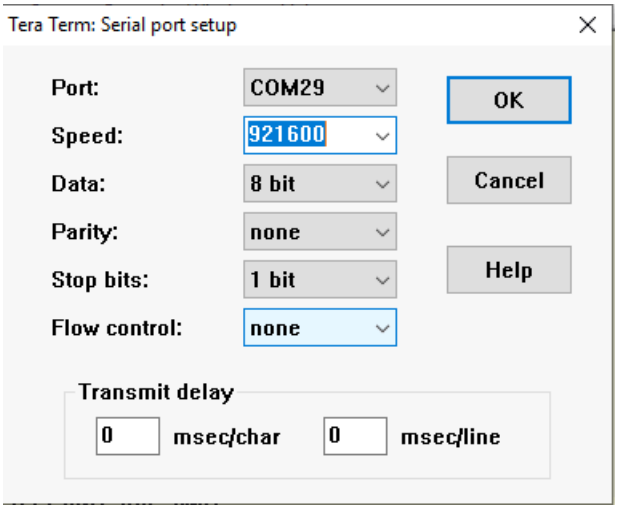


图 19-1. Tera Term 端口设置

File → Log...

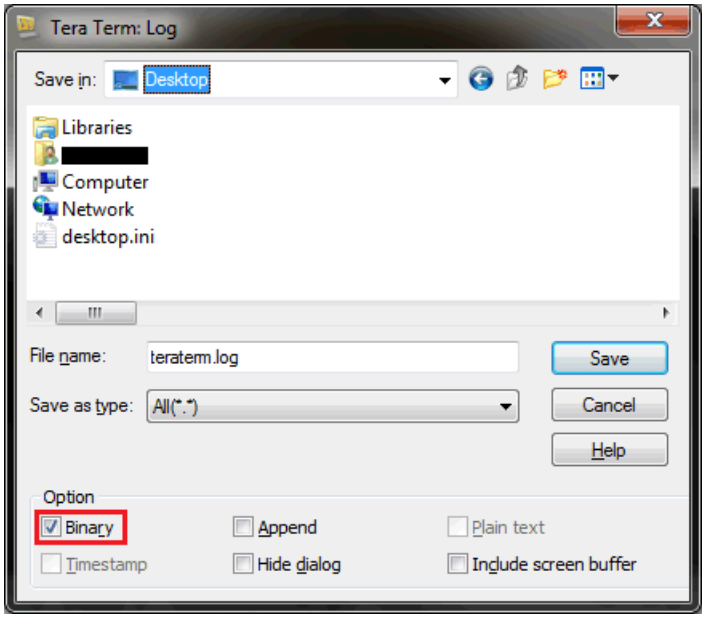


图 19-2. Tera Term 日志设置

Putty

在屏幕左侧，选择 Connection → Serial：

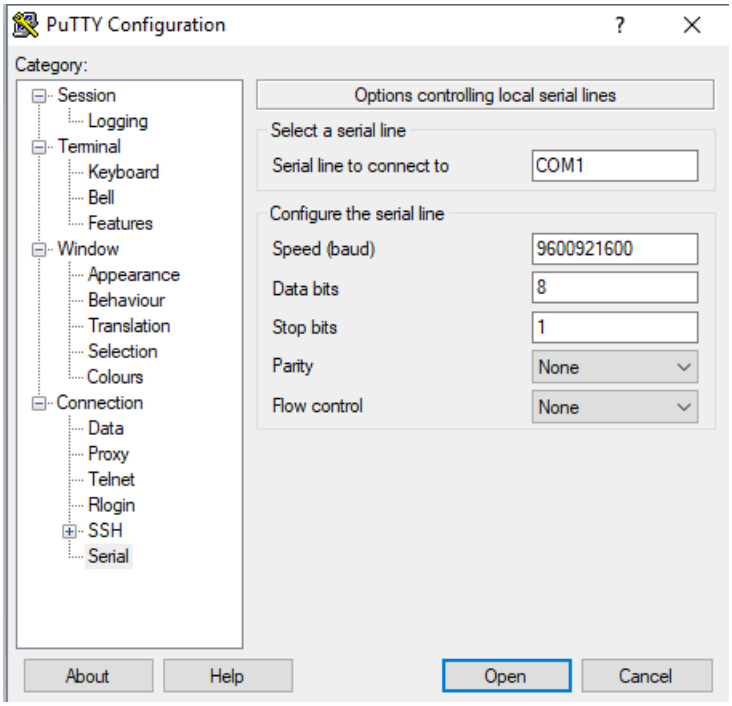


图 19-3. Putty 端口设置

在连接到端口之前的菜单屏幕中：

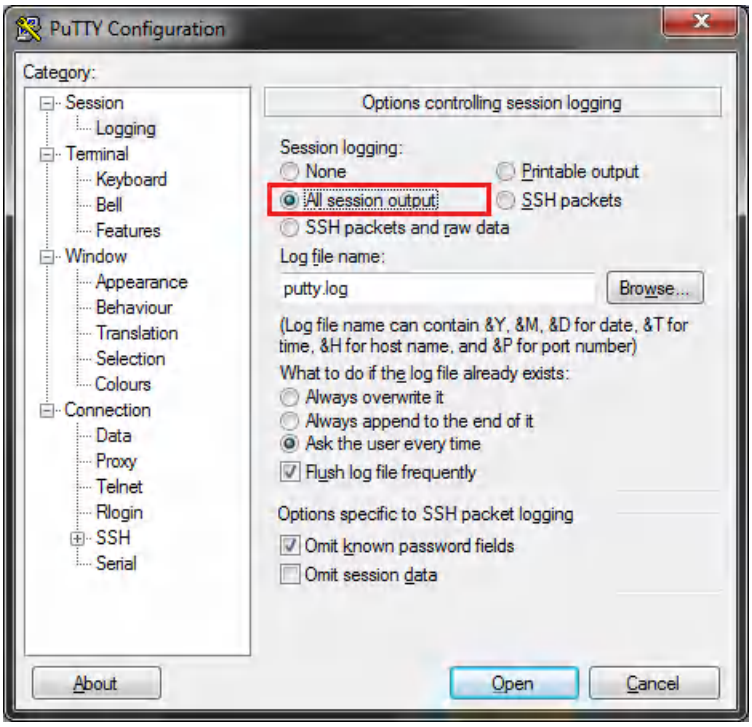


图 19-4. Putty 日志设置

19.1.2.3 运行程序

运行应用程序并确保正在记录和保存日志。只要控制台上有输出，就可以随时截取日志。控制台输出应该是不可读的。

19.1.2.4 发送给 TI 工程师

将日志文件交付给 TI 工程师进行调查。

This page intentionally left blank.

参考文献

- 德州仪器 (TI)：*适用于 MCU 应用的 CC3100 SimpleLink™ Wi-Fi® 网络处理器、物联网解决方案数据表*
- 德州仪器 (TI)：*CC3200 SimpleLink™ Wi-Fi® 和物联网解决方案（单芯片无线 MCU）数据表*
- CC31xx 主机驱动程序 API
- www.ti.com/simplelink Wiki
- TI E2E™ 在线社区 — TI 的工程师交流 (E2E) 社区。创建此社区的目的在于促进工程师之间的协作。在 e2echina.ti.com 中，您可以咨询问题、分享知识、拓展思路并与同行工程师一道帮助解决问题。

This page intentionally left blank.

A.1 概述

从软件的角度来看，系统可以分为几个部分：

- 用户应用程序
- SimpleLink WiFi 主机驱动程序 – 与平台无关的部分
  - 主机驱动程序 API
  - 主驱动程序逻辑和流程
- SimpleLink WiFi 主机驱动程序 – 与平台相关的部分
  - 操作系统包装器实现
  - 传输层（SPI 和 UART）实现

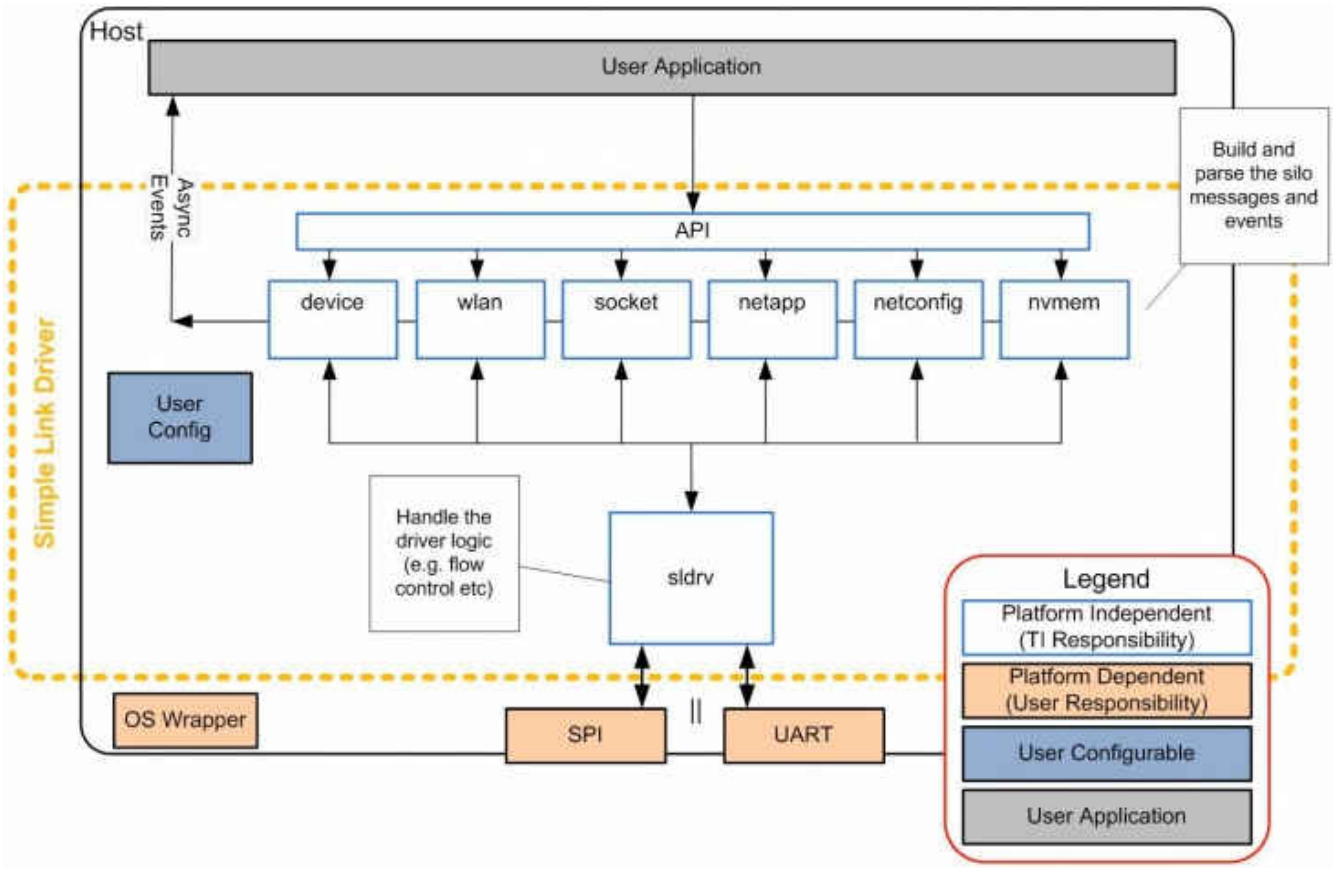


图 A-1. SimpleLink WiFi 主机驱动程序配置

A.1.1 SimpleLink WiFi 主机驱动程序 – 与平台无关的部分

主机驱动程序的实现包括驱动程序 API、SimpleLink 网络初始化、SimpleLink 网络命令、命令响应处理、异步事件处理、数据流以及传输层接口。所有这些都是 TI 提供的与平台无关且与操作系统无关的代码。驱动程序 API 分成六个孤岛，反映六种不同的逻辑 API 类型：

- **器件 API** – 处理与硬件相关的 API
- **WLAN API** – 处理与 WLAN 802.11 协议相关的功能
- **套接字 API** – 用户应用程序最常用的 API 集。SimpleLink 网络套接字 API 与 Berkeley 套接字 API 相符。
- **NetApp API** – 处理其他网络协议和功能（作为片上内容的补充部分提供）。
- **NetCfg API** – 处理不同网络参数的配置
- **FS API** – 处理对串行闪存组件的访问，以进行网络或用户专有数据的读取和写入操作

A.1.2 SimpleLink WiFi 主机驱动程序 – 与平台相关的部分

该驱动程序具有两个由用户提供的软件组件：

- 传输层实现 – *必需*
  驱动程序代码提供所需的传输接口和总线接口。
  用户必须根据选择的传输层（SPI 或 UART）和使用的平台提供函数实现。
- 操作系统包装器 – *可选*
  驱动程序代码提供所需的操作系统包装器接口。
  若要使用操作系统，首先应根据使用的操作系统和平台提供函数实现。

A.1.3 SimpleLink WiFi 驱动程序配置

该驱动程序提供了一个配置文件，使用户能够控制支持的 API 集、存储器分配模块和操作系统使用情况。此外，还提供了一些预配置选项。预配置包含一组已由 TI 针对特定场景创建和测试的驱动程序自定义配置和设置。

A.1.4 用户应用程序

用户应用程序由用户开发和拥有。应用程序使用驱动程序 API 和异步驱动程序事件与 SimpleLink 网络驱动程序进行连接。

A.2 驱动程序数据流

A.2.1 传输层协议

主机驱动程序使用以下类型的消息：

- 命令 [主机 -> SimpleLink 网络处理器]
- 命令完成 [SimpleLink 网络处理器 -> 主机]
- 异步事件 [SimpleLink 网络处理器 -> 主机]
- 数据 [主机 <-/-> SimpleLink 网络处理器]

A.2.2 命令和命令完成

命令是从主机到 SimpleLink 网络处理器的任何消息，而不是数据包（发送或接收）。主机驱动程序一次仅支持一条命令。在发送下一条命令之前，驱动程序等待从 SimpleLink 网络处理器收到“命令完成”消息。就 RT 和嵌入式系统而言，为避免长时间阻塞驱动程序，在某些命令中，SimpleLink 网络处理器会在收到并验证命令后立即发送“命令完成”消息，并在命令成功执行后发送异步事件。

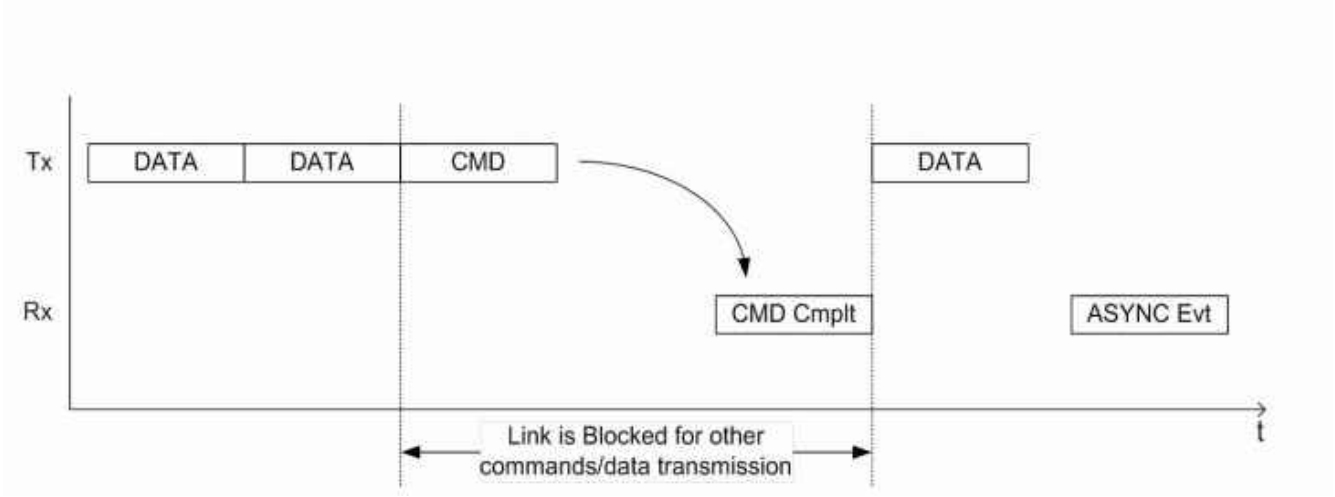


图 A-2. 阻塞的链路

A.2.3 数据事务

数据消息是使用套接字发送和接收 API 向用户发送或来自用户的任何信息（通常为 TCP 或 UDP 数据包）。

A.2.3.1 数据发送（从主机到 SimpleLink 网络处理器）

向 SimpleLink 网络处理器发送数据类似于命令流。区别在于数据包没有命令完成消息，因此主机未阻塞或正等待消息。可以按顺序从用户应用发送数据包；驱动程序负责使用数据流控制逻辑来避免缓冲区溢出。

A.2.3.2 数据流控制

状态字段 – 包含在从 SimpleLink 网络处理器到主机的每条消息（异步事件）中：

- TxBuff – 指示 SimpleLink 网络处理器内可用 Tx 缓冲区的数量。

图 A-3. 数据流控制

- 主机可以发送最多 1500 字节的 TxBuff 数据包，而无需等待来自 SimpleLink 网络处理器的响应。
- 如果数字发生更改，SimpleLink 网络处理器会生成一个虚拟事件（如果更改值小于阈值，则使用上次更新/超时的阈值）。

A.2.3.3 数据接收（从 SimpleLink 网络处理器到主机）

从 SimpleLink 网络处理器接收数据的方式有两种：阻塞式和非阻塞式。

A.2.3.4 阻塞接收

调用阻塞 *sl_Recv* 后，主机驱动程序会向 SimpleLink 网络处理器发出命令。系统会阻塞驱动程序，等到异步事件通知驱动程序：准备在 SimpleLink 网络处理器中读取数据包。接收到此异步事件之后，驱动程序继续从主机读取数据包。

A.2.3.5 非阻塞接收

如果命令 complete status 返回一个状态，通知没有数据在等待主机，则该命令以挂起的返回状态 (SL_EAGAIN) 返回。如果数据在等待主机，则驱动程序继续从主机读取数据包。

This page intentionally left blank.

B.1 错误代码

表 B-1. 一般错误代码

错误名称	值	注释
SL_RET_CODE_OK	0	
SL_RET_CODE_INVALID_INPUT	-2	
SL_RET_CODE_SELF_ERROR	-3	
SL_RET_CODE_NWP_IF_ERROR	-4	
SL_RET_CODE_MALLOC_ERROR	-5	

表 B-2. 器件错误代码

错误名称	值	注释
SL_ERROR_STATIC_ADDR_SUBNET_ERROR	-60	网络堆栈错误
SL_ERROR_ILLEGAL_CHANNEL	-61	请求方错误
SL_ERROR_SUPPLICANT_ERROR	-72	初始化错误代码
SL_ERROR_HOSTAPD_INIT_FAIL	-73	初始化错误代码
SL_ERROR_HOSTAPD_INIT_IF_FAIL	-74	初始化错误代码
SL_ERROR_WLAN_DRV_INIT_FAIL	-75	初始化错误代码
SL_ERROR_WLAN_DRV_START_FAIL	-76	wlan 启动错误
SL_ERROR_FS_FILE_TABLE_LOAD_FAILED	-77	初始化文件系统失败
SL_ERROR_PREFERRED_NETWORKS_FILE_LOAD_FAILED	-78	初始化文件系统失败
SL_ERROR_HOSTAPD_BSSID_VALIDATION_ERROR	-79	Ap 配置 BSSID 错误
SL_ERROR_HOSTAPD_FAILED_TO_SETUP_INTERFACE	-80	Ap 配置接口错误
SL_ERROR_MDNS_ENABLE_FAIL	-81	mDNS 启用失败
SL_ERROR_HTTP_SERVER_ENABLE_FAILED	-82	HTTP 服务器启用失败
SL_ERROR_DHCP_SERVER_ENABLE_FAILED	-83	DHCP 服务器启用失败
SL_ERROR_PREFERRED_NETWORK_LIST_FULL	-93	请求方错误
SL_ERROR_PREFERRED_NETWORKS_FILE_WRITE_FAILED	-94	请求方错误
SL_ERROR_DHCP_CLIENT_RENEW_FAILED	-100	DHCP 客户端错误
SL_ERROR_CON_MGMT_STATUS_UNSPECIFIED	-102	WLAN 连接
SL_ERROR_CON_MGMT_STATUS_AUTH_REJECT	-103	WLAN 连接
SL_ERROR_CON_MGMT_STATUS_ASSOC_REJECT	– 104	WLAN 连接
SL_ERROR_CON_MGMT_STATUS_SECURITY_FAILURE	-105	WLAN 连接
SL_ERROR_CON_MGMT_STATUS_AP_DEAUTHENTICATE	-106	WLAN 连接
SL_ERROR_CON_MGMT_STATUS_AP_DISASSOCIATE	-107	WLAN 连接
SL_ERROR_CON_MGMT_STATUS_ROAMING_TRIGGER	-108	WLAN 连接
SL_ERROR_CON_MGMT_STATUS_DISCONNECT_DURING_ CONNECT	-109	WLAN 连接
SL_ERROR_CON_MGMT_STATUS_SG_RESELECT	-110	WLAN 连接

表 B-2. 器件错误代码 (continued)

错误名称	值	注释
SL_ERROR_CON_MGMT_STATUS_ROC_FAILURE	-111	WLAN 连接
SL_ERROR_CON_MGMT_STATUS_MIC_FAILURE	-112	WLAN 连接
SL_ERROR_WAKELOCK_ERROR_PREFIX	-115	唤醒锁定过期
SL_ERROR_LENGTH_ERROR_PREFIX	-116	Uart 标头长度错误
SL_ERROR_MDNS_CREATE_FAIL	-121	mDNS 创建失败
SL_ERROR_GENERAL_ERROR	-127	

表 B-3. 套接字错误代码

错误名称	值	注释
SL_SOC_OK	0	
SL_SOC_ERROR	-1	
SL_INEXE	-8	套接字命令正在执行中
SL_EBADF	-9	文件编号错误
SL_ENSOCK	-10	已达到最大套接字数目
SL_EAGAIN	-11	重试 (适用于非阻塞命令)
SL_ENOMEM	-12	内存不足
SL_EACCES	-13	权限被拒绝
SL_EFAULT	-14	地址错误
SL_ECLOSE	-15	关闭套接字操作无法传输所有排队的数据包
SL_EALREADY_ENABLED	-21	收发器 - 收发器已开启
SL_EINVAL	-22	参数无效
SL_EAUTO_CONNECT_OR_CONNECTING	-69	收发器 - 在连接期间，已连接或自动模式已启动
SL_CONNECTION_PENDING	-72	收发器 - 器件已连接，先断开连接以打开收发器
SL_EUNSUPPORTED_ROLE	-86	收发器 - 在 WLAN 角色为 AP 或 P2P GO 时尝试启动
SL_EDESTADDRREQ	-89	需要目标地址
SL_EPROTOTYPE	-91	套接字的协议类型错误
SL_ENOPROTOOPT	-92	协议不可用
SL_EPROTONOSUPPORT	-93	不支持的协议
SL_ESOCKTNOSUPPORT	-94	不支持的套接字类型
SL_EOPNOTSUPP	-95	在传输终点上不支持的操作
SL_EAFNOSUPPORT	-97	协议不支持的地址系列
SL_EADDRINUSE	-98	地址已使用
SL_EADDRNOTAVAIL	-99	无法分配请求的地址
SL_ENETUNREACH	-101	网络无法访问
SL_ENOBUFS	-105	无可用的缓冲区空间
define SL_EISCONN	-106	传输终点已连接
SL_ENOTCONN	-107	传输终点未连接
SL_ETIMEDOUT	-110	连接超时
SL_ECONNREFUSED	-111	拒绝连接
SL_EALREADY	-114	正在进行非阻塞连接，请重试
SL_ESEC_RSA_WRONG_TYPE_E	-130	RSA 函数的 RSA 块类型错误
SL_ESEC_RSA_BUFFER_E	-131	RSA 缓冲区错误，输出太小
SL_ESEC_BUFFER_E	-132	输出缓冲区太小或输入太大
SL_ESEC_ALGO_ID_E	-133	设置 algo ID 错误

表 B-3. 套接字错误代码 (continued)

错误名称	值	注释
SL_ESEC_PUBLIC_KEY_E	-134	设置公钥错误
SL_ESEC_DATE_E	-135	设置日期有效性错误
SL_ESEC_SUBJECT_E	-136	设置主题名称错误
SL_ESEC_ISSUER_E	-137	设置发行人名称错误
SL_ESEC_CA_TRUE_E	-138	设置 CA 基本约束 true 错误
SL_ESEC_EXTENSIONS_E	-139	设置扩展错误
SL_ESEC_ASN_PARSE_E	-140	ASN 解析错误，无效的输入
SL_ESEC_ASN_VERSION_E	-141	ASN 版本错误，无效的编号
SL_ESEC_ASN_GETINT_E	-142	ASN 获取 big _i16 错误，无效的数据
SL_ESEC_ASN_RSA_KEY_E	-143	ASN 密钥初始化错误，无效的输入
SL_ESEC_ASN_OBJECT_ID_E	-144	ASN 对象 ID 错误，无效的 ID
SL_ESEC_ASN_TAG_NULL_E	-145	ASN 标记错误，非空值
SL_ESEC_ASN_EXPECT_0_E	-146	ASN 预期错误，非零
SL_ESEC_ASN_BITSTR_E	-147	ASN 位字符串错误，错误的 ID
SL_ESEC_ASN_UNKNOWN_OID_E	-148	ASN oid 错误，未知总和 ID
SL_ESEC_ASN_DATE_SZ_E	-149	ASN 日期错误，错误的大小
SL_ESEC_ASN_BEFORE_DATE_E	-150	ASN 日期错误，当前日期之前
SL_ESEC_ASN_AFTER_DATE_E	-151	ASN 日期错误，当前日期之后
SL_ESEC_ASN_SIG_OID_E	-152	ASN 签名错误，不匹配的 oid
SL_ESEC_ASN_TIME_E	-153	ASN 时间错误，未知的时间类型
SL_ESEC_ASN_INPUT_E	-154	ASN 输入错误，没有足够的数据
SL_ESEC_ASN_SIG_CONFIRM_E	-155	ASN sig 错误，确认失败
SL_ESEC_ASN_SIG_HASH_E	-156	ASN sig 错误，不支持的哈希类型
SL_ESEC_ASN_SIG_KEY_E	-157	ASN sig 错误，不支持的密钥类型
SL_ESEC_ASN_DH_KEY_E	-158	ASN 密钥初始化错误，无效的输入
SL_ESEC_ASN_NTRU_KEY_E	-159	ASN ntru 密钥解码错误，无效的输入
SL_ESEC_ECC_BAD_ARG_E	-170	ECC 输入参数类型错误
SL_ESEC_ASN_ECC_KEY_E	-171	ASN ECC 错误输入
SL_ESEC_ECC_CURVE_OID_E	-172	不支持的 ECC OID 曲线类型
SL_ESEC_BAD_FUNC_ARG	-173	提供了错误的函数参数
SL_ESEC_NOT_COMPILED_IN	-174	未编译的功能
SL_ESEC_UNICODE_SIZE_E	-175	Unicode 密码太大
SL_ESEC_NO_PASSWORD	-176	用户未提供密码
SL_ESEC_ALT_NAME_E	-177	alt 名称大小问题，太大
SL_ESEC_AES_GCM_AUTH_E	-180	AES-GCM 身份验证检查失败
SL_ESEC_AES_CCM_AUTH_E	-181	AES-CCM 身份验证检查失败
SL_ESEC_CLOSE_NOTIFY	-300	ssl/tls 警报
SL_ESEC_UNEXPECTED_MESSAGE	-310	ssl/tls 警报
SL_ESEC_BAD_RECORD_MAC	-320	ssl/tls 警报
SL_ESEC_DECRYPTION_FAILED	-321	ssl/tls 警报
SL_ESEC_RECORD_OVERFLOW	-322	ssl/tls 警报
SL_ESEC_DECOMPRESSION_FAILURE	-330	ssl/tls 警报
SL_ESEC_HANDSHAKE_FAILURE	-340	ssl/tls 警报

表 B-3. 套接字错误代码 (continued)

错误名称	值	注释
SL_ESEC_NO_CERTIFICATE	-341	ssl/tls 警报
SL_ESEC_BAD_CERTIFICATE	-342	ssl/tls 警报
SL_ESEC_UNSUPPORTED_CERTIFICATE	-343	ssl/tls 警报
SL_ESEC_CERTIFICATE_REVOKED	-344	ssl/tls 警报
SL_ESEC_CERTIFICATE_EXPIRED	-345	ssl/tls 警报
SL_ESEC_CERTIFICATE_UNKNOWN	-346	ssl/tls 警报
SL_ESEC_ILLEGAL_PARAMETER	-347	ssl/tls 警报
SL_ESEC_UNKNOWN_CA	-348	ssl/tls 警报
SL_ESEC_ACCESS_DENIED	-349	ssl/tls 警报
SL_ESEC_DECODE_ERROR	-350	ssl/tls 警报
SL_ESEC_DECRYPT_ERROR	-351	ssl/tls 警报
SL_ESEC_EXPORT_RESTRICTION	-360	ssl/tls 警报
SL_ESEC_PROTOCOL_VERSION	-370	ssl/tls 警报
SL_ESEC_INSUFFICIENT_SECURITY	-370	ssl/tls 警报
SL_ESEC_INTERNAL_ERROR	-380	ssl/tls 警报
SL_ESEC_USER_CANCELLED	-390	ssl/tls 警报
SL_ESEC_NO_RENEGOTIATION	-400	ssl/tls 警报
SL_ESEC_UNSUPPORTED_EXTENSION	-410	ssl/tls 警报
SL_ESEC_CERTIFICATE_UNOBTAINABLE	-411	ssl/tls 警报
SL_ESEC_UNRECOGNIZED_NAME	-412	ssl/tls 警报
SL_ESEC_BAD_CERTIFICATE_STATUS_RESPONSE	-413	ssl/tls 警报
SL_ESEC_BAD_CERTIFICATE_HASH_VALUE	-414	ssl/tls 警报
SL_ESECGENERAL	-450	错误安全级别一般错误
SL_ESECDECRYPT	-451	错误安全级别，解密接收数据包失败
SL_ESECCLOSED	-452	安全层按其他大小（长度）封闭，tcp 仍处于连接状态
SL_ESECSNOVERIFY	-453	在没有服务器验证的情况下连接
SL_ESECNOCAFILE	-454	错误安全级别，找不到 CA 文件
SL_ESECMEMORY	-455	错误安全级别，没有可用的存储器空间
SL_ESECBADCAFILE	-456	错误安全级别，错误的 CA 文件
SL_ESECBADCERTFILE	-457	错误安全级别，错误的证书文件
SL_ESECBADPRIVATEFILE	-458	错误安全级别，错误的私密文件
SL_ESECBADDHFILE	-459	错误安全级别，错误的 DH 文件
SL_ESECT00MANYSSLOPENED	-460	开放了最大 SSL 套接字数
SL_ESECDATEERROR	-461	已连接，但出现证书日期验证错误
SL_ESECHANDSHAKETIMEDOUT	-462	由于握手时间过长，连接超时

表 B-4. WLAN 错误代码

错误名称	值	注释
SL_ERROR_KEY_ERROR	-3	
SL_ERROR_WIFI_NOT_CONNECTED	-59	
SL_ERROR_INVALID_ROLE	-71	
SL_ERROR_INVALID_SECURITY_TYPE	-84	
SL_ERROR_PASSPHRASE_TOO_LONG	-85	

表 B-4. WLAN 错误代码 (continued)

错误名称	值	注释
SL_ERROR_WPS_NO_PIN_OR_WRONG_PIN_LEN	-87	
SL_ERROR_EAP_WRONG_METHOD	-88	
SL_ERROR_PASSWORD_ERROR	-89	
SL_ERROR_EAP_ANONYMOUS_LEN_ERROR	-90	
SL_ERROR_SSID_LEN_ERROR	-91	
SL_ERROR_USER_ID_LEN_ERROR	-92	
SL_ERROR_ILLEGAL_WEP_KEY_INDEX	-95	
SL_ERROR_INVALID_DWELL_TIME_VALUES	-96	
SL_ERROR_INVALID_POLICY_TYPE	-97	
SL_ERROR_PM_POLICY_INVALID_OPTION	-98	
SL_ERROR_PM_POLICY_INVALID_PARAMS	-99	
SL_ERROR_WIFI_ALREADY_DISCONNECTED	-129	

表 B-5. NetApp 错误代码

错误名称	值	注释
SL_ERROR_DEVICE_NAME_LEN_ERR	-117	设置器件名称错误
SL_ERROR_DEVICE_NAME_INVALID	-118	设置器件名称错误
SL_ERROR_DOMAIN_NAME_LEN_ERR	-119	设置域名称错误
SL_ERROR_DOMAIN_NAME_INVALID	-120	设置域名称错误
SL_NET_APP_DNS_QUERY_NO_RESPONSE	-159	DNS 查询失败，无响应
SL_NET_APP_DNS_NO_SERVER	-161	未指定 DNS 服务器
SL_NET_APP_DNS_PARAM_ERROR	-162	mDNS 参数错误
SL_NET_APP_DNS_QUERY_FAILED	-163	DNS 查询失败
SL_NET_APP_DNS_INTERNAL_1	-164	
SL_NET_APP_DNS_INTERNAL_2	-165	
SL_NET_APP_DNS_MALFORMED_PACKET	-166	接收到格式不正确或损坏的 DNS 数据包
SL_NET_APP_DNS_INTERNAL_3	-167	
SL_NET_APP_DNS_INTERNAL_4	-168	
SL_NET_APP_DNS_INTERNAL_5	-169	
SL_NET_APP_DNS_INTERNAL_6	-170	
SL_NET_APP_DNS_INTERNAL_7	-171	
SL_NET_APP_DNS_INTERNAL_8	-172	
SL_NET_APP_DNS_INTERNAL_9	-173	
SL_NET_APP_DNS_MISMATCHED_RESPONSE	-174	服务器响应类型与查询请求不匹配
SL_NET_APP_DNS_INTERNAL_10	-175	
SL_NET_APP_DNS_INTERNAL_11	-176	
SL_NET_APP_DNS_NO_ANSWER	-177	一次性查询无响应
SL_NET_APP_DNS_NO_KNOWN_ANSWER	-178	查询答案未知
SL_NET_APP_DNS_NAME_MISMATCH	-179	根据 RFC，服务名称不合法
SL_NET_APP_DNS_NOT_STARTED	-180	mDNS 未运行
SL_NET_APP_DNS_HOST_NAME_ERROR	-181	主机名错误
SL_NET_APP_DNS_NO_MORE_ENTRIES	-182	找不到更多条目
SL_NET_APP_DNS_MAX_SERVICES_ERROR	-200	已配置最大广播服务数
SL_NET_APP_DNS_IDENTICAL_SERVICES_ERROR	-201	尝试注册的服务已存在
SL_NET_APP_DNS_NOT_EXISTED_SERVICE_ERROR	-203	尝试删除的服务不存在

表 B-5. NetApp 错误代码 (continued)

错误名称	值	注释
SL_NET_APP_DNS_ERROR_SERVICE_NAME_ERROR	-204	重试请求
SL_NET_APP_DNS_RX_PACKET_ALLOCATION_ERROR	-205	
SL_NET_APP_DNS_BUFFER_SIZE_ERROR	-206	列表大小缓冲区大于 NWP 内部允许的大小
SL_NET_APP_DNS_NET_APP_SET_ERROR	-207	其中一个 mDNS Set 函数的长度不合法
SL_NET_APP_DNS_GET_SERVICE_LIST_FLAG_ERROR	-208	
SL_NET_APP_DNS_NO_CONFIGURATION_ERROR	-209	
SL_ERROR_NETAPP_RX_BUFFER_LENGTH_ERROR	-230	

表 B-6. FS 错误代码

错误名称	值	注释
SL_FS_OK	0	
SL_FS_ERR_NOT_SUPPORTED	-1	
SL_FS_ERR_FAILED_TO_READ	-2	
SL_FS_ERR_INVALID_MAGIC_NUM	-3	
SL_FS_ERR_DEVICE_NOT_LOADED	-4	
SL_FS_ERR_FAILED_TO_CREATE_LOCK_OBJ	-5	
SL_FS_ERR_UNKNOWN	-6	
SL_FS_ERR_FS_ALREADY_LOADED	-7	
SL_FS_ERR_FAILED_TO_CREATE_FILE	-8	
SL_FS_ERR_INVALID_ARGS	-9	
SL_FS_ERR_EMPTY_ERROR	-10	
SL_FS_ERR_FILE_NOT_EXISTS	-11	
SL_FS_ERR_INVALID_FILE_ID	-12	
SL_FS_ERR_READ_DATA_LENGTH	-13	
SL_FS_ERR_ALLOC	-14	
SL_FS_ERR_OFFSET_OUT_OF_RANGE	-15	
SL_FS_ERR_FAILED_TO_WRITE	-16	
SL_FS_ERR_INVALID_HANDLE	-17	
SL_FS_ERR_FAILED_LOAD_FILE	-18	
SL_FS_ERR_CONTINUE_WRITE_MUST_BE_MOD_4	-19	
SL_FS_ERR_FAILED_INIT_STORAGE	-20	
SL_FS_ERR_FAILED_READ_NVFILE	-21	
SL_FS_ERR_BAD_FILE_MODE	-22	
SL_FS_ERR_FILE_ACCESS_IS_DIFFERENT	-23	
SL_FS_ERR_NO_ENTRIES_AVAILABLE	-24	
SL_FS_ERR_PROGRAM	-25	
SL_FS_ERR_FILE_ALREADY_EXISTS	-26	
SL_FS_ERR_INVALID_ACCESS_TYPE	-27	
SL_FS_ERR_FILE_EXISTS_ON_DIFFERENT_DEVICE_ID	-28	
SL_FS_ERR_FILE_MAX_SIZE_BIGGER_THAN_EXISTING_FILE	-29	
SL_FS_ERR_NO_AVAILABLE_BLOCKS	-30	
SL_FS_ERR_FAILED_TO_READ_INTEGRITY_HEADER_1	-31	
SL_FS_ERR_FAILED_TO_READ_INTEGRITY_HEADER_2	-32	
SL_FS_ERR_FAILED_TO_ALLOCATE_MEM	-33	
SL_FS_ERR_NO_AVAILABLE_NV_INDEX	-34	

表 B-6. FS 错误代码 (continued)

错误名称	值	注释
SL_FS_ERR_FAILED_WRITE_NVMEM_HEADER	-35	
SL_FS_ERR_DEVICE_IS_NOT_FORMATTED	-36	
SL_FS_WARNING_FILE_NAME_NOT_KEPT	-37	
SL_FS_ERR_SIZE_OF_FILE_EXT_EXCEEDED	-38	
SL_FS_ERR_FILE_IMAGE_IS_CORRUPTED	-39	
SL_FS_INVALID_BUFFER_FOR_WRITE	-40	
SL_FS_INVALID_BUFFER_FOR_READ	-41	
SL_FS_FILE_MAX_SIZE_EXCEEDED	-42	
SL_FS_ERR_MAX_FS_FILES_IS_SMALLER	-43	
SL_FS_ERR_MAX_FS_FILES_IS_LARGER	-44	
SL_FS_FILE_HAS_RESERVED_NV_INDEX	-45	
SL_FS_ERR_OVERLAP_DETECTION_THRESHHOLD	-46	
SL_FS_DATA_IS_NOT_ALIGNED	-47	
SL_FS_DATA_ADDRESS_SHOUD_BE_IN_DATA_RAM	-48	
SL_FS_NO_DEVICE_IS_LOADED	-49	
SL_FS_ERR_TOKEN_IS_NOT_VALID	-50	
SL_FS_FILE_UNVALID_FILE_SIZE	-51	
SL_FS_SECURITY_ALLERT	-52	
SL_FS_FILE_SYSTEM_IS_LOCKED	-53	
SL_FS_WRONG_FILE_NAME	-54	
SL_FS_ERR_FAILED_READ_NVMEM_HEADER	-55	
SL_FS_ERR_INCORRECT_OFFSET_ALIGNMENT	-56	
SL_FS_SECURE_FILE_MUST_BE_COMMIT	-57	
SL_FS_SECURITY_BUF_ALREADY_ALLOC	-58	
SL_FS_FILE_NAME_EXIST	-59	
SL_FS_CERT_CHAIN_ERROR	-60	
SL_FS_NOT_16_ALIGNED	-61	
SL_FS_WRONG_SIGNATURE_OR_CERTIFIC_NAME_LENGTH	-62	
SL_FS_WRONG_SIGNATURE	-63	
SL_FS_FILE_HAS_NOT_BEEN_CLOSE_CORRECTLY	-64	
SL_FS_ERASING_FLASH	-65	
SL_FS_ERR_FILE_IS_NOT_SECURE_AND_SIGN	-66	
SL_FS_ERR_EMPTY_SFLASH	-67	

表 B-7. Rx 过滤器错误代码

错误名称	值	注释
RXFL_OK	0	
RXFL_NUMBER_OF_FILTER_EXCEEDED	23	已超过最大过滤器数目
RXFL_NO_FILTERS_ARE_DEFINED	24	系统中未定义过滤器
RXFL_UPDATE_NOT_SUPPORTED	31	不支持更新
RXFL_RULE_HEADER_FIELD_ID_OUT_OF_RANGE	32	规则字段 ID 超出范围
RXFL_RULE_HEADER_COMBINATION_OPERATOR_OUT_OF_RANGE	33	组合函数 ID 超出范围
RXFL_RULE_HEADER_OUT_OF_RANGE	34	标头规则超出范围
RXFL_RULE_HEADER_NOT_SUPPORTED	35	当前版本不支持标头规则
RXFL_RULE_HEADER_FIELD_ID_ASCII_NOT_SUPPORTED	36	不支持该 ASCII 字段 ID

表 B-7. Rx 过滤器错误代码 (continued)

错误名称	值	注释
RXFL_RULE_FIELD_ID_NOT_SUPPORTED	37	规则字段 ID 超出范围
RXFL_FRAME_TYPE_NOT_SUPPORTED	38	ASCII 帧类型字符串非法
RXFL_RULE_HEADER_COMPARE_FUNC_OUT_OF_RANGE	39	规则比较函数超出范围
RXFL_RULE_HEADER_TRIGGER_OUT_OF_RANGE	40	触发器超出范围
RXFL_RULE_HEADER_TRIGGER_COMPARE_FUNC_OUT_OF_RANGE	41	触发器比较函数超出范围
RXFL_RULE_HEADER_ACTION_TYPE_NOT_SUPPORTED	42	不支持该操作类型
RXFL_DEPENDENT_FILTER_DO_NOT_EXIST_1	43	父过滤器为空
RXFL_DEPENDENT_FILTER_DO_NOT_EXIST_2	44	父过滤器不存在
RXFL_DEPENDENT_FILTER_SYSTEM_STATE_DO_NOT_FIT	45	过滤器与其依赖项系统状态不匹配
RXFL_DEPENDENT_FILTER_LAYER_DO_NOT_FIT	46	过滤器及其依赖项应来自同一层
RXFL_ACTION_NO_REG_NUMBER	47	操作需要计数器编号
RXFL_NUMBER_OF_ARGS_EXCEEDED	48	已超过参数数量
RXFL_DEPEDENCY_NOT_ON_THE_SAME_LAYER	49	过滤器及其依赖项必须在同一层
RXFL_FILTER_DO_NOT_EXISTS	50	过滤器不存在
RXFL_DEPENDENT_FILTER_DEPENDENCY_ACTION_IS_DROP	51	参考列过滤器具有 DROP 操作，因而无法创建过滤器
RXFL_NUMBER_OF_CONNECTION_POINTS_EXCEEDED	52	已超过连接点数目
RXFL_DEPENDENCY_IS_DISABLED	58	如果禁用了过滤器的参考列过滤器，则无法启用该过滤器
RXFL_CHILD_IS_ENABLED	59	启用了子过滤器时，无法禁用过滤器
RXFL_FILTER_HAS_CHILDS	60	该过滤器包含子过滤器，无法删除
RXFL_DEPENDENT_FILTER_IS_NOT_ENABLED	61	未启用参考列过滤器
RXFL_DEPENDENT_FILTER_IS_NOT_PERSISTENT	62	参考列过滤器不持续
RXFL_WRONG_MULTICAST_ADDRESS	63	地址应为多播类型
RXFL_WRONG_COMPARE_FUNC_FOR_BROADCAST_ADDRESS	64	比较函数不适用于广播地址
RXFL_THE_FILTER_IS_NOT_OF_HEADER_TYPE	65	过滤器应具有标头类型
RXFL_WRONG_MULTICAST_BROADCAST_ADDRESS	66	地址应为多播或广播类型
RXFL_FIELD_SUPPORT_ONLY_EQUAL_AND_NOTEQUAL	67	规则比较函数 ID 超出范围
RXFL_ACTION_USE_REG1_TO_REG4	68	仅允许将计数器 1 - 4 用于操作
RXFL_ACTION_USE_REG5_TO_REG8	69	仅允许将计数器 5 - 8 用于操作
RXFL_TRIGGER_USE_REG1_TO_REG4	70	仅允许将计数器 1 - 4 用于触发
RXFL_TRIGGER_USE_REG5_TO_REG8	71	仅允许将计数器 5 - 8 用于触发
RXFL_SYSTEM_STATE_NOT_SUPPORTED_FOR_THIS_FILTER	72	不支持系统状态
RXFL_DEPENDENCY_IS_NOT_PERSISTENT	74	参考列过滤器不持续
RXFL_DEPENDENT_FILTER_SOFTWARE_FILTER_NOT_FIT	75	节点过滤器不能是软件过滤器的子节点，反之亦然
RXFL_OUTPUT_OR_INPUT_BUFFER_LENGTH_TOO_SMALL	76	输出缓冲区长度小于该操作所需的长度

C.1 证书生成

如何生成证书、公钥和 CA：

1.下载并安装最新版本的 OpenSSL（Windows 或 Linux）。

2.在安装路径 \bin 库中，找到 openssl.exe。

私钥

若要为证书创建新的私钥，请使用：

```
openssl genrsa -out privkey.pem 2048
```

注：

- 默认密钥大小为 2048，但用户可以使用任何所需的协议密钥大小（1024、2048、4096 等）。
- 文件名称可替换。
- 默认格式为 PEM，它是 ASCII 形式的。在许多系统中，因为二进制格式 DER 的尺寸更小，所以更受欢迎。若要在这些格式之间进行转换，请使用：*openssl rsa -in privkey.pem -inform PEM -out privkey.der -outform DER*

证书和 CA

CA（证书颁发机构）是自签署证书，用于签署其他证书。

若要生成 CA，请使用以下命令并插入所需的值：

```
openssl req -new -x509 -days 3650 -key privkey.pem -out root-ca.pem
```

关于该示例的几条注释：

- days 参数确定了证书的有效期。
- 在本文档的“私钥”部分以 PEM 格式生成密钥。
- 输出为 PEM 格式。若要从 PEM 转换为 DER，请使用：
  - ```
    openssl x509 -in input.crt -inform PEM -out output.crt
    -outform DER
    ```

若要生成证书，请先准备证书文件。与制作 CA 类似，使用以下命令填写所需的值，例如国家/地区代码名称等：

```
openssl req -new -key privkey.pem -out cert.pem
```

证书私钥与 CA 使用的私钥不同。每个证书应有自己的私钥。

生成证书表单（也称证书请求）之后，使用其他证书对其进行签署。该表单通常使用 CA 进行签署，但要创建证书链，请使用另一个证书对其进行签署。
```

若要执行签署过程，请使用：

```
openssl x509 -req -days 730 -in cert.pem -CA ca.pem -CAkey CAPrivate.pem              -set_serial 01
-out cert.pem
```

关于该示例的几条注释：

- 使用 CA。根据需要使用任何证书对生成的证书进行签署。
- 此处使用的密钥是 CA 私钥。
- “days”参数用于确定此证书的有效期，需要使用 -set_serial 01 作为默认值。

总之，如果用户想要生成 CA，然后由 CA 签署证书，请执行以下操作：

- 为 CA 生成私钥。
- 为证书生成私钥
- 使用 CA 私钥制作 CA
- 使用证书私钥生成证书请求
- 使用 CA 和 CA 私钥签署证书
- 如果用户想要创建证书链，请再生成一个私钥和证书请求，并使用其他证书对其进行签署

如何生成 sha1 并使用私钥对其进行签署

```
openssl dgst -sha1 data.txt > hash
```

若要从 data.txt 文件生成 sha1 代码，请使用：

若要使用私钥对这个 sha1 代码进行 RSA 签署，请使用：

```
openssl dgst -binary -out signature.bin -sha1 -sign privatekey.pem BufferToSign.bin
```

注：以前版本的页码可能与当前版本的页码不同

Changes from Revision B (May 2018) to Revision C (January 2021)	Page
• 更新了整个文档中的表格、图和交叉参考的编号格式。	7
• 添加了新的 节 3.2。	21
• 对节 3.2.1 进行了更新。	22
• 添加了新的 节 15.11。	131
• 添加了 章节 19。	163

This page intentionally left blank.

重要声明和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、某特定用途方面的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他功能安全、信息安全、监管或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的应用。严禁对这些资源进行其他复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。您应全额赔偿因在这些资源的使用中对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，TI 对此概不负责。

TI 提供的产品受 TI 的销售条款或 ti.com 上其他适用条款/TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

邮寄地址：Texas Instruments, Post Office Box 655303, Dallas, Texas 75265