# Contents

1 Overview .................................................................................................................. 9
   1.1 Document Scope ................................................................................................. 9
   1.2 Overview ............................................................................................................. 9
      1.2.1 Features List ............................................................................................... 9
   1.3 Host Driver Overview ......................................................................................... 11
   1.4 Configurable Information Element .................................................................... 13
2 Writing a Simple Networking Application .............................................................. 14
   2.1 Overview ........................................................................................................... 15
      2.1.1 Basic Example Code ................................................................................. 16
3 Device Initialization ................................................................................................. 20
   3.1 Overview ........................................................................................................... 21
4 Device Configurations ............................................................................................ 23
   4.1 Overview ........................................................................................................... 24
   4.2 Device Parameters ............................................................................................ 24
   4.3 WLAN Parameters ............................................................................................ 24
      4.3.1 Advanced ................................................................................................... 25
   4.4 Network Parameters ......................................................................................... 26
   4.5 Internet and Networking Services Parameters .................................................. 26
   4.6 Power-Management Parameters ....................................................................... 26
      4.6.1 Overview ................................................................................................... 26
      4.6.2 Power Policy ............................................................................................... 27
   4.7 Scan Parameters ................................................................................................ 27
      4.7.1 Scan Policy ................................................................................................ 27
5 WLAN Connection ................................................................................................... 28
   5.1 Manual Connection ............................................................................................ 29
      5.1.1 STA ............................................................................................................. 29
      5.1.2 P2P ............................................................................................................. 29
   5.2 Connection Using Profiles .................................................................................. 29
   5.3 Connection Policies ............................................................................................ 29
   5.4 Connection-Related Async Events ..................................................................... 30
      5.4.1 WLAN Events ........................................................................................... 30
      5.4.2 Network Events ......................................................................................... 30
      5.4.3 Events for Different Connection Scenarios ................................................ 30
   5.5 WLAN Connection Using BSSID ....................................................................... 30
6 Socket ....................................................................................................................... 31
   6.1 Overview ........................................................................................................... 32
      6.1.1 Transmission Control Protocol (TCP) ......................................................... 32
      6.1.2 User Datagram Protocol (UDP) .................................................................. 32
   6.2 Socket Connection Flow ..................................................................................... 32
   6.3 TCP Connection Flow ........................................................................................ 33
      6.3.1 Client Side .................................................................................................. 33
      6.3.2 Server Side ................................................................................................. 34
   6.4 UDP Connection Flow ....................................................................................... 35
      6.4.1 Client Side .................................................................................................. 35
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>19.3</td>
<td>Netapp Events</td>
<td>149</td>
</tr>
<tr>
<td>19.4</td>
<td>Socket Events</td>
<td>150</td>
</tr>
<tr>
<td>19.5</td>
<td>Device Events</td>
<td>150</td>
</tr>
<tr>
<td>20</td>
<td>Configurable Info Element</td>
<td>152</td>
</tr>
<tr>
<td>20.1</td>
<td>General</td>
<td>152</td>
</tr>
<tr>
<td>20.2</td>
<td>Interface to Application</td>
<td>153</td>
</tr>
<tr>
<td>20.2.1</td>
<td>API Output</td>
<td>155</td>
</tr>
<tr>
<td>20.3</td>
<td>Total Maximum Size of all Information Elements</td>
<td>155</td>
</tr>
<tr>
<td>21</td>
<td>References</td>
<td>157</td>
</tr>
<tr>
<td>A</td>
<td>Host Driver Architecture</td>
<td>158</td>
</tr>
<tr>
<td>A.1</td>
<td>Overview</td>
<td>158</td>
</tr>
<tr>
<td>A.1.1</td>
<td>SimpleLink WiFi Host Driver — Platform-Independent Part</td>
<td>159</td>
</tr>
<tr>
<td>A.1.2</td>
<td>SimpleLink WiFi Host Driver – Platform-Dependent Part</td>
<td>159</td>
</tr>
<tr>
<td>A.1.3</td>
<td>SimpleLink WiFi Driver Configuration</td>
<td>159</td>
</tr>
<tr>
<td>A.1.4</td>
<td>User Application</td>
<td>159</td>
</tr>
<tr>
<td>A.2</td>
<td>Driver Data Flows</td>
<td>159</td>
</tr>
<tr>
<td>A.2.1</td>
<td>Transport Layer Protocol</td>
<td>159</td>
</tr>
<tr>
<td>A.2.2</td>
<td>Command and Command Complete</td>
<td>160</td>
</tr>
<tr>
<td>A.2.3</td>
<td>Data Transactions</td>
<td>160</td>
</tr>
<tr>
<td>B</td>
<td>Error Codes</td>
<td>162</td>
</tr>
<tr>
<td>B.1</td>
<td>Error Codes</td>
<td>162</td>
</tr>
<tr>
<td>C</td>
<td>How to Generate Certificates, Public Keys, and CAs</td>
<td>170</td>
</tr>
<tr>
<td>C.1</td>
<td>Certificate Generation</td>
<td>170</td>
</tr>
<tr>
<td></td>
<td>Revision History</td>
<td>172</td>
</tr>
</tbody>
</table>
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>Host Driver Architecture</td>
<td>12</td>
</tr>
<tr>
<td>1-2</td>
<td>Host Driver Anatomy</td>
<td>13</td>
</tr>
<tr>
<td>2-1</td>
<td>Basic Networking Application State Machine</td>
<td>16</td>
</tr>
<tr>
<td>3-1</td>
<td>Basic Initialization Flow</td>
<td>21</td>
</tr>
<tr>
<td>4-1</td>
<td>TX o/p vs TX Level</td>
<td>26</td>
</tr>
<tr>
<td>5-1</td>
<td>Socket Connection Flow</td>
<td>33</td>
</tr>
<tr>
<td>8-1</td>
<td>AP Mode Connect</td>
<td>44</td>
</tr>
<tr>
<td>8-2</td>
<td>Profiles</td>
<td>44</td>
</tr>
<tr>
<td>8-3</td>
<td>Device Config Tab</td>
<td>45</td>
</tr>
<tr>
<td>9-1</td>
<td>WLAN Connect Command</td>
<td>51</td>
</tr>
<tr>
<td>12-1</td>
<td>HTTP GET Request</td>
<td>78</td>
</tr>
<tr>
<td>12-2</td>
<td>High Level Block Diagram</td>
<td>80</td>
</tr>
<tr>
<td>13-1</td>
<td>mDNS Get Service Sequence</td>
<td>101</td>
</tr>
<tr>
<td>13-2</td>
<td>Find Full Service After Query</td>
<td>102</td>
</tr>
<tr>
<td>15-1</td>
<td>Trees Example 1</td>
<td>120</td>
</tr>
<tr>
<td>15-2</td>
<td>Trees Example 2</td>
<td>121</td>
</tr>
<tr>
<td>16-1</td>
<td>802.11 Frame Structure</td>
<td>125</td>
</tr>
<tr>
<td>16-2</td>
<td>Sniffer</td>
<td>128</td>
</tr>
<tr>
<td>16-3</td>
<td>Sniffer</td>
<td>129</td>
</tr>
<tr>
<td>16-4</td>
<td>Tx Continues</td>
<td>130</td>
</tr>
<tr>
<td>18-1</td>
<td>Host Driver API Silos</td>
<td>135</td>
</tr>
<tr>
<td>20-1</td>
<td>802.11 Spec - Info Element</td>
<td>152</td>
</tr>
<tr>
<td>20-2</td>
<td>Information Elements With Same ID and OUI</td>
<td>153</td>
</tr>
<tr>
<td>A-1</td>
<td>SimpleLink WiFi Host Driver Configuration</td>
<td>158</td>
</tr>
<tr>
<td>A-2</td>
<td>Blocked Link</td>
<td>160</td>
</tr>
<tr>
<td>A-3</td>
<td>Data Flow Control</td>
<td>160</td>
</tr>
</tbody>
</table>
## List of Tables

1-1. Features List .......................................................................................................................... 9
1-2. Acronyms Used in This Document ....................................................................................... 13
6-1. SimpleLink Supported Socket API ...................................................................................... 37
8-1. Provisioning Methods ........................................................................................................... 47
9-1. Supported Cryptographic Algorithms .................................................................................... 54
9-2. STA Mode ............................................................................................................................... 54
9-3. AP Mode ................................................................................................................................. 55
10-1. WLAN Parameters ............................................................................................................... 58
10-2. Event Parameters ................................................................................................................. 61
10-3. Event Parameters .................................................................................................................. 62
10-4. Event Parameters .................................................................................................................. 62
12-1. Enable or Disable HTTP Server ......................................................................................... 86
12-2. Configure HTTP Port Number ............................................................................................ 87
12-3. Enable or Disable Authentication Check ............................................................................ 87
12-4. Set or Get Authentication Name .......................................................................................... 88
12-5. Set or Get Authentication Password ................................................................................... 88
12-6. Set or Get Authentication Realm ....................................................................................... 88
12-7. Set or Get Domain Name ...................................................................................................... 89
12-8. Set or Get URN Name .......................................................................................................... 89
12-9. Enable or Disable ROM Web Pages Access ...................................................................... 90
12-10. System Information .......................................................................................................... 90
12-11. Version Information ........................................................................................................... 91
12-12. Network Information ........................................................................................................ 91
12-13. Tools .................................................................................................................................. 92
12-14. Connection Policy Status .................................................................................................. 92
12-15. Display Profiles Information ............................................................................................ 92
12-16. P2P Information ............................................................................................................... 93
12-17. System Configurations ....................................................................................................... 94
12-18. Network Configurations .................................................................................................... 95
12-19. Connection Policy Configuration ..................................................................................... 95
12-20. Profiles Configuration ...................................................................................................... 96
12-21. Tools .................................................................................................................................. 96
12-22. P2P Configuration ............................................................................................................. 97
12-23. POST Actions ................................................................................................................... 97
14-1. Summary of Instructions ..................................................................................................... 109
14-2. Parameters ........................................................................................................................ 110
16-1. Network Layers .................................................................................................................. 125
20-1. API Input ............................................................................................................................. 154
20-2. Beacon & Probe Response Parameters .............................................................................. 155
B-1. General Error Codes ............................................................................................................. 162
B-2. Device Error Codes ............................................................................................................. 162
B-3. Socket Error Codes ............................................................................................................. 163
B-4. WLAN Error Codes ........................................................................................................... 165
B-5. NetApp Error Code ............................................................................................................. 166
B-6. FS Error Codes ................................................................................................................... 167
B-7. Rx Filter Error Codes .......................................................................................................... 168
1.1 Document Scope

The purpose of this document is to provide software programmers working with the Wi-Fi® subsystem all the required knowledge on its networking capabilities and how to use them, through the host driver. This includes an overview on how to write a networking application, a detailed description of the entire device’s API networking operation modes and features, and a review of each of the driver APIs, accompanied with source code examples for each topic.

1.2 Overview

The SimpleLink™ Wi-Fi® CC3100 and CC3200 are the next generation in embedded Wi-Fi. The CC3100 Internet-on-a-chip™ can add Wi-Fi and internet to any microcontroller (MCU), such as TI's ultra-low power MSP430™. The CC3200 is a programmable Wi-Fi MCU that enables true, integrated Internet-of-things (IoT) development. The Wi-Fi network processor subsystem in both SimpleLink Wi-Fi devices integrates all protocols for Wi-Fi and Internet, greatly minimizing MCU software requirements. With built-in security protocols, SimpleLink Wi-Fi provides a simple yet robust security experience.

1.2.1 Features List

<table>
<thead>
<tr>
<th>Wi-Fi</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported channels</td>
<td>1-13</td>
</tr>
<tr>
<td>Supported channels</td>
<td>WEP, WPA and WPA2</td>
</tr>
<tr>
<td>Personal security</td>
<td>WPA-2 Enterprise</td>
</tr>
<tr>
<td>Enterprise security</td>
<td>EAP Fast, EAP PEAPv0 MSCHAPv2, EAP PEAPv0 TLS, EAP PEAPv1 TLS, EAP TLS EAP TTLS TLS, EAP TTLS MSCHAPv2</td>
</tr>
<tr>
<td>Provisioning</td>
<td>SmartConfig™ technology</td>
</tr>
<tr>
<td>Standards</td>
<td>Wi-Fi protected setup (WPS2)</td>
</tr>
<tr>
<td>Standards</td>
<td>Access point mode with internal HTTP web server</td>
</tr>
<tr>
<td>Clients</td>
<td>802.11b/g access point and Wi-Fi direct group owner</td>
</tr>
<tr>
<td>Clients</td>
<td>1</td>
</tr>
<tr>
<td>Personal security</td>
<td>WEP, WPA, and WPA2</td>
</tr>
<tr>
<td>Networking</td>
<td>IPv4</td>
</tr>
<tr>
<td>Transport</td>
<td>UDP, UDP, RAW, ICMP</td>
</tr>
<tr>
<td>Cross-layer</td>
<td>DHCP, ARP, DNS</td>
</tr>
</tbody>
</table>
## Table 1-1. Features List (continued)

<table>
<thead>
<tr>
<th>Wi-Fi</th>
</tr>
</thead>
</table>
| Application | mDNS  
  DNS-SD  
  HTTP 1.0 web server |
| Transport layer security | 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 |
| User application sockets | Up to 8 open sockets  
  Up to 2 secured application sockets:  
  • One server (listen socket and accept socket) + client (data socket)  
  • Up to two clients (data socket) |
| Advanced Features |
| 802.11 Transceiver | Transmit and receive raw Wi-Fi packets with full control over payload. Wi-Fi disconnect mode.  
  Can be used for general-purpose applications (tags, sniffer, RF tests) |
| Interfaces |
| SPI | Standard SPI up to 20 MHz on production device and up to 14 MHz on pre-production device |
| UART | 4-wire UART up to 3 MHz |
| Power Modes |
| Low-power mode | Uses 802.11 power save and device deep sleep power with three user-configurable policies |
| Configurable power policies | • Normal (default) - Best tradeoff between traffic delivery time and power performance  
  • Low power – Used only for transceiver mode application (disconnect mode)  
  • Long sleep interval – wakes up for the next DTIM after a configurable sleep interval, up to 2 seconds. This policy is only applicable for client socket mode. |
1.3 Host Driver Overview

The SimpleLink Wi-Fi Internet-on-a-chip devices provide comprehensive networking functionality. To simplify the integration and development of networking applications using the SimpleLink Wi-Fi devices, TI provides a simple and user-friendly host driver.

The SimpleLink host driver is responsible for:
- Providing a simple API to the user application
- Handling the communication with the device, including:
  - Building and parsing commands
  - Handling asynchronous events
  - Handling the flow control for the data path
  - Serialization of concurrent commands
- Working with the existing UART or SPI physical interface drivers
- Working with an OS adaption layer, providing flexibility in working with or without an OS
- Enable porting to any platform

The host driver is written in strict ANSI C89 for full compatibility with most embedded platforms and development environments.

The host driver key architecture concepts are:
- Microcontroller
  - Can run on 8-bit, 16-bit, or 32-bit microcontrollers
  - Can run on any clock speed – no performance or time dependency
  - Supports both big and little endian formats
  - Small memory footprint – configurable at the time of compiling, the driver requires as low as TBD KB of code memory and TBD B of RAM memory.

Standard interface communication port:
- SPI – Supports standard 4-wire SPI:
  - 8-, 16-, or 32-bit word length
  - Default mode 0 (CPOL=0, CPHA=0)
  - SPI clock can be configured up to 20 Mbps.
  - CS is required.
  - Additional IRQ line is required for async operations.
- UART
  - Standard UART with hardware flow control (RTS/CTS) up to 3 Mbps.
  - The default baud rate is 115200 (8 bits, no parity, 1 start/stop bit).
- Supporting systems using or not using OS:
  - Simple OS wrapper, requiring only two object wrappers:
    - Sync Obj (event-binary semaphore)
    - Lock Obj (mutex-binary semaphore)
  - Built-in logic within the driver for system not running OS
The SimpleLink host driver includes a set of six logical and simple API modules:

- **Device API** – Manages hardware-related functionality such as start, stop, set, and get device configurations.
- **WLAN API** – Manages WLAN, 802.11 protocol-related functionality such as device mode (station, AP, or P2P), setting provisioning method, adding connection profiles, and setting connection policy.
- **Socket API** – The most common API set for user applications, and adheres to BSD socket APIs.
- **NetApp API** – Enables different networking services including the Hypertext Transfer Protocol (HTTP) server service, DHCP server service, and MDNS client/server service.
- **NetCfg API** – Configures different networking parameters, such as setting the MAC address, acquiring the IP address by DHCP, and setting the static IP address.
- **File System API** – Provides access to the serial flash component for read and write operations of networking or user proprietary data.
Figure 1-2 shows the host driver anatomy.

Figure 1-2. Host Driver Anatomy

1.4 Configurable Information Element

Table 1-2. Acronyms Used in This Document

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description/Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>AP</td>
<td>WLAN access point</td>
</tr>
<tr>
<td>DEV</td>
<td>Device role</td>
</tr>
<tr>
<td>FW</td>
<td>Firmware</td>
</tr>
<tr>
<td>LAN</td>
<td>Local area network</td>
</tr>
<tr>
<td>STA</td>
<td>WLAN station</td>
</tr>
<tr>
<td>WLAN</td>
<td>Wireless LAN</td>
</tr>
<tr>
<td>Info element</td>
<td>Information element</td>
</tr>
<tr>
<td>IE</td>
<td>Information element</td>
</tr>
<tr>
<td>OUI</td>
<td>Organization unique ID</td>
</tr>
<tr>
<td>NWP</td>
<td>Network processor</td>
</tr>
<tr>
<td>P2P</td>
<td>Peer to peer</td>
</tr>
<tr>
<td>GO</td>
<td>Group owner</td>
</tr>
<tr>
<td>Topic</td>
<td>Page</td>
</tr>
<tr>
<td>------------</td>
<td>------</td>
</tr>
<tr>
<td>2.1 Overview</td>
<td>15</td>
</tr>
</tbody>
</table>
2.1 Overview

This chapter explains the software blocks needed to build a networking application. In addition, this chapter describes the recommended flow for most applications. The information provided is for guidance only. Programmers have complete flexibility on how to use the various software blocks. Programs using the SimpleLink device consist of the following software blocks:

- **Wi-Fi subsystem initialization** – Wakes the Wi-Fi subsystem from the hibernate state.
- **Configuration** – Refers to init time configuration that occurs infrequently, such as when changing the Wi-Fi subsystem from a WLAN STA to WLAN soft AP, changing the MAC address, and so forth.
- **WLAN connection** – The physical interface must be established. There are numerous ways to do so; the simplest way is to manually connect to an AP as a wireless station.
  - **DHCP** – Although not an integral part of the WLAN connection, the user must wait for the receiving IP address before continuing to the next step of working with TCP and UDP sockets.
- **Socket connection** – At this point, the application must set up the TCP/IP layer. Separate this phase into the following parts:
  - **Creating the socket** – Choose TCP, UDP, or RAW sockets, whether to use a client or a server socket, defining socket characteristics such as blocking or non-blocking, socket timeouts, and so forth.
  - **Querying for the server IP address** – In most occasions, when implementing a client side communication, the remote server side IP address is unknown, which is required for establishing the socket connection. This can be done by using DNS protocol to query the server IP address by using the server name.
  - **Creating socket connection** – When using the TCP socket, a proper socket connection must be established before performing a data transaction.
- **Data transactions** – Once the socket connection is established, transmit data both ways between the client and the server, by implementing the application logic.
- **Socket disconnection** – Upon finishing the required data transactions, TI recommends performing a graceful closure of the socket communication channel.
- **Wi-Fi subsystem hibernate** – When not working with the Wi-Fi subsystem for a long period of time, TI recommends putting it into hibernate mode.
2.1.1 Basic Example Code

When implementing a networking application, consider the different application blocks, the host driver software concepts described above, and system aspects such as hardware and operating system.

Figure 2-1 shows a state machine diagram that describes the basic software design.

Figure 2-1. Basic Networking Application State Machine

Figure 2-1 shows the different states described in this chapter, the host driver events that trigger the code to move between different states, and basic error-handling events.

An example of the state machine is implemented in the following code:

- **Init state** – The following is an example of initializing the Wi-Fi subsystem as a WLAN station:

```c
    case INIT:
        status = sl_Start(0, 0, 0);
        if (status == ROLE_STA)
        {
            g_State = CONFIG;
        }
        else
        {
            g_State = SIMPLERLINK_ERR;
        }
        break;
```
• **WLAN connection** – The following is an example of WLAN and network event handlers, demonstrating the WLAN connection, waiting for a successful connection, and acquiring an IP address:

```c
/* SimpleLink WLAN event handler */
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 Networking event handler */
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;
    }
}

/* initiating the WLAN connection */
case WLAN_CONNECTION:
    status = sl_WlanConnect(User.SSID,strlen(User.SSID),0,
                             &secParams, 0);
    if (status == 0)
    {
        g_State = WLAN_CONNECTING;
    }
    else
    {
        g_State = SIMPLERLINK_ERR;
    }

/* waiting for SL_WLAN_CONNECT_EVENT to notify on a successful connection */
case WLAN_CONNECTING:
    if (g_Event & EVENT_CONNECTED)
    {
        printf("Connected to %s\n", g_AP_Name);
    }
```

Writing a Simple Networking Application

```c
{g_State = WLAN_CONNECTED;
 }
break;

/* waiting for SL_NETAPP_IPV4_ACQUIRED to notify on a receiving an IP address */
case WLAN_CONNECTED:
    if (g_Event & EVENT_IP_ACQUIRED)
        printf("Received IP address:%d.%d.%d.%d
", (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;

• Socket connection — The following is an example of querying for the remote server IP address by using the server name, creating a TCP socket, and connecting to the remote server socket:

    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);
        /* Change the DestinationIP endianity, to big endian */
        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;
             }
        else
            {printf("Error connecting to socket\n\r");
             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;
```
• **Data transactions** – The following is an example of sending and receiving TCP data over the open socket:

```c
case SOCKET_CONNECTED:
    /* Send data to the remote server */
    sl_Send(appData.SockID, appData.SendBuff, strlen(appData.SendBuff), 0);

    /* Receive data from the remote server */
    sl_Recv(appData.SockID, &appData.Recvbuff[0], MAX_RECV_BUFF_SIZE, 0);
break;
```

• **Socket disconnection** – The following is an example of closing a socket:

```c
case SOCKET_DISCONNECT:
    sl_Close(appData.SockID);
    /* Reopening the socket */
    g_State = SOCKET_CONNECTION;
    break;
```

• **Device hibernate** – The following is an example of putting the Wi-Fi subsystem into hibernate state:

```c
case SIMPLELINK_HIBERNATE:
    sl_Stop();
    g_State = ...
    break;
```
# Device Initialization

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Overview</td>
<td>21</td>
</tr>
</tbody>
</table>

---

Copyright © 2014–2018, Texas Instruments Incorporated
3.1 Overview

The Wi-Fi subsystem is enabled by calling the `sl_Start()` API. During the initialization, the host driver performs the following key steps:

- Enables the bus interface (in CC3200 – SPI; in CC3100 – SPI or UART)
- Registers the asynchronous events handler
- Enables the Wi-Fi subsystem (in the CC3200, this is done by the internal applications microcontroller. In the CC3100, it is done by the host processor)
- Sends a synchronization message to the Wi-Fi subsystem and waits for an IRQ in return, signaling on completion of the initialization phase.

Figure 3-1 shows the basic initialization flow:

Figure 3-1. Basic Initialization Flow

The Wi-Fi subsystem initialization can take tens of mS to complete. The host driver supports two main options of initialization using "sl_Start(const void* pIfHdl, char* pDevName and const P_INIT_CALLBACK pInitCallback)" API:

- Blocking – pInitCallback must be set to NULL. The calling application is blocked until the entire initialization process completes (upon receiving the Init complete interrupt). See the following code example:

```c
if( sl_Start(NULL, NULL, NULL) == 0)
{
    LOG("Error opening interface to device\n");
}
```
Asynchronous – pInitCallBack is given a pointer to a function that is called when the initialization process completes. In this case, the call to `sl_Start()` returns immediately. See the following code example:

```c
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));
    ...
}
```
### Device Configurations

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 Overview</td>
<td>24</td>
</tr>
<tr>
<td>4.2 Device Parameters</td>
<td>24</td>
</tr>
<tr>
<td>4.3 WLAN Parameters</td>
<td>24</td>
</tr>
<tr>
<td>4.4 Network Parameters</td>
<td>26</td>
</tr>
<tr>
<td>4.5 Internet and Networking Services Parameters</td>
<td>26</td>
</tr>
<tr>
<td>4.6 Power-Management Parameters</td>
<td>26</td>
</tr>
<tr>
<td>4.7 Scan Parameters</td>
<td>27</td>
</tr>
</tbody>
</table>
4.1 Overview

This chapter covers all user-configurable parameters, but also read-only parameters used for reflecting the device and networking status (read-only parameters are appropriately noted in this document).

The Wi-Fi subsystem has several configurable parameters that control its behavior. The host driver uses different APIs to configure these parameters. The parameters are grouped based on their functionality.

Most of the parameters described in this chapter are stored in the serial flash (parameters that are volatile and not stored in the serial flash are noted appropriately in this document).

In addition, there are default values to each of the parameters. If they are not set, the internet Wi-Fi subsystem uses the default value. A value stored in the serial flash always gets priority on top of a the default value.

Usually, an application must only configure its parameters once, out of cold boot, or when a specific configuration change is required.

NOTE: All of the parameters that are sorted in the serial flash will take effect in the next device boot.

4.2 Device Parameters

Time and date: Configures the internal date and time of the device. For more details, see Section 18.1.

Firmware version: A read-only parameter that returns the Wi-Fi subsystem firmware version. For more details, see Section 18.1.

Device status: Return status for the last events that accord in the internet subsystem. For more details, see Section 18.1.

Asynchronous events mask: Masked events do not generate asynchronous messages (IRQs) from the Wi-Fi subsystem. For more details, see `sl_EventMaskGet` and `sl_EventMaskSet` in the Section 18.1.

UART configuration: When using the UART interface, the application can set several UART parameters: baud rate, flow control, and COM port. For more details, see Section 18.1.

NOTE:

- Partly volatile parameter – parameter is retained in hibernate mode but resets in shutdown
- Read-only parameter

4.3 WLAN Parameters

Device Mode – The Wi-Fi subsystem can operate in several WLAN roles. The different options are:

- WLAN station
- WLAN AP
- WLAN P2P

For more details, see Section 18.3.

AP mode – If set to an access-point role, the Wi-Fi subsystem has numerous specific configurations for AP mode that can be set:

- SSID – AP name
- Country code
- Beacon interval
- Operational channel
- Hidden SSID – Enable or disable
- DTIM period
• Security type – Possible options are:
  – Open security
  – WEP security
  – WPA security
• Security password:
  – For WPA: 8 to 63 characters
  – For WEP: 5 to 13 characters (ASCII)
• WPS state
For more details, see Section 8.4.

P2P – If set to peer-to-peer role, the Wi-Fi subsystem has numerous specific configurations for P2P mode that can be set:
• Device Name
• Device type
• Operational channels – Listen channel and regulatory class determine the device listen channel during p2p find listen phase.
  Operational channel and regulatory class determine the operating channel preferred by this device (if it is group owner, this will be the operating channel). Channels should be one of the social channels (1/6/11). If no listen or operational channel is selected, a random 1/6/11 will be selected.
• Information elements – The application can be set to MAX_PRIVATE_INFO_ELEMENTS_SUPPORTED information elements per role (AP / P2P GO). To delete an information element, use the relevant index and length = 0.
  The application can be set to MAX_PRIVATE_INFO_ELEMENTS_SUPPORTED to the same role. However, for AP no more than INFO_ELEMENT_MAX_TOTAL_LENGTH_AP bytes can be stored for all information elements.
  For P2P GO, no more than INFO_ELEMENT_MAX_TOTAL_LENGTH_P2P_GO bytes can be stored for all information elements.
• Scan channels – Changes the scan channels and RSSI threshold.
For more details, see Chapter 11.

4.3.1 Advanced

Country code – Sets the Wi-Fi subsystem regulatory domain country code. Relevant for WLAN station and P2P client modes only. The supported country codes are: US, JP, and EU. For more details, see Section 18.3.

TX power – Sets the maximal transmit power of the Wi-Fi subsystem. For more details, see Section 18.3.

The relation between the TX power level to the actual out power in dBm can be seen in Figure 4-1
4.4 Network Parameters

MAC address – Sets the MAC address of the device. For more details, see Section 18.2.

IP address – Configures the Wi-Fi subsystem to use DHCP or static IP configuration. In case of static configuration, the user can set the IP address, DNS address, GW address, and subnet mask. For more details, see Section 18.2.

4.5 Internet and Networking Services Parameters

HTTP Server: For more details, see Chapter 12.

DHCP Server: For more details, see Section 10.6.

mDNS: For more details, see Chapter 13.

SmartConfig: For more details, see Section 8.2.

4.6 Power-Management Parameters

4.6.1 Overview

Power management and energy preservation are among the most cumbersome issues and of the primary focus areas for online systems. Handling power regimes effectively is fundamental for any power-aware solution. This problem becomes even more challenging in cases where a certain component of the overall solution is more power hungry than the rest of the system, such as with many embedded Wi-Fi capable systems.
4.6.2 Power Policy

From host application perspective, only two modes of operation are explicitly selected by the host: hibernate or enabled.

The Wi-Fi subsystem is equipped with a policy management entity that allows a developer (host application programmer) to guide the behavior of the power management algorithm through pre-defined power policies. The sl_PolicySet API is used to configure the device power management policy.

The available policies are:

- Normal (default) – Features the best tradeoff between traffic delivery time and power performance. For setting normal power-management policy use:
  
  ```
  sl_WlanPolicySet(SL_POLICY_PM, SL_NORMAL_POLICY, NULL, 0)
  ```

- Always on – The Wi-Fi subsystem is kept fully active at all times, providing the best WLAN traffic performance. This policy is user-directed; the user may provide the target latency figure. For setting always-on power-management policy use:
  
  ```
  sl_WlanPolicySet(SL_POLICY_PM, SL_ALWAYS_ON_POLICY, NULL, 0)
  ```

- Long sleep interval – This low-power mode comes with a desired maximum sleep time parameter. The parameter reflects the desired sleep interval between two consecutive wakeups for beacon reception. The Wi-Fi module computes the desired time and wakes up to the next DTIM that does not exceed the specified time. The maximum allowed desired maximum sleep time parameter is 2 seconds. For setting the low-latency power management policy, use:
  
  ```
  sl_WlanPolicySet(SL_POLICY_PM, SL_LOW_LATENCY_POLICY, NULL, 0)
  ```

  **NOTE:** This policy only works in client mode. It automatically terminates mDNS and the internal HTTP server running on the device. TCP and UDP servers initiated by the user application will lead to unpredictable system behavior and performance.

- Low power – Device power management algorithm is more opportunistic exploiting opportunities to lower its power mode. Tradeoff tends toward power conservation performance (sensor application).

  ```
  sl_WlanPolicySet(SL_POLICY_PM, SL_LOW_LATENCY_POLICY, NULL, 0)
  ```

  **NOTE:** Low-power mode is only supported when the Wi-Fi subsystem is not connected to an AP. Therefore, it is mostly relevant to the transceiver mode.

4.7 Scan Parameters

4.7.1 Scan Policy

SL_POLICY_SCAN defines a system scan time interval if there is no connection. The default interval is 10 minutes. After the scan interval is set, an immediate scan is activated. The next scan is based on the scan interval settings.

- For example, use the following to set the scan interval to a 1-minute interval:
  
  ```
  unsigned long intervalInSeconds = 60;
  #define SL_SCAN_ENABLE 1
  sl_WlanPolicySet(SL_POLICY_SCAN, SL_SCAN_ENABLE, (unsigned char *) &intervalInSeconds, sizeof(intervalInSeconds));
  ```

- For example, to disable scan:

  ```
  #define SL_SCAN_DISABLE 0
  sl_WlanPolicySet(SL_POLICY_SCAN, SL_SCAN_DISABLE, 0, 0);
  ```
Connecting to a WLAN network is the first step in initiating a socket communication. The Wi-Fi subsystem supports two ways of establishing a WLAN connection:

1. Manual connection – The application calls an API that triggers the connection process.
2. Connection using profiles – The Wi-Fi subsystem automatically connects to pre-defined connection profiles.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1 Manual Connection</td>
<td>29</td>
</tr>
<tr>
<td>5.2 Connection Using Profiles</td>
<td>29</td>
</tr>
<tr>
<td>5.3 Connection Policies</td>
<td>29</td>
</tr>
<tr>
<td>5.4 Connection-Related Async Events</td>
<td>30</td>
</tr>
<tr>
<td>5.5 WLAN Connection Using BSSID</td>
<td>30</td>
</tr>
</tbody>
</table>
5.1 Manual Connection

5.1.1 STA

For a manual connection, the user application must implement the following steps:

1. Call to the `sl_WlanConnect` API. This API call accepts the SSID of the AP, the security type, and key, if applicable.
2. Implement a callback function to handle the asynchronous connection event (`SL_WLAN_CONNECT_EVENT`), signaling the completion of the connection process.

For additional information about these APIs, see Section 18.3 or the doxygen API manual.

5.1.2 P2P

For details, see Chapter 11.

5.2 Connection Using Profiles

A WLAN profile provides the information required to connect to a given AP. This includes the SSID, security type, and security keys. Each profile refers to a certain AP. The profiles are stored in the serial flash file system (nonvolatile memory), and preserved during device reset. The following APIs are available for handling profiles:

- `sl_WlanProfileAdd` – Adds a new profile. SSID and security information must be provided, where the returned value refers to the stored index (out of the seven available).
- `sl_WlanProfileDel` – Deletes a certain stored profile, or deletes all profiles at once. Index should be the input parameter.
- `sl_WlanProfileGet` – Retrieves information from a specific stored profile. Index should be the input parameter.

For additional information about these APIs, see Section 18.3 or the doxygen API manual.

```c
/* Delete all profiles (0xFF) stored */
sl_WlanProfileDel(0xFF);

/* Add unsecured AP profile with priority 0 (lowest) */
sl_WlanProfileAdd(SL_SEC_TYPE_OPEN, (unsigned char*)UNSEC_SSID_NAME, strlen(UNSEC_SSID_NAME), g_BSSID, 0, 0, 0, 0);

/* Add WPA2 secured AP profile with priority 1 (0 is lowest) */
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 Connection Policies

`SL_POLICY_CONNECTION` passes the type parameters to the `sl_WlanPolicySet` API to modify or set the connection policies arguments. For additional information about this API, see the doxygen API manual.

Download the latest SDK for the complete example code.

The WLAN connection policy defines five options to connect the SimpleLink device to a given AP. The five options of the connection policy are:

- **Auto** – The device tries to connect to an AP from the stored profiles based on priority. Up to seven profiles are supported. Upon a connection attempt, the device selects the highest priority profile. If several profiles are within the same priority, the decision is made based on security type (WPA2 → WPA → OPEN). If the security type is the same, the selection is based on the received signal strength. Use the following to set this option:

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

- **Fast** – The device tries to connect to the last connected AP. In this mode, the probe request is not transmitted before the authentication request, as both the SSID and channel are already known.
Use the following to set this option:
\[
\text{sl\_WlanPolicySet(SL\_POLICY\_CONNECTION,SL\_CONNECTION\_POLICY(0,1,0,0),NULL,0)}
\]
- **anyP2P** (relevant for P2P mode only) – The CC31xx device tries to automatically connect to the first P2P device available, supporting push-button only.
  Use the following to set this option:
  \[
  \text{sl\_WlanPolicySet(SL\_POLICY\_CONNECTION,SL\_CONNECTION\_POLICY(0,0,0,1),NULL,0)}
  \]
- **autoSmartConfig** – For auto SmartConfig upon restart (any command from the host ends this state).
  To set this option, use
  \[
  \text{sl\_WlanPolicySet(SL\_POLICY\_CONNECTION,SL\_CONNECTION\_POLICY(0,0,0,1),NULL,0)}
  \]

Use the following to set the long sleep interval policy:
\[
\text{unsigned short PolicyBuff[4] = \{0,0,800,0\}; // PolicyBuff[2] is max sleep time in mSec}
\text{sl\_WlanPolicySet(SL\_POLICY\_PM, SL\_LONG\_SLEEP\_INTERVAL\_POLICY, PolicyBuff,sizeof(PolicyBuff))}.
\]

5.4 Connection-Related Async Events

5.4.1 WLAN Events

This event handles async WLAN events.
\[
\text{sl\_WlanEvtHdlr} \text{ is an event handler for WLAN connection or disconnection indication.}
\]
Possible events are:
- **SL\_WLAN\_CONNECT\_EVENT** – Indicates WLAN is connected.
- **SL\_WLAN\_DISCONNECT\_EVENT** – Indicates WLAN is disconnected.

5.4.2 Network Events

This event handles networking events.

\[
\text{sl\_NetAppEvtHdlr} \text{ is an event handler for an IP address asynchronous event; usually accepted after a new WLAN connection.}
\]
Possible events are:
- **SL\_NETAPP\_IPV4\_ACQUIRED** – IP address was acquired (DHCP or static).

5.4.3 Events for Different Connection Scenarios

- If the device connects to the AP – The host processor will receive two events:
  **SL\_WLAN\_CONNECT\_EVENT** and **SL\_NETAPP\_IPV4\_ACQUIRED** (once the DHCP process is completed).
- If the device does not find an AP from the profiles configured – The SimpleLink device will continue scanning for Access points forever, no connection will be established and no events will be sent to the host processor.
- If the AP exists and the device has a profile for it, but it does not connect – If the SimpleLink device has a profile with the same SSID as the AP but deferent security key; the device will try to connect and will fail to connect. The host processor will receive a general event with error code of **SL\_ERROR\_CON\_MGMT\_STATUS\_DISCONNECT\_DURING\_CONNECT**.

5.5 WLAN Connection Using BSSID

SimpleLink WiFi device is not able to connect to an Access-point based on its’ BSSID only (manually or by using profiles). The BSSID parameter can be used to differentiate between two Access-points using the same SSID. Meaning, if the BSSID is entered the SimpleLink device will try to connect to the SSID+BSSID combination and not just based on SSID.
## Chapter 6

### Socket

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1 Overview</td>
<td>32</td>
</tr>
<tr>
<td>6.2 Socket Connection Flow</td>
<td>32</td>
</tr>
<tr>
<td>6.3 TCP Connection Flow</td>
<td>33</td>
</tr>
<tr>
<td>6.4 UDP Connection Flow</td>
<td>35</td>
</tr>
<tr>
<td>6.5 Socket Options</td>
<td>36</td>
</tr>
<tr>
<td>6.6 SimpleLink Supported Socket API</td>
<td>37</td>
</tr>
<tr>
<td>6.7 Number of Available Sockets</td>
<td>37</td>
</tr>
<tr>
<td>6.8 Packet Aggregation</td>
<td>38</td>
</tr>
</tbody>
</table>
Overview

6.1 Overview

The de facto networking API standard used in SimpleLink is BSD (Berkeley) sockets, upon which the Linux™, POSIX, and Windows™ sockets APIs are based. We try to be tied to Linux variant (as much as we can). The major differences are in error codes (return directly without errno) and additional setsockopt() options, which is a small subset of BSD Sockets API with some TI-specific socket option additions.

- See Simplelink documentation and examples.
- Berkeley sockets on Wikipedia

The content of this page assumes a basic understanding of Internet protocol suite and the differences between TCP and UDP connections. Here are some basic concepts:

6.1.1 Transmission Control Protocol (TCP)

A definition of TCP from Wikipedia follows:

The Transmission Control Protocol (TCP) is one of the core protocols of the Internet protocol suite (IP), and is so common that the entire suite is often called TCP/IP. TCP provides reliable, ordered, and error-checked delivery of a stream of octets between programs running on computers connected to a local area network, intranet, or the public Internet. It resides at the transport layer. Applications that do not require the reliability of a TCP connection may instead use the connectionless User Datagram Protocol (UDP), which emphasizes low-overhead operation and reduced latency rather than error checking and delivery validation.

6.1.2 User Datagram Protocol (UDP)

A definition of UDP from Wikipedia follows:

The User Datagram Protocol (UDP) is one of the core members of the Internet protocol suite (the set of network protocols used for the Internet). With UDP, computer applications can send messages, in this case referred to as datagrams, to other hosts on an Internet Protocol (IP) network without prior communications to set up special transmission channels or data paths. UDP is suitable for purposes where error checking and correction is either not necessary or performed in the application, avoiding the overhead of such processing at the network interface level. Time-sensitive applications often use UDP because dropping packets is preferable to waiting for delayed packets, which may not be an option in a real-time system. If error correction facilities are needed at the network interface level, an application may use the Transmission Control Protocol (TCP) or Stream Control Transmission Protocol (SCTP), which are designed for this purpose.

6.2 Socket Connection Flow

Figure 6-1 describes a general flow of TCP or UDP connection between a server and a client. The overall flow is almost identical to the Linux implementation. If you have previously been working on networking applications on Linux, this should be very straight forward.
6.3 TCP Connection Flow

The following program structure provides some basic ideas of how to use the SimpleLink API. For a full sample application code, see tcp_socket in the SDK examples.

6.3.1 Client Side

The first step is to create a socket. The returned socket handler is the most important element of the application. Networking does not work without the returned socket handler.

```c
int SockID;
SockID = sl_Socket(SL_AF_INET, SL_SOCK_STREAM, 0);
```
SL_AF_INET indicates using IPv4 and SL_SOCK_STREAM indicates using TCP. Definitions for both values are in the socket.h header file. The example sets 0 in the third parameter to select a default protocol from the selected domain and type. More details can be found in the online documentation. Definitions of some structures and constants are located in the socket.h header file inside the SDK.

As a TCP client, the application executes sl_Connect() to connect to a server. The server implementation can be found below.

```c
/* 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 specifies destination address and relevant information. Because struct type SlSockAddr is generic, use SlSockAddrIn_t to fill the details and cast it into SlSockAddr. Upon successful connection, the SockID socket handler is ready to perform data exchange.

The sl_Send() and sl_Recv() functions can be used for data exchange. Define the buffer size.

```c
#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);
```

Upon completion, close the socket with sl_Close() to allow the remaining applications to reuse the resource.

```c
sl_Close(SockID);
```

### 6.3.2 Server Side

Unlike a TCP client, a TCP server must establish several things before communication can occur.

- Similar to client implementation, create a TCP-based IPv4 socket.
  ```c
  SockID = sl_Socket(SL_AF_INET,SL_SOCK_STREAM, 0);
  ```

- In a TCP server implementation, the socket must perform Bind and Listen. Bind gives the server socket an address. Listen puts the socket in listening mode for an incoming client connection.
  ```c
  #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, (SlSockAddr_t *) &LocalAddr, sizeof(SlSockAddrIn_t));
  
  Status = sl_Listen(SockID, 0);
  ```

- With the socket now listening, accept any incoming connection request with sl_Accept(). There are two ways to perform this: blocking and nonblocking. This example uses the nonblocking mechanism with sl_SetSockOpt() and has the sl_Accept() placed in a loop to ensure it always retries connection regardless of each failure. Details about blocking and nonblocking can be found in Section 6.5.1.
  ```c
  
  Upon a successful connection, a new socket handler newSockID returns, which is then used for future
communication.

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 )
    {
        /* Wait for 1 ms */
        Delay(1);
    }
    else if( newSockID < 0 )
    {
        return -1;
    }
}

• Data exchange is exactly the same as implemented in client. The user may need to reverse the order; when one side is sending, the other side must be receiving.

#define BUF_SIZE 1400
char SendBuf[BUF_SIZE];

/* Write data to your buffer*/
<write buffer action>

Status = sl_Send(newSockID, SendBuf, BUF_SIZE, 0 );
char RecvBuf[BUF_SIZE];
Status = sl_Recv(newSockID, RecvBuf, BUF_SIZE, 0);

• At the end, close both sockets with sl_Close() to allow the remaining applications run to reuse the resource.

sl_Close(newSockID);
sl_Close(SockID);

6.4 UDP Connection Flow

The following program structure provides some basic ideas of how to use the SimpleLink API. For a full sample application code, see udp_socket in the SDK examples.

6.4.1 Client Side

Similar to the previous TCP example, create a IPv4-based socket. However, change the second parameter to SL_SOCKET_DGRAM, which indicates the socket will be used for a UDP connection.

SockID = sl_Socket(SL_AF_INET, SL_SOCK_DGRAM, 0);

Because UDP is a connectionless protocol, the client can start sending data to a specified target address without checking whether the target is alive or not.

#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));

Finally, close the socket.

sl_Close(SockID);
6.4.2 Server Side

The server side of the socket is identical to the client side.

SockID = sl_Socket(SL_AF_INET, SL_SOCK_DGRAM, 0);

Similar to TCP, bind the socket to the local address. No listening is required as UDP is connectionless.

#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);

The socket now tries to receive information on the socket. If the user did not specify the socket option as nonblocking, this command is blocked until an amount of BUF_SIZE of data is received. The fifth parameter specifies the source address sending the data.

#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);

Close the socket once communication is finished.

sl_Close(SockID);

6.5 Socket Options

6.5.1 Blocking vs NonBlocking

Depending on the implementation, choose to run the application with or without OS. Normally, when the application is run without OS, set the socket option to nonblocking with \texttt{sl\_SetSockOpt()} with the third parameter as SL\_SO\_NONBLOCKING. An OS-based application, however, has the option to perform multithreading and can handle blocking functions.

Status = sl_SetSockOpt(SockID, SL_SOCKET, SL\_SO\_NONBLOCKING, &nonBlockingValue, sizeof(nonBlockingValue)) ;

If the blocking mechanism is used, these functions block until execution is finished.

If the nonblocking mechanism is used, these functions return with an error code. The value of the error codes depends the function used. For details, see the online documentation.

\texttt{sl\_Connect()}, \texttt{sl\_Accept()}, \texttt{sl\_Aend()}, \texttt{sl\_Aendto()}, \texttt{sl\_Recv()}, and \texttt{sl\_Recvfrom()} are affected by this flag. If not set, the default is blocking.

An example with \texttt{sl\_Connect()} on a client application:

- Blocking: \texttt{sl\_Connect()} blocks until it is connecting to a server, or until an error occurs. Do not use blocking if the application is single-threaded and needs to perform other tasks (such as handling multiple sockets to read/write). However, using blocking is fine if the application is OS-based (like FreeRTOS).

\texttt{Status = sl\_Connect(SockID, ( SlSockAddr_t * ) &Addr, AddrSize);}

- Non-Blocking: \texttt{sl\_Connect()} returns immediately, regardless of connection. If connection is successful, a value of 0 returns. If not, the function returns SL\_EALREADY under normal conditions. TI recommends that the function is called in a loop so that the function keeps retrying the connection until the user decides to quit. The advantage of a non-blocking mechanism is to prevent the application from getting stuck in one place forever. This is particularly useful if your application needs to perform other tasks (such as blinking LED, reading sensor data, or having other connections simultaneously) at the same time.
while( Status < 0 )
{
    Status = sl_Connect(SockID, ( SlSockAddr_t *)&Addr, AddrSize);
    if( Status == SL_EALREADY )
    {
        /* Wait for 1 ms before the next retry */
        Delay(1);
    }
    else if( Status < 0 )
    {
        return -1; //Error
    }
    /* Perform other tasks before we retry the connection */
}

6.5.2 Secure Sockets
For more information, see Section 9.2.

6.6 SimpleLink Supported Socket API
Table 6-1 describes a list of supported BSD sockets and the corresponding SimpleLink implementation.

<table>
<thead>
<tr>
<th>BSD Socket</th>
<th>Simplelink Implementation</th>
<th>Server/Client Side</th>
<th>TCP/UDP</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>socket()</td>
<td>sl_Socket()</td>
<td>Both</td>
<td>Both</td>
<td>Creates an endpoint for communication</td>
</tr>
<tr>
<td>bind()</td>
<td>sl_Bind()</td>
<td>Server</td>
<td>Both</td>
<td>Assigns a socket to an address</td>
</tr>
<tr>
<td>listen()</td>
<td>sl_Listen()</td>
<td>Server</td>
<td>Both</td>
<td>Listens for connections on a socket</td>
</tr>
<tr>
<td>connect()</td>
<td>sl_Connect()</td>
<td>Client</td>
<td>Both</td>
<td>Initiates a connection on a socket</td>
</tr>
<tr>
<td>accept()</td>
<td>sl_Accept()</td>
<td>Server</td>
<td>TCP</td>
<td>Accepts an incoming connection on a socket</td>
</tr>
<tr>
<td>send(), recv()</td>
<td>sl_Send(), sl_Recv()</td>
<td>Both</td>
<td>TCP</td>
<td>Writes and reads data to and from TCP socket.</td>
</tr>
<tr>
<td>write(), read()</td>
<td>Not Supported</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sendto(), recvfrom()</td>
<td>sl_SendTo(), sl_RecvFrom()</td>
<td>Both</td>
<td>UDP</td>
<td>Causes the system to release resources allocated to a socket. In case of TCP, the connection is terminated.</td>
</tr>
<tr>
<td>close()</td>
<td>sl_Close()</td>
<td>Both</td>
<td>Both</td>
<td></td>
</tr>
<tr>
<td>gethostbyname(), gethostbyaddr()</td>
<td>Not Supported</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>select()</td>
<td>sl_Select()</td>
<td>Both</td>
<td>Both</td>
<td>Used to pend, waiting for one or more of a provided list of sockets to be ready to read, ready to write, or that have errors</td>
</tr>
<tr>
<td>poll()</td>
<td>Not supported</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getsockopt()</td>
<td>sl_SockOpt()</td>
<td>Both</td>
<td>Both</td>
<td>Retrieves the current value of a particular socket option for the specified socket</td>
</tr>
<tr>
<td>setsockopt()</td>
<td>sl_SetSockOpt()</td>
<td>Both</td>
<td>Both</td>
<td>Sets a particular socket option for the specified socket</td>
</tr>
<tr>
<td>htons(), ntohs()</td>
<td>sl_Htons(), sl_Ntohs()</td>
<td>Both</td>
<td>Both</td>
<td>Reorders the bytes of a 16-bit unsigned value from processor order to network order</td>
</tr>
<tr>
<td>htonl(), ntohl()</td>
<td>sl_Htonl(), sl_Ntohl()</td>
<td>Both</td>
<td>Both</td>
<td>Reorders the bytes of a 32-bit unsigned value from processor order to network order</td>
</tr>
</tbody>
</table>

6.7 Number of Available Sockets
There are a total of eight regular (non-secured) sockets, from which two sockets can be secured* sockets. If all of the sockets are client sockets, a total of eight sockets can be used.


- 6 “regular” sockets and 2 “secured” sockets
- 7 “regular” sockets and 1 “secured” socket

When some of the sockets are server sockets, then the number of available sockets for communication depends on the number of listening sockets. If one socket reserved for public sockets, is used to listen to incoming client requests, this leaves seven private sockets for actual client communication. If two server sockets are reserved for listening, only six private sockets will remain for communication, and so forth.

The number of available server sockets for UDP connection remains eight because UDP is a connectionless socket. Since it does not require a socket to be in listening mode all the time, all eight sockets can be used for client communication.

The server side secure socket (SSL/TLS) connection number is not affected because you can use a regular server socket for listening. Once accepting a new client connection, you can switch to the secure socket.

### 6.8 Packet Aggregation

By default, the SimpleLink device concatenates incoming received packets (Rx side). This is done to allow higher Rx throughput and is applicable to any type of sockets. This means that when calling to `sl_Recv()` or `sl_RecvFrom()`, the SimpleLink device tries to return the requested number of bytes asked by these API function calls.

When using UDP socket (`sl_RecvFrom`) this might be a problematic feature, since UDP is a connectionless data socket, a server socket can receive data from multiple client, thus aggregating these packets might be a wrong implementation.

The packet aggregation feature can be disabled upon using `sl_NetCfgSet()` API.

```c
u8 RxAggrEnable = 0;

sl_NetCfgSet(SL_SET_HOST_RX_AGGR, 0, sizeof(RxAggrEnable), (_u8 *) &RxAggrEnable);
```
Chapter 7

Device Hibernate

7.1 Overview

Hibernate is the lowest power state of the device. In this state the volatile memory of the Wi-Fi subsystem is not maintained. Only the RTC is maintained, for faster boot time and for keeping the system date and time.

The Wi-Fi subsystem goes into hibernate on a call to the `sl_Stop` API. This API receives only one parameter, a timeout parameter that configures the device to use the minimum amount of time to wait before going to hibernate.

For more details, refer to Chapter 18 and the API Doxygen application report.
# Provisioning

## Chapter 8

### Topic |
<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.1 Overview</td>
</tr>
<tr>
<td>8.2 SmartConfig</td>
</tr>
<tr>
<td>8.3 AP Mode</td>
</tr>
<tr>
<td>8.4 WPS</td>
</tr>
</tbody>
</table>
8.1 Overview

Wi-Fi provisioning is the process of connecting a new Wi-Fi device (station) to a Wi-Fi network (access point). The provisioning process involves loading the station with the network name (often referred to as SSID) and its security credentials. The Wi-Fi security standard distinguishes between personal security, mostly used in homes and businesses, and enterprise security, used in large offices and campuses. Provisioning a station for enterprise security usually involves installing certificates, which are used to verify the integrity of the station and the network by interaction with a security server managed by the IT department. Personal Wi-Fi security, on the other hand, needs to be handled by users at home, and it simply involves typing a pre-defined password. To provide robust security, the password can be as long as 64 characters.

For more details on provisioning code example and usage, see the simplelink wiki:

8.2 SmartConfig

8.2.1 General Description

SmartConfig technology is a TI proprietary provisioning method designed for headless devices introduced in 2012. It uses a mobile application to broadcast the network credentials from a smartphone, tablet, or to an unprovisioned TI Wi-Fi device. When SmartConfig is triggered in the un-provisioned device, it enters a special scan mode, waiting to pick up the network information that is being broadcasted by the phone app. The phone needs to be connected to a Wi-Fi network to be able to transmit the SmartConfig signal over the air. Typically this is the same home network onto which the new device is going to be provisioned.

For more details on the SmartConfig smartphone application, see the simplelink wiki

8.2.2 How to Use / API

8.2.2.1 Automatic Activation (Out of the Box)

To enable the feature, start the SimpleLink device. The device should start as STA role. Assuming no profile was added earlier, after a few seconds with no commands SmartConfig should start.

To start the SmartConfig application on a smart phone or PC:
1. Connect the smart phone to any Wi-Fi network.
2. Enter the WLAN credentials (SSID, security credentials).
3. Supply a key (optional: encrypts the Wi-Fi password).
4. Press the Start button.

The SmartConfig operation should complete in a few seconds, but may take up to two minutes to complete. If the requested network is in the proximity of the device, the device connects to it immediately.

The following topics apply when using automatic activation:
• Any command sent to the device terminates the SmartConfig operation.
• If a key is stored at the device serial flash, the password will be encrypted and the key must be supplied.
• If the device configuration was changed before SmartConfig automatically started, problems may occur. In this case, ensure the following:
  – Auto Start policy is set
  – Auto SmartConfig policy is set
  – No profile was added earlier

To verify the configuration, call:

sl_WlanPolicyGet(SL_POLICY_CONNECTION, 0, pVal, pValLen);

The returned policy is stored in the allocated buffer pointed by pVal.
Bits 0 and 4 should be set if Auto Start and Auto SmartConfig policies are set.

If this is not the case, set these policies manually by calling:

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

To ensure no profile is saved, remove all saved profiles by calling:

```c
sl_WlanProfileDel(255);
```

After sending these commands, reset the device and SmartConfig should operate successfully.

### 8.2.2.2 Manual Activation

To start SmartConfig manually, send the following command:

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

Parameters description:
- **groupIdBitmask** – Use 1 as the default group ID bitmask (group ID 0).
- **cipher** – To encrypt the password when the encryption key is not stored in the serial flash of the device, use:
  - `cipher = 1`
  - `publicKeyLen = 16`
  - `group1KeyLen = 0`
  - `group2KeyLen = 0`
  - `publicKey = put the key here (use a 16-character string)`
  - `group1Key = NULL`
  - `group2Key = NULL`
- **publicKeyLen** – To encrypt the password when the encryption key is stored in the serial flash of the device, use:
  - `cipher = 0`
  - `publicKeyLen = 0`
  - `group1KeyLen = 0`
  - `group2KeyLen = 0`
  - `publicKey = NULL`
  - `group1Key = NULL`
  - `group2Key = NULL`
- **group1KeyLen** – To avoid encrypting the password use:
  - `cipher = 1`
  - `publicKeyLen = 0`
  - `group1KeyLen = 0`
  - `group2KeyLen = 0`
  - `publicKey = NULL`
  - `group1Key = NULL`
  - `group2Key = NULL`

After sending this command, SmartConfig starts.

When running SmartConfig manually, if a key to encrypt the password is stored in the external serial flash or supplied at the command, this key must be supplied in the SmartConfig application.
8.2.2.3 Stopping Smart Config

To stop the SmartConfig operation call:

```c
sl_WlanSmartConfigStop()
```

**NOTE:**

- After the device is connected to the requested network it should receive an IP address from the AP or router.
- The SmartConfig operation may not end successfully if the Wi-Fi network to which the smart phone is connected uses transmissions modes and rates not suitable for SmartConfig. In this case, use the AP Provisioning method.

8.3 AP Mode

8.3.1 General Description

Access Point (AP) mode is the most common provisioning method today for headless devices. In AP mode the un-provisioned device wakes-up initially as an AP with an SSID defined by the equipment manufacturer. Before trying to connect to the home network for the first time, the un-provisioned device creates a network of its own, allowing a PC or a smart phone to connect to it directly to facilitate its initial configuration.

8.3.2 How to Use / API

1. First, start the SimpleLink device in AP Role. There are two methods to start:
   a. Call `sl_WlanSetMode(mode)` where the mode should be ROLE_AP (2). Reset the device. For additional information, see Chapter 10.
   b. Force the device to start as an AP.
2. Search for the device with a smart phone or any other device with Wi-Fi connectivity. The device should appear in the list containing all available networks. The device name should be `mysimplelink-xxxyyyzz`, where xxyyzz are the last six digits of the device MAC address. If you have already changed the device SSID by calling `sl_WlanCfgSet(0, 0, ssid_length, ssid)`, this SSID will be shown on the list. If you have not changed the SSID, but changed the device URN by calling `sl_NetAppSet(16, 0, urn_length, urn)`, then the device name should be `urn-xxxyyyzz`, where xxyyzz are the last six digits of the device MAC address.
3. At the device screen, choose the device network and connect to it.

**NOTE:** The connection should be unsecured unless you changed the mode. In this case, supply the password entered when the device requests it.

4. Open a browser once the device is connected and go to: [http://www.mysimplelink.net](http://www.mysimplelink.net).
5. Go to the Profiles tab.
6. Enter the WLAN credentials (SSID, security type, and key), select the priority for this profile (any value between 0-7), and press Add. See Figure 8-1.
Figure 8-1. AP Mode Connect

7. Scroll to the bottom of the web page and check that the device is added to the Profiles (the password will not be displayed). See Figure 8-2.

Figure 8-2. Profiles

8. Start the device in STA Role. Go to the Device Config tab and set Device Mode to Station (or remove the Force AP constraint) and reset the device. After resetting the device, the device connects automatically to the requested network. See Figure 8-3.
8.3.3  Things to Note When Configuring AP Provisioning

When connecting with a smart phone to the (AP) device, the device allocates an IP address as it runs the DHCP server. The smart phone (or other configuring device) should not be using a static IP address.

After entering the WLAN credentials (at the Profiles tab), move to STA mode and reset the device. The device should connect to the requested network and receive an IP address from the AP or router.

8.4  WPS

8.4.1  General Description

Wi-Fi Protected Setup (WPS) is the only industry standard available today for provisioning of headless devices. It was introduced by the Wi-Fi Alliance in 2006 as an easy and secure method to provision devices without knowing the network name and without typing long passwords. The standard defines two mandatory methods for WPS-enabled Access Points (APs): Personal Identification Number (PIN) method and PushButton-Connect (PBC) method.

**Push-button:** Push the W+PS button in the AP or, if the button is not available, start the WPS process using the GUI of the AP. The AP will enter the WPS provisioning process for 2 minutes. During this period, the SimpleLink device should also enter the WPS provisioning process by calling the `sl_WlanConnect` API with WPS parameters (see Section 8.4.3). For example, calling this API can be mapped to a push-button in the MCU. A wireless network with a network name and security is configured automatically at the end of this process.

**PIN-based:** Enter the pin generated by the host using the GUI of the AP. The AP will enter WPS provisioning process for 2 minutes. During this period, the SimpleLink device should also enter the WPS provisioning process by calling the `sl_WlanConnect` API with WPS parameters (see Section 8.4.3). A wireless network with a network name and security is configured automatically at the end of this process.

Once the WPS process completes successfully, connection with the AP is established in the correct security setting according to the configuration of the AP (Open, WEP, WPA, or WPA2). The connection parameters are saved as a profile. Using the connection policy AUTO triggers a re-connection after a reset.

8.4.2  How to Use / API

```c
sl_WlanConnect(char* pName, int NameLen, unsigned char *pMacAddr, SlSecParams_t* pSecParams, SlSecParamsExt_t* pSecExtParams);
```
This API with the correct settings can trigger a WPS connection with both configurations: push-button and PIN-based.

Parameter configuration:

- **pName** – NULL
- **NameLen** – 0
- **unsigned char *pMacAddr** – NULL
- **SlSecParamsExt_t** *pSecExtParams** – not relevant for WPS, set as NULL
- **Push-button:**
  - **SlSecParams_t** *pSecParams**: Type – SL_SEC_TYPE_WPS_PBC (3)
    - Key – NULL
    - Key length – set to 0
- **PIN-based:**
  - **SlSecParams_t** *pSecParams**: Type – SL_SEC_TYPE_WPS_PIN (4)
    - Key – WPS pin code
    - Key length – WPS pin code length

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

This API retrieves the profile parameters saved during the WPS connection process.

```
sl_WlanProfileDel(int Index)
```

This API is used to delete a profile saved during the WPS connection process. Calling this API with index set as 255 erases all stored profiles.

### 8.4.3 Example of Using WPS

```c
// 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);
    }
  }

}  

8.4.4 Tradeoffs Between Provisioning Options

<table>
<thead>
<tr>
<th>Provisioning Method</th>
<th>Access Point Mode</th>
<th>SmartConfig</th>
<th>WPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>What’s needed</td>
<td>Web browser</td>
<td>Android or iOS phone app</td>
<td>Push button on router</td>
</tr>
<tr>
<td>Networks supported</td>
<td>Any network</td>
<td>Networks connections with MIMO, 5 GHz, ISO-40 MHz, and proprietary modulation schemes are not supported</td>
<td>WPS enabled routers only</td>
</tr>
<tr>
<td>How many steps</td>
<td>Multiple steps</td>
<td>1 step</td>
<td>1 step (push button)</td>
</tr>
<tr>
<td>Number of devices configured</td>
<td>Configure one device</td>
<td>Configure multiple steps</td>
<td>Configure one device</td>
</tr>
<tr>
<td>Home network connection</td>
<td>Phone must disconnect from home network</td>
<td>Phone stays connected to the home network</td>
<td>N/A</td>
</tr>
<tr>
<td>Secure</td>
<td>Can be secure</td>
<td>Can be secure</td>
<td>Not secured</td>
</tr>
<tr>
<td>Remote applications</td>
<td>Not required</td>
<td>Required (supported by Android 4.2+ and iOS 6+)</td>
<td>N/A</td>
</tr>
<tr>
<td>Additional notes</td>
<td>N/A</td>
<td>SSID in Chinese or Asian characters are not recognized</td>
<td>N/A</td>
</tr>
</tbody>
</table>

- Each provisioning method has its merits and limitations.
- Since no provisioning method is perfect, a good practical approach would be to support more than one option in any given product. This would ensure maximum provisioning robustness of the final product.

**NOTE:** Products with SmartConfig, should also have AP mode or WPS as provisioning fall backs.
Chapter 9
SWRU368B—April 2014—Revised May 2018

Security

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.1 WLAN Security</td>
<td>49</td>
</tr>
<tr>
<td>9.2 Secured Socket</td>
<td>51</td>
</tr>
<tr>
<td>9.3 Limitations</td>
<td>54</td>
</tr>
</tbody>
</table>
9.1 WLAN Security

9.1.1 Personal

The Wi-Fi subsystem supports the Wi-Fi security types AES, TKIP, and WEP. The personal security type and personal security key are set in both the manual connection API or profiles connection API by the same parameter type – `SlSecParams_t`. This structure consists of the following fields:

- **Security Type** – The type of security being used. Options include:
  - `SL_SEC_TYPE_OPEN` – No security (default value).
  - `SL_SEC_TYPE_WEP` – WEP security.
  - `SL_SEC_TYPE_WPA` – Used for both WPA\PSK and WPA2\PSK security types, or a mixed mode of WPA\WPAPSK security type (for example, TKIP, AES, mixed mode).
  - `SL_SEC_TYPE_WPA_ENT` – WPS security. For more information refer to Section 8.4.
  - `SL_SEC_TYPE_WPS_PBC_ENT` – Push-button WPS security. For more information refer to Section 8.4.
  - `SL_SEC_TYPE_WPS_PIN_ENT` – Pin-based WPS security. For more information refer to Section 8.4.

- **Key** – A character area containing the pre-shared key (PSK) value.

- **Key length** – The number of characters of the pre-shared key.

Example code for adding a WPA2 secured AP profile:

```c
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);
```

9.1.2 Enterprise

9.1.2.1 General Description

The SimpleLink device supports Wi-Fi enterprise connection, according to the 802.1x authentication process. An enterprise connection requires the radio server behind the AP to authenticate the station. The following authentication methods are supported on the device:

- EAP-TLS
- EAP-TTLS with MSCHAP
- EAP-TTLS with TLS
- EAP-TTLS with PSK
- EAP-PEAP with TLS
- EAP-PEAP with MSCHAP
- EAP-PEAP with PSK
- EAP-FAST

After the station has been authenticated, the AP and the station negotiate WPA(1/2) security.

9.1.2.2 How to Use / API

When connecting to an enterprise network, three files are needed:

- **Private Key** – The station (client) RSA private key file in PEM format
- **Client Certificate** – The certificate of the client given by the authenticating network (ensures the public key matches to the private key) in PEM format
- **Server Root Certificate Authority file** – This file authenticates the server. This file must be in PEM format.

These three files must be programmed with the following names so the device will use them:
• Certificate authority: /cert/ca.pem
• Client certificate: /cert/client.pem
• Private key: /cert/private.key

Establishing a connection:

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)

The `sl_WlanConnect` and the `sl_WlanProfileAdd` commands are used for different types of Wi-Fi
connection. The connect command is for a one-shot connection, and the add profile used when auto
connect is on (see the add profile white papers). Use those commands with extra security parameters for
the enterprise connection – SlSecParamsExt_t.

A short view of the first five parameters of those commands follows. Learn more of the extra parameters
of the add profile command in its white paper.

- **SSID name** – The name of the Wi-Fi network
- **SSID length**
- **Flags** – Not applicable for enterprise connection
- **Pointer to SlSecParams_t** –

```c
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;
```

- **Pointer to SlSecParamsExt_t** –

```c
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_PAE0_TLS
                             // SL_ENT_EAP_METHOD_PAE0_MSCHAPv2
                             // SL_ENT_EAP_METHOD_PAE0_PSK
                             // SL_ENT_EAP_METHOD_PAE1_TLS
                             // SL_ENT_EAP_METHOD_PAE1_MSCHAPv2
                             // SL_ENT_EAP_METHOD_PAE1_PSK
                             // SL_ENT_EAP_METHOD_FAST_AUTH_PROVISIONING
                             // SL_ENT_EAP_METHOD_FAST_UNAUTH_PROVISIONING
                             // SL_ENT_EAP_METHOD_FAST_NO_PROVISIONING
} SlSecParamsExt_t;
```

9.1.2.3 Example

Figure 9-1 shows an example of a simple WLAN connect command to an enterprise network.
9.1.2.4 Limitations

There is no command to bind a certificate file to a WLAN enterprise connection. The certificates of the network must be programmed with the names specified in Section 9.1.2.2.

9.2 Secured Socket

9.2.1 General Description

SSL is a secured socket over TCP that lets the user connect to servers (and Internet sites) securely, or to open a secured server.

This chapter covers how to use the socket with the host driver, and how to generate certificates and keys for the SSL.

9.2.2 How to Use / API

sl_Socket(SL_AF_INET, SL_SOCK_STREAM, SL_SEC_SOCKET) – This command opens a secured socket. The first two parameters are typical TCP socket parameters, and the last parameter enables the security.

Use any standard BSD commands (sl_Close, sl_Listen, sl_Accept, sl_Bind, sl_SetSockOpt and so forth) to open client, open server, change socket parameters, and more.

The BSD commands let you connect without choosing the SSL method (SSLv3 TLS1.0/1.1/1.2) and without choosing the connection cipher suit (those two are negotiated in the SSL handshake).

In client socket, the certificate of the server is not verified until the root CA verifies the server.

In server socket, the user must supply the server certificate and private key.

Use the setsockopt command with proprietary options to configure the secured parameters of the socket.

```c
void WlanConnect()
{
    SLSecParams_t secParams;
    SLSecParamsExt_t extParams;

    // fill the security parameters
    secParams.Key = "xfdsknke34wki4de";
    secParams.KeyLen = strlen("xfdsknke34wki4de");
    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_PEA0_MSCHAPv2;

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

Figure 9-1. WLAN Connect Command
9.2.2.1 Selecting a Method
Manually defines the SSL method. In simplelink WiFi device, SSLv3 TLSv 1.0/1.1/1.2 is supported.

SlSockSecureMethod method; method.secureMethod = Choose one of the following defines:
- 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_SSSLv3_TLSV1_2
- SL_SO_SEC_METHOD_DLSV1

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

9.2.2.2 Selecting a Cipher Suit
Manually defines the SSL connection and handshake security algorithms, also known as cipher suits.

SlSockSecureMask Mask; mask.secureMask = Choose logic or between the following:
- 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));

9.2.2.3 Selecting the Secured Files for the Socket
Defines which files the socket will use for the connection. There are four files that can be attached to a socket:
- Private key
- Certificate
- Root CA
- DH file

These are the files in client and server sockets:

Client
- Private Key, Certificate – If a client must be authenticated by the server, both private key and certificate are mandatory together. Use two setsockopt commands to configure each file.
- Root CA – This is the root certificate authority that issued the server certificate and is used to validate that the server is authentic. This file is not mandatory. If the server is not verified, the connection occurs, but the connect command returns an error, SL_ESECSNOVERIFY. This error can be ignored as it is only a warning for an unauthenticated connection.
- DH file – No use in client

Server
- Private Key, Certificate – Mandatory for server
- Root CA – The certificate issued to the client. When file is set, it obligates the client to send their certificate for client authentication.
• DH file – Used to support the DH cipher suit – TLS_DHE_RSA_WITH_AES_256_CBC_SHA.

To bind files to a socket, program the file to the device. Then use the setsockopt to enter the file name. All secured files must be in DER format.

Setsockopt options for the secured files:
• 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

For example, use the file rootCA.der that is in the device:

```
strlen("rootCA.der"));
```

Note the strlen in the setsockopt, and not sizeof.

### 9.2.2.4 Set Domain Name for Verification and SNI

Set the domain name to verify the desired domain during the SSL handshake. The domain verification is used to help against "man in the middle" attacks, where a third party could buy a fake certificate from the same root CA that signed the certificate of the server, and redirect the traffic to their server. In addition to the full chain verification, TI recommends checking the domain name. This option is only available for client mode. This option should be called before sl_Connect or sl_Listen. Setting a domain name also enables the SNI extension of the client hello message, according to RFC 6066.

Example:

```
-il6 status;

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

### 9.2.3 Example of Using the SSL

```c
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;

9.2.4 Supported Cryptographic Algorithms

Table 9-1. Supported Cryptographic Algorithms

<table>
<thead>
<tr>
<th>Cryptographic Algorithms</th>
<th>Standard Protocol</th>
<th>Purpose</th>
<th>Length of Encryption Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>RC4</td>
<td>WEP, TKIP</td>
<td>Data encryption</td>
<td>128 bits</td>
</tr>
<tr>
<td>AES</td>
<td>WPA2</td>
<td>Data encryption, authentication</td>
<td>256 bits</td>
</tr>
<tr>
<td>DES</td>
<td>–</td>
<td>Data encryption</td>
<td>56 bits</td>
</tr>
<tr>
<td>3DES</td>
<td>–</td>
<td>Data encryption</td>
<td>56 bits</td>
</tr>
<tr>
<td>SHA1</td>
<td>EAP-SSL/TLS</td>
<td>Authentication</td>
<td>160 bits</td>
</tr>
<tr>
<td>SHA256</td>
<td>EAP-SSL/TLS</td>
<td>Authentication</td>
<td>256 bits</td>
</tr>
<tr>
<td>MD5</td>
<td>EAP-SSL/TLS</td>
<td>Authentication</td>
<td>128 bits</td>
</tr>
<tr>
<td>RSA</td>
<td>EAP-SSL/TLS</td>
<td>Authentication</td>
<td>2048 bits</td>
</tr>
<tr>
<td>DHE</td>
<td>EAP-SSL/TLS</td>
<td>Authentication</td>
<td>2048 bits</td>
</tr>
<tr>
<td>ECDHE</td>
<td>EAP-SSL/TLS</td>
<td>Authentication</td>
<td>160 bits</td>
</tr>
</tbody>
</table>

- Supported SSL key sizes:
  - Client mode
    - Client mode
    - Key exchange and challenge – must be less or equal to 4096
    - Client authentication – must be less or equal to 2048
  - Server mode
    - Signature authentication check – must be less or equal to 2048
    - Key exchange and challenge – must be less or equal to 2048
    - Client authentication – must be less or equal to 2048

9.3 Limitations

CC31xx and CC32xx can support WEP Hex format with some small limitations

9.3.1 Main Known Limitations

9.3.1.1 STA Mode

Table 9-2. STA Mode

<table>
<thead>
<tr>
<th>Action</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>sl_connect(Host) OR Add profile(Host)</td>
<td>WEP Hex can be supported</td>
</tr>
<tr>
<td></td>
<td>WPA Hex is not supported</td>
</tr>
<tr>
<td>Smart Config</td>
<td>WPA ASCII is not supported, supports only WEP Hex</td>
</tr>
<tr>
<td></td>
<td>WPA Hex is not supported</td>
</tr>
<tr>
<td>HTTP (add profile)</td>
<td>WEP Hex can be supported</td>
</tr>
<tr>
<td></td>
<td>WPA Hex is not supported</td>
</tr>
</tbody>
</table>
9.3.1.2 **AP Mode**

<table>
<thead>
<tr>
<th>ApConfigCmd(Host)</th>
<th>WEP Hex can be supported</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>WPA Hex is not supported</td>
</tr>
<tr>
<td>HTTP(AP_config)</td>
<td>WEP Hex can be supported</td>
</tr>
<tr>
<td></td>
<td>WPA Hex is not supported</td>
</tr>
</tbody>
</table>

### Table 9-3. AP Mode

9.3.1.3 **JavaScript Example**

```javascript
<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>
```

9.3.1.4 **Host Driver Example**

```c
static INT32 establishConnectionWithAP()
{
    char SSID[MAX_SSID_SIZE];
    char password[MAX_PASSKEY_SIZE];
    SlSecParams_t secParams = {0};
    INT32 retVal = -1;
    /*********************************************************
    /** AP is configured to WEP with 40bit Hex password of: "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;
}
# Chapter 10

## AP Mode

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.1 General Description</td>
<td>58</td>
</tr>
<tr>
<td>10.2 Setting AP Mode – API</td>
<td>58</td>
</tr>
<tr>
<td>10.3 WLAN Parameters Configuration – API</td>
<td>58</td>
</tr>
<tr>
<td>10.4 WLAN Parameters Query – API</td>
<td>59</td>
</tr>
<tr>
<td>10.5 AP Network Configuration</td>
<td>59</td>
</tr>
<tr>
<td>10.6 DHCP Server Configuration</td>
<td>60</td>
</tr>
<tr>
<td>10.7 Setting Device URN</td>
<td>61</td>
</tr>
<tr>
<td>10.8 Asynchronous Events Sent to the Host</td>
<td>61</td>
</tr>
<tr>
<td>10.9 Example Code</td>
<td>62</td>
</tr>
</tbody>
</table>
10.1 General Description

The AP should be set and configured by calling APIs allowing the configuring of some WLAN parameters and some IP parameters. This chapter describes the parameters that can be configured and how to configure them.

10.2 Setting AP Mode – API

sl_WlanSetMode(const unsigned char mode)

Where mode should be ROLE_AP (2). This change takes affect after reset.

NOTE: This change takes affect after Reset.

10.3 WLAN Parameters Configuration – API

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

This function sets the user parameter to configure. The input parameters are:
• ConfigId – Should be set to SL_WLAN_CFG_AP_ID (0) or SL_WLAN_CFG_GENERAL_PARAM_ID (1), according to the parameter
• ConfigOpt – Identifies the parameter to configure
• ConfigLen – The parameter size in bytes
• pValues – Pointer to memory containing the parameter

NOTE: This change takes affect after reset.

Table 10-1 describes how to configure each parameter.

Table 10-1. WLAN Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>ConfigId</th>
<th>ConfigOpt</th>
<th>ConfigLen</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSID</td>
<td>Service set identifier (SSID) - a 1- to 32-byte string, commonly called the &quot;network name&quot;. Default: Composed from the device URN and last 6 digits of the MAC Address. For example – mysimplelink-112233.</td>
<td>SL_WLAN_CFG_AP_ID (0)</td>
<td>0</td>
<td>1-32</td>
</tr>
<tr>
<td>Beacon interval</td>
<td>The time interval between beacon transmissions. Range: 15 &lt;= interval &lt;= 65,535 milliseconds. Default value: 100.</td>
<td>SL_WLAN_CFG_AP_ID (0)</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>SSID Hidden</td>
<td>Whether to NOT broadcast the SSID inside the Beacon frame. Default value: Disabled</td>
<td>SL_WLAN_CFG_AP_ID (0)</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>
Table 10-1. WLAN Parameters (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>ConfigId</th>
<th>ConfigOpt</th>
<th>ConfigLen</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTIM</td>
<td>Indicates the interval of the Delivery Traffic Indication Message (DTIM). A DTIM field is a countdown field informing clients of the next window for listening to broadcast and multicast messages. DTIM field is in the Beacon frame. Range: DTIM &gt; 0 Default value: 2</td>
<td>SL_WLAN_CFG_AP_ID (0)</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>Security Type</td>
<td>Security mode for the network. Range: Open (no security) / WEP / WPA Default: Open</td>
<td>SL_WLAN_CFG_AP_ID (0)</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>Password</td>
<td>Password to use for the network in case Security Type is not open. Password should be human-readable string. For WEP, it should be 5 to 13 characters. For WPA it should be 8 to 63 characters.</td>
<td>SL_WLAN_CFG_AP_ID (0)</td>
<td>7</td>
<td>5-63</td>
</tr>
<tr>
<td>WPS State</td>
<td>Wi-Fi Protected Setup (originally Wi-Fi Simple Config) is a network security standard that lets users easily secure a wireless home network. Default value: Disabled</td>
<td>SL_WLAN_CFG_AP_ID (0)</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>Country code</td>
<td>Identifies the country code of the AP. Possible values are US (USA) or JP (Japan) or EU (Europe). Default value is “US”.</td>
<td>SL_WLAN_CFG_GENERAL_PARAM_ID (1)</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>AP TX Power</td>
<td>AP transmission power. Range: 0 (maximal) – 15 (minimal). Default: 0 (maximal TX power).</td>
<td>SL_WLAN_CFG_GENERAL_PARAM_ID (1)</td>
<td>11</td>
<td>1</td>
</tr>
</tbody>
</table>

10.4 WLAN Parameters Query – API

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

This function gets the parameter the user requested where:

- **ConfigId** – Should be set to SL_WLAN_CFG_AP_ID (0) or SL_WLAN_CFG_GENERAL_PARAM_ID (1) as in the SET function
- **ConfigOpt** – Identifies the parameter to configure. Should be the address of the field. The value of the field is the same as in the SET function.
- **ConfigLen** – Output: pointer to the parameter size returned in bytes
- **pValues** – Output: pointer to memory containing the parameter

10.5 AP Network Configuration

The user must set the AP IP parameters, specifically the IP address, gateway address, DNS address, and network mask.

To set the IP addresses call:

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

- **ConfigId** – Should be set to SL_IPV4_AP_P2P_GO_STATIC_ENABLE (7)
- **ConfigOpt** – Should be set to 1
- **ConfigLen** – Should be the parameter size in bytes
- **pValues** – Pointer to memory containing the parameter

This example shows how to configure IP, gateway, and DNS addresses to 9.8.7.6 and the subnet to 255.255.255.0:

```c
_NetCfgIpV4Args_t ipV4;
ipV4.ipV4 = 0x09080706;
ipV4.ipV4Mask = 0xFFFFFFFF00;
sl_NetCfgSet(SL_IPV4_AP_P2P_GO_STATIC_ENABLE,
1
,sizeof(_NetCfgIpV4Args_t),
(unsigned char *)&ipV4);
```

**NOTE:** The changes take affect after reset.

Default values:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>ipV4</td>
<td>0xc0a80101;</td>
<td>/* 192.168.1.1 */</td>
</tr>
<tr>
<td>defaultGatewayV4</td>
<td>0xc0a80101;</td>
<td>/* 192.168.1.1 */</td>
</tr>
<tr>
<td>ipV4DnsServer</td>
<td>0xc0a80101;</td>
<td>/* 192.168.1.1 */</td>
</tr>
<tr>
<td>subnetV4</td>
<td>0xFFFFFFFF00;</td>
<td>/* 255.255.255.0 */</td>
</tr>
</tbody>
</table>

### 10.6 DHCP Server Configuration

The user must enable the DHCP server and configure DHCP server parameters, DHCP addresses, and lease time.

To start the DHCP server call:

```c
sl_NetAppStart(SL_NET_APP_DHCP_SERVER_ID);
```

Where SL_NET_APP_DHCP_SERVER_ID is 2.

To stop DHCP server call:

```c
sl_NetAppStop(SL_NET_APP_DHCP_SERVER_ID);
```

Default value: DHCP server enabled.

To configure the DHCP parameters use the following API:

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

- **AppId** – Should be set to SL_NET_APP_DHCP_SERVER_ID (2)
- **Option** – Should be set to NETAPP_SET_DHCP_SRV_BASIC_OPT (0)
- **OptionLen** – Should be the parameter size in bytes
- **pOptionValue** – Pointer to memory containing the parameter

This example shows how to configure the parameters (starting IP address 9.8.7.1, ending IP address 9.8.7.5, lease time = 1000 seconds):

```c
SlNetAppDhcpServerBasicOpt_t dhcpParams;
unsigned char outLen = sizeof(SlNetAppDhcpServerBasicOpt_t);
dhcpParams.lease_time = 1000;
```
**10.7 Setting Device URN**

To set the device name call:

```c
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

Default value: `mysimplelink`

**10.8 Asynchronous Events Sent to the Host**

When a station is newly connected or disconnected to or from the AP, an event is sent to the host.

The event opcodes are:  
`SL_OPCODE_WLAN_STA_CONNECTED` 0x082E  
`SL_OPCODE_WLAN_STA_DISCONNECTED` 0x082F

The events include the parameters shown in Table 10-2.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Bytes</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peer device name</td>
<td>32</td>
<td>Relevant for P2P</td>
</tr>
<tr>
<td>Peer MAC address</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Peer device name length</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>WPS device password ID</td>
<td>1</td>
<td>0 – not available</td>
</tr>
<tr>
<td>Own SSID</td>
<td>32</td>
<td>Relevant for P2P</td>
</tr>
<tr>
<td>Own SSID length</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Padding</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

When the IP address is released to a station, an event is sent to the host.

The event opcode is:  
`SL_OPCODE_NETAPP_IP_LEASED` 0x182C

The event includes the parameters shown in Table 10-3.
Table 10-3. Event Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Bytes</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP address</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Lease time</td>
<td>4</td>
<td>In seconds</td>
</tr>
<tr>
<td>Peer MAC address</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Padding</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

When an IP address is released by a station, an event is sent to the host.
The event opcode is: SL_OPCODE_NETAPP_IP_RELEASED 0x 182D
The event includes the parameters shown in Table 10-4.

Table 10-4. Event Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Bytes</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP address</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Peer MAC address</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Reason</td>
<td>2</td>
<td>0 – peer released the IP address</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 – peer declined to this IP address</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 – Lease time was expired</td>
</tr>
</tbody>
</table>

10.9 Example Code

An example code showing how to configure the AP WLAN parameters and network parameters (IP addresses and DHCP parameters) follows. WLAN parameters are also read back.

```c
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
```
Example Code

```c
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);

// 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);
# Peer to Peer (P2P)

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.1 General Description</td>
<td>66</td>
</tr>
<tr>
<td>11.2 P2P APIs and Configuration</td>
<td>67</td>
</tr>
<tr>
<td>11.3 P2P Connection Events</td>
<td>73</td>
</tr>
<tr>
<td>11.4 Use Cases and Configuration</td>
<td>73</td>
</tr>
<tr>
<td>11.5 Example Code</td>
<td>75</td>
</tr>
</tbody>
</table>
11.1 General Description

11.1.1 Scope
P2P mode/role in simplelink WiFi devices is powerful feature that gives the device to connect to other devices without a need for an AP and by inheriting of the entire station AP attributes, and moreover this connection has more abilities and qualities like in power save.

11.1.2 Wi-Fi Direct Advantage
- Provide an easy and convenient manner to share, show, print and synchronize content wherever users go – home, public places, travel, work.
- Provide new connectivity scenarios that include the 100's of millions of Wi-Fi CERTIFIED devices that users already own.
- Eliminate the need for routers as it is for local area networks, and may also replace the need of Bluetooth for applications (that do not rely on low power).
- Deliver an industry-wide peer-to-peer solution based on broadly deployed Wi-Fi technologies.
- The use for peer-to-peer opportunistic power-save and notice of absence allows a full low power link in both sides (Group owner and Client), this is a big advantage compares to legacy AP-STA link.

11.1.3 Support and Abilities of Wi-Fi Direct
- P2P configuring device name, device type, listen, and operation channels
- P2P device discovery (FULL/SOCIAL)
- P2P negotiation with all intents (0 to 15)
- P2P negotiation Initiator policy – Active / Passive / Random Back Off
- P2P WPS method push-button and pin code (keypad and display)
- P2P establish as client role:
  - P2P device can join an existing P2P group.
  - P2P device can invite to reconnect a persistent group (fast-connect).
- P2P establish as group owner role:
  - P2P group owner can accept a join request.
  - P2P persistent group owner can respond to invite requests.
- P2P group remove
- P2P connect-disconnect-connect transition, also between different roles (for example GO-CL-GO)
- P2P client legacy PS and NoA support
- Separate IP configuration for P2P role
- Separate Net-Apps configuration on top of P2P-CL/GO role

11.1.4 Limitations
- No service discovery supported
- No GO-NoA supported
- Smart configuration for P2P device entry is limited to 15 entries, and is dynamically allocated rather than optimized as a station scan single entry.
- No autonomous group support
- P2P group owner mode supports single peer (client) connected (similar to AP).
- P2P find in profile search and connection is infinite, meaning if the remote device is not found the search will continue indefinitely.
11.2 P2P APIs and Configuration

P2P configuration uses the host driver APIs that control the simplelink WiFi device. Different options and modes of P2P configuration can be accessed by using specific APIs. By using a specific APIs, the user can set P2P abilities and use them. The configuration is divided into few easy sections. The configuration of specific sections has an influence on the configuration of others sections.

The configuration is divided into these sections:
- Configure P2P global parameters
- Configure P2P policy
- Configure P2P profiles policy
- Configure manual connection
- Configure a search for P2P devices and use these devices in the steps above
- P2P events

The next section explains how to configure P2P and how to use its options. Note that:

---

NOTE:
- Configuration is flushed and can be only done once.
- Not all APIs need to be used. Default parameters are used in the absence of a user configuration.
- All set APIs have a GET operation.

### 11.2.1 Configuring P2P Global Parameters

This section shows how to configure the device to be in a state of P2P and set general parameters. Configuration of the P2P role, P2P network parameters and P2P device name, device type and channels are part of this section.

#### 11.2.1.1 Set P2P Role

Set the device to P2P mode using an API:

```c
sl_WlanSetMode(ROLE_P2P)
```

This API puts the device in P2P mode. All other P2P configurations are not effected until entering P2P mode.

All other P2P configurations will not be effected until entering to P2P mode.

#### 11.2.1.2 Set P2P Network Configuration

Configure network configuration used by the P2P API:
- P2P client (same API as station):
  - Static IP:
    ```c
    sl_NetCfgSet(SL_IPV4_STA_P2P_CL_STATIC_ENABLE,1,sizeof(_NetCfgIpV4Args_t),(unsigned char *)&ipV4)
    ```
  - DHCP client:
    ```c
    sl_NetCfgSet(SL_IPV4_STA_P2P_CL_DHCP_ENABLE,1,1,&val);
    ```
- P2P GO (same API as AP):
  - GO own static IP:
    ```c
    sl_NetCfgSet(SL_IPV4_AP_P2P_GO_STATIC_ENABLE,1,sizeof(_NetCfgIpV4Args_t),(unsigned char *)&ipV4)
    ```

This API sets the network configuration used when the P2P role is set.
11.2.1.3 Set P2P Device Name

The following macro sets the P2P device name. Setting a unique P2P device name for every individual device is necessary, because the connection is based on device-name.

Default: TI_SIMPLELINK_P2P_xx (xx = random two characters)

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

11.2.1.4 Set P2P Device Type

The following macro sets the P2P device type. The device type allows P2P discovery parameters to recognize the device.

Default: 1-0050F204-1

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

11.2.1.5 Set P2P Listen and Operation Channels

The following macro sets the P2P operation and listen channels. The listen channel is used for the discovery state and can be 1, 6, or 11. The device will be in this channel when waiting for P2P probe requests. The operation channel is used only by the GO. The GO will move to this channel after the negotiation phase.

Default: Random between channels 1,6, or 11

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

**NOTE:** Regulator domain class should be 81 in 2.4G.

For example:
```
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);
```

11.2.2 Configuring P2P Policy

This section depicts the P2P policy configuration, including two more P2P working mode options given by the simplelink device.

- P2P intent value option – The P2P role of the device (client, GO, or don't care).
- Negotiation initiator option – Value used during P2P negotiation to indicate which side will initiate the negotiation, and which side will passively wait for the remote side to send negotiation and then respond.

11.2.2.1 Configure P2P Intent Value and Negotiation Initiator

This configuration uses macro SL_P2P_POLICY, the second parameters sent to function `sl_WlanPolicySet`.

For the intent value, three defines options can be used:
• SL_P2P_ROLE_CLIENT (intent 0): Indicates that the device is forced to be a P2P client.
• SL_P2P_ROLE_NEGOTIATE (intent 7): Indicates that the device can be either a P2P client or GO, depending on the P2P negotiation tie-breaker. This is the system default.
• SL_P2P_ROLE_GROUP_OWNER (intent 15): Indicates that the device is forced to be a P2P GO.

For negotiation initiator, three defines options can be used:
• SL_P2P_NEG_INITIATOR_ACTIVE: When the remote peer is found after discovery, the device immediately sends negotiation requests to the peer device.
• SL_P2P_NEG_INITIATOR_PASSIVE: When the remote peer is found after discovery, the device passively waits for the peer to start the negotiation before responding.
• SL_P2P_NEG_INITIATOR_RAND_BACKOFF: When the remote peer is found after discovery, the device triggers a random timer (1 to 7 seconds). The device waits passively for the peer to negotiate during this period. If the timer expires without negotiation, the device immediately sends negotiation requests to the peer device. This is the system default as two simplelink devices do not require any negotiation synchronization.

Use this configuration when working with two simplelink devices. The user may not have a GUI to start the negotiation, thus this option is offered to prevent the simultaneous negotiation of both devices after discovery.

API:

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

For example:

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

### 11.2.3 Configuring P2P Profile Connection Policy

This section discusses the profile connection policy. This policy lets the system connect to a peer without a reset or disconnect operation by the remote peer.

The mechanism describes how the device uses these profiles in relation to P2P automatic connection. A manual connection is also described in Section 11.2.6.

There is a general mechanism for peer profile and peer profile configuration, which is not described in this document. An example of how to add the profile is provided in Section 11.2.8.

This configuration uses the macro SL_CONNECTION_POLICY, the second parameter sent to function sl_WlanPolicySet.

There are four connection policy options:
• Auto Start – As in STA mode, if the device is not connected it starts P2P find to search for all P2P profiles configured on the device. If at least one candidate is found, the device tries to connect to it. If more than one device is found, the best candidate according to the profiles parameter is chosen.
• Fast Connect – In the P2P role, it is the equivalent to the P2P Persistent group but with a different meaning between GO and CL. This option is very useful to make fast connections after reset, but pay attention because it is dependent on the last connection state. This option has no meaning if there was no prior connection by the device, because the last connection parameters are saved and used by the fast connection option.
  – If the device was a P2P client in its last connection (before a reset or remote disconnect operation), then following reset it will send a p2p_invite to the previously connected GO to perform fast-reconnection.
– If the device was P2P GO in its last connection (before reset or remote disconnect operation), then following reset it will re-invoke the p2p_group_owner and wait for the previous connected peer to reconnect.

• OpenAP – Not relevant for P2P mode
• AnyP2P policy – Policy value that makes a connection to any P2P peer device found during discovery. This option does not need a profile. Relevant for negotiation with push-button only.

Each option should be sent or set in this macro as true or false. More than one option can be used; for example, the user can set both the auto-start and the fast connect options to true.

API:

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

For example:

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

11.2.4 Discovering Remote P2P Peers

This section shows how to start a P2P search and discovery, and how to see the remote P2P devices that are discovered.

Discovery is used for:

• Scanning and finding nearby devices, because P2P connection is based on the remote device name published during the discovery phase
• Manually connecting by host commands (not using existing profiles)
• Discovering the remote peers in the neighborhood and then configuring willing profiles, if the neighborhood is unknown and the user wants to set P2P profiles in the system.

11.2.4.1 How to Start P2P Discovery

A scan policy is needed to start the P2P find and discover remote P2P peers. Setting the scan policy to P2P performs a full P2P scan.

The setting of the scan policy should be under the P2P role. P2P discovery is performed as part of any connection, but can also be activated using the SCAN_POLICY.

NOTE:

• The setting of the scan policy should be under P2P role.
• P2P discovery is also performed as part of any connection, but can also be activated using SCAN_POLICY as well

API:

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

The second parameter enables the P2P scan operation, and the interval indicates the waiting time between P2P find cycles.

11.2.4.2 How to See/Get P2P Remote Peers (Network P2P List)

There are two ways to see and get P2P remote devices discovered during a P2P find and search operation:

• Listening to the event SL_WLAN_P2P_DEV_FOUND_EVENT

Copyright © 2014–2018, Texas Instruments Incorporated
• Calling to API `sl_WlanGetNetworkList`

`SL_WLAN_P2P_DEV_FOUND_EVENT`: This event is sent to each remote P2P that is found. It contains the MAC address, the name, and the name length of the remote device. By listening to this event, the user can find each remote P2P in the neighborhood.

`SL_WlanGetNetworkList`: By calling to this API, the user gets a list of remote peers found and saved in the device cache. This API is also used in station mode.

**API:**

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

- **Index** – Indicates the index in the list tables the P2P devices return to.
- **Count** – Shows how many peer devices should be returned.
- **pEntries** – The results are entered in to this, which is allocated by the user.

### 11.2.5 Negotiation Method

The next sections show how to make a P2P connection manually or automatically by profile, and the negotiation method before the WPS connection.

As stated in Section 11.2.3, the negotiation starts according to the intent and negotiation initiator parameters, but other parameters should be configured to finish this step successfully. These parameters influence the negotiation method and are supplied during the manual connection API command that comes from the host or when setting the profile for automatic connection. The negotiation method is done by the device without user interference.

There are two P2P negotiation methods to indicate the WPS phase that follows the negotiation:

- **P2P push-button connection** – Both sides negotiate with PBC method. Define `SL_SEC_TYPE_P2P_PBC`.
- **P2P pin code connection** – Divided to two options. PIN_DISPLAY looks for a pin to be written by its remote P2P. PIN_KEYPAD sends a pin code to its remote P2P.
  - Define `SL_SEC_TYPE_P2P_PIN_KEYPAD`.
  - Define `SL_SEC_TYPE_P2P_PIN_DISPLAY`.

If no pin code is entered, the NWP auto-generates the pin code from the device MAC using the following method:

1. Take the 7 ISB decimal digits in the device MAC address, and add checksum of those 7 digits to the LSB (total 8 digits). For example, if MAC is `03:4A:22:3B:FA:42`
2. Convert to decimal: `.059:250:066`
3. Seven ISB decimal digits are: `9250066`
4. WPS Pin Checksum digit: `2`
5. Default pin code for this MAC: `92500662`

There are two options to configure the negotiation method:

- **Setting the value in secParams struct and sending it as a parameter through the manual connection command.**
  - For push-button: `secParams.Type = SL_SEC_TYPE_P2P_PBC`
  - For pin code keypad: `secParams.Type = SL_SEC_TYPE_PIN_KEYPAD secParams.Key = 12345670`
  - For pin code display: `secParams.Type = SL_SEC_TYPE_PIN_DISPLAY secParams.Key = 12345670`
- **Sending the negotiation method defines and key as parameters through the P2P profile configuration.**
11.2.6 Manual P2P Connection

After finding a remote device by getting event SL_WLAN_P2P_DEV_FOUND_EVENT or by calling to the get_networkList API, there is an option to start a connection by using a command originating from the host. This command performs immediate P2P discovery. Once the remote device is found, the negotiation phase is started according to the negotiation initiator policy, method, and intent selected.

NOTE:

• This connection is not flashed, so in case of disconnection or reset, a re-connection is done only if the fast-connect policy is on.
• This connection is stronger than a connection made through profiles, as a P2P connection already exists in the system. The current connection is disconnected in favor of the manual connection.

API:

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

pName – The name of the remote device that is known to the user after getting event SL_WLAN_P2P_DEV_FOUND_EVENT or by calling to the get_networkList API.
NameLen – The length of pName
pMacAddr – The option to connect to a remote P2P according to its BSSID. Use {0,0,0,0,0,0} to connect according to MAC address.
pSecParams – See Section 11.2.5.
pSecExtParams – Value should be zero.

For example:

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

11.2.7 Manual P2P Disconnection

A manual disconnect option lets the user make a disconnection from its peer through a host command. This command performs a P2P group remove of the current active role, whether it is p2p-device, p2p-group-owner, or p2p-client.

API:

sl_WlanDisconnect();

11.2.8 P2P Profiles

Profile configuration makes an automatic P2P connection after a reset or disconnection from the remote peer device. This command stores the P2P remote device parameters in flash as a new profile along with profile priority. These profiles are similar to station profiles and have the same automatic connection behavior. The connection depends on the profile policy configuration.

If auto-start policy is on, a P2P discovery is performed. If one or more of the remote devices that are found are adapted to the profiles, a negotiation phase is started according to the negotiation initiator policy, method, and intent selected. The highest priority profile is chosen.

NOTE:

• If fast-connect policy is on and there already was a connection, then after a reset or disconnection from the remote peer, a fast connection starts according to the last connection parameters, without consulting the P2P profile list.
• If a manual connection is sent, then the profile connection is stopped by a disconnect command, and manual connection is initiated.

To add a profile with the push-button negotiation method, call to:
sl_WlanProfileAdd(char* pName, int NameLen, unsigned char *pMacAddr,
                   SlSecParams_t* pSecParams, SlSecParamsExt_t*
                   pSecExtParams, unsigned long Priority,
                   unsigned long Options)

An example of adding a profile with the push-button negotiation method:

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

An example of adding a profile with the keypad negotiation method:

```c
sl_WlanProfileAdd(SL_SEC_TYPE_P2P_PIN_DISPLAY,
                  remote_p2p_device,
                  strlen(remote_p2p_device),
                  bssidEmpty,
                  0/*Priority*/,key,8/*keylen*/,0);
```

### 11.2.9 Removing P2P Profiles

To delete a specific profile or all profiles use the following API:

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

### 11.3 P2P Connection Events

This section describes the P2P connection events. These events are sent by the device during connection and the user decides how to handle them.

- **SL_WLAN_P2P_NEG_REQ_RECEIVED_EVENT** – This event is sent if the negotiation is ended successfully.
  - It contains:
    - Device peer MAC address
    - Device peer name
    - Length of device peer name
    - Device peer WPS password ID

- **SL_WLAN_CONNECTION_FAILED_EVENT** – This event is sent if the connection fails, with the failure reason.
  - It contains:
    - Failure reason

- **SL_WLAN_CONNECT_EVENT** – This event is shared by P2P client and station. Indicates that the P2P connection has ended successfully and that the device is a P2P client.
  - It contains:
    - Remote device parameters

- **SL_WLAN_CONNECT_EVENT** – This event is shared for P2P GO and AP. Indicates that the P2P connection has ended successfully and that the device is P2P GO.
  - It contains:
    - Remote device parameters
    - Information about the group owner parameters

### 11.4 Use Cases and Configuration

This section describes common P2P use cases, their meanings, and how to configure them.
11.4.1 Case 1 – Nailed P2P Client Low-Power Profile

The device is a P2P client by automatic connection, using fast connection after each reset or disconnection. The device stores the parameters of the last connection.

At least one profile should be configured in the system.

On fast connect, add the device to P2P supplicant to avoid find.

Send P2P-Invite to remote persistent GO with network parameters.

If fast connection failed, fall back to profile P2P find and perform the regular connection method (join or full negotiation).

1. Configure P2P global parameters.
2. Configure P2P profile policy with SL_P2P_ROLE_CLIENT (intent 0), and with
   SL_P2P_NEG_INITIATOR_RAND_BACKOFF (negotiation initiator don't care). SL_P2P_POLICY
   (SL_P2P_ROLE_CLIENT, SL_P2P_NEG_INITIATOR_RAND_BACKOFF)
3. Configure P2P profile policy connection with auto-start and fast connection.
   SL_CONNECTION_POLICY(true, true, false, false)
4. Configure P2P profile according to the willing parameters of the remote device such as name, MAC address, and so forth.
5. If the peer parameters are unknown, operate a discover P2P operation, find P2P peers, and then configure profiles.

11.4.2 Case 2 – Mobile Client Low-Power Profile

The device is a P2P client by automatic connection, with no fast connection after each reset or disconnection. The device does not store the parameters of the last connection.

At least one profile should be configured in the system.

As in the first case, perform a profile P2P find with stored networks, examine the results, and send a negotiation request according to remote device GO capabilities (group owner).

1. Configure P2P global parameters.
2. Configure P2P profile policy with SL_P2P_ROLE_CLIENT (intent 0), and with
   SL_P2P_NEG_INITIATOR_RAND_BACKOFF (negotiation initiator don't care).
   SL_P2P_POLICY(SL_P2P_ROLE_CLIENT, SL_P2P_NEG_INITIATOR_RAND_BACKOFF)
3. Configure P2P profile policy connection with auto-start and fast connection.
   SL_CONNECTION_POLICY(true, false, false, false)
4. Configure P2P profile according to the willing parameters of the remote device such as name, MAC address, and so forth.
5. If the peer parameters are unknown, operate a discover P2P operation, find P2P peers, and then configure profiles.

11.4.3 Case 3 – Nailed Center Plugged-in Profile

The device is a P2P GO by automatic connection, with fast connection after each reset or disconnection. The device stores the parameters of the last connection.

At least one profile should be configured in the system.

• HARD RESET ONLY – Launch Group (as GO) with previous persistent network parameters stored in fast-connect, set WPS method and set timer.
• PEER DISCONNECT – Set WPS method and set timer.
• WPS timer expired with no peers connected – Tear down group, go back and perform a profile P2P find with stored network. Examine the results and initiate a negotiation request with the policy parameters (negotiate with intent = 15 results as GO).

1. Configure P2P global parameters.
2. Configure P2P profile policy with SL_P2P_ROLE_GROUP_OWNER (intent 15), and with
   SL_P2P_NEG_INITIATOR_RAND_BACKOFF (negotiation initiator don't care).
   SL_P2P_POLICY(SL_P2P_ROLE_GO, SL_P2P_NEG_INITIATOR_RAND_BACKOFF)
3. Configure P2P profile policy connection with auto-start and fast connection.
   SL_CONNECTION_POLICY(true, true, false, false)
4. Configure P2P profile according to the willing parameters of the remote device such as name, MAC
   address, and so forth.
5. If the peer parameters are unknown, operate a discover P2P operation, find P2P peers, and then
   configure profiles.

### 11.4.4 Case 4 – Mobile Center Profile

Perform a group formation and launch as GO (nonpersistent) by automatic connection with no fast
connection after each reset or disconnection. The device does not store the parameters of the last
connection. At least one profile should be configured in the system.

- DISCONNECT – Tear down the group immediately.
- Hard reset or after tearing down – Perform a profile P2P find with the stored network, examine the
  results, and initiate a negotiation request with the policy parameters (negotiate with intent = 15 results
  as GO).

1. Configure P2P global parameters.
2. Configure P2P profile policy with SL_P2P_ROLE_GROUP_OWNER (intent 15), and with
   SL_P2P_NEG_INITIATOR_RAND_BACKOFF (negotiation initiator don't care).
   SL_P2P_POLICY(SL_P2P_ROLE_GO, SL_P2P_NEG_INITIATOR_RAND_BACKOFF)
3. Configure P2P profile policy connection with auto-start and fast connection.
   SL_CONNECTION_POLICY(true, true, false, false)
4. Configure P2P profile according to the willing parameters of the remote device such as name, MAC
   address, and so forth.
5. If the peer parameters are unknown, operate a discover P2P operation, find P2P peers, and then
   configure profiles.

### 11.4.5 Case 5 – Mobile General-Purpose Profile

Perform a group formation and launch as GO (nonpersistent) or CL, without storing any parameter. At
least one profile should be configured in the system.

- DISCONNECT – If GO, then tear down the group immediately.
- Hard reset or after tearing down – Perform a profile P2P find with the stored network, examine the
  results, and initiate a negotiation request with the policy parameters (negotiate with intent = 7 results
  as GO or CL).

1. Configure P2P global parameters.
2. Configure P2P profile policy with SL_P2P_ROLE_NEGOTIATE (intent 7), and with
   SL_P2P_NEG_INITIATOR_RAND_BACKOFF (negotiation initiator don't care).
   SL_P2P_POLICY(SL_P2P_ROLE_GO, SL_P2P_NEG_INITIATOR_RAND_BACKOFF)
3. Configure P2P profile policy connection with auto-start and fast connection.
   SL_CONNECTION_POLICY(true, false, false, false)
4. Configure P2P profile according to the willing parameters of the remote device such as name, MAC
   address, and so forth.
5. If the peer parameters are unknown, operate a discover P2P operation, find P2P peers, and then
   configure profiles.

### 11.5 Example Code

The following is sample code of a simple profile connection, configuring the device to act as a P2P client
device using a profile connection with a known remote GO device name (without MAC), using the push-
button method.
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_DEV_CONFIG_ID,
          NETAPP_SET_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);
## Chapter 12

### HTTP Server

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>12.1 Overview</td>
<td>78</td>
</tr>
<tr>
<td>12.2 Supported Features</td>
<td>78</td>
</tr>
<tr>
<td>12.3 HTTP Web Server Description</td>
<td>79</td>
</tr>
<tr>
<td>12.4 HTTP GET Processing</td>
<td>81</td>
</tr>
<tr>
<td>12.5 HTTP POST Processing</td>
<td>82</td>
</tr>
<tr>
<td>12.6 Internal Web Page</td>
<td>83</td>
</tr>
<tr>
<td>12.7 Force AP Mode Support</td>
<td>84</td>
</tr>
<tr>
<td>12.8 Accessing the Web Page</td>
<td>84</td>
</tr>
<tr>
<td>12.9 HTTP Authentication Check</td>
<td>84</td>
</tr>
<tr>
<td>12.10 Handling HTTP Events in Host Using the SimpleLink Driver</td>
<td>85</td>
</tr>
<tr>
<td>12.11 SimpleLink Driver Interface the HTTP Web Server</td>
<td>86</td>
</tr>
<tr>
<td>12.12 SimpleLink Predefined Tokens</td>
<td>90</td>
</tr>
</tbody>
</table>
12.1 Overview

The HTTP web server allows end-users to remotely communicate with the SimpleLink device using a standard web browser.

The HTTP web server enables the following functions:

- Device configuration
- Device status and diagnostic
- Application-specific functionality

HTTP is a client/server protocol used to deliver hypertext resources (HTML web pages, images, query results, and so forth) to the client side. HTTP works on top of a predefined TCP/IP socket, usually port 80.

```
GET /page.html HTTP/1.0
Host: host_name
```

![Figure 12-1. HTTP GET Request](image)

12.2 Supported Features

- HTTP version support: 1.0
- HTTP requests: GET, POST
- Supported file types: .html, .htm, .css, .xml, .png, .gif
- HTML form submission of data is using POST method
- Supported Content-Type of POST request: 'application/x-www-form-urlencoded'
- HTTP Port number can be configured – default is port 80
- HTTP web server authentication
  - Can be enabled/disabled (Disabled by default)
  - Authentication name, password and realm are configurable
- SimpleLink Domain name (in AP mode) can be configured
- Built-in default page that provide device configuration, status and analyzing tools (with zero configuration from the user side…)
- Option to set device configuration as part of user provided pages
• For security purposes, the web server can access only the following root folders on the file system:
  – www/
  – www/safe/
• ‘Force AP’ support mode – can only access the following root folders on the folder system in this mode
  – The server will permit access only to the ‘www/safe/’ folder in the file system.
  – Special ‘Clear all Profiles’ button in the internal configuration web pages.
• Default values:
  – Domain name: ‘www.mysimplelink.net’ or ‘mysimplelink.net’
  – Authentication Name: admin
  – Authentication Password: admin
  – Authentication Realm: Simple Link CC31xx

12.3 HTTP Web Server Description

12.3.1 Overview

The Simple link HTTP web server is an embedded network application running on the Network processor, which runs on top of the TCP stack.

The Simplelink HTTP web server listens on a pre-configured TCP/IP socket (default is 80) and waits for the HTTP requests from a client. Once an HTTP request arrives, the server processes it and responds with the proper HTTP response.
12.3.1.1 Details

The Simple link server is responsible to implement the handling of the HTTP request. It listens on the HTTP socket (default is 80) and according to the request type, i.e. HTTP GET or HTTP POST, it handles the request URI resource and content. It then composes a proper HTTP response and sends it back to the client.

The simple link server communicates with the serial flash file system, which hosts the web page files. The files are saved in the serial flash under their own file name. File names can include full path, in order to achieve a directory structure like behavior.

For security purposes, the web server’s access to the file system is limited to the following root folders:

• www/
• www/safe/

**NOTE:** When downloading files to the file system, one of the above root folders should be prefixed to the file name. The ‘www/safe/’ folder is also used in ‘Force AP’ mode. Details can be found later in the document.
12.4 HTTP GET Processing

12.4.1 Overview

When the HTTP web server gets an HTTP GET request, it first checks the resource URN of the HTTP request for the name of the requested resource. The server then checks if this resource exists in the serial flash. If the resource exists, the server returns it as part of the HTTP response.

If the server does not find the requested resource on the flash, the server checks if this resource is one of the files of the internal web page in the ROM. If the resource is a file of the web page, the server returns the resource; if not, an HTTP error message is sent (HTTP/1.0 404 Not Found response).

12.4.2 Default Web Page

In case the HTTP GET does not contain any resource name (for example, /), the HTTP web server looks for the following filenames in this order:

- Index.html
- Main.html

The server first checks them on the serial flash, and then in the internal web page.

12.4.3 SimpleLink GET Tokens

To support HTML pages with data that is generated dynamically by the server, the HTTP web server supports a set of predefined tokens replaced on-the-fly by the server, with dynamically generated content.

- The tokens have a fixed length of 10 characters.
- The token prefix is identical to all tokens and seven characters in length.
- The token prefix is: __SL_G_

- A typical token should look like this: __SL_G_XYZ where XYZ is one of the predefined tokens.

As a consequence of a GET request, the HTTP server will scan the HTML page for the ‘__SL_G_’ prefix. If it finds a prefix, it checks the complete token. Once it matches a known token, it replaces the token in the HTML with the appropriate data (string) that matches that token (for example, '__SL_G_N.A' will be replaced by STA IP address and '__SL_G_V.A' will be replaced by the NWP version). This string is eventually displayed by the browser, instead of displaying the token.

For a complete list of the predefined tokens, see Section 12.12.

12.4.4 User-Defined Tokens

The user can define new tokens that are not known by the HTTP web server.

The token should follow the same rules as the predefined tokens:

- The tokens have a fixed length of 10 characters.
- The token prefix is identical to all tokens and seven characters in length.
- The token prefix is: __SL_G_

- A typical token should look like this: __SL_G_XYZ where XYZ is user-defined and can contain any character or number.

If the HTTP web server scans the HTML file and finds a token that is not in the predefined list, the server generates a get_token_value asynchronous event with the token name, requesting the token value from the host.

The host should respond with a send_token_value command, with the token value. The HTTP web server will use this token value and return it to the client.

Note: The maximum length of the token value is 64 bytes.
If the host is not responding to the `get_token_value` request, the server implements a time-out of two seconds. After the time-out, the Not Available string is used as the token value (and is eventually be displayed by the browser).

---

**NOTE:** To prevent user tokens from colliding with the internal tokens, use tokens with the following structure: `__SL_G_UXX`, where XX can be any character or number.

Description of the host interface can be found in paragraph: Host / HTTP web server API.

### 12.4.5 HTML Sample Code With Dynamic HTML Content

In the following code the token `__SL_G_N.A` will be replaced by the actual IP address:

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

### 12.5 HTTP POST Processing

#### 12.5.1 Overview

The client uses HTTP POST requests to update data in the server. The SimpleLink HTTP web server supports HTML forms with content type of application/x-www-form-urlencoded. The POST information that is sent by the browser includes the form action name and one or more pairs of variable name and variable value.

#### 12.5.2 SimpleLink POST Tokens

In SimpleLink, the variable name in the POST should follow the same rules of the GET tokens.
- The tokens have a fixed length of 10 characters.
- The token prefix is identical to all tokens and seven characters in length.
- The token prefix is: `__SL_P_`
- A typical token should look like this: `__SL_P_XYZ` where XYZ is user-defined and can contain any character or number.

When the HTTP web server receives an HTTP POST request, the server first checks the form action name to understand if this POST should be handled internally. The server then goes over the parameters list, and checks each variable name to see if it matches one of the known predefined tokens. If the variable names match the predefined tokens, the server processes the values.

#### 12.5.3 SimpleLink POST Actions

There are two types of POST operations: simple actions and complex actions.

In simple POST actions, the server processes the list of POST parameters and saves the new information (such as set domain name). In this case, the action value is not important and the server does not identify it.

In complex POST actions, the server should gather all the needed POST parameters and then trigger a specific action (such as add profile). In this case, the action is identified by the action value in the HTTP POST command.

#### 12.5.4 User-Defined Tokens

User-defined tokens are used in POST requests to send information to the host.
If the HTTP web server receives an HTTP POST request that contains tokens that are not in the predefined list, the server generates a post_token_value asynchronous event to the host, which contains the following information: form action name, token name, and token value. The host can then process the required information.

**NOTE:** To prevent user tokens from colliding with the internal tokens, use tokens with the following structure: __SL_P_UXX, where XX can be any character or number.

### 12.5.5 Redirect after POST

In SimpleLink, the user redirects the browser to a different web page after the POST submission. After the POST is processed, the HTTP web server checks the action-URI received in the POST request. If the action-URI includes a valid web page in the SimpleLink (Serial Flash or ROM), the server issues an HTTP 302 Found response with the action-URI value, to perform redirection. If the action-URI does not contain a valid web page, the HTTP web server issues an HTTP 204 No content response and the browser remains on the current web page.

### 12.5.6 HTML Sample Code With POST and Dynamic HTML Content

In the following POST example, once the user clicks on the submit button the POST request includes the profiles_add.html as the action resource and the variables __SL_P_P.A and __SL_P_P.B with the values that the user requested.

```html
<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<br/>
<input type="radio" name="__SL_P_P.B" value="1" />WEP<br/>
<input type="radio" name="__SL_P_P.B" value="2" />WPA1<br/>
<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>
```

When the page is displayed in the following example (HTTP GET), the __SL_G_N.A is replaced by the HTTP web server with the current IP address value, displayed in the input box. When the user changes and submits the IP address, the new value is sent with the __SL_P_N.A variable.

```html
<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>
```

### 12.6 Internal Web Page

The SimpleLink device has a default web page already embedded in ROM. This web page can be used to perform the following:
- Get versions and general information about the device
- IP configuration
12.7 Force AP Mode Support

The Force AP mode returns the SimpleLink to its default configuration. This mode is entered with a special external GPIO.

When the SimpleLink enters Force AP mode, the HTTP server behaves as follows:

- The server only permits access to the www/safe/ folder in the file system. This lets the user put a set of web pages in the www/ folder that will not be accessible in Force AP mode.
- In this mode, a Clear all Profiles button appears in the internal configuration web pages, enabling the user to clear all the saved profiles from the device.

12.8 Accessing the Web Page

12.8.1 SimpleLink in Station Mode

When the SimpleLink is in station mode, the user can access the web page from the browser using the IP address. The HTTP service is also published by the mDNS server, so the IP address can be acquired from the mDNS publications.

12.8.2 SimpleLink in AP Mode

When in AP mode, access to the web page is performed using the domain name. The domain name is configured with the host API.

In AP mode, the SimpleLink is also accessed using the IP address.

The default domain name is mysimplelink.net.

Accessing the web page can be performed with either mysimplelink.net or www.mysimplelink.net.

Using the www. prefix in the domain name when the domain name does not match, the server internally removes this prefix and tries to search without it.

NOTE: Use the www. prefix in the domain name search because commercial browsers behave better from a DNS perspective.

12.9 HTTP Authentication Check

If enabled, the SimpleLink performs an authentication check when the client first connects to the server. The authentication check can be enabled or disabled using the host API.
Authentication user name, password, and realm can also be configured by the host API. By default, the authentication is disabled. The default authentication values are:

- Authentication name: admin
- Authentication password: admin
- Authentication realm: Simple Link CC31xx

A description of the host interface can be found in paragraph: Chapter 12.

12.10 Handling HTTP Events in Host Using the SimpleLink Driver

When the HTTP server locates user tokens in the HTML files, the server generates get_token_value (for GET tokens) or post_token_value (for post tokens) events to the host for the user to correctly handle them.

When the host gets a get_token_value event with a specific token name, the server returns the token value for this token name by using the send_token_value command.

If the host does not have any token value to return, the server uses zero as the length of the token value.

When the user gets a post_token_value event with the token name and value, the user must save this new token value.

In the SimpleLink driver, when one of the preceding events is generated the driver calls a predefined callback called `SimpleLinkHttpServerCallback();`

The callback is defined as follows:

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

Where serverEvent and serverResponse are defined as follows:

```c
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;
```

The following is user callback sample code:

```c
/*HTTP Server Callback example */
```
void SimpleLinkHttpServerCallback(SlHttpServerEvent_t *pHttpServerEvent,
    SlHttpServerResponse_t *pHttpServerResponse)
{
    switch (pHttpServerEvent->Event)
    {
    /* Handle Get token value */
    case SL_NETAPP_HTTPGETTOKENVALUE:
        char * tokenValue;
        
        tokenValue = GetTokenValue (pHttpServerEvent >EventData.httpTokenName);

        /* Response using driver memory - Copy the token value to the Event response
        Important - Token value len should be < MAX_TOKEN_VALUE_LEN (64 bytes) */
        strcpy (pHttpServerResponse->ResponseData.token_value.data, tokenValue);
        pHttpServerResponse->ResponseData.token_value.len = strlen (tokenValue);
        break;

    /* Handle Post token */
    case SL_NETAPP_HTTPPOSTTOKENVALUE:
        HandleTokenPost (pHttpServerEvent->EventData.httpPostData.action,
            pHttpServerEvent->EventData.httpPostData.token_name,
            pHttpServerEvent->EventData.httpPostData.token_value);
        break;
    default:
        break;
    }
}

NOTE: For the HTTP callback to work the following line should be uncommented in user.h:
#define sl_HttpServerCallback SimpleLinkHttpServerCallback

12.11 SimpleLink Driver Interface the HTTP Web Server

The SimpleLink driver supplies an API to access and configure the HTTP server.
The API definition can be found in netapp.h.

12.11.1 Enable or Disable HTTP Server

Functions used for enabling or disabling the HTTP server. By default, the server is enabled.

Table 12-1. Enable or Disable HTTP Server

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>void sl_NetAppStart(UINT32 appld)</td>
<td>Starts the HTTP server</td>
<td>appld = SL_NET_APP_HTTP_SERVER_ID</td>
</tr>
<tr>
<td>void sl_NetAppStop(UINT32 appld)</td>
<td>Stops the HTTP server</td>
<td>appld = SL_NET_APP_HTTP_SERVER_ID</td>
</tr>
</tbody>
</table>

12.11.2 Configure HTTP Port Number

Functions used for configuring the port number.
Table 12-2. Configure HTTP Port Number

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>long sl_NetAppSet (unsigned char appId,</td>
<td>Sets the port number on which the HTTP server will listen.</td>
<td>appId = SL_NET_APP_HTTP_SERVER_ID&lt;br&gt;Option = NETAPP_SET_GET_HTTP_OPT_PORT_NUMBER&lt;br&gt;pOptionLen = 2 (size of UINT16)&lt;br&gt;pOptionValue = pointer to port number (UINT16)</td>
</tr>
<tr>
<td>unsigned char Option,</td>
<td>Port number is UINT16.</td>
<td></td>
</tr>
<tr>
<td>unsigned char OptionLen,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>unsigned char *pOptionValue)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>long sl_NetAppGet (unsigned char appId,</td>
<td>Gets the current HTTP server port number.</td>
<td>appId = SL_NET_APP_HTTP_SERVER_ID&lt;br&gt;Option = NETAPP_SET_GET_HTTP_OPT_PORT_NUMBER&lt;br&gt;pOptionLen = user-supplied pointer for the returned value len&lt;br&gt;pOptionValue = user-supplied pointer for the returned value</td>
</tr>
<tr>
<td>unsigned char Option,</td>
<td>Port number is UINT16.</td>
<td></td>
</tr>
<tr>
<td>unsigned char *pOptionLen,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>unsigned char *pOptionValue)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

NOTE: After setting a new port number, restart the server for the configuration to occur.

12.11.3 Enable or Disable Authentication Check

Functions used for enabling or disabling authentication check. By default, authentication check is disabled.

Table 12-3. Enable or Disable Authentication Check

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>long sl_NetAppSet (unsigned char appId,</td>
<td>Enables or disables the HTTP server authentication check</td>
<td>appId = SL_NET_APP_HTTP_SERVER_ID&lt;br&gt;Option = NETAPP_SET_GET_HTTP_OPT_AUTH_CHECK&lt;br&gt;pOptionLen = 1 (size of UINT8)&lt;br&gt;pOptionValue = pointer to auth_enable (true/false)</td>
</tr>
<tr>
<td>unsigned char Option,</td>
<td>Auth_enable value is true/false.</td>
<td></td>
</tr>
<tr>
<td>unsigned char OptionLen,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>unsigned char *pOptionValue)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>long sl_NetAppGet (unsigned char appId,</td>
<td>Gets the current authentication status. Return auth_enable value is true/false.</td>
<td>appId = SL_NET_APP_HTTP_SERVER_ID&lt;br&gt;Option = NETAPP_SET_GET_HTTP_OPT_AUTH_CHECK&lt;br&gt;pOptionLen = user-supplied pointer for the returned value len&lt;br&gt;pOptionValue = user-supplied pointer for the returned value</td>
</tr>
<tr>
<td>unsigned char Option,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>unsigned char *pOptionLen,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>unsigned char *pOptionValue)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

12.11.4 Set or Get Authentication Name, Password, and Realm

Functions used to set or get the authentication name, password, and realm.
### Table 12-4. Set or Get Authentication Name

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>long sl_NetAppSet (unsigned char appId ,</td>
<td>Sets authentication name.</td>
<td>appId = SL_NET_APP_HTTP_SERVER_ID</td>
</tr>
<tr>
<td>unsigned char Option,</td>
<td>Name format is string.</td>
<td>Option = NETAPP_SET_GET_HTTP_OPT_AUTH_NAME</td>
</tr>
<tr>
<td>unsigned char OptionLen,</td>
<td></td>
<td>OptionLen = authentication name length</td>
</tr>
<tr>
<td>unsigned char *pOptionValue)</td>
<td></td>
<td>pOptionValue = pointer to authentication name</td>
</tr>
<tr>
<td>long sl_NetAppGet (unsigned char appId,</td>
<td>Gets current authentication name.</td>
<td>appId = SL_NET_APP_HTTP_SERVER_ID</td>
</tr>
<tr>
<td>unsigned char Option,</td>
<td>Name format is string (not null terminated).</td>
<td>Option = NETAPP_SET_GET_HTTP_OPT_AUTH_NAME</td>
</tr>
<tr>
<td>unsigned char *pOptionLen,</td>
<td></td>
<td>pOptionLen = user-supplied pointer for the returned name len</td>
</tr>
<tr>
<td>unsigned char *pOptionValue)</td>
<td></td>
<td>pOptionValue = user-supplied pointer for the returned name</td>
</tr>
</tbody>
</table>

### Table 12-5. Set or Get Authentication Password

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>long sl_NetAppSet (unsigned char appId ,</td>
<td>Sets authentication password.</td>
<td>appId = SL_NET_APP_HTTP_SERVER_ID</td>
</tr>
<tr>
<td>unsigned char Option,</td>
<td>Password format is string.</td>
<td>Option = NETAPP_SET_GET_HTTP_OPT_AUTH_PASSWORD</td>
</tr>
<tr>
<td>unsigned char OptionLen,</td>
<td></td>
<td>OptionLen = authentication password length</td>
</tr>
<tr>
<td>unsigned char *pOptionValue)</td>
<td></td>
<td>pOptionValue = pointer to authentication password</td>
</tr>
<tr>
<td>long sl_NetAppGet (unsigned char appId,</td>
<td>Gets current authentication password.</td>
<td>appId = SL_NET_APP_HTTP_SERVER_ID</td>
</tr>
<tr>
<td>unsigned char Option,</td>
<td>Password format is string (not null terminated).</td>
<td>Option = NETAPP_SET_GET_HTTP_OPT_AUTH_PASSWORD</td>
</tr>
<tr>
<td>unsigned char *pOptionLen,</td>
<td></td>
<td>pOptionLen = user-supplied pointer for the returned password len</td>
</tr>
<tr>
<td>unsigned char *pOptionValue)</td>
<td></td>
<td>pOptionValue = user-supplied pointer for the returned password</td>
</tr>
</tbody>
</table>

### Table 12-6. Set or Get Authentication Realm

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>long sl_NetAppSet (unsigned char appId,</td>
<td>Sets authentication realm.</td>
<td>appId = SL_NET_APP_HTTP_SERVER_ID</td>
</tr>
<tr>
<td>unsigned char Option,</td>
<td>Realm format is string.</td>
<td>Option = NETAPP_SET_GET_HTTP_OPT_AUTH_REALM</td>
</tr>
<tr>
<td>unsigned char OptionLen,</td>
<td></td>
<td>OptionLen = authentication realm length</td>
</tr>
<tr>
<td>unsigned char *pOptionValue)</td>
<td></td>
<td>pOptionValue = pointer to authentication realm</td>
</tr>
<tr>
<td>long sl_NetAppGet (unsigned char appId,</td>
<td>Gets current authentication realm.</td>
<td>appId = SL_NET_APP_HTTP_SERVER_ID</td>
</tr>
<tr>
<td>unsigned char Option,</td>
<td>Realm format is string (not null terminated).</td>
<td>Option = NETAPP_SET_GET_HTTP_OPT_AUTH_REALM</td>
</tr>
<tr>
<td>unsigned char *pOptionLen,</td>
<td></td>
<td>pOptionLen = user-supplied pointer for the returned realm len</td>
</tr>
<tr>
<td>unsigned char *pOptionValue)</td>
<td></td>
<td>pOptionValue = user-supplied pointer for the returned realm</td>
</tr>
</tbody>
</table>
### 12.11.5 Set or Get Domain Name

Functions used to set or get the domain name (used for accessing the web server in AP mode).

**Table 12-7. Set or Get Domain Name**

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>long sl_NetAppSet (unsigned char appId, unsigned char Option, unsigned char OptionLen, unsigned char *pOptionValue)</code></td>
<td>Sets the domain name. Domain name format is string.</td>
<td><code>appId = SL_NET_APP_DEVICE_CONFIG_ID</code>&lt;br&gt;<code>Option = NETAPP_SET_GET_DEV_CONF_OPT_DOMAIN_NAME</code>&lt;br&gt;<code>OptionLen = domain name length</code>&lt;br&gt;<code>pOptionValue = pointer to domain name</code></td>
</tr>
<tr>
<td><code>long sl_NetAppGet (unsigned char appId, unsigned char Option, unsigned char *pOptionLen, unsigned char *pOptionValue)</code></td>
<td>Gets current domain name. Domain name format is string (not null terminated).</td>
<td><code>appId = SL_NET_APP_DEVICE_CONFIG_ID</code>&lt;br&gt;<code>Option = NETAPP_SET_GET_DEV_CONF_OPT_DOMAIN_NAME</code>&lt;br&gt;<code>pOptionLen = user-supplied pointer for the returned domain name len</code>&lt;br&gt;<code>pOptionValue = user-supplied pointer for the returned domain name</code></td>
</tr>
</tbody>
</table>

### 12.11.6 Set or Get URN Name

Functions used to set or get the device unique URN name.

**Table 12-8. Set or Get URN Name**

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>long sl_NetAppSet (unsigned char appId, unsigned char Option, unsigned char OptionLen, unsigned char *pOptionValue)</code></td>
<td>Sets the URN name. URN name format is string.</td>
<td><code>appId = SL_NET_APP_DEVICE_CONFIG_ID</code>&lt;br&gt;<code>Option = NETAPP_SET_GET_DEV_CONF_OPT_DEVICE_URN</code>&lt;br&gt;<code>OptionLen = URN name length</code>&lt;br&gt;<code>pOptionValue = pointer to URN name</code></td>
</tr>
<tr>
<td><code>long sl_NetAppGet (unsigned char appId, unsigned char Option, unsigned char *pOptionLen, unsigned char *pOptionValue)</code></td>
<td>Gets current URN name. URN name format is string (not null terminated).</td>
<td><code>appId = SL_NET_APP_DEVICE_CONFIG_ID</code>&lt;br&gt;<code>Option = NETAPP_SET_GET_DEV_CONF_OPT_DEVICE_URN</code>&lt;br&gt;<code>pOptionLen = user-supplied pointer for the returned URN name len</code>&lt;br&gt;<code>pOptionValue = user-supplied pointer for the returned URN name</code></td>
</tr>
</tbody>
</table>

### 12.11.7 Enable or Disable ROM Web Pages Access

Functions used to enable or disable the access to the ROM internal web pages. By default, web access is enabled.
Table 12-9. Enable or Disable ROM Web Pages Access

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>long sl_NetAppSet (unsigned char appId, unsigned char Option, unsigned char OptionLen, unsigned char *pOptionValue)</td>
<td>Enables or disables the HTTP server ROM pages access. Value is true or false.</td>
<td>appld = SL_NET_APP_HTTP_SERVER_ID Option = NETAPP_SET_GET_HTTP_OPT_ROM_PAGES_ACCESS OptionLen = 1 (sizeof UINT8) pOptionValue = pointer to boolean (true/false)</td>
</tr>
<tr>
<td>long sl_NetAppGet (unsigned char appId, unsigned char Option, unsigned char *pOptionLen, unsigned char *pOptionValue)</td>
<td>Gets the current ROM pages access status. Return value is true or false.</td>
<td>appld = SL_NET_APP_HTTP_SERVER_ID Option = NETAPP_SET_GET_HTTP_OPT_ROM_PAGES_ACCESS pOptionLen = user-supplied pointer for the returned value len pOptionValue = user-supplied pointer for the returned value</td>
</tr>
</tbody>
</table>

12.12 SimpleLink Predefined Tokens

This section contains information about the predefined internal tokens, and describes the tokens for the GET operations, POST operations, and the POST actions handled internally.

**NOTE:** To prevent user tokens from colliding with the internal tokens, use tokens with the following structure:
- For GET operations: __SL_G_UXX, where XX can be any character or number.
- For POST operations: __SL_P_UXX, where XX can be any character or number.

12.12.1 GET Values

GET system information values are shown in Table 12-10.

Table 12-10. System Information

<table>
<thead>
<tr>
<th>Token</th>
<th>Name</th>
<th>Value / Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>__SL_G_S.A</td>
<td>System Up Time</td>
<td>Returns the system up time since the last reset in the following format: 000 days 00:00:00</td>
</tr>
<tr>
<td>__SL_G_S.B</td>
<td>Device name (URN)</td>
<td>Returns the device name</td>
</tr>
<tr>
<td>__SL_G_S.C</td>
<td>Domain name</td>
<td>Returns the domain name</td>
</tr>
<tr>
<td>__SL_G_S.D</td>
<td>Device mode (role)</td>
<td>Returns the device role. Values: Station, Access Point, P2P</td>
</tr>
<tr>
<td>__SL_G_S.E</td>
<td>Device role station</td>
<td>Drop-down menu select/not select Returns “selected” if the device is a station, otherwise it returns “not_selected.”</td>
</tr>
<tr>
<td>__SL_G_S.F</td>
<td>Device role AP</td>
<td>Drop-down menu select/not select Returns “selected” if the device is an AP, otherwise it returns “not_selected.”</td>
</tr>
<tr>
<td>__SL_G_S.G</td>
<td>Device role P2P</td>
<td>Drop-down menu select/not select Returns “selected” if the device is in P2P, otherwise it returns “not_selected.”</td>
</tr>
<tr>
<td>__SL_G_S.H</td>
<td>Device name URN (truncated to 16 bytes)</td>
<td>Returns the URN string name with up to 16-byte len. Longer names are truncated.</td>
</tr>
</tbody>
</table>
## Table 12-10. System Information (continued)

<table>
<thead>
<tr>
<th>Token</th>
<th>Name</th>
<th>Value / Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>__SL_G_S.I</td>
<td>System requires reset (after parameters change)</td>
<td>If the system requires a reset, the return value is the following string: -- Some parameters were changed, System may require reset --, otherwise it returns an empty string. Every internal POST that was handled will cause this token to return TRUE.</td>
</tr>
<tr>
<td>__SL_G_S.J</td>
<td>Get system time and date</td>
<td>Returned value is a string with the following format: Year, month, day, hours, minutes, seconds</td>
</tr>
<tr>
<td>__SL_G_S.K</td>
<td>Safe mode status</td>
<td>If the device is in safe mode it returns “Safe Mode”, if not it returns an empty string.</td>
</tr>
</tbody>
</table>

### GET version information is shown in Table 12-11.

## Table 12-11. Version Information

<table>
<thead>
<tr>
<th>Token</th>
<th>Name</th>
<th>Value / Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>__SL_G_V.A</td>
<td>NWP version</td>
<td>Returns string with the version information</td>
</tr>
<tr>
<td>__SL_G_V.B</td>
<td>MAC version</td>
<td>—</td>
</tr>
<tr>
<td>__SL_G_V.C</td>
<td>PHY version</td>
<td>—</td>
</tr>
<tr>
<td>__SL_G_V.D</td>
<td>HW version</td>
<td>—</td>
</tr>
</tbody>
</table>

### GET network information is shown in Table 12-12.

## Table 12-12. Network Information

<table>
<thead>
<tr>
<th>Token</th>
<th>Name</th>
<th>Value / Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>__SL_G_N.A</td>
<td>STA IP address</td>
<td>String format: xxx.yyy.zzz.ttt</td>
</tr>
<tr>
<td>__SL_G_N.B</td>
<td>STA subnet mask</td>
<td>—</td>
</tr>
<tr>
<td>__SL_G_N.C</td>
<td>STA default gateway</td>
<td>—</td>
</tr>
<tr>
<td>__SL_G_N.D</td>
<td>MAC address</td>
<td>String format: 00:11:22:33:44:55</td>
</tr>
<tr>
<td>__SL_G_N.E</td>
<td>STA DHCP state</td>
<td>Returns value: Enabled or Disabled</td>
</tr>
<tr>
<td>__SL_G_N.F</td>
<td>STA DHCP disable state</td>
<td>If DHCP is disabled, returns Checked, otherwise it returns Not_Checked. Used in the DHCP radio button.</td>
</tr>
<tr>
<td>__SL_G_N.G</td>
<td>STA DHCP enable state</td>
<td>If DHCP is enabled, returns Checked, otherwise it returns Not_Checked. Used in the DHCP radio button.</td>
</tr>
<tr>
<td>__SL_G_N.H</td>
<td>STA DNS server</td>
<td>String format: xxx.yyy.zzz.ttt</td>
</tr>
<tr>
<td>__SL_G_N.I</td>
<td>DHCP server</td>
<td>—</td>
</tr>
<tr>
<td>__SL_G_N.J</td>
<td>DHCP start address</td>
<td>—</td>
</tr>
<tr>
<td>__SL_G_N.K</td>
<td>DHCP last address</td>
<td>—</td>
</tr>
<tr>
<td>__SL_G_N.L</td>
<td>DHCP lease time</td>
<td>String of the lease time in seconds</td>
</tr>
<tr>
<td>__SL_G_P.A</td>
<td>AP IP address</td>
<td>String format: xxx.yyy.zzz.ttt</td>
</tr>
<tr>
<td>__SL_G_W.A</td>
<td>Channel # in AP mode</td>
<td>Returned values: Open, WEP, WPA</td>
</tr>
<tr>
<td>__SL_G_W.B</td>
<td>SSID</td>
<td>—</td>
</tr>
<tr>
<td>__SL_G_W.C</td>
<td>Security type</td>
<td>—</td>
</tr>
</tbody>
</table>
Table 12-12. Network Information (continued)

<table>
<thead>
<tr>
<th>Token</th>
<th>Name</th>
<th>Value / Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>__SL_G_W.D</td>
<td>Security type Open</td>
<td>If the security type is open, it returns Checked, otherwise it returns Not_Checked. Used in the security type radio button check/not checked.</td>
</tr>
<tr>
<td>__SL_G_W.E</td>
<td>Security type WEP</td>
<td>If the security type is WEP, returns Checked, otherwise it returns Not_Checked. Used in the security type radio button check/not checked.</td>
</tr>
<tr>
<td>__SL_G_W.F</td>
<td>Security type WPA</td>
<td>If the security type is WPA, it returns Checked, otherwise it returns Not_Checked. Used in the security type radio button check/not checked.</td>
</tr>
</tbody>
</table>

GET tools are shown in Table 12-13.

Table 12-13. Tools

<table>
<thead>
<tr>
<th>Token</th>
<th>Name</th>
<th>Value / Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>__SL_G_T.A</td>
<td>IP address</td>
<td>String format: xxx.yyy.zzz.ttt</td>
</tr>
<tr>
<td>__SL_G_T.B</td>
<td>Packet size</td>
<td></td>
</tr>
<tr>
<td>__SL_G_T.C</td>
<td>Number of pings</td>
<td></td>
</tr>
<tr>
<td>__SL_G_T.D</td>
<td>Ping results – total sent</td>
<td>Number of total pings sent</td>
</tr>
<tr>
<td>__SL_G_T.E</td>
<td>Ping results – successful sent</td>
<td>Number of successful pings sent</td>
</tr>
<tr>
<td>__SL_G_T.E</td>
<td>Ping test duration</td>
<td>In seconds</td>
</tr>
</tbody>
</table>

GET connection policy status is shown in Table 12-14

Table 12-14. Connection Policy Status

<table>
<thead>
<tr>
<th>Token</th>
<th>Name</th>
<th>Value / Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>__SL_G_P.E</td>
<td>Auto connect</td>
<td>If auto connect is enabled, returns Checked, otherwise it returns Not_Checked. Used in the auto connect check box.</td>
</tr>
<tr>
<td>__SL_G_P.F</td>
<td>Fast connect</td>
<td>If fast connect is enabled, returns Checked, otherwise it returns Not_Checked. Used in the fast connect check box.</td>
</tr>
<tr>
<td>__SL_G_P.G</td>
<td>Any P2P</td>
<td>If any P2P is enabled, returns Checked, otherwise it returns Not_Checked. Used in the any P2P checkbox.</td>
</tr>
<tr>
<td>__SL_G_P.P</td>
<td>Auto SmartConfig</td>
<td>If auto SmartConfig is enabled, returns Checked, otherwise it returns Not_Checked. Used in the auto SmartConfig checkbox.</td>
</tr>
</tbody>
</table>

GET display profiles information is shown in Table 12-15.

Table 12-15. Display Profiles Information

<table>
<thead>
<tr>
<th>Token</th>
<th>Name</th>
<th>Value / Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>__SL_G_PN1</td>
<td>Return profile 1 SSID</td>
<td>SSID string</td>
</tr>
<tr>
<td>__SL_G_PN2</td>
<td>Return profile 2 SSID</td>
<td>—</td>
</tr>
</tbody>
</table>
### Table 12-15. Display Profiles Information (continued)

<table>
<thead>
<tr>
<th>Token</th>
<th>Name</th>
<th>Value / Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>_SL_G_PN3</td>
<td>Return profile 3 SSID</td>
<td></td>
</tr>
<tr>
<td>_SL_G_PN4</td>
<td>Return profile 4 SSID</td>
<td></td>
</tr>
<tr>
<td>_SL_G_PN5</td>
<td>Return profile 5 SSID</td>
<td></td>
</tr>
<tr>
<td>_SL_G_PN6</td>
<td>Return profile 6 SSID</td>
<td></td>
</tr>
<tr>
<td>_SL_G_PN7</td>
<td>Return profile 7 SSID</td>
<td></td>
</tr>
<tr>
<td>_SL_G_PS1</td>
<td>Return profile 1 security status</td>
<td>Returned values: Open, WEP, WPA, WPS, ENT, P2P_PBC, P2P_PIN or – for empty profile.</td>
</tr>
<tr>
<td>_SL_G_PS2</td>
<td>Return profile 2 security status</td>
<td></td>
</tr>
<tr>
<td>_SL_G_PS3</td>
<td>Return profile 3 security status</td>
<td></td>
</tr>
<tr>
<td>_SL_G_PS4</td>
<td>Return profile 4 security status</td>
<td></td>
</tr>
<tr>
<td>_SL_G_PS5</td>
<td>Return profile 5 security status</td>
<td></td>
</tr>
<tr>
<td>_SL_G_PS6</td>
<td>Return profile 6 security status</td>
<td></td>
</tr>
<tr>
<td>_SL_G_PS7</td>
<td>Return profile 7 security status</td>
<td></td>
</tr>
<tr>
<td>_SL_G_PP1</td>
<td>Return profile 1 priority</td>
<td>Profile priority: 0-7</td>
</tr>
<tr>
<td>_SL_G_PP2</td>
<td>Return profile 2 priority</td>
<td></td>
</tr>
<tr>
<td>_SL_G_PP3</td>
<td>Return profile 3 priority</td>
<td></td>
</tr>
<tr>
<td>_SL_G_PP4</td>
<td>Return profile 4 priority</td>
<td></td>
</tr>
<tr>
<td>_SL_G_PP5</td>
<td>Return profile 5 priority</td>
<td></td>
</tr>
<tr>
<td>_SL_G_PP6</td>
<td>Return profile 6 priority</td>
<td></td>
</tr>
<tr>
<td>_SL_G_PP7</td>
<td>Return profile 7 priority</td>
<td></td>
</tr>
</tbody>
</table>

GET P2P information is shown in Table 12-16.

### Table 12-16. P2P Information

<table>
<thead>
<tr>
<th>Token</th>
<th>Name</th>
<th>Value / Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>_SL_G_R.A</td>
<td>P2P Device name</td>
<td>String</td>
</tr>
<tr>
<td>_SL_G_R.B</td>
<td>P2P Device type</td>
<td>String</td>
</tr>
<tr>
<td>_SL_G_R.C</td>
<td>P2P Listen channel</td>
<td>Returns string of the listen channel number</td>
</tr>
<tr>
<td>_SL_G_R.T</td>
<td>Listen channel 1</td>
<td>If the current listen channel is 1, returns Selected, otherwise it returns Not_selected. Used for the drop-down menu of the listen channel.</td>
</tr>
<tr>
<td>_SL_G_R.U</td>
<td>Listen channel 6</td>
<td>If the current listen channel is 6, returns Selected, otherwise it returns Not_selected. Used for the drop-down menu of the listen channel.</td>
</tr>
<tr>
<td>_SL_G_R.V</td>
<td>Listen channel 11</td>
<td>If the current listen channel is 11, returns Selected, otherwise it returns Not_selected. Used for the drop-down menu of the listen channel.</td>
</tr>
<tr>
<td>_SL_G_R.E</td>
<td>P2P Operation channel</td>
<td>Returns string of the operational channel number</td>
</tr>
<tr>
<td>_SL_G_R.W</td>
<td>Operational channel 1</td>
<td>If the current operational channel is 1, returns Selected, otherwise it returns Not_selected. Used for the drop-down menu of the operational channel.</td>
</tr>
</tbody>
</table>
### Table 12-16. P2P Information (continued)

<table>
<thead>
<tr>
<th>Token</th>
<th>Name</th>
<th>Value / Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>__SL_G_R.X</td>
<td>Operational channel 6</td>
<td>If the current operational channel is 6, returns Selected, otherwise it returns Not_selected. Used for the drop-down menu of the operational channel.</td>
</tr>
<tr>
<td>__SL_G_R.Y</td>
<td>Operational channel 11</td>
<td>If the current operational channel is 11, returns Selected, otherwise it returns Not_selected. Used for the drop-down menu of the operational channel.</td>
</tr>
<tr>
<td>__SL_G_R.L</td>
<td>Negotiation intent value</td>
<td>Returned values: Group Owner, Negotiate, Client</td>
</tr>
<tr>
<td>__SL_G_R.M</td>
<td>Role group owner</td>
<td>If the intent is Group Owner, it returns Checked, otherwise it returns Not_Checked. Used for the negotiation intent radio button.</td>
</tr>
<tr>
<td>__SL_G_R.N</td>
<td>Role negotiate</td>
<td>If the intent is Negotiate, it returns Checked, otherwise it returns Not_Checked. Used for the negotiation intent radio button.</td>
</tr>
<tr>
<td>__SL_G_R.O</td>
<td>Role client</td>
<td>If the intent is Client, it returns Checked, otherwise it returns Not_Checked. Used for the negotiation intent radio button.</td>
</tr>
<tr>
<td>__SL_G_R.P</td>
<td>Negotiation initiator policy</td>
<td>Returned values: Active, Passive, Random Backoff</td>
</tr>
<tr>
<td>__SL_G_R.Q</td>
<td>Neg initiator active</td>
<td>If the negotiation initiator policy is Active, it returns Checked, otherwise it returns Not_Checked. Used for the negotiation initiator policy radio button.</td>
</tr>
<tr>
<td>__SL_G_R.R</td>
<td>Neg initiator passive</td>
<td>If the negotiation initiator policy is Passive, it returns Checked, otherwise it returns Not_Checked. Used for the negotiation initiator policy radio button.</td>
</tr>
<tr>
<td>__SL_G_R.S</td>
<td>Neg initiator random backoff</td>
<td>If the negotiation initiator policy is Random Backoff, it returns Checked, otherwise it returns Not_Checked. Used for the negotiation initiator policy radio button.</td>
</tr>
</tbody>
</table>

### 12.12.2 POST Values

POST system configuration is shown in **Table 12-17**.

### Table 12-17. System Configurations

<table>
<thead>
<tr>
<th>Token</th>
<th>Name</th>
<th>Value / Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>__SL_P_S.B</td>
<td>Device name (URN)</td>
<td>Sets device name</td>
</tr>
<tr>
<td>__SL_P_S.C</td>
<td>Domain name</td>
<td>Sets domain name</td>
</tr>
<tr>
<td>__SL_P_S.D</td>
<td>Device mode (role)</td>
<td>Sets device mode</td>
</tr>
<tr>
<td>__SL_P_S.D</td>
<td>Values: Station, AP, P2P</td>
<td></td>
</tr>
<tr>
<td>__SL_P_S.J</td>
<td>Post system time and date</td>
<td>Sets system time and date. The value is a string with the following format: Year, month, day, hours, minutes, seconds</td>
</tr>
</tbody>
</table>
Table 12-17. System Configurations (continued)

<table>
<thead>
<tr>
<th>Token</th>
<th>Name</th>
<th>Value / Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>__SL_P_S.R</td>
<td>Redirect after post</td>
<td>Value should contain a valid web page. If the page exists, the web server issues an HTTP 302 response to redirect to the web page. Can be used for redirection after submitting a form (with HTTP post).</td>
</tr>
</tbody>
</table>

POST network configurations are shown in Table 12-18.

Table 12-18. Network Configurations

<table>
<thead>
<tr>
<th>Token</th>
<th>Name</th>
<th>Values / Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>__SL_P_N.A</td>
<td>STA IP address</td>
<td>Sets STA IP address. Value format: xxx.yyy.zzz.ttt</td>
</tr>
<tr>
<td>__SL_P_N.B</td>
<td>STA subnet mask</td>
<td>Sets STA subnet mask. Value format: xxx.yyy.zzz.ttt</td>
</tr>
<tr>
<td>__SL_P_N.C</td>
<td>STA default gateway</td>
<td>Sets STA default gateway. Value format: xxx.yyy.zzz.ttt</td>
</tr>
<tr>
<td>__SL_P_N.D</td>
<td>STA DHCP state (must be disabled for the IP setting to take effect)</td>
<td>Enables or disables DHCP state. If value is Enable, then DHCP is enabled, any other value disables the DHCP.</td>
</tr>
<tr>
<td>__SL_P_N.H</td>
<td>STA DNS server</td>
<td>Sets STA DNS server address. Value format: xxx.yyy.zzz.ttt</td>
</tr>
<tr>
<td>__SL_P_N.I</td>
<td>DHCP start address</td>
<td>Sets start address. Value format: xxx.yyy.zzz.ttt</td>
</tr>
<tr>
<td>__SL_P_N.J</td>
<td>DHCP last address</td>
<td>Sets last address. Value format: xxx.yyy.zzz.ttt</td>
</tr>
<tr>
<td>__SL_P_N.K</td>
<td>DHCP lease time</td>
<td>Sets lease time, in seconds</td>
</tr>
<tr>
<td>__SL_P_N.P</td>
<td>AP IP address</td>
<td>Sets AP IP address. Value format: xxx.yyy.zzz.ttt</td>
</tr>
<tr>
<td>__SL_P_W.A</td>
<td>Channel # in AP mode</td>
<td>Sets channel number. Values: 1 to 13</td>
</tr>
<tr>
<td>__SL_P_W.B</td>
<td>SSID</td>
<td>Sets SSID</td>
</tr>
<tr>
<td>__SL_P_W.C</td>
<td>Security type</td>
<td>Sets security type: 0 for Open, 1 for WEP, 2 for WPA.</td>
</tr>
<tr>
<td>__SL_P_W.G</td>
<td>Password</td>
<td>Sets password</td>
</tr>
</tbody>
</table>

POST connection policy configuration is shown in Table 12-19.

Table 12-19. Connection Policy Configuration

<table>
<thead>
<tr>
<th>Token</th>
<th>Name</th>
<th>Values / Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection policy configuration</td>
<td>Used with the connection policy form (policy_config.html action)</td>
<td></td>
</tr>
<tr>
<td>__SL_P_P.E</td>
<td>Auto connect</td>
<td>Enable or Disable. If this parameter exists in the POST (with any value), this policy is set. If this parameter does not exist in the POST, this policy flag is cleared.</td>
</tr>
<tr>
<td>__SL_P_P.F</td>
<td>Fast connect</td>
<td>Enable or Disable. If this parameter exists in the POST (with any value), this policy is set. If this parameter does not exist in the POST, this policy flag is cleared.</td>
</tr>
</tbody>
</table>
Table 12-19. Connection Policy Configuration (continued)

<table>
<thead>
<tr>
<th>Token</th>
<th>Name</th>
<th>Values / Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>__SL_P_P.G</td>
<td>Any P2P</td>
<td>Enable or Disable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If this parameter exists in the POST (with any value), this policy is set. If</td>
</tr>
<tr>
<td></td>
<td></td>
<td>this parameter does not exist in the POST, this policy flag is cleared.</td>
</tr>
<tr>
<td>__SL_P_P.P</td>
<td>Auto SmartConfig</td>
<td>Enable or Disable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If this parameter exists in the POST (with any value), this policy is set. If</td>
</tr>
<tr>
<td></td>
<td></td>
<td>this parameter does not exist in the POST, this policy flag is cleared.</td>
</tr>
</tbody>
</table>

POST profiles configuration is shown in Table 12-20.

Table 12-20. Profiles Configuration

<table>
<thead>
<tr>
<th>Token</th>
<th>Name</th>
<th>Values / Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add new Profile</td>
<td></td>
<td>Used with the add new profile form</td>
</tr>
<tr>
<td>__SL_P_P.A</td>
<td>SSID</td>
<td>(profiles_add.html action)</td>
</tr>
<tr>
<td>__SL_P_P.B</td>
<td>Security type</td>
<td>Security type: 0 for Open, 1 for WEP, 2 for WPA1, 3 for WPA2</td>
</tr>
<tr>
<td>__SL_P_P.C</td>
<td>Security key</td>
<td>Smaller than 32 characters</td>
</tr>
<tr>
<td>__SL_P_P.D</td>
<td>Profile priority</td>
<td>0 to 7</td>
</tr>
<tr>
<td>Add P2P Profile</td>
<td></td>
<td>Used with the add P2P profile form</td>
</tr>
<tr>
<td>__SL_P_P.A</td>
<td>P2P Remote device name</td>
<td>String</td>
</tr>
<tr>
<td>__SL_P_P.B</td>
<td>P2P Security type</td>
<td>Security type: 6 for push-button, 7 for PIN keypad, 8 for PIN display</td>
</tr>
<tr>
<td>__SL_P_P.C</td>
<td>P2P PIN code</td>
<td>Digits only</td>
</tr>
<tr>
<td>__SL_P_P.D</td>
<td>P2P Profile priority</td>
<td>0 to 7</td>
</tr>
<tr>
<td>Add Enterprise Profile</td>
<td></td>
<td>Used with the add enterprise profile form</td>
</tr>
<tr>
<td>__SL_P_P.H</td>
<td>SSID</td>
<td>String</td>
</tr>
<tr>
<td>__SL_P_P.I</td>
<td>Identity</td>
<td>String</td>
</tr>
<tr>
<td>__SL_P_P.J</td>
<td>Anonymous identity</td>
<td>String</td>
</tr>
<tr>
<td>__SL_P_P.K</td>
<td>Password</td>
<td>String</td>
</tr>
<tr>
<td>__SL_P_P.L</td>
<td>Profile priority</td>
<td>0 to 7</td>
</tr>
<tr>
<td>__SL_P_P.M</td>
<td>EAP method</td>
<td>Values: TLS, TTLS, PEAP0, PEAP1, FAST</td>
</tr>
<tr>
<td>__SL_P_P.N</td>
<td>PHASE 2 Authentication</td>
<td>Values: TLS, MSCHAPV2, PSK</td>
</tr>
<tr>
<td>__SL_P_P.O</td>
<td>Provisioning (0,1, 2) (for fast method only)</td>
<td>Values: None, 0, 1, 2, 3 Relevant for fast method only (values 0 to 3)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For other methods, use None.</td>
</tr>
<tr>
<td>Profile remove</td>
<td></td>
<td>Remove selected profile Value: 1 to 7</td>
</tr>
</tbody>
</table>

POST tools are shown in Table 12-21.

Table 12-21. Tools

<table>
<thead>
<tr>
<th>Token</th>
<th>Name</th>
<th>Values / Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>__SL_P_T.A</td>
<td>IP address</td>
<td>IP address of the remote device</td>
</tr>
<tr>
<td>__SL_P_T.B</td>
<td>Packet size</td>
<td>In bytes (32 to 1472)</td>
</tr>
</tbody>
</table>
Table 12-21. Tools (continued)

<table>
<thead>
<tr>
<th>Token</th>
<th>Name</th>
<th>Values / Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>__SL_P_T.C</td>
<td>Number of pings</td>
<td>0 to unlimited, 1 to 255</td>
</tr>
</tbody>
</table>

POST P2P configuration is shown in Table 12-22.

Table 12-22. P2P Configuration

<table>
<thead>
<tr>
<th>Token</th>
<th>Name</th>
<th>Values / Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>__SL_P_R.E</td>
<td>P2P Channel (operational)</td>
<td>Set P2P operational channel. Values: 1, 6, 11</td>
</tr>
<tr>
<td>__SL_P_R.L</td>
<td>Negotiation intent value</td>
<td>Set Negotiation intent value. Values: CL for client, NEG for negotiate, GO for group owner</td>
</tr>
</tbody>
</table>

12.12.3 POST Actions

Table 12-23 describes the POST actions handled internally as complex actions and the respective token parameters used in each post. All other remaining post parameters are handled by the server by updating their respective value. For more information, Section 12.5.3.

Table 12-23. POST Actions

<table>
<thead>
<tr>
<th>Action Name</th>
<th>Description</th>
<th>POST Tokens Parameters</th>
</tr>
</thead>
</table>
| sta_ip_config       | Station network configuration      | • STA IP address  
|                     |                                    | • STA subnet mask  
|                     |                                    | • STA default gateway  
|                     |                                    | • STA DHCP state  
|                     |                                    | • STA DNS server  
| ap_ip_config        | Access point network configuration | • AP IP address  
|                     |                                    | • DHCP start address  
|                     |                                    | • DHCP last address  
|                     |                                    | • DHCP lease time  
| profiles_add.html   | Add new profile                    | • SSID  
|                     |                                    | • Security type  
|                     |                                    | • Security key  
|                     |                                    | • Profile priority  
| p2p_profiles_add    | Add peer to peer profile          | • P2P Remote device name  
|                     |                                    | • P2P Security type  
|                     |                                    | • P2P PIN code  
|                     |                                    | • P2P Profile priority  
| eap Profiles_add    | Add Enterprise profile             | • SSID  
|                     |                                    | • Identity  
|                     |                                    | • Anonymous identity  
|                     |                                    | • Password  
|                     |                                    | • Profile priority  
|                     |                                    | • EAP method  
|                     |                                    | • PHASE 2 Authentication  
|                     |                                    | • Provisioning  
| remove_all_profiles | Remove all profiles                | Not relevant  
|                     |                                    | Note: Keep at least one parameter in the HTML so the HTTP POST will not be empty. The server will not check the parameter value.  
| ping.html           | Start the ping test                | • IP address  
|                     |                                    | • Packet size  
|                     |                                    | • Number of pings  

submit documentation feedback
Table 12-23. POST Actions (continued)

<table>
<thead>
<tr>
<th>Action Name</th>
<th>Description</th>
<th>POST Tokens Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>ping_stop</td>
<td>Stop the ping test</td>
<td>Not relevant Note: Have at least one parameter in the HTML so the HTTP POST will not be empty. The server will not check the parameter value.</td>
</tr>
<tr>
<td>policy_config.html</td>
<td>Connection policy configuration</td>
<td>• Auto connect • Fast connect • Any P2P • Auto SmartConfig</td>
</tr>
</tbody>
</table>

12.12.4 **HTTP Server Limitations**

- HTTPS is currently not supported
- The HTTP web server can host a single domain only
- HTML form submission using GET method is not supported
- When configuring the Access point IP address, the address should be on the following subnet – 192.168.1.xxx.
### Chapter 13: mDNS

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>13.1 Overview</td>
<td>100</td>
</tr>
<tr>
<td>13.2 Protocol Detail</td>
<td>100</td>
</tr>
<tr>
<td>13.3 Implementation</td>
<td>103</td>
</tr>
<tr>
<td>13.4 Supported Features</td>
<td>106</td>
</tr>
<tr>
<td>13.5 Limitations</td>
<td>106</td>
</tr>
</tbody>
</table>
13.1 Overview

Domain Name System (DNS), specified by RFC1034, is a hierarchical distributed naming system for computers, services, or any resource connected to the network. In a unicast DNS system, all domain name information is stored in the DNS server. Clients get domain name of others by simply fetching such information from server.

multicast DNS (mDNS), specified by RFC6762, resolve host names to Internet Protocol (IP) addresses within small networks that do not include a local name server. It is a zero configuration service, using essentially the same programming interfaces, packet formats and operating semantics as the unicast DNS. While it is designed to be stand-alone capable, it can work in concert with unicast DNS servers. This protocol uses IP multicast User Datagram Protocol (UDP) packets instead of the Transmission Control Protocol (TCP) that unicast DNS uses.

mDNS together with DNS Service Discovery (DNS-SD), specified by RFC6763, forms the Apple’s implementation of zero-configuration network known as Bonjour. TI’s Simplelink WiFi currently supports such implementation in advertise mode. Query mode only supports PTR and the full Bonjour support will be implemented in the future.

13.2 Protocol Detail

By default, mDNS exclusively resolves host names ending with the .local top-level domain (TLD). The mDNS Ethernet frame is a multicast UDP packet to:

- MAC address 01:00:5E:00:00:FB (for IPv4) or 33:33:00:00:00:FB (for IPv6)
- IPv4 address 224.0.0.251 or IPv6 address FF02::FB
- UDP port 5353

The Ethernet frame content like Questions, Answers, Authority, Additional, and Data fields, including their formats, are identical to those inside a unicast DNS packet. Detailed specification can be found in RFC1034.

Services are found using resource record (RR) queries identical to unicast DNS. RRs store a large variety of records about a domain such as IP addresses (A & AAAA types), pointer record (PTR type), Service locator (SRV type), text (TXT type) and so forth. A full list can be found in the Wikipedia link: List of DNS record types.

A DNS zone database is a collection of resource records. Each resource record specifies information about a particular object. For example, address mapping records a host name to an IP address.

RR queries with types of PTR, SRV, TXT, and A (or AAAA in case of IPv6) are needed for discovering the full-service details.

- PTR RR returns the URN/full-service name
- SRV RR is used for the discovery services provided by the hosts and returns the service types, including the domain name.
- TXT RR gets the arbitrary text associated with a domain, or depicts the service. A record maps the host names to an IPV4 address.

All answers must be sent in a single response to a query. For example:

An mDNS client must resolve a host name after receiving a PTR record, which includes a service in the interest of the application:

1. The client sends an IP multicast query message asking the host with that name to identify it.
2. That target machine then multicasts a message that includes its IP address.
3. All machines in that subnet can then use that information to update their mDNS caches.
Figure 13-1. mDNS Get Service Sequence
Figure 13-2. Find Full Service After Query
13.3 Implementation

13.3.1 Default Implementation

The mDNS service in the Simplelink implementation is started by default with the following names:

- **Target name**: `<MAC address>-mysimplelink.local`
  
  The target name is the name that should be in 'A' query (query for getting the IPv4). The name is assembled from the MAC address of the target, the word “mysimplelink”, and the domain of the local network. The user can change this name by setting the URN name or by a smart-configure operation.

- **Internal services**: `<MAC address>@mysimplelink._http._local`

  
  The complete list of functions, structures, and constants can be found in the Netapp library. The following sections show examples of top-specific implementation. For details, see the API.

13.3.2 Start and Stop mDNS

mDNS is an NWP service that can be started or stopped with:

- `sl_NetAppStart(SL_NET_APP_MDNS_ID)`
- `sl_NetAppStop(SL_NET_APP_MDNS_ID)`

The macro `SL_NET_APP_MDNS_ID` is the definition of mDNS service.

mDNS frames (advertise, response to queries) are sent only if there is a valid IP address, which means mDNS works if one of the following wlan status is met:

- Simplelink is connected to AP as a station
- Simplelink P2P mode is running, either as a group owner (GO) or a client
- Simplelink AP mode is running

Be aware that `sl_NetAppStart()` and `sl_NetAppStop()` functions are:

- **Persistent** - The state is stored in the serial flash and will retain its previous state even after a power cycle.
- **Role depended** - The state cannot be shared across different roles. For example, if Simplelink is currently in AP mode with NetApp started, switching to STA mode will require the NetApp to start again, if it was previously disabled during STA mode. Other Netapp configurations such as registered services (for example, `sl_NetAppMDNSRegisterService()`) are not affected by role changes.

13.3.3 mDNS Query – One Shot

In a one-shot query, discovery result depends on the very first respond received by the discoverer. The function `sl_NetAppDnsGetHostByService()` is used to perform such task. Upon successful discovery, the function will return:

- IP address of the Service
- Port of the Service
- Text of the Service

The following code snippet shows a one-shot query discovering the "_workstation._tcp.local" service:

```c
#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)
```
13.3.4 mDNS Query – Continuous

In a continuous mDNS query, having received one response is not necessarily an indication that there will be no more relevant responses, and the querying operation continues until no further responses are required. Since one-shot query is the default setting, extra settings needs to be done before continuous query can be used. However, since \texttt{sl\_NetAppDnsGetHostByService()} only returns one discovered host at a time, use \texttt{sl\_NetAppGetServiceList()} to get a complete list of services. Here’s an example with return list in the format of \texttt{SlNetAppGetFullServiceWithTextIpv4List\_t}: 

```c
#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 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_NetAppGet() return: %d
    iretvalmDNS);
if(iretvalmDNS > 0)
{
    // Now just prints all the services
    Report("Complete List:\n");
    for(i=0; i<iretvalmDNS; i++)
    {
        Report("Host %d: %s, IP: %d.%d.%d.%d, Name: %s Port: %d, Text: %s
");
        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);
    }
    Report("\n");
}

### 13.3.5 mDNS Service Registration

Besides the default HTTP, users can register up to five mDNS to broadcast. Registration is persistent regardless of NWP being active or not because registration information is permanently stored inside the serial flash. Use `sl_NetAppMDNSRegisterService()` and `sl_NetAppMDNSUnRegisterService()` to register and unregister an mDNS service, respectively.

The following example shows the register process:

```
#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 that 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));
    sl_NetAppMDNSRegisterService(SERVICE_NAME,
        (unsigned char) strlen(SERVICE_NAME),
        "Apple",
        200,
        2000,
        0x00000001);
    Report("mDNS 1 Registered with NAME: %s & ERROR code: %d
", SERVICE_NAME,
        Service_Name," \n");
```
Supported Features

To unregister all mDNS services at once, use the following call:

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

Register, just once, a specific service. The service is saved and advertised after each reset (if mDNS is started and STA with IP or P2P/AP up).

- Responses to queries that contain the service name or type are also sent.
- Set, just once, the advertising timing parameters if the default parameters are not required.

### 13.4 Supported Features

- Supported Features
  - Fully supports Bonjour
  - Register and advertise services
  - External services advertise up to five
  - Internal services advertise:
    - Services that are advertised by default without a relation to mDNS database
    - The internal capabilities of the product
    - HTTP server – Peers can find the IP and browse without a DNS server
  - Responds
    - Response to queries
    - Ignores queries that are already familiar with services
  - Searches for services in the local network by using:
    - One-shot query
    - Continuous query
  - Gathers the known information on peer services using the GET service list API:
    - Full services with text – name, host, IP, port, text
    - Full services without text – name, host, IP, port
    - Short services – IP, port
  - Masks peer services received in responses or peer advertisements.
    - Sets advertising timing. Reconfigures the timing parameters employed by mDNS when sending the service announcements: how many times and when a service is advertised (not related to responding queries).

### 13.5 Limitations

The following are the known limitations of the mDNS feature:

- Support or receive up to eight different services in the peer cache (use filter for storing only the required services).
- The mDNS machine stops and starts when all services are deleted; the peer cache is deleted.
- If the user registers a unique service but a service with the same name already exists in the network,
then the service name is changed to "name (number)," for example PC1 (2) _ipp._tcp.local. In this case, the name in the DB is the original name but the advertising uses the new name.

- Limited Bonjour support. Currently supports Bonjour in advertise mode. Query mode only supports PTR and the full Bonjour support will be implemented in the future.
- Deleting a unique name that was changed because of mismatch between the names (the advertising name and the DB name) causes the mDNS machine to stop and start, and deletes the peer cache.
- If there is a one-shot query but the peer cache is full, there is no place to set the query. The peer cache is deleted, and then the query is sent.
- There is a partial check if the service name that is registered is legal. The user must send a legal name.
- When using GET host by service, only one answer is returned. To see all the answers, wait for all peer answers to be sent and received, then read the answers by using the API GET service list.
- The max buffer list size of the API GET service list is 1480 bytes. A request for a larger list returns the SL_ERROR_NETAPP_RX_BUFFER_LENGTH_ERROR code.
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>14.1 Overview</td>
<td>109</td>
</tr>
</tbody>
</table>
14.1 Overview

The CC31xx/CC32xx system is accompanied by a serial flash, which serves multiple functions. It holds all the relevant system files for the networking subsystem, patches and configuration files. In the case of CC32xx, it also holds the user application code. Additionally, the product enables the use of the remaining flash area as a free storage resource left for the user data files.

The aim of this paragraph is to present the common API and configuration of the file system and also list the file system limitations.

Table 14-1. Summary of Instructions

<table>
<thead>
<tr>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Creation</td>
<td>110</td>
</tr>
<tr>
<td>File Opening</td>
<td>111</td>
</tr>
<tr>
<td>File Closing</td>
<td>112</td>
</tr>
<tr>
<td>File Writing</td>
<td>113</td>
</tr>
<tr>
<td>File Reading</td>
<td>114</td>
</tr>
<tr>
<td>File Deleting</td>
<td>115</td>
</tr>
<tr>
<td>File Information</td>
<td>116</td>
</tr>
</tbody>
</table>
### File Creation

**Description**
This operation enables creation of a non-existing file. Upon creation request, the device checks for available space on the serial flash. Failing to allocate space for the new file generates an error. Additionally, creating an already existing file generates an error.

**API**

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

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Parameter</th>
<th>In/Out</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_u8*</td>
<td>pFileName</td>
<td>In</td>
<td>Pointer to the target file name. NULL terminated</td>
</tr>
<tr>
<td>_u32</td>
<td>AccessModeAndMaxSize</td>
<td>In</td>
<td>Size and flags as described below</td>
</tr>
<tr>
<td>_u32*</td>
<td>pToken</td>
<td>In</td>
<td>Reserved for future use in secured file system. Should be NULL</td>
</tr>
<tr>
<td>_i32*</td>
<td>pFileHandle</td>
<td>Out</td>
<td>Handle to the created file, in case success is returned</td>
</tr>
</tbody>
</table>

AccessModeAndMaxSize can be configured using the following MACRO:

```c
FS_MODE_OPEN_CREATE(maxSizeInBytes,accessModeFlags)
```

- **maxSizeInBytes**: each file includes 512 bytes of metadata. For optimal size allocation on the serial flash, use max size as in the following formula:

  (((x*4096)- 512) % Granularity) * Granularity, where x is an integer and the supported granularities are {256, 1024, 4096, 16384, 65536}

- **accessModeFlags**: only **_FS_FILE_OPEN_FLAG_COMMIT** flag is applicable (fail safe flag). All other flags are reserved for future use in secured file system

**Return Value**

On success, zero is returned. On error, an error code is returned.

**Example**

```c
_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 );
```

**Limitations**

- File size is allocated upon creation and cannot be extended later on
- Creating a file with COMMIT flag results in file mirroring. It is used for fail-safe purposes only. The user cannot choose from which instance to load.
- Maximal file size is 1MB
- Minimal allocated size on flash is 4KB, i.e. the size of a block
File Opening

Description

This operation enables opening of an existing file for either read or write. Upon opening request, the device checks that the file exists on the serial flash and if it doesn’t, an error is generated. Note that no size is configured as the size is allocated on file creation only.

API

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

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Parameter</th>
<th>In/Out</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_u8*</td>
<td>pFileName</td>
<td>In</td>
<td>Pointer to the target file name. NULL terminated</td>
</tr>
<tr>
<td>_u32</td>
<td>AccessModeAndMaxSize</td>
<td>In</td>
<td>Size and flags as described below</td>
</tr>
<tr>
<td>_u32*</td>
<td>pToken</td>
<td>In</td>
<td>Reserved for future use in secured file system. Should be NULL</td>
</tr>
<tr>
<td>_i32*</td>
<td>pFileHandle</td>
<td>Out</td>
<td>Handle to the created file, in case success is returned</td>
</tr>
</tbody>
</table>

AccessModeAndMaxSize can be configured using the following MACRO:

- FS_MODE_OPEN_READ: for file reading
- FS_MODE_OPEN_WRITE: for file writing

Return Value

On success, zero is returned. On error, an error code is returned.

Example

```c
_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 );
```

Limitations

- Upon opening a file for write access, entire file is erased
File Closing

Description
This operation enables closing an existing file.

API

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

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Parameter</th>
<th>In/Out</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_i32</td>
<td>pFileName</td>
<td>In</td>
<td>Handle to the created file</td>
</tr>
<tr>
<td>_u8*</td>
<td>pCertificateFileName</td>
<td>In</td>
<td>Reserved for future use in secured file system. Should be NULL</td>
</tr>
<tr>
<td>_u8*</td>
<td>pSignature</td>
<td>In</td>
<td>Reserved for future use in secured file system. Should be NULL</td>
</tr>
<tr>
<td>_u32</td>
<td>SignatureLen</td>
<td>In</td>
<td>Reserved for future use in secured file system. Should be NULL</td>
</tr>
</tbody>
</table>

Return Value
On success, zero is returned. On error, an error code is returned.

Example

```c
_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);
```

Limitations

- None
File Writing

Description

This operation enables writing to an existing file. The purpose of the offset is mainly to allow random access write without using the fseek method as in "standard" file system. Using this method you can access any location in the file with a single API call.

For write purposes, user can write the file in chunks without taking care of the order of the chunks as long as the same location if not repeated twice while the file is opened.

API

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

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Parameter</th>
<th>In/Out</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_u32</td>
<td>FileHandle</td>
<td>In</td>
<td>Handle to an existing file (returned on sl_FsOpen())</td>
</tr>
<tr>
<td>_u32</td>
<td>Offset</td>
<td>In</td>
<td>Offset within a file</td>
</tr>
<tr>
<td>_u32*</td>
<td>pData</td>
<td>In</td>
<td>Pointer to the written data</td>
</tr>
<tr>
<td>_i32*</td>
<td>Len</td>
<td>In</td>
<td>Length of the written data</td>
</tr>
</tbody>
</table>

Return Value

On success, zero is returned. On error, an error code is returned.

Example

```c
_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 );
```

Limitations

File appending is not embedded into the file system. Appending can be achieved by read-modify-write implementation in the host.
File Reading

Description
This operation enables reading from an existing file. The purpose of the offset is mainly to allow random access read without using the fseek method as in "standard" file system. Using this method you can access any location in the file with a single API call.

API

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

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Parameter</th>
<th>In/Out</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_u32</td>
<td>FileHandle</td>
<td>In</td>
<td>Handle to an existing file (returned on sl_FsOpen())</td>
</tr>
<tr>
<td>_u32</td>
<td>Offset</td>
<td>In</td>
<td>Offset within a file</td>
</tr>
<tr>
<td>_u32*</td>
<td>pData</td>
<td>In</td>
<td>Pointer to the written data</td>
</tr>
<tr>
<td>_i32*</td>
<td>Len</td>
<td>in</td>
<td>Length of the written data</td>
</tr>
</tbody>
</table>

Return Value
On success, zero is returned. On error, an error code is returned.

Example

```c
_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);
```

Limitations
File appending is not embedded into the file system. Appending can be achieved by read-modify-write implementation in the host.
File Deleting

Description
This operation enables deleting an existing file from the serial flash. If the file does not exist on the serial flash, an error is generated. Upon deleting a file, its occupied space on the serial flash is freed and made available for re-allocation by the system.

API

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

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Parameter</th>
<th>In/Out</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_u8</td>
<td>pFileName</td>
<td>In</td>
<td>Pointer to the target file name. NULL terminated</td>
</tr>
<tr>
<td>_u32</td>
<td>Token</td>
<td>In</td>
<td>Reserved for future use in secured file system. Should be 0</td>
</tr>
</tbody>
</table>

Return Value
On success, zero is returned. On error, an error code is returned.

Example

```c
_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);
```

Limitations
None
File Information

Description
This operation enables getting file information of an existing file. If the file does not exist on the serial flash, an error is generated.

API

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

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Parameter</th>
<th>In/Out</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_u8*</td>
<td>pFileName</td>
<td>In</td>
<td>Pointer to the target file name. NULL terminated</td>
</tr>
<tr>
<td>_u32</td>
<td>Token</td>
<td>In</td>
<td>Reserved for future use in secured file system. Should be 0</td>
</tr>
<tr>
<td>SlFsFileInfo_t*</td>
<td>pFsFileInfo</td>
<td>Out</td>
<td>File Information: flags, file size, allocated size and Tokens. Look on below structure.</td>
</tr>
</tbody>
</table>

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

- Flags are for future use
- FileLen is the actual length of the file
- AllocatedLen is the requested allocated length during file creation
- Token is for future use

Return Value
On success, zero is returned. On error, an error code is returned.

Example

```c
_u8 DeviceFileName[] = "MyFile.txt";
_u32 MaxSize = 63 * 1024;
_i32 DeviceFileHandle = -1;
_i32RetVal;
_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);
```

Limitations
None

File system limitations

- File size is allocated upon creation and cannot be extended later on
- File appending is not embedded into the file system. Appending can be achieved by read-modify-write implementation in the host.
• Upon opening a file for write access, entire file is erased
• Creating a file with COMMIT flag results in file mirroring. It is used for fail-safe purposes only. The user cannot choose from which instance to load.
• Maximal file size is 1MB.
• Minimal allocated size on flash is 4KB, for example, the size of a block
• Fragmentation is not applicable.
• File system is not directory structured. Nevertheless, it is highly recommended to keep a directory prefix structure for better readability and maintenance. For example, all user files should begin with /usr/ prefix.

**NOTE:** The ability of the web server to access the file system is limited to www/ prefix as a root folder.
# Chapter 15

**Rx Filter**

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>15.1 Overview</td>
<td>119</td>
</tr>
<tr>
<td>15.2 Detailed Description</td>
<td>119</td>
</tr>
<tr>
<td>15.3 Examples</td>
<td>119</td>
</tr>
<tr>
<td>15.4 Creating Trees</td>
<td>121</td>
</tr>
<tr>
<td>15.5 Host API</td>
<td>121</td>
</tr>
<tr>
<td>15.6 Notes and Limitations</td>
<td>123</td>
</tr>
</tbody>
</table>
15.1 Overview

The Rx-filters module lets the user define a set of rules that will determine which of the received frames will be transferred by the SimpleLink WiFi device to the host, and which frames will be dropped. The Rx-filters can be activated during AP-connection and during promiscuous mode (“disconnect mode”).

15.2 Detailed Description

Every received frame traverses through a series of decision trees that determine how the frame is treated. The decision trees are composed of filter nodes. Each node has its filter rule, action, and trigger. The tree traversal process starts with the root nodes of the trees: if the filter rule and trigger of the root node are TRUE, the action of the root node is performed and the frame continues to the child nodes.

Filter rules are specific protocol header values:
- MAC layer: Frame type, frame subtype, BSSID, source MAC address, destination MAC address, and frame length
- LLC layer: Protocol type
- Upper layers: IP version, IP protocol, source IP address, destination IP address, ARP operation, ARP target IP address, source port number, and destination port number

Possible triggers:
- When role is… (station/AP/promiscuous)
- When connection state is… (connected/disconnected)
- When counter reaches X

Possible actions:
- Drop the packet (do not pass it to the host)
- Increase or decrease the counter value

The trees traversal process stops when the frame reaches a DROP action in one of the trees. Traversing is done layer by layer among all the trees.

The user can define a combined-filter node. This node has two parent nodes (unlike a regular node which has only one parent node), and is checked only if one or both (user-defined) of its two parent nodes is TRUE. For example: if (node_1 OR node_2).

15.3 Examples

Example 1: Supposing a user has the following requirements:
- Receive WLAN data broadcast frames only from two specific MAC addresses.
- Receive all WLAN unicast frames, except for frames with a certain SRC_IP address range.
- If a unicast frame is received from MAC address AA.AA.AA, increase counter_1.
Examples

- If a unicast frame is received from MAC address BB.BB.BB, increase counter_2.
- If a unicast UDP frame is received from MAC address AA.AA.AA or BB.BB.BB, pass only packets from port 5001.

The following trees should be created: (see Figure 15-1).

![Diagram](image)

**Figure 15-1. Trees Example 1**

**Example 2:** Supposing a user has the following requirement:
- Receive only WLAN management beacon frames from all MAC addresses.

The following trees should be created: (see Figure 15-2).
15.4 Creating Trees

- Trees are created by the user. The user adds the filter nodes individually and defines the filter tree hierarchy.
- Trees can also be created and applied internally by the system.
- Filters can be created as persistent filters that are saved in the flash memory and loaded at the system startup.
- The maximum number of filter nodes is 64. The system uses 14 filter nodes; the remaining 50 nodes are for the user.
- Filters can be created, removed, enabled, and disabled. After filters are created, they must be enabled to start filtering.

15.5 Host API

- WlanRxFilterAdd – Adds a new filter to the system
- sl_WlanRxFilterSet – Sets parameters to Rx filters possible operations:
  - SL_ENABLE_DISABLE_RX_FILTER - Enables/disables filter in a filter list
  - SL_ENABLE_DISABLE_RX_FILTER - Enables/disables filter in a filter list
  - SL_STORE_RX_FILTERS - Save the filters for persistent
  - SL_UPDATE_RX_FILTER_ARGS - Update the arguments of existing filter
  - SL_FILTER_PRE_PREPARED_SET_CREATE_REMOVE_STATE - Change the default creation of the pre-prepared filters
- sl_WlanRxFilterGet – Gets parameters of Rx filters possible operations:
  - SL_FILTER_RETRIEVE_ENABLE_STATE - Retrieves the enable disable status
  - SL_FILTER_PRE_PREPARED_RETRIEVE_CREATE_REMOVE_STATE - Retrieves the pre-prepared filters creation status

15.5.1 Code Example

```c
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));

/*
 * First Filter is:
 * if FrameType != Management --> Action == Drop
 */

/* Build filter parameters */
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);

/*
 * 2nd Filter is:
 * 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);

*/
  * 3rd Filter is:
  * 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],&FrameSubT
  ype, 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);

15.6 Notes and Limitations

While using the Rx filter database update commands (add/remove/update/enable/disable), all sockets in
the host must be closed.

The Rx filter database update commands are not intended to be used while the SL devices are exposed
to incoming receptions on their sockets (standard or raw).
Chapter 16
SWRU368B—April 2014—Revised May 2018

Transceiver Mode

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>16.1 General Description</td>
<td>125</td>
</tr>
<tr>
<td>16.2 How to Use / API</td>
<td>125</td>
</tr>
<tr>
<td>16.3 Sending and Receiving</td>
<td>126</td>
</tr>
<tr>
<td>16.4 Changing Socket Properties</td>
<td>126</td>
</tr>
<tr>
<td>16.5 Internal Packet Generator</td>
<td>127</td>
</tr>
<tr>
<td>16.6 Transmitting CW (Carrier-Wave)</td>
<td>127</td>
</tr>
<tr>
<td>16.7 Connection Policies and Transceiver Mode</td>
<td>127</td>
</tr>
<tr>
<td>16.8 Notes about Receiving and Transmitting</td>
<td>127</td>
</tr>
<tr>
<td>16.9 Use Cases</td>
<td>128</td>
</tr>
<tr>
<td>16.10 TX Continues</td>
<td>130</td>
</tr>
<tr>
<td>16.11 Transceiver Mode Limitations</td>
<td>130</td>
</tr>
</tbody>
</table>
16.1 General Description

The 802.11 transceiver mode is a powerful tool for sending and receiving raw data in either Layer 2 (MAC) or Layer 1 (physical).

Table 16-1 lists the eight network layers:

<table>
<thead>
<tr>
<th>Layer</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td></td>
</tr>
<tr>
<td>Presentation</td>
<td></td>
</tr>
<tr>
<td>Session</td>
<td></td>
</tr>
<tr>
<td>Transport</td>
<td></td>
</tr>
<tr>
<td>Network</td>
<td></td>
</tr>
<tr>
<td>LLC – Logical Link Control</td>
<td></td>
</tr>
<tr>
<td>MAC – Medium Access Control</td>
<td></td>
</tr>
<tr>
<td>Physical</td>
<td></td>
</tr>
</tbody>
</table>

Use the space of the entire frame starting from the 802.11 header to receive and transmit data.

In transceiver mode, there are no frame acknowledgments or retries. Therefore, there are no guarantees that the frame will reach its destination. When working in L1 mode, there may be collisions with other frames or energy.

Figure 16-1 shows the 802.11 frame structure. The white sections are user-configurable, and the grayed part cannot be overwritten.

16.2 How to Use / API

Using the host driver that controls the SimpleLink WiFi device (which includes the SimpleLink studio), the user needs only five commands to work with the abilities of the 802.11 transceiver. Because the SimpleLink WiFi device is a networking device that complies with BSD socket implementation, use the command `sl_Socket` with the following arguments to start the transceiver.

```c
soc = sl_Socket(SL_AF_RF ,SL_SOCK_RAW/SL_SOCK_DGRAM,channel);
```
16.3 Sending and Receiving

The user can open and close the transceiver using two commands. To start the traffic flow, use the `sl_Send` command to transmit and the `sl_Recv` command to receive.

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

- `soc` – Socket ID
- `RawData` – The char *array holds the data to send, beginning with the first byte of the 802.11 MAC header.
- `len` – The size of the data in bytes
- `flags` – Usually the user sets this parameter to 0, but the user can change any of the default channel/rate/tx-power/11b-preamble parameters using this parameter. Use the `SL_RAW_RF_TX_PARAMS` macro to specify the mentioned parameters.

**Returns** – The number of bytes sent

For example, to transmit a frame on channel 1 using the 1MBPS data rate with Tx power setting of 1 and short preamble (valid only for 11b), use:

```
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` – Socket ID
- `buffer` – The char *array used for containing the received packet
- `500` – The maximum size of the packet received. The maximum size is 1472: if the packet is longer, the rest are discarded.

The last argument should always be 0, which indicates no flags.

**Return** – The number of bytes received

16.4 Changing Socket Properties

The command `sl_SetSockOpt` changes the socket properties (after opening the socket).

Examples:

**Change the operating channel:**
```
sl_SetSockOpt(soc, SL_SOL_SOCKET, SO_CHANGE_CHANNEL, &channel,1);
```

**Change the default PHY data rate:**
```
sl_SetSockOpt(soc, SL_SOL_PHY_OPT, SO_PHY_RATE, &rate,1);
```

**Change the default Tx power:**
```
sl_SetSockOpt(soc, SL_SOL_PHY_OPT, SO_PHY_TX_POWER, &power,1);
```

**Change the number of frames to transmit** (see Section 16.5):
```
sl_SetSockOpt(soc, SL_SOL_PHY_OPT, SO_PHY_NUM_FRAMES_TO_TX, &numFrames,1);
```
Change the 802.11b preamble:

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

16.5 **Internal Packet Generator**

For testing purposes, there is an internal packet generator in the SimpleLink WiFi device capable of repeating a predefined pattern of data.

To use, before calling `sl_Send`, set the number of frames using the `sl_SetSockOpt` to either 0 (an infinite number of frames) or to the given number of frames needed to transmit.

Following that, use a single call to the `sl_Send` API to trigger the frames transmission.

The SimpleLink WiFi device keeps transmitting until it has sent all the requested frames, or until the socket is closed or another socket property changes (in case an infinite number of frames were used).

16.6 **Transmitting CW (Carrier-Wave)**

To transmit a carrier-wave signal, use the `sl_Send` API with NULL buffer and 0 (zero) length.

Use the flags parameter in the `sl_Send` API to signal the tone offset (-25 to 25).

Stopping CW transmission is done by triggering another `sl_Send` API with flags = 128 (decimal) as follows: `sl_Send (soc,NULL,0,128);`

16.7 **Connection Policies and Transceiver Mode**

To use transceiver mode, disable previous connection policies that might try to automatically connect to an AP.

Example:

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

16.8 **Notes about Receiving and Transmitting**

16.8.1 **Receiving**

A SimpleLink proprietary radio header is attached to the packet being received. The header has some information about the packet. This is the structure of the header.

```c
typedef struct }
}TransceiverRxOverHead_t;
```

The rate is an index from 0 to 20 in the following order:

- 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

The channel is 1 to 11.

If using the `sl_Recv` command results in a frame in a buffer, extract the header by casting the start of the buffer to a pointer variable in type of `TransceiverRxOverHead_t`

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

### 16.9 Use Cases

The following key applications can be developed using this feature.

#### 16.9.1 Sniffer

The transceiver can be used as a sniffer. Open a socket and receive packets in a loop using the `sl_Recv` command. The following code describes how to capture frames and present them in wireshark:

```c
void Sniffer(Channel_e channel)
{
    UINT16 soc;
    char buffer[1536];
    int counter = 0;
    int receivedBytes = 0;
    long count = 0;
    long bytesSent = 0;
    DWORD startTick = 0;
    int fConnected = 0;
    HANDLE hPipe = NULL;
    LPTSTR lpszPipeName = TEXT("\\.\\pipe\cc3100");
    DWORD written;
    DWORD result;
    wireSharkGlobalHeader_t gHeader;
}```

**Figure 16-2. Sniffer**
```c
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(pszPipeName,
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
\\\\\\\\\\"\n);  /*Connected = 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 = 0x000000FF;
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
recvedBytes = 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 802.11 radio header for pcap format.
// here you can put the capture time in windows format
frame.ts_sec = 160285;
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 + recvedBytes - 8;
frame.orig_len = 24 + recvedBytes - 8;
frame.it_version = 0;
frame.it_pad = 0;
frame.it_len = 24;
frame.it_present = 0x0000002d;

// present the timestamp
frame.ts_low = frameRadioHeader->timestamp;
frame.ts_high = 0;

// present the rate
frame.rate = RateIndexToRate[fameRadioHeader->rate];
frame.padi = 0x11;

// present the channel
frame.channel_low = ChannelToFrequency[(Channel_t)frameRadioHeader->channel];
frame.channel_high = 0x0000;

// present the rssi
frame.antena = frameRadioHeader->rssi;
frame.padi = 0x11;

// send the wireshark frame header
result = WriteFile(hPipe,&frame,sizeof(frame),&BYTEwritten,NULL);
// send the frame minus the 8 bytes radio header
result = WriteFile(hPipe,&buffer[8]),(recvedBytes - 8),&BYTEwritten,NULL);
}
sl_close(soc);
```

Figure 16-3. Sniffer
16.10 TX Continues

This application transmits the same packet in continues. This application tags and measures loss using the Rx Statistics feature. The following code shows how to use this feature:

```c
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);
}
```

Figure 16-4. Tx Continues

16.11 Transceiver Mode Limitations

- The user can open one transceiver socket in the system.
- The length of a received packet will be trimmed if it exceeds 1472 bytes.
- Cannot transmit frame over 1472 bytes and below 14 bytes.
- If using the SimpleLink studio, note that the maximum throughput of the SPI is 5~6Mbps, so in a crowded WIFI medium, there will be packet loss.
- The transceiver will not open in connection or connected mode. Notice that auto connection mode is also considered as connected mode, even if not connected.
## RX STATISTICS

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>17.1 General Description</td>
<td>132</td>
</tr>
<tr>
<td>17.2 How to Use / API</td>
<td>132</td>
</tr>
<tr>
<td>17.3 Notes about Receiving and Transmitting</td>
<td>133</td>
</tr>
<tr>
<td>17.4 Use Cases</td>
<td>133</td>
</tr>
<tr>
<td>17.5 Rx Statistics Limitations</td>
<td>134</td>
</tr>
</tbody>
</table>
17.1 General Description

The Rx Statistics feature is used to determine certain important parameters about the medium and the SimpleLink WiFi device RX mechanism.

Rx Statistics provides data about:

- RSSI – Receive power in dbm units.
  - RSSI histogram, from -40 to -87 dbm in resolution of 8 dbm
  - Average RSSI divided to DATA + CTRL / MANAGEMENT
- Received frames – Divided into valid frames, FCS error and PLCP error frames
- Rate histogram – Creates a histogram of all BGN rates 1mbps-MCS7

17.2 How to Use / API

Three commands get the needed statistics. The first command is `sl_WlanRxStatStart()`, which starts collecting the data about all Rx frames.

`sl_WlanRxStatStop()` stops collecting the data.

`sl_WlanRxStatGet()`, gets the statistics that have been collected and returns them in a variable type `SlGetRxStatResponse_t`. The command is used as follows:

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

The second parameter is flagged and it is not currently in use.

`SlGetRxStatResponse_t` holds the following variables:

- `ReceivedValidPacketsNumber` – Holds the number of valid packets received
- `ReceivedFcsErrorPacketsNumber` – Holds the number of FCS error packets dropped
- `ReceivedAddressMismatchPacketsNumber` - Holds the number of packets that has been received but filtered out by one of the HW filters

In connection mode:

- The data is valid only after `sl_WlanRxStatStart` is called
- The counter will indicate the number of frames that were filtered out by one of the HW filters – since this is connection mode most of the packets will be based on address missmatch

In Transceiver mode (disconnect mode):

- The data is valid only after `sl_WlanRxStatStart` is called
- The counter will indicate the number of frames that were filtered out by one of the HW filters
- Configure a filter in the Rx Transceiver (MAC address, frame type and so forth) in order to see this counter incremented

- `avarageDataCtrlRssi` – Holds the average data + control frames RSSI
- `avarageMgMntRssi` – Holds the average management frames RSSI

`RateHistogram[NUM_OF_RATE_INDEXES]` – Histogram of all rates of the received valid frames. The rates are sorted as follows:

- `RATE_1M`
- `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 is 21.

RssiHistogram[SIZE_OF_RSSI_HISTOGRAM] – Histogram that holds the accumulated RSSI of all received packets from -40 to -87 dbm every 8 dbm.

SIZE_OF_RSSI_HISTOGRAM is 6

StartTimeStamp – Holds the start collecting time in microsec

GetTimeStamp – Holds the statistics get time in microsec

17.3 Notes about Receiving and Transmitting

Every packet that exceeds the upper or lower boundary of the RSSI histogram in dbm (-40 or -87) is accumulated in the higher or lower cell, respectively.

Every call to `sl_WlanRxStatGet` resets the statistics database.

When calling only `sl_WlanRxStatGet`, without the start command, only the RSSI is available.

17.4 Use Cases

The Rx Statistics feature inspects the medium in terms of congestion and distance, validates the RF hardware, and uses the RSSI information to position the SimpleLink WiFi device in an ideal location.

Example:

Connect the SimpleLink device to an AP, run a UDP flow of packets to the device from the AP, and use the following code with the SimpleLink Studio to get Rx statistics.

```c
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)
```c
{    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\n");    for(idx = 0 ; idx < NUM_OF_RATE_INDEXES; idx++)    {        printf("Rate Histogram cell %d is %d\n", idx, rxStatResp_RATE_Histogram[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;
}

17.5 Rx Statistics Limitations

The maximum histogram cell capacity is 65535 for both RSSI and rate. If more packets will be received, the accumulation will stop. All the other statistics are held in unsigned integer and will wraparound when exceed the maximum.
This chapter discusses the different SimpleLink APIs. The chapter does not cover the number of parameters, parameter types, or return values. For that information, refer to the API doxygen guide.

The APIs are separated into six main groups:

- Device
- NetConfig
- WLAN
- Socket
- NetApp
- File System

![Figure 18-1. Host Driver API Silos](image)

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>18.1 Device</td>
<td>136</td>
</tr>
<tr>
<td>18.2 NetCfg</td>
<td>138</td>
</tr>
<tr>
<td>18.3 WLAN</td>
<td>139</td>
</tr>
<tr>
<td>18.4 Socket</td>
<td>142</td>
</tr>
<tr>
<td>18.5 NetApp</td>
<td>144</td>
</tr>
<tr>
<td>18.6 File System</td>
<td>146</td>
</tr>
</tbody>
</table>
18.1 Device

The device APIs handle the device power and general configuration.

*Sl_Start* – This function initializes the communication interface, sets the enable pin of the device, and calls to the init complete callback. If no callback function is provided, the function is blocking until the device finishes the initialization process. The device returns to its functional role in case of success: ROLE_STA, ROLE_AP, ROLE_P2P. Otherwise, in case of a failure it returns: ROLE_STA_ERR, ROLE_AP_ERR, ROLE_P2P_ERR

*Sl_Stop* – This function clears the enable pin of the device, closes the communication interface, and invokes the stop complete callback (if it exists). The time-out parameter lets the user control the hibernate timing:

- 0 – Enter to hibernate immediately
- 0xFFFF – The host waits for the device to respond before hibernating, without timeout protection.
- 0 < Timeout[msec] < 0xFFFF – The host waits for the device to respond before hibernating, with a defined time-out protection. This time-out defines the maximum time to wait. The NWP response can be sent earlier than this time-out.

*sl_DevSet* – This function configures different device parameters. The main parameters used are the DeviceSetID and Option. The possible DeviceSetID and Option combinations are:

- SL_DEVICE_GENERAL_CONFIGURATION – The general configuration options are:
  - SL_DEVICE_GENERAL_CONFIGURATION_DATE_TIME – Configures the device internal date and time. The time parameter is retained in hibernate mode but will reset at shutdown.

Setting device time and date example:

```c
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(SlDateTime_t), (unsigned char *) &dateTime);
```

*sl_DevGet* – This function lets the user read different device parameters. The main parameters used are the DeviceSetID and Option parameter. The possible DeviceSetID and Option combinations are:

- SL_DEVICE_GENERAL_CONFIGURATION
  - SL_DEVICE_GENERAL_CONFIGURATION_DATE_TIME
  - SL_DEVICE_GENERAL_VERSION – Returns the device firmware versions
- SL_DEVICE_STATUS – The device status options are:
  - SL_EVENT_CLASS_DEVICE – Possible values are:
    - EVENT_DROPPEDDEVICEASYNCGENERALERROR – General system error, check your system configuration.
    - STATUSDEVICESMARTCONFIGACTIVE – Device in SmartConfig mode.
  - SL_EVENT_CLASS_WLAN
    - EVENT_DROPPEDWLANWLANASYNCONNECTEDRESPONSE
    - EVENT_DROPPEDWLANWLANASYNCONNECTEDRESPONSE
    - EVENT_DROPPEDWLANSTACONNECTED
    - EVENT_DROPPEDWLANSTA_DISCONNECTED
    - STATUS_WLAN_STA_CONNECTED
  - SL_EVENT_CLASS_BSD – Possible values are:
    - EVENT_DROPPEDSOCKETTXFAILEDASYNCRESPONSE
  - SL_EVENT_CLASS_NETAPP – Possible values are:
- EVENT_DROPPED_NETAPP_IPACQUIRED
- EVENT_DROPPED_NETAPP_IPACQUIRED_V6
- EVENT_DROPPED_NETAPP_IP_LEASED
- EVENT_DROPPED_NETAPP_IP_RELEASED

- SL_EVENT_CLASS_NETCFG (Currently unused)
- SL_EVENT_CLASS_NVMEM (Currently unused)

Example for getting version:

```c
SlVersionFull ver;
pConfigOpt = SL_DEVICE_GENERAL_VERSION;
sl_DevGet(SL_DEVICE_GENERAL_CONFIGURATION, &pConfigOpt,&pConfigLen,
(unsigned char*)(&ver));
printf("CHIP %d\nMAC %d.%d.%d.%d\nPHY %d.%d.%d.%d\nNWP %d.%d.%d.%d\nHOST %d.%d.%d.%d\n", 
ver.ChipFwAndPhyVersion.ChipId,
ver.RomVersion, SL_MAJOR_VERSION_NUM, SL_MINOR_VERSION_NUM, SL_VERSION_NUM,
SL_SUB_VERSION_NUM);
```

`sl_EventMaskSet` – Masks asynchronous events from the device. Masked events do not generate asynchronous messages from the device. This function receives an EventClass and a bit mask. The events and mask options are:

- **SL_EVENT_CLASS_WLAN** user events:
  - 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** user events:
  - SL_DEVICE_FATAL_ERROR_EVENT
  - SL_DEVICE_ABORT_ERROR_EVENT

- **SL_EVENT_CLASS_BSD** user events:
  - SL_SOCKET_TX_FAILED_EVENT
  - SL_SOCKET_ASYNC_EVENT

- **SL_EVENT_CLASS_NETAPP** user events:
  - SL_NETAPP_IPV4_IPACQUIRED_EVENT
  - SL_NETAPP_IPV4_IPACQUIRED_V6_EVENT
  - SL_NETAPP_IP_RELEASED_EVENT
  - SL_NETAPP_HTTPGETTOKENVALUE_EVENT
  - SL_NETAPP_HTTPPOSTTOKENVALUE_EVENT

An example of masking out connection and disconnection from WLAN class:

```c
sl_EventMaskSet(SL_EVENT_CLASS_WLAN, (SL_WLAN_CONNECT_EVENT | SL_WLAN_DISCONNECT_EVENT) );
```
sl_EventMaskGet – Returns the events bit mask from the device. If that event is masked, the device does not send this event. The function is similar to sl_EventMaskSet.

An example of getting an event mask for WLAN class:

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

sl_EventMaskGet – Returns the events bit mask from the device. If an event is masked, the device does not send the event. The function is similar to sl_EventMaskSet.

An example of getting an event mask for WLAN class:

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

sl_Task

- **Non-Os Platform** – Should be called from the main loop
- **Multi-threaded Platform** – When the user does not implement the external spawn functions, the function should be called from a dedicated thread allocated to the SimpleLink driver. In this mode the function never returns.

sl_UartSetMode – This function should be used if the user’s chosen host interface is UART. The function sets the user’s UART configuration:

- Baud rate
- Flow control
- COM port

### 18.2 NetCfg

sl_NetCfgSet – Manages the configuration of the following networking functionalities:

- **SL_MAC_ADDRESS_SET** – The new MAC address overrides the default MAC address and is saved in the Serial Flash file system.
- **SL_IPV4_STA_P2P_CL_DHCP_ENABLE** – Sets the device to acquire an IP address by DHCP when in WLAN STA mode or P2P client. This is the default mode of the system for acquiring an IP address after a WLAN connection.
- **SL_IPV4_STA_P2P_CL_STATIC_ENABLE** – Sets a static IP address to the device working in STA mode or P2P client. The IP address is stored in the Serial Flash file system. To disable the static IP and get the address assigned from DHCP, use SL_STA_P2P_CL_IPV4_DHCP_SET.
- **SL_SET_HOST_RX_AGGR** - Turn on/off RX aggregation
  
  Example:
  ```c
  /* Disabling the Rx aggregation */
  _u8 RxAggrEnable = 0;
  sl_NetCfgSet(SL_SET_HOST_RX_AGGR, 0, sizeof(RxAggrEnable), (_u8 *)&RxAggrEnable);
  ```
- **SL_IPV4_AP_P2P_GO_STATIC_ENABLE** – Sets a static IP address to the device working in AP mode or P2P go. The IP address is stored in the Serial Flash file system.
  
  Example:
  ```c
  _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),(_u8 *)&ipV4);
  sl_Stop(0);
  sl_Start(NULL,NULL,NULL);
  ```
NOTE:

- AP mode must use static IP settings.
- All set functions require system restart for changes to take effect.

`sl_NetCfgGet` – Reads the network configurations. The options are:
- `SL_MAC_ADDRESS_GET` - For MAC address.
- `SL_IPV4_STA_P2P_CL_GET_INFO` - Gets an IP address from the WLAN station or P2P client. A DHCP flag is returned to indicate if the IP address is static or from DHCP.
- `SL_IPV4_AP_P2P_GO_GET_INFO` - Returns the IP address of the AP.

An example of getting an IP address from a WLAN station or P2P client:

```c
unsigned char dhcpIsOn = 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
", (dhcpIsOn > 0) ? "ON" : "OFF", SL_IPV4_BYTE(ipV4,3), SL_IPV4_BYTE(ipV4,2),
SL_IPV4_BYTE(ipV4,1), SL_IPV4_BYTE(ipV4,0), SL_IPV4_BYTE(ipV4Mask,3),
SL_IPV4_BYTE(ipV4Mask,2), SL_IPV4_BYTE(ipV4Mask,1),
SL_IPV4_BYTE(ipV4Mask,0), SL_IPV4_BYTE(ipV4Gateway,3),
SL_IPV4_BYTE(ipV4Gateway,2), SL_IPV4_BYTE(ipV4Gateway,1),
SL_IPV4_BYTE(ipV4Gateway,0), SL_IPV4_BYTE(ipV4DnsServer,3),
SL_IPV4_BYTE(ipV4DnsServer,2), SL_IPV4_BYTE(ipV4DnsServer,1),
SL_IPV4_BYTE(ipV4DnsServer,0));
```

18.3 WLAN

`sl_WlanSetMode` – The WLAN device has several WLAN modes of operation. By default, the device acts as a WLAN station, but it can also act in other WLAN roles. The different options are:
- `ROLE_STA` – For WLAN station mode
- `ROLE_AP` – For WLAN AP mode
- `ROLE_P2P` – For WLAN P2P mode

NOTE: The set mode functionality only takes effect in the next device boot.

An example of switching from any role to WLAN AP roles:

```c
sl_WlanSetMode(ROLE_AP);
```

```
/* Turning the device off and on in order for the roles change to take effect */
sl_Stop(0);
sl_Start(NULL,NULL,NULL);
```

`sl_WlanSet` – Lets the user configure different WLAN-related parameters. The main parameters used are ConfigID and ConfigOpt.

The possible ConfigID and ConfigOpt combinations are:
- `SL_WLAN_CFG_GENERAL_PARAM_ID` – The different general WLAN parameters are:
  - `WLANGENERALPARAM_OPT_COUNTRY_CODE`
  - `WLANGENERALPARAM_OPT_STA_TX_POWER` – Sets STA mode Tx power level, a number from 0 to 15, as the dB offset from max power (0 will set maximum power).
  - `WLANGENERALPARAM_OPT_AP_TX_POWER` – Sets AP mode Tx power level, a number from 0 to 15, as the dB offset from max power (0 will set maximum power).
- `SL_WLAN_CFG_AP_ID` – The different AP configuration options are:
- WLAN_AP_OPT_SSID
- WLAN_AP_OPT_CHANNEL
- WLAN_AP_OPT_HIDDEN_SSID – Sets the AP to be hidden or not hidden
- WLAN_AP_OPT_SECURITY_TYPE – Possible options are:
  - Open security: SL_SEC_TYPE_OPEN
  - WEP security: SL_SEC_TYPE_WEP
  - WPA security: SL_SEC_TYPE_WPA
- WLAN_AP_OPT_PASSWORD – Sets the security password for AP mode:
  - For WPA: 8 to 63 characters
  - For WEP: 5 to 13 characters (ASCII)
- SL_WLAN_CFG_P2P_PARAM_ID
- WLAN_P2P_OPT_DEV_NAME
- WLAN_P2P_OPT_DEV_TYPE
- WLAN_P2P_OPT_CHANNEL_N_REGS – The listen channel and regulatory class determine the device listen channel during the P2P find and listen phase. The operational channel and regulatory class determines the operating channel preferred by the device (if the device is the group owner, this is the operating channel). Channels should be one of the social channels (1, 6, or 11). If no listen or operational channel is selected, a random 1, 6, or 11 will be selected.
- WLAN_GENERAL_PARAM_OPT_INFO_ELEMENT – The application sets up to MAX_PRIVATE_INFO_ELEMENTS_SUPPORTED information elements per role (AP / P2P GO). To delete an information element, use the relevant index and length = 0. The application sets up to MAX_PRIVATE_INFO_ELEMENTS_SUPPORTED to the same role. However, for AP no more than INFO_ELEMENT_MAX_TOTAL_LENGTH_AP bytes are stored for all information elements. For P2P GO no more than INFO_ELEMENT_MAX_TOTAL_LENGTH_P2P_GO bytes are stored for all information elements.
- WLAN_GENERAL_PARAM_OPT_SCAN_PARAMS – Changes the scan channels and RSSI threshold

An example of setting SSID for AP mode:

```c
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 – Enables the user to configure different WLAN-related parameters. The main parameters used are ConfigID and ConfigOpt. The usage of sl_WlanGet is similar to sl_WlanSet.

sl_WlanConnect – Manually connects to a WLAN network

sl_WlanDisconnect – Disconnects WLAN connection

sl_WlanProfileAdd – When auto-start connection policy is enabled, the device connects to an AP from the profiles table. Up to seven profiles are supported. If several profiles are configured, the device selects the highest priority profile. Within each priority group, the device choses the profile based on the following parameters in descending priority: security policy, signal strength.

sl_WlanProfileGet – Reads a WLAN profile from the device

sl_WlanProfileDel – Deletes an existing profile

sl_WlanPolicySet – Manages the configuration of the following WLAN functionalities:
- SL_POLICY_CONNECTION – SL_POLICY_CONNECTION type defines three options available to connect the CC31xx device to the AP:
  - Auto Connect – The CC31xx device tries to automatically reconnect to one of its stored profiles each time the connection fails or the device is rebooted. To set this option, use:
    ```c
    sl_WlanPolicySet(SL_POLICY_CONNECTION, SL_CONNECTION_POLICY(1, 0, 0, 0, 0), NULL, 0)
    ```
- **Fast Connect** – The CC31xx device tries to establish a fast connection to AP. To set this option, use:

  \[
  \text{sl\_WlanPolicySet(SL\_POLICY\_CONNECTION,SL\_CONNECTION\_POLICY(0,1,0,0),NULL,0)}
  \]

- **P2P Connect** – If Any P2P mode is set, the CC31xx device tries to automatically connect to the first P2P device available, supporting push-button only. To set this option, use:

  \[
  \text{sl\_WlanPolicySet(SL\_POLICY\_CONNECTION,SL\_CONNECTION\_POLICY(0,0,0,1),NULL,0)}
  \]

- **Auto SmartConfig upon restart** – The device wakes up in SmartConfig mode. Any command from the host ends this state. To set this option use:

  \[
  \text{sl\_WlanPolicySet(SL\_POLICY\_CONNECTION,SL\_CONNECTION\_POLICY(0,0,0,0,1),NULL,0)}
  \]

- **SL\_POLICY\_SCAN** – Defines the system scan time interval if there is no connection. The default interval is 10 minutes. After the settings scan interval, an immediate scan is activated. The next scan is based on the interval settings. To set the scan interval to a 1-minute interval, use the following example:

  \[
  \text{unsigned long intervalInSeconds = 60; }
  \]

  \[
  \text{#define SL\_SCAN\_ENABLE 1}
  \]

  \[
  \text{sl\_WlanPolicySet(SL\_POLICY\_SCAN,SL\_SCAN\_ENABLE, (unsigned char *)}
  \]

  \[
  \text{intervalInSeconds,sizeof(intervalInSeconds));}
  \]

  To disable the scan, use:

  \[
  \text{#define SL\_SCAN\_DISABLE 0}
  \]

  \[
  \text{sl\_WlanPolicySet(SL\_POLICY\_SCAN,SL\_SCAN\_DISABLE,0,0);}
  \]

- **SL\_POLICY\_PM** – Defines a power-management policy for station mode only. Four power policies are available:
  - **SL\_NORMAL\_POLICY** (default) – For setting normal power-management policy use:

    \[
    \text{sl\_WlanPolicySet(SL\_POLICY\_PM , SL\_NORMAL\_POLICY, NULL,0)}
    \]

  - **SL\_ALWAYS\_ON\_POLICY** – For setting always-on power-management policy use:

    \[
    \text{sl\_WlanPolicySet(SL\_POLICY\_PM , SL\_ALWAYS\_ON\_POLICY, NULL,0)}
    \]

  - **SL\_LONG\_SLEEP\_INTERVAL\_POLICY** – For setting long-sleep interval policy use:

    \[
    \text{unsigned short PolicyBuff[4] = \{0,0,800,0\}; // 800 is max sleep time in mSec}
    \]

    \[
    \text{sl\_WlanPolicySet(SL\_POLICY\_PM , SL\_LONG\_SLEEP\_INTERVAL\_POLICY,}
    \]

    \[
    \text{PolicyBuff,sizeof(PolicyBuff));}
    \]

- **SL\_POLICY\_P2P** – Defines P2P negotiation policy parameters for a P2P role. To set the intent negotiation value, set one of the following:
  - **SL\_P2P\_ROLE\_NEGOTIATE** – intent 3
  - **SL\_P2P\_ROLE\_GROUP\_OWNER** – intent 15
  - **SL\_P2P\_ROLE\_CLIENT** – intent 0

  To set the negotiation initiator value (the initiator policy of the first negotiation action frame), set one of the following:
  - **SL\_P2P\_NEG\_INITIATOR\_ACTIVE**
  - **SL\_P2P\_NEG\_INITIATOR\_PASSIVE**
  - **SL\_P2P\_NEG\_INITIATOR\_RAND\_BACKOFF**

  For example:

  \[
  \text{set sl\_WlanPolicySet(SL\_POLICY\_P2P,}
  \]

  \[
  \text{SL\_P2P\_POLICY(SL\_P2P\_ROLE\_NEGOTIATE,SL\_P2P\_NEG\_INITIATOR\_RAND\_BACKOFF),NULL,0);}
  \]

- **sl\_WlanPolicyGet** – Reads the different WLAN policy settings. The possible options are:

  - **SL\_POLICY\_CONNECTION**
  - **SL\_POLICY\_SCAN**
  - **SL\_POLICY\_PM**
  - **SL\_POLICY\_P2P**
**Socket**

(sl_WlanGetNetworkList) – Gets the latest WLAN scan results

(sl_WlanSmartConfigStart) – Puts the device into SmartConfig state. Once SmartConfig has ended successfully, an asynchronous event will be received: SL_OPCODE_WLAN_SMART_CONFIG_START_ASYNC_RESPONSE. The event holds the SSID and an extra field that might also have been delivered (for example, device name).

(sl_WlanSmartConfigStop) – Stops the SmartConfig procedure. Once SmartConfig is stopped, an asynchronous event is received: SL_OPCODE_WLAN_SMART_CONFIG_STOP_ASYNC_RESPONSE

(sl_WlanRxStatStart) – Starts collecting WLAN Rx statistics (unlimited time)

(sl_WlanRxStatStop) – Stops collecting WLAN Rx statistics

(sl_WlanRxStatGet) – Gets WLAN Rx statistics. Upon calling this command, the statistics counters are cleared. The statistics returned are:
- Received valid packets – Sum of the packets received correctly (including filtered packets)
- Received FCS error packets – Sum of the packets dropped due to FCS error
- Received PLCP error packets – Sum of the packets dropped due to PLCP error
- Average data RSSI – Average RSSI for all valid data packets received
- Average management RSSI – Average RSSI for all valid management packets received
- Rate histogram – Rate histogram for all valid packets received
- RSSI histogram – RSSI histogram from -40 until -87 (all values below and above RSSI appear in the first and last cells)
- Start time stamp – The timestamp of starting to collect the statistics in µSec
- Get time stamp – The timestamp of reading the statistics in µSec

### 18.4 Socket

(sl_Socket) – Creates a new socket of a socket type identified by an integer number, and allocates system resources to the socket. The supported socket types are:
- SOCK_STREAM (TCP – Reliable stream-oriented service or Stream Sockets)
- SOCK_DGRAM (UDP – Datagram service or Datagram Sockets)
- SOCK_RAW (Raw protocols atop the network layer)

(sl_Close) – Gracefully closes socket. This function causes the system to release resources allocated to a socket. In case of TCP, the connection is terminated.

(sl_Accept) – This function is used with connection-based socket types (SOCK_STREAM) to extract the first connection request on the queue of pending connections, creates a new connected socket, and returns a new file descriptor referring to that socket. The newly created socket is not in the listening state. The original socket sd is unaffected by this call.

(sl_Bind) – Gives the socket the local address addr. addr is addrlen bytes long. Traditionally, this function is called when a socket is created, exists in a name space (address family) but has no name assigned. A local address must be assigned before a SOCK_STREAM socket receives connections.

(sl_Listen) – Specifies the willingness to accept incoming connections and a queue limit for incoming connections. The listen() call applies only to sockets of type SOCK_STREAM, and the backlog parameter defines the maximum length for the queue of pending connections.

(sl_Connect) – Connects the socket referred to by the socket descriptor sd to the address specified by addr. The addrlen argument specifies the size of addr. The format of the address in addr is determined by the address space of the socket. If the socket is of type SOCK_DGRAM, this call specifies the peer with which the socket is associated. Datagrams should be sent to this address, the only address from which datagrams should be received. If the socket is of type SOCK_STREAM, this call tries to make a connection to another socket. The other socket is specified by an address in the communications space of the socket.
**sl_Select** – Allows a program to monitor multiple file descriptors, waiting until one or more of the file descriptors become ready for a class of I/O operation. *sl_Select* has several subfunctions to set the file descriptor options:

- **SL_FD_SET** – Selects the `SlFdSet_t` SET function. Sets the current socket descriptor on the `SlFdSet_t` container.
- **SL_FD_CLR** – Selects the `SlFdSet_t` CLR function. Clears the current socket descriptor on the `SlFdSet_t` container.
- **SL_FD_ISSET** – Selects the `SlFdSet_t` ISSET function. Checks if the current socket descriptor is set (TRUE/FALSE).
- **SL_FD_ZERO** – Selects the `SlFdSet_t` ZERO function. Clears all socket descriptors from the `SlFdSet_t` container.

**sl_SetSockOpt** – Manipulates the options associated with a socket. Options exist at multiple protocol levels and are always present at the uppermost socket level. The supported socket options are:

- **SL_SOL_SOCKET** - Socket option category:
  - **SL_SO_KEEPALIVE** – Keeps TCP connections active by enabling the periodic transmission of messages; Enable or Disable, periodic keep alive. *Default: Enabled, keep alive time-out 300 seconds.*
  - **SL_SO_RCVTIMEO** – Sets the time-out value that specifies the maximum amount of time an input function waits until it completes. *Default: No time-out*
  - **SL_SO_RCVBUF** – Sets TCP max receive window
  - **SL_SO_NONBLOCKING** – Sets the socket to nonblocking operation. Impact on: *sl_Connect, sl_Accept, sl_Send, sl_Sendto, sl_Recv and sl_Recvfrom*. Default: Blocking.
  - **SL_SO_SECMETHOD** – Sets the method to the TCP-secured socket (SL_SEC_SOCKET). *Default: SL_SO_SEC_METHOD_SSLv3_TLSv1_2.*
  - **SL_SO_SECURE_MASK** – Sets a specific cipher to the TCP-secured socket (SL_SEC_SOCKET). *Default: “Best” cipher suitable to method*
  - **SL_SO_SECURE_FILES** – Maps programmed files to the secured socket (SL_SEC_SOCKET)
  - **SL_SO_SECURE_FILES_PRIVATE_KEY_FILE_NAME** - This option used to configure secure
  - **SL_SO_SECURE_FILES_CERTIFICATE_FILE_NAME** - This option used to configure secure
  - **SL_SO_SECURE_FILES_CA_FILE_NAME** - This option used to configure secure file
  - **SL_SO_SECURE_FILES_DH_KEY_FILE_NAME** - This option used to configure secure
  - **SL_SO_CHANGE_CHANNEL** – Sets the channel in transceiver mode

- **SL_IPPROTO_IP** - IP option category:
  - **SL_IP_MULTICAST_IF** - Specify outgoing multicast interface.
  - **SL_IP_MULTICAST_TTL** – Sets the time-to-live value of the outgoing multicast packets for the socket
  - **SL_IP_RAW_RX_NO_HEADER** – Raw socket; removes the IP header from received data. *Default: data includes IP header.*
  - **SL_IP_HDRINCL** – RAW socket only; the IPv4 layer generates an IP header when sending a packet unless the IP_HDRINCL socket option is enabled on the socket. When the socket option is enabled, the packet must contain an IP header. *Default: disabled, IPv4 header generated by Network Stack.*
  - **SL_IP_ADD_MEMBERSHIP** – UDP socket; joins a multicast group.
  - **SL_IP_DROP_MEMBERSHIP** – UDP socket; leaves a multicast group.

- **SL_SOL_PHY_OPT** - PHY option category:
  - **SL_SO_PHY_RATE** – RAW socket; sets WLAN PHY transmit rate.
  - **SL_SO_PHY_TX_POWER** – RAW socket; sets WLAN PHY Tx power.
  - **SL_SO_PHY_NUM_FRAMES_TO_TX** – RAW socket; sets the number of frames to transmit in transceiver mode.
  - **SL_SO_PHY_PREAMBLE** – RAW socket; sets WLAN PHY preamble.
**sl_GetSockOpt** – Manipulates the options associated with a socket. Options may exist at multiple protocol levels and are always present at the uppermost socket level. The socket options are the same as in **sl_SetSockOpt**.

**sl_Recv** – Reads data from the TCP socket

**sl_RecvFrom** – Reads data from the UDP socket

**sl_Send** – Writes data to the TCP socket. Returns immediately after sending data to device. If TCP fails, an async event SL_NETAPP_SOCKET_TX_FAILED is received. In case of a RAW socket (transceiver mode), an extra 4 bytes should be reserved at the end of the frame data buffer for WLAN FCS.

**sl_SendTo** – Writes data to the UDP socket. This function transmits a message to another socket (connectionless socket SOCK_DGRAM, SOCK_RAW). Returns immediately after sending data to the device. If transmission fails, an async event SL_NETAPP_SOCKET_TX_FAILED is received.

**sl_Htonl** – Reorders the bytes of a 32-bit unsigned value from processor order to network order.

**sl_Htons** – Reorders the bytes of a 16-bit unsigned value from processor order to network order.

### 18.5 NetApp

**sl_NetAppStart** – Enables or starts different networking services. Could be one or a combination of the following:

- **SL_NET_APP_HTTP_SERVER_ID** – HTTP server service
- **SL_NET_APP_DHCP_SERVER_ID** – DHCP server service (DHCP client is always supported)
- **SL_NET_APP_MDNS_ID** – MDNS Client/Server service

**sl_NetAppStop** – Disables or stops a networking service. Similar options as in **sl_NetAppStart**.

**sl_NetAppSet** Sets various network application parameters

- **SL_NET_APP_DHCP_SERVER_ID** - DHCP server identifier number
  - **NETAPP_SET_DHCP_SRV_BASIC_OPT** - DHCP server basic options configurations identifier (e.g. IP ranges, lease time)
- **SL_NET_APP_HTTP_SERVER_ID** - HTTP server identifier number
  - **NETAPP_SET_GET_HTTP_OPT_PORT_NUMBER** - HTTP server port number
  - **NETAPP_SET_GET_HTTP_OPT_AUTH_CHECK** - HTTP authentication check status
  - **NETAPP_SET_GET_HTTP_OPT_AUTH_NAME** - HTTP authentication name (Default: "admin")
  - **NETAPP_SET_GET_HTTP_OPT_AUTH_PASSWORD** - HTTP authentication password (Default: "admin")
  - **NETAPP_SET_GET_HTTP_OPT_AUTH_REALM** - HTTP authentication realm get (Default: "Simple Link CC31xx")
  - **NETAPP_SET_GET_HTTP_OPT_ROM_PAGES_ACCESS** - HTTP get status if default pages ("ROM") are allowed
- **SL_NET_APP_MDNS_ID**
  - **NETAPP_SET_GET_MDNS_CONT_QUERY_OPT** - MDNS set continuous query
  - **NETAPP_SET_GET_MDNS_QEVETN_MASK_OPT** - MDNS mask service types to ignore
  - **NETAPP_SET_GET_MDNS_TIMING_PARAMS_OPT** - MDNS reconfigure timing parameters for MDNS like factors, intervals
- **SL_NET_APP_DEVICE_CONFIG_ID**
  - **NETAPP_SET_GET_DEV_CONF_OPT_DEVICE_URN** - Set/Get device URN name
  - **NETAPP_SET_GET_DEV_CONF_OPT_DOMAIN_NAME** - Set/Get device domain name

**Example:**

```c
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, (void *)str, len);
```
len, (unsigned char*)str);

*sl_NetAppGet* – Retrieves various network application parameters.

*sl_NetAppDnsGetHostByName* – Obtains the IP address of a machine on the network, by machine name.

**Example:**

```c
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* – Returns service attributes such as IP address, port, and text according to the service name. The user sets a service name full or part (see the following example), and should get:

- The IP of service
- The port of service
- The text of service

This is similar to the GET host by name method, and is done with a single-shot query with PTR type on the service name. An example for full-service name follows:

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

An example for partial-service name follows:

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

*sl_NetAppGetServiceList* – Gets the list of peer services. The list is in a form of service structure. The user chooses the type of the service structure. The supported structures are:

- Full-service parameters with text
- Full-service parameters
- Short-service parameters (port and IP only), especially for tiny hosts

**NOTE:** The different types of structures save memory in the host.

*sl_NetAppMDNSRegisterService* – Registers a new mDNS service to the mDNS package and the DB. This registered service is offered by the application. The service name should be a full-service name according to DNS-SD RFC, meaning the value in the name field in the SRV answer.

**Example for service name:**

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

If the option is_unique is set, mDNS probes the service name to ensure it is unique before announcing the service on the network.

*sl_NetAppMDNSUnRegisterService* – Deletes the mDNS service from the mDNS package and the database.

*sl_NetAppPingStart* – Sends an ICMP ECHO_REQUEST (or ping) to the network hosts. The following is an example of sending 20 ping requests and reporting the results to a callback routine when all requests are sent:

```c
// 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);
}

18.6 File System

sl_FsOpen – Opens a file for read or write from or to Serial Flash storage AccessModeAndMaxSize.
Possible inputs are:
  • FS_MODE_OPEN_READ – Reads a file
  • FS_MODE_OPEN_WRITE – Opens an existing file for write
  • FS_MODE_OPEN_CREATE(maxSizeInBytes,accessModeFlags) – Opens for creating a new file. The maximum file size is defined in bytes. For optimal file system size, use maximum size in 4K to 512 bytes (for example, 3584, 7680). Several access modes can be combined together from SlFileOpenFlags_e.
Example:
   sl_FsOpen("FileName.html",FS_MODE_OPEN_CREATE(3584,_FS_FILE_OPEN_FLAG_COMMIT|_FS_FILE_PUBLIC_WRITE ),NULL, &FileHandle);

  • sl_FsRead – Reads a block of data from a file
  • sl_FsWrite – Writes a block of data to a file
  • sl_FsDel – Deletes a specific file from serial flash storage or all files (format)
  • sl_FsGetInfo – Returns the file information: flags, file size, allocated size, and tokens
# Chapter 19

## Asynchronous Events

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>19.1 Overview</td>
<td>148</td>
</tr>
<tr>
<td>19.2 WLAN Events</td>
<td>148</td>
</tr>
<tr>
<td>19.3 Netapp Events</td>
<td>149</td>
</tr>
<tr>
<td>19.4 Socket Events</td>
<td>150</td>
</tr>
<tr>
<td>19.5 Device Events</td>
<td>150</td>
</tr>
</tbody>
</table>
19.1 Overview

The SimpleLink host driver interacts with the SimpleLink device through commands transmitted to the device over the SPI or UART bus interface. Because some of the commands might trigger long processes that can take several hundred milliseconds or even seconds, the device and the host driver support a mechanism of asynchronous events sent from the device to the host driver.

Events notify on process completion such as a SmartConfig process done, notify on device status changes such as a WLAN disconnection event, or notify on errors such as a fatal error event.

These events can be classified in the following logical sections:

- WLAN events
- Network application events
- Socket events
- General device events

19.2 WLAN Events

SL_WLAN_CONNECT_EVENT – Notifies that the device is connected to the AP.

Event parameters:
- connection_type
- ssid_len
- ssid_name
- go_peer_device_name_len – relevant for P2P
- go_peer_device_name
- bssid

SL_WLAN_DISCONNECT_EVENT – Notifies that the device is disconnected from the AP.

Event parameters:
- connection_type
- ssid_len
- ssid_name
- go_peer_device_name_len – Relevant for P2P
- go_peer_device_name
- bssid
- reason_code – WLAN disconnection reason

SL_WLAN_SMART_CONFIG_START_EVENT – Notifies the host that SmartConfig has ended. Event parameters:
- Status
- ssid_len
- ssid
- private_token_len
- private_token

SL_WLAN_SMART_CONFIG_STOP_EVENT – Notifies the host that SmartConfig has stopped.

Event parameters:
- status

SL_WLAN_STA_CONNECTED_EVENT – Notifies that STA is connected; relevant in AP mode or P2P GO.

Event parameters:
- 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 – Notifies that STA is disconnected; relevant in AP mode or P2P GO.

Event parameters:
• Status
• 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 – Notifies that the device is found; relevant in P2P mode.

Event parameters:
• 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 – Notifies that the negotiation request received an event; relevant in P2P mode.

Event parameters:
• go_peer_device_name
• mac
• go_peer_device_name_len
• wps_dev_password_id
• own_ssid
• own_ssid_len

SL_WLAN_CONNECTION_FAILED_EVENT – Notifies negotiation failure; relevant in P2P mode.

Event parameters:
• status

19.3 Netapp Events

SL_NETAPP_IPV4_IPACQUIRED_EVENT – Notifies IPv4 enquired.

Event parameters:
• ip
• gateway
• dns

SL_NETAPP_IP_LEASED_EVENT – Notifies STA IP lease; relevant in AP or P2P GO mode.
Event parameters:
- ip_address
- lease_time
- mac

SL_NETAPP_IP_RELEASED_EVENT – Notifies STA IP release; relevant in AP or P2P GO mode.

Event parameters:
- ip_address
- mac
- reason

SL_NETAPP_HTTPGETTOKENVALUE_EVENT – Notifies token is missing. Tries to retrieve this value from host.

Event parameters:
- httpTokenName
- httpTokenName length

SL_NETAPP_HTTPPOSTTOKENVALUE_EVENT – Notifies a new post with the included parameters.

Event parameters:
- action
- action length
- token_name
- token_name length
- token_value
- token_value length

19.4 Socket Events

SL_SOCKET_TX_FAILED_EVENT – Notifies of TX failure.

Event parameters:
- Status
- sd

SL_SOCKET_ASYNC_EVENT – Notifies of asynchronous event.

Event parameters:
- sd
- type – The event type can be one of the following:
  - SSL_ACCEPT – Accept failed due to SSL issue (TCP pass)
  - RX_FRAGMENTATION_TOO_BIG – Connectionless mode, RX packet fragmentation > 16K, packet is released.
  - OTHER_SIDE_CLOSE_SSL_DATA_NOT_ENCRYPTED – Remote side down from secure to unsecure
- value

19.5 Device Events

- SL_DEVICE_FATAL_ERROR_EVENT – Notifies of fatal error; must perform device reset. Event parameters:
  - Status: An error code indication from the device
  - Sender: The sender originator which is based on SlErrorSender_e enum
- SL_DEVICE_ABORT_ERROR_EVENT - Indicates a severe error occurred and the device stopped:
– AbortType: An indication of the event type
– AbortData: Additional info about the data error
20.1 General

This chapter describes the requirements, design and implementation of info elements. The Host should be able to add / remove Information elements from Beacons / Probe Responses at AP/P2P GO Roles and also allows obtaining published IEs from connected AP/P2P GO or peering STA/P2P Client (proprietary). The purpose of this capability is to allow publishing vendor-specific information using 802.11 standard capabilities.

Figure 20-1 shows the Information Element declaration taken from 802.11 Spec.

---

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 \( j \leq n \leq 255 \). The length of the vendor-specific content is \( n-j \) octets.

Octets: 

<table>
<thead>
<tr>
<th>Element ID</th>
<th>Length</th>
<th>Organization Identifier (see 8.4.1.31)</th>
<th>Vendor-specific content</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>( j )</td>
<td>( n-j )</td>
</tr>
</tbody>
</table>

**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.

---

**Figure 20-1. 802.11 Spec - Info Element**

Info elements can have the same ID and OUI and will be differ only in the Payload.

**Figure 20-2** shows that four information elements exist with same ID (221) and same OUI (00:40:96).
20.2 Interface to Application

Host has a single API, using sl_WlanSet(), which allows adding and removing the info element. After command is executed, Basic Response Event will be sent to the Host.

Example:

```c
sl_protocol_WlanSetInfoElement_t infoele;
infoele.index = Index; // Index of the info element. range: 0 -
MAX_PRIVATE_INFO_ELEMENTS_SUPPORTED
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;
infoele.ie.oui[1] = Oui1;
infoele.ie.oui[2] = Oui2;
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 \([0-252]\)

```c
sl_WlanSet(SL_WLAN_CFG_GENERAL_PARAM_ID,WLAN_GENERAL_PARAM_OPT_INFO_ELEMENT,sizeof(sl_protocol_WlanSetInfoElement_t),(_u8*) &infoele);
```

```c
sl_WlanSet(SL_WLAN_CFGGENERAL_PARAM_ID,WLAN_GENERAL_PARAM_OPT_INFO_ELEMENT,sizeof(sl_protocol_WlanSetInfoElement_t),(_u8*) &infoele);
```

**Table 20-1. API Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Number of Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>1</td>
<td>Index of the entry</td>
</tr>
<tr>
<td>Role</td>
<td>1</td>
<td>AP = 0, P2P GO = 1</td>
</tr>
<tr>
<td>Index</td>
<td>1</td>
<td>Index of the IE</td>
</tr>
<tr>
<td>Role</td>
<td>1</td>
<td>0 – AP, 1 – P2P GO</td>
</tr>
<tr>
<td>ID</td>
<td>1</td>
<td>IE number (null = vendor-specific ([221]))</td>
</tr>
<tr>
<td>OUI</td>
<td>3</td>
<td>Organization ID in a vendor-specific IE (null = MAC_ADDR_OUI)</td>
</tr>
<tr>
<td>IE Length</td>
<td>2</td>
<td>IE Payload Length. 0 means remove this IE. Value is in the range of ([0-252]). Total length of all configured info elements should not be greater than ((INFO_ELEMENT_MAX_TOTAL_LENGTH))</td>
</tr>
<tr>
<td>IE</td>
<td>IE Length (0-252)</td>
<td>Payload of information element</td>
</tr>
</tbody>
</table>

- When a User wants to add/remove info element it will specify the index of the entry it wishes to add or remove.
- If the user set the ID to be 0 it will be replaced in the NWP with a value of 221.
- If the user set the OUI to be 00:00:00 it will be replaced in the NWP with a value of 08:00:28 (OUI of Texas Instruments).

The following structure is used for the command.

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

Where `sl_protocol_InfoElement_t` is defined as follow:

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

The Host can configure up to 4 IE’s.

```c
#define MAX_PRIVATE_INFO_ELEMENTS_SUPPORTED 4
```

**Info Element max size is 252 Bytes**

```c
#define INFO_ELEMENT_MAX_SIZE 252
```

The total length of all info elements at AP mode is 300 bytes (for example - 4 info elements of 75 bytes each)

```c
#define INFO_ELEMENT_MAX_TOTAL_LENGTH_AP 300
```

The total length of all info elements at P2P GO mode is 160 bytes

```c
#define INFO_ELEMENT_MAX_TOTAL_LENGTH_AP 160
```

---

**NOTE:** This limit includes the ID (1 byte) + length (1) + OUI (3).

The Role is defined as follow:
#define INFO_ELEMENT_AP_ROLE 0
#define INFO_ELEMENT_P2P_GO_ROLE 1

20.2.1 API Output

Basic response event will be transmitted by the NWP at completion of the command processing.

A return code will reflect the success of the operation.

Possible error code:
• STATUS_OK (0) – success
• ERROR_INVALID_PARAM(-2) – invalid length (IE length is greater than 252 bytes or overall length of new IE length with all configured lengths is greater than INFO_ELEMENT_MAX_TOTAL_LENGTH bytes)
• ERROR_MALLOC_FAILED (-4) – length of the IE will pass the INFO_ELEMENT_MAX_TOTAL_LENGTH threshold
• ERROR_FS_ACCESS_FAILED (-7) – fail to get info elements from memory

20.3 Total Maximum Size of all Information Elements

The Beacon & Probe response template frame in the MAC is 512 Bytes each; do not add private information elements that will cause the frame to be larger than this limit.

Beacon & Probe response frames include information elements that:
• Always exist with static length (TIM)
• Always exist with varying length (SID)
• Not always exist (WPA)
• Exist in P2P GO but not in AP or vice versa (P2P IE)
• Exist with different length in Probe Res and in the Beacon (WPS in Probe Response)

Therefore, the Info Elements that we can add will have different size limit at AP and P2P GO.

NOTE: If the Probe Response was transmitted by the NWP, the size is limited in case FW transmits the frame. Since we want private IE’s to be identical in all frames, bigger IE’s are not allowed in such cases.

Table 20-2 summarizes all of the fields included in a Beacon & Probe response frames at SimpleLink WiFi device.

Table 20-2. Beacon & Probe Response Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>AP id</th>
<th>Size</th>
<th>Beacon</th>
<th>Probe</th>
<th>P2P Beacon</th>
<th>P2P Probe</th>
<th>More Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAC Header</td>
<td>24</td>
<td>0</td>
<td>24</td>
<td>24</td>
<td>24</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>timestamp</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>beacon interval</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>capability info</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>SSID</td>
<td>32</td>
<td>32</td>
<td>34</td>
<td>34</td>
<td>34</td>
<td>34</td>
<td></td>
</tr>
<tr>
<td>supported rates</td>
<td>1</td>
<td>8</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Direct Seq</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>TIM</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>country</td>
<td>7</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>ERP</td>
<td>42</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>EX Rates</td>
<td>50</td>
<td>4</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>RSN</td>
<td>48</td>
<td>20</td>
<td>22</td>
<td>22</td>
<td>22</td>
<td>22</td>
<td>20 in P2P, 24 in AP</td>
</tr>
</tbody>
</table>
Table 20-2. Beacon & Probe Response Parameters (continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>id</th>
<th>Size</th>
<th>Beacon</th>
<th>Probe</th>
<th>Beacon</th>
<th>Probe</th>
<th>More Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>WPA</td>
<td>221</td>
<td>26</td>
<td>28</td>
<td>28</td>
<td>0</td>
<td>0</td>
<td>WPA is not presented in P2P with WPS</td>
</tr>
<tr>
<td>HT capabilities</td>
<td>45</td>
<td>26</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>not set</td>
</tr>
<tr>
<td>HT operation</td>
<td>61</td>
<td>22</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>not set</td>
</tr>
<tr>
<td>WMM</td>
<td>221</td>
<td>24</td>
<td>0</td>
<td>0</td>
<td>26</td>
<td>26</td>
<td>Set in Autonomous P2P GO (due to certification)</td>
</tr>
<tr>
<td>WPS</td>
<td>221</td>
<td>178</td>
<td>0</td>
<td>0</td>
<td>80</td>
<td>180</td>
<td>OUI=0050F2, type=4. only in P2P:78 in beacon, 178 in probe response (depend if config method if presented and on the Device name (up-to 33 chars)</td>
</tr>
<tr>
<td>P2P (wifi direct)</td>
<td>221</td>
<td>61</td>
<td>0</td>
<td>0</td>
<td>20</td>
<td>0</td>
<td>OUI=506F9A, type=9. 18 in beacon, 61 in probe response. Inside Probe response, it is contained only if transmitted by the supplicant. It is not part of Probe Response template!!!</td>
</tr>
<tr>
<td>Summary</td>
<td></td>
<td>449</td>
<td>156</td>
<td>150</td>
<td>254</td>
<td>328</td>
<td></td>
</tr>
</tbody>
</table>

As can be seen, Beacon & Probe response frames at AP mode consume 160 bytes. At P2P GO, Beacon consumes 238 Bytes and Probe Response consumes 348 bytes. This leaves approximately 350 bytes at AP role for private info elements and approximately 120 bytes at P2P GO role for private info elements.
References

- CC3100 SimpleLink™ Wi-Fi® Network Processor, Internet-of-Things Solution for MCU Applications Data Sheet (SWAS031)
- CC3200 SimpleLink™ Wi-Fi® and Internet-of-Things Solution, a Single-Chip Wireless MCU Data Sheet (SWAS032)
- CC31xx Host Driver APIs
- CC31xx Host Interface wiki
- www.ti.com/simplelinkwifi
- CC31xx SimpleLink wiki
- TI E2E™ Online Community — TI's Engineer-to-Engineer (E2E) Community. Created to foster collaboration among engineers. At e2e.ti.com, you can ask questions, share knowledge, explore ideas and help solve problems with fellow engineers.
A.1 Overview

From a software perspective, the system can be separated into several parts:

- User application
- SimpleLink WiFi host driver – Platform-independent part
  - Host driver API
  - Main driver logic and flow
- SimpleLink WiFi host driver – Platform-dependent part
  - OS wrapper implementation
  - Transport layer (SPI and UART) implementation

Figure A-1. SimpleLink WiFi Host Driver Configuration
A.1.1 *SimpleLink WiFi Host Driver -- Platform-Independent Part*

The host driver implementation includes the driver API, SimpleLink networking initialization, SimpleLink networking commands, commands response handling, asynchronous event handling, data flow, and transport layer interface. All are platform-independent and OS-independent code provided by TI. The driver APIs are organized into six silos reflecting six different logical API types:

- **Device API** – Handles the hardware-related API
- **WLAN API** – Handles the WLAN, 802.11 protocol-related functionality
- **Socket API** – The most common API set to be used by the user application. The SimpleLink networking socket API complies with the Berkeley socket APIs.
- **NetApp API** – Handles additional networking protocols and functionality, delivered as a complementary part of the on-chip content.
- **NetCfg API** – Handles configuration of different networking parameters
- **FS API** – Handles access to the Serial Flash component, for read and write operations of networking or user proprietary data

A.1.2 *SimpleLink WiFi Host Driver – Platform-Dependent Part*

The driver has two software components supplied by the user:

- **Transport layer implementation** – *Mandatory*
  The driver code provides the required transport and bus interfaces. The user must provide the function implementation per the transport layer chosen (SPI or UART), and the platform used.
- **OS Wrapper** – *Optional*
  The driver code provides the required OS wrapper interface. To use an OS, first provide the function implementation per the OS and platform used.

A.1.3 *SimpleLink WiFi Driver Configuration*

The driver provides a configuration file, allowing the user to control the supported API sets, memory allocation module, and OS use. In addition, some preconfiguration options are provided. A preconfiguration is a set of customizations and settings of the driver already made and tested by TI for a specific scenario.

A.1.4 *User Application*

The user application is developed and owned by the user. The application interfaces with the SimpleLink networking driver using the driver APIs and asynchronous driver events.

A.2 *Driver Data Flows*

A.2.1 *Transport Layer Protocol*

The following types of messages are used by the host driver:

- Command [ Host -> SimpleLink network processor ]
- Command Complete [ SimpleLink network processor -> Host ]
- Async Event [ SimpleLink network processor -> Host ]
- Data [ Host <-> SimpleLink network processor ]
A.2.2 Command and Command Complete

A command is any message from the host to the SimpleLink network processor device which is not a data packet (send or receive). The host driver supports only one command at a time. Before sending the next command, the driver waits for a 'command complete' message received from the SimpleLink network processor. To avoid blocking the driver for long periods of time in terms of RT and embedded systems, in some commands the SimpleLink network processor sends the 'command complete' message immediately after the command is received and validated, and sends an asynchronous event when the command executes successfully.

![Blocked Link Diagram]

Figure A-2. Blocked Link

A.2.3 Data Transactions

A data message is any information (usually TCP or UDP packets) sent to or from the user using the socket send and receive APIs.

A.2.3.1 Data Send (From Host to SimpleLink Network Processor)

Sending data to the SimpleLink network processor device is similar to a command flow. The difference is that there is no command complete message for a data packet, thus the host is not blocked or waiting for a message. Data packets can be sent sequentially from the user application; the driver is responsible for avoiding buffer overrun by using data flow control logic.

A.2.3.2 Data Flow Control

Status Field – Part of each message (async event) from SimpleLink network processor device to the host:

- TxBuff – Indicates the number of available Tx buffers within the SimpleLink network processor device.

![Data Flow Control]

Figure A-3. Data Flow Control

- The host can send up to TxBuff packets (<1500 bytes) without waiting for a response from the SimpleLink network processor device.
- The SimpleLink network processor device generates a dummy event if the number is changed (threshold from last update / time-out if change is smaller than the threshold).
A.2.3.3 Data Receive (From SimpleLink Network Processor to Host)

Receiving data from the SimpleLink network processor device is done in two manners: blocking and nonblocking.

A.2.3.4 Blocking Receive

Upon a call to a blocking `sl_Recv`, the host driver issues a command to the SimpleLink network processor device. The driver is blocked, waiting until an async event notifies the driver of a data packet ready to be read within the SimpleLink network processor device. After receiving this async event, the driver continues to read the data packet from the host.

A.2.3.5 NonBlocking Receive

If the command complete status returns with a status notifying that no data is waiting for the host, the command returns with pending return status (SL_EAGAIN). If data is waiting for the host, the driver continues to read the data packet from the host.
## Appendix B

### Error Codes

#### B.1 Error Codes

#### Table B-1. General Error Codes

<table>
<thead>
<tr>
<th>Error Name</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL_RET_CODE_OK</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>SL_RET_CODE_INVALID_INPUT</td>
<td>-2</td>
<td></td>
</tr>
<tr>
<td>SL_RET_CODE_SELF_ERROR</td>
<td>-3</td>
<td></td>
</tr>
<tr>
<td>SL_RET_CODE_NWP_IF_ERROR</td>
<td>-4</td>
<td></td>
</tr>
<tr>
<td>SL_RET_CODE_MALLOC_ERROR</td>
<td>-5</td>
<td></td>
</tr>
</tbody>
</table>

#### Table B-2. Device Error Codes

<table>
<thead>
<tr>
<th>Error Name</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL_ERROR_STATIC_ADDR_SUBNET_ERROR</td>
<td>-60</td>
<td>Network stack error</td>
</tr>
<tr>
<td>SL_ERROR_ILLEGAL_CHANNEL</td>
<td>-61</td>
<td>Supplicant error</td>
</tr>
<tr>
<td>SL_ERROR_SUPPLICANT_ERROR</td>
<td>-72</td>
<td>init error code</td>
</tr>
<tr>
<td>SL_ERROR_HOSTAPD_INIT_FAIL</td>
<td>-73</td>
<td>init error code</td>
</tr>
<tr>
<td>SL_ERROR_HOSTAPD_INIT_IF_FAIL</td>
<td>-74</td>
<td>init error code</td>
</tr>
<tr>
<td>SL_ERROR_WLAN_DRV_INIT_FAIL</td>
<td>-75</td>
<td>init error code</td>
</tr>
<tr>
<td>SL_ERROR_WLAN_DRV_START_FAIL</td>
<td>-76</td>
<td>wlan start error</td>
</tr>
<tr>
<td>SL_ERROR_FS_FILE_TABLE_LOAD_FAILED</td>
<td>-77</td>
<td>init file system failed</td>
</tr>
<tr>
<td>SL_ERROR_PREFERRED_NETWORKS_FILE_LOAD_FAILED</td>
<td>-78</td>
<td>init file system failed</td>
</tr>
<tr>
<td>SL_ERROR_HOSTAPD_BSSID_VALIDATION_ERROR</td>
<td>-79</td>
<td>Ap configurations BSSID error</td>
</tr>
<tr>
<td>SL_ERROR_HOSTAPD_FAILED_TO_SETUP_INTERFACE</td>
<td>-80</td>
<td>Ap configurations interface error</td>
</tr>
<tr>
<td>SL_ERROR_MDNS_ENABLE_FAIL</td>
<td>-81</td>
<td>mDNS enable failed</td>
</tr>
<tr>
<td>SL_ERROR_HTTP_SERVER_ENABLE_FAILED</td>
<td>-82</td>
<td>HTTP server enable failed</td>
</tr>
<tr>
<td>SL_ERROR_DHCP_SERVER_ENABLE_FAILED</td>
<td>-83</td>
<td>DHCP server enable failed</td>
</tr>
<tr>
<td>SL_ERROR_PREFERRED_NETWORK_LIST_FULL</td>
<td>-93</td>
<td>Supplicant error</td>
</tr>
<tr>
<td>SL_ERROR_PREFERRED_NETWORKS_FILE_WRITE_FAILED</td>
<td>-94</td>
<td>Supplicant error</td>
</tr>
<tr>
<td>SL_ERROR_DHCP_CLIENT_RENEW_FAILED</td>
<td>-100</td>
<td>DHCP client error</td>
</tr>
<tr>
<td>SL_ERROR_CON_MGMT_STATUS_UNSPECIFIED</td>
<td>-102</td>
<td>WLAN Connection</td>
</tr>
<tr>
<td>SL_ERROR_CON_MGMT_STATUS_AUTH_REJECT</td>
<td>-103</td>
<td>WLAN Connection</td>
</tr>
<tr>
<td>SL_ERROR_CON_MGMT_STATUS_ASSOC_REJECT</td>
<td>-104</td>
<td>WLAN Connection</td>
</tr>
<tr>
<td>SL_ERROR_CON_MGMT_STATUS_SECURITY_FAILURE</td>
<td>-105</td>
<td>WLAN Connection</td>
</tr>
<tr>
<td>SL_ERROR_CON_MGMT_STATUS_AP_DEAUTHENTICATE</td>
<td>-106</td>
<td>WLAN Connection</td>
</tr>
<tr>
<td>SL_ERROR_CON_MGMT_STATUS_AP_DISASSOCIATE</td>
<td>-107</td>
<td>WLAN Connection</td>
</tr>
<tr>
<td>SL_ERROR_CON_MGMT_STATUS_ROAMING_TRIGGER</td>
<td>-108</td>
<td>WLAN Connection</td>
</tr>
<tr>
<td>SL_ERROR_CON_MGMT_STATUS_DISCONNECT_DURING_CONNECT</td>
<td>-109</td>
<td>WLAN Connection</td>
</tr>
<tr>
<td>SL_ERROR_CON_MGMT_STATUS_SG_RESELECT</td>
<td>-110</td>
<td>WLAN Connection</td>
</tr>
<tr>
<td>SL_ERROR_CON_MGMT_STATUS_ROC_RESELECT</td>
<td>-111</td>
<td>WLAN Connection</td>
</tr>
<tr>
<td>SL_ERROR_CON_MGMT_STATUS_MIC_FAILURE</td>
<td>-112</td>
<td>WLAN Connection</td>
</tr>
</tbody>
</table>
### Table B-2. Device Error Codes (continued)

<table>
<thead>
<tr>
<th>Error Name</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL_ERROR_WAKELOCK_ERROR_PREFIX</td>
<td>-115</td>
<td>Wake lock expired</td>
</tr>
<tr>
<td>SL_ERROR_LENGTH_ERROR_PREFIX</td>
<td>-116</td>
<td>Uart header length error</td>
</tr>
<tr>
<td>SL_ERROR_MDNS_CREATE_FAIL</td>
<td>-121</td>
<td>mDNS create failed</td>
</tr>
<tr>
<td>SL_ERROR_GENERAL_ERROR</td>
<td>-127</td>
<td></td>
</tr>
</tbody>
</table>

### Table B-3. Socket Error Codes

<table>
<thead>
<tr>
<th>Error Name</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL_SOC_OK</td>
<td>0</td>
<td>Socket command in execution</td>
</tr>
<tr>
<td>SL_SOC_ERROR</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>SL_INEXE</td>
<td>-8</td>
<td>Socket command in execution</td>
</tr>
<tr>
<td>SL_EBADF</td>
<td>-9</td>
<td>Bad file number</td>
</tr>
<tr>
<td>SL_ENSOCK</td>
<td>-10</td>
<td>Max number of socket has been reached</td>
</tr>
<tr>
<td>SL_EAGAIN</td>
<td>-11</td>
<td>Try Again (for non blocking command)</td>
</tr>
<tr>
<td>SL_ENOMEM</td>
<td>-12</td>
<td>Out of memory</td>
</tr>
<tr>
<td>SL_EACCES</td>
<td>-13</td>
<td>Permission denied</td>
</tr>
<tr>
<td>SL_EFAULT</td>
<td>-14</td>
<td>Bad address</td>
</tr>
<tr>
<td>SL_ECLOSE</td>
<td>-15</td>
<td>Close socket operation failed to transmit all queued packets</td>
</tr>
<tr>
<td>SL_EALREADY_ENABLED</td>
<td>-21</td>
<td>Transceiver - Transceiver already ON</td>
</tr>
<tr>
<td>SL EINVAL</td>
<td>-22</td>
<td>Invalid argument</td>
</tr>
<tr>
<td>SL_EAUTO_CONNECT_OR_CONNECTING</td>
<td>-69</td>
<td>Transceiver - During connection, connected or auto mode started</td>
</tr>
<tr>
<td>SL_CONNECTION_PENDING</td>
<td>-72</td>
<td>Transceiver - Device is connected, disconnect first to open transceiver</td>
</tr>
<tr>
<td>SL_EUNSUPPORTED_ROLE</td>
<td>-86</td>
<td>Transceiver - Trying to start when WLAN role is AP or P2P GO</td>
</tr>
<tr>
<td>SL_EDESTADDRREQ</td>
<td>-89</td>
<td>Destination address required</td>
</tr>
<tr>
<td>SL_EPROTOTYPE</td>
<td>-91</td>
<td>Protocol wrong type for socket</td>
</tr>
<tr>
<td>SL_ENOPROTOOPT</td>
<td>-92</td>
<td>Protocol not available</td>
</tr>
<tr>
<td>SL_EPROTONOSUPPORT</td>
<td>-93</td>
<td>Protocol not supported</td>
</tr>
<tr>
<td>SL_ESOCKTNOSUPPORT</td>
<td>-94</td>
<td>Socket type not supported</td>
</tr>
<tr>
<td>SL_EOPNOTSUPP</td>
<td>-95</td>
<td>Operation not supported on transport endpoint</td>
</tr>
<tr>
<td>SL_EAFNOSUPPORT</td>
<td>-97</td>
<td>Address family not supported by protocol</td>
</tr>
<tr>
<td>SL_EADDRINUSE</td>
<td>-98</td>
<td>Address already in use</td>
</tr>
<tr>
<td>SL_EADDRNOTAVAIL</td>
<td>-99</td>
<td>Cannot assign requested address</td>
</tr>
<tr>
<td>SL_ENETUNREACH</td>
<td>-101</td>
<td>Network is unreachable</td>
</tr>
<tr>
<td>SL_ENOBUFS</td>
<td>-105</td>
<td>No buffer space available</td>
</tr>
<tr>
<td>define SL_EISCONN</td>
<td>-106</td>
<td>Transport endpoint is already connected</td>
</tr>
<tr>
<td>SL_ENOTCONN</td>
<td>-107</td>
<td>Transport endpoint is not connected</td>
</tr>
<tr>
<td>SLETIMEOUT</td>
<td>-110</td>
<td>Connection timed out</td>
</tr>
<tr>
<td>SL_ECONNREFUSED</td>
<td>-111</td>
<td>Connection refused</td>
</tr>
<tr>
<td>SL_EALREADY</td>
<td>-114</td>
<td>Non blocking connect in progress, try again</td>
</tr>
<tr>
<td>SL_ESEC_RSA_WRONG_TYPE_E</td>
<td>-130</td>
<td>RSA wrong block type for RSA function</td>
</tr>
<tr>
<td>SL_ESEC_RSA_BUFFER_E</td>
<td>-131</td>
<td>RSA buffer error, output too small</td>
</tr>
<tr>
<td>SL_ESEC_BUFFER_E</td>
<td>-132</td>
<td>Output buffer too small or input too large</td>
</tr>
<tr>
<td>SL_ESEC_ALGO_ID_E</td>
<td>-133</td>
<td>Setting algo id error</td>
</tr>
<tr>
<td>SL_ESEC_PUBLIC_KEY_E</td>
<td>-134</td>
<td>Setting public key error</td>
</tr>
</tbody>
</table>
Table B-3. Socket Error Codes (continued)

<table>
<thead>
<tr>
<th>Error Name</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL_ESEC_DATE_E</td>
<td>-135</td>
<td>Setting date validity error</td>
</tr>
<tr>
<td>SL_ESEC_SUBJECT_E</td>
<td>-136</td>
<td>Setting subject name error</td>
</tr>
<tr>
<td>SL_ESEC_ISSUER_E</td>
<td>-137</td>
<td>Setting issuer name error</td>
</tr>
<tr>
<td>SL_ESEC_CA_TRUE_E</td>
<td>-138</td>
<td>Setting CA basic constraint true error</td>
</tr>
<tr>
<td>SL_ESEC_EXTENSIONS_E</td>
<td>-139</td>
<td>Setting extensions error</td>
</tr>
<tr>
<td>SL_ESECASN_PARSE_E</td>
<td>-140</td>
<td>ASN parsing error, invalid input</td>
</tr>
<tr>
<td>SL_ESEC_VERSION_E</td>
<td>-141</td>
<td>ASN version error, invalid number</td>
</tr>
<tr>
<td>SL_ESEC_GETINT_E</td>
<td>-142</td>
<td>ASN get big _i16 error, invalid data</td>
</tr>
<tr>
<td>SL_ESEC_RSA_KEY_E</td>
<td>-143</td>
<td>ASN key init error, invalid input</td>
</tr>
<tr>
<td>SL_ESEC_OBJECT_ID_E</td>
<td>-144</td>
<td>ASN object id error, invalid id</td>
</tr>
<tr>
<td>SL_ESEC_TAG_NULL_E</td>
<td>-145</td>
<td>ASN tag error, not null</td>
</tr>
<tr>
<td>SL_ESEC_EXPECT_0_E</td>
<td>-146</td>
<td>ASN expect error, not zero</td>
</tr>
<tr>
<td>SL_ESEC_BITSTR_E</td>
<td>-147</td>
<td>ASN bit string error, wrong id</td>
</tr>
<tr>
<td>SL_ESEC_UNKNOWN_OID_E</td>
<td>-148</td>
<td>ASN oid error, unknown sum id</td>
</tr>
<tr>
<td>SL_ESEC_DATE_SZ_E</td>
<td>-149</td>
<td>ASN date error, bad size</td>
</tr>
<tr>
<td>SL_ESEC_BEFORE_DATE_E</td>
<td>-150</td>
<td>ASN date error, current date before</td>
</tr>
<tr>
<td>SL_ESEC_AFTER_DATE_E</td>
<td>-151</td>
<td>ASN date error, current date after</td>
</tr>
<tr>
<td>SL_ESEC_SIG_OID_E</td>
<td>-152</td>
<td>ASN signature error, mismatched oid</td>
</tr>
<tr>
<td>SL_ESEC_TIME_E</td>
<td>-153</td>
<td>ASN time error, unknown time type</td>
</tr>
<tr>
<td>SL_ESEC_INPUT_E</td>
<td>-154</td>
<td>ASN input error, not enough data</td>
</tr>
<tr>
<td>SL_ESEC_SIG_CONFIRM_E</td>
<td>-155</td>
<td>ASN sig error, confirm failure</td>
</tr>
<tr>
<td>SL_ESEC_SIG_HASH_E</td>
<td>-156</td>
<td>ASN sig error, unsupported hash type</td>
</tr>
<tr>
<td>SL_ESEC_SIG_KEY_E</td>
<td>-157</td>
<td>ASN sig error, unsupported key type</td>
</tr>
<tr>
<td>SL_ESEC_DH_KEY_E</td>
<td>-158</td>
<td>ASN key init error, invalid input</td>
</tr>
<tr>
<td>SL_ESEC_NTRU_KEY_E</td>
<td>-159</td>
<td>ASN ntru key decode error, invalid input</td>
</tr>
<tr>
<td>SL_ESEC_ECC_BAD_ARG_E</td>
<td>-170</td>
<td>ECC input argument of wrong type</td>
</tr>
<tr>
<td>SL_ESEC_ECC_KEY_E</td>
<td>-171</td>
<td>ASN ECC bad input</td>
</tr>
<tr>
<td>SL_ESEC_CURVE_OID_E</td>
<td>-172</td>
<td>Unsupported ECC OID curve type</td>
</tr>
<tr>
<td>SL_ESEC_BAD_FUNC_ARG</td>
<td>-173</td>
<td>Bad function argument provided</td>
</tr>
<tr>
<td>SL_ESEC_NOT_COMPILED_IN</td>
<td>-174</td>
<td>Feature not compiled in</td>
</tr>
<tr>
<td>SL_ESEC_UNICODE_SIZE_E</td>
<td>-175</td>
<td>Unicode password too big</td>
</tr>
<tr>
<td>SL_ESEC_NO_PASSWORD</td>
<td>-176</td>
<td>No password provided by user</td>
</tr>
<tr>
<td>SL_ESEC_ALT_NAME_E</td>
<td>-177</td>
<td>alt name size problem, too big</td>
</tr>
<tr>
<td>SL_ESEC_AES_GCM_AUTH_E</td>
<td>-180</td>
<td>AES-GCM Authentication check failure</td>
</tr>
<tr>
<td>SL_ESEC_AES_CCM_AUTH_E</td>
<td>-181</td>
<td>AES-CCM Authentication check failure</td>
</tr>
<tr>
<td>SL_ESEC_CLOSE_NOTIFY</td>
<td>-300</td>
<td>ssl/tls alerts</td>
</tr>
<tr>
<td>SL_ESEC_UNEXPECTED_MESSAGE</td>
<td>-310</td>
<td>ssl/tls alerts</td>
</tr>
<tr>
<td>SL_ESEC_BAD_RECORD_MAC</td>
<td>-320</td>
<td>ssl/tls alerts</td>
</tr>
<tr>
<td>SL_ESEC_DECRIPTION_FAILED</td>
<td>-321</td>
<td>ssl/tls alerts</td>
</tr>
<tr>
<td>SL_ESEC_RECORD_OVERFLOW</td>
<td>-322</td>
<td>ssl/tls alerts</td>
</tr>
<tr>
<td>SL_ESEC_DECOMPRESSION_FAILURE</td>
<td>-330</td>
<td>ssl/tls alerts</td>
</tr>
<tr>
<td>SL_ESEC_HANDSHAKE_FAILURE</td>
<td>-340</td>
<td>ssl/tls alerts</td>
</tr>
<tr>
<td>SL_ESEC_NO_CERTIFICATE</td>
<td>-341</td>
<td>ssl/tls alerts</td>
</tr>
<tr>
<td>SL_ESEC_BAD_CERTIFICATE</td>
<td>-342</td>
<td>ssl/tls alerts</td>
</tr>
<tr>
<td>SL_ESEC_UNSUPPORTED_CERTIFICATE</td>
<td>-343</td>
<td>ssl/tls alerts</td>
</tr>
<tr>
<td>SL_ESEC_CERTIFICATE_REVOKED</td>
<td>-344</td>
<td>ssl/tls alerts</td>
</tr>
<tr>
<td>SL_ESEC_CERTIFICATE_EXPIRED</td>
<td>-345</td>
<td>ssl/tls alerts</td>
</tr>
</tbody>
</table>
### Table B-3. Socket Error Codes (continued)

<table>
<thead>
<tr>
<th>Error Name</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL_ESEC_CERTIFICATE_UNKNOWN</td>
<td>-346</td>
<td>ssl/tls alerts</td>
</tr>
<tr>
<td>SL_ESEC_ILLEGAL_PARAMETER</td>
<td>-347</td>
<td>ssl/tls alerts</td>
</tr>
<tr>
<td>SL_ESEC_UNKNOWN_CA</td>
<td>-348</td>
<td>ssl/tls alerts</td>
</tr>
<tr>
<td>SL_ESEC_ACCESS_DENIED</td>
<td>-349</td>
<td>ssl/tls alerts</td>
</tr>
<tr>
<td>SL_ESEC_DECODE_ERROR</td>
<td>-350</td>
<td>ssl/tls alerts</td>
</tr>
<tr>
<td>SL_ESEC_DECRYPT_ERROR</td>
<td>-351</td>
<td>ssl/tls alerts</td>
</tr>
<tr>
<td>SL_ESEC_EXPORT_RESTRICTION</td>
<td>-360</td>
<td>ssl/tls alerts</td>
</tr>
<tr>
<td>SL_ESEC_PROTOCOL_VERSION</td>
<td>-370</td>
<td>ssl/tls alerts</td>
</tr>
<tr>
<td>SL_ESEC_INSUFFICIENT_SECURITY</td>
<td>-370</td>
<td>ssl/tls alerts</td>
</tr>
<tr>
<td>SL_ESEC_INTERNAL_ERROR</td>
<td>-380</td>
<td>ssl/tls alerts</td>
</tr>
<tr>
<td>SL_ESEC_USER_CANCELLED</td>
<td>-390</td>
<td>ssl/tls alerts</td>
</tr>
<tr>
<td>SL_ESEC_NO_RENEGOTIATION</td>
<td>-400</td>
<td>ssl/tls alerts</td>
</tr>
<tr>
<td>SL_ESEC_UNSUPPORTED_EXTENSION</td>
<td>-410</td>
<td>ssl/tls alerts</td>
</tr>
<tr>
<td>SL_ESEC_CERTIFICATE_UNOBTAINABLE</td>
<td>-411</td>
<td>ssl/tls alerts</td>
</tr>
<tr>
<td>SL_ESEC_UNRECOGNIZED_NAME</td>
<td>-412</td>
<td>ssl/tls alerts</td>
</tr>
<tr>
<td>SL_ESEC_BAD_CERTIFICATE_STATUS_RESPONSE</td>
<td>-413</td>
<td>ssl/tls alerts</td>
</tr>
<tr>
<td>SL_ESEC_BAD_CERTIFICATE_HASH_VALUE</td>
<td>-414</td>
<td>ssl/tls alerts</td>
</tr>
<tr>
<td>SL_ESECGENERAL</td>
<td>-450</td>
<td>Error secure level general error</td>
</tr>
<tr>
<td>SL_ESECDECRIPT</td>
<td>-451</td>
<td>Error secure level, decrypt recv packet fail</td>
</tr>
<tr>
<td>SL_ESECCLOSED</td>
<td>-452</td>
<td>Secure layer is closed by other size , tcp is still connected</td>
</tr>
<tr>
<td>SL_ESECSNOVERIFY</td>
<td>-453</td>
<td>Connected without server verification</td>
</tr>
<tr>
<td>SL_ESECNOCAFILE</td>
<td>-454</td>
<td>Error secure level CA file not found</td>
</tr>
<tr>
<td>SL_ESECMEMORY</td>
<td>-455</td>
<td>Error secure level No memory space available</td>
</tr>
<tr>
<td>SL_ESECBADCAFILE</td>
<td>-456</td>
<td>Error secure level, bad CA file</td>
</tr>
<tr>
<td>SL_ESECBADCERTFILE</td>
<td>-457</td>
<td>Error secure level bad Certificate file</td>
</tr>
<tr>
<td>SL_ESECBADPRIVATEFILE</td>
<td>-458</td>
<td>Error secure level bad private file</td>
</tr>
<tr>
<td>SL_ESECBADDHFILE</td>
<td>-459</td>
<td>Error secure level bad DH file</td>
</tr>
<tr>
<td>SL_ESECT00MANYSSLOPENED</td>
<td>-460</td>
<td>MAX SSL Sockets are opened</td>
</tr>
<tr>
<td>SL_ESECDATEERROR</td>
<td>-461</td>
<td>Connected with certificate date verification error</td>
</tr>
<tr>
<td>SL_ESECHANDSHAKETIMEDOUT</td>
<td>-462</td>
<td>Connection timed out due to handshake time</td>
</tr>
</tbody>
</table>

### Table B-4. WLAN Error Codes

<table>
<thead>
<tr>
<th>Error Name</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL_ERROR_KEY_ERROR</td>
<td>-3</td>
<td></td>
</tr>
<tr>
<td>SL_ERROR_WIFI_NOT_CONNECTED</td>
<td>-59</td>
<td></td>
</tr>
<tr>
<td>SL_ERROR_INVALID_ROLE</td>
<td>-71</td>
<td></td>
</tr>
<tr>
<td>SL_ERROR_INVALID_SECURITY_TYPE</td>
<td>-84</td>
<td></td>
</tr>
<tr>
<td>SL_ERROR_PASSPHRASE_TOO_LONG</td>
<td>-85</td>
<td></td>
</tr>
<tr>
<td>SL_ERROR_WPS_NO_PIN_OR_WRONG_PIN_LEN</td>
<td>-87</td>
<td></td>
</tr>
<tr>
<td>SL_ERROR_EAP_WRONG_METHOD</td>
<td>-88</td>
<td></td>
</tr>
<tr>
<td>SL_ERROR_PASSWORD_ERROR</td>
<td>-89</td>
<td></td>
</tr>
<tr>
<td>SL_ERROR_EAP_ANONYMOUS_LEN_ERROR</td>
<td>-90</td>
<td></td>
</tr>
<tr>
<td>SL_ERROR_SSID_LEN_ERROR</td>
<td>-91</td>
<td></td>
</tr>
<tr>
<td>SL_ERROR_USER_ID_LEN_ERROR</td>
<td>-92</td>
<td></td>
</tr>
</tbody>
</table>
Table B-4. WLAN Error Codes (continued)

<table>
<thead>
<tr>
<th>Error Name</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL_ERROR_ILLEGAL_WEP_KEY_INDEX</td>
<td>-95</td>
<td></td>
</tr>
<tr>
<td>SL_ERROR_INVALID_DWELL_TIME_VALUES</td>
<td>-96</td>
<td></td>
</tr>
<tr>
<td>SL_ERROR_INVALID_POLICY_TYPE</td>
<td>-97</td>
<td></td>
</tr>
<tr>
<td>SL_ERROR_PM_POLICY_INVALID_OPTION</td>
<td>-98</td>
<td></td>
</tr>
<tr>
<td>SL_ERROR_PM_POLICY_INVALID_PARAMS</td>
<td>-99</td>
<td></td>
</tr>
<tr>
<td>SL_ERROR_WIFI_ALREADY_DISCONNECTED</td>
<td>-129</td>
<td></td>
</tr>
</tbody>
</table>

Table B-5. NetApp Error Code

<table>
<thead>
<tr>
<th>Error Name</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL_ERROR_DEVICE_NAME_LEN_ERR</td>
<td>-117</td>
<td>Set Device name error</td>
</tr>
<tr>
<td>SL_ERROR_DEVICE_NAME_INVALID</td>
<td>-118</td>
<td>Set Device name error</td>
</tr>
<tr>
<td>SL_ERROR_DOMAIN_NAME_LEN_ERR</td>
<td>-119</td>
<td>Set domain name error</td>
</tr>
<tr>
<td>SL_ERROR_DOMAIN_NAME_INVALID</td>
<td>-120</td>
<td>Set domain name error</td>
</tr>
<tr>
<td>SL_NET_APP_DNS_QUERY_NO_RESPONSE</td>
<td>-159</td>
<td>DNS query failed, no response</td>
</tr>
<tr>
<td>SL_NET_APP_DNS_NO_SERVER</td>
<td>-161</td>
<td>No DNS server was specified</td>
</tr>
<tr>
<td>SL_NET_APP_DNS_PARAM_ERROR</td>
<td>-162</td>
<td>mDNS parameters error</td>
</tr>
<tr>
<td>SL_NET_APP_DNS_QUERY_FAILED</td>
<td>-163</td>
<td>DNS query failed</td>
</tr>
<tr>
<td>SL_NET_APP_DNS_INTERNAL_1</td>
<td>-164</td>
<td></td>
</tr>
<tr>
<td>SL_NET_APP_DNS_INTERNAL_2</td>
<td>-165</td>
<td></td>
</tr>
<tr>
<td>SL_NET_APP_DNS_MALFORMED_PACKET</td>
<td>-166</td>
<td>Improperly formed or corrupted DNS packet</td>
</tr>
<tr>
<td></td>
<td></td>
<td>received</td>
</tr>
<tr>
<td>SL_NET_APP_DNS_INTERNAL_3</td>
<td>-167</td>
<td></td>
</tr>
<tr>
<td>SL_NET_APP_DNS_INTERNAL_4</td>
<td>-168</td>
<td></td>
</tr>
<tr>
<td>SL_NET_APP_DNS_INTERNAL_5</td>
<td>-169</td>
<td></td>
</tr>
<tr>
<td>SL_NET_APP_DNS_INTERNAL_6</td>
<td>-170</td>
<td></td>
</tr>
<tr>
<td>SL_NET_APP_DNS_INTERNAL_7</td>
<td>-171</td>
<td></td>
</tr>
<tr>
<td>SL_NET_APP_DNS_INTERNAL_8</td>
<td>-172</td>
<td></td>
</tr>
<tr>
<td>SL_NET_APP_DNS_INTERNAL_9</td>
<td>-173</td>
<td></td>
</tr>
<tr>
<td>SL_NET_APP_DNS_MISMATCHED_RESPONSE</td>
<td>-174</td>
<td>Server response type does not match the query request</td>
</tr>
<tr>
<td>SL_NET_APP_DNS_INTERNAL_10</td>
<td>-175</td>
<td></td>
</tr>
<tr>
<td>SL_NET_APP_DNS_INTERNAL_11</td>
<td>-176</td>
<td></td>
</tr>
<tr>
<td>SL_NET_APP_DNS_NO_ANSWER</td>
<td>-177</td>
<td>No response for one-shot query</td>
</tr>
<tr>
<td>SL_NET_APP_DNS_NO_KNOWN_ANSWER</td>
<td>-178</td>
<td>No known answer for query</td>
</tr>
<tr>
<td>SL_NET_APP_DNS_NAME_MISMATCH</td>
<td>-179</td>
<td>Illegal service name according to the RFC</td>
</tr>
<tr>
<td>SL_NET_APP_DNS_NOT_STARTED</td>
<td>-180</td>
<td>mDNS is not running</td>
</tr>
<tr>
<td>SL_NET_APP_DNS_HOST_NAME_ERROR</td>
<td>-181</td>
<td>Host name error</td>
</tr>
<tr>
<td>SL_NET_APP_DNS_NO_MORE_ENTRIES</td>
<td>-182</td>
<td>No more entries be found</td>
</tr>
<tr>
<td>SL_NET_APP_DNS_MAX_SERVICES_ERROR</td>
<td>-200</td>
<td>Maximum advertise services are already configured</td>
</tr>
<tr>
<td>SL_NET_APP_DNS_IDENTICAL_SERVICE_ERROR</td>
<td>-201</td>
<td>Trying to register a service that is already exists</td>
</tr>
<tr>
<td>SL_NET_APP_DNS_NOT_EXISTED_SERVICE_ERROR</td>
<td>-203</td>
<td>Trying to delete service that does not existed</td>
</tr>
<tr>
<td>SL_NET_APP_DNS_ERROR_SERVICE_NAME_ERROR</td>
<td>-204</td>
<td>Retry request</td>
</tr>
<tr>
<td>SL_NET_APP_DNS_RX_PACKET_ALLOCATION_ERROR</td>
<td>-205</td>
<td></td>
</tr>
<tr>
<td>SL_NET_APP_DNS_BUFFER_SIZE_ERROR</td>
<td>-206</td>
<td>List size buffer is bigger than internally allowed in the NWP</td>
</tr>
</tbody>
</table>
### Table B-5. NetApp Error Code (continued)

<table>
<thead>
<tr>
<th>Error Name</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL_NET_APP_DNS_NET_APP_SET_ERROR</td>
<td>-207</td>
<td>Illegal length of one of the mDNS Set functions</td>
</tr>
<tr>
<td>SL_NET_APP_DNS_GET_SERVICE_LIST_FLAG_ERROR</td>
<td>-208</td>
<td></td>
</tr>
<tr>
<td>SL_NET_APP_DNS_NO_CONFIGURATION_ERROR</td>
<td>-209</td>
<td></td>
</tr>
<tr>
<td>SL_ERROR_NETAPP_RX_BUFFER_LENGTH_ERROR</td>
<td>-230</td>
<td></td>
</tr>
</tbody>
</table>

### Table B-6. FS Error Codes

<table>
<thead>
<tr>
<th>Error Name</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL_FS_OK</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>SL_FS_ERR_NOT_SUPPORTED</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>SL_FS_ERR_FAILED_TO_READ</td>
<td>-2</td>
<td></td>
</tr>
<tr>
<td>SL_FS_ERR_INVALID_MAGIC_NUM</td>
<td>-3</td>
<td></td>
</tr>
<tr>
<td>SL_FS_ERR_DEVICE_NOT_LOADED</td>
<td>-4</td>
<td></td>
</tr>
<tr>
<td>SL_FS_ERR_FAILED_TO_CREATE_LOCK_OBJ</td>
<td>-5</td>
<td></td>
</tr>
<tr>
<td>SL_FS_ERR_UNKNOWN</td>
<td>-6</td>
<td></td>
</tr>
<tr>
<td>SL_FS_ERR_FS_ALREADY_LOADED</td>
<td>-7</td>
<td></td>
</tr>
<tr>
<td>SL_FS_ERR_FAILED_TO_CREATE_FILE</td>
<td>-8</td>
<td></td>
</tr>
<tr>
<td>SL_FS_ERR_INVALID_ARGS</td>
<td>-9</td>
<td></td>
</tr>
<tr>
<td>SL_FS_ERR_EMPTY_ERROR</td>
<td>-10</td>
<td></td>
</tr>
<tr>
<td>SL_FS_ERR_FILE_NOT_EXISTS</td>
<td>-11</td>
<td></td>
</tr>
<tr>
<td>SL_FS_ERR_INVALID_FILE_ID</td>
<td>-12</td>
<td></td>
</tr>
<tr>
<td>SL_FS_ERR_READ_DATA_LENGTH</td>
<td>-13</td>
<td></td>
</tr>
<tr>
<td>SL_FS_ERR_ALLOC</td>
<td>-14</td>
<td></td>
</tr>
<tr>
<td>SL_FS_ERR_OFFSET_OUT_OF_RANGE</td>
<td>-15</td>
<td></td>
</tr>
<tr>
<td>SL_FS_ERR_FAILED_TO_WRITE</td>
<td>-16</td>
<td></td>
</tr>
<tr>
<td>SL_FS_ERR_INVALID_HANDLE</td>
<td>-17</td>
<td></td>
</tr>
<tr>
<td>SL_FS_ERR_FAILED_LOAD_FILE</td>
<td>-18</td>
<td></td>
</tr>
<tr>
<td>SL_FS_ERR_CONTINUE_WRITE_MUST_BE_MOD_4</td>
<td>-19</td>
<td></td>
</tr>
<tr>
<td>SL_FS_ERR_FAILED_INIT_STORAGE</td>
<td>-20</td>
<td></td>
</tr>
<tr>
<td>SL_FS_ERR_FAILED_READ_NVFILE</td>
<td>-21</td>
<td></td>
</tr>
<tr>
<td>SL_FS_ERR_BAD_FILE_MODE</td>
<td>-22</td>
<td></td>
</tr>
<tr>
<td>SL_FS_ERR_FILE_ACCESS_IS_DIFFERENT</td>
<td>-23</td>
<td></td>
</tr>
<tr>
<td>SL_FS_ERR_NO_ENTRIES_AVAILABLE</td>
<td>-24</td>
<td></td>
</tr>
<tr>
<td>SL_FS_ERR_PROGRAM</td>
<td>-25</td>
<td></td>
</tr>
<tr>
<td>SL_FS_ERR_FILE_ALREADY_EXISTS</td>
<td>-26</td>
<td></td>
</tr>
<tr>
<td>SL_FS_ERR_INVALID_ACCESS_TYPE</td>
<td>-27</td>
<td></td>
</tr>
<tr>
<td>SL_FS_ERR_FILE_EXISTS_ON_DIFFERENT_DEVICE_ID</td>
<td>-28</td>
<td></td>
</tr>
<tr>
<td>SL_FS_ERR_FILE_MAX_SIZE_BIGGER_THAN_EXISTING_FILE</td>
<td>-29</td>
<td></td>
</tr>
<tr>
<td>SL_FS_ERR_NO_AVAILABLE_BLOCKS</td>
<td>-30</td>
<td></td>
</tr>
<tr>
<td>SL_FS_ERR_FAILED_TO_READ_INTEGRITY_HEADER_1</td>
<td>-31</td>
<td></td>
</tr>
<tr>
<td>SL_FS_ERR_FAILED_TO_READ_INTEGRITY_HEADER_2</td>
<td>-32</td>
<td></td>
</tr>
<tr>
<td>SL_FS_ERR_FAILED_TO_ALLOCATE_MEM</td>
<td>-33</td>
<td></td>
</tr>
<tr>
<td>SL_FS_ERR_NO_AVAILABLE_NV_INDEX</td>
<td>-34</td>
<td></td>
</tr>
<tr>
<td>SL_FS_ERR_FAILED_WRITE_NVMEM_HEADER</td>
<td>-35</td>
<td></td>
</tr>
<tr>
<td>SL_FS_ERR_DEVICE_IS_NOT_FORMATTED</td>
<td>-36</td>
<td></td>
</tr>
<tr>
<td>SL_FS_WARNING_FILE_NAME_NOT_KEPT</td>
<td>-37</td>
<td></td>
</tr>
<tr>
<td>SL_FS_ERR_SIZE_OF_FILE_EXT_EXCEEDED</td>
<td>-38</td>
<td></td>
</tr>
</tbody>
</table>
### Table B-6. FS Error Codes (continued)

<table>
<thead>
<tr>
<th>Error Name</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL_FS_ERR_FILE_IMAGE_IS_CORRUPTED</td>
<td>-39</td>
<td></td>
</tr>
<tr>
<td>SL_FS_INVALID_BUFFER_FOR_WRITE</td>
<td>-40</td>
<td></td>
</tr>
<tr>
<td>SL_FS_INVALID_BUFFER_FOR_READ</td>
<td>-41</td>
<td></td>
</tr>
<tr>
<td>SL_FS_FILE_MAX_SIZE_EXCEEDED</td>
<td>-42</td>
<td></td>
</tr>
<tr>
<td>SL_FS_ERR_MAX_FS_FILES_IS_SMALLER</td>
<td>-43</td>
<td></td>
</tr>
<tr>
<td>SL_FS_ERR_MAX_FS_FILES_IS_LARGER</td>
<td>-44</td>
<td></td>
</tr>
<tr>
<td>SL_FS_FILE_HAS_RESERVED_NV_INDEX</td>
<td>-45</td>
<td></td>
</tr>
<tr>
<td>SL_FS_ERR_OVERLAP_DETECTION_THRESHOLD</td>
<td>-46</td>
<td></td>
</tr>
<tr>
<td>SL_FS_DATA_IS_NOTAligned</td>
<td>-47</td>
<td></td>
</tr>
<tr>
<td>SL_FS_DATA_ADDRESS_SHOULD_BE_IN_DATA_RAM</td>
<td>-48</td>
<td></td>
</tr>
<tr>
<td>SL_FS_NO_DEVICE_IS_LOADED</td>
<td>-49</td>
<td></td>
</tr>
<tr>
<td>SL_FS_ERR_TOKEN_IS_NOT_VALID</td>
<td>-50</td>
<td></td>
</tr>
<tr>
<td>SL_FS_FILE_INVALID_FILE_SIZE</td>
<td>-51</td>
<td></td>
</tr>
<tr>
<td>SL_FS_SECURITY_ALERT</td>
<td>-52</td>
<td></td>
</tr>
<tr>
<td>SL_FS_FILE_SYSTEM_IS_LOCKED</td>
<td>-53</td>
<td></td>
</tr>
<tr>
<td>SL_FS_WRONG_FILE_NAME</td>
<td>-54</td>
<td></td>
</tr>
<tr>
<td>SL_FS_ERR_FAILED_READ_NVMEM_HEADER</td>
<td>-55</td>
<td></td>
</tr>
<tr>
<td>SL_FS_ERR_INCORRECT_OFFSET_ALIGNMENT</td>
<td>-56</td>
<td></td>
</tr>
<tr>
<td>SL_FS_SECURE_FILE_MUST_BE_COMMIT</td>
<td>-57</td>
<td></td>
</tr>
<tr>
<td>SL_FS_SECURITY_BUF_ALREADY_ALLOC</td>
<td>-58</td>
<td></td>
</tr>
<tr>
<td>SL_FS_FILE_NAME_EXIST</td>
<td>-59</td>
<td></td>
</tr>
<tr>
<td>SL_FS_CERT_CHAIN_ERROR</td>
<td>-60</td>
<td></td>
</tr>
<tr>
<td>SL_FS_WRONG_FILE_NAME</td>
<td>-61</td>
<td></td>
</tr>
<tr>
<td>SL_FS_NOT_16_ALIGNED</td>
<td>-62</td>
<td></td>
</tr>
<tr>
<td>SL_FS_WRONG_SIGNATURE_OR_CERTIFIC_NAME_LENGTH</td>
<td>-63</td>
<td></td>
</tr>
<tr>
<td>SL_FS_WRONG_SIGNATURE</td>
<td>-64</td>
<td></td>
</tr>
<tr>
<td>SL_FS_FILE_HAS_NOT_BEEN_CLOSE_CORRECTLY</td>
<td>-65</td>
<td></td>
</tr>
<tr>
<td>SL_FS_ERASING_FLASH</td>
<td>-66</td>
<td></td>
</tr>
<tr>
<td>SL_FS_ERR_FILE_IS_NOT_SECURE_AND_SIGN</td>
<td>-67</td>
<td></td>
</tr>
<tr>
<td>SL_FS_ERR_EMPTY_SFLASH</td>
<td>-68</td>
<td></td>
</tr>
</tbody>
</table>

### Table B-7. Rx Filter Error Codes

<table>
<thead>
<tr>
<th>Error Name</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>RXFL_OK</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>RXFL_NUMBER_OF_FILTER_EXCEEDED</td>
<td>23</td>
<td>Number of max filters exceeded</td>
</tr>
<tr>
<td>RXFL_NO_FILTERS_ARE_DEFINED</td>
<td>24</td>
<td>No filters are defined in the system</td>
</tr>
<tr>
<td>RXFL_UPDATE_NOT_SUPPORTED</td>
<td>31</td>
<td>Update not supported</td>
</tr>
<tr>
<td>RXFL_RULE_HEADER_FIELD_ID_OUT_OF_RANGE</td>
<td>32</td>
<td>rule field Id is out of range</td>
</tr>
<tr>
<td>RXFL_RULE_HEADER_COMBINATION_OPERATOR_OUT_OF_RANGE</td>
<td>33</td>
<td>Combination function Id is out of range</td>
</tr>
<tr>
<td>RXFL_RULE_HEADER_OUT_OF_RANGE</td>
<td>34</td>
<td>The header rule is out of range</td>
</tr>
<tr>
<td>RXFL_RULE_HEADER_NOT_SUPPORTED</td>
<td>35</td>
<td>The header rule is not supported on current release</td>
</tr>
<tr>
<td>RXFL_RULE_HEADER_FIELD_ID_ASCII_NOT_SUPPORTED</td>
<td>36</td>
<td>This ASCII field Id is not supported</td>
</tr>
<tr>
<td>RXFL_RULE_FIELD_ID_NOT_SUPPORTED</td>
<td>37</td>
<td>Rule field Id is out of range</td>
</tr>
<tr>
<td>RXFL_FRAME_TYPE_NOT_SUPPORTED</td>
<td>38</td>
<td>ASCII frame type string is illegal</td>
</tr>
<tr>
<td>RXFL_RULE_HEADER_COMPARE_FUNC_OUT_OF_RANGE</td>
<td>39</td>
<td>The rule compare function is out of range</td>
</tr>
<tr>
<td>RXFL_RULE_HEADER_TRIGGER_OUT_OF_RANGE</td>
<td>40</td>
<td>The Trigger is out of range</td>
</tr>
<tr>
<td>Error Name</td>
<td>Value</td>
<td>Comment</td>
</tr>
<tr>
<td>---------------------------------------------------------------</td>
<td>-------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>RXFL_RULE_HEADER_TRIGGER_COMPARE_FUNC_OUT_OF_RANGE</td>
<td>41</td>
<td>The Trigger comparison function is out of range</td>
</tr>
<tr>
<td>RXFL_RULE_HEADER_ACTION_TYPE_NOT_SUPPORTED</td>
<td>42</td>
<td>The action type is not supported</td>
</tr>
<tr>
<td>RXFL_DEPENDENT_FILTER_DO_NOT_EXIST_1</td>
<td>43</td>
<td>The parent filter is null</td>
</tr>
<tr>
<td>RXFL_DEPENDENT_FILTER_DO_NOT_EXIST_2</td>
<td>44</td>
<td>The parent filter don't exist</td>
</tr>
<tr>
<td>RXFL_DEPENDENT_FILTER_SYSTEM_STATE_DO_NOT_FIT</td>
<td>45</td>
<td>The filter and its dependency system state don't fit</td>
</tr>
<tr>
<td>RXFL_DEPENDENT_FILTER_LAYER_DO_NOT_FIT</td>
<td>46</td>
<td>The filter and its dependency should be from the same layer</td>
</tr>
<tr>
<td>RXFL_ACTION_NO_REG_NUMBER</td>
<td>47</td>
<td>Action require counter number</td>
</tr>
<tr>
<td>RXFL_NUMBER_OF_ARGS_EXCEEDED</td>
<td>48</td>
<td>Number of arguments exceeded</td>
</tr>
<tr>
<td>RXFL_DEPEDENCY_NOT_ON_THESAME_LAYER</td>
<td>49</td>
<td>The filter and its dependency must be on the same layer</td>
</tr>
<tr>
<td>RXFL_FILTER_DO_NOT_EXISTS</td>
<td>50</td>
<td>The filter doesn't exists</td>
</tr>
<tr>
<td>RXFL_DEPENDENT_FILTER_DEPENDENCY_ACTION_IS_DROP</td>
<td>51</td>
<td>The dependent filter has Drop action, thus the filter can't be created</td>
</tr>
<tr>
<td>RXFL_NUMBER_OF_CONNECTION_POINTS_EXCEEDED</td>
<td>52</td>
<td>Number of connection points exceeded</td>
</tr>
<tr>
<td>RXFL_DEPENDENCY_IS_DISABLED</td>
<td>58</td>
<td>Can't enable filer in case its dependency filter is disabled</td>
</tr>
<tr>
<td>RXFL_CHILD_IS_ENABLED</td>
<td>59</td>
<td>Can't disable filter while the child is enabled</td>
</tr>
<tr>
<td>RXFL_FILTER_HAS_CHILDREN</td>
<td>60</td>
<td>The filter has children and can't be removed</td>
</tr>
<tr>
<td>RXFL_DEPENDENT_FILTER_IS_NOT_ENABLED</td>
<td>61</td>
<td>The dependency filter is not enabled</td>
</tr>
<tr>
<td>RXFL_DEPENDENT_FILTER_IS_NOT_PERSISTENT</td>
<td>62</td>
<td>The dependency filter is not persistent</td>
</tr>
<tr>
<td>RXFL_WRONG_MULTICAST_ADDRESS</td>
<td>63</td>
<td>The address should be of multicast type</td>
</tr>
<tr>
<td>RXFL_WRONGCOMPARE_FUNC_FOR_BROADCAST_ADDRESS</td>
<td>64</td>
<td>The compare function is not suitable for broadcast address</td>
</tr>
<tr>
<td>RXFL_THE_FILTER_IS_NOT_OF_HEADER_TYPE</td>
<td>65</td>
<td>The filter should be of header type</td>
</tr>
<tr>
<td>RXFL_WRONGMULTICAST_BROADCAST_ADDRESS</td>
<td>66</td>
<td>The address should be of type multicast or broadcast</td>
</tr>
<tr>
<td>RXFL_FIELD_SUPPORT ONLY_EQUAL AND NOTEQUAL</td>
<td>67</td>
<td>Rule compare function Id is out of range</td>
</tr>
<tr>
<td>RXFL_ACTION_USE_REG1_TO_REG4</td>
<td>68</td>
<td>Only counters 1 - 4 are allowed, for action</td>
</tr>
<tr>
<td>RXFL_ACTION_USE_REG5_TO_REG8</td>
<td>69</td>
<td>Only counters 5 - 8 are allowed, for action</td>
</tr>
<tr>
<td>RXFL_TRIGGER_USE_REG1_TO_REG4</td>
<td>70</td>
<td>Only counters 1 - 4 are allowed, for trigger</td>
</tr>
<tr>
<td>RXFL_TRIGGER_USE_REG5_TO_REG8</td>
<td>71</td>
<td>Only counters 5 - 8 are allowed, for Tigger</td>
</tr>
<tr>
<td>RXFL_SYSTEM_STATE_NOT_SUPPORTED_FOR_THIS_FILTER</td>
<td>72</td>
<td>System state is not supported</td>
</tr>
<tr>
<td>RXFL_DEPENDENCY_IS_NOT_PERSISTENT</td>
<td>74</td>
<td>Dependency filter is not persistent</td>
</tr>
<tr>
<td>RXFL_DEPENDENT_FILTERSOFTWARE_FILTER_NOT_FIT</td>
<td>75</td>
<td>Node filter can't be child of software filter and vice versa</td>
</tr>
<tr>
<td>RXFL_OUTPUT_OR_INPUT_BUFFER_LENGTH_TOO_SMALL</td>
<td>76</td>
<td>The output buffer length is smaller than required for that operation</td>
</tr>
</tbody>
</table>
Appendix C

How to Generate Certificates, Public Keys, and CAs

C.1 Certificate Generation

How to generate certificates, public keys, and CAs:

1. Download and install the latest package of OpenSSL (either Windows or Linux).
2. In the installation path `bin` library, find openssl.exe.

**Private Key**

To create a new private key for a certificate, use:

```bash
openssl genrsa -out privkey.pem 2048
```

**Notes:**

- The default key size is 2048, but the user can use any desired protocol key size (1024, 2048, 4096 and so forth).
- The name of the file is replaceable.
- The default format is PEM, which is in ASCII form. In many systems the binary format, DER, is more popular due to its smaller size. To convert between the formats use: `openssl rsa -in privkey.pem -inform PEM -out privkey.der -outform DER`

**Certificate and CA**

The CA (certificate authority) is a self-signed certificate used for signing other certificates.

To generate a CA, use the following command and insert the desired values:

```bash
openssl req -new -x509 -days 3650 -key privkey.pem -out root-ca.pem
```

Several notes about the example:

- The days argument determines how long the certificate is valid.
- The key is generated in the Private Key section of this document, in PEM format.
- The output is in PEM format. To convert from PEM to DER use: `openssl x509 -in input.crt -inform PEM -out output.crt -outform DER`

To generate a certificate, first prepare the certificate document. Similar to making a CA, fill the desired values such as country code name and so forth with the command:

```bash
openssl req -new -key privkey.pem -out cert.pem
```

The private key is different from the one used for the CA. Each certificate should have its own private key.

After generating a certificate form (also called certificate request), sign it with another certificate. The form is usually signed with the CA, but to make a chain, sign it with another certificate.

To do the signing process use:

```bash
openssl x509 -req -days 730 -in cert.pem -CA ca.pem -CAkey CApriate.pem -set_serial 01 -out cert.pem
```

Several notes about the example:

- Use the CA. Use whatever certificate you like to sign on the generated certificate.
- The key used here is the CA private key.
- The “days” argument used to determine how long will this certificate be valid for -set_serial 01 is needed as default.
In conclusion, if you want to generate a CA and then a certificate signed by the CA do the following:

- Generate Private Key for the CA.
- Generate Private Key for the certificate
- Make a CA with its private key
- Make a certificate request with its private key
- Sign the certificate with the CA and the CA private key
- If you want to make a chain, create another private key and certificate request and sign it with the other certificate

**How to generate sha1 and sign it with a private key**

```bash
openssl dgst -sha1 data.txt > hash
```

To make a sha1 code out of data.txt file, use:

```
To RSA sign this sha1 code with a private key, use:
openssl dgst -binary -out signature.bin -sha1 -sign privatekey.pem BufferToSign.bin
```
Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

<table>
<thead>
<tr>
<th>Changes from A Revision (March 2016) to B Revision</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Added Set Domain Name for Verification and SNI section</td>
<td>53</td>
</tr>
<tr>
<td>• Updated Sending and Receiving section</td>
<td>126</td>
</tr>
</tbody>
</table>
IMPORTANT NOTICE FOR TI DESIGN INFORMATION AND RESOURCES

Texas Instruments Incorporated ("TI") technical, application or other design advice, services or information, including, but not limited to, reference designs and materials relating to evaluation modules, (collectively, "TI Resources") are intended to assist designers who are developing applications that incorporate TI products; by downloading, accessing or using any particular TI Resource in any way, you (individually or, if you are acting on behalf of a company, your company) agree to use it solely for this purpose and subject to the terms of this Notice.

TI's provision of TI Resources does not expand or otherwise alter TI's applicable published warranties or warranty disclaimers for TI products, and no additional obligations or liabilities arise from TI providing such TI Resources. TI reserves the right to make corrections, enhancements, improvements and other changes to its TI Resources.

You understand and agree that you remain responsible for using your independent analysis, evaluation and judgment in designing your applications and that you have full and exclusive responsibility to assure the safety of your applications and compliance of your applications (and of all TI products used in or for your applications) with all applicable regulations, laws and other applicable requirements. You represent that, with respect to your applications, you have all the necessary expertise to create and implement safeguards that (1) anticipate dangerous consequences of failures, (2) monitor failures and their consequences, and (3) lessen the likelihood of failures that might cause harm and take appropriate actions. You agree that prior to using or distributing any applications that include TI products, you will thoroughly test such applications and the functionality of such TI products as used in such applications. TI has not conducted any testing other than that specifically described in the published documentation for a particular TI Resource.

You are authorized to use, copy and modify any individual TI Resource only in connection with the development of applications that include the TI product(s) identified in such TI Resource. NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY OTHER TI INTELLECTUAL PROPERTY RIGHT, AND NO LICENSE TO ANY TECHNOLOGY OR INTELLECTUAL PROPERTY RIGHT OF TI OR ANY THIRD PARTY IS GRANTED HEREIN, including but not limited to any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information regarding or referencing third-party products or services does not constitute a license to use such products or services, or a warranty or endorsement thereof. Use of TI Resources may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

TI RESOURCES ARE PROVIDED "AS IS" AND WITH ALL FAULTS. TI DISCLAIMS ALL OTHER WARRANTIES OR REPRESENTATIONS, EXPRESS OR IMPLIED, REGARDING TI RESOURCES OR USE THEREOF, INCLUDING BUT NOT LIMITED TO ACCURACY OR COMPLETENESS, TITLE, ANY EPIDEMIC FAILURE WARRANTY AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

TI SHALL NOT BE LIABLE FOR AND SHALL NOT DEFEND OR INDEMNIFY YOU AGAINST ANY CLAIM, INCLUDING BUT NOT LIMITED TO ANY INFRINGEMENT CLAIM THAT RELATES TO OR IS BASED ON ANY COMBINATION OF PRODUCTS EVEN IF DESCRIBED IN TI RESOURCES OR OTHERWISE. IN NO EVENT SHALL TI BE LIABLE FOR ANY ACTUAL, DIRECT, SPECIAL, COLLATERAL, INDIRECT, PUNITIVE, INCIDENTAL, CONSEQUENTIAL OR EXEMPLARY DAMAGES IN CONNECTION WITH OR ARISING OUT OF TI RESOURCES OR USE THEREOF, AND REGARDLESS OF WHETHER TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You agree to fully indemnify TI and its representatives against any damages, costs, losses, and/or liabilities arising out of your non-compliance with the terms and provisions of this Notice.

This Notice applies to TI Resources. Additional terms apply to the use and purchase of certain types of materials, TI products and services. These include, without limitation, TI's standard terms for semiconductor products (http://www.ti.com/sc/docs/stdterms.htm), evaluation modules, and samples (http://www.ti.com/sc/docs/sampterms.htm).

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2018, Texas Instruments Incorporated