WL1271 Code Descriptions and Examples

Literature Number: SPRUGT5
January 2010

Contents

Revision History

Version	Date	Description
1.0	January 2010	Release

Reference Documents

The documents listed below provide complementary specifications and information for the device:

- *Bluetooth Patch Description*

About This Document

This document provides code examples for several basic WLAN scenarios.

This document contains the following chapters:

- **Chapter 1, Introduction,** page 4, provides a general description of the purpose and organization of this document.
- **Chapter 2, General,** page 6**,** describes general commands and their usage.
- **Chapter 3, Active Scan,** page 10, describes the two methods for performing scanning.
- **Chapter 4, Connect,** page 16, describes how to connect to various types of networks.
- **Chapter 5, Disconnect,** page 28, describes how to disconnect from the network.
- **Appendix A, Structure Type Definitions,** page 30, describes structure definitions for various parameters.

Introduction

Topic **Page**

This document includes code examples of the following basic WLAN scenarios:

- Active scan
- Connection to an open network, WEP-secured network, WPA-secured network and WPA2-secured network
- Disconnection from a network

This document does not provide a full set of commands and examples, but rather the basic set needed for initial use.

The interface level used in this document is described in a dedicated document. The interface is not a direct connection to the driver. The main reason for that is the high level of specialty required by the programmer as a result of complexity (few API interfaces with many argument types).

1.1 Purpose

The purpose of this document is to provide the customer with several code examples for a basic set of operations that may be used to test the TI WLAN chip. The customer may develop a utility that interacts with the driver indirectly by using the interface described herein.

1.2 File Name

The file name of this document is WL1271 Code Descriptions and Examples.doc.

General

Topic **Page**

Before using the commands described in this document, some parameters must be initialized. The structure type containing these parameters is *CuCmd_t. CuCmd_t*, which is defined in *Section A.2, CU_CLI Control Parameter*.

This structure includes the following important parameters:

- **hCuCommon:** Handle to communicate commands with the driver
- **hConsole:** Console handle created when allocating memory for the console
- **hIpcEvent:** Handle to communicate events with the driver
- **hWpaCore:** Handle to communicate with the WPA supplicant
- **appScanParams:** Scan parameters. You may refer to *Section A.3, Scan Parameters*, for more information.

2.1 Initialization Command

Example 1: Initialization command

Description: In this example, all necessary handles and default scan parameters are initialized. This command is mandatory and must be applied before any other command can be applied.

The arguments to this function include:

- **device_name:** The name of the driver, such as **tiwlan0**
- **hConsole:** The console handle
- **BypassSupplicant:** A flag denoting whether or not a supplicant is used
- **PSupplIfFile:** The full path (directory and file name) of the supplicant control interface (the interface created for communication between the user utility; for example, CU_CLI and the WPA supplicant)

Code Segment:

```
{
    S8 device_name[IF_NAME_SIZE];
    Console_t* pConsole;
    S32 BypassSupplicant = FALSE;
    S8 pSupplIfFile[50];

    // arguments initialization
    os_strcpy(device_name, (PS8)”tiwlan0”);
    os_strcpy(pSupplIfFile, (PS8) "/var/run/tiwlan0");
    pConsole = (Console_t*)os_MemoryCAlloc(sizeof(Console_t), sizeof(U8));

    // executing command
    pConsole->hCuCmd = CuCmd_Create(device_name, pConsole, BypassSupplicant,
pSupplIfFile);

    //returned value should not be a NULL pointer
}
```

Explanation:

Initialization parameters are configured as follows:

- The driver name: tiwlan0
- Allocate memory on kernel
- Don't bypass the supplicant: FALSE
- The full path of the supplicant's control interface: /var/run/tiwlan0

The following are obtained (the relevant structure is defined in *Section A.2, CU_CLI Control Parameter*):

- Handle to communicate commands with the driver: *pCuCmd->hCuCommon*
- Handle to communicate events with the driver: *pCuCmd->hIpcEvent*
- Handle to communicate with the WPA supplicant: *pCuCmd->hWpaCore*
- Default scan parameters, as described in *Section 3.2, Initializing Scanning*

2.2 General Command Structure

Each command has a constant structure, as follows:

```
VOID CuCmd_<command name>(THandle hCuCmd, ConParm_t parm[], U16 nParms);
```

hCuCmd argument is the initialized handle.

Parm and *nParms* are optional.

There are some commands that use default values or that do not use any parameter. The relevant parameters (if needed) are described, per command, in the following section:

- *ConParm_t* structure definition can be found in *Section A.1, CU_CLI Command Parameter*.

This page was intentionally left blank.

Active Scan

Topic **Page**

3.1 Introduction

There are two ways to start an application scan. The difference between the two approaches is transparent to the user. In one way, it is applied using a private command and in the second way, it is applied using a WEXT command. The different sequences are executed in the background so that the user is unaware of them.

3.2 Initializing Scanning

The scan parameters initialization is done as part of the general initialization described in the previous section.

Example 2: Initializing scan parameters

Description: In this example, all necessary scan parameters are initialized. Note that the values appearing in this example are the default ones.

Code Segment:

```
{
    CuCmd_t* pCuCmd;

    // arguments initialization
    CuCmd_t* pCuCmd = (CuCmd_t*)os_MemoryCAlloc(sizeof(CuCmd_t), sizeof(U8));

    // executing command
    CuCmd_Init_Scan_Params(pCuCmd);
}
```

Explanation:

Initialization parameters are configured, as follows:

- *pCuCmd.appScanParams.desiredSsid.len* = 0
- *pCuCmd.appScanParams.scanType* = SCAN_TYPE_NORMAL_ACTIVE
- *pCuCmd.appScanParams.band* = RADIO_BAND_2_4_GHZ
- *pCuCmd.appScanParams.probeReqNumber* = 3 (denoting the number of probe requests to send to each channel)
- *pCuCmd.appScanParams.probeRequestRate* = RATE_MASK_UNSPECIFIED
- *pCuCmd.numOfChannels* = 14

3.3 Application Scan Using Private Command

Example 3: Application scan

Description: In this example, an application scan is initiated and scan results are obtained. When using a private command, scanning is applied over all SSIDs. Taking into account the default scan parameters (see *Section 3.2, Initializing Scanning*), the scan is configured over 14 channels on a 2.4GHz band with three probe requests per channel.

Code Segment:

```
{
    ConParm_t parm;
    U16 nParm;
    // pConsole->hCuCmd  is used as returned from CuCmd_Create()

    // arguments initialization
    parm.flags = CON_PARM_STRING | CON_PARM_OPTIONAL;
    parm.low_val = 0;
    parm.hi_val = 32;
    nParm = 0;

    // executing command
    CuCmd_StartScan(pConsole->hCuCmd, &parm, nParm);

    // get results
    CuCmd_BssidList(pConsole->hCuCmd, 0, 0);
}
```

Expected Output / Results:

```
Application scan started.
BssId List: Num=8
         MAC        Privacy Rssi  Mode    Channel    SSID
 00.50.f1.12.03.38     0    -45  Infra      1        yaelb
 00.12.01.4d.da.70     0    -45  Infra      1        Cat
 00.16.46.b8.bf.e0     0    -58  Infra      1        123
 00.19.a9.fc.f9.20     0    -52  Infra      6        Cat
 00.16.46.c6.2b.90     1    -54  Infra      6        ****
 00.0f.f7.0c.e8.e0     1    -45  Infra     10        shlomi_net
 00.15.c6.5f.62.50     0    -45  Infra     11        Rachel_11g
 00.14.bf.3c.4c.4d     0    -59  Infra     11        linksys
```

In this example, there is no connection to an AP. If an AP was connected (such as Cat), the output is as follows:

```
BssId List: Num=8
         MAC        Privacy Rssi  Mode    Channel    SSID
 00.50.f1.12.03.38     0    -45  Infra      1        yaelb
*00.12.01.4d.da.70     0    -45  Infra      1        Cat
 00.16.46.b8.bf.e0     0    -58  Infra      1        123
 00.19.a9.fc.f9.20     0    -52  Infra      6        Cat
 00.16.46.c6.2b.90     1    -54  Infra      6        ****
 00.0f.f7.0c.e8.e0     1    -45  Infra     10        shlomi_net
 00.15.c6.5f.62.50     0    -45  Infra     11        Rachel_11g
 00.14.bf.3c.4c.4d     0    -59  Infra     11        linksys
```

Note: The MAC address of the AP appears on the printout with an asterisk on the far left.

Explanation:

- Initially, the driver is created and a handle to it is obtained (*pConsole->hCuCmd*).
- Scan parameters are configured, as follows:
  - No parameters are defined, meaning that all SSID are scanned (it is not possible to scan a specific SSID).
- Scan results are obtained and printed to the screen.

3.4 Application Scan Using the WEXT Command

Example 4: Application scan

Description: In this example, an application scan is initiated and scan results are obtained. When using a private command, scanning is applied over all SSIDs. Taking into account the default scan parameters (see *Section 3.2, Initializing Scanning*), the scan is configured over 14 channels on a 2.4GHz band with three probe requests per channel.

Code Segment:

```
{
    ConParm_t parm;
    U16 nParm;
    S8 ssid[32];
    // pConsole->hCuCmd  is used as returned from CuCmd_Create()

    // arguments initialization
    parm.flags = CON_PARM_STRING | CON_PARM_OPTIONAL;
    parm.low_val = 0;
    parm.hi_val = 32;
    os_strcpy(ssid, (PS8)”Cat”);
    parm.value = (PS8)ssid;
    nParm = 1;

    // executing command
    CuCmd_WextStartScan(pConsole->hCuCmd, &parm, nParm);
```

```
    // get results
    CuCmd_BssidList(pConsole->hCuCmd, 0, 0);
}
```

Expected Output / Results:

```
application scan started.
BssId List: Num=1
         MAC        Privacy Rssi  Mode    Channel    SSID
00.19.a9.fc.f9.20     0    -52  Infra      6        Cat
```

Explanation:

- Initially, the driver is created and a handle to it is obtained (*pConsole->hCuCmd*).
- Scan parameters are configured, as follows:
  - SSID **Cat** is defined, meaning that only this SSID is scanned and reported.
- Scan results are obtained and printed to the screen.

3.5 Stop Scanning

Example 5: Stop scanning

Description: In this example, an application scan is stopped.

Code Segment:

```
{
    ConParm_t parm;
    U16 nParm;
    // pConsole->hCuCmd  is used as returned from CuCmd_Create()

    // arguments initialization
    parm.flags = 0;
    parm.low_val = 0;
    parm.hi_val = 0;
    nParm = 0;

    // executing command
    CuCmd_StopScan(pConsole->hCuCmd, &parm, nParm);
}
```

Expected Output / Results:

The application scan stops.

Explanation:

- Initially, the driver is created and a handle to it is obtained (*pConsole->hCuCmd*).
- No parameters are defined. Only the scan is stopped.
- A Stop message is printed on the screen.

This page was intentionally left blank.

Connect

Topic **Page**

4.1 Connecting to an Open Network

Connecting to an open network comprises three operations:

- Getting the BSS type
- Setting the BSSID
- Setting the SSID

Example 6: Connecting to an open network

Description: In this example, a connection to an open network is made. The SSID configured for connection is **Cat** and no BSSID is specified. The BSSID option is used when there are several APs with the same SSID. It is possible to distinguish between them according to the BSSID.

Code Segment:

```
{
    ConParm_t parm;
    U16 nParm;
    S8 ssid[32];
    // pConsole->hCuCmd  is used as returned from CuCmd_Create()

    // arguments initialization
    parm.flags = CON_PARM_STRING | CON_PARM_OPTIONAL;
    parm.low_val = 0;
    parm.hi_val = 32;
    os_strcpy(ssid, (PS8)"Cat");
    parm.value = (PS8)ssid;
    nParm = 1;

    // executing command
    CuCmd_Connect(pConsole->hCuCmd, &parm, nParm);

}
```

Note: If the BSS type is IBSS (*ad hoc*), then the SSID is mandatory.

Expected Output / Results:

The CLI printout when connecting to the SSID (for example, Cat) is:

```
************ NEW CONNECTION ************
-- SSID  = Cat
-- BSSID = 0-12-1-4d-da-70
*************************************
```

Explanation:

- Initially, the driver is created and a handle to it is obtained (*pConsole->hCuCmd*).
- Scan parameters are configured, as follows:
  - SSID **Cat** is to be connected.
- A NEW CONNECTION message is printed on the screen.

4.2 Before Connecting to Any Secured Network

Before connecting to any secured network, the WPA supplicant must be initialized and started.

The WPA supplicant communicates with both the driver and a utility (for example, CU_CLI, wpa_cli, wpa_gui and so on) responsible for configuring and managing the driver. The customer can create its own utility to control the driver. TI uses a utility called Configuration Utility (CU_CLI).

The communication between the utility and the WPA supplicant is performed via a control interface. For Linux, the control interface is created by a UNIX socket between two user space processes. It is very much like using a virtual file system (the UNIX /proc system is implemented in a similar way).

- **How does the WPA supplicant know what is the control interface?**
  The name of this virtual file is supplied as one of the arguments when running the supplicant (under -i option). The path to this file is supplied in a configuration file as the **ctrl_interface** parameter. The name of the configuration file is supplied as one of the arguments when running the supplicant (under -c option).
- **How does the user utility know what is the control interface?**
  The name of this virtual file is supplied either as one of its arguments for example, under the -i option for wpa_cli) or hard coded as the driver name for CU_CLI (tiwlan0). The path to this file is supplied either as one of its arguments (for example, under the -p option for wpa_cli) or hard coded for CU_CLI (as /var/run).

After the WPA supplicant is running in the background, the WPA core in the utility can be initialized. The WPA initialization is applied as part of the general initialization process (see *Section 2.1, Initialization Command*). As part of the initialization process, the following operations are generated:

- Create a socket to communicate with the supplicant (through the virtual file system)
- Initialize the WPA supplicant parameters
- Send the configuration to the supplicant

Note that the WPA supplicant parameters are set according to the structures described in *Section A.4, Security Parameters*.

The default parameters set are:

- WpaSupplParams:
  - WpaSupplParams.mode = IEEE80211_MODE_INFRA
  - WpaSupplParams.proto = WPA_PROTO_WPA
  - WpaSupplParams.key_mgmt = WPA_KEY_MGMT_NONE
  - WpaSupplParams.auth_alg = WPA_AUTH_ALG_OPEN
  - WpaSupplParams.pair_wise = WPA_CIPHER_NONE
  - WpaSupplParams.group = WPA_CIPHER_NONE
  - WpaSupplParams.anyWpaMode = 0
- WpaParams:
  - WpaParams.AuthMode = os802_11AuthModeOpen
  - WpaParams.EncryptionTypeGroup = OS_ENCRYPTION_TYPE_NONE
  - WpaParams.EncryptionTypePairWise = OS_ENCRYPTION_TYPE_NONE

4.3 Connecting to a WEP-secured Network

Example 7: Connecting to a WEP secured network

Description: In this example, a connection to a WEP secured network is conducted. First, the WEP parameters are defined. These parameters include an authentication algorithm as OPEN with no key management (os802_11AuthModeOpen), an encryption type of WEP (OS_ENCRYPTION_TYPE_WEP) and the key itself, 1234567890. The SSID configured for connection is Cat and no BSSID is specified.

Code Segment:

```
{
    // pConsole->hCuCmd is used as returned from CuCmd_Create()

    // authentication algorithm parameters
    ConParm_t parm1;
    U16 nParm1;

    // encryption type parameters
    ConParm_t parm2;
    U16 nParm2;

    // key value parameters
    ConParm_t parm3[3];
    U16 nParm3;
    S8 key[64];

    // SSID parameters
    ConParm_t parm4;
    U16 nParm4;
    S8 ssid[32];



    // arguments initialization

    // authentication algorithm parameters
    parm1.flags = CON_PARM_OPTIONAL;
    parm1.value = os802_11AuthModeOpen;
    nParm1 = 1;

    // encryption type parameters
    parm2.flags = CON_PARM_OPTIONAL;
    parm2.value = OS_ENCRYPTION_TYPE_WEP;
    nParm2 = 1;

    // key phrase parameter
    parm3[0].flags = CON_PARM_STRING;
    parm3[0].low_val = 0;
    parm3[0].hi_val = 64;
    os_strcpy(key, (PS8)”1234567890”);
```

```
    parm3[0].value = (PS8)key;
    // TX key index parameter
    parm3[1].value = 0;
    // default key parameter
    parm3[2].value = 1;
    nParm3 = 3;

    // SSID parameter
    parm4.flags = CON_PARM_STRING | CON_PARM_OPTIONAL;
    parm4.low_val = 0;
    parm4.hi_val = 32;
    os_strcpy(ssid, (PS8)"Cat");
    parm4.value = (PS8)ssid;
    nParm4 = 1;

    // executing authentication command
    CuCmd_SetPrivacyAuth(pConsole->hCuCmd, &parm1, nParm1);

    // executing encryption command
    CuCmd_SetPrivacyEncryption(pConsole->hCuCmd, &parm2, nParm2);

    // executing key command
    CuCmd_AddPrivacyKey(pConsole->hCuCmd, &parm3, nParm3);

    // executing connction command
    CuCmd_Connect(pConsole->hCuCmd, &parm4, nParm4);

}
```

Expected Output / Results:

```
The CLI print when connecting to SSID (for example, Cat) is:
************ NEW CONNECTION ************
-- SSID  = Cat
-- BSSID = 0-f-f7-c-e8-e0
*************************************
Associated with 00:0f:f7:0c:e8:e0
CTRL-EVENT-CONNECTED - Connection to 00:0f:f7:0c:e8:e0 completed (auth) [id=3 id_str=]
```

Explanation:

- Initially, the driver is created and a handle to it is obtained (*pConsole->hCuCmd*).
- WEP parameters are configured, as follows:
  - Authentication algorithm = OPEN
  - Encryption type = WEP
  - Key phrase = 1234567890
- The SSID used for the connection is Cat.
- A NEW CONNECTION message is printed on the screen.

The relevant parameters are set according to the structures described in *Section A.4, Security Parameters*.

- WpaSupplParams:
  - WpaSupplParams.mode = IEEE80211_MODE_INFRA
  - WpaSupplParams.proto = 0
  - WpaSupplParams.key_mgmt = WPA_KEY_MGMT_NONE
  - WpaSupplParams.auth_alg = WPA_AUTH_ALG_OPEN
  - WpaSupplParams.pair_wise = WPA_CIPHER_WEP40
  - WpaSupplParams.group = WPA_CIPHER_WEP40
- WpaParams:
  - WpaParams.AuthMode = os802_11AuthModeOpen
  - WpaParams.EncryptionTypeGroup = WPA_CIPHER_WEP40
  - WpaParams.EncryptionTypePairWise = WPA_CIPHER_WEP40

4.4 Connecting to a WPA-secured Network

Example 8: Connecting to a WPA secured network

Description: In this example, a connection to a WPA secured network is made. First, the WPA parameters are defined. These parameters include an authentication algorithm as open with a WPA protocol and with a PSK key management (os802_11AuthModeWPA), an encryption type of TKIP (OS_ENCRYPTION_TYPE_TKIP) and the passphrase itself, 1234567890. Second, the SSID configured for connection is Cat and no BSSID is specified.

Code Segment:

```
{
    // pConsole->hCuCmd is used as returned from CuCmd_Create()

    // authentication algorithm parameters
    ConParm_t parm1;
    U16 nParm1;

    // encryption type parameters
    ConParm_t parm2;
    U16 nParm2;

    // passphrase parameters
    ConParm_t parm3;
    U16 nParm3;
    S8 passphrase[64];

    // SSID parameters
    ConParm_t parm4;
    U16 nParm4;
    S8 ssid[32];
```

```
    // arguments initialization

    // authentication algorithm parameters
    parm1.flags = CON_PARM_OPTIONAL;
    parm1.value = os802_11AuthModeWPA;
    nParm1 = 1;

    // encryption type parameters
    parm2.flags = CON_PARM_OPTIONAL;
    parm2.value = OS_ENCRYPTION_TYPE_TKIP;
    nParm2 = 1;

    // key phrase parameter
    parm3.flags = CON_PARM_STRING;
    parm3.low_val = WPACORE_MIN_PSK_STRING_LENGTH /* equals 8 */;
    parm3.hi_val = WPACORE_MAX_PSK_STRING_LENGTH /* equals 64 */;
    os_strcpy(passphrase, (PS8)"1234567890");
    parm3.value = (PS8)passphrase;
    nParm3 = 1;

    // SSID parameter
    parm4.flags = CON_PARM_STRING | CON_PARM_OPTIONAL;
    parm4.low_val = 0;
    parm4.hi_val = 32;
    os_strcpy(ssid, (PS8)"Cat");
    parm4.value = (PS8)ssid;
    nParm4 = 1;

    // executing authentication command
    CuCmd_SetPrivacyAuth(pConsole->hCuCmd, &parm1, nParm1);

    // executing encryption command
    CuCmd_SetPrivacyEncryption(pConsole->hCuCmd, &parm2, nParm2);

    // executing passphrase command
    CuCmd_SetPrivacyPskPassPhrase(pConsole->hCuCmd, &parm3, nParm3);

    // executing connection command
    CuCmd_Connect(pConsole->hCuCmd, &parm4, nParm4);

}
```

Expected Output / Results:

```
Associated with 00:0f:f7:0c:e8:e0
************ NEW CONNECTION ************
-- SSID  = Cat
-- BSSID = 0-f-f7-c-e8-e0
**************************************
WPA: Key negotiation completed with 00:0f:f7:0c:e8:e0 [PTK=TKIP GTK=TKIP]
CTRL-EVENT-CONNECTED - Connection to 00:0f:f7:0c:e8:e0 completed (reauth) [id=3 id_str=]
```

Explanation:

- Initially, the driver is created and a handle to it is obtained (*pConsole->hCuCmd*).
- WPA parameters are configured, as follows:
  - Authentication algorithm = OPEN
  - Protocol = WPA
  - Key management = PSK
  - Encryption type = TKIP
  - Passphrase = 1234567890
- The SSID used for the connection is Cat.
- A NEW CONNECTION message is printed on the screen.

The relevant parameters are set according to the structures described in *Section A.4, Security Parameters*.

- WpaSupplParams :
  - WpaSupplParams.mode = IEEE80211_MODE_INFRA
  - WpaSupplParams.proto = WPA_PROTO_WPA
  - WpaSupplParams.key_mgmt = WPA_KEY_MGMT_PSK
  - WpaSupplParams.auth_alg = WPA_AUTH_ALG_OPEN
  - WpaSupplParams.anyWpaMode = 0
  - WpaSupplParams.pair_wise = WPA_CIPHER_TKIP
  - WpaSupplParams.group = WPA_CIPHER_TKIP
  - WpaSupplParams.pass_phrase = “1234567890” (example)
- WpaParams:
  - WpaParams.AuthMode = os802_11AuthModeOpen
  - WpaParams.EncryptionTypeGroup = WPA_CIPHER_TKIP
  - WpaParams.EncryptionTypePairWise = WPA_CIPHER_TKIP

4.5 Connecting to a WPA2-secured Network

Example 9: Connecting to a WPA2 secured network

Description: In this example, a connection to a WPA2 secured network is made. First, the WPA2 parameters are defined. These parameters include an authentication algorithm as OPEN with an RSN protocol and with a PSK key management (os802_11AuthModeWPA2PSK), an encryption type of CCMP (OS_ENCRYPTION_TYPE_AES) and the passphrase itself, 1234567890. Second, the SSID configured for connection is Cat and no BSSID is specified.

Code Segment:

```
{
    // pConsole->hCuCmd is used as returned from CuCmd_Create()

    // authentication algorithm parameters
    ConParm_t parm1;
    U16 nParm1;

    // encryption type parameters
    ConParm_t parm2;
    U16 nParm2;

    // passphrase parameters
    ConParm_t parm3;
    U16 nParm3;
    S8 passphrase[64];

    // SSID parameters
    ConParm_t parm4;
    U16 nParm4;
    S8 ssid[32];


    // arguments initialization

    // authentication algorithm parameters
    parm1.flags = CON_PARM_OPTIONAL;
    parm1.value = os802_11AuthModeWPA2PSK;
    nParm1 = 1;

    // encryption type parameters
    parm2.flags = CON_PARM_OPTIONAL;
    parm2.value = OS_ENCRYPTION_TYPE_AES;
    nParm2 = 1;

    // key phrase parameter
    parm3.flags = CON_PARM_STRING;
    parm3.low_val = WPACORE_MIN_PSK_STRING_LENGTH /* equals 8 */;
    parm3.hi_val = WPACORE_MAX_PSK_STRING_LENGTH /* equals 64 */;
    os_strcpy(passphrase, (PS8)”1234567890”);
```

```
    parm3.value = (PS8)passphrase;
    nParm3 = 1;

    // SSID parameter
    parm4.flags = CON_PARM_STRING | CON_PARM_OPTIONAL;
    parm4.low_val = 0;
    parm4.hi_val = 32;
    os_strcpy(ssid, (PS8)"Cat");
    parm4.value = (PS8)ssid;
    nParm4 = 1;

    // executing authentication command
    CuCmd_SetPrivacyAuth(pConsole->hCuCmd, &parm1, nParm1);

    // executing encryption command
    CuCmd_SetPrivacyEncryption(pConsole->hCuCmd, &parm2, nParm2);

    // executing passphrase command
    CuCmd_SetPrivacyPskPassPhrase(pConsole->hCuCmd, &parm3, nParm3);

    // executing connection command
    CuCmd_Connect(pConsole->hCuCmd, &parm4, nParm4);

}
```

Expected Output / Results:

```
Associated with 00:0f:f7:0c:e8:e0
************ NEW CONNECTION ************
-- SSID  = Cat
-- BSSID = 0-f-f7-c-e8-e0
**************************************
WPA: Key negotiation completed with 00:0f:f7:0c:e8:e0 [PTK=CCMP GTK=CCMP]
CTRL-EVENT-CONNECTED - Connection to 00:0f:f7:0c:e8:e0 completed (auth) [id=0 id_str=]
```

Explanation:

- Initially, the driver is created and a handle to it is obtained (*pConsole->hCuCmd*).
- WPA2 parameters are configured, as follows:
  - Authentication algorithm = OPEN
  - Protocol = RSN
  - Key management = PSK
  - Encryption type = CCMP
  - Passphrase = 1234567890
- The SSID used for the connection is Cat.
- A NEW CONNECTION message is printed on the screen.

The relevant parameters are set according to the structures described in *Section A.4, Security Parameters*.

- WpaSupplParams:
  - WpaSupplParams.mode = IEEE80211_MODE_INFRA
  - WpaSupplParams.proto = WPA_PROTO_RSN
  - WpaSupplParams.key_mgmt = WPA_KEY_MGMT_PSK
  - WpaSupplParams.auth_alg = WPA_AUTH_ALG_OPEN
  - WpaSupplParams.anyWpaMode = 0
  - WpaSupplParams.pair_wise = WPA_CIPHER_CCMP
  - WpaSupplParams.group = WPA_CIPHER_CCMP
  - WpaSupplParams.pass_phrase = “1234567890” (example)
- WpaParams:
  - WpaParams.AuthMode = os802_11AuthModeOpen
  - WpaParams.EncryptionTypeGroup = WPA_CIPHER_CCMP

WpaParams.EncryptionTypePairWise = WPA_CIPHER_CCMP

This page was intentionally left blank.

Disconnect

Example 10: Disconnecting from a network

Description: In this example, a disconnection from a network is performed. No parameters are required.

Code Segment:

```
{
    ConParm_t parm;
    U16 nParm;
    // pConsole->hCuCmd is used as returned from CuCmd_Create()

    // arguments initialization
    parm.flags = 0;
    parm.low_val = 0;
    parm.hi_val = 0;
    nParm = 0;

    // executing command
    CuCmd_Disassociate(pConsole->hCuCmd, &parm, nParm);
}
```

Explanation:

- Initially, the driver is created and a handle to it is obtained (*pConsole->hCuCmd*).
- No parameters are required.

This page was intentionally left blank.

Structure Type Definitions

Topic **Page**

A.1 Co CU_CLI Command Parameter

```
typedef struct ConParm_t
{
   PS8         name;      /* Parameter name */
   U8          flags;     /* Combination of CON_PARM_ flags */
   U32         low_val;   /* Low val for range checking */
   U32         hi_val;    /* Hi val for range checking/max length of string */
   U32         value;     /* Value/address of string parameter */
} ConParm_t;
```

```
#define CON_PARM_OPTIONAL       0x01  /* Parameter is optional */
#define CON_PARM_DEFVAL         0x02  /* Default value is set */
#define CON_PARM_RANGE          0x04  /* Range is set */
#define CON_PARM_STRING         0x08  /* String parm */
#define CON_PARM_LINE           0x10  /* String from the current parser position till EOL */
#define CON_PARM_SIGN           0x20  /* Signed param */
#define CON_PARM_NOVAL          0x80  /* Internal flag: parameter is anassigned */
#define CON_LAST_PARM           { NULL, 0, 0, 0, 0 }
```

A.2 CU_CLI Control Parameter

```
typedef struct CuCmd_t
{
    THandle                 hCuWext;
    THandle                 hCuCommon;
    THandle                 hConsole;
    THandle                 hIpcEvent;
    THandle                 hWpaCore;

    U32                     isDeviceRunning;

    scan_Params_t             appScanParams;
    TPeriodicScanParams     tPeriodicAppScanParams;
    scan_Policy_t             scanPolicy;

} CuCmd_t;
```

```
typedef struct Console_t
{
    THandle hCuCmd;

    S32 isDeviceOpen;

    ConEntry_t *p_mon_root;
    ConEntry_t *p_cur_dir;
    PS8         p_inbuf;
    volatile S32 stop_UI_Monitor;
} Console_t;
```

A.3 Scan Parameters

```
typedef TScanParams                         scan_Params_t;

typedef struct
{
    TSsid                   desiredSsid;
    EScanType               scanType;
    ERadioBand              band;
    TI_UINT8                probeReqNumber;
    ERateMask               probeRequestRate;
    TI_UINT8                Tid;
    TI_UINT64               latestTSFValue;
    TI_UINT32               SPSScanDuration;
    TI_UINT8                numOfChannels;
    TScanChannelEntry       channelEntry[MAX_NUMBER_OF_CHANNELS_PER_SCAN]; } TScanParams;
```

```
typedef struct
{
    TI_UINT8    len;                                /**< SSID Length              */

    char        str[ MAX_SSID_LEN ];  /**< SSID string buffer       */

}  TSsid;
```

```
typedef enum
{
/*  0   */      SCAN_TYPE_NORMAL_PASSIVE = 0,
/*  1   */      SCAN_TYPE_NORMAL_ACTIVE,
/*  2   */      SCAN_TYPE_SPS,
/*  3   */      SCAN_TYPE_TRIGGERED_PASSIVE,
/*  4   */      SCAN_TYPE_TRIGGERED_ACTIVE,
/*  5   */      SCAN_TYPE_NO_SCAN,
/*  6   */      SCAN_TYPE_PACTSIVE
} EScanType;
```

```
typedef enum
{
    RADIO_BAND_2_4_GHZ                  = 0,
    RADIO_BAND_5_0_GHZ                  = 1,
    RADIO_BAND_DUAL                     = 2,
    RADIO_BAND_NUM_OF_BANDS             = 2

} ERadioBand;
```

```
typedef enum
{
    DRV_RATE_MASK_AUTO          = DRV_RATE_AUTO,
    DRV_RATE_MASK_1_BARKER      = RATE_TO_MASK(DRV_RATE_1M),
    DRV_RATE_MASK_2_BARKER      = RATE_TO_MASK(DRV_RATE_2M),
```

```
    DRV_RATE_MASK_5_5_CCK       = RATE_TO_MASK(DRV_RATE_5_5M),
    DRV_RATE_MASK_11_CCK        = RATE_TO_MASK(DRV_RATE_11M),
    DRV_RATE_MASK_22_PBCC       = RATE_TO_MASK(DRV_RATE_22M),
    DRV_RATE_MASK_6_OFDM        = RATE_TO_MASK(DRV_RATE_6M),
    DRV_RATE_MASK_9_OFDM        = RATE_TO_MASK(DRV_RATE_9M),
    DRV_RATE_MASK_12_OFDM       = RATE_TO_MASK(DRV_RATE_12M),
    DRV_RATE_MASK_18_OFDM       = RATE_TO_MASK(DRV_RATE_18M),
    DRV_RATE_MASK_24_OFDM       = RATE_TO_MASK(DRV_RATE_24M),
    DRV_RATE_MASK_36_OFDM       = RATE_TO_MASK(DRV_RATE_36M),
    DRV_RATE_MASK_48_OFDM       = RATE_TO_MASK(DRV_RATE_48M),
    DRV_RATE_MASK_54_OFDM       = RATE_TO_MASK(DRV_RATE_54M),
    DRV_RATE_MASK_MCS_0_OFDM    = RATE_TO_MASK(DRV_RATE_MCS_0),
    DRV_RATE_MASK_MCS_1_OFDM    = RATE_TO_MASK(DRV_RATE_MCS_1),
    DRV_RATE_MASK_MCS_2_OFDM    = RATE_TO_MASK(DRV_RATE_MCS_2),
    DRV_RATE_MASK_MCS_3_OFDM    = RATE_TO_MASK(DRV_RATE_MCS_3),
    DRV_RATE_MASK_MCS_4_OFDM    = RATE_TO_MASK(DRV_RATE_MCS_4),
    DRV_RATE_MASK_MCS_5_OFDM    = RATE_TO_MASK(DRV_RATE_MCS_5),
    DRV_RATE_MASK_MCS_6_OFDM    = RATE_TO_MASK(DRV_RATE_MCS_6),
    DRV_RATE_MASK_MCS_7_OFDM    = RATE_TO_MASK(DRV_RATE_MCS_7)
} ERateMask;
```

```
typedef union
{
    TScanNormalChannelEntry   normalChannelEntry;
    TScanSpsChannelEntry      SPSChannelEntry;
} TScanChannelEntry;
```

```
typedef struct
{
    TMacAddr               bssId;
    TI_UINT32              maxChannelDwellTime;
    TI_UINT32              minChannelDwellTime;
    EScanEtCondition       earlyTerminationEvent;
    TI_UINT8               ETMaxNumOfAPframes;
    TI_UINT8               txPowerDbm;
    TI_UINT8               channel;
} TScanNormalChannelEntry;
```

```
typedef struct
{
    TMacAddr               bssId;
    TI_UINT32              scanDuration;
    TI_UINT32              scanStartTime;
    EScanEtCondition       earlyTerminationEvent;
    TI_UINT8               ETMaxNumOfAPframes;
    TI_UINT8               channel;
} TScanSpsChannelEntry;
```

A.4 Security Parameters

```
typedef struct TWpaCore
{
    THandle hIpcWpa;

    S32 CurrentNetwork;

    TWpaCore_WpaSupplParams WpaSupplParams;
    TWpaCore_WpaParams WpaParams;
} TWpaCore;
```

```
typedef struct
{
    S32 mode;
    S32 proto;
    S32 key_mgmt;
    S32 auth_alg;
    S32 pair_wise;
    S32 group;
    U8 pass_phrase[WPACORE_MAX_PSK_STRING_LENGTH];
    U8 wep_key[4][32];
    U8 default_wep_key;
    U8 wep_key_length;
    U8  WscMode;
    PS8 pWscPin;
    S32 eap;
    U8 Identity[WPACORE_MAX_CERT_PASSWORD_LENGTH];
    U8 private_key_passwd[WPACORE_MAX_CERT_PASSWORD_LENGTH];
    U8 private_key[WPACORE_MAX_CERT_PASSWORD_LENGTH];
    U8 client_cert[WPACORE_MAX_CERT_FILE_NAME_LENGTH];
    U8 password[WPACORE_MAX_CERT_PASSWORD_LENGTH];
    U8 anyWpaMode;
    U16 ccx;
} TWpaCore_WpaSupplParams;
```

```
typedef struct
{
    OS_802_11_AUTHENTICATION_MODE AuthMode;
    OS_802_11_ENCRYPTION_TYPES EncryptionTypePairWise;
    OS_802_11_ENCRYPTION_TYPES EncryptionTypeGroup;
} TWpaCore_WpaParams;
```

```
typedef enum _OS_802_11_AUTHENTICATION_MODE
{
    os802_11AuthModeOpen,
    os802_11AuthModeShared,
    os802_11AuthModeAutoSwitch,
    os802_11AuthModeWPA,
```

```
    os802_11AuthModeWPAPSK,
    os802_11AuthModeWPANone,
    os802_11AuthModeWPA2,
    os802_11AuthModeWPA2PSK,
    os802_11AuthModeMax
} OS_802_11_AUTHENTICATION_MODE;
```

```
typedef enum _OS_802_11_AUTHENTICATION_MODE
{
    os802_11AuthModeOpen,
    os802_11AuthModeShared,
    os802_11AuthModeAutoSwitch,
    os802_11AuthModeWPA,
    os802_11AuthModeWPAPSK,
    os802_11AuthModeWPANone,
    os802_11AuthModeWPA2,
    os802_11AuthModeWPA2PSK,
    os802_11AuthModeMax
} OS_802_11_AUTHENTICATION_MODE;
```

```
typedef enum _OS_802_11_ENCRYPTION_TYPES
{
    OS_ENCRYPTION_TYPE_NONE = 0,
    OS_ENCRYPTION_TYPE_WEP,
    OS_ENCRYPTION_TYPE_TKIP,
    OS_ENCRYPTION_TYPE_AES
} OS_802_11_ENCRYPTION_TYPES;
```