Summit Software Developer's Kit

User's Guide
Version 3.7

global solutions: local support™

Americas : +1-800-492-2320 Option 2
Europe : +44-1628-858-940
Hong Kong : +852-2923-0610
www.lairdtech.com/wireless

REVISION HISTORY

Version	Date	Description	Approved By
3.0	11 Jun 2013	Converted to Laird Formatting, Inclusion of "Getting Started" with Linux, Edited for SCU references (New SCU), Minor updates	Sue White
3.1	07 Nov 2013	Documented Linux Specific Functions.	Dan Kephart

3.2	03 Mar 2014	Removed all references to the following:
		`GetCerts(TRUE, 2, userCert);`
		`GetCerts(FALSE, 1, caCert);`
			Dan Kephart

3.5	19 Sept 2014	Added Platform Independent Layer (PIL) (Linux only) and Events (linux only) sections Added APIs: LRD_WF_GetaLRSBitmsk, LRD_WF_GetaLRSChannels, LRD_WF_GetbLRSBitmask, LRD_WF_GetbLRSChannels, LRD_WF_GetDHCPLease, LRD_WF_GetBSSIDList, LRD_WF_GetFipsStatus, LRD_WF_GetPillnfo, LRD_WF_GetSSID, GetWAPICertCred, SetWAPICertCred	Dan Kephart
3.6	14 Jan 2015	Added description of unsigned short roamPeriodms to SDCGlobalConfig structure's elements.	Dan Kephart
3.7	16 Oct 2015	Added **Approved By** column to Revision History	Sue White

CONTENTS

INTRODUCTION

This document is a reference guide for the software developer's kit (SDK) for Summit radio modules and cards from Laird. For an overview of Summit radio modules and cards, go to http://www.lairdtech.com/wi-fi.

Summit utilities, such as the Summit Client Utility (SCU), use the SDK to interact with other components of Summit software. Those components are:

- A device driver for the operating system running on the computing device that houses the radio.
- An integrated IEEE 802.1X supplicant.
- The registry, which is used to store configuration information.

SCU is designed for end users and administrators of mobile devices that use a Summit radio module. Using SCU, an administrator can configure radio and security settings in a configuration profile. An administrator also can use SCU to define a set of global settings which apply to all profiles and to SCU.

Note: For details on SCU functions, profile settings, and global settings, consult the SCU User's Manual. Visit www.lairdtech.com/wi-fi and click **Summit Documentation**.

On Windows platforms, SCU provides a GUI for access to all of its functions. On Linux platforms, the command line utility sdc_cli provides access to these functions. Access to these functions is also available through the Summit SDK, which can be used to manage the radio from other applications. This guide explains how to use the SDK from an application.

This SDK Programmer's Guide includes the following sections:

SDK Usage and Operation

API Reference

SDK USAGE AND OPERATION

Getting Started

For instructions on installing Summit software and a Summit radio on your development device, consult the Summit *User's Guide* accessible from the documentation tab of your product's page at Lairdtech.com.

Once Summit software and a Summit radio are installed on a device, you can use that device to write and test an application that uses the SDK.

Note: Before incorporating any Summit files, make sure that you have downloaded the latest files from the Summit website.

Linux Specific Functions

Some functions of the SDK are exclusive to Linux, or may behave differently in Linux. These functions are labelled as such within the document with a note as follows:

Note: This command is ONLY supported in Linux.

Getting started with Windows

If you want to use the SDK from a .NET application, you must create a wrapper DLL for the SDK. Summit does not provide a static library (LIB file) or dynamic link library (DLL file) for .NET.

To use the SDK from Visual Studio, you must add the Summit SDK to a new Visual Studio project as follows:

Step 1: Navigate to the tool bar and go to Project > Properties

Step 2: Link to the libraries (**Project > Settings**).
On the **Link** tab, in the **Object/library modules** box, enter:

```
sdk.lib ws2.lib iphlpapi.lib
```

Note: sdk.lib static library – Summit's SDK.
ws2.lib and iphlpapi.lib – Microsoft libraries (available in the standard Microsoft SDKs)

Make sure that you use the appropriate *sdk.lib* file for your platform.

Getting Started with Linux

To use the SDK in a Linux application:

Step 1: Use a Makefile to build an application. Make sure to include directories for header and library.

```
CFLAGS = -I$(SDKPATH)/include
LIBS = -L$(SDKPATH)/libs
```

Step 2: Include the header file in your source file (xxx.cpp or elsewhere):

```
#include "sdc_sdk.h"
```

Step 3: Link to the library.

```
-lsdc_sdk
```

Global Settings Management

Global settings include radio and security settings for all profiles and settings that apply to the configuration of the radio.

Global settings affect all properties and are applied whether a ThirdPartyConfig or a user profile is active.

Note: There is only one global configuration. Global settings always replace the existing settings.

When changing global settings using the *SetGlobalSettings* functions, the changes take effect immediately if the function returns successfully. Some settings, such as the *WMEenabled* setting, require a power-cycle if the radio is inserted.

Global settings are accessed using the GetGlobalSettings and SetGlobalSettings functions.

Related Structures for Global Settings

Structure: SDCGlobalConfig

The global settings are stored in the SDCGlobalConfig structure.

Note: Although all global settings may be retrieved and set via the SDK, some global settings are not relevant to user applications. For example, the *adminPassword* is used only for the SCU application (adjusting this global setting changes the SCU password).

Generally, to modify global settings, it is best to retrieve existing global settings, make changes, and then save global settings.

```
SDCGlobalConfig gc;
memset(&gc, 0, sizeof(gc));

//retrieve existing settings
GetGlobalSettings(&gc);

//make changes
gc.fragThreshold = ;
gc.roamTrigger = ;

//set changes
SetGlobalSettings(&gc);
```

Related Global Settings Functions

SDK global settings functions include:

- GetGlobalSettings
- SetGlobalSettings
- RadioEnable
- testTxData
- updateSROM

Profile Management

Profile settings are radio and security settings that are stored in the registry as part of a configuration profile. When a profile is selected as the active profile, the settings for that profile become active.

Note: When the profile named *ThirdPartyConfig* is selected, a power cycle also must be performed.

On the Manage Profiles window (or Profile tab in previous releases of SCU), an administrator can:

- Define up to 20 profiles, in addition to the special ThirdPartyConfig profile
- Change the settings in any profile
- Delete any profile except the special ThirdPartyConfig profile and the active profile

Profile changes made on the window are saved to the profile only when **Commit** is tapped.

Here are the primary profile management functions:

- Select and edit the applicable profile
- Create and edit a new profile
- Rename a profile
- Delete a profile
- Scan for additional radios

Using the SDK to perform profile functions is covered in the following subsections.

Edit a Profile: Set a Single Static WEP Key

Use the WEPKey structure.

Set the length of the WEP key with WEPLEN_40BIT or WEPLEN_128BIT.

To specify which key to transmit, set the XMITBIT flag (using the bitwise-OR operator) in WEPKey's length member.

Put this structure into the myConfig.WEPKeys.buffer[0] spot. Starting at the buffer[0] spot it is assumed to be an array of 4 WEPKey structures packed on a single byte boundary instead of just one WEPKey structure. The buffer should always be zeroed before filling it in.

For example:

```
unsigned char yourActiveWepKey[13] =
{0x11,0x11,0x11,0x11,0x11,0x11,0x11,0x11,0x11,0x11,0x11,0x11,0x11};

SummitCfg.wepType= WEP_ON;
```

```
SummitCfg.WEPKeys.size = sizeof(WEPKey);
SummitCfg.WEPKeys.offset = 0;

WEPKey *wKey;
wKey = (WEPKey *) &SummitCfg[0].WEPKeys.buffer[0];
wKey->length = WEPLEN_128BIT | XMITBIT; //this is the transmit key, not just a
stored key
memmove(&wKey->data[0], yourActiveWepKey, 13);
```

Edit a Profile: Set Four Static WEP Keys

Copy all four WEP keys to the SDCConfig's WEPKeys.buffer[0]. WEPKeys.buffer[0] is assumed to be the start of an array of four WEPKey structures packed on a single byte boundary instead of just a single WEPKey structure. Use a WEPKey pointer to navigate through the buffer and set the data for each key. To specify which key to transmit, set the XMITBIT flag (using the bitwise-OR operator) in WEPKey's length member.

```
WEPKey *wepK;
int nTransmitKey;
unsigned char myWEPKey[13] =
{0x11,0x11,0x11,0x11,0x11,0x11,0x11,0x11,0x11,0x11,0x11,0x11,0x11};

// The WEPKeys buffer contains all 4 keys.
// Set pointer to the first key.
wepK = (WEPKey *) &myConfig.WEPKeys.buffer[0];

// Which of the 4 WEP keys to transmit
nTransmitKey = 4;

for (int index=0; index<4; index++)
{
   //set length
   wepK->length = WEPLEN_128BIT;
   //or WEPLEN_40BIT or WEPLEN_NOT_SET

   //copy the key over
   memmove(&wepK->data[0], myWEPKey, 13);

   //is this the transmit key?
   if (index  == nTransmitKey)
      wepK->length |= XMITBIT;

   // Advance to the next key
   wepK++;
}
```

Edit a Profile: Configure LEAP

Set the SDCConfig eapType to EAP_LEAP. You can specify the LEAP credentials in the SDCConfig's userName and userPwd:

```
char credUser[65];
char credPwd[65];
//fill in above variables
```

```
memcpy(myConfig.userName.buffer, credUser, 65);
memcpy(myConfig.userPwd.buffer, credPwd, 33);
```

Related Structures for Configuration Profiles

Structure: SDCConfig

The structure SDCConfig is for a configuration profile. It stores information such as SSID, ClientName, BitRate, and all encryption and EAP credential information.

Structure: CRYPT

The structure CRYPT stores secure information that must be encrypted for storage in the registry (such as WEP keys, PSKs, EAP usernames and passwords). It is better to use functions such as SetWEPKey and SetEAPFAStCred rather than modify CRYPTs directly.

Related Profile Management Functions

Profile functions include:

- ActivateConfig
- AddConfig
- CreateConfig
- DeleteConfig
- GetAllConfigs
- GetConfig
- GetCurrentConfig
- GetEAPFASTCred
- GetEAPTLSCred
- GetEAPTTLSCred
- GetLEAPCred
- GetMultipleWEPKeys
- GetNumConfigs
- GetPEAPGTCCred
- GetPEAPMSCHAPCert
- GetPSK
- GetUserCertPassword
- GetWAPICertCred
- GetWEPKey
- ModifyConfig
- SetAllConfigs
- SetDefaultConfigValues
- SetEAPFASTCred
- SetEAPTLSCred
- SetEAPTTLSCred
- SetLEAPCred
- SetMultipleWEPKeys
- SetPEAPGTCCred
- SetPEAPMSCHAPCred
- SetPSK
- SetUserCertPassword
- SetWAPICertCred
- SetWEPKey
- Validate_WEP_EAP_Combo

Monitoring and Status

SCU includes various mechanisms for monitoring and status. The following sections illustrate how to implement SCU monitoring features in the SDK.

Obtain Status Information

To tell when you have entered an area where the SSID is available, use our SDK to poll the status. Once the AP/SSID is available, the status will change from 'not associated' to 'associated.' Check the cardState member of the CF10G_STATUS structure returned by the GetCurrentStatus function:

```
SDCERR GetCurrentStatus(CF10G_STATUS *status);
typedef enum _CARDSTATE {
  CARDSTATE_NOT_INSERTED = 0,
  CARDSTATE_NOT_ASSOCIATED,
  CARDSTATE_ASSOCIATED,
  CARDSTATE_AUTHENTICATED,
  CARDSTATE_FCCTEST,
  CARDSTATE_NOT_SDC ,
  CARDSTATE_DISABLED,
  CARDSTATE_ERROR,
  CARDSTATE_AP_MODE,
} CARDSTATE;
```

When the status is *CARDSTATE_ASSOCIATED* or *CARDSTATE_AUTHENTICATED*, the network is available.

CARDSTATE_AP_MODE is only available on Linux and only on radios that support AP mode (45 series)

Determine Signal Quality

Determining signal quality requires three values in CF10G_STATUS:

1. unsigned long DTIM (range 1-100; no associated unit)
2. unsigned long beaconPeriod (range 20-4000 Kusec or roughly 20-4000 msec)
3. unsigned long beaconsReceived

The SDK or driver fills in these values each time UpdateStatus is called. SCU calls UpdateStatus every 1500 ms. It keeps track of the last four readings and averages them to display Signal Quality.

To determine signal quality, your application should get beacons on one of the following intervals:

- In CAM powerSave mode, every beaconPeriod
- In PSP powerSave mode, every (beaconPeriod * DTIM)

Notes and Usage Tips:

In PSP, you can get more beacons than expected if the radio is transmitting data, so always round down to 100% signal quality. You'll see this especially the first 20 seconds or so.

When the driver roams or connects for the first time, *beaconsReceived* will be reset to zero.

If (*beaconPeriod * DTIM*) > sampling interval, then you should display signal quality only if there is enough data to make a decision.

Related Structures for Monitoring and Status

TBD

Related Monitoring and Status Functions

- GetCurrentStatus

ThirdPartyConfig (Windows-only feature)

Related Structures for ThirdPartyConfig

Structure: SDC3rdPartyConfig

The structure SDC3rdPartyConfig is a subset of the structure SDCConfig, because the special profile ThirdPartyConfig supports only certain configuration elements. Other elements are configured through Windows Zero Config or another application.

Related ThirdPartyConfig Functions

- Get3rdPartyConfig
- Set3rdPartyConfig

Regulatory Domains

Related ENUM for Regulatory Domains

- REGDOMAIN

Related Regulatory Domains Functions

- GetCurrentDomain

FCC (Windows only feature)

Related ENUM for FCC

- FCC_TEST

Related FCC Functions

- FirstFCCTest
- NextFCCTest

Events (Linux only feature)

SDK Events is an event driven mechanism that allows programs to monitor the wireless subsystem.

Events can aid in developing a connection manager or allow for better debugging of the wireless subsystem. Laird supplies an example program with source called Event Monitor (event_mon) that outputs events as they occur to console or syslog. The Event Monitor source is an exhaustive example on how to use SDK Events. The following is a simple example on of how to use SDK Events .

Events Code Example

```
int quit = 0;
unsigned long long eventMask = SDC_E_READY | SDC_E_ASSOC | SDC_E_ROAM;

SDCERR event_handler(unsigned long event_type, SDC_EVENT *event)
{
    if (event_type == eventMask) {
        printf(Found a registered event”);
        quit = 1;
    }
    return SDCERR_SUCCESS;
}

int main(int argc, char *argv[])
{
    rc = SDCRegisterForEvents(eventMask, event_handler);
    if(rc != SDCERR_SUCCESS) {
        printf("Failed to Register for Events with rc (%d)", rc);
        SDCDeregisterEvents();
        return 1;
    }
    SDCRegisteredEventsList(&eventMask);
    printf("Current Registered Bitmask 0x%016llX\n", eventMask);

    while(!quit)
        sleep(1);

    SDCDeregisterEvents();
    exit(0);
}
```

Implementing DHCP Events on MSD/SSD products

DHCP Injector

Laird supplies a command line program called DHCP Injector (dhcp_injector) to inject DHCP events into programs using SDK Events. Using dhcp_injector by calling it with the flag -s or --s with the appropriate status listed in LRD_WF_EvtDHCPStatus.

Example:

dhcp_injector -s BOUND

Reason Code

In order for SDK Events to determine if the IP address is the same or different, the PIL function

LRD_WF_PIL_GetDHCPLease must be implemented.

Related Events Functions

- SDCRegisterForEvents
- SDCDeregisterEvents
- SDCRegisteredEventsList

Platform Independent Layer (Linux only feature)

The platform independent layer (PIL) is used to supply functionality that is platform dependent and supplied by the developer. This is functionality that will allow the developer to use their own methods to accomplish the desired action.

The PIL functionality is provided by the use of a customer's supplied library. The library is created with the name liblrd_pil_wf.so. The SDK looks for this library and when found will use the functions within for the PIL functionality.

Currently the PIL is required in order to set and retrieve regulatory domain information, and retrieve DHCP Lease information. Regulator information should be stored in a manner that allows the protection of the setting should the user remove the profiles, in order to be persistent. To accommodate the DHCP client the customer uses, DHCP Lease information retrieval will need to be adjusted.

There are required portions that must be created in the library as well as optional portions. Optional functions that are not desired need not be instantiated in code.

Information on the structures and functions are in the API Reference section dealing with the PIL.

API REFERENCE

This section describes the available functions in the SDK and provides sample code for each. Each function is provided with a description, usage parameters, returns, and additional information.

Functions

- ActivateConfig
- AddConfig
-
- CreateConfigCreateConfig
- DeleteConfig
-
- exportSettingsexportSettings
- FirstFCCTest
- FlushAllConfigKeys
- FlushConfigKeys
- Get3rdPartyConfig
- GetAllConfigs
- GetAllConfigs Sample Code
- GetBSSIDList
- GetConfig
- GetConfigFileInfo
- GetCurrentConfig
- GetCurrentDomain
- GetCurrentStatus
- GetEAPFASTCred
-
- GetEAPTLSCredGetEAPTLSCred
- GetEAPTTLSCred
- GetGlobalSettings
- GetLEAPCred
-
- GetMultipleWEPKeysGetMultipleWEPKeys
- GetNumConfigs
- GetPEAPGTCCred
- GetPEAPMSCHAPCred
- GetPEAPTLSCred
- GetPSK
- GetSDKVersion
- GetUserCertPassword
- GetWAPICertCred
- importSettings
- LRD_WF_GetaLRSBitmask
- LRD_WF_GetaLRSChannels
- LRD_WF_GetbLRSBitmask
- LRD_WF_GetbLRSChannels
- LRD_WF_GetDHCPLease (linux only)
- LRD_WF_GetBSSIDList
- LRD_WF_GetFipsStatus
- LRD_WF_GetPILInfo
- LRD_WF_GetSSID
- NextFCCTest
- QueryOID
- RadioEnable
- RadioDisable
- SDCDeRegisterEvents
- SDCRegisterForEvents
- SDCRegisteredEventsList
- Set3rdPartyConfig
- SetAllConfigs
- SetDefaultConfigValues
- SetEAPFASTCred
- SetEAPTLSCred
- SetEAPTTLSCred
- SetGlobalSettings
- SetLEAPCred
- SetMultipleWEPKeys
- SetOID
-
-
- SetOID
- This function sets an NDIS OID with DeviceIOControl.

```
LONG SetOID (ULONG ndis_oid, void
*buffer, ULONG bufSize)
```

Parameters:

- [in] *ndis_oid* – The NDIS OID to query
- [in] *buffer* – In/Out
- [in] *bufSize* – In/Out

Returns:

- 0 – Failure. Call GetLastError for error information
- Non-zero – Success

SetOID Sample Code

- SetPEAPGTCCred
- SetPEAPMSCHAPCred
- SetPEAPTLSCred
- SetPSK
- SetUserCertPassword
- SetWAPICertCred
- SetWEPKey
- testTxData (Windows Only)
- updateSROM (Windows Only)
- Validate_WEP_EAP_Combo

Function Descriptions

ActivateConfig

This function activates the configuration with the given name.

```
SDCERR ActivateConfig(char *name)
```

Parameters:

- [in] *name* – Name of the configuration to make the active one.

This function succeeds even if the card is not present so, when it is inserted, this becomes the active configuration.

To use a third party WLAN framework, pass in '*ThirdPartyConfig*' for the name.

Note: In order for ThirdPartyConfig to work, a power cycle is required (going to or from it).

Returns:

- **SDCERR_SUCCESS** – Successful
- **SDCERR_INVALID_NAME** – No match for the name
- **SDCERR_POWERCYCLE_REQUIRED** – A power cycle is required for this to take effect

ActivateConfig Sample Code

AddConfig

This function adds the configuration.

```
SDCERR AddConfig(SDCConfig *cfg)
```

Parameters:

- [in] *cfg* – Configuration.

Returns:

- SDCERR_SUCCESS – Successful.
- SDCERR_INVALID_NAME – Name already exists.
- SDCERR_INVALID_CONFIG – Configuration contains bad parameters.
- SDCERR_FAIL – Internal error or the maximum number of configurations has been exceeded.

AddConfig Sample Code

CreateConfig

This function creates a configuration from the default values.

```
SDCERR CreateConfig(SDCConfig *cfg)
```

Rules:

- You must allocate the config memory.
- You must add the config after it is created.

Parameters:

- [in] *cfg* – Configuration.

Returns:

- **SDCERR_SUCCESS** – Successful.
- **SDCERR_FAIL** – Internal error.

CreateConfig Sample Code

DeleteConfig

This function deletes the configuration matching 'name'. You are not allowed to delete the active configuration.

```
SDCERR DeleteConfig(char *name)
```

Rules:

- You are not allowed to delete the active configuration.
- '*ThirdPartyConfig*' is not allowed with this function.
- *NULL* is not a valid name.

Parameters:

- [in] *name* – Name of the configuration that you want to delete.

Returns:

- **SDCERR_SUCCESS** – Successful.
- **SDCERR_INVALID_NAME** – Cannot match name.
- **SDCERR_INVALID_DELETE** – Trying to delete the active configuration.

DeleteConfig Sample Code

exportSettings

This function exports configurations, global settings, and third party config to the specified file.

```
SDCERR exportSettings (char *filename, SDC_ALL *all)
```

Parameters:

- [in] *filename* – A valid filename (required)
- [in] *all* – Specifies which information to export
  - *configGlobal* – Either NULL to skip global config export or a valid pointer
  - *configThirdParty* – Either NULL to skip third party config export or a valid pointer
  - *configs* – Either NULL to skip configs export or a valid pointer to one or more SDCConfig structures
  - *numConfigs 0* – To skip all SDCConfigs or the number of configurations (SDCConfig) to export

---

Note: Don't include *configGlobal* or *configThirdParty* in this count.

Returns:

- SDCERR_INVALID_PARAMETER – Invalid filename or all structure
- SDCERR_INVALID_CONFIG – Invalid configuration (global, third party, or config)
- SDCERR FAIL – Other error

exportSettings Sample Code

FirstFCCTest (Windows only)

This function puts the radio into FCC testing mode on the next power cycle.

```
SDCERR FirstFCCTest(FCC_TEST test, BITRATE rate, int channel, TXPOWER
testPower, unsigned long timeout)
```

Parameters:

- [in] *test* – Type of test including:
  - 1 – Continuous transmit
  - 2 – Frequency accuracy
  - 3 – Continuous receive
- [in] *rate* – Test rate
- [in] *channel* – Test channel
- [in] *testPower* – Test power
- [in] *timeout* – Test timeout

Returns:

- **SDCERR_POWERCYCLE** – Successful
- **SDCERR_FAIL** – Error

FlushAllConfigKeys

This function flushes all Summit configuration registry keys. Depending on the system, registry changes are flushed to disk after a system-specified interval of time and at shutdown. This function forces a flush so the Summit parameters are saved if a power-cycle occurs before the system flushes the registry.

Note: This is an expensive operation.

```
SDCERR FlushAllConfigKeys ()
```

Returns:

- **SDCERR_SUCCESS** – Successful
- **SDCERR_FAILURE** – Error

FlushAllConfigKeys Sample Code

FlushConfigKeys

This function flushes the specified registry keys. Depending on the system, registry changes are flushed to disk after a system-specified interval of time and at shutdown. This function forces a flush so the Summit parameters are saved if a power-cycle occurs before the system flushes the registry.

Note: This is an expensive operation.

```
SDCERR FlushConfigKeys (int configNumber)
```

Parameters:

- [in] *configNumber* – The configuration to flush
  - -1 – Flushes the Global Settings
  - 0 – Flushes the ThirdPartyConfig
  - +1 – MAX_CFGS flushes the specified config number

Returns:

- SDCERR_SUCCESS – Successful
- SDCERR_INVALID_PARAMETER – Invalid configNumber

FlushConfigKeys Sample Code

Get3rdPartyConfig

The function retrieves the third party configuration settings.

```
SDCERR Get3rdPartyConfig(SDC3rdPartyConfig *cfg3rd)
```

Parameters:

- [in]*cfg3rd* – 3rd party config.

Returns:

- **SDCERR_SUCCESS** – Successful
- **SDCERR_INVALID_CONFIG** - ***gcfg*** is NULL
- **SDCERR_FAIL** – Internal error

Get3rdPartyConfig Sample Code

GetAllConfigs

This function retrieves all of the configurations (except ThirdPartyConfig).

```
SDCERR GetAllConfigs (SDCConfig *cfgs, unsigned long *num)
```

Parameters:

- [out] *cfgs* – Space for at least MAX_CFGS configs
- [out] *num* – Number of configurations
- ignored – NULL

Returns:

- SDCERR_SUCCESS – Successful

Note: The order of configurations is always maintained when other configurations are added or deleted. For example, if you delete config #3, then config #4 moves into its spot (become config #3). Configs #1 and #2 do not change.
Newly added configurations are added to the end of the profile list.

GetAllConfigs Sample Code

GetBSSIDList

This function gets a list of BSSIDs from a scan.

```
SDCERR GetBSSIDList(SDC_802_11_BSSID_LIST_EX *list, int *numBuffEntries)
```

Parameters:

- [out] *list* – pointer to an 802_11_BSSID_LIST_EX structure
- [out] *numBuffEntries – pointer to int with number of SCAN_ITEM_INFO elements in the srtructure*

Returns:

- SDCERR_SUCCESS – Successful
- SDCERR_INVALID_PARAMETER if invalide parameter
- SDCERR_INSUFFICIENT_MEMORY if list structure is not large enough
- SDCERR_FAIL if error
- SDCERR_NOT_SUPPORTED if not supported

Note: On Linux, Laird recommends the API LRD_WF_GetBSSIDList instead of GetBSSIDList as the former handles for non-ASCII SSIDs and returns multiple supported encryption types of each AP.

GetConfig

This function retrieves the configuration information for the configuration profile with the specified name.

```
SDCERR GetConfig(char *name, SDCConfig *cfg);
```

Parameters:

- [in] name – Name of the configuration to retrieve
- [out] cfg - Configuration

Rules:

- Cannot be NULL
- Cannot be "ThirdPartyConfig"; use the function Get3rdPartyConfig instead.

Return values:

- SDCERR_SUCCESS – Successful
- SDCERR_INVALID_NAME – No profile has specified name
- SDC_INVALID_CONFIG – *cfg isn't valid
- SDCERR_FAIL – Internal error or *cfgs is NULL

GetConfig Sample Code

GetConfigFileInfo

This function retrieves file details from a Summit configuration file.

```
SDCERR GetConfigFileInfo (char *filename, CONFIG_FILE_INFO *info)
```

Parameters:

- [in] *filename* – A valid filename (required)
- [out] *info* – Pass in a pointer to an allocated CONFIG_FILE_INFO structure

Returns:

- SDCERR_INVALID_PARAMETER – Invalid filename or info
- SDCERR FAIL – Other error

GetConfigFileInfo Sample Code

GetCurrentConfig

This function returns the number and name of the active configuration profile.

```
SDCERR GetCurrentConfig (unsigned long *num, char *name)
```

Parameters:

- [in] *num* – If NULL, item is skipped
  - 0 – ThirdPartyConfig is active
  - >0 – Number of active configuration profile
- [out] *name* – If NULL, item is skipped. 'ThirdPartyConfig' if the ThirdPartyConfig is active, otherwise the name of the active profile is stored here.
- Rule: You must allocate and pass in at least CONFIG_NAME_SZ bytes of storage with this argument.

Returns:

- SDCERR_SUCCESS – Successful
- SDCERR_FAIL – Unsuccessful

GetCurrentConfig Sample Code

GetCurrentDomain

Note: This command returns certain values ONLY in Linux. See the list below.

This function returns the current regulatory domain set in the SROM.

```
REG_DOMAIN GetCurrentDomain()
```

Returns:

- REG_FCC – If the regulatory domain is FCC
- REG_ETSI – If the regulatory domain is ETSI
- REG_TELEC – If the regulatory domain is TELEC
- REG_KCC – If the regulatory domain is KCC
- REG_WW – If it is set in WorldWide mode.

In Linux, this command may also return the following:

- REG_CA – If using CA country code
- REG_FR – If using FR country code
- REG_GB – If using GB country code
- REG_AU – If using AU country code
- REG_NZ – If using NZ country code

Note: If set in REG_WW mode, it should be safe for all regulatory domains (but is not optimized for any particular domain).

Note 2: This is a lengthy call. It should not and need not be called frequently. The value is stored in SROM and it requires significant time to access it.

GetCurrentDomain sample Code

GetCurrentStatus

This function retrieves status for the card, IP information, MAC information, AP association information, etc.

```
SDCERR GetCurrentStatus(CF10G_STATUS *status)
```

Parameters:

- [out] *status* – Area to retrieve the card status information

Returns:

- SDCERR_SUCCESS – Successful
- SDCERR_FAIL – Internal error or *status is NULL.

GetCurrentStatus Sample Code

GetEAPFASTCred

This function retrieves the EAP-FAST credentials.

```
SDCERR GetEAPFASTCred (SDCConfig *cfg, char *username, char *password, char
*pacfilename, char *pacpassword)
```

Parameters:

- [in] *cfg* – Valid configuration (required)
- [out] *username* – Pass a valid pointer with allocated buffer of at least USER_NAME_SZ characters. If NULL, this parameter is ignored
- [out] *password* – Pass in a valid pointer with an allocated buffer of at least USER_PWD_SZ characters. If NULL, the parameter is ignored
- [out] *pacfilename* – Pass in a valid pointer with an allocated buffer of at least CRED_PFILE_SZ characters. If NULL, this parameter is ignored
- [out] *pacpassword* – Pass in a valid pointer with an allocated buffer of at least CRED_PFILE_SZ characters. If NULL, the parameter is ignored

Returns:

- SDCERR_INVALID_PARAMETER – Invalid parameter
- SDCERR_INVALID_CONFIG – Invalid configuration
- SDCERR_FAIL – Other error

GetEAPFASTCred Sample Code

GetEAPTLSCred

This function retrieves the EAP-TLS credentials.

```
SDCERR GetEAPTLSCred (SDCConfig *cfg, char *username, char *password, CERTLOCATION
certLocation, char *caCert)
```

Parameters:

- [in] *cfg* – Valid configuration (required)
- [out] *username* – Pass in a valid pointer with an allocated buffer of at least USER_NAME_SZ characters. If NULL, this parameter is ignored
- [out] *userCert* – Pass a valid pointer with allocated buffer of at least 20 characters. If NULL, it is ignored
- [out] *certLocation* – Pass in a valid pointer. If NULL, this parameter is ignored
- [out] *caCert* – Pass in a valid pointer with an allocated buffer of at least CRED_CERT_SZ characters. If NULL, this parameter is ignored.
  Depending on the caCertLocation field, caCert will contain:
  - CERT_NONE – caCert is NULL. Do not validate the server
  - CERT_FILE – caCert will specify the cert filename, up to CRED_CERT_SZ characters
  - CERT_FULL_STORE – caCert is NULL. The full MS certificate store will be searched for a valid certificate.
  - CERT_IN_STORE – caCert is a 20-byte hash representing one specific cert from the MS-store

Returns:

- SDCERR_INVALID_PARAMETER – Invalid parameter
- SDCERR_INVALID_CONFIG – Invalid configuration
- SDCERR_FAIL – Other error

GetEAPTLSCred Sample Code

GetEAPTTLSCred

This function retrieves the EAP-TTLS credentials.

```
SDCERR GetEAPTTLSCred (SDCConfig *cfg, char *username, char *password,
CERTLOCATION *certLocation, char *caCert)
```

Parameters:

- [in] *cfg* – Valid configuration (required)
- [out] *username* – Pass in a valid pointer with an allocated buffer of at least USER_NAME_SZ characters. If NULL, this parameter is ignored
- [out] *password* – Pass in a valid pointer with an allocated buffer of at least USER_PWD_SZ characters. If NULL, this parameter is ignored
- [out] *certLocation* – Pass in a valid pointer. If NULL, this parameter is ignored
- [out] *caCert*- Pass in a valid pointer with an allocated buffer of at least CRED_CERT_SZ characters. If NULL, this parameter is ignored

  Depending on the caCertLocation field, caCert will contain:
  - CERT_NONE – caCert is NULL. Do not validate the server
  - CERT_FILE – caCert will specify the cert filename, up to CRED_CERT_SZ characters
  - CERT_FULL_STORE – caCert is NULL. The full MS certificate store will be searched for a valid certificate
  - CERT_IN_STORE – caCert is a 20-byte hash representing one specific certificate from the MS-store.

Returns:

- SDCERR_INVALID_PARAMETER – Invalid parameter
- SDCERR_INVALID_CONFIG – Invalid configuration
- SDCERR_FAIL – Other error

GetEAPTTLSCred Sample Code

GetGlobalSettings

This function retrieves the global configuration settings.

```
SDCERR GetGlobalSettings(SDCGlobalConfig *gcfg)
```

Parameters:

- [out] *gcfg* – Global configuration

Returns:

- SDCERR_SUCCESS – Successful
- SDCERR_INVALID_CONFIG - **gcfg* is NULL
- SDCERR_FAIL – Internal error

GetGlobalSettings Sample Code

GetLEAPCred

This function retrieves the LEAP credentials.

```
SDCERR GetLEAPCred (SDCConfig *cfg, char *username, char *password)
```

Parameters:

- [in] *cfg* – Valid configuration (required)
- [out] *username* – Pass in a valid pointer with an allocated buffer of at least USER_NAME_SZ characters. If NULL, this parameter is ignored
- [out] *password* – Pass in a valid pointer with an allocated buffer of at least USER_PWD_SZ characters. If NULL, this parameter is ignored

Returns:

- SDCERR_INVALID_PARAMETER – Invalid parameter
- SDCERR_INVALID_CONFIG – Invalid configuration
- SDCERR_FAIL – Other error

GetMultipleWEPKeys

This function retrieves all four WEP keys.

```
SDCERR GetMultipleWEPKeys (SDCConfig *cfg, int *nTxKey, WEPLEN *key1Length,
unsigned char *key1, WEPLEN *key2Length, unsigned char *key2, WEPLEN
*key3Length, unsigned char *key3, WEPLEN *key4Length, unsigned char *key4)
```

Parameters:

- [in] *cfg* – Valid configuration (required)
- [out] *nTxKey* – Returns which key (1, 2, 3, or 4) is currently the transmit key
- [out]*key1Length* – Returns the length of key 1
- [out] *key1* – Pass in an allocated buffer of at least 26 (hex) characters
- [out] *key2Length* – Returns the length of key 2
- [out] *key2* – Pass in an allocated buffer of at least 26 (hex) characters
- [out] *key3Length* – Returns the length of key 3
- [out] *key3* – Pass in an allocated buffer of at least 26 (hex) characters
- [out] *key4Length* – Returns the length of key 4
- [out] *key4* – Pass in an allocated buffer of at least 26 (hex) characters

Returns:

- SDCERR_INVALID_WEP_TYPE – wepType is not WEP_ON or WEP_CKIP
- SDCERR_INVALID_EAP_TYPE – eapType is not EAP_NONE
- SDCERR_INVALID_PARAMETER – Invalid parameter
- SDCERR_INVALID_CONFIG – Invalid configuration
- SDCERR_FAIL – Other error

GetMultipleWEPKeys Sample Code

GetNumConfigs

This function retrieves the number of configurations present.

```
SDCERR GetNumConfigs(unsigned long *num)
```

Parameters:

- [out] *num* – Number of current configurations ('ThirdPartyConfig' is not counted as a configuration)

Returns:

- SDCERR_SUCCESS – Successful
- SDCERR_FAIL – Internal error

GetNumConfigs Sample Code

GetPEAPGTCCred

This function retrieves the PEAP-GTC credentials.

```
SDCERR GetPEAPGTCCred (SDCConfig *cfg, char *username, char *password,
CERTLOCATION *CAcertLocation, char *caCert)
```

Parameters:

- [in] *cfg* – Valid configuration (required)
- [out] *username* – Pass in a valid pointer with an allocated buffer of at least USER_NAME_SZ characters. If NULL, this parameter is ignored
- [out] *password* – Pass in a valid pointer with an allocated buffer of at least USER_PWD_SZ characters. If NULL, this parameter is ignored
- [out] *CAcertLocation* – Pass in a valid pointer. If NULL, this parameter is ignored
- [out] *caCert* – Pass in a valid pointer with an allocated buffer of at least CRED_CERT_SZ characters. If NULL, this parameter is ignored

  Depending on the caCertLocation field, caCert will contain:

  - CERT_NONE – caCert is NULL. Do not validate the server
  - CERT_FILE – caCert will specify the certificate filename up to CRED_CERT_SZ characters
  - CERT_FULL_STORE – caCert is NULL. The full MS certificate store is searched for a valid certificate
  - CERT_IN_STORE – caCert is a 20-byte hash representing one specific certificate from the MS-store

Returns:

- SDCERR_INVALID_PARAMETER – Invalid parameter
- SDCERR_INVALID_CONFIG – Invalid configuration
- SDCERR_FAIL – Other error

GetPEAPGTCCred Sample Code

GetPEAPMSCHAPCred

This function retrieves the PEAP-MSCHAP credentials.

```
SDCERR GetPEAPMSCHAPCred (SDCConfig *cfg, char *username, char *password,
CERTLOCATION *CAcertLocation, char *caCert)
```

Parameters:

- [in] *cfg* – Valid configuration (required)
- [out] *username* – Pass in a valid pointer with an allocated buffer of at least USER_NAME_SZ characters. If NULL, this parameter is ignored
- [out] *password*– Pass in a valid pointer with an allocated buffer of at least USER_PWD_SZ characters. If NULL, this parameter is ignored
- [out] CA*certLocation* – Pass in a valid pointer. If NULL, this parameter is ignored
- [out] *caCert* – Pass in a valid pointer with an allocated buffer of at least CRED_CERT_SZ characters. If NULL, this parameter is ignored.

  Depending on the caCertLocation field, caCert will contain:

  - CERT_NONE – caCert is NULL. Do not validate the server
  - CERT_FILE – caCert will specify the cert filename, up to CRED_CERT_SZ characters
  - CERT_FULL_STORE – caCert is NULL. The full MS certificate store will be searched for a valid certificate.
  - CERT_IN_STORE – caCert is a 20-byte hash representing one specific cert from the MS-store

Returns:

- SDCERR_INVALID_PARAMETER – Invalid parameter
- SDCERR_INVALID_CONFIG – Invalid configuration
- SDCERR_FAIL – Other error

GetPEAPMSCHAPCert Sample Code

GetPEAPTLSCred

This function returns the PEAPTLS credentials

```
SDCERR GetPEAPTLSCred (SDCConfig * cfg,  char * username, char* userCert,
CERTLOCATION* certLocation, char* caCert);
```

Parameters:

- [in] cfg--a valid configuration, required
- [out] username--pass in a valid pointer with an allocated buffer of at least USER_NAME_SZ characters. if NULL, this parameter is ignored
- [out] userCert--pass in a valid pointer with an allocated buffer of at least CRED_CERT_SZ characters. if NULL, this parameter is ignored
- [out] CAcertLocation--pass in a valid pointer. if NULL, this parameter is ignored
- [out] caCert--pass in a valid pointer with an allocated buffer of at least CRED_CERT_SZ characters. if NULL, this parameter is ignored.

  Depending on the caCertLocation field, caCert will contain:

  - CERT_NONE – caCert is NULL. Do not validate the server
  - CERT_FILE – caCert will specify the cert filename, up to CRED_CERT_SZ characters
  - CERT_FULL_STORE – caCert is NULL. The full MS certificate store will be searched for a valid certificate.
  - CERT_IN_STORE – caCert is a 20-byte hash representing one specific cert from the MS-store

Returns:

- SDCERR_INVALIDPARAMETER if an invalid parameter
- SDCERR_INVALID_CONFIG if an invalid config
- SDCERR_FAIL if other err

GetPEAPTLSCred Sample Code

GetPSK

This function retrieves the PSK.

```
SDCERR GetPSK (SDCConfig *cfg, char *psk)
```

Parameters:

- [in] *cfg* – Valid configuration (required)
- [out] *psk* – Pass in an allocated buffer of at least PSK_SZ

Returns:

- SDCERR_INVALID_WEP_TYPE – webType is not WPA PSK or WPA2 PSK
- SDCERR_INVALID_EAP_TYPE – eapType is not EAP_NONE
- SDCERR_INVALID_PARAMETER – Invalid parameter
- SDCERR_INVALID_CONFIG – Invalid configuration
- SDCERR_FAIL – Other error

GetPSK Sample Code

GetSDKVersion

This function returns the version of the SDK.

```
SDCERR GetSDKVersion (unsigned long *version)
```

Parameters:

- [out] version

Returns:

- SDCERR_SUCCESS – Successful
- SDCERR_FAIL – Error

GetSDKVersion Sample Code

GetUserCertPassword

This function retrieves the user certificate password.

```
SDCERR GetUserCertPassword(SDCConfig *cfg, char * userPswd)
```

Parameters:

- [in] cfg a valid configuration
- [out] userPswd pointer to buffer USER_PWD_SZ long

Returns:

- SDCERR_SUCCESS – Successful
- SDCERR_INVALID_PARAMETER if an invalid parameter
- SDCERR_INVALID_CONFIG if an invalid config
- SDCERR_FAIL – Error

GetWAPICertCred

This function retrieves the WAPI EAP credentials

```
SDCERR GetWAPICertCred(SDCConfig * cfg,  char * username, char* userCert,
CERTLOCATION* certLocation, char* caCert);
```

Parameters:

- [in] cfg--a valid configuration, required
- [out] username--pass in a valid pointer with an allocated buffer of at least USER_NAME_SZ characters. if NULL, this parameter is ignored
- [out] userCert--pass in a valid pointer with an allocated buffer of at least 20 characters. if NULL, this parameter is ignored
- [out] CAcertLocation--pass in a valid pointer. if NULL, this parameter is ignored
- [out] caCert--pass in a valid pointer with an allocated buffer of at least CRED_CERT_SZ characters. if NULL, this parameter is ignored; depending on the caCertLocation field, caCert will contain:
  - if CERT_NONE, caCert is NULL - don't validate the server
  - if CERT_FILE, caCert will specify the cert filename, up to CRED_CERT_SZ chars

Return:

- SDCERR_SUCCESS if successful
- SDCERR_INVALIDPARAMETER if an invalid parameter
- SDCERR_INVALID_CONFIG if an invalid config
- SDCERR_FAIL if other err

GetWEPKey

This function retrieves a WEP key.

```
SDCERR GetWEPKey (SDCConfig *cfg, int nWepKey, WEPLEN *keyLength, unsigned char
*key, BOOLEAN *txKey)
```

Parameters:

- [in] *cfg* – Valid configuration (required)
- [out] *nWepKey* – Indicates which of the four stored WEP keys (1, 2, 3, or 4) to retrieve
- [out] *keyLength* – WEP key length. If NULL, this parameter is ignored
- [out] *key* – Pass in an allocated buffer of at least 26 (hex) characters. If NULL, this parameter is ignored
- [out] *txKey* – Returns if this is the active transmit key. If NULL, this parameter is ignored

Returns:

- SDCERR_INVALID_WEP_TYPE – wepType is not WEP_ON or WEP_CKIP

- SDCERR_INVALID_EAP_TYPE – eapType is not EAP_NONE
- SDCERR_INVALID_PARAMETER – Invalid parameter
- SDCERR_INVALID_CONFIG – Invalid configuration
- SDCERR_FAIL – Other error

GetWEPKey Sample Code

importSettings

This function imports Summit settings from the file and writes to the SDC_ALL structure. You must allocate the memory in the SDC_ALL structure - configGlobal, configThirdParty, and up to MAX_CFGS configs.

```
SDCERR importSettings (char *filename, SDC_ALL *all)
```

Parameters:

- [in] *filename* – Valid filename (required)
- [in] *all* – Specifies which information to import (imported information will be saved in this file).
  - configGlobal – NULL to skip global configuration import or a valid pointer to an allocated structure
  - configThirdParty –NULL to skip ThirdPartyConfig import or a valid pointer to an allocated structure
  - configs –NULL to skip configs import or a valid pointer to 1+ allocated SDCConfig structures
  - numConfigs – Set to the number of configurations (SDCConfig) exported. This count doesn't include configGlobal or configThirdParty.

Returns:

- SDCERR_INVALID_PARAMETER – Invalid parameter
- SDCERR_FAIL – Other error

importSettings Sample Code

LRD_WF_GetaLRSBitmask

This function converts an array of a band channels to a bit mask for use in a SDCGlobalConfig structure

```
SDCERR LRD_WF_GetaLRSBitmask(unsigned long numChannels, unsigned long *channels,
unsigned long *bitmask)
```

Parameters:

- [in] numChannels - number of channels in array
- [in] channels - array of channels
- [in/out] bitmask - pointer to ulong; bit mask is valid if SDCERR_SUCCESS returned

Returns:

- SDCERR_SUCCESS if successful
- SDCERR_INVALID_PARAMETER - if an channel given is invalid or not supported

LRD_WF_GetaLRSBitmask Sample Code

LRD_WF_GetaLRSChannels

This function converts a channel bit mask to an array of a band channels.

```
SDCERR LRD_WF_GetaLRSChannels(unsigned long *numChannels, unsigned long
*channels, unsigned long bitmask)
```

Parameters:

- [out] numChannels - pointer to number of channels found from bitmask conversion
- [in] channels - array of channels to fill in
- [in] bitmask to convert

Returns:

- SDCERR_SUCCESS if successful
- SDCERR_FAIL internal err

LRD_WF_GetaLRSChannels sample code

LRD_WF_GetbLRSBitmask

This function converts an array of b band channels to a bit mask for use in a SDCGlobalConfig structure

```
SDCERR LRD_WF_GetbLRSBitmask(unsigned long numChannels, unsigned long *channels,
unsigned long *bitmask)
```

Parameters:

- [in] numChannels - number of channels in array
- [in] channels - array of channels
- [in/out] bitmask - pointer to ulong; bit mask is valid if SDCERR_SUCCESS returned

Returns:

- SDCERR_SUCCESS
- SDCERR_INVALID_PARAMETER - if an channel given is invalid or not supported

LRD_WF_GetbLRSBitmask sample code

LRD_WF_GetbLRSChannels

This function converts a channel bitmask to an array of b band channels

```
SDCERR LRD_WF_GetbLRSChannels(unsigned long *numChannels, unsigned long
*channels, unsigned long bitmask)
```

Parameters:

- [out] numChannels - pointer to number of channels found from bitmask conversion
- [in] channels - array of channels to fill in
- [in] bitmask - bitmask to convert

Return:

- SDCERR_SUCCESS if successful
- SDCERR_FAIL internal error

LRD_WF_GetbLRSChannels sample code

LRD_WF_GetDHCPLease (Linux only)

This function returns the current dhcp lease information for the wifi interface

```
SDCERR LRD_WF_GetDHCPLease(DHCP_LEASE *dhcpLease)
```

Parameters:

- DHCP_LEASE *dhcpLease - will contain the structure filled with the current DHCP lease.

Returns:

- SDCERR_SUCCESS - *dhcpLease has the current lease info
- SDCERR_FAIL - unable to find current lease in file
- SDCERR_INVALID_FILE - error opening leases file
- SDCERR_INSUFFICIENT_MEMORY - error allocating memory

LRD_WF_GetDHCPLease sample code

LRD_WF_GetBSSIDList (Linux only)

This function returns a list of BSSIDs from a scan, and includes all supported encryption types

```
SDCERR LRD_WF_GetBSSIDList(LRD_WF_BSSID_LIST *list, int *numBufEntries)
```

Parameters:

- [out] list – Pointer to a user supplied list of LRD_WF_BSSID_LIST elements,
- [in/out] numBufEntries – user supplies the number of elelments available in the list. Function returns the number of BSSIDs elements required if not enough were supplied. See note below.

Returns:

- SDCERR_SUCCESS if successful
- SDCERR_INVALID_PARAMETER if invalid parameter,
- SDCERR_INSUFFICIENT_MEMORY if list structure is not large enough (all data that will fit will be copied)
- SDCERR_NOT_IMPLEMENTED if not implemented on the platform
- SDCERR_FAIL if internal err

Note: The number of elements returned is indicated in NumberOfItems. If an error occurs due to insufficient memory, the total number of needed elements is returned in the numBufEntries entry of the LRD_WF_BSSID_LIST structure.

LRD_WF_GetBSSIDList sample code

LRD_WF_GetFipsStatus (linux only)

This function returns the status of FIPS based on the current state the supplicant was started, as well as the state that is set for the next invocation of wireless startup.

```
SDCERR LRD_WF_GetFipsStatus(char * current, char * nextStart);
```

Parameters:

- [out] current - pointer to a byte value. 1 is enabled, 0 is disabled, -1 indicates error
- [out] nextStart - pointer to a byte value. 1 is enabled, 0 is disabled

Returns:

- SDCERR_INVALID_PARAMETER - if ether pointer is invalid
- SDCERR_FAILURE - unable to get status

LRD_WF_GetFipsStatus sample code

LRD_WF_GetPilInfo (linux only)

This function returns the LARD_WF_pilInfo structure

```
SDCERR LRD_WF_GetPilInfo(LRD_WF_PilInfo *pil_info);
```

Parameters:

- [out]pil_info - pointer to a LRD_WF_pilInfo structure.

Returns:

- SDCERR_SUCCESS if successful
- SDCERR_INVALID_PARAMETER if pil_info is NULL
- SDCERR_FAIL is no PIL is registered.

LRD_WF_GetPilInfo sample code

LRD_WF_GetSSID

```
SDCERR LRD_WF_GetSSID(LRD_WF_SSID *ssid);
```

This function returns the current SSID if associated.

Parameter:

- [out] SSID - a valid pointer to a ssidStruct

Returns:

- SDCERR_INVALID_PARAMETER - if parameter is NULL
- SDCERR_SUCCESS - SSID structure is filled in with value and len of SSID
- SDCERR_FAIL - no SSID

Note: The returned ssid.val need not be a string and could contain null characters which are allowed in SSIDs. ssid.len will indicate the length of the ssid.val field. Do not treat ssid.val as a string.

LRD_WF_GetSSID sample code

ModifyConfig

This function updates the config matching 'name'. If this is the current config, then it restarts the driver with the new config.

```
SDCERR ModifyConfig(char *name, SDCConfig *cfg)
```

Parameters:

- [in] *name* – Name of the configuration to update. 'ThirdPartyConfig' is not modifiable with this function; use Set3rdPartyConfig
- [in] *cfg* – Configuration

Returns:

- SDCERR_SUCCESS – Invalid parameter
- SDCERR_INVALID_NAME – Can't match name
- SDCERR_INVALID_CONFIG – Configuration data is invalid

ModifyConfig Sample Code

NextFCCTest (Windows only)

This function changes the FCC test for a unit that is currently running in FCC test mode. A valid call to FirstFCCTest() followed by a power cycle puts the unit into FCC test mode.

```
SDCERR NextFCCTest(FCC_TEST test, BITRATE rate, int channel, TXPOWER testPower,
unsigned long timeout)
```

Parameters:

- [in] *test* – Test type including:
  - 1 – continuous transmit
  - 2 – frequency accuracy
  - 3 – continuous receive
- [in] *rate* – Test rate
- [in] *channel* – Test channel
- [in] *testPower* – Test power
- [in] *timeout* – Timeout

Note: After FCC testing, a Warm Reset is required to bring the unit back to a normal state.

Returns:

- SDCERR_SUCCESS – Successful
- SDCERR_FAIL – Internal error

QueryOID

This function queries an NDIS OID with DeviceIOControl.

```
LONG QueryOID (ULONG ndis_oid, void *buffer, ULONG bufSize)
```

Parameters:

- [in] *ndis_oid* – Indicates the NDIS OID to query
- [out] *buffer* – In/Out
- [in] *bufSize* – In/Out

Returns:

- 0 – Failed (Call GetLastError for error information)
- Non-zero – Success

QueryOID Sample Code

RadioEnable

This function enables the radio.

```
SDCERR RadioEnable()
```

Returns:

- SDCERR_SUCCESS – Successful
- SDCERR_FAIL – Internal error

RadioEnable Sample Code

RadioDisable

This function disables the radio.

```
SDCERR RadioDisable()
```

Returns:

- SDCERR_SUCCESS – Successful
- SDCERR_FAIL – Internal error

RadioDisable Sample Code

Set3rdPartyConfig (Windows only)

This function stores the third party configuration settings.

```
SDCERR Set3rdPartyConfig(SDC3rdPartyConfig *cfg3rd)
```

Parameters:

- [in] *cfg3rd* – Third party configuration

Returns:

- SDCERR_SUCCESS – Successful
- SDCERR_INVALID_CONFIG - *gcfg is NULL or data is invalid
- SDCERR_FAIL – Internal error

Set3rdPartyConfig Sample Code

SetAllConfigs

This function sets all of the configurations (except ThirdPartyConfig) to the given list; all previous configurations are lost. If the active configuration is not ThirdParty, it auto-resets to the first configuration.

```
SDCERR SetAllConfigs(unsigned long num, SDCConfig *cfgs)
```

Parameters:

- [in] *num* – Number of configurations
- [in] *cfgs* - Configurations

Returns:

- SDCERR_SUCCESS – Successful
- SDCERR_FAIL – Internal error or **cfgs* is NULL, or number is 0 or > MAX_CFGS

SetAllConfigs Sample Code

SetDefaultConfigValues

This function sets default values for new configurations.

```
SDCERR SetDefaultConfigValues(SDCConfig *cfg)
```

Parameters:

- [in] *cfg* – Configuration

Returns:

- SDCERR_SUCCESS – Successful
- SDCERR_INVALID_CONFIG – Configuration contains bad parameters
- SDCERR_FAIL – Internal error

SetDefaultConfigValues Sample Code

SetEAPFASTCred

This function sets the EAP-FAST credentials. It validates the configuration's *wepType* and *eapType*.

```
SDCERR SetEAPFASTCred (SDCConfig *cfg, char *username, char *password, char
*pacfilename, char *pacpassword)
```

Parameters:

- [in] *cfg* – Valid configuration (required)
- [in] *username* – Null-terminated username up to USER_NAME_SZ characters. If NULL, then the username field is cleared
- [in] *password* – Null-terminated password up to USER_PWD_SZ characters. If NULL, then the username field is cleared
- [in] *pacfilename* – Null-terminated filename up to CRED_PFILE_SZ characters. If NULL, then the pacfilename field is cleared
- [in] *pacpassword* – Null-terminated password up to CRED_PFILE_SZ characters. If NULL, then the password field is cleared

Returns:

- SDCERR_INVALID_WEP_TYPE – wepType is not WEP_AUTO, WPA_TKIP, WPA2_AES, CCKM_TKIP, or WEP_AUTO_CKIP
- SDCERR_INVALID_EAP_TYPE – eapTYPE is not EAP_EAPFAST
- SDCERR_INVALID_PARAMETER – Invalid parameter
- SDCERR_INVALID_CONFIG – Invalid configuration
- SDCERR_FAIL – Other error

SetEAPFASTCred Sample Code

SetEAPTLSCred

This function sets the EAP-TLS credentials. It validates the configuration's *wepType* and *eapType* members.

```
SDCERR SetEAPTLSCred (SDCConfig *cfg, char *username, char *userCert, CERTLOCATION
certLocation, char *caCert)
```

Parameters:

- [in] *cfg* – Valid configuration (required)
- [in] *username* – Null-terminated username up to USER_NAME_SZ characters. If NULL, then the username field is cleared

- [in] *userCert* – 20-byte hash representing one specific user cert from the MS-store. If NULL, then the user cert field is cleared
- [in] *certLocation* – Specifies where the CA cert if stored. It determines the value of the caCert parameter
- [in] *caCert* – If NULL, this parameter is ignored.

  Depending on the caCertLocation field, caCert contains:

  - CERT_NONE – caCert should be NULL. Do not validate the server
  - CERT_FILE – caCert specifies the cert filename, up to CRED_CERT_SZ characters
  - CERT_FULL_STORE – caCert is NULL. The full MS cert store will be searched for a valid cert
  - CERT_IN_STORE – caCert is a 20-byte hash representing one specific cert from the MS-store

Returns:

- SDCERR_INVALID_WEP_TYPE – wepType is not WEP_AUTO, WPA_TKIP, WPA2_AES, CCKM_TKIP, or WEP_AUTO_CKIP
- SDCERR_INVALID_EAP_TYPE – eapType is not EAP_EAPTLS
- SDCERR_INVALID_PARAMETER – Invalid parameter
- SDCERR_INVALID_CONFIG – Invalid configuration
- SDCERR_FAIL – Other error

SetEAPTLSCred Sample Code

SetEAPTTLSCred

This function sets the EAP-TTLS credentials. It validates the configuration's *webType* and *eapType* members.

```
SDCERR SetEAPTTLSCred (SDCConfig *cfg, char *username, char *password,
CERTLOCATION certLocation, char *caCert)
```

Parameters:

- [in] *cfg* – Valid configuration (required)
- [in] *username* – Null-terminated username up to USER_NAME_SZ characters. If NULL, then the username field is cleared
- [in] *password* – Null-terminated password up to USER_PWD_SZ characters. If NULL, then the username field is cleared
- [in] *certLocation* – Specifies where the CA cert is stored; it determines the value of the caCert parameter
- [in] *caCert* – If NULL, this parameter is ignored

  Depending on the caCertLocation field, caCert will contain:

  - CERT_NONE – caCert is NULL. Do not validate the server
  - CERT_FILE – caCert will specify the cert filename, up to CRED_CERT_SZ characters
  - CERT_FULL_STORE – caCert is NULL. The MS certificate store will be searched for a valid certificate.
  - CERT_IN_STORE – caCert is a 20-byte hash representing one specific cert from the MS-store

Returns:

- SDCERR_INVALID_WEP_TYPE – wepType is not WEP_AUTO, WPA_TKIP, WPA2_AES, CCKM_TKIP, or WEP_AUTO_CKIP
- SDCERR_INVALID_EAP_TYPE – eapType is not EAP_EAPTTLS

- SDCERR_INVALID_PARAMETER – Invalid parameter
- SDCERR_INVALID_CONFIG – Invalid configuration
- SDCERR_FAIL – Other error

SetEAPTTLSCred Sample Code

SetGlobalSettings

This function sets the global configuration settings and restarts the card.

```
SDCERR SetGlobalSettings(SDCGlobalConfig *gcfg)
```

Parameters:

- [in] *gcfg* – Global configuration

Returns:

- **SDCERR_SUCCESS** – Successful
- **SDCERR_INVALID_CONFIG** - **gcfg* is NULL or the data is invalid
- **SDCERR_FAIL** – Internal error

SetGlobalSettings Sample Code

SetLEAPCred

This function sets the LEAP credentials. It validates the configuration's *wepType* and *eapType*.

```
SDCERR SetLEAPCred (SDCConfig *cfg, char *username, char *password)
```

Parameters:

- [in] *cfg* – Valid configuration (required)
- [in] *username* – Null-terminated username up to USER_NAME_SZ characters. If NULL, then the username field is cleared
- [in] *password* – Null-terminated password up to USER_PWD_SZ characters. If NULL, then the username field is cleared

Returns:

- SDCERR_INVALID_WEP_TYPE – wepType is not WEP_AUTO, WPA_TKIP, WPA2_AES, CCKM_TKIP, or WEP_AUTO_CKIP
- SDCERR_INVALID_EAP_TYPE – eapTYPE is not EAP_EAPFAST
- SDCERR_INVALID_PARAMETER – Invalid parameter
- SDCERR_INVALID_CONFIG – Invalid configuration
- SDCERR_FAIL – Other error

SetLEAPCred Sample Code

SetMultipleWEPKeys

This function sets the WEP key information. It validates the config's wepType and eapType members.

```
SDCERR SetMultipleWEPKeys (SDCConfig *cfg, int nTxKey, WEPLEN key1Length, unsigned char *key1, WEPLEN key2Length, unsigned char *key2, WEPLEN key3Length, unsigned char *key3, WEPLEN key4Length, unsigned char *key4)
```

Parameters:

- [in] *cfg* – Valid configuration (required)
- [in] *nTxKey* – Specifies the transmit key (1, 2, 3, or 4)
- [in] *key1Length* – Must be one of three values:
  - WEPLEN_NOT_SET – Clear this key
  - WEPLEN_40BIT – Key must be 10 hex characters
  - WEPLEN_128BIT – Key must be 26 hex characters
- [in] *key1* – The WEP key in hexadecimal must be 0, 10, or 26 hex characters
- [in] *key2Length* – Must be one of three values:
  - WEPLEN_NOT_SET – Clear this key
  - WEPLEN_40BIT – Key must be 10 hex characters
  - WEPLEN_128BIT – Key must be 26 hex characters
- [in] *key2* – The WEP key in hexadecimal must be 0, 10, or 26 hex characters
- [in] *key3Length* – Must be one of three values:
  - WEPLEN_NOT_SET – Clear this key
  - WEPLEN_40BIT – Key must be 10 hex characters
  - WEPLEN_128BIT – Key must be 26 hex characters
- [in] *key3* – The WEP key in hexadecimal must be 0, 10, or 26 hex characters
- [in] *key4Length* – Must be one of three values:
  - WEPLEN_NOT_SET – Clear this key
  - WEPLEN_40BIT – Key must be 10 hex characters
  - WEPLEN_128BIT – Key must be 26 hex characters
- [in] *key4* – The WEP key in hexadecimal must be 0, 10, or 26 hex characters

Returns:

- SDCERR_INVALID_WEP_TYPE – wepType is not WEP_ON or WEP_CKIP
- SDCERR_INVALID_EAP_TYPE – eapType is not EAP_NONE
- SDCERR_INVALID_PARAMETER – Invalid parameter
- SDCERR_INVALID_CONFIG – Invalid configuration
- SDCERR_FAIL – Other error

SetMultipleWEPKeys Sample Code

SetOID

This function sets an NDIS OID with DeviceIOControl.

```
LONG SetOID (ULONG ndis_oid, void *buffer, ULONG bufSize)
```

Parameters:

- [in] *ndis_oid* – The NDIS OID to query
- [in] *buffer* – In/Out
- [in] *bufSize* – In/Out

Returns:

- 0 – Failure. Call GetLastError for error information
- Non-zero – Success

SetOID Sample Code

SetPEAPGTCCred

This function sets the PEAP-GTC credentials. It validates the configuration's wepType and eapType members.

```
SDCERR SetPEAPGTCCred (SDCConfig *cfg, char *username, char *password,
CERTLOCATION CAcertLocation, char *caCert)
```

Parameters:

- [in] *cfg* – Valid configuration (required)
- [in] *username* – Pass in a valid pointer with an allocated buffer of at least USER_NAME_SZ characters. If NULL, this parameter is ignored
- [in] *password* – Pass in a valid pointer with an allocated buffer of at least USER_PWD_SZ characters. If NULL, this parameter is ignored
- [in] *CAcertLocation* – Pass in a valid pointer. If NULL, this parameter is ignored
- [in] *caCert* – Pass in a valid pointer with an allocated buffer of at least CRED_CERT_SZ characters. If NULL, this parameter is ignored

  Depending on the caCertLocation field, caCert will contain:

  - CERT_NONE – caCert is NULL. Do not validate the server
  - CERT_FILE – caCert will specify the certificate filename up to CRED_CERT_SZ characters
  - CERT_FULL_STORE – caCert is NULL. The MS certificate store will be searched for a valid certificate
  - CERT_IN_STORE – caCert is a 20-byte hash representing one specific certificate from the MS-store

Returns:

- SDCERR_INVALID_WEP_TYPE – wepType is not WEP_AUTO, WPA_TKIP, WPA2_AES, CCKM_TKIP, or WEP_AUTO_CKIP
- SDCERR_INVALID_EAP_TYPE – eapTYPE is not EAP_EAPFAST
- SDCERR_INVALID_PARAMETER – Invalid parameter
- SDCERR_INVALID_CONFIG – Invalid configuration
- SDCERR_FAIL – Other error

SetPEAPGTCCred Sample Code

SetPEAPMSCHAPCred

This function sets PEAP-MSCHAP credentials. It validates the configuration's wepType and eapType members.

```
SDCERR SetPEAPMSCHAPCred (SDCConfig *cfg, char *username, char *password,
CERTLOCATION CAcertLocation, char *caCert)
```

Parameters:

- [in] *cfg* – Valid configuration (required)
- [in] *username* – Pass in a valid pointer with an allocated buffer of at least USER_NAME_SZ characters. If NULL, this parameter is ignored
- [in] *password* – Pass in a valid pointer with an allocated buffer of at least USER_PWD_SZ characters. If NULL, this parameter is ignored
- [in] *CAcertLocation* – Pass in a valid pointer. If NULL, this parameter is ignored
- [in] *caCert* – Pass in a valid pointer with an allocated buffer of at least CRED_CERT_SZ characters. If NULL, this parameter is ignored

  Depending on the caCertLocation field, caCert will contain:
  - CERT_NONE – caCert is NULL. Do not validate the server
  - CERT_FILE – caCert will specify the certificate filename up to CRED_CERT_SZ characters
  - CERT_FULL_STORE – caCert is NULL. The MS certificate store will be searched for a valid certificate
  - CERT_IN_STORE – caCert is a 20-byte hash representing one specific certificate from the MS-store

Returns:

- SDCERR_INVALID_WEP_TYPE – wepType is not WEP_AUTO, WPA_TKIP, WPA2_AES, CCKM_TKIP, or WEP_AUTO_CKIP
- SDCERR_INVALID_EAP_TYPE – eapTYPE is not EAP_EAPFAST
- SDCERR_INVALID_PARAMETER – Invalid parameter
- SDCERR_INVALID_CONFIG – Invalid configuration
- SDCERR_FAIL – Other error

SetPEAPMSCHAPCred Sample Code

SetPEAPTLSCred

This function sets the PEAP-TLS credentials. It validates the configuration's wepType and eapType members.

```
SDCERR SetPEAPTLSCred (SDCConfig *cfg, char *username, char *password,
CERTLOCATION certLocation, char *caCert)
```

Parameters:

- [in] *cfg* – Valid configuration (required)
- [in] *username* – Null-terminated username up to USER_NAME_SZ characters. If NULL, then the username field is cleared
- [in] *password* – Pass in a valid pointer with an allocated buffer of at least USER_PWD_SZ characters. If NULL, this parameter is ignored
- [in] *certLocation* – Specifies where the CA cert if stored. It determines the value of the caCert parameter
- [in] *caCert* – If NULL, this parameter is ignored.

  Depending on the caCertLocation field, caCert contains:

  - CERT_NONE – caCert should be NULL. Do not validate the server
  - CERT_FILE – caCert specifies the cert filename, up to CRED_CERT_SZ characters
  - CERT_FULL_STORE – caCert is NULL. The full MS cert store will be searched for a valid cert
  - CERT_IN_STORE – caCert is a 20-byte hash representing one specific cert from the MS-store

Returns:

- SDCERR_INVALID_WEP_TYPE – wepType is not WEP_AUTO, WPA_TKIP, WPA2_AES, CCKM_TKIP, or WEP_AUTO_CKIP
- SDCERR_INVALID_EAP_TYPE – eapType is not EAP_EAPTLS
- SDCERR_INVALID_PARAMETER – Invalid parameter
- SDCERR_INVALID_CONFIG – Invalid configuration
- SDCERR_FAIL – Other error

SetPeapTLSCred Sample Code

SetPSK

This function sets the PSK. It validates the configuration's wepType and eapType.

```
SDCERR SetPSK (SDCConfig *cfg, char *psk)
```

Parameters:

- [in] *cfg* – Valid configuration (required)
- [in] *psk* – Null-terminated psk up to PSK_SZ characters for psk.
  - PSK – Must be 64 hex characters.
  - Passphrase – Must be 8-63 characters (printable ASCII). If NULL, the PSK field is cleared.

Returns:

- SDCERR_INVALID_WEP_TYPE – wepType is not WPA PSK or WPA2 PSK
- SDCERR_INVALID_EAP_TYPE – eapTYPE is not EAP_NONE
- SDCERR_INVALID_PARAMETER – Invalid parameter
- SDCERR_INVALID_CONFIG – Invalid configuration
- SDCERR_FAIL – Other error

SetPSK Sample Code

SetUserCertPassword

This function sets the user certificate password.

```
SDCERR SetUserCertPassword(SDCConfig *cfg, char * userPswd)
```

Parameters:

- [in] cfg a valid configuration
- [in] userPswd pointer to buffer USER_PWD_SZ long

Returns:

- SDCERR_SUCCESS – Successful
- SDCERR_INVALID_PARAMETER if an invalid parameter

- SDCERR_INVALID_CONFIG if an invalid config
- SDCERR_FAIL – Error

SetWAPICertCred

This function sets the WAPI credentials

```
SDCERR SetWAPICertCred(SDCConfig * cfg,  char * username, char* userCert,
CERTLOCATION* certLocation, char* caCert);
```

Parameters:

- [in] cfg--a valid configuration, required
- [out] username--pass in a valid pointer with an allocated buffer of at least USER_NAME_SZ characters. if NULL, this parameter is ignored
- [out] userCert--pass in a valid pointer with an allocated buffer of at least CRED_CERT_SZ characters. if NULL, this parameter is ignored
- [out] CAcertLocation--pass in a valid pointer. if NULL, this parameter is ignored
- [out] caCert--pass in a valid pointer with an allocated buffer of at least CRED_CERT_SZ characters. if NULL, this parameter is ignored;
  depending on the caCertLocation field, caCert will contain:
  - if CERT_NONE, caCert is NULL - don't validate the server
  - if CERT_FILE, caCert will specify the cert filename, up to CRED_CERT_SZ chars

Returns:

- SDCERR_SUCCESS if successful
- SDCERR_INVALIDPARAMETER if an invalid parameter
- SDCERR_INVALID_CONFIG if an invalid config
- SDCERR_FAIL if other err

SetWEPKey

This function sets the WEP key information. It validates the config's wepType and eapType members.

```
SDCERR SetWEPKey (SDCConfig *cfg, int nWepKey, WEPLEN keyLength, unsigned char
*key, BOOLEAN txKey)
```

Parameters:

- [in] *cfg* – Valid configuration (required)
- [in] *nWepKey* – Indicates which of the four stored WEP keys (1, 2, 3, or 4) to modify
- [in] *keyLength* – Must be one of three values:
  - WEPLEN_NOT_SET – Clear this key
  - WEPLEN_40BIT- Key must be 10 hex characters
  - WEPLEN_128BIT – Key must be 26 hex characters
- [in] *key* – If the WEP key (hexadecimal) is NULL, this field will be cleared. Must be 0, 10, or 26 hex characters
- [in] *txKey* – Set if this is the active transmit key (only one of the four keys can be the *txKey*)

Returns:

- SDCERR_INVALID_WEP_TYPE – wepType is not WEP_ON or WEP_CKIP

- SDCERR_INVALID_EAP_TYPE – eapTYPE is not EAP_NONE
- SDCERR_INVALID_PARAMETER – Invalid parameter
- SDCERR_INVALID_CONFIG – Invalid configuration
- SDCERR_FAIL – Other error

SetWEPKey Sample Code

testTxData (Windows only)

This function starts and stops blasting data in FCCTEST mode.

```
SDCERR testTxData(BOOLEAN start, char pattern)
```

Parameters:

- [in] *start* – TRUE starts data; FALSE stops data
- [in] *pattern* – The data in the packet is filled with this value

Returns:

- SDCERR_SUCCESS – Successful
- SDCERR_FAIL – Not in FCC test mode or internal failure

Note: This creates a process that sends out data to IP address FF.FF.FF.FF as quickly as possible; so that it can slow down operation of the system on slower machines. It can be a lengthy call.

---

updateSROM (Windows only)

This function sets the Bluetooth coexistence, regulatory domain, and the maximum Tx percentage (%).

```
SDCERR updateSROM()
```

Returns:

- SDCERR_SUCCESS – Successful
- SDCERR_INVALID_CONFIG – If current global configuration is not valid
- SDCERR_FAIL – Failed to set it

Note: This is a lengthy call. It should not (and need not) be called frequently. The value is stored in SROM and it takes significant time to access it.

Validate_WEP_EAP_Combo

Note: This command is ONLY supported in Linux.

This function verifies the comination of WEP_TYPE and EAP_TYPE is valid.

```
SDCERR Validate_WEP_EAP_Combo(WEPTYPE wt, EAPTYPE et)
```

Parameters:

- [in] *wt* - WEPTYPE wepType
- [in] et – EAPTYPE eapType

Returns:

- SDCERR_SUCCESS – Successful
- SDCERR_INVALID_WEP_TYPE if wepType is invalid
- SDCERR_INVALID_EAP_TYPE if eapType is invalid
- SDCERR_INVALID_PARAMETER if invalid combination

STRUCTURES

CF10G_STATUS

```
typedef struct _CF10G_STATUS {
  CARDSTATE             cardState;
  char                  configName[CONFIG_NAME_SZ];
  UCHAR                 client_MAC[6];
  UCHAR                 client_IP[4];
  char                  clientName[CLIENT_NAME_SZ];
  UCHAR                 AP_MAC[6];
  UCHAR                 AP_IP[4];
  Char                  APName[CLIENT_NAME_SZ];
  EAPTYPE               eapType;
  unsigned long         channel;
  int                   rssi;
  BITRATE               bitRate;
  Int                   txPower;
  unsigned long         driverVersion;
  RADIOTYPE             radioType;
  unsigned long         DTIM;
  unsigned long         eaconPeriod;
  unsigned long         beaconsReceived;
} CF10G_STATUS;
```

Elements:

- CARDSTATE cardState
  - Meaning: Association status
  - Values: CARDSTATE enum
- char configName
  - Meaning: Name of the active configuration profile
  - Length: 32 characters
- UCHAR client_MAC
  - Meaning: Client MAC address
  - Length: 6 byte values
- UCHAR client_IP
  - Meaning: Client IPv4 address
  - Length: 4 byte values
- char clientName
  - Meaning: The name assigned to the Summit radio and the client device that uses it
  - Length: 16 characters
- UCHAR AP_MAC
  - Meaning: MAC address of the access point to which the radio is associated
  - Length: 6 byte values
- UCHAR AP_IP
  - Meaning: IPv4 address of the access point to which the radio is associated
  - Length: 4 byte values

- char APNAME
  - Meaning: Name of the access point to which the radio is associated
  - Length: CLIENT_NAME_SZ
- EAPTYPE eapType
  - Meaning: Indicates the Extensible Authentication Protocol type used for 802.1X authentication to the AP
  - Values: EAPTYPE enum
- unsigned long channel
  - Meaning: Channel of the WLAN connection between the Summit radio and the AP
- int rssi
  - Meaning: Signal strength (RSSI) of the WLAN connection between the Summit radio and the AP, displayed graphically and in dBm
- BITRATE bitRate
  - Meaning: Data rate of the WLAN connection between the Summit radio and the AP
  - Values: BITRATE enum
- int txPower
  - Meaning: Transmit power of the WLAN connection between the Summit radio and the AP
- unsigned long driverVersion
  - Meaning: Driver software version number
- RADIOTYPE radioType
  - Meaning: The bands supported by the current LAIRD radio
  - Length: RADIOTYPE enum
- unsigned long DTIM
  - Meaning: A multiple of the beacon period that specifies how often the beacon contains a delivery traffic indication message (DTIM), which tells power-save client devices that a packet is waiting (e.g. a DTIM interval of 3 means that every third beacon contains a DTIM)
- unsigned long beaconPeriod
  - Meaning: The amount of time between access point beacons in Kilomicroseconds, where one Kμsec equals 1,024 microseconds
- unsigned long beaconsReceived
  - Meaning: The number of beacons received

CRYPT

The structure CRYPT stores secure information that must be encrypted for storage in the registry (such as WEP keys, PSKs, EAP usernames and passwords). It is better to use functions such as SetWEPKey and SetEAPFAStCred rather than modify CRYPTs directly.

SDCConfig

The structure SDCConfig is for a configuration profile.

```
typedef struct _SDCConfig {
  char          configName[CONFIG_NAME_SZ];
  char          SSID[SSID_SZ];
  char          clientName[CLIENT_NAME_SZ];
  int           txPower;
  AUTH          authType;
  EAPTYPE       eapType;
  POWERSAVE     powerSave;
  WEPTYPE       wepType;
  BITRATE       bitRate;
  RADIOMODE     radioMode;
} SDCConfig;
```

Elements:

- char configName
  - Meaning: Name of configuration profile
  - Length: 32 characters
- char SSID
  - Meaning: Service set identifier for the WLAN to which the radio connects
  - Length: 32 characters
- char clientName
  - Meaning: The name assigned to the Summit radio and the client device that uses it
  - Length: 16 characters
- int txPower
  - Meaning: Maximum transmit power in milliwatts (mW)
  - Value: Any integer in the range of 0 to TXPOWER (See Global Settings)
- AUTH authType
  - Meaning: 802.11 authentication type, used when associating to AP
  - Value: AUTH_OPEN, AUTH_SHARED, or AUTH_NETWORK_EAP
- EAPTYPE eapType
  - Meaning: Extensible Authentication Protocol (EAP) type for 802.1X authentication
  - Value: EAP_NONE, EAP_LEAP, EAP_EAPFAST, EAP_PEAPMSCHAP, EAP_PEAPGTC, or EAP_EAPTLS
- POWERSAVE powerSave
  - Meaning: Power Save Protocol (PSP) method
  - Value: POWERSAVE_OFF = 0, POWERSAVE_MAX, POWERSAVE_FAST
- WEPTYPE wepType
  - Meaning: Indicates the WEP type
- BITRATE bitRate
  - Meaning: Indicates the bit rate used by a radio when interacting with a WLAN AP
  - Value: Auto (rate negotiated automatically with AP) or one of the valid BITRATE vales in megabits per second (Mbps): BITRATE enum
- RADIOMODE radioMode
  - Meaning: Use of 802.11a, 802.11g, 802.11b, and 802.11n frequencies and data rates when interacting with an AP, or use of ad hoc mode to associate to a client radio instead of an AP

SDCGlobalConfig

The global settings are stored in the SDCGlobalConfig structure:

```
typedef struct _SDCGlobalConfig {
   unsigned long    fragThreshold;
   unsigned long    RTSThreshold;
   RX_DIV           RxDiversity;
   TX_DIV           TxDiversity;
   ROAM_TRIG        roamTrigger;
   ROAM_DELTA       roamDelta;
   ROAM_PERIOD      roamPeriod;
   PREAMBLE         preamble;
   GSHORTSLOT       g_shortslot;
   BT_COEXIST       BTcoexist;
   PING_PAYLOAD     pingPayload;
   unsigned long    pingTimeout;
   unsigned long    pingDelay;
   unsigned long    radioState;
   unsigned long    displayPasswords;
   unsigned long    reserved for internal use;
   unsigned long    txMax;
   FCC_TEST         FCCtest;
   unsigned long    testChannel;
   BITRATE          testRate;
   TXPOWER          testPower;
   unsigned long    regDomain;
   unsigned long    ledUsed;
   unsigned long    txTestTimeout;
   unsigned long    WMEenabled;
   unsigned long    CCXfeatures;
   char             certPath[MAX_CERT_PATH];
   unsigned long    bLRS;
   unsigned long    avgWindow;
   unsigned long    probeDelay;
   unsigned long    polledIRQ;
   unsigned long    keepAlive;
   unsigned long    trayIcon;
   unsigned long    aggScanTimer;
   unsigned long    authTimeout;
   unsigned long    autoProfile;
   unsigned long    defAdHocMode;
   unsigned long    PMKcaching;
   unsigned long    defAdhocChannel;
   unsigned long    silentRunning;
   unsigned long    scanDFSTime;
   unsigned long    suppInfo;
   unsigned long    uAPSD;
   unsigned long    txMaxA;
   unsigned long    adminFiles;
   unsigned long    DFSchannels;
   unsigned long    interferenceMode;
```

```
unsigned long    authServerType;
unsigned long    TTLSInnerMethod;
unsigned long    aLRS;
unsigned short   roamPeriodms;
unsigned short   Reserved;
unsigned long    Reserved1;
 } SDCGlobalConfig;
```

Elements:

- unsigned long fragThreshold
  - Meaning: If packet size (in bytes) exceeds the threshold, then the packet is fragmented.
  - Value: Any integer in the range of 256 to 2346
- unsigned long RTSThreshold
  - Meaning: The packet size above which RST/CTS is required on link.
  - Value: Any integer in the range of 0 to 2347
- RX_DIV RxDiversity
  - Meaning: Indicates how to handle antenna diversity when receiving data from the access point.
  - Value: RX_DIV enum
- TX_DIV TxDiversity
  - Meaning: Indicates how to handle antenna diversity when receiving data from the access point.
  - Value: TX_DIV
- ROAM_TRIG
  - Meaning: When moving average RSSI from the current AP is weaker than Roam Trigger, the radio performs a roam scan where it probes for an AP with a signal that is at least Roam Delta dBm stronger.
  - Value: -50, -55, -60, -65, -70, -75, -80, -85, -90, or Custom
- ROAM_DELTA
  - Meaning: When Roam Trigger is met, a second AP's signal strength (RSSI) must be Roam Delta dBm stronger than moving average RSSI for current AP before the radio attempts to roam to the second AP.
  - Value: 5, 10, 15, 20, 25, 30, 35, or Custom
- ROAM_PERIOD
  - Meaning: After association or roam scan (with no roam), radio will collect RSSI scan data for Roam Period seconds before considering roaming.
  - Value: 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, or Custom
- PREAMBLE preamble
  - Meaning: no longer in use
- GSHORTSLOT g_shortslot
  - Meaning: no longer in use
- BT_COEXIST BTcoexist
  - Meaning: Bluetooth coexistence control
  - Value: BT_COEXIST enume
- PING_PAYLOAD pingPayload
  - Meaning: Amount of data in bytes to be transmitted on a ping
  - Value: 32, 64, 128, 256, 512, or 1024
- unsigned long pingTimeout
  - Meaning: Amount of time in milliseconds (ms) without a response before the ping request is considered a failure
  - Value: Any integer in the range of 1 to 30000
- unsigned long pingDelay
  - Meaning: Amount in time in milliseconds (ms) between successive ping requests
  - Value: Any integer in the range of 0 to 7200000

- unsigned long radioState
  - Meaning:
  - Value: Enabled = 1, Disabled = 0
- unsigned long displayPasswords
  - Meaning: indicates if passwords should be displayed in the SCU
  - Value: No = 0, Yes = 1
- unsigned long txMax
  - Meaning: Maximum transmission power
  - Value: on Windows: Maximum power out desired * 100; on linux: the Maximum value is in mW.
- FCC_TEST Fcctest (Windows only; unused for Linux)
  - Meaning: Which fcctest is active
  - Value: Off = 0, Tx = 1, Frequency = 2, Rx = 3
- unsigned long testChannel (Windows only; unused for Linux)
  - Meaning: The current channel being tested
  - Value: Any channel in the range of 1 to 14
- BITRATE testRate (Windows only; unused for Linux)
  - Meaning: Test date rate
  - Value: BITRATE enum
- TXPOWER testPower (Windows only; unused for Linux)
  - Meaning: test power level
  - Value: Percentage in the range of 0 to 100
- unsigned long regDomain
  - Meaning: Regulatory domain (for status purposes only)
  - Value: REGDOMAIN enum
- unsigned long ledUsed
  - Meaning: LED used (for mini-module GPIO 0; requires resistor off board to make it work)
  - Value: desired GPIO number
- unsigned long txTestTimeout
  - Meaning: Transmission test timeout in seconds; 60000 (decimal) indicates no timeout.
  - Value: 60000 (decimal) indicates no timeout.
- unsigned long WMEenabled
  - Meaning: Wifi Alliance interoperability, must be turned on for N-rates
  - Value: Enable = 1
- unsigned long CCXfeatures
  - Meaning: CCX features
  - Value: Enable = 1 or CCX radio management and AP control of TX power
- Char certPath
  - Meaning: Used to change the path of the certificate store
  - Value:
- unsigned long bLRS
  - Meaning: bitmask of B channel channels.
  - Value: bit 0 = chan 1, bit 1 = chan 2, etc. 0x3fff or higher value indicates all channels
- unsigned long avgWindow
  - Meaning: RSSI moving average window
  - Value: 2 to 8
- unsigned long probeDelay
  - Meaning: Delay before sending out probes when APs are not located (not configured for WZC)
  - Value: 2 to 60
- unsigned long polledIRQ (Windows only)
  - Meaning: Intended for units that cannot share IRQs successfully
  - Value:irq used when polledIRQ is enabled
- unsigned long keepAlive
  - Meaning: When in CAM mode, indicates how often (in seconds) a null packet gets sent
  - Value: 0 = never

- unsigned long trayIcon
  - Meaning: Enabling the tray icon
  - Value: 1 = Enable
- unsigned long aggScanTimer (Windows only)
  - Meaning: Aggressive scan timer
  - Value: 1 = Enable
- unsigned long authTimeout
  - Meaning: The length (in seconds) of the wait time for an EAP authentication request to succeed or fail
  - Value: Any integer in the range of 3 to 60
- unsigned long autoProfile
  - Meaning: Auto Profile enable/disable
  - Value: 1 = Enable
- unsigned long adHocMode
  - Meaning: ADHOC mode enable/disable
  - Value: 1 = Enable
- unsigned long PMKcaching
  - Meaning: standard, 1 opportunistic key caching enabled
  - Value: 1 = Enabled
- unsigned long defAdhocChannel
  - Meaning: when no beacons found this channel is used
  - Value: unsigned long channel
- unsigned long silentRunning
  - Meaning: enables silent running mode (no active scans unless connected)
  - Value: 1 = Enabled
- unsigned long scanDFSTime
  - Meaning: 20-500 ms, default of 160 ms. Maximum time spent scanning each DFS channel during a scan.
  - Value: unsigned long ms
- unsigned long supplnfo
  - Meaning: Turn on or off other protocols.
  - Value: bit 0 is Summit FIPS on/off; bit 1 is reserved; bit 2 is CA cert date-check enable; bit 3 is pre 2014 WPA1 operation
- unsigned long UAPSD
  - Meaning: bitmask of UAPSD capabilities
  - Value: bit 0 is voice; bit 1 is video; bit 2 is background; bit 3 is best effort
- unsigned long txMaxA
  - Meaning: A radio - to account for high gain antennae.
  - Value: unsigned long %
- unsigned long adminFiles
  - Meaning: allows import/export of settings to file
  - Value: 0 = disabled; 1 = enabled
- unsigned long DFSchannels
  - Meaning: Use DFS channels
  - Value: 1 = enabled, 0 = disabled
- unsigned long interferenceMode; (Windows only)
  - Meaning:
  - Value: 0 off, 1 nonWLAN, 2 WLAN, 3 auto
- unsigned long authServerType
  - Meaning: Type of authentication server radio is authenticating against
  - Value: 0 ACS (type 1), 1 SBR (type 2)
- unsigned long TTLSInnerMethod
  - Meaning: The inner authentication method used by an EAP-TTLS profile
  - Value: 0 auto-EAP

- unsigned long aLRS
  - Meaning: bitmask of enabled a band channels
  - Value: Use LRD_WF_GetaLRSBitmask and LRD_WF_GetaLRSChannels to covert a list channels to/from aLRS bitmask.
- unsigned short roamPeriodms
  - Meaning: Roam period in milliseconds – The amount of time between roam scans. Roam scans occur after the radio has fallen below the roam trigger.
  - Value: 10 - 60000
- unsigned short reserved
  - Meaning: future expansion of the global config
  - Value: n/a
- unsigned long Reserved1
  - Meaning: future expansion of the global config……
  - Value: n/a

SDC3rdPartyConfig (Windows only)

The structure SDC3rdPartyConfig is a subset of the structure SDCConfig, because the special profile ThirdPartyConfig supports only certain configuration elements. Other elements are configured through Windows Zero Config or another application.

```
typedef struct _SDC3rdPartyConfig {
  char                  clientName[CLIENT_NAME_SZ];
  POWERSAVE             powerSave;
  int                   txPower;
  BITRATE               bitRate;
  RADIOMODE             radioMode;
} SDC3rdPartyConfig;
```

Elements:

- Char clientName [CLIENT_NAME_SZ]
  - Meaning: Name of configuration profile
  - Value: 32 Characters
- POWERSAVE powerSave
  - Meaning:power save protocol (PSP) method
  - Value: POWERSAVE_OFF, POWERSAVE_MAX, POWERSAVE_FAST
- Int txPower
  - Meaning: Maximum transmit power in milliwatts (mW)
  - Value: Any integer in the range of 0 to TXPOWER (See Global Settings)

- BITRATE bitRate
  - Meaning: Indicates the bit rate used by a radio when interacting with a WLAN AP
  - Value: Auto (rate negotiated automatically with AP) or one of the valid BITRATE vales in megabits per second (Mbps): BITRATE enum
- RADIOMODE radioMode
  - Meaning: Use of 802.11a, 802.11g, 802.11b, and 802.11n frequencies and data rates when interacting with an AP, or use of ad hoc mode to associate to a client radio instead of an AP

LRD_WF_Pil_Info(linux only)

The structure LRD_WF_Pil_Info is used to identify the customer supplied pil library. The values in this structure are used to reveal information with the LRD_WF_GetPilInfo API call.

```
typedef struct _pil_info {
    uint32_t api_version;
    char * company_name;
    char * version_string;  //optional
    char * serial_number;  // optional
    char * product_id;   // optional
    void * data;  // optional - customer use
} LRD_WF_PilInfo;
```

Elements:

- unit32_t api_version
  - Meaning: The pil API version that is supported by this library
- char * company_name
  - Meaning: pointer to a null terminated string containing the company name of the library creator
- char * version_string
  - Meaning: optional pointer to a null terminated string containing version info (NULL if unused)
- char * serial_number
  - Meaning: optional pointer to a null terminated string containing serial number info (NULL if unused)
- char * product_id
  - Meaning: optional pointer to a null terminated string containing product_id info (NULL if unused)
- void * data
  - Meaning: a void pointer for customer use. Unused by the SDK.

DHCP_LEASE

The DHCP_LEASE structure returns information regarding the current DHCP lease.

```
typedef struct _DHCP_LEASE {
        char interface[20];
        char address[20];
        char subnet_mask[20];
        char routers[100];
        long lease_time;
        int message_type;
        char dns_servers[100];
        char dhcp_server[20];
        char domain_name[200];
        char renew[30];
        char rebind[30];
        char expire[30];
} DHCP_LEASE;
```

Elements:

- char interface[20]
  - Meaning: ethernet device name
- char address[20]
  - Meaning: dotted-quad ip address
- char subnet_mask[20]
  - Meaning: dotted-quad netmask
- char routers[100]
  - Meaning: routing gateways to use, in preferred order - usually just one
- long lease_time
  - Meaning: lease time in seconds until it becomes invalid
- int message_type
  - Meaning: 1-of-8 values as used in negotiation
- char dns_servers[100]
  - Meaning: name servers list
- char dhcp_server[20]
  - Meaning: ip address of the server
- char domain_name[200]
  - Meaning: network domain
- char renew[30]
  - Meaning: calculated date to request renewal = 50% lease time
- char rebind[30]
  - Meaning: calculated date to request a new lease = 87.5% lease time
- char expire[30]
  - Meaning: calculated date of lease expiration = 100% lease time

LRD_WF_COMPONENT_VERSIONS (Windows only)

Structure LRD_WF_COMPONENT_VERSIONS is used to retrieve software name and corresponding version information. Defined for Windows CE/Mobile only.

```
typedef struct _LRD_WF_COMPONENT_VERSIONS{
    char componentName[32];
    char componentVersion[32];
}LRD_WF_COMPONENT_VERSIONS;
```

LRD_WF_SSID

The LRD_WF_SSID structure allows the use of non-string SSIDs (SSIDs that contain NULL or non-printable characters).

```
typedef struct _LRD_WF_SSID{
    unsigned char len;
    unsigned char val[LRD_WF_MAX_SSID_LEN];
                        // Note that the val is not a string and can have embedded\
                        // NULL and non-printable characters.  Also note that val
                        // does not have a null termination character.
} LRD_WF_SSID;
```

Elements:

- Unsigned char len
  - Meaning: the number of characters in the val element (including any NULL characters)
- Unsigned char val[]
  - Meaning: the characters of the SSID. This can include non-printable and NULL characters. Does not include a termination NULL character. DO NOT PRINT AS A STRING.

LRD_WF_SCAN_ITEM_INFO

The LRD_WF_SCAN_ITEM_INFO structure contains the information for a related SSID.

```
typedef struct _LRD_WF_SCAN_INFO_ITEM{
    int           channel;
    int           rssi;
    unsigned int    securityMask;    // bit mask of WEPTYPE enums indicating
                                     // supported types
    LRD_WF_BSSTYPE  bssType;
    unsigned int    reserved;
    unsigned char   bssidMac[LRD_WF_MAC_ADDR_LEN];
    LRD_WF_SSID     ssid;
} LRD_WF_SCAN_ITEM_INFO ;
```

Elements:

- Int channel
  - Meaning: The channel number the SSID is operating
- Int rssi:
  - Meaning: The reported rssi
- Unsigned int securityMask
  - Meaning: A bitmask of all the supported encryption types supported by the current SSID
- LRD_WF_BSSTYPE bssType
  - Meaning: set to either INFRASTRUCTURE or ADHOC
- Unsigned int reserved
  - Meaning – reserved for future use
- Unsigned char bssidMac
  - Meaning: the MAC address for the bssid
- LRD_WF_SSID ssid
  - Meaning – this structure contains the SSID data

LRD_WF_BSSID_LIST

The LRD_WF_BSSID_LIST structure is the containment structure to handle a variable number of LRD_WF_SCAN_ITEM_INFO elements.

```
typedef struct _LRD_WF_BSSID_LIST{
    unsigned long  NumberOfItems;
    LRD_WF_SCAN_ITEM_INFO Bssid[1];
} LRD_WF_BSSID_LIST;
```

ENUMERATION TYPES

AUTH

Description: 802.11 authentication type; used when associating to the AP

Possible Values:

- *AUTH_OPEN:* Open Authentication (Default value)
- *AUTH_SHARED:* Shared Key Authentication
- *AUTH_NETWORK_EAP:* Network EAP or LEAP Authentication BITRATE

Description: The bit rate used by a radio when interacting with a WLAN AP

Possible Values:

- *BITRATE_AUTO:* Bit rate is negotiated automatically with the AP
- *BITRATE_1:* 1 Mbps
- *BITRATE_2:* 2 Mbps
- *BITRATE_5_5:* 5.5 Mbps
- *BITRATE_6:* 6 Mbps
- *BITRATE_9:* 9 Mbps
- *BITRATE_11:* 11 Mbps
- *BITRATE_12:* 12 Mbps
- *BITRATE_18:* 18 Mbps
- *BITRATE_24:* 24 Mbps
- *BITRATE_36:* 36 Mbps
- *BITRATE_48:* 48 Mbps
- *BITRATE_54:* 54 Mbps
- *BITRATE_6_5: 13 Mbps*
- *BITRATE_13: 26 Mbps*
- *BITRATE_19_5: 39 Mbps*
- *BITRATE_26: 52 Mbps*
- *BITRATE_39: 78 Mbps*
- *BITRATE_52: 104 Mbps*
- *BITRATE_58_5: 117 Mbps*
- *BITRATE_65: 130 Mbps*
- *BITRATE_72: 144 Mbps*

BT_COEXIST

Description: Enables or disables Bluetooth coexistence

Possible Values:

- *BT_OFF:* Bluetooth off
- *BT_ON:* Bluetooth on

CARDSTATE

Description:

Possible Values:

- CARDSTATE_NOT_INSERTED
- CARDSTATE_NOT_ASSOCIATED
- CARDSTATE_ASSOCIATED
- CARDSTATE_AUTHENTICATED
- CARDSTATE_FCCTEST
- CARDSTATE_NOT_SDC

CCX_FEATURES

Description: Use of Cisco information element (IE) and CCX version number; support for CCX features

Possible Values:

- *CCX_OPTIMIZED:* Use Cisco IE and CCX version number; support all CCX features except AP-assisted roaming, AP-specified maximum transmit power, and radio management
- *CCX_FULL:* Use Cisco IE and CCX version number; support all CCX features
- *CCX_OFF:* Do not use Cisco IE and CCX version number

CERTLOCATION

Description: Location of the root certificate authority (CA) digital certificate

Possible Values:

- *CERT_NONE* – Don't validate the server
- *CERT_FILE* – Specify the filename for the CA Cert
- *CERT_FULL_STORE* – Use the entire MS-store
- *CERT_IN_STORE* – Use one specific cert from the MS-store; specify the cert's hash

EAPTYPE

Description: Indicates Extensible Authentication Protocol (EAP) type used for 802.1X authentication to the AP

Possible Values:

- *EAP_NONE:* No EAP type (default)
- *EAP_LEAP*
- *EAP_EAPFAST*
- *EAP_PEAPMSCHAP*
- *EAP_PEAPGTC*
- *EAP_EAPTLS*
- *EAP_EAPTTLS*
- *EAP_PEAPTLS*
- *EAP_WAPI_CERT*

FCCTEST

Description:

Possible Values:

- *FCCTEST_OFF*
- *FCCTEST_TX*
- *FCCTEST_RX*
- *FCCTEST_FREQ*

GSHORTSLOT

Description:

Possible Values:

- *GSHORT_AUTO*
- *GSHORT_OFF*
- *GSHORT_ON*

INTERFERENCE

Description:

Possible Values:

- INTER_NONE – Off
- INTER_NONWLAN – Reduces CCA Tx threshold
- INTER_WLAN – Reduces interchannel noise
- INTER_AUTO – Automatic

LRD_WF_BSSTYPE

Description: SSID types

Possible values:

- INFRASTRUCTURE
- ADHOC

PING_PAYLOAD

Description: Amount of data in bytes to be transmitted on a ping

Possible Values:

- *PP_32*: 32 bytes of data (default)
- *PP_64*: 64 bytes of data
- *PP_128*: 128 bytes of data
- *PP_256*: 256 bytes of data
- *PP_512*: 512 bytes of data
- *PP_1024*: 1024 bytes of data

POWERSAVE

Description: The radio's power save mode

Possible Values:

- *POWERSAVE_OFF*: Constantly Awake Mode (CAM)
- *POWERSAVE_MAX*: Maximum power savings
- *POWERSAVE_FAST*: Fast power save mode (Default)

PREAMBLE

Description:

Possible Values:

- *PRE_AUTO*
- *PRE_SHORT*

RADIOMODE

Description: Use of 802.11a, 802.11g, 802.11b, and 802.11n frequencies and data rates when interacting with AP, or use of ad hoc mode to associate to a client radio instead of an AP.

Possible Values:

- *RADIOMODE_B_ONLY:* 1, 2, 5.5, and 11 Mbps
- *RADIOMODE_BG:* All B and G rates (Default for B/G radios)
- *RADIOMODE_G_ONLY:* 6, 9, 12, 18, 24, 36, 48, and 54 Mbps
- *RADIOMODE_BG_LRS*
- *RADIOMODE_A_ONLY:* 6, 9, 12, 18, 24, 36, 48, and 54 Mbps
- *RADIOMODE_ABG:* All A rates and all B and G rates, with A rates preferred (Default for A/B/G radios)
- *RADIOMODE_BGA:* All B and G rates and all A rates, with B and G rates preferred
- *RADIOMODE_ADHOC:* Rates optimized - 1, 2, 5.5, 6, 11, 24, 36, and 54 Mbps.
- RADIOMODE_GN: All G and N rates
- RADIOMODE_AN: All A and N rates
- RADIOMODE_ABGN: All A,B,G, and N rates with A rates preferred
- RADIOMODE_BGAN: All B,G,A, and N rates with B/G rates preferred
- RADIOMODE_BGN: All B,G and N rates

Note: If the administrator selects Ad Hoc for radio mode, then the Summit radio uses ad hoc mode instead of infrastructure mode. In infrastructure mode, the radio associates to an AP. In ad hoc mode, the radio associates to another client radio that is in ad hoc mode and has the same SSID and, if configured, static WEP key.

RADIOTYPE

Description: Radio type of the device

Possible Values:

- *RADIOTYPE_BG:* Summit 802.11g radio (supports 802.11b and 802.11g)
- *RADIOTYPE_ABG:* Summit 802.11a/g radio (supports 802.11a, 802.11b, and 802.11g)

- *RADIOTYPE_NBG:* Summit 802.11nb/g radio (802.11b, and 802.11g and 802.11n)
- *RADIOTYPE_NABG:* Summit 802.11n radio (supports 802.11a, 802.11b, 802.11g, and 802.11n)
- *RADIOTYPE_NOT_SDC:* Not a Summit SDC radio
- *RADIOTYPE_NOT_SDC_1:* Reserved

REGDOMAIN

Description: Indicates the regulatory domain(s) for which the radio is configured. The domain(s) cannot be configured by an administrator or user.

Possible Values:

- *REG_FCC:* Federal Communications Commission; the regulatory agency and standards body for the Americas and parts of Asia
- *REG_ETSI:* European Telecommunications Standards Institute; the standards body applicable to most Europe, Africa, the Middle East, and parts of Asia
- *REG_TELEC:* Telecom Engineering Center; the standards body for Japan
- *REG_WW:* Worldwide domain; enables the radio to be used in any domain
- *REG_KCC:* Korea

Note: The following domains can only be returned with the REG_DOMAIN function in the Linux SDK.

- *REG_CA: CA country code used*
- *REG_FR: FR country code used*
- *REG_GB: GB country code used*
- *REG_AU: AU country code used*
- *REG_NZ: NZ country code used*
- *REG_CN: CN country code used*

ROAM_DELTA

Description: When Roam Trigger is met, a second AP's signal strength (RSSI) must be Roam Delta dBm stronger than moving average RSSI for current AP before radio will attempt to roam to the second AP.

Possible Values:

- RDELTA_5: 5 dBm
- RDELTA_10: 10 dBm
- RDELTA_15: 15 dBm (Default)
- RDELTA_20: 20 dBm
- RDELTA_25: 25 dBm
- RDELTA_30: 30 dBm
- RDELTA_35: 35 dBm

ROAM_PERIOD

Description: After association or roam scan (with no roam), radio will collect RSSI scan data for Roam Period seconds before considering roaming.

Possible Values:

- RPERIOD_5 – 5ms
- RPERIOD_10 – 10ms (Default)
- RPERIOD_15 – 15ms
- RPERIOD_20 – 20ms
- RPERIOD_25 – 25ms
- RPERIOD_30 – 30ms
- RPERIOD_35 – 35ms
- RPERIOD_40 – 40ms
- RPERIOD_45 – 45ms
- RPERIOD_50 – 50ms
- RPERIOD_55 – 55ms
- RPERIOD_60 – 60ms

ROAM_TRIG

Description: When moving average RSSI from the current AP is weaker than Roam Trigger, the radio performs a roam scan where it probes for an AP with a signal that is at least Roam Delta dBm stronger.

Possible Values:

- RTRIG_50: -50 dBm
- RTRIG_55: -55 dBm
- RTRIG_60: -60 dBm
- RTRIG_65: -65 dBm
- RTRIG_70: -70 dBm (Default)
- RTRIG_75: -75 dBm
- RTRIG_80: -80 dBm
- RTRIG_85: -85 dBm
- RTRIG_90: -90 dBm

RX_DIV

Description: Method for handling antenna diversity when receiving data from the AP

Possible Values:

- *RXDIV_MAIN:* Use the main antenna only
- *RXDIV_AUX:* Use the auxiliary antenna only
- *RXDIV_START_AUX:* On startup, use the auxiliary antenna
- *RXDIV_START_MAIN:* On startup, use the main antenna (Default)

SDCERR

Description:

Possible Values:

- SDCERR_SUCCESS
- SDCERR_FAIL
- SDCERR_INVALID_NAME

- SDCERR_INVALID_CONFIG
- SDCERR_INVALID_DELETE
- SDCERR_POWERCYCLE_REQUIRED
- SDCERR_INVALID_PARAMETER
- SDCERR_INVALID_EAP_TYPE
- SDCERR_INVALID_WEP_TYPE
- SDCERR_INVALID_FILE
- SDCERR_INSUFFICIENT_MEMORY,
- SDCERR_NOT_IMPLEMENTED,
- SDCERR_NO_HARDWARE
- SDCERR_INVALID_VALUE

TTLS_INNER_METHOD

Description: Authentication method used within the secure tunnel created by EAP-TTLS

Possible Values:

- *TTLS_AUTO* – Uses any available EAP method (Default)
- *TTLS_MSCHAPV2*
- *TTLS_MSCHAP*
- *TTLS_PAP*
- *TTLS_CHAP*
- *TTLS_EAP_MSCHAPV2*

TX_DIV

Description: Method of handling antenna diversity when transmitting data to the AP

Possible Values:

- *TXDIV_MAIN:* Use main antenna only
- *TXDIV_AUX:* Use auxiliary antenna only
- *TXDIV_ON:* Use diversity (Default)

Note: To enable diversity (for MSD30AG and SSD30AG radio modules), set Tx Diversity to On. To disable diversity, set Tx Diversity to Main Only. You must power-cycle for these changes to take effect.

TXPOWER

Description: Indicates transmit power.

Possible Values:

- *TXPOWER_MAX:* Maximum defined for the current regulatory domain (Default)
- *TXPOWER_1:* 1 mW
- *TXPOWER_5:* 5 mW
- *TXPOWER_10:* 10 mW
- *TXPOWER_20:* 20 mW
- *TXPOWER_30:* 30 mW
- *TXPOWER_50:* 50 mW

WEPLEN

Description: WEP encryption

Possible Values:

- WEPLEN_NOT_SET:
- WEPLEN_40BIT: 40-bit static keys
- WEPLEN_128BIT: 128-bit static keys

WEPTYPE

Description: Type of encryption (and decryption) used to protect transmitted data

Possible Values:

- *WEP_OFF:* No encryption
- *WEP_ON:* WEP with up to four static keys (40-bit or 128-bit in ASCII or hex) defined under WEP/PSK keys
- *WEP_AUTO*
- *WEP_PSK*
- *WEP_TKIP*
- *WEP_AES*
- *CCKM_TKIP*
- *WEP_CKIP*
- *WEP_AUTO_CKIP*
- *CCKM_AES*
- *WPA_PSK_AES*
- *WPA_AES*
- *WPA2_PSK_TKIP*
- *WPA2_TKIP*
- *WAPI_PSK*
- *WAPI_CERT*

PLATFORM INDEPENDENT LAYER (PIL) (LINUX ONLY)

Structures

pil_info

Synopsis: structure contains string pointers and data the SDK can use to display during debug. In addition, the API to retrieve this structure is exposed in the SDK API for customer use. The API LRD_WI_PIL_Init() (customer created) can be used to initialize this structure.

Elements:

```
uint32_t api_version:   should always return PIL_API
char * company_name:    printable string - company name
char * version_string:  printable string - version of this library
char * serial_number:   printable string customer can use this to identify their
                        hardware
char * product_id:      printable string. Customer can use this to provide a
                        product_id
void * data:            customer use
*/
extern LRD_WF_PilInfo pil_info;
```

Functions

LRD_WF_PIL_Init

Description: Allows the initialization of any data that the library may need

```
SDCERR LRD_WF_PIL_Init();
```

Returns:

- SDCERR_SUCCESS - the pil initialized successfully
- SDCERR_FAIL or any Failure value - The SDK will not load any additional functions from the PIL

LRD_WF_PIL_Deinit

Description: allows the cleanup of anything from the PIL initialization

```
SDCERR LRD_WF_PIL_Deinit();
```

Returns:

- SDCERR_SUCCESS - successful
- SDCERR_FAIL - the SDK will report the error SDCERR LRD_WF_PIL_Deinit();

LRD_WF_PIL_GetRegDomain

Description: returns the value for the desired regDomain

```
SDCERR LRD_WF_PIL_GetRegDomain( REG_DOMAIN * regDomain );
```

Parameters:

- [in] regDomain - will contain the value from sdc_sdk.h representing the desired regulatory domain. Must be valid if SDCERR_SUCCESS is returned.

Returns:

- SDCERR_SUCCESS - *regDomain has the desired regulatory domain
- SDCERR_INVALID_PARAMETER - regDomain is NULL

LRD_WF_PIL_SetRegDomain

Description: sets the value for the desired regDomain

```
SDCERR LRD_WF_PIL_SetRegDomain( REG_DOMAIN regDomain );
```

Parameters:

- [in] regDomain - the value from sdc_sdk.h representing the desired regulatory domain.

Returns:

- SDCERR_SUCCESS - regDomain is the desired regulatory domain
- SDCERR_INVALID_PARAMETER - regDomain is not valid.

LRD_WF_PIL_GetDHCPLease

Description: Returns the current DHCP lease information for the wi-fi interface

SDCERR LRD_WF_PIL_GetDHCPLease(DHCP_LEASE *dhcpLease);

Parameters:

- dhcpLease - will contain the structure filled with the current DHCP lease.

Return values:

- SDCERR_SUCCESS - *dhcpLease has the current lease info
- SDCERR_INVALID_PARAMETER - dhcplease is NULL
- SDCERR_FAIL - unable to find current lease

EVENTS

Functions

SDCRegisterForEvents

This function registers the events that the user wants to be notified of SDCERR

```
SDCRegisterForEvents(unsigned long long eventMask, SDC_EVENT_HANDLER ehandler);
```

Parameters:

- [in] eventMask - 64 bit bitmask of events to signal
- [in[ehandler - user defined function to be called on each event

Returns:

- SDCERR_SUCCESS if successful
- SDCERR_INVALID_PARAMETER if invalid parameter
- SDCERR_INVALID_CONFIG if attempting to call again without calling SDCRegisterForEvents()
- SDCERR_FAIL if internal err

Note: If a LOST_COM status from a SDC_E_INTERNAL event is received and the program wants to attempt to recover, SDCDeregisterEvents() must be called before attempting to call SDCRegisterForEvents() again.

SDCRegisteredEventsList

This function returns the current registered event mask.

```
SDCERR SDCRegisteredEventsList( unsigned long long *currentMask);
```

Parameters:

- [out[currentMask -unsigned long pointer for currentMask

Returns:

- SDCERR_SUCCESS if successful
- SDCERR_INVALID_PARAMETER if invalid parameter
- SDCERR_FAIL if internal err

SDCDeregisterEvents

This function deregisters the events handler and stops the SDK event notification.

```
SDCERR SDCDeregisterEvents();
```

Returns:

- SDCERR_SUCCESS if successful
- SDCERR_FAIL if internal err

Structures

sdc_ether_addr

The structure sdc_ether_addr contains a mac address.

```
typedef struct  _sdc_ether_addr {
    unsigned char octet[SDC_ETHER_ADDR_LEN];
} sdc_ether_addr;
```

SDC_EVENT

The structure SDC_EVENT contains information about each SDK event.

```
typedef struct _SDC_EVENT
{
    unsigned int event_type;
    unsigned int status;
    unsigned int reason;
    unsigned int auth_type;
    struct _sdc_ether_addr addr;
    unsigned short flags;
} SDC_EVENT;
```

Elements:

- unsigned int event_type
  - Defined by SDC_EVENT
- unsigned int status
  - See each SDC_EVENT type for what enum defines this field.
- unsigned reason
  - See each SDC_EVENTS type for what enum defines this field.
- unsigned int auth_type
  - For Broadcom auth events, a non zero value will indicate shared key while a 0 will indicate open key.
  - For Atheros auth events see AUTH enum struct
- _sdc_ether_addr addr;
  - mac address of AP currently connected to.
- unsigned short flags
  - Currently unused

Enumerated Types

SDC_EVENTS

The following table (Table 1) displays SDC_EVENTS events that are supported by the 45 series (SDC_EVENTS).

Note: Events not included in the following list are not supported by the 45 series.

Table 1: WB45NBT Events

Event	Description
SDC_E_CONNECTION_STATE	A change to the WiFi's connection state has occurred. See: ▪ LRD_WF_EvtConStatus for status field ▪ LRD_WF_EvtAuthReason or SDC_ATH_DISCONNECT_REASON for reason field ▪ 802.11 reason codes for auth_type field.
SDC_E_DHCP	Indicates a DHCP event has occurred. Note that on systems using the SD45 see section on dhcp_injector and reason code to implement this event. See: ▪ LRD_WF_EvtDHCPStatus for status field ▪ LRD_WF_EvtDHCPReason for reason field
SDC_E_READY	Indicates the wireless device is ready. Sent once after a power on or reset and after firmware recovery
SDC_E_CONNECT_REQ	Indicates a request to connect to a network.
SDC_E_RECONNECT_REQ	Indicates a request to reconnect to a network to which the device was previously connected.
SDC_E_DISCONNECT_REQ	Indicates a request to disconnect from a network.
SDC_E_ASSOC	Indicates that a connection to a network has occurred.
SDC_E_AUTH	Indicates that the authentication state has changed. See: ▪ LRD_WF_EvtAuthStatus for status field ▪ LRD_WF_EvtAuthReason for reason field
SDC_E_DISASSOC	Indicates that the device has lost connectivity to a network or failed to associate. See: ▪ SDC_ATH_DISCONNECT_REASON for status field ▪ 802.11 reason codes for reason field.
SDC_E_ROAM	Indicates a roam has occurred.
SDC_E_SCAN_REQ	Indicates a request to initiate a scan from the host.
SDC_E_SCAN	Indicates a host-initiated scan is complete. Check status field for scan success or failure.
SDC_E_REGDOMAIN	Indicates the firmware's regulatory domain has changed.
SDC_E_CMDERROR	Indicates the firmware has reported an error. See: ▪ SDC_ATH_CMDERROR_REASON for reason field.
SDC_E_INTERNAL	Indicates a status update or error from within the SDK events. See: ▪ LRD_WF_EvtIntStatus for status field ▪ LRD_WF_EvtIntReason for reason field.

Event	Description
SDC_E_FW_ERROR	Indicates a firmware crash has occurred. If recovery is enabled, the event to indicate the firmware has been recovered is SDC_E_READY. See: ▪ LRD_WF_EvtFwErrorReason for reason field
SDC_E_AP_STA_CONNECTED	While in AP mode, indicates that a client has connected.
SDC_E_AP_STA_DISCONNECTED	While in AP mode, indicates that a client has disconnected.

SDC_ATH_DISCONNECT_REASON

The following table (Table 2) describes applicable SDC_E_DISCONNECT Reasons (SDC_ATH_DISCONNECT_REASON).

Table 2: SDC_E_DISCONNECT Reasons

Reason	Description
DISCON_REASON_UNSPEC	No reason specified.
NO_NETWORK_AVAIL	Unable to find or establish a connection to the desired network.
LOST_LINK	Missed too many beacons.
DISCONNECT_CMD	A Disconnect request was processed.
BSS_DISCONNECTED	The device is on an AP blacklist (mac block) or not on the AP whitelist, the AP is too busy to accept connections, or too many encryption errors have occurred.
AUTH_FAILED	Not used.
ASSOC_FAILED	Not used.
NO_RESOURCES_AVAIL	The firmware is out of memory.
CSERV_DISCONNECT	The firmware has decided to disconnect from network. This can occur from host-influenced settings such as marking an AP as 'bad' or because there have been too many decryption errors. If in Ad-Hoc mode, the firmware does not see the other client.
INVALID_PROFILE	The host has sent a bad BSSID.
DOT11H_CHANNEL_SWITCH	The AP sent a DOT11H CSA IE (802.11h Channel Switch Announcement.
PROFILE_MISMATCH	Occurs if the device is in ad-hoc mode and powersave is enabled.
CONNECTION_EVICTED	Not used.
IBSS_MERGE	The station has merged with another IBSS.

SDC_ATH_CMDERROR_REASON

The following table (Table 3) describes applicable SDC_E_CMDERROR Reasons (SDC_ATH_CMDERROR_REASON).

Table 3: SDC_E_CMDERROR Reasons

Reason	Description
INVALID_PARAM	An invalid parameter was sent to the firmware.
ILLEGAL_STATE	The firmware is in an illegal state.
INTERNAL_ERROR	An internal error has occurred in the firmware.

LRD_WF_EvtConStatus

The following table (Table 4) describes applicable LRD_WF_EvtConStatus Reasons.

Table 4: LRD_WF_EvtConStatus Reasons

Reason	Description
CON_STATUS_UNSPEC	The status is unknown.
NOT_CONNECTED	The device is not currently connected.
ASSOCIATING	The device is associating to the network.
ASSOCIATED	The device is associated to the network.
ASSOC_ERROR	There was an error while associating. See: ▪ SDC_ATH_DISCONNECT_REASON for reason field ▪ 802.11 reason codes for auth_type field
AUTHENTICATING	The device is authenticating.
AUTHENTICATED	The device is authenticated.
AUTH_ERROR	There was an error while authenticating. See: ▪ LRD_WF_EvtAuthReason for reason field

LRD_WF_EvtAuthStatus

The following table (Table 5) describes applicable LRD_WF_EvtAuthStatus Reasons.

Table 5: LRD_WF_EvtAuthStatus Reasons

Reason	Description
AUTH_STATUS_UNSPEC	Status not specified.

AUTH_STARTED	Authentication started.
AUTH_SUCCESS	Authentication succeeded.
AUTH_FAILURE	Authentication failed. See: ▪ LRD_WF_EvtAuthReason for reason field

LRD_WF_EvtAuthReason

The following table (Table 6) describes applicable LRD_WF_EvtAuthReason Reasons.

Table 6: LRD_WF_EvtAuthReason Reasons

Reason	Description
AUTH_REASON_UNSPEC	The reason is unspecified.
AUTH_SERVER_NO_RESP	Indicates that there was no response from the RADIUS server. This can indicate the RADIUS server did not respond, the connection is very poor, or the connection was too short to receive a response.
INVALID_CREDENTIALS	Indicates that the credentials are invalid.
METHOD_NOT_SUPPORTED	Indicates that the authentication method is not supported by the RADIUS server.
INVALID_CERT_PASS	Indicates that the certificate password is invalid.
FOUR_WAY_HAND_SHAKE_FAILURE	Indicates that the four way handshake failed.

LRD_WF_EvtDHCPStatus

The following table (Table 7) describes applicable LRD_WF_EvtDHCPStatus Reasons.

Table 7: LRD_WF_EvtDHCPStatus Reasons

Reason	Description
DHCP_STATUS_UNSPEC	Indicates that the status is not specified.
DECONFIG	The DHCP has requested that the interface configuration be removed.
REQUESTING	Indicates that the Disover was sent and the DHCPOFFER replay was received.
RENEWING	Indicates that half of the lease was passed or that the station has reconnected to the network and wants to renew. A unicast renew request is being sent.
RENEWED	Indicates that the lease has renewed. See: ▪ LRD_WF_EvtDHCPReason for reason field.

Reason	Description
REBINDING	Indicates that the renew requests were not answered and a broadcast renew is being sent.
BOUND	Indicates that a select/renew was sent and a DHCPACK reply was received. The interface will be configured with lease. See: ▪ LRD_WF_EvtDHCPReason for reason field.
NAK	Indicates that Nak was received from the server.
LEASEFAIL	Indicates that the DHCP client has failed to obtain a lease.
RELEASED	Indicates that the DHCP client has sent a release.

LRD_WF_EvtDHCPReason

The following table (Table 8) describes applicable LRD_WF_EvtDHCPReason Reasons.

Table 8: LRD_WF_EvtDHCPReason Reasons

Reason	Description
DHCP_REASON_UNSPEC	The reason is not specified.
IP_ADDRESS_SAME	Indicates that the IP address is the same as the previous lease.
IP_ADDRESS_DIFFERENT	Indicates that the IP address is different from the previous lease.

LRD_WF_EvtIntStatus

The following table (Table 8) describes applicable LRD_WF_EvtIntStatus Reasons.

Table 9: LRD_WF_EvtIntStatus Reasons

Reason	Description
INT_STATUS_UNSPEC	Status is not specified.
LOST_COM_DRV	Lost communication with the driver.
LOST_COM_KERN	Lost communication with the kernel.
LOST_COM_SUPP	Lost communication with the supplicant.
LOST_COM_INJ	Lost communication with injected events.

LRD_WF_EvtIntReason

The following table (Table 10) describes applicable LRD_WF_EvtIntReason Reasons.

Table 10: LRD_WF_EvtIntReason Reasons

Reason	Description
INT_REASON_UNSPEC	Reason is not specified.
COM_EXITED	Lost communication due to the other side exiting.
COM_ERROR	Lost communication due to error.

LRD_WF_EvtFwErrorReason

The following table (Tables 11) describes applicable LRD_WF_EvtFwErrorReason Reasons.

Table 11: LRD_WF_EvtFwErrorReason Reasons

Reason	Description
FW_ASSERT	Firmware asserted.
FW_HB_RESP_FAILURE	Firmware did not respond to enough heartbeats.
FW_EP_FULL	Firmware stopped servicing firmware commands.

Defines

802.11 Reason Codes

The following table (Table 12) describes applicable 802.11 Reason codes.

Note: For codes not listed in Table 12, refer to the 802.11 specification.

Table12: 802.11 Reasons codes

Reason	Code	Description
DOT11_RC_RESERVED	0	Reserved
DOT11_RC_UNSPECIFIED	1	Indicates an unspecified reason.
DOT11_RC_AUTH_INVAL	2	Indicates that the previous authentication is no longer valid.
DOT11_RC_DEAUTH_LEAVING	3	Indicates a deauthentication because the sending station is

Reason	Code	Description
		leaving (or has left) IBSS or ESS.
DOT11_RC_INACTIVITY	4	Indicates a disassociation due to inactivity.
DOT11_RC_BUSY	5	Indicates a disassociation because the AP is unable to handle all currently associated stations.
DOT11_RC_INVAL_CLASS_2	6	Indicates that a Class 2 frame was received from a nonauthenticated station.
DOT11_RC_INVAL_CLASS_3	7	Indicates that a Class 3 frame was received from a nonauthenticated station.
DOT11_RC_DISASSOC_LEAVING	8	Indicates a disassociation because the sending station is leaving (or has left) BSS.
DOT11_RC_NOT_AUTH	9	Indicates that the station that is requesting (re)association is not authenticated with the responding station.
DOT11_RC_BAD_PC	10	Indicates an unacceptable power capability element.
DOT11_RC_BAD_CHANNELS	11	Indicates an unacceptable supported channels element.
DOT11_RC_BSS_TRANSIT_MGMT	12	Indicates a disassociation due to BSS Transition Management.
DOT11_RC_INVALID_WPA_IE	13	Indicates an invalid info. element.
DOT11_RC_MIC_FAILURE	14	Indicates a Michael failure.
DOT11_RC_4WH_TIMEOUT	15	Indicates a four-way handshake timeout.
DOT11_RC_GTK_UPDATE_TIMEOUT	16	Indicates a group key update timeout.
DOT11_RC_WPA_IE_MISMATCH	17	Indicates that a WPA IE in a four-way handshake differs from a (re)association request/probe response.
DOT11_RC_INVALID_MC_CIPHER	18	Indicates an invalid multicast cipher.
DOT11_RC_INVALID_UC_CIPHER	19	Indicates an invalid unicast cipher.
DOT11_RC_INVALID_AKMP	20	Indicates an invalid authenticated key management protocol.
DOT11_RC_BAD_WPA_VERSION	21	Indicates an unsupported WPA version.
DOT11_RC_INVALID_WPA_CAP	22	Indicates invalid WPA IE capabilities.
DOT11_RC_8021X_AUTH_FAIL	23	Indicates an 802.1X authentication failure.
DOT11_RC_UNSPECIFIED_QOS	32	Indicates an unspecified QoS-related reason.
DOT11_RC_INSUFFICIENT_BW	33	Indicates that the QoS AP lacks sufficient bandwidth for this QoS station.

Reason	Code	Description
DOT11_RC_EXCESSIVE_FRAMES	34	Indicates that excessive frames need to be acknowledged due to AP transmissions or poor channel conditions.
DOT11_RC_TX_OUTSIDE_TXOP	35	Indicates that the station is transmitting outside the limits of its TXOPs.
DOT11_RC_LEAVING_QBSS	36	Indicates a request from the peer station as the station is leaving the BSS (or resetting).
DOT11_RC_BAD_MECHANISM	37	Indicates a request from the peer station that it does not want to use the mechanism.
DOT11_RC_SETUP_NEEDED	38	Indicates a request from the peer station that the station received frames using the mechanism that require setup.
DOT11_RC_TIMEOUT	39	Indicates a request from the peer station that there was a timeout.

SAMPLE CODE

ActivateConfig Sample Code

```
SDCERR sdcErr;

//Use the name of a valid config
  sdcErr = ActivateConfig("config 1");

if (sdcErr == SDCERR_SUCCESS)
    AfxMessageBox(_T("Activate Config OK"));
else
        AfxMessageBox(_T("Activate Config FAILED"));
```

Function: ActivateConfig

AddConfig Sample Code

```
SDCConfig config;
SDCERR sdcErr;

memset(&config, 0, sizeof(SDCConfig));

// Setting Defaults
sprintf(config.configName,"config 1");
sprintf(config.SSID,"Summit1");
sprintf(config.clientName,"client 1");
config.txPower = TXPOWER_MAX;
config.authType = AUTH_OPEN;
config.eapType = EAP_NONE;
config.powerSave = POWERSAVE_FAST;
config.wepType = WEP_OFF;
config.bitRate = BITRATE_AUTO;
config.radioMode = RADIOMODE_BG;

sdcErr = AddConfig(&config);

if (sdcErr == SDCERR_SUCCESS)
    AfxMessageBox(_T("Added Config OK"));
else
     AfxMessageBox(_T("Added Config FAILED"));
```

Function: AddConfig

CreateConfig Sample Code

```
SDCConfig config;
SDCERR sdcErr;

memset(&config, 0, sizeof(SDCConfig));

sdcErr = CreateConfig(&config);

if (sdcErr == SDCERR_SUCCESS)
    AfxMessageBox(_T("Created Config OK"));
else
     AfxMessageBox(_T("Create Config FAILED"));

// Don't forget to add the config
sdcErr = AddConfig(&config);

if (sdcErr == SDCERR_SUCCESS)
    AfxMessageBox(_T("Added Config OK"));
else
     AfxMessageBox(_T("Added Config FAILED"));
```

Function: CreateConfig

DeleteConfig Sample Code

```
SDCERR sdcErr;

//Can't be the name of the active config
     sdcErr = DeleteConfig("Config 1");

if (sdcErr == SDCERR_SUCCESS)
    AfxMessageBox(_T("Deleted Config 1"));
else
     AfxMessageBox(_T("Delete Config FAILED"));
```

Function: DeleteConfig

exportSettings Sample Code

```
SDC_ALL all;
SDCConfig configs[MAX_CFGS];
SDCGlobalConfig globalConfig;
SDC3rdPartyConfig thirdPartyConfig;
unsigned long numberOfConfigs;
SDCERR sdcErr;

//Get config structures
GetGlobalSettings(&globalConfig);
GetAllConfigs(&configs, &numberOfConfigs);
Get3rdPartyConfig(&thirdPartyConfig);

//Load the configs into the SDC_ALL struct
all.configGlobal = &globalConfig;
all.configs = &configs;
all.configThirdParty = &thirdPartyConfig;
all.numConfigs = numberOfConfigs;

//export to "summit.sdc"
sdcErr = exportSettings("summit.sdc",&all);

if (sdcErr == SDCERR_SUCCESS)
AfxMessageBox(_T("Exported"));

configGlobal.adminOverride = 0;
```

Function: exportSettings

FlushAllConfigKeys Sample Code

```
SDCERR sdcErr;

//Flush all Summit config keys
      sdcErr = FlushAllConfigKeys();

if (sdcErr == SDCERR_SUCCESS)
    AfxMessageBox(_T("Config Keys Flushed"));
else
      AfxMessageBox(_T("Failed"));
```

Function: FlushAllConfigKeys

FlushConfigKeys Sample Code

```
SDCERR sdcErr;

//Flushes specified config number
      sdcErr = FlushConfigKeys(1);

if (sdcErr == SDCERR_SUCCESS)
    AfxMessageBox(_T("Config Flushed"));
else
      AfxMessageBox(_T("Failed"));
```

Function: FlushConfigKeys

Get3rdPartyConfig Sample Code

```
SDC3rdPartyConfig config3;
SDCERR sdcErr;

memset(&config3, 0, sizeof(SDC3rdPartyConfig));

sdcErr = Get3rdPartyConfig(&config3);

if (sdcErr == SDCERR_SUCCESS)
    AfxMessageBox(_T("Got ThirdPartyConfig"));
else
      AfxMessageBox(_T("Failed"));
```

Function: Get3rdPartyConfig

GetAllConfigs Sample Code

```
SDCConfig allConfigs[MAX_CFGS];
unsigned long numberOfConfigs;
SDCERR sdcErr;

//Get config structures
sdcErr = GetAllConfigs(&allConfigs, &numberOfConfigs);


if (sdcErr == SDCERR_SUCCESS)
      AfxMessageBox(_T("Got All Configs"));

else
      AfxMessageBox(_T("Didn't Get All Configs"));
```

Function: GetAllConfigs

GetConfig Sample Code

```
SDCERR result;
SDCConfig cfg = {0};

if(GetConfig("Default", &cfg)!=SDCERR_SUCCESS)
 printf("error in GetConfig\n");
else
 printf("config %s's SSID is ->%s<-\n", cfg.configName, cfg.SSID);
```

Function: GetConfig

GetConfigFileInfo Sample Code

```
CONFIG_FILE_INFO info;
SDCERR sdcErr;

//Load config file info into the CONFIG_FILE_INFO struct
sdcErr = GetConfigFileInfo("summit.sdc", &info);

if (SDCERR_SUCCESS == sdcErr)
      AfxMessageBox(_T("Got config file info"));

else
      AfxMessageBox(_T("Didn't get config file info"));
```

Function: GetConfigFileInfo

GetCurrentConfig Sample Code

```
char szName[80];
unsigned long dwNum;
SDCERR sdcErr;

sdcErr = GetCurrentConfig(&dwNum, szName);

if (SDCERR_SUCCESS == sdcErr)
      AfxMessageBox(_T("Got current config"));
      else
      AfxMessageBox(_T("Didn't get current config"));
```

Function: GetCurrentConfig

GetCurrentDomain Sample Code

```
REG_DOMAIN reg;
reg = GetCurrentDomain();
```

Function: GetCurrentDomain

GetCurrentStatus Sample Code

```
CF10G_STATUS st;
SDCERR sdcErr;

     sdcErr = GetCurrentStatus(&st);

if (sdcErr == SDCERR_SUCCESS)
     AfxMessageBox(_T("GetStatus OK"));
else
     AfxMessageBox(_T("GetStatus FAILED"));
```

Function: GetCurrentStatus

GetEAPFASTCred Sample Code

```
SDCConfig config;
SDCERR sdcErr;
char user[65];
char pwd[65];
char file[65];
char pacpwd[65];

//Get a valid config using GetConfig or another call...
memset(&config, 0, sizeof(SDCConfig));
sdcErr = GetConfig("Config 1", &config);

sdcErr = GetEAPFASTCred(&config, user, pwd, file, pacpwd);

if (sdcErr == SDCERR_SUCCESS)
     AfxMessageBox(_T("Got EAPFAST creds"));
else
     AfxMessageBox(_T("Didn't get EAPFAST creds"));
```

Function: GetEAPFASTCred

GetEAPTLSCred Sample Code

```
CERTLOCATION certLoc;
char user[65];
BYTE* pbHash = new BYTE[20];;
BYTE* pbHashEmpty = new BYTE[20];
SDCConfig config;

//Get a valid config using GetConfig or another call...
memset(&config, 0, sizeof(SDCConfig));
sdcErr = GetConfig("Config 1", &config);

sdcErr = GetEAPTLSCred(&config, user, (char*)pbHash, &certLoc, (char*)
pbHashEmpty);

if (sdcErr == SDCERR_SUCCESS)
     AfxMessageBox(_T("Got EAPTLS creds"));
```

```
else
     AfxMessageBox(_T("Didn't get EAPTLS creds"));
```

Function: GetEAPTLSCred

GetEAPTTLSCred Sample Code

```
CERTLOCATION certLoc;
 char user[65];
char password[65];
 BYTE* pbHash = new BYTE[20];
SDCConfig config;

//Get a valid config using GetConfig or another call...
memset(&config, 0, sizeof(SDCConfig));
sdcErr = GetConfig("Config 1", &config);

sdcErr = GetEAPTLSCred(&config, user, password, &certLoc, (char*) pbHash);

if (sdcErr == SDCERR_SUCCESS)
     AfxMessageBox(_T("Got EAPTTLS creds"));
else
     AfxMessageBox(_T("Didn't get EAPTTLS creds"));
```

Function: GetEAPTTLSCred

GetGlobalSettings Sample Code

```
SDCGlobalConfig gcfg;
SDCERR sdcErr;

memset(&gcfg, 0, sizeof(gcfg));

sdcErr = GetGlobalSettings(&gcfg);

if (sdcErr == SDCERR_SUCCESS)
     AfxMessageBox(_T("Got Global Settings"));
else
     AfxMessageBox(_T("Didn't global settings"));
```

Function: GetGlobalSettings

GetMultipleWEPKeys Sample Code

```
SDCConfig config;
SDCERR sdcErr;
unsigned long configNumber = 0;
WEPLEN len1, len2, len3, len4;
unsigned char k1[30], k2[30], k3[30], k4[30];
int tx;


//Get a valid config using GetConfig or another call...
memset(&config, 0, sizeof(SDCConfig));
sdcErr = GetConfig("Config 1", &config);

sdcErr = GetMultipleWEPKeys(&config, &tx, &len1, k1, &len2, k2, &len3,     k3,
&len4, k4);

if (sdcErr == SDCERR_SUCCESS)
         AfxMessageBox(_T("Got WEP keys"));
else
AfxMessageBox(_T("Didn't get WEP keys"));
```

Function: GetMultipleWEPKeys

GetNumConfigs Sample Code

```
SDCERR sdcErr;
unsigned long numConfigs

sdcErr = GetNumConfigs(&numConfigs);


if (sdcErr == SDCERR_SUCCESS)
     AfxMessageBox(_T("Got number of configs"));
else
     AfxMessageBox(_T("Didn't get number of configs"));
```

Function: GetNumConfigs

GetPEAPGTCCred Sample Code

```
char user[65];
char pwd[65];
char cert[65];
CERTLOCATION certLoc = CERT_NONE;
SDCConfig config;

//Get a valid config using GetConfig or another call...
memset(&config, 0, sizeof(SDCConfig));
sdcErr = GetConfig("Config 1", &config);

sdcErr = GetPEAPGTCCred(&config, user, pwd, &certLoc, cert);
```

```
if (sdcErr == SDCERR_SUCCESS)
     AfxMessageBox(_T("Got PEAPGTC creds"));
else
     AfxMessageBox(_T("Didn't get PEAPGTC creds"));
```

Function: GetPEAPGTCCred

GetPEAPMSCHAPCert Sample Code

```
char user[65];
char pwd[65];
char cert[65];
CERTLOCATION certLoc = CERT_NONE;
SDCConfig config;

//Get a valid config using GetConfig or another call...
memset(&config, 0, sizeof(SDCConfig));
sdcErr = GetConfig("Config 1", &config);

sdcErr = GetPEAPMSCHAPCred(&config, user, pwd, &certLoc, cert);

if (sdcErr == SDCERR_SUCCESS)
     AfxMessageBox(_T("Got PEAP-MSCHAP creds"));
else
     AfxMessageBox(_T("Didn't get PEAP-MSCHAP creds"));
```

Function: GetPEAPMSCHAPCert

GetPEAPTLSCred Sample Code

```
char username[USER_NAME_SZ];
CERTLOCATION certPath = CERT_FILE;
char caCert[CRED_CERT_SZ];
char usercert[CRED_CERT_SZ];
char usercerttemp[CRED_CERT_SZ];
Result result(SDC);
SDCConfig cfg={0};

Result = GetConfig("Default", &cfg);

result = GetPEAPTLSCred(&cfg, username, usercerttemp, &certPath, caCert);
if (result==SDCERR_SUCCESS)
{
 // valid data
}
```

Function: GetPEAPTTLSCred

GetPSK Sample Code

```
char myPSK[65];
SDCConfig config;

//Get a valid config using GetConfig or another call...
memset(&config, 0, sizeof(SDCConfig));
sdcErr = GetConfig("Config 1", &config);

sdcErr = GetPSK(&config,  myPSK);

if (sdcErr == SDCERR_SUCCESS)
     AfxMessageBox(_T("Got PSK creds"));
else
     AfxMessageBox(_T("Didn't get PSK creds"));
```

Function: GetPSK

GetSDKVersion Sample Code

```
unsigned long version;
SDCERR sdcErr;

sdcErr = GetSDKVersion(&version);

if (SDCERR_SUCCESS == sdcErr)
     AfxMessageBox(_T("Got version"));
else
     AfxMessageBox(_T("Didn't get version"));
```

Function: GetSDKVersion

GetWEPKey Sample Code

```
SDCConfig config;
char configName[80];
unsigned long configNumber = 0;
unsigned char key[26];

//Get a valid config using GetConfig or another call...
memset(&config, 0, sizeof(SDCConfig));
sdcErr = GetCurrentConfig(&configNumber, configName);
sdcErr = GetConfig(configName, &config);

//Get the WEP key
sdcErr = GetWEPKey(&config, 1, NULL, (unsigned char *)key, NULL);

if (sdcErr == SDCERR_SUCCESS)
     AfxMessageBox(_T("Got wep key"));
```

Function: GetWEPKey

importSettings Sample Code

```
SDC_ALL all;
SDCERR sdcErr;

memset(&all, 0, sizeof(SDC_ALL));

//import from file into SDC_ALL struct
sdcErr = importSettings("summit.sdc",&all);

if (sdcErr == SDCERR_SUCCESS)
     AfxMessageBox(_T("Imported"));
```

Function: importSettings

LRD_WF_GetaLRSBitmask Sample Code

```
SDCERR result;
int numChannels = 5;
LRD_WF_LRSChannels channels = {36,40,44,132,165};
unsigned long bitmask;

if(LRD_WF_GetaLRSBitmask(numChannels, channels, &bitmask) != SDCERR_SUCCESS)
{
     printf("error in LRD_WF_GetaLRSBitmask()\n");
} else {
     printf("channel bitmask: 0x%06x\n", bitmask);
}
```

Function: LRD_WF_GetaLRSBitmask

LRD_WF_GetaLRSChannels Sample Code

```
SDCERR result;
unsigned long numChannels;
LRD_WF_LRSChannels channels = {0};
unsigned long bitmask = 0x810007;

if(LRD_WF_GetaLRSChannels(&numChannels, &channels, bitmask) !=
SDCERR_SUCCESS) {
     printf("error in LRD_WF_GetaLRSChannels()\n");
}else{
     int i;
     printf("%d channels: ", numChannels);
     for (i=0; i< numChannels; i++)
         printf("%d,",channels.chan[i]);
     printf("\n");
}
```

Function: LRD_WF_GetaLRSChannels

LRD_WF_GetbLRSBitmask Sample Code

```
SDCERR result;
int numChannels = 3;
LRD_WF_LRSChannels channels = {1,2,3,};
unsigned long bitmask;

if(LRD_WF_GetbLRSBitmask(numChannels, channels, &bitmask) != SDCERR_SUCCESS)
{
 printf("error in LRD_WF_GetbLRSBitmask()\n");
} else {
 printf("channel bitmask: 0x%04x\n", bitmask);
}
```

Function: LRD_WF_GetbLRSBitmask

LRD_WF_GetbLRDChannels Sample Code

```
SDCERR result;
unsigned long numChannels;
LRD_WF_LRSChannels channels = {0};
unsigned long bitmask = 0x17;

if(LRD_WF_GetbLRSChannels(&numChannels, &channels, bitmask) !=
SDCERR_SUCCESS) {
     printf("error in LRD_WF_GetbLRSChannels()\n");
}else{
     int i;
     printf("%d channels: ", numChannels);
     for (i=0; i< numChannels; i++)
         printf("%d,",channels.chan[i]);
     printf("\n");
}
```

Function: LRD_WF_GetbLRDChannels

LRD_WF_GetDHCPLease Sample Code

```
SDCERR result;
DHCP_LEASE dhcplease = {0};

if(LRD_WF_GetDHCPLease(&dhcplease)!=SDCERR_SUCCESS){
     printf("error in LRD_WF_GetDHCPLease()\n");
}else{
     printf("interface:    %s\n", dhcplease.interface);
     printf("address:      %s\n", dhcplease.address);
     printf("dns_server(s): %s\n", dhcplease.dns_servers);
}
```

Function: LRD_WF_GetDHCPLease

LRD_WF_GetBSSIDList Sample Code

```
//helper function to output ssid
void ssidToStdOut(LRD_WF_SSID ssid){
	char printAsHex=0;
	int i;

	if (ssid.len==0)
	{
		printf("\"\""); //0 length ssid
	}

	//check if output needs to be in hex, could use isascii() if available
	for (i=0; i<ssid.len; i++)
		if((ssid.val[i]<32) || (ssid.val[i]>126))
			printAsHex=1;

	if(printAsHex)
		printf("\\x");

	for (i=0; i<ssid.len; i++)
		if(printAsHex)
			printf("%02x",ssid.val[i]);
		else
			printf("%c",ssid.val[i]);
}
// helper function to output security mask
void securityMaskToStdOut(unsigned int mask){
	struct securityType {
		WEPTYPE type;
		char* str;
	} securityList[17] ={
		{WAPI_CERT,"WAPI_CERT"},
		{WAPI_PSK,"WAPI_PSK"},
		{WPA2_AES,"WPA2_AES"},
		{CCKM_AES,"CCKM_AES"},
		{WPA_AES,"WPA_AES"},
		{WPA2_PSK,"WPA2_PSK"},
		{WPA_PSK_AES,"WPA_PSK_AES"},
		{WPA2_TKIP,"WPA2_TKIP"},
		{CCKM_TKIP,"CCKM_TKIP"},
		{WPA_TKIP,"WPA_TKIP"},
		{WPA2_PSK_TKIP,"WPA2_PSK_TKIP"},
		{WPA_PSK,"WPA_PSK"},
		{WEP_ON,"WEP_ON"},
		{WEP_AUTO,"WEP_AUTO"},
		{WEP_OFF,"WEP_OFF"},
		{WEP_AUTO_CKIP,"WEP_AUTO_CKIP"},
		{WEP_CKIP,"WEP_CKIP"}
	};
	int i;
	for (i=0; i<17; i++)
```

```
            if(mask & (1<<securityList[i].type))
                  printf(" %s", securityList[i].str);
}
…
SDCERR result;
LRD_WF_BSSID_LIST *list = NULL;
int numEntries = 100;
int numEntriesRequested;
LRD_WF_SCAN_ITEM_INFO *bss;
int retry = 1;

list = (LRD_WF_BSSID_LIST
*)malloc(numEntries*sizeof(LRD_WF_SCAN_ITEM_INFO)+sizeof(unsigned long));

numEntriesRequested = numEntries;
if (list != NULL) {
      do{
            result = LRD_WF_GetBSSIDList(list, &numEntries);
            if (result==SDCERR_INSUFFICIENT_MEMORY) {
                  if (numEntries==-1) {
                        printf("Scan API indicated system insufficient
memory\n");
                        retry =0;
                  } else {
                        if(retry) {
                              numEntries *= 1.25; //allow for 25% more then asked
for
                              free(list);
                              list = (LRD_WF_BSSID_LIST
*)malloc(numEntries*sizeof(LRD_WF_SCAN_ITEM_INFO)+sizeof(unsigned long));
                              }else{
                                    printf("Scan truncated.  Showing %d of %d
APs.  Try again for larger list.\n", numEntriesRequested, numEntries);
                              }
                        }
                  }else if (result==SDCERR_FAIL){
                        if(retry) {
                              printf("Retrying scan in 1 second\n");
                              sleep(1);
                        }else{
                              printf("scan aborted.  Please try again\n");
                        }
                  }
                  if (result==SDCERR_SUCCESS){
                        int i;
                        for (i=0; i< list->NumberOfItems; i++){
                              printf("BSS %d:\n", i);
                              bss=&list->Bssid[i];
                              printf("SSID: ");
                              ssidToStdOut(bss->ssid);
                              printf("\nChannel: %d\n", bss->channel);
```

```
                                    printf("RSSI: %dBm\n", bss->rssi/100);
                                    printf("Security: ");
                                    securityMaskToStdOut(bss->securityMask);
                                    printf("\n");
                              }
                              retry =0;
                        }
                  } while (retry--);
            }
free(list);
```

Function: LRD_WF_GetBSSIDList

LRD_WF_GetFIPSStatus Sample Code

```
char current, next;
typedef enum {
      FIPS_INACTIVE =0,
      FIPS_INACTIVE_ENABLED,
      FIPS_ACTIVE_DISABLED,
      FIPS_ACTIVE,
      FIPS_UNKNOWN
} FIPS_STATUS;

FIPS_STATUS combined = FIPS_UNKNOWN;

if (LRD_WF_GetFipsStatus(&current, &next)==SDCERR_SUCCESS)
      combined = (FIPS_STATUS)((current << 1) | next);
switch (combined) {
      case FIPS_INACTIVE:
            printf("Disabled and Inactive\n");
            break;
      case FIPS_INACTIVE_ENABLED:
            printf("Inactive - Enabled on next start\n");
            break;
      case FIPS_ACTIVE_DISABLED:
            printf("Active - Disabled on next start\n");
            break;
      case FIPS_ACTIVE:
            printf("Enabled and Active\n");
            break;
      default:
            printf("Unable to determine\n");
}
```

Function: LRD_WF_GetFIPSStatus

LRD_WF_GetPilInfo Sample Code

```
LRD_WF_PilInfo pil_info;

if(LRD_WF_GetPilInfo(&pil_info)!=SDCERR_SUCCESS)
    printf("Error in LRD_WF_GetPilInfo()\n");
else {
    printf("API: %x\n", pil_info.api_version);
    printf("Company: %s\n", pil_info.company_name);
    printf("version_string: %s\n",pil_info.version_string);
}
```

Function: LRD_WF_GetPilInfo

LRD_WF_GetSSID Sample Code

```
// uses ssidToStdOut() from the LRD_WF_GetBSSID() Sample code above
LRD_WF_SSID ssid={0};

if(LRD_WF_GetSSID(&ssid)!=SDCERR_SUCCESS)
    printf("error in LRD_WF_GetSSID()\n");
else {
    printf("SSID: ");
    ssidToStdOut(ssid);
    printf("\n");
}
```

Function: LRD_WF_GetSSID

ModifyConfig Sample Code

```
SDCConfig config;
SDCERR sdcErr;

memset(&config, 0, sizeof(SDCConfig));

sdcErr = GetConfig("Config1", &config);

//change the ssid of the configuration to Summit1
sprintf(config.SSID,"Summit1");

//update the config
sdcErr = ModifyConfig("Config1", &config);
```

Function: ModifyConfig

QueryOID Sample Code

```
ULONG size = sizeof(NDIS_STATISTICS_VALUE)+512;
UCHAR QueryBuffer[sizeof(NDIS_STATISTICS_VALUE)+512];
int err;

//OID_GEN_XMIT_OK OID specifies the number of frames that are transmitted
without errors
err = QueryOID(OID_GEN_XMIT_OK,QueryBuffer,size);

NDIS_STATISTICS_VALUE* queryOID = (PNDIS_STATISTICS_VALUE) &QueryBuffer[0];

unsigned long value = *(unsigned long*)&queryOID->Data[0];
char temp[100] = {0};
_itoa(value, temp, 10);

CString str = "";
str += temp;
str = str ;

if ( err > 0 )
     AfxMessageBox(str);
Else
     AfxMessageBox("Query failed");
```

Function: QueryOID

RadioEnable Sample Code

```
SDCGlobalConfig globalConfig;
SDCERR sdcErr;

memset(&globalConfig, 0, sizeof(SDCGlobalConfig));

sdcErr = GetGlobalSettings(&globalConfig);

//check to see if radio is enabled/disabled
if (globalConfig.radioState == 0)
     sdcErr = RadioEnable();

if ( sdcErr == SDCERR_SUCCESS )
     AfxMessageBox("Enabled");
else
     AfxMessageBox("Failed");
```

Function: RadioEnable

RadioDisable Sample Code

```
SDCGlobalConfig globalConfig;
SDCERR sdcErr;

memset(&globalConfig, 0, sizeof(SDCGlobalConfig));

sdcErr = GetGlobalSettings(&globalConfig);

//check to see if radio is enabled/disabled
if (globalConfig.radioState == 1)
    sdcErr = RadioDisable();

if ( sdcErr == SDCERR_SUCCESS )
    AfxMessageBox("Disabled");
else
        AfxMessageBox("Failed");
```

Function: RadioDisable

Set3rdPartyConfig Sample Code

```
SDC3rdPartyConfig config3;
SDCERR sdcErr;
char name[17] = "Summit";

memset(&config3, 0, sizeof(SDC3rdPartyConfig));

//Build a third party config
config3.bitRate = BITRATE_54;
memcpy(config3.clientName, name, 17);
config3.powerSave = POWERSAVE_FAST;
config3.radioMode = RADIOMODE_ABG;
config3.txPower = TXPOWER_50;

sdcErr = Set3rdPartyConfig(&config3);

if ( sdcErr == SDCERR_SUCCESS )
    AfxMessageBox(_T("Set third party config"));
else
    AfxMessageBox(_T("Failed"));
```

Function: Set3rdPartyConfig

SetAllConfigs Sample Code

```
SDCConfig config[2];
unsigned long numberOfConfigs = 2;
SDCERR sdcErr;

memset(&config, 0, sizeof(SDCConfig)*2);

//build a couple of configs from scratch or use GetAllConfigs()
sprintf(config[0].configName,"Config 1");
sprintf(config[0].SSID,"Summit1");
sprintf(config[0].clientName,"Client 1");
config[0].txPower = TXPOWER_MAX;
config[0].authType = AUTH_OPEN;
config[0].eapType = EAP_NONE;
config[0].powerSave = POWERSAVE_FAST;
config[0].wepType = WEP_OFF;
config[0].bitRate = BITRATE_AUTO;
config[0].radioMode = RADIOMODE_BG;

sprintf(config[1].configName,"Config 2");
sprintf(config[1].SSID,"Summit2");
sprintf(config[1].clientName,"Client 2");
config[1].txPower = TXPOWER_MAX;
config[1].authType = AUTH_OPEN;
config[1].eapType = EAP_NONE;
config[1].powerSave = POWERSAVE_FAST;
config[1].wepType = WEP_OFF;
config[1].bitRate = BITRATE_AUTO;
config[1].radioMode = RADIOMODE_BG;

sdcErr = SetAllConfigs(numberOfConfigs, &config[0]);

if (sdcErr == SDCERR_SUCCESS)
    AfxMessageBox(_T("Set All Configs"));
else
    AfxMessageBox(_T("Didn't Set All Configs"));
```

Function: SetAllConfigs

SetDefaultConfigValues Sample Code

```
SDCConfig config;
SDCERR sdcErr;

memset(&config, 0, sizeof(SDCConfig));

//create the default config
sprintf(config[0].configName,"Config 1");
sprintf(config[0].SSID,"Summit1");
sprintf(config[0].clientName,"Client 1");
config[0].txPower = TXPOWER_MAX;
config[0].authType = AUTH_OPEN;
config[0].eapType = EAP_NONE;
config[0].powerSave = POWERSAVE_FAST;
config[0].wepType = WEP_OFF;
config[0].bitRate = BITRATE_AUTO;
config[0].radioMode = RADIOMODE_BG;

sdcErr = SetDefaultConfigValues(&config);

if (sdcErr == SDCERR_SUCCESS)
        AfxMessageBox(_T("Set Default Values"));
else if ( sdcErr == SDCERR_INVALID_CONFIG )
        AfxMessageBox(_T("Invalid Config"));
else
        AfxMessageBox(_T("Didn't Set Default Values"));
```

Function: SetDefaultConfigValues

SetEAPFASTCred Sample Code

```
SDCConfig config;
SDCERR sdcErr;
char user[] = "myUserName";
char pwd[] = "myPassWord";
char pac1[] = "000pac000";
char pac2[] = "12345678901234567890123456789012345678901234567890";
char configName[80];
unsigned long configNumber = 0;

//Get a valid config using GetConfig or another call...
memset(&config, 0, sizeof(SDCConfig));
sdcErr = GetCurrentConfig(&configNumber, configName);
sdcErr = GetConfig(configName, &config);

//Set WEP type and EAP type
config.wepType = WPA_TKIP;
config.eapType = EAP_EAPFAST;

//Set the EAP-FAST credentials
sdcErr=SetEAPFASTCred(&config, user, pwd, pac1, pac2);

//Save the config by using ModifyConfig, AddConfig, etc.
sdcErr = ModifyConfig(configName, &config);
```

Function: SetEAPFASTCred

SetEAPTLSCred Sample Code

```
SDCConfig config;
SDCERR sdcErr;
BYTE* userCert = new BYTE[20];
BYTE* caCert = new BYTE[20];
char configName[80];
unsigned long configNumber = 0;
CERTLOCATION certLocation = CERT_IN_STORE;

//Get a valid config using GetConfig or another call...
memset(&config, 0, sizeof(SDCConfig));
sdcErr = GetCurrentConfig(&configNumber, configName);
sdcErr = GetConfig(configName, &config);

//Set WEP type and EAP type
config.wepType = WPA_TKIP;
config.eapType = EAP-EAPTLS;

//Set the EAP-TLS credentials
sdcErr=SetEAPTLSCred(&config, "user", (char*)userCert, certLocation,
(char*)caCert);

//Save the config by using ModifyConfig, AddConfig, etc.
sdcErr = ModifyConfig(configName, &config);
```

Function: SetEAPTLSCred

SetEAPTTLSCred Sample Code

```
SDCConfig config;
SDCERR sdcErr;
BYTE* userCert = new BYTE[20];
BYTE* caCert = new BYTE[20];
char configName[80];
unsigned long configNumber = 0;
CERTLOCATION certLocation = CERT_IN_STORE;

//Get a valid config using GetConfig or another call...
memset(&config, 0, sizeof(SDCConfig));
sdcErr = GetCurrentConfig(&configNumber, configName);
sdcErr = GetConfig(configName, &config);

//Set WEP type and EAP type
config.wepType = WPA_TKIP;
config.eapType = EAP-EAPTLS;

//Set the EAP-TLS credentials
sdcErr=SetEAPTTLSCred(&config, "user", "password", certLocation,
(char*)caCert);

//Save the config by using ModifyConfig, AddConfig, etc.
sdcErr = ModifyConfig(configName, &config);
```

Function: SetEAPTTLSCred

SetGlobalSettings Sample Code

```
SDCGlobalConfig configG;
SDCERR sdcErr;

memset(&configG, 0, sizeof(configG));

//Build the global config
configG.fragThreshold=FRAG_HIGH;
configG.RTSThreshold=RTS_HIGH;
configG.RxDiversity=RXDIV_START_MAIN;
configG.TxDiversity=TXDIV_ON;
configG.roamTrigger = RTRIG_70;
configG.roamDelta   = RDELTA_20;
configG.roamPeriod = RPERIOD_20;
configG.preamble = PRE_AUTO;
configG.g_shortslot =  GSHORT_AUTO;
configG.BTcoexist   = BT_OFF;
configG.pingPayload =  PP_32;
configG.pingTimeout = 5000;
configG.pingDelay   = 1000;
configG.authTimeout = 8;

sdcErr = SetGlobalSettings(&configG);

if (sdcErr == SDCERR_SUCCESS)
    AfxMessageBox(_T("Added Global Config OK"));
else
      AfxMessageBox(_T("Added Global Config FAILED"));
```

Function: SetGlobalSettings

SetLEAPCred Sample Code

```
SDCConfig config;
SDCERR sdcErr;
char user[] = "myUserName";
char pwd[] = "myPassWord";
char configName[80];
unsigned long configNumber = 0;

//Get a valid config using GetConfig or another call...
memset(&config, 0, sizeof(SDCConfig));
sdcErr = GetCurrentConfig(&configNumber, configName);
sdcErr = GetConfig(configName, &config);

//Set WEP type and EAP type
config.wepType = WPA_TKIP;
config.eapType = EAP_LEAP;

//Set the LEAP credentials
sdcErr=SetLEAPCred(&config, user, pwd);

//Save the config by using ModifyConfig, AddConfig, etc.
sdcErr = ModifyConfig(configName, &config);
```

Function: SetLEAPCred

SetMultipleWEPKeys Sample Code

```
SDCConfig config;
SDCERR sdcErr;
char configName[80];
unsigned long configNumber = 0;

//Get a valid config using GetConfig or another call...
memset(&config, 0, sizeof(SDCConfig);
sdcErr = GetCurrentConfig(&configNumber, configName);
sdcErr = GetConfig(configName, &config);

//Set WEP type and EAP type
config.wepType = WEP_ON;
config.eapType = EAP_NONE;

//Set the WEP key info
sdcErr = SetMultipleWEPKeys(&config, 3, WEPLEN_40BIT, (unsigned char*)
"1111111111", WEPLEN_NOT_SET,
(unsigned char*)"2222222222", WEPLEN_40BIT, (unsigned char*)"3333333333",
WEPLEN_128BIT,
(unsigned char*)"1234567890123456789012345678");

//Save the config by using ModifyConfig, AddConfig, etc.
      sdcErr = ModifyConfig(configName, &config);
```

Function: SetMultipleWEPKeys

SetOID Sample Code

```
UCHAR
QueryBuffer[sizeof(NDIS_STATISTICS_VALUE)+sizeof(NDIS_802_11_BSSID_LIST_EX
)*100];
 int err;

//OID_802_11_BSSID_LIST_SCAN requests that the miniport driver direct the
802.11 NIC to request a survey of BSSs
retval = SetOID(OID_802_11_BSSID_LIST_SCAN, QueryBuffer,
sizeof(NDIS_STATISTICS_VALUE)+sizeof(NDIS_802_11_BSSID_LIST_EX)*100);

if ( err > 0 )
     AfxMessageBox("OID Set");
else
     AfxMessageBox("OID Not Set");
```

Function: SetOID

SetPEAPGTCCred Sample Code

```
SDCConfig config;
SDCERR sdcErr;
char user[] = "userName";
char pwd[] = "passWord";
char cert[] = "000pac000.cer";
CERTLOCATION certLocation = CERT_FILE;
char configName[80];
unsigned long configNumber;

//Get a valid config using GetConfig or another call…
memset(&config, 0, sizeof(SDCConfig));
sdcErr = GetCurrentConfig(&configNum, configName);
sdcErr = GetConfig(configName, &config);

//Set WEP type and EAP type
config.wepType = WPA_TKIP;
config.eapType = EAP_PEAPGTC;

//Set the PEAP-GTC credentials
sdcErr=SetPEAPGTCCred(&config, user, pwd, certLocation, cert);

//Save the config by using ModifyConfig, AddConfig, etc.
sdcErr = ModifyConfig(configName, &config);
```

Function: SetPEAPGTCCred

SetPEAPMSCHAPCred Sample Code

```
SDCConfig config;
SDCERR sdcErr;
char user[] = "userName";
char pwd[] = "passWord";
char cert[] = "000pac000.cer";
CERTLOCATION certLocation = CERT_FILE;
char configName[80];
unsigned long configNumber;

//Get a valid config using GetConfig or another call…
memset(&config, 0, sizeof(SDCConfig));
sdcErr = GetCurrentConfig(&configNum, configName);
sdcErr = GetConfig(configName, &config);

//Set WEP type and EAP type
config.wepType = WPA_TKIP;
config.eapType = EAP_PEAPGTC;

//Set the PEAP-GTC credentials
sdcErr=SetPEAPMSCHAPCred(&config, user, pwd, certLocation, cert);

//Save the config by using ModifyConfig, AddConfig, etc.
sdcErr = ModifyConfig(configName, &config);
```

Function: SetPEAPMSCHAPCred

SetPEAPTLSCred Sample Code

```
SDCConfig config;
SDCERR sdcErr;
BYTE* caCert = new BYTE[20];
char configName[80];
unsigned long configNumber = 0;
CERTLOCATION certLocation = CERT_IN_STORE;

//Get a valid config using GetConfig or another call...
memset(&config, 0, sizeof(SDCConfig));
sdcErr = GetCurrentConfig(&configNumber, configName); sdcErr =
GetConfig(configName, &config);

//Set WEP type and EAP type
config.wepType = WPA_TKIP;
config.eapType = EAP-PEAPTLS;

//Set the EAP-TLS credentials
sdcErr=SetPEAPTLSCred(&config, username, password, &certPath,
"000pac000.cer");

//Save the config by using ModifyConfig, AddConfig, etc.
sdcErr = ModifyConfig(configName, &config);
```

Function: SetPEAPTLSCred

SetPSK Sample Code

```
SDCConfig config;
SDCERR sdcErr;
char configName[80];
unsigned long configNumber = 0;
char hexPSK[] =
"0123456789012345678901234567890123456789012345678901234567890123456789abcd";

//Get a valid config using GetConfig or another call...
memset(&config, 0, sizeof(SDCConfig));
sdcErr = GetCurrentConfig(&configNumber, configName);
sdcErr = GetConfig(configName, &config);

//Set WEP type and EAP type
config.wepType = WPA_PSK;
config.eapType = EAP_NONE;

//Set the PSK
sdcErr = SetPSK(&config, hexPSK);

//Save the config by using ModifyConfig, AddConfig, etc.
sdcErr = ModifyConfig(configName, &config);
```

Function: SetPSK

SetWEPKey Sample Code

```
SDCConfig config;
SDCERR sdcErr;
char configName[80];
unsigned long configNumber = 0;
unsigned char theWepKey[13] =
{0x11,0x11,0x11,0x11,0x11,0x11,0x11,0x11,0x11,0x11,0x11,0x11,0x11};

//Get a valid config using GetConfig or another call...
memset(&config, 0, sizeof(SDCConfig));
sdcErr = GetCurrentConfig(&configNumber, configName);
sdcErr = GetConfig(configName, &config);

//Set WEP type and EAP type
config.wepType = WEP_ON;
config.eapType = EAP_NONE;

//Set the WEP key info
sdcErr = SetWEPKey(&config, 1, WEPLEN_128BIT, theWepKey, FALSE);

//Save the config by using ModifyConfig, AddConfig, etc.
sdcErr = ModifyConfig(configName, &config);
```

Function: SetWEPKey