## Revision History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Notes</th>
<th>Approver</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>12 Sept 2016</td>
<td>Initial Release</td>
<td>N. Zach Hogya</td>
</tr>
</tbody>
</table>
CONTENTS

Introduction .................................................................................................................. 6
SDK Usage and Operation ......................................................................................... 6
API Reference ............................................................................................................ 6
SDK Usage and Operation ......................................................................................... 7
Introduction ................................................................................................................ 7
  Getting Started ......................................................................................................... 7
Global Settings Management ...................................................................................... 7
  Related Structures for Global Settings .................................................................. 7
  Related Global Settings Functions ......................................................................... 7
Profile Management ................................................................................................... 8
  Edit a Profile: Set a Single Static WEP Key ......................................................... 8
  Edit a Profile: Set Four Static WEP Keys ............................................................... 8
  Edit a Profile: Configure LEAP ............................................................................ 8
  Related Structures for Configuration Profiles ....................................................... 9
  Related Profile Management Functions ............................................................... 9
Monitoring and Status ............................................................................................... 9
  Obtain Status Information ..................................................................................... 9
  Determine Signal Quality ...................................................................................... 10
  Related Monitoring and Status Functions ............................................................ 10
ThirdPartyConfig ........................................................................................................ 10
  Related Structures for ThirdPartyConfig ............................................................ 10
  Related ThirdPartyConfig Functions .................................................................. 11
Regulatory Domains ................................................................................................. 11
  Related ENUM for Regulatory Domains ............................................................. 11
  Related Regulatory Domains Functions ............................................................... 11
FCC (Windows only feature) ..................................................................................... 11
  Related ENUM for FCC ....................................................................................... 11
API Reference ............................................................................................................ 12
Functions .................................................................................................................... 12
Function Descriptions .............................................................................................. 13
  ActivateConfig ....................................................................................................... 13
  AddConfig ............................................................................................................... 13
  CreateConfig .......................................................................................................... 13
  DeleteConfig ........................................................................................................... 14
  exportSettings ....................................................................................................... 14
  Get3rdPartyConfig ............................................................................................... 14
  GetAllConfigs ........................................................................................................ 15
  GetBSSIDList .......................................................................................................... 15
  GetConfig ............................................................................................................... 16
  GetConfigFileInfo ................................................................................................. 16
  GetCurrentConfig ................................................................................................. 16
  GetCurrentDomain ................................................................................................. 17
  GetCurrentDomainEx ............................................................................................. 17
  GetCurrentStatus ................................................................................................. 17
  GetEAPFASTCred ................................................................................................. 17
  GetEAPTLSAuth .................................................................................................. 18
  GetEAPTTLSAuth ................................................................................................. 18
  GetFrameworkVersion .......................................................................................... 19
  GetGlobalSettings ............................................................................................... 19
  GetLEAPCred ......................................................................................................... 19
  GetMultipleWEPKeys ......................................................................................... 20
Laird Wi-Fi SDK
User Guide

GetNumConfigs ................................................................. 20
GetOSVersion ................................................................. 20
GetPEAPGTCCred ............................................................ 21
GetPEAPMSCHAPCred ....................................................... 21
GetPEAPTLSCred ............................................................ 22
GetPSK .......................................................................... 22
GetWEPKey ................................................................. 23
importSettings ............................................................... 23
LRD_WF_GetSSID ........................................................... 23
LRD_WF_IcmpSendEcho .................................................... 24
ModifyConfig ............................................................... 24
RadioDisable ................................................................. 24
RadioEnable ................................................................. 25
Set3rdPartyConfig ......................................................... 25
SetAllConfigs ............................................................... 25
SetDefaultConfigValues .................................................. 25
SetEAPFASTCred ........................................................... 26
SetEAPTLSCred ............................................................. 26
SetEAPTTLSrced ............................................................. 27
SetGlobalSettings ........................................................... 27
SetLEAPCred ................................................................. 28
SetMultipleWEPKeys ........................................................ 28
SetPEAPGTCCred ........................................................... 29
SetPEAPMSCHAPCred ....................................................... 29
SetPEAPTLSCred ........................................................... 30
SetPSK ......................................................................... 31
SetWEPKey ................................................................. 31

Structures ........................................................................ 32
CF10G_STATUS .............................................................. 32
CONFIG_FILE_INFO .......................................................... 33
CRYPT ......................................................................... 33
SCAN_ITEM_INFO ............................................................. 34
SDC_802_11_BSSID_LIST_EX ........................................... 35
SDC_ALL ...................................................................... 35
SDC3rdPartyConfig ........................................................ 35
SDCConfig ................................................................. 36
SDCGlobalConfig ........................................................... 37
LRD_WF_COMPONENT_VERSIONS ......................................... 42
LRD_WF_SSID ................................................................. 42
LRD_WF_SCAN_ITEM_INFO ................................................. 42
LRD_WF_BSSID_LIST ....................................................... 43

Enumeration Types .......................................................... 43
AUTH .......................................................................... 43
BITRATE ................................................................. 43
BT_CoEXIST .............................................................. 44
CARDSTATE .............................................................. 44
CCX_FEATURES ............................................................. 44
CERTLOCATION .......................................................... 45
DFS_CHANNELS ............................................................. 45
DriverNDISVersion ......................................................... 45
EAPTYPE ................................................................. 45
FCCTEST ................................................................. 45
GSHORTSLOT ............................................................. 46
INTERFERENCE ............................................................ 46
<table>
<thead>
<tr>
<th>Define</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LRD_WF_BSSTYPE</td>
<td>46</td>
</tr>
<tr>
<td>PING PAYLOAD</td>
<td>46</td>
</tr>
<tr>
<td>POWERSAVE</td>
<td>46</td>
</tr>
<tr>
<td>PREAMBLE</td>
<td>46</td>
</tr>
<tr>
<td>RADIOCHIPSET</td>
<td>47</td>
</tr>
<tr>
<td>RADIO MODE</td>
<td>47</td>
</tr>
<tr>
<td>RADIOTYPE</td>
<td>47</td>
</tr>
<tr>
<td>REGDOMAIN</td>
<td>48</td>
</tr>
<tr>
<td>ROAM_DELTA</td>
<td>48</td>
</tr>
<tr>
<td>ROAM_PERIOD</td>
<td>48</td>
</tr>
<tr>
<td>ROAM_TRIG</td>
<td>49</td>
</tr>
<tr>
<td>RX_DIV</td>
<td>49</td>
</tr>
<tr>
<td>SDCERR</td>
<td>49</td>
</tr>
<tr>
<td>TTLS_INNER_METHOD</td>
<td>50</td>
</tr>
<tr>
<td>TX_DIV</td>
<td>50</td>
</tr>
<tr>
<td>TXPOWER</td>
<td>50</td>
</tr>
<tr>
<td>WEPLLEN</td>
<td>50</td>
</tr>
<tr>
<td>WEP TYPE</td>
<td>51</td>
</tr>
<tr>
<td>Defines</td>
<td>51</td>
</tr>
<tr>
<td>802.11 Reason Codes</td>
<td>51</td>
</tr>
</tbody>
</table>
INTRODUCTION

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

Laird utilities, such as the Laird Connection Manager (LCM), use the SDK to interact with other components of Laird’s Wi-Fi 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.

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

Note: For details on LCM functions, profile settings, and global settings, consult the LCM User Guide.

On Windows platforms, LCM provides a GUI for access to all of its functions. Access to these functions is also available through the Laird Wi-Fi Software Development Kit, which can be used to manage the radio from other applications. This guide explains how to use the Laird Wi-Fi SDK from an application.

This SDK Programmer’s Guide includes the following sections:

SDK Usage and Operation

- Introduction
- Global Settings Management
- Profile Management
- Monitoring and Status
- ThirdPartyConfig
- Regulatory Domains
- FCC (Windows only feature)

API Reference

- Functions
- Function Descriptions
- Structures
- Enumeration Types
- Defines
**SDK USAGE AND OPERATION**

**Introduction**

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 driver files from the Software Downloads tab of your product’s page at [www.lairdtech.com](http://www.lairdtech.com).

**Getting Started**

**Step 1:** Open an application and navigate to “Solution Explorer”

**Step 2:** Right click “References” and select “Add Reference”

**Step 3:** Navigate to the LrdSdk.dll directory and add the reference.

Make sure that you use the appropriate LrdWifiLib.dll file for your platform.

**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 `GetFrameworkVersion` setting.

This function gets the version of the SDK.

**Returns:** String value of the C# SDK.

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 LCM application (adjusting this global setting changes the LCM password).
Related Global Settings Functions

SDK global settings functions include:

- GetFrameworkVersion

This function gets the version of the SDK.

**Returns:** String value of the C# SDK.

- GetGlobalSettings
- SetGlobalSettings
- RadioEnable

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, 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 *Save* 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.

**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.

**Edit a Profile: Configure LEAP**

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

**Related Structures for Configuration Profiles**

**Structure: SDCCConfig**

The structure SDCCConfig 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 SetEAPFASStCred rather than modify CRYPTs directly.

**Related Profile Management Functions**

Profile functions include:

- ActivateConfig
- AddConfig
- CreateConfig
- DeleteConfig
- GetAllConfigs
- GetConfig
- GetCurrentConfig
- GetEAPFASTCred
- GetEAPTLScred
- GetEAPPTTLScred
- GetLEAPCred
- GetMultipleWEPKeys
- GetNumConfigs
- GetPEAPGTCCred
- GetPEAPMSCHAPCert
- GetPSK

- GetWEPKey
- ModifyConfig
- SetAllConfigs
- SetDefaultConfigValues
- SetEAPFASTCred
- SetEAPTLScred
- SetEAPPTTLScred
- SetLEAPCred
- SetMultipleWEPKeys
- SetPEAPGTCCred
- SetPEAPMSCHAPCred
- SetPSK

**Monitoring and Status**

LCM includes various mechanisms for monitoring and status. The following sections illustrate how to implement LCM 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:

```c
SDK.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, /*Not available on windows */
} CARDSTATE;
```

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

**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. LCM 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 Monitoring and Status Functions**

- GetCurrentStatus
ThirdPartyConfig

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
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
- CreateConfig
- DeleteConfig
- exportSettings
- Get3rdPartyConfig
- GetAllConfigs
- GetBSSIDList
- GetConfig
- GetConfigFileInfo
- GetCurrentConfig
- GetCurrentDomain
- GetCurrentDomainEx
- GetCurrentStatus
- GetEAPFASTCred
- GetEAPTLSCred
- GetEAPTTLSCred
- GetFrameworkVersion
- GetGlobalSettings
- GetLEAPCred
- GetMultipleWEPKeys
- GetNumConfigs
- GetOSVersion
- GetPEAPGTCCred
- GetPEAPMSCHAPCred
- GetPEAPTLSCred
- GetPSK
- GetWEPKey
- importSettings
- LRD_WF_GetSSID
- LRD_WF_IcmpSendEcho
- ModifyConfig
- RadioDisable
- RadioEnable
- Set3rdPartyConfig
- SetAllConfigs
- SetDefaultConfigValues
- SetEAPFASTCred
- SetEAPTLSCred
- SetEAPTTLSCred
- SetGlobalSettings
- SetLEAPCred
- SetMultipleWEPKeys
- SetPEAPGTCCred
- SetPEAPMSCHAPCred
- SetPEAPTLSCred
- SetPSK
- SetWEPKey
Function Descriptions

ActivateConfig

This function activates the configuration with the given name.

SDK.SDCERR ActivateConfig(string 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

AddConfig

This function adds the configuration.

SDK.SDCERR AddConfig(ref SDK.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.

CreateConfig

This function creates a configuration from the default values.

SDK.SDCERR CreateConfig(ref SDK.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.

DeleteConfig

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

```c
SDK.SDCERR DeleteConfig(string 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.

exportSettings

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

```c
SDK.SDCERR ExportSettings(string filename, ref SDK.SDC3rdPartyConfig configThirdParty, ref SDK.SDCGlobalConfig configGlobal, ref SDK.SDCConfig[] configs, uint confignum, bool export3rdparty, bool exportglobal, bool exportconfigs)
```

Parameters:

- [in] filename – Name of the file (including the path) where the settings will be exported.
- [in] SDCConfig – Array of profile configs.
- [in] confignum – Number of profile configs.
- [in] export3rdparty – Set to ‘TRUE’ to export 3rd party config.
- [in] exportglobal – Set to ‘TRUE’ to export config config.
- [in] exportconfigs – Set to ‘TRUE’ to export profile configs.

Returns:

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

Get3rdPartyConfig

The function retrieves the third party configuration settings.

```c
SDK.SDCERR Get3rdPartyConfig(ref SDK.SDC3rdPartyConfig cfg3rd)
```
Parameters:


Returns:

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

### GetAllConfigs

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

```
SDK.SDCERR GetAllConfigs(SDK.SDCConfig[] cfgs, ref uint 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.

### GetBSSIDList

This function gets a list of BSSIDs from a scan.

```
SDK.SDCERR GetBSSIDList(SDK.SDC_802_11_BSSID_LIST_EX[] list, ref uint numBufEntries)
```

Parameters:

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

Returns:

- SDCERR_SUCCESS – Successful
- SDCERR_INVALID_PARAMETER – Invalid parameter
- SDCERR_INSUFFICIENT_MEMORY – List structure is not large enough
- SDCERR_FAIL – Error
- SDCERR_NOT_SUPPORTED – 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.

```csharp
SDK.SDCERR GetConfig(string name, ref SDK.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

GetConfigFileInfo
This function retrieves file details from a configuration file.

```csharp
SDK.SDCERR GetConfigFileInfo(string name, ref SDK.CONFIG_FILE_INFO info);
```

Parameters:
- `[in]` filename – A valid filename (required) to an exported .sdc document.
- `[out]` info – Pass in a pointer to an allocated CONFIG_FILE_INFO structure

Returns:
- **SDCERR_INVALID_PARAMETER** – Invalid filename or info
- **SDK.SDCERR FAIL** – Other error

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

```csharp
SDK.SDCERR GetCurrentConfig(ref uint num, StringBuilder 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
**GetCurrentDomain**

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

```cpp
SDK.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.

**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.

**GetCurrentDomainEx**

This function returns the current regulatory domain as defined by the SDK.REG_DOMAIN ENUM. If the return value is greater than the last item in the ENUM list then the value is populated with the AP’s country code as defined by the 802.11d standard. The value can be converted from a REG_DOMAIN into 2 ASCII characters. (I.E. US or UK).

```cpp
SDK.REG_DOMAIN GetCurrentDomainEx()
```

**Parameters:** None

**Returns:** The current regulatory domain of the radio.

**GetCurrentStatus**

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

```cpp
SDK.SDCERR GetCurrentStatus(ref SDK.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.

**GetEAPFASTCred**

This function retrieves the EAP-FAST credentials.

```cpp
SDK.SDCERR GetEAPFASTCred(ref SDK.SDCConfig cfg, StringBuilder username, StringBuilder password, StringBuilder pacfilename, StringBuilder pacpassword)
```

**Parameters:**

- `[in] cfg` – Valid configuration (required)
GetEAPTLSCred

This function retrieves the EAP-TLS credentials.

```c
SDK.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 an 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 `certLocation` 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

GetEAPTTLSCred

This function retrieves the EAP-TTLS credentials.

```c
SDK.SDCERR GetEAPTTLSCred (ref SDK.SDCConfig cfg, StringBuilder username,
StringBuilder password, ref SDK.CERTLOCATION CAcertLocation, byte[] 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 `certLocation` 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

**GetFrameworkVersion**

This function gets the version of the SDK.

Returns: String value of the C# SDK.

**GetGlobalSettings**

This function retrieves the global configuration settings.

```csharp
SDK.SDCERR GetGlobalSettings(ref SDK.SDCGlobalConfig gcfg)
```

Parameters:
- [out] `gcfg` – Global configuration

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

**GetLEAPCred**

This function retrieves the LEAP credentials.

```csharp
SDK.SDCERR GetLEAPCredGetLEAPCred(ref SDK.SDCCConfig cfg, StringBuilder username, StringBuilder 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.

```c
SDK.SDCERR GetMultipleWEPKeys(ref SDK.SDCConfig cfg, ref int nTxKey, ref SDK.WEPLEN key1Length, byte[] key1, ref SDK.WEPLEN key2Length, byte[] key2, ref SDK.WEPLEN key3Length, byte[] key3, ref SDK.WEPLEN key4Length, byte[] 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

**getNumConfigs**

This function retrieves the number of configurations present.

```c
SDK.SDCERR GetNumConfigs(ref uint num)
```

**Parameters:**
- `[out] num` – Number of current configurations (‘ThirdPartyConfig’ is not counted as a configuration)

**Returns:**
- `SDCERR_SUCCESS` – Successful
- `SDCERR_FAIL` – Internal error

**GetOSVersion**

This function returns the operation system version number.

```c
void GetOSVersion(ref uint Majorversion, ref uint Minorversion)
```

**Parameters:**
- `Majorversion` – The major revision number of the operating system.
Minorversion – The minor revision number of the operating system.

Returns:
None

GetPEAPGTCCred

This function retrieves the PEAP-GTC credentials.

SDK.SDCERR GetPEAPGTCCred(ref SDK.SDCConfig cfg, StringBuilder username, StringBuilder password, ref SDK.CERTLOCATION CAcertLocation, byte[] 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

GetPEAPMSCHAPCred

This function retrieves the PEAP-MSCHAP credentials.

SDK.SDCERR GetPEAPMSCHAPCred(ref SDK.SDCConfig cfg, StringBuilder username, StringBuilder password, ref SDK.CERTLOCATION CAcertLocation, byte[] 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 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

GetPEAPTLS_cred

This function returns the PEAPTLS credentials

SDK.SDCERR GetPEAPTLS_cred(ref SDK.SDCConfig cfg, StringBuilder username, byte[] userCert, ref SDK.CERTLOCATION CAcertLocation, byte[] 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_INVALID_FORMAT_PARAMETER – Invalid parameter
- SDCERR_INVALID_CONFIG – Invalid config

GetPSK

This function retrieves the PSK.

SDK.SDCERR GetPSK(ref SDK.SDCConfig cfg, StringBuilder 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
GetWEPKey

This function retrieves a WEP key.

```c
SDK.SDCERR GetWEPKey(ref SDK.SDCConfig cfg, int nWepKey, ref SDK.WEPLEN keyLength, ref byte[] key, ref byte 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

importSettings

This function imports Laird 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.

```c
SDK.SDCERR ImportSettings(string filename, ref SDK.SDC3rdPartyConfig configThirdParty, ref SDK.SDCGlobalConfig configGlobal, ref SDK.SDCConfig[] configs, bool import3rdparty, bool importglobal, bool importconfigs, ref uint numconfigs);
```

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

LRD_WF_GetSSID

```c
SDK.SDCERR LRD_WF_GetSSID(ref SDK.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_IcmpSendEcho**

This function sends a PING command over a network to a specified IP address

```c
LRD_WF_IcmpSendEcho(byte[] IPAddress, uint IPAddressLen, uint PayloadSize, uint Timeout)
```

Parameters:

- [in] IPAddress – The Destination IP address of the device to ping
- [in] IPAddressLen – The length of the byte array of the IPAddress parameter.
- [in] PayloadSize – The size of the Payload to be sent over the ping.
- [in] Timeout – The network timeout option for the ping request.

Returns:

- SDCERR_SUCCESS
- SDCERR_FAIL

**ModifyConfig**

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

```c
SDK.SDCERR ModifyConfig(string name, ref SDK.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

**RadioDisable**

This function disables the radio.

```c
SDK.SDCERR RadioDisable()
```
Returns:
- **SDCERR_SUCCESS** – Successful
- **SDCERR_FAIL** – Internal error

**RadioEnable**

This function enables the radio.

```
SDK.SDCERR RadioEnable()
```

Returns:
- **SDCERR_SUCCESS** – Successful
- **SDCERR_FAIL** – Internal error

**Set3rdPartyConfig**

This function stores the third party configuration settings.

```
SDK.SDCERR Set3rdPartyConfig(ref SDK.SDC3rdPartyConfig cfg3rd)
```

Parameters:
- `[in] cfg3rd` – Third party configuration

Returns:
- **SDCERR_SUCCESS** – Successful
- **SDCERR_INVALID_CONFIG** - *cfg is NULL or data is invalid
- **SDCERR_FAIL** – Internal error

**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.

```
SDK.SDCERR SetAllConfigs(uint num, SDK.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

**SetDefaultConfigValues**

This function sets default values for new configurations.

```
SetDefaultConfigValues(ref SDK.SDCConfig cfg)
```

Parameters:
- `[in] cfg` – Configuration

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

### SetEAPFASTCred

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

```csharp
SDK.SDCERR SetEAPFASTCred(ref SDK.SDCConfig cfg, string username, string password, string pacfilename, string 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

### SetEAPTLSCred

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

```csharp
SDK.SDCERR SetEAPTLSCred(ref SDK.SDCConfig cfg, string username, byte[] userCert, SDK.CERTLOCATION CAcertLocation, byte[] 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 `certLocation` 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

**SetEAPTTLSCred**

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

```csharp
SDK.SDCERR SetEAPTTLSCred(ref SDK.SDCConfig cfg, string username, string password, SDK.CERTLOCATION CAcertLocation, byte[] 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 `certLocation` 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_EAPTLS
- **SDCERR_INVALID_PARAMETER** – Invalid parameter
- **SDCERR_INVALID_CONFIG** – Invalid configuration
- **SDCERR_FAIL** – Other error

**SetGlobalSettings**

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

```csharp
SDK.SDCERR SetGlobalSettings(ref SDK.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
SetLEAPCred

This function sets the LEAP credentials. It validates the configuration’s `wepType` and `eapType`.

```
SDK.SDCERR SetLEAPCred(ref SDK.SDCConfig cfg, string username, string 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

SetMultipleWEPKeys

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

```
SDK.SDCERR SetMultipleWEPKeys(ref SDK.SDCConfig cfg, int nTxKey, SDK.WEPLEN key1Length, byte[] key1, SDK.WEPLEN key2Length, byte[] key2, SDK.WEPLEN key3Length, byte[] key3, SDK.WEPLEN key4Length, byte[] 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
- `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

**SetPEAPGTCCred**

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

```csharp
SDK.SDCERR SetPEAPGTCCred(ref SDK.SDCConfig cfg, string username, string password, SDK.CERTLOCATION CAcertLocation, byte[] 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**

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

```csharp
SDK.SDCERR SetPEAPMSCHAPCred(ref SDK.SDCConfig cfg, string username, string password, SDK.CERTLOCATION CAcertLocation, byte[] caCert)
```

**Parameters:**

- `[in] cfg` – Valid configuration (required)
SDK.SDCERR_SetPEAPTLScred(ref SDK.SDCConfig cfg, string username, byte[] userCert, SDK.CERTLOCATION caCertLocation, byte[] 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 `certLocation` 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_EAPFAST
- **SDCERR_INVALID_PARAMETER** – Invalid parameter
- **SDCERR_INVALID_CONFIG** – Invalid configuration
- **SDCERR_FAIL** – Other error
SetPSK

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

```csharp
SDK.SDCERR SetPSK(ref SDK.SDCConfig cfg, string 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

SetWEPKey

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

```csharp
SDK.SDCERR SetWEPKey(ref SDK.SDCConfig cfg, int nWepKey, SDK.WEPLEN keyLength, byte[] key, byte 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.
Structures

**CF10G_STATUS**

```csharp
public struct CF10G_STATUS
{
    public byte[] AP_IP;
    public byte[] AP_MAC;
    public string APName;
    public uint beaconPeriod;
    public uint beaconsReceived;
    public SDK.BITRATE bitRate;
    public SDK.CARDSTATE cardState;
    public uint channel;
    public byte[] client_IP;
    public byte[] client_MAC;
    public string clientName;
    public string configName;
    public int driverVersion;
    public uint DTIM;
    public SDK.EAPTYPE eapType;
    public SDK.RADIOTYPE radioType;
    public int rssi;
    public SDK.TXPOWER txPower;
}
```

**Elements:**

- **byte[] AP_IP**
  - Meaning: IPv4 address of the access point to which the radio is associated
  - Length: 4 byte values

- **byte[] AP_MAC**
  - Meaning: MAC address of the access point to which the radio is associated
  - Length: 6 byte values

- **string APName**
  - Meaning: Name of the access point to which the radio is associated
  - Length: CLIENT_NAME_SZ

- **uint beaconPeriod**
  - Meaning: The amount of time between access point beacons in Kilomicroseconds, where one Kµsec equals 1,024 microseconds
  - a DTIM interval of 3 means that every third beacon contains a DTIM

- **uint beaconsReceived**
  - Meaning: The number of beacons received

- **SDK.BITRATE bitRate**
  - Meaning: Data rate of the WLAN connection between the Laird radio and the AP
  - Values: BITRATE enum

- **SDK.CARDSTATE cardState**
  - Meaning: Association status
  - Values: CARDSTATE enum

- **uint channel**
  - Meaning: Channel of the WLAN connection between the Laird radio and the AP
- byte[] client_IP
  - Meaning: Client IPv4 address
  - Length: 4 byte values
- byte[] client_MAC
  - Meaning: Client MAC address
  - Length: 6 byte values
- string clientName
  - Meaning: The name assigned to the Laird radio and the client device that uses it
  - Length: 16 characters
- string configName
  - Meaning: Name of the active configuration profile
  - Length: 32 characters
- int driverVersion
  - Meaning: Driver software version number
- uint 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. CRYPT)
- SDK.EAPTYPE eapType
  - Meaning: Indicates the Extensible Authentication Protocol type used for 802.1X authentication to the AP
  - Values: SDK.EAPTYPE enum
- SDK.RADIOTYPE radioType
  - Meaning: The bands supported by the current LAIRD radio
  - Length: RADIOTYPE enum
- int rssi
  - Meaning: Signal strength (RSSI) of the WLAN connection between the Summit radio and the AP, displayed graphically and in dBm
- SDK.TXPOWER txPower
  - Meaning: Transmit power of the WLAN connection between the Summit radio and the AP

**Note:** 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.

**CONFIG_FILE_INFO**

```csharp
public struct CONFIG_FILE_INFO
{
    public byte globalConfigPresent;
    public uint numConfigs;
    public uint sdkVersion;
    public byte thirdPartyConfigPresent;
}
```

**Definitions**

**CRYPT**

```csharp
public struct CRYPT
```
{ 
    public byte[] buffer;
    public uint offset;
    public uint size;
}

- Definitions
- 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.

**SCAN_ITEM_INFO**

```csharp
public struct SCAN_ITEM_INFO
{
    public int adHocMode;
    public string bssid;
    public int channel;
    public SDK.EAPTYPE eapType;
    public int freq;
    public int rssi;
    public int security;
    public string ssid;
    public SDK.WEPTYPE wepType;
}
```

The LRD_WF_SCAN_ITEM_INFO structure contains the information for a related SSID.

Elements:

- **string adHocMode**
  - Meaning: Describes the mode of the item returned in the scan list.
- **string bssidMac**
  - Meaning: the MAC address for the BSSID
- **Int channel**
  - Meaning: The channel number the SSID is operating
- **EAPTYPE eapType**
  - Meaning: EAP security determined by the scan result.
- **Int freq**
  - Meaning: Frequency of the item discovered in the scan list.
- **Int rssi**
  - Meaning: The reported RSSI
- **int security**
  - 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
- **string ssid**
  - Meaning – this structure contains the SSID data
- **EAPTYPE eapType**
  - Meaning: EAP security determined by the scan result.
SDC_802_11_BSSID_LIST_EX

```csharp
public struct SDC_802_11_BSSID_LIST_EX
{
    public SDK.SCAN_ITEM_INFO[] Bssid;
    public uint NumberOfItems;
}
```

Elements:

- **SCAN_ITEM_INFO Bssid**
  - Meaning: An array of BSSID's discovered during a scan.
- **Uint NumberOfItems**
  - Meaning: The number of items returned in the SCAN_ITEM_INFO array.

SDC_ALL

```csharp
public struct SDC_ALL
{
    public IntPtr configGlobal;
    public IntPtr configs;
    public IntPtr configThirdParty;
    public uint numConfigs;
}
```

Elements:

- **IntPtr configGlobal**
  - Meaning: A pointer to a configuration that contains SDCGlobal stuct.
- **IntPtr configs**
  - Meaning: A pointer to an array of SDCConfig stucts.
- **IntPtr configThirdParty**
  - Meaning: A pointer to a configuration that contains SDC3rdPartyConfig stuct.
- **Uint numConfigs**
  - Meaning: The number of items returned in the ‘configs’ array.

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.

```csharp
public struct SDC3rdPartyConfig
{
    public SDK.BITRATE bitrate;
    public string clientName;
    public SDK.POWERSAVE powerSave;
    public SDK.RADIOMODE radioMode;
    public SDK.TXPOWER txPower;
}
```

Elements:
SDK.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 values in megabits per second (Mbps): BITRATE enum

string clientName
- Meaning: Name of configuration profile
- Value: 32 Characters

SDK.POWERSAVE powerSave
- Meaning: Power save protocol (PSP) method
- Value: POWERSAVE_OFF, POWERSAVE_MAX, POWERSAVE_FAST

SDK.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

SDK.TXPOWER txPower
- Meaning: Maximum transmit power in milliwatts (mW)
- Value: Any integer in the range of 0 to TXPOWER (See Global Settings)

SDCConfig
The structure SDCConfig is for a configuration profile.

```csharp
public struct SDCConfig
{
    public SDK.AUTH authType;
    public SDK.BITRATE bitRate;
    public string clientName;
    public string configName;
    public SDK.EAPTYPE eapType;
    public SDK.POWERSAVE powerSave;
    public SDK.CRYPT PSK;
    public SDK.RADIOMODE radioMode;
    public string SSID;
    public int txPower;
    public SDK.CRYPT userName;
    public SDK.CRYPT userPwd;
    public SDK.CRYPT WEPKeys;
    public SDK.WEPTYPE wepType;
}
```

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 values 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:

```csharp
public struct SDCGlobalConfig
{
    public uint adminFiles;
    public uint adminOverride;
    public SDK.CRYPT adminPassword;
    public uint aggScanTimer;
    public uint aLRS;
    public uint authServerType;
    public uint authTimeout;
    public uint autoProfile;
    public uint avgWindow;
    public uint bLRS;
    public SDK.BT_COEXIST BTcoexist;
    public SDK.CCX_FEATURES CCXFeatures;
    public string certPath;
    public uint defAdhocChannel;
    public uint DFSchannels;
    public uint displayPasswords;
    public SDK.FCC_TEST FCCtest;
    public uint fragThreshold;
    public SDK.GSHORTSLOT g_shortslot;
    public uint interferenceMode;
    public uint keepAlive;
    public uint ledUsed;
    public uint pingDelay;
    public SDK.PING_PAYLOAD pingPayload;
}
```
public uint pingTimeout;
public uint PMKcaching;
public uint polledIRQ;
public SDK.PREAMBLE preamble;
public uint probeDelay;
public uint radioState;
public uint regDomain;
public ushort Reserved;
public uint Reserved1;
public SDK.ROAM_DELTA roamDelta;
public SDK.ROAM_PERIOD roamPeriod;
public ushort roamPeriodms;
public SDK.ROAM_TRIG roamTrigger;
public uint RTSThreshold;
public SDK.RX_DIV RxDiversity;
public uint scanDFSTime;
public uint silentRunning;
public uint suppInfo;
public uint testChannel;
public SDK.TXPOWER testPower;
public SDK.BITRATE testRate;
public uint trayIcon;
public uint TTLSInnerMethod;
public SDK.TX_DIV TxDiversity;
public uint txMax;
public uint txMaxA;
public uint txTestTimeout;
public uint uAPSD;
public uint WMEenabled;
}

Elements:

- **uint adminFiles**
  - Meaning: allows import/export of settings to file
  - Value: 0 = disabled; 1 = enabled

- **uint aggScanTimer**
  - Meaning: Aggressive scan timer
  - Value: 1 = Enable

- **uint aLRS**
  - Meaning: bitmask of enabled a band channels
  - Value: Use LRD_WF_GetaLSBitmapmask and LRD_WF_GetaLSChannels to covert a list channels to/from aLRS bitmask.

- **uint authServerType**
  - Meaning: Type of authentication server radio is authenticating against
  - Value: 0 ACS (type 1), 1 SBR (type 2)

- **uint 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

- **uint autoProfile**
<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Meaning</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint avgWindow</td>
<td>RSSI moving average window</td>
<td>2 to 8</td>
</tr>
<tr>
<td>uint bLRS</td>
<td>bitmask of B channel channels.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Value: bit 0 = chan 1, bit 1 = chan 2, etc. 0x3fff or higher value indicates all channels</td>
<td></td>
</tr>
<tr>
<td>uint defAdhocChannel</td>
<td>when no beacons found this channel is used</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Value: unsigned long channel</td>
<td></td>
</tr>
<tr>
<td>uint DFSchannels</td>
<td>Use DFS channels</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Value: 1 = enabled, 0 = disabled</td>
<td></td>
</tr>
<tr>
<td>uint displayPasswords</td>
<td>indicates if passwords should be displayed in the LCM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Value: No = 0, Yes = 1</td>
<td></td>
</tr>
<tr>
<td>uint fragThreshold</td>
<td>If packet size (in bytes) exceeds the threshold, then the packet is fragmented.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Value: Any integer in the range of 256 to 2346</td>
<td></td>
</tr>
<tr>
<td>uint keepAlive</td>
<td>in CAM mode, indicates how often (in seconds) a null packet gets sent</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Value: 0 = never</td>
<td></td>
</tr>
<tr>
<td>uint ledUsed</td>
<td>LED used (for mini-module GPIO 0; requires resistor off board to make it work)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Value: desired GPIO number</td>
<td></td>
</tr>
<tr>
<td>uint pingDelay</td>
<td>Amount in time in milliseconds (ms) between successive ping requests</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Value: Any integer in the range of 0 to 7200000</td>
<td></td>
</tr>
<tr>
<td>uint pingPayload</td>
<td>Amount of data in bytes to be transmitted on a ping</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Meaning: Auto Profile enable/disable
- Value: 1 = Enable
- Mean: Bluetooth coexistence control
- Value: BT_COEXIST enum
- Meaning: CCX features
- Value: Enable = 1 or CCX radio management and AP control of TX power
- Meaning: Used to change the path of the certificate store
- Value:
- Meaning: when no beacons found this channel is used
- Value: unsigned long channel
- Meaning: Use DFS channels
- Value: 1 = enabled, 0 = disabled
- Meaning: indicates if passwords should be displayed in the LCM
- Value: No = 0, Yes = 1
- Meaning: Which fcctest is active
- Value: Off = 0, Tx = 1, Frequency = 2, Rx = 3
- Meaning: If packet size (in bytes) exceeds the threshold, then the packet is fragmented.
- Value: Any integer in the range of 256 to 2346
- Meaning: no longer in use
- Meaning: no longer in use
- Meaning: When in CAM mode, indicates how often (in seconds) a null packet gets sent
- Value: 0 = never
- Meaning: LED used (for mini-module GPIO 0; requires resistor off board to make it work)
- Value: desired GPIO number
- Meaning: Amount in time in milliseconds (ms) between successive ping requests
- Value: Any integer in the range of 0 to 7200000
- Meaning: Amount of data in bytes to be transmitted on a ping
- Value: 32, 64, 128, 256, 512, or 1024

- **uint 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

- **uint PMKcaching**
  - Meaning: standard, 1 opportunistic key caching enabled
  - Value: 1 = Enabled

- **uint polledIRQ**
  - Meaning: Intended for units that cannot share IRQs successfully
  - Value: irq used when polledIRQ is enabled

- **SDK.PREAMBLE preamble**
  - Meaning: no longer in use

- **uint probeDelay**
  - Meaning: Delay before sending out probes when APs are not located (not configured for WZC)
  - Value: 2 to 60

- **uint radioState**
  - Meaning: Determines the state of the Wi-Fi Radio.
  - Value: Enabled = 1, Disabled = 0

- **uint regDomain**
  - Meaning: Regulatory domain (for status purposes only)
  - Value: REGDOMAIN enum

- **ushort reserved**
  - Meaning: future expansion of the global config
  - Value: n/a

- **uint Reserved1**
  - Meaning: future expansion of the global config
  - Value: n/a

- **SDK.ROAM_DELTA roamDelta**
  - 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

- **SDK.ROAM_PERIOD roamPeriod**
  - 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

- **ushort 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

- **SDK.ROAM_TRIG roamTrigger**
  - 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

- **uint RTSThreshold**
  - Meaning: The packet size above which RST/CTS is required on link.
- **Value:** Any integer in the range of 0 to 2347

- **SDK.RX_DIV RxDiversity**
  - **Meaning:** Indicates how to handle antenna diversity when receiving data from the access point.
  - **Value:** RX_DIV enum

- **uint scanDFSTime**
  - **Meaning:** 20-500 ms, default of 160 ms. Maximum time spent scanning each DFS channel during a scan.
  - **Value:** unsigned long ms

- **uint silentRunning**
  - **Meaning:** Enables silent running mode (no active scans unless connected)
  - **Value:** 1 = Enabled

- **uint supInfo**
  - **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.

- **uint testChannel (Windows only; unused for Linux)**
  - **Meaning:** The current channel being tested
  - **Value:** Any channel in the range of 1 to 14

- **SDK.TXPOWER testPower (Windows only; unused for Linux)**
  - **Meaning:** Test power level
  - **Value:** Percentage in the range of 0 to 100

- **SDK.BITRATE testRate (Windows only; unused for Linux)**
  - **Meaning:** Test data rate
  - **Value:** BITRATE enum

- **uint trayIcon**
  - **Meaning:** Enabling the tray icon
  - **Value:** 1 = Enable

- **uint TTLSInnerMethod**
  - **Meaning:** The inner authentication method used by an EAP-TTLS profile
  - **Value:** 0 auto-EAP

- **SDK.TX_DIV TxDiversity**
  - **Meaning:** Indicates how to handle antenna diversity when receiving data from the access point.
  - **Value:** TX_DIV

- **uint txMax**
  - **Meaning:** Maximum transmission power
  - **Value:** on Windows: Maximum power out desired * 100; on linux: the Maximum value is in mW.

- **uint txMaxA**
  - **Meaning:** A radio - to account for high gain antennae.
  - **Value:** unsigned long %

- **uint txTestTimeout**
  - **Meaning:** Transmission test timeout in seconds; 60000 (decimal) indicates no timeout.
  - **Value:** 60000 (decimal) indicates no timeout.

- **uint uAPSD**
  - **Meaning:** bitmask of UAPSD capabilities
  - **Value:** bit 0 is voice; bit 1 is video; bit 2 is background; bit 3 is best effort

- **uint WMEenabled**
  - **Meaning:** Wifi Alliance interoperability, must be turned on for N-rates
  - **Value:** Enable = 1
LRD_WF_COMPONENT_VERSIONS
Structure LRD_WF_COMPONENT_VERSIONS is used to retrieve software name and corresponding version information. Defined for Windows CE/Mobile only.

```csharp
public struct LRD_WF_COMPONENT_VERSIONS
{
    public string componentName;
    public string componentVersion;
}
```

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

```csharp
public struct LRD_WF_SSID
{
    public byte len;
    public byte[] val;
}
```

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.

```csharp
public struct LRD_WF_SCAN_ITEM_INFO
{
    public string bssidMac;
    public SDK.LRD_WF_BSSTYPE bssType;
    public int channel;
    public uint reserved;
    public int rssi;
    public uint securityMask;
    public SDK.LRD_WF_SSID ssid;
}
```

Elements:
- **string bssidMac**
  - Meaning: the MAC address for the bssid
- **SDK.LRD_WF_BSSTYPE bssType**
  - Meaning: set to either INFRASTRUCTURE or ADHOC
- **int channel**
  - Meaning: The channel number the SSID is operating
- **uint reserved**
  - Meaning: reserved for future use
- **int rssi**
  - Meaning: The reported rssi
- **uint securityMask**
  - Meaning: A bitmask of all the supported encryption types supported by the current SSID
- **SDK.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.

```csharp
public struct LRD_WF_BSSID_LIST
{
    public SDK.LRD_WF_SCAN_ITEM_INFO[] Bssid;
    public uint NumberOfItems;
}
```

- **SDK.LRD_WF_SCAN_ITEM_INFO[] Bssid**
  - Meaning: The LRD_WF_SCAN_ITEM_INFO structure contains the information for a related SSID.
- **uint NumberOfItems**
  - Meaning: The number of scan items returned in the Bssid Array.

### 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
- **AUTHNETWORK.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.13**: 26 Mbps
- **BITRATE_14_4**: 28 Mbps
- **BITRATE_18**: 18 Mbps
- **BITRATE_19_5**: 39 Mbps
- **BITRATE_21_7**: 42 Mbps
- **BITRATE_24**: 24 Mbps
- **BITRATE_26**: 52 Mbps
- **BITRATE_28_9**: 56 Mbps
- **BITRATE_36**: 36 Mbps
- **BITRATE_39**: 78 Mbps
- **BITRATE_43_3**: 86 Mbps
- **BITRATE_48**: 48 Mbps
- **BITRATE_52**: 104 Mbps
- **BITRATE_54**: 54 Mbps
- **BITRATE_57_8**: 114 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**: Provides the state of the adapter with respect to connection statuses.

**Possible Values**:
- **CARDSTATE_NOT_INSERTED**
- **CARDSTATE_NOT_ASSOCIATED**
- **CARDSTATE_ASSOCIATED**
- **CARDSTATE_AUTHENTICATED**
- **CARDSTATE_FCCTEST**
- **CARDSTATE_NOT_SDC**
- **CARDSTATE_DISABLED**
- **CARDSTATE_ERROR**
- **CARDSTATE_AP_MODE**

### 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

DFS_CHANNELS

Description: Enables DFS settings that are available for the driver.

Possible Values:
- `DFS_OFF`
- `DFS_FULL`
- `DFS_OPTIMIZED`

DriverNDISVersion

Description: Retrieve the NDIS version from the operating system.

Possible Values:
- `NDIS0`
- `NDIS5`
- `NDIS60`
- `NDIS61`
- `NDIS62`
- `NDIS63`

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_PEAPEMSCHAP`
- `EAP_PEAPEAPGTC`
- `EAP_EAPTLS`
- `EAP_EAPTTLS`
- `EAP_PEAPTLS`
- `EAP_WAPI_CERT`

FCCTEST

Description: Possible states of the Wi-Fi radio during regulatory testing.

Possible Values:
- `FCCTEST_OFF`
FCCTEST_TX
FCCTEST_RX
FCCTEST_FREQ

GSHORTSLOT
Possible Values:
- GSHORT_AUTO
- GSHORT_OFF
- GSHORT_ON

INTERFERENCE
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
Possible Values:
- PRE_AUTO
RADIOCHIPSET

Description: Returns the Laird defined radio integrated into the operating system.

Possible Values:
- RADIOCHIPSET_NONE
- RADIOCHIPSET_SDC10
- RADIOCHIPSET_SDC15
- RADIOCHIPSET_SDC30
- RADIOCHIPSET_SDC40L
- RADIOCHIPSET_SDC40NBT
- RADIOCHIPSET_SDC45
- RADIOCHIPSET_SDC50

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 Laird 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: Laird 802.11g radio (supports 802.11b and 802.11g)
- RADIOTYPE_ABG: Laird 802.11a/g radio (supports 802.11a, 802.11b, and 802.11g)
- RADIOTYPE_NBG: Laird 802.11nb/g radio (802.11b, and 802.11g and 802.11n)
- **RADIOTYPE_NABG**: Laird 802.11n radio (supports 802.11a, 802.11b, 802.11g, and 802.11n)
- **RADIOTYPE_NOT_SDC**: Not a Laird SDC radio
- **RADIOTYPE_NOT_SDC_1**: Reserved
- **RADIOTYPE_AR_BG**: (30 Series) Laird 802.11g radio (supports 802.11b and 802.11g)
- **RADIOTYPE_AR_ABG**: (30 Series) Laird 802.11a/g radio (supports 802.11a, 802.11b, and 802.11g)
- **RADIOTYPE_AR_NBG**: (30 Series) Laird 802.11nb/g radio (802.11b, and 802.11g and 802.11n)
- **RADIOTYPE_AR_NABG**: (30 Series) Laird 802.11n radio (supports 802.11a, 802.11b, 802.11g, and 802.11n)

### 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_TELC**: Telecom Engineering Center; the standards body for Japan
- **REG_WW**: Worldwide domain; enables the radio to be used in any domain
- **REG_KCC**: Korea

### 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
- **RDELTA_40**: 40 dBm
- **RDELTA_45**: 45 dBm
- **RDELTA_50**: 50 dBm
- **RDELTA_55**: 55 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**: 5 ms
- **RPERIOD_10**: 10 ms (Default)
- **RPERIOD_15**: 15 ms
- **RPERIOD_20**: 20 ms
• 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: Returns an error type for a given operation.

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**

Defines

### 802.11 Reason Codes

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

<table>
<thead>
<tr>
<th>Reason</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOT11_RC_RESERVED</td>
<td>0</td>
<td>Reserved</td>
</tr>
<tr>
<td>DOT11_RC_UNSPECIFIED</td>
<td>1</td>
<td>Indicates an unspecified reason.</td>
</tr>
<tr>
<td>DOT11_RC_AUTH_INVAL</td>
<td>2</td>
<td>Indicates that the previous authentication is no longer valid.</td>
</tr>
<tr>
<td>DOT11_RC_DEAUTH_LEAVING</td>
<td>3</td>
<td>Indicates a deauthentication because the sending station is leaving (or has left) IBSS or ESS.</td>
</tr>
<tr>
<td>DOT11_RC_INACTIVITY</td>
<td>4</td>
<td>Indicates a disassociation due to inactivity.</td>
</tr>
<tr>
<td>DOT11_RC_BUSY</td>
<td>5</td>
<td>Indicates a disassociation because the AP is unable to handle all currently associated stations.</td>
</tr>
<tr>
<td>DOT11_RC_INVAL_CLASS_2</td>
<td>6</td>
<td>Indicates that a Class 2 frame was received from a nonauthenticated station.</td>
</tr>
<tr>
<td>DOT11_RC_INVAL_CLASS_3</td>
<td>7</td>
<td>Indicates that a Class 3 frame was received from a nonauthenticated station.</td>
</tr>
<tr>
<td>DOT11_RC_DISASSOC_LEAVING</td>
<td>8</td>
<td>Indicates a disassociation because the sending station is leaving (or has left) BSS.</td>
</tr>
<tr>
<td>Reason</td>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DOT11_RC_NOT_AUTH</td>
<td>9</td>
<td>Indicates that the station that is requesting (re)association is not authenticated with the responding station.</td>
</tr>
<tr>
<td>DOT11_RC_BAD_PC</td>
<td>10</td>
<td>Indicates an unacceptable power capability element.</td>
</tr>
<tr>
<td>DOT11_RC_BAD_CHANNELS</td>
<td>11</td>
<td>Indicates an unacceptable supported channels element.</td>
</tr>
<tr>
<td>DOT11_RC_BSS_TRANSIT_MGMT</td>
<td>12</td>
<td>Indicates a disassociation due to BSS Transition Management.</td>
</tr>
<tr>
<td>DOT11_RC_INVALID_WPA_IE</td>
<td>13</td>
<td>Indicates an invalid info. element.</td>
</tr>
<tr>
<td>DOT11_RC_MIC_FAILURE</td>
<td>14</td>
<td>Indicates a Michael failure.</td>
</tr>
<tr>
<td>DOT11_RC_4WH_TIMEOUT</td>
<td>15</td>
<td>Indicates a four-way handshake timeout.</td>
</tr>
<tr>
<td>DOT11_RC_GTK_UPDATE_TIMEOUT</td>
<td>16</td>
<td>Indicates a group key update timeout.</td>
</tr>
<tr>
<td>DOT11_RC_WPA_IE_MISMATCH</td>
<td>17</td>
<td>Indicates that a WPA IE in a four-way handshake differs from a (re)association request/probe response.</td>
</tr>
<tr>
<td>DOT11_RC_INVALID_MC_CIPHER</td>
<td>18</td>
<td>Indicates an invalid multicast cipher.</td>
</tr>
<tr>
<td>DOT11_RC_INVALID_UC_CIPHER</td>
<td>19</td>
<td>Indicates an invalid unicast cipher.</td>
</tr>
<tr>
<td>DOT11_RC_INVALID_AKMP</td>
<td>20</td>
<td>Indicates an invalid authenticated key management protocol.</td>
</tr>
<tr>
<td>DOT11_RC_BAD_WPA_VERSION</td>
<td>21</td>
<td>Indicates an unsupported WPA version.</td>
</tr>
<tr>
<td>DOT11_RC_INVALID_WPA_CAP</td>
<td>22</td>
<td>Indicates invalid WPA IE capabilities.</td>
</tr>
<tr>
<td>DOT11_RC_8021X_AUTH_FAIL</td>
<td>23</td>
<td>Indicates an 802.1X authentication failure.</td>
</tr>
<tr>
<td>DOT11_RC_UNSPECIFIED_QOS</td>
<td>32</td>
<td>Indicates an unspecified QoS-related reason.</td>
</tr>
<tr>
<td>DOT11_RC_INSUFFICIENT_BW</td>
<td>33</td>
<td>Indicates that the QoS AP lacks sufficient bandwidth for this QoS station.</td>
</tr>
<tr>
<td>DOT11_RC_EXCESSIVE_FRAMES</td>
<td>34</td>
<td>Indicates that excessive frames need to be acknowledged due to AP transmissions or poor channel conditions.</td>
</tr>
<tr>
<td>DOT11_RC_TX_OUTSIDE_TXOP</td>
<td>35</td>
<td>Indicates that the station is transmitting outside the limits of its TXOPs.</td>
</tr>
<tr>
<td>DOT11_RC_LEAVING_QBSS</td>
<td>36</td>
<td>Indicates a request from the peer station as the station is leaving the BSS (or resetting).</td>
</tr>
<tr>
<td>DOT11_RC_BAD_MECHANISM</td>
<td>37</td>
<td>Indicates a request from the peer station that it does not want to use the mechanism.</td>
</tr>
<tr>
<td>DOT11_RC_SETUP_NEEDED</td>
<td>38</td>
<td>Indicates a request from the peer station that the station received frames using the mechanism that require setup.</td>
</tr>
<tr>
<td>DOT11_RC_TIMEOUT</td>
<td>39</td>
<td>Indicates a request from the peer station that there was a timeout.</td>
</tr>
</tbody>
</table>