UMTS SDK

Developer's Guide

2130143
Rev 6.0

SIERRA WIRELESS
HEART OF THE WIRELESS MACHINE®

Important Notice

Due to the nature of wireless communications, transmission and reception of data can never be guaranteed. Data may be delayed, corrupted (i.e., have errors) or be totally lost. Although significant delays or losses of data are rare when wireless devices such as the Sierra Wireless modem are used in a normal manner with a well-constructed network, the Sierra Wireless modem should not be used in situations where failure to transmit or receive data could result in damage of any kind to the user or any other party, including but not limited to personal injury, death, or loss of property. Sierra Wireless accepts no responsibility for damages of any kind resulting from delays or errors in data transmitted or received using the Sierra Wireless modem, or for failure of the Sierra Wireless modem to transmit or receive such data.

Safety and Hazards

Do not operate the Sierra Wireless modem in areas where blasting is in progress, where explosive atmospheres may be present, near medical equipment, near life support equipment, or any equipment which may be susceptible to any form of radio interference. In such areas, the Sierra Wireless modem **MUST BE POWERED OFF**. The Sierra Wireless modem can transmit signals that could interfere with this equipment.

Do not operate the Sierra Wireless modem in any aircraft, whether the aircraft is on the ground or in flight. In aircraft, the Sierra Wireless modem **MUST BE POWERED OFF**. When operating, the Sierra Wireless modem can transmit signals that could interfere with various onboard systems.

Note: Some airlines may permit the use of cellular phones while the aircraft is on the ground and the door is open. Sierra Wireless modems may be used at this time.

The driver or operator of any vehicle should not operate the Sierra Wireless modem while in control of a vehicle. Doing so will detract from the driver or operator's control and operation of that vehicle. In some states and provinces, operating such communications devices while in control of a vehicle is an offence.

Limitation of Liability

The information in this manual is subject to change without notice and does not represent a commitment on the part of Sierra Wireless. SIERRA WIRELESS AND ITS AFFILIATES SPECIFICALLY DISCLAIM LIABILITY FOR ANY AND ALL DIRECT, INDIRECT, SPECIAL, GENERAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES INCLUDING, BUT NOT LIMITED TO, LOSS OF PROFITS OR

REVENUE OR ANTICIPATED PROFITS OR REVENUE ARISING OUT OF THE USE OR INABILITY TO USE ANY SIERRA WIRELESS PRODUCT, EVEN IF SIERRA WIRELESS AND/OR ITS AFFILIATES HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES OR THEY ARE FORESEEABLE OR FOR CLAIMS BY ANY THIRD PARTY.

Notwithstanding the foregoing, in no event shall Sierra Wireless and/or its affiliates aggregate liability arising under or in connection with the Sierra Wireless product, regardless of the number of events, occurrences, or claims giving rise to liability, be in excess of the price paid by the purchaser for the Sierra Wireless product.

Patents

Portions of this product may be covered by some or all of the following US patents:

5,515,013	5,629,960	5,845,216	5,847,553	5,878,234
5,890,057	5,929,815	6,169,884	6,191,741	6,199,168
6,339,405	6,359,591	6,400,336	6,516,204	6,561,851
6,643,501	6,653,979	6,697,030	6,785,830	6,845,249
6,847,830	6,876,697	6,879,585	6,886,049	6,968,171
6,985,757	7,023,878	7,053,843	7,106,569	7,145,267
7,200,512	7,295,171	7, 287,162	D442,170	D459,303
D599,256	D560,911			

and other patents pending.

This product includes technology licensed from: QUALCOMM® 3G

Licensed by QUALCOMM Incorporated under one or more of the following United States patents and/or their counterparts in other nations:

4,901,307	5,056,109	5,101,501	5,109,390	5,228,054
5,267,261	5,267,262	5,337,338	5,414,796	5,416,797
5,490,165	5,504,773	5,506,865	5,511,073	5,535,239
5,544,196	5,568,483	5,600,754	5,657,420	5,659,569
5,710,784	5,778,338			

Manufactured or sold by Sierra Wireless or its licensees under one or more patents licensed from InterDigital Group.

Copyright

Trademarks

AirCard® and "Heart of the Wireless Machine®" are registered trademarks of Sierra Wireless. Watcher® is a trademark of Sierra Wireless, registered in the European Community. Sierra Wireless, the Sierra Wireless logo, the red wave design, and the red-tipped antenna are trademarks of Sierra Wireless.

Contact Information

Sales Desk:	Phone:	1-604-232-1488
	Hours:	8:00 AM to 5:00 PM Pacific Time
	E-mail:	sales@sierrawireless.com
Technical Support	Web	www.sierrawireless.com/corporate/contact.aspx
Post:	Sierra Wireless 13811 Wireless Way Richmond, BC Canada V6V 3A4	
Fax:	1-604-231-1109	
Web:	www.sierrawireless.com	

Consult our website for up-to-date product descriptions, documentation, application notes, firmware upgrades, troubleshooting tips, and press releases:

www.sierrawireless.com

Revision History

Revision number	Release date	Changes
6.0	Apr 2008	Created

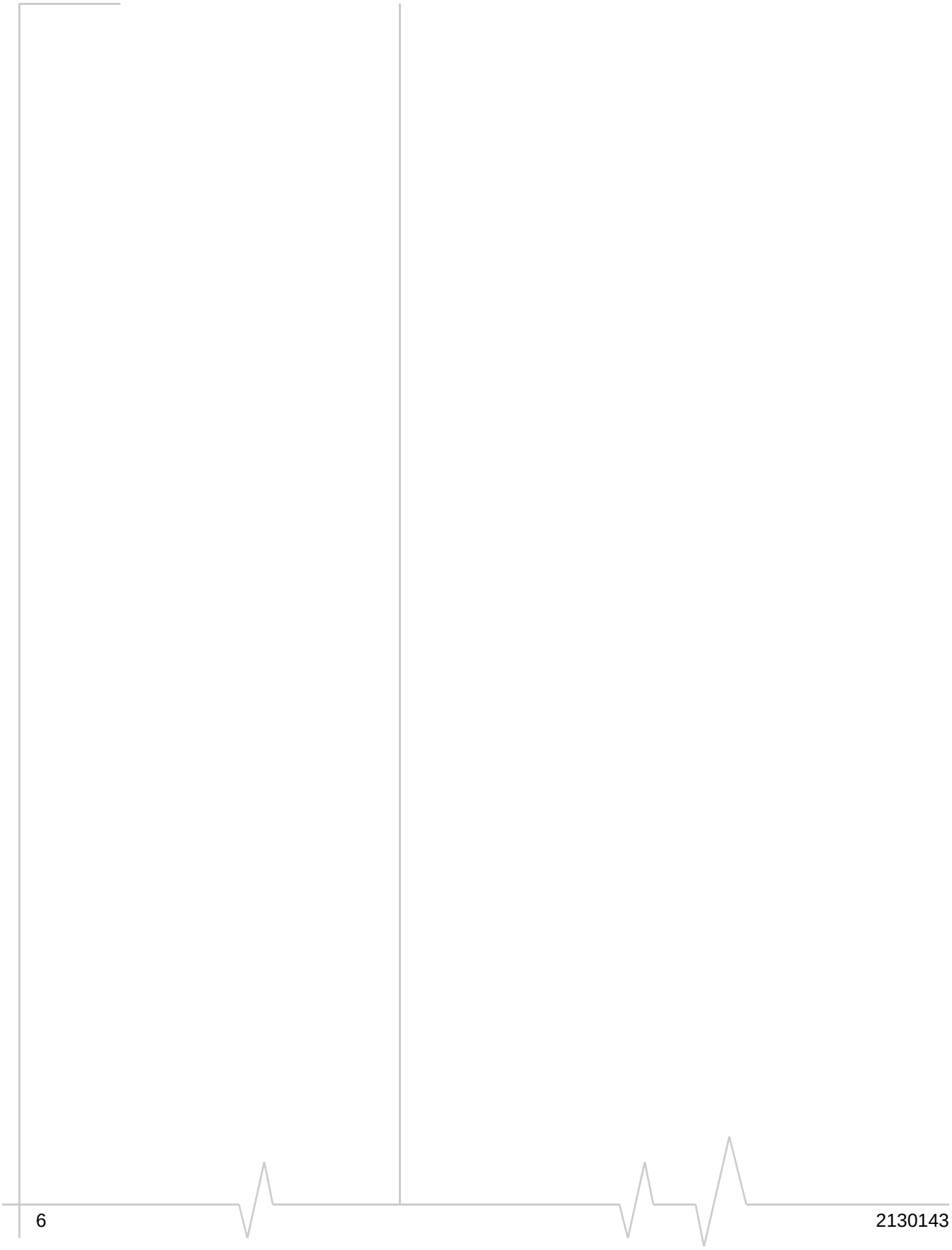
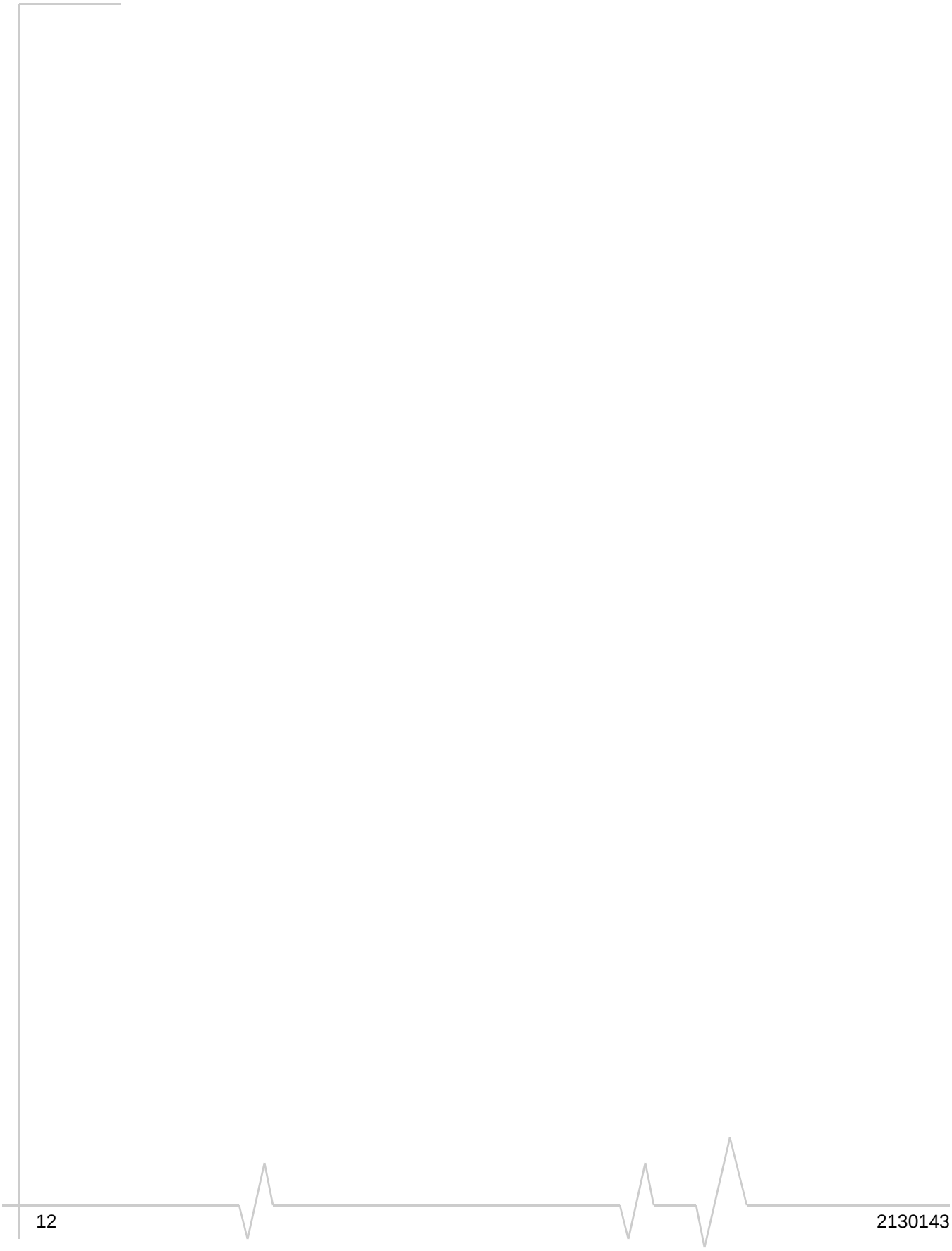


Table of Contents



1: About This Guide

1

SDK introduction

The Software Development Kit (SDK) allows Windows® software developers to create applications for Sierra Wireless' UMTS products.

The SDK includes:

- This document (the SDK Developer's Guide)
- Application Programming Interface (API) with supporting online documentation describing API modules, classes, and files
- Sample programs showing how to use several API calls
- End User License Agreement (EULA) — You are required to accept the terms of this EULA before distributing any products developed using this SDK.

Scope of this guide

This guide describes system requirements for installing and using the SDK, and performing typical tasks from the modem's feature set. It is current with the SDK it is bundled with.

It does not describe the use of the AT, USB, or other interfaces. Refer to the appropriate product-specific references available at www.sierrawireless.com.

System requirements

Development systems:

- O/S: Microsoft Windows® XP (SP1 or later), Vista®, or Vista (SP1)
- Languages: Any that can load a.dll file and call exported functions.

Client systems:

- O/S: Microsoft Windows® XP (SP1 or later) or Vista. For assistance with Windows CE, contact Sierra Wireless Technical Support. (See "Contact Information" on page 5.)

Other reference material

The following are some sources for additional references that may help you use the SDK to develop your applications:

- Sierra Wireless (www.sierrawireless.com) — Product specifications for your products, interface references (AT, CnS), glossary of terms and acronyms, etc.
- Microsoft Developer's Network (MSDN) library — Information explaining data types and syntax used in this guide.
- GSM Association (www.gsmworld.com)
- Palowireless Resource Control (www.palowireless.com) — Links to sources for GPRS/EDGE/UMTS articles, white papers, definitions, etc.

2: R4 API to SwiApiX API differences

2

Applications written using the R4 API will require modifications to work with the SwiApiX API. These modifications are the result of the following API changes:

- API initialization functions
  - Multiple applications can now access multiple modems simultaneously
  - Improved device detection and connection during application startup, modem resets, and system suspend/resume cycles
- Standardization of API component names and content
  - Functions, data structures, and enumerations

Refer to Table 2-1, Table 2-2, and for lists of enumerations, data structures, and functions that have been replaced, updated, or added in the SwiApiX API.

Table 2-1: Enumerations

R4 enumeration	SwiApiX enumeration	Comments
SWI_TYPE_Device	SWI_TYPE_Device	Includes new device types, and obsolete device types have been removed.
SWI_TYPE_Notify	SWI_TYPE_Notify	Includes new notifications, and obsolete notifications have been removed.
SWI_TYPE_CallError	SWI_TYPE_GSM_CallError	No content changes. Name changed for standardization.
SWI_TYPE_CallType	SWI_TYPE_GSM_CallType	
SWI_TYPE_CallState	SWI_TYPE_GSM_CallState	
n/a	SWI_TYPE_WirelessTech	New enumerations related to improved device detection.
	SWI_TYPE_ProductClass	
	SWI_TYPE_LockServ	New enumeration related to multiple application support.

Table 2-2: Data structures

R4 structure	SwiApiX structure	Comments
SwiNotifyVariant	SwiNotifyVariant	Includes new definitions, and obsolete definitions have been removed.

Table 2-2: Data structures (Continued)

R4 structure	SwiApiX structure	Comments
SWI_STRUCT_SMS_SendStatus	SWI_STRUCT_GSM_SMS_SendStatus	No content changes. Name changed for standardization.
SWI_STRUCT_SpeakerVolume	SWI_STRUCT_GSM_SpeakerVolume	
SWI_STRUCT_SpeakerMute	SWI_STRUCT_GSM_SpeakerMute	
SWI_STRUCT_NetworkStatus	SWI_STRUCT_GSM_NetworkStatus	
n/a	SWI_STRUCT_AirServer	New structure related to improved device detection.
	SWI_STRUCT_AirServerChange	
	SWI_STRUCT_AirServerExtended	
	SWI_STRUCT_AirServerList	
	SWI_STRUCT_ApiStartup	New structure related to multiple application support.
	SWI_STRUCT_LockAirServer	

Table 2-3: Functions

R4 function	SwiApiX function	Comments
SwiApiOpen	SwiApiStartup	Replaced for multiple application support
SwiApiClose	SwiApiShutdown	
SwiGetIMSI	SwiGetGsmIMSI	No content changes. Name changed for standardization.
SwiSetSpeakerVolume	SwiSetGsmSpeakerVolume	
SwiGetSpeakerVolume	SwiGetGsmSpeakerVolume	
SwiSetSpeakerMute	SwiSetGsmSpeakerMute	
SwiGetSpeakerMute	SwiGetGsmSpeakerMute	
n/a	SwiGetAvailAirServers	New function related to improved device detection.
	SwiSelectAirServer	
	SwiGetAirServerinfo	

3: Using the API documentation

3

The SDK includes on-line API documentation that details the different API modules, data constructs, function calls, etc.

Table 3-4 provides a quick reference to locating information in the API documentation, and Table 3-5 is a listing of available API modules (groups of related commands).

Table 3-4: Locations of information in API documentation

To find this ...	Look in this area of the documentation
API modules	Modules
Data structures	Classes > Class List
'class members'	Classes > Class Members
Header files	Files > File List
Functions	Files > File Members > Functions
Typedefs	Files > File Members > Typedefs
Enumerations	Files > File Members > Enumerations
Enumerators	Files > File Members > Enumerator
Defines	Files > File Members > Defines

Table 3-5: API Modules

Module
API Management and Device Selection
Notifications
Modem Information and Management
Error Handling
Network Management and Status
Audio Profiles
Radio Power and Status
Location Based Services
Driver
Connection Profiles
Call Management and Voice Features

Table 3-5: API Modules (Continued)

Module
Phonebook
SIM
Supplementary Services
SMS

4: SDK Installation and Setup

The SDK is delivered as a .zip file by Sierra Wireless Technical Support. (See "Contact Information" on page 5.)

Installing the SDK

Note: Installing the SDK over an older version is not recommended.

If you are upgrading from a previous version of the SDK, uninstall and delete all the files in the SDK folder ($INSTALL_FOLDER from Step 2), then install the later version.

To install the SDK:

1. Open the file in WinZip.
2. Extract the contents to a destination folder (any folder you choose. For example, **\Program Files\Sierra Wireless Inc**). This is referred to as $INSTALL_FOLDER through the remainder of this chapter.

All SDK components (header files, documentation, libraries, drivers, etc.) install under the $INSTALL_FOLDER in meaningfully-named folders.

Setting up your environment

Set your development environment to locate the files required for your host application's platform and operating system. These files (.dll, .lib, .h) are found in folders below the $INSTALL_FOLDER.

Installing the device driver

To install the Sierra Wireless device driver for the modem:

1. Download the driver installer (an .exe or .msi installable package) from www.sierrawireless.com/developers/downloads.asp.
2. Run the installer package.

Distributing files

When you distribute your host application to a user, include:

- The following files in the same folder as your application: SwiApiInterface.dll, SwiCardDirect.dll, SwiApiMux.exe, and devices.xml.
- The Sierra Wireless driver installer package.

Dynamic Link Library (DLL) usage

To use the .dll supplied with the SDK:

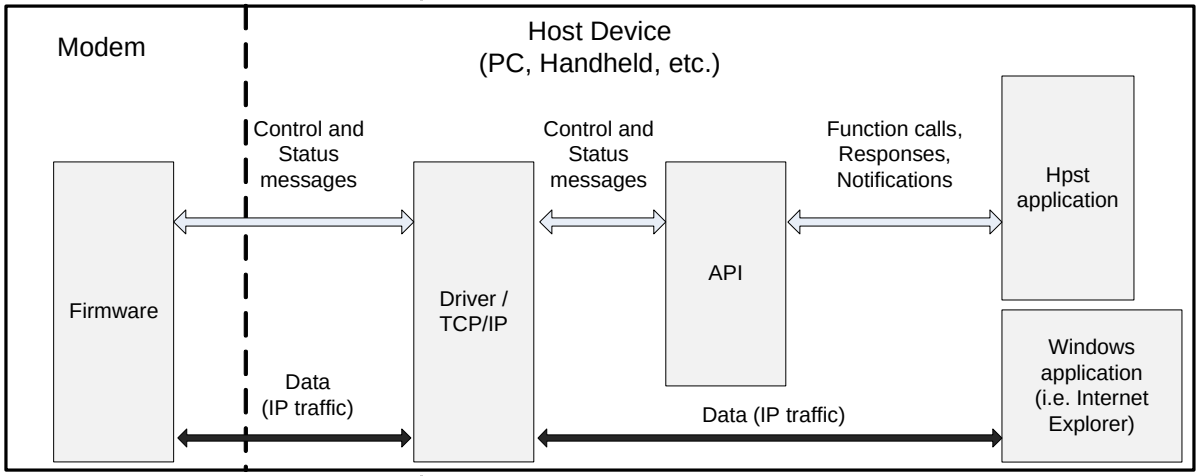
- Include the header files in your host applications when compiling.
- The API returns strings in UNICODE format.

5: Software architecture

This chapter describes how the API interfaces with the host application and modem, and how the modem driver interfaces with Windows.

Software architecture

Figure 5-1: Software layers



The API facilitates the exchange of control and status messages between your 'host application' and the modem. These messages pass through the four software layers described below.

Host application layer

This is the client application that you develop to control the modem and present a user interface. Your application communicates with the modem using the functions and data structures of the API.

Note: You can also use other interfaces (AT, CnS, NMEA, etc.) over serial COM ports, USB ports, etc., to communicate directly with the modem. See the appropriate product-specific references available at www.sierrawireless.com.

Modem driver layer

Modem driver software is installed on the host PC to provide the interface between the modem and the Windows operating system.

Firmware layer

The firmware manages data traffic between the modem and the cellular network. It can be upgraded as new releases become available. (Firmware releases are posted on the Sierra Wireless web site, www.sierrawireless.com as part of new software downloads.)

API layer

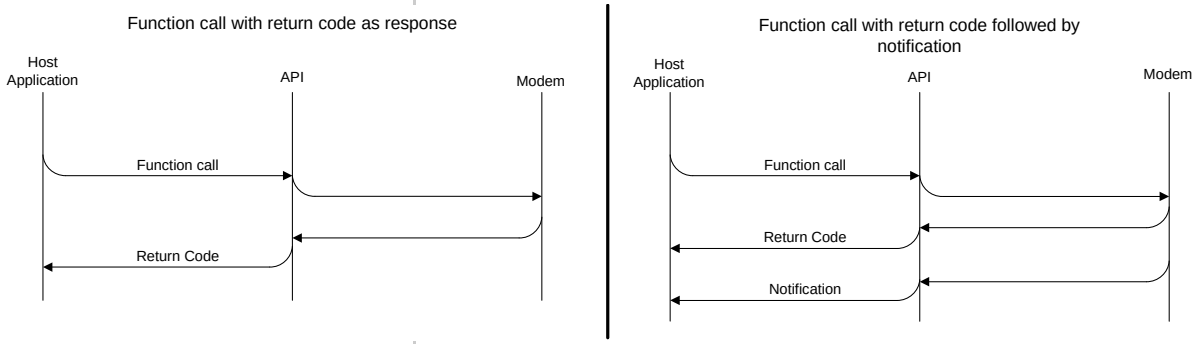
Note: Data traffic is not handled by the API. All data traffic is handled by a Windows application (like Internet Explorer) communicating directly with Winsock libraries or the TCP stack.

The API layer includes several types of files (.dll, .xml, .exe, etc.) on the host system. These files provide functions and data structures that your host application uses to communicate with the modem. This communication takes the form of:

- Host-initiated requests/responses (symmetric notifications)
- Modem-initiated notifications (asymmetric notifications)

Figure 5-2 illustrates the data flow patterns of a function that receives only a return code, and a function that receives a return code and a symmetric notification.

Figure 5-2: Request - response patterns



Host-initiated requests / responses

API function calls have the following characteristics:

- Requests — Host-to-modem function calls passed to the modem via the modem driver. Each function call returns a standard return code (see SwiRcodes.h in the API documentation).
- Responses (symmetric notifications) — Some function calls receive a response (modem-to-host message) containing additional information after the return code is sent. The API documentation for each call indicates if such notifications will be received.
  The API indicates if you have to wait for the notification (or for it to time out) before calling related functions. For example, the host may have to wait for a SIM notification before calling additional SIM functions, but can call other functions (such as LBS) while it waits.
- Timeouts — Each call requires a non-zero timeout value. If the modem doesn't respond to the request in time, the API returns a timeout error code to your application; if the modem responds after the function times out, the API ignores the response. (Make sure that you set your timeout values appropriately — 3000 ms is recommended as an initial setting, which you can adjust during development.)

Note: Where possible, use available notifications (asymmetric) to monitor modem status, rather than polling with function calls.
For example, register to receive the notification SWI_NOTIFY_NetworkStatus rather than frequently calling the function SwiGetServiceStatusEx.

Modem-initiated notifications

Asymmetric notifications have the following characteristics:

- The host application must explicitly register to receive most desired notifications, otherwise the messages are ignored. The only ones that do not have to be registered are API-driven notifications (such as AirServerChange). Refer to the API documentation for details of all available notifications.
- These notifications occur when specific system state changes occur (for example, available networks, modem temperature, etc.).

Note: Notifications are the preferred method for managing the modem. This lets the modem operate at optimum power savings by not continually polling the modem for responses.

Note: When the host registers a specific notification type, a notification is generated when the current state information first becomes available.

Managing notifications

To manage notifications from the API, your application must:

- Register a callback function(s): Use **SwiRegisterCallback** to register a function that takes action on all event types that your application enables for notification. The API calls this function through a separate thread created in the API layer.
- Use appropriate notification API calls to enable the notifications required by your application. (You can also disable

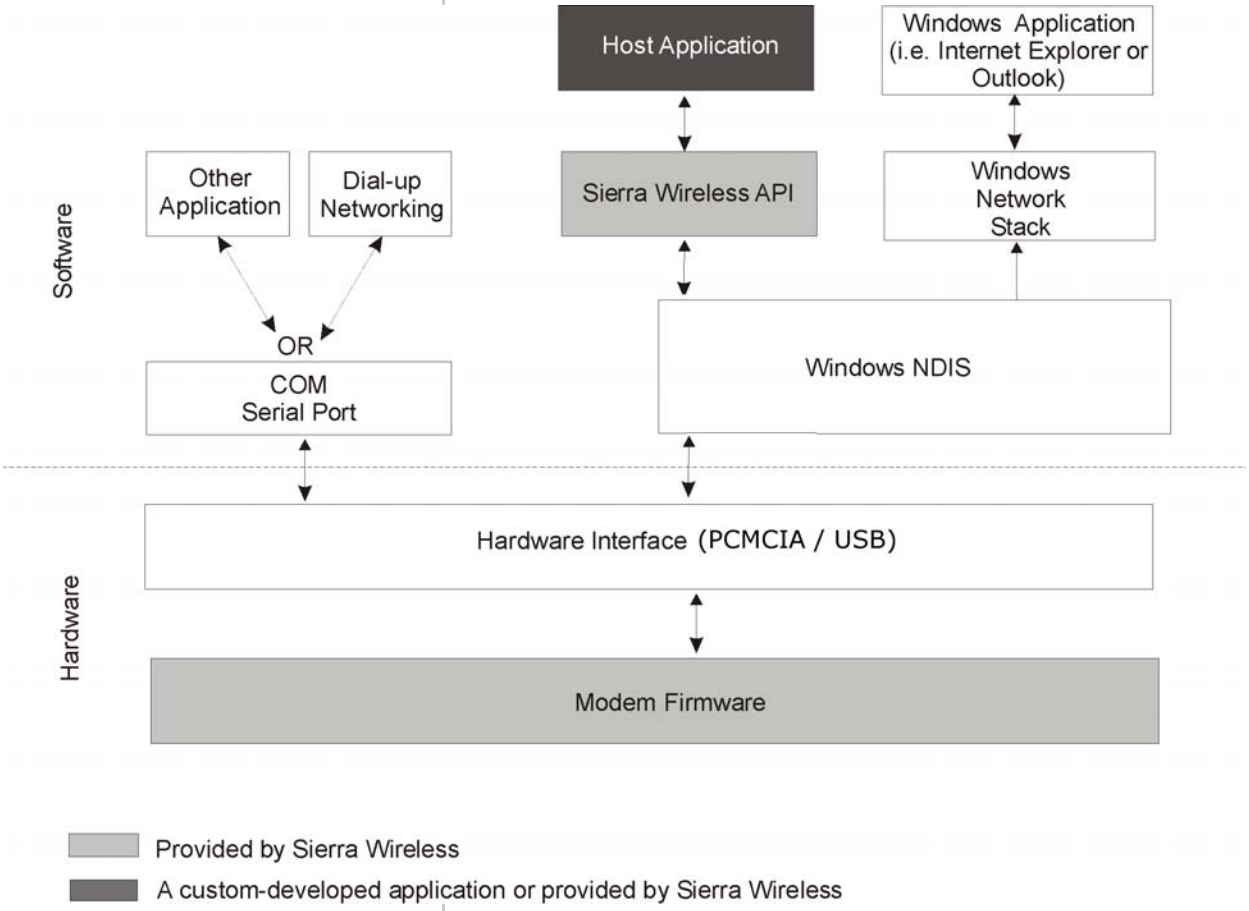
notifications if necessary.) See "Handling notifications" on page 31 for details.

Note: If the modem resets unexpectedly, notifications are disabled. The host applications must detect this condition and explicitly re-enable the notifications.

Interaction between components

The figure below shows the interaction between the API and other software and hardware components.

Figure 5-3: API , software, and hardware interaction



6: API Initialization, Device Management, and Notifications

Note: 'Air servers", in the context of this document, are IP tunnels used by the modem to communicate with remote networks.

This chapter describes how to use API functions to perform the following tasks:

- Initialize or shut down the API sub-system
- Identify available air servers, get information on them, and connect to a specific server
- Handle modem suspend/resume, and resets
- Prepare the host application to work with the API, including handling notifications

The API supports simultaneous access to multiple modems by multiple applications developed using this API.

Using these API functions

The following are key usage notes for functions described in this chapter (see the API documentation for additional details):

- **SwiApiStartup**: Call this function first to initialize the API sub-system before using any other API calls.
  - Once initialized, you can use control and status API calls.
  - You do not have to call it again when a device is removed.
  - Do not call it again unless it fails or you receive a shutdown notification. If this happens, call **SwiApiShutdown**, and then call **SwiApiStartup** to start again.
- **SwiGetAvailAirServers**: This can successfully return an empty list when there are no available air servers.
- **SwiSelectAirServer**: If you call this while you are connected to an air server, all pending requests are cancelled and the modem binds to the new air server.
- **SwiRegisterCallback**: Always call this function immediately after you register (or re-register) with an air server.
- **SwiNotify**: Enable each notification that the callback should handle (see SwiStructs.h in the API documentation for the list of available notifications). If the modem resets, you must re-enable the notifications after you re-register the callback function.
- **SwiApiShutdown**: Always call when finished. This stops the API sub-system, drops communication links with servers, and cleans up resources.

Examples

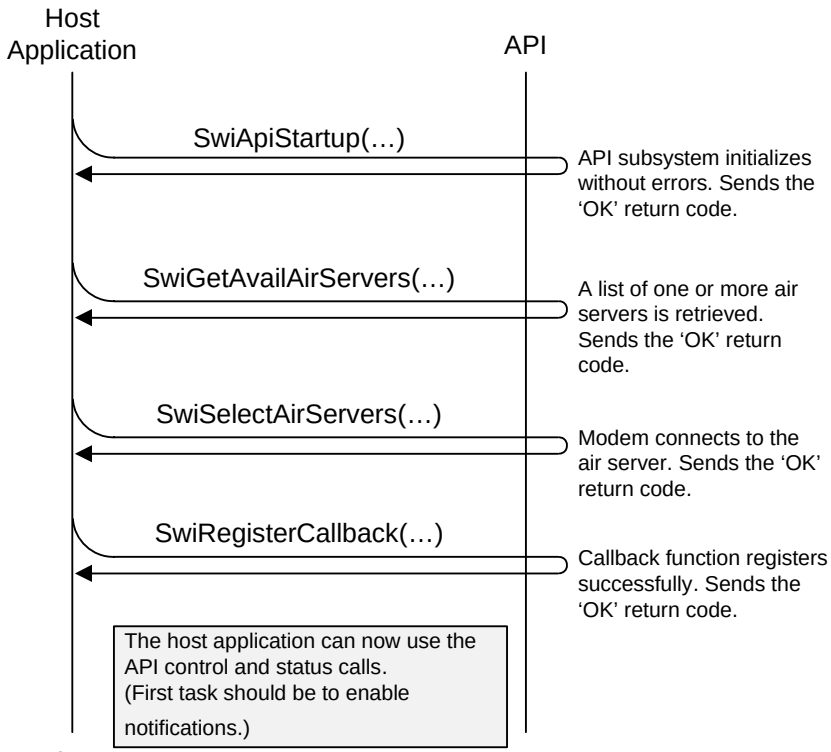
The following flow diagrams illustrate common situations addressed using these modules.

Initializing the API

When air servers are available Initialize the API and bind the modem to a specific air server.

1. Call **SwiApiStartup** to initialize the API (enabling device detection).
2. Call **SwiGetAvailAirServers** to get a list of all available air servers.
3. Call **SwiSelectAirServers** to choose the server to bind to.
4. Call **SwiRegisterCallback** to register a callback function to receive API notifications.

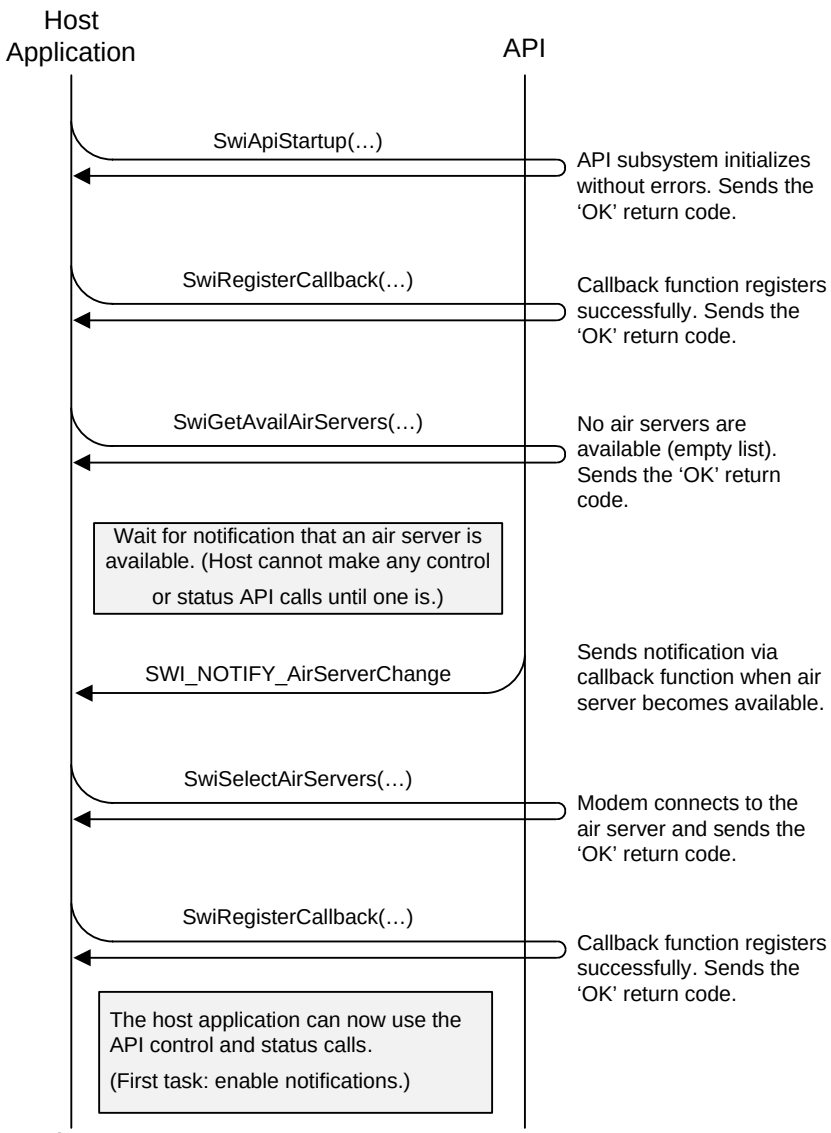
Figure 6-4: API initialization when air servers are available



When no air servers are available Initialize the API and wait for notification that a server has become available. Then bind the modem to that specific air server.

1. Call **SwiApiStartup** to initialize the API (enabling device detection).
2. Call **SwiRegisterCallback** to register a callback function to receive API notifications.
3. Call **SwiGetAvailAirServers** to get a list of all available air servers, and wait for notification that a server is available (**SWI_NOTIFY_AirServerChange**). If no air servers are available, the modem returns an empty list.
4. Call **SwiSelectAirServers** to bind to the server.
5. Call **SwiRegisterCallback** again to re-register the callback function.

Figure 6-5: API initialization when no air servers are available.



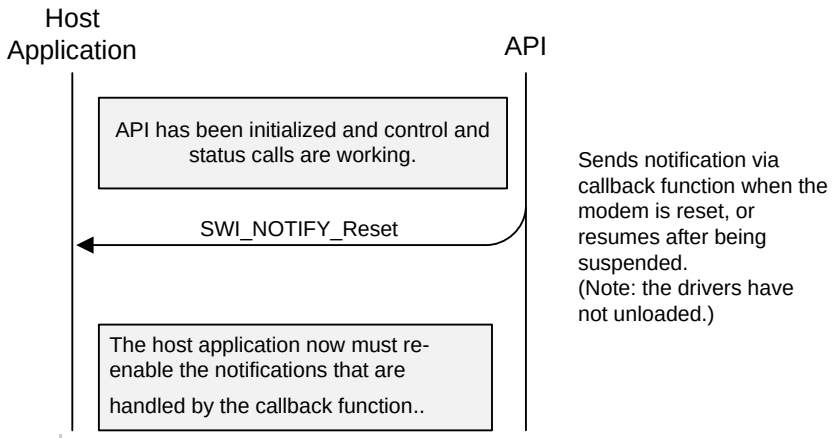
Handling a modem reset or suspend/ resume cycle

When the modem is reset, or after it is suspended and then resumes, the modem drivers may be unloaded and reloaded, depending on the modem's factory configuration.

If the drivers do not unload:

1. The host receives SWI_NOTIFY_Reset when the modem resets. The modem drivers have remained loaded, and the callback function remains registered.
2. Re-enable the API notifications that are handled by the callback function.

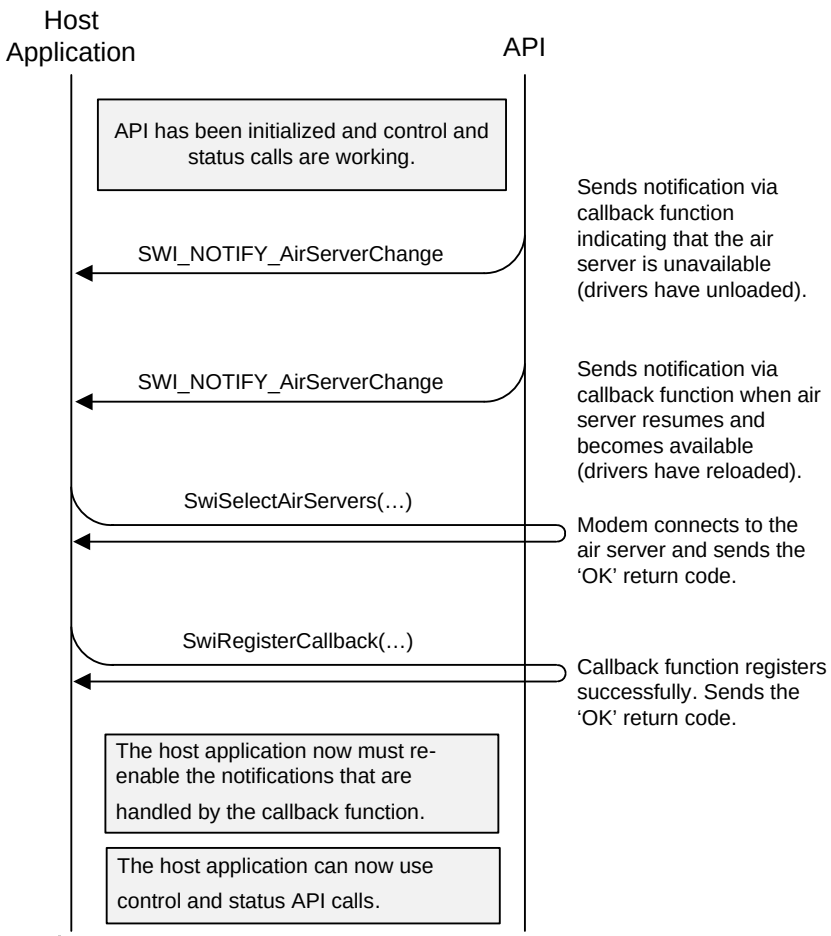
Figure 6-6: Handling a modem reset or suspend/resume cycle when drivers do not unload



If the drivers unload and reload:

1. The host receives a SWI_NOTIFY_AirServerChange notification indicating the drivers have unloaded.
2. The host must wait for another SWI_NOTIFY_AirServerChange notification indicating the drivers have reloaded.
3. Call **SwiSelectAirServers** to bind to a server.
4. Call **SwiRegisterCallback** to re-register the callback function to receive API notifications.
5. Re-enable the API notifications that are handled by the callback function.

Figure 6-7: Handling modem reset or suspend/resume cycle when drivers reload

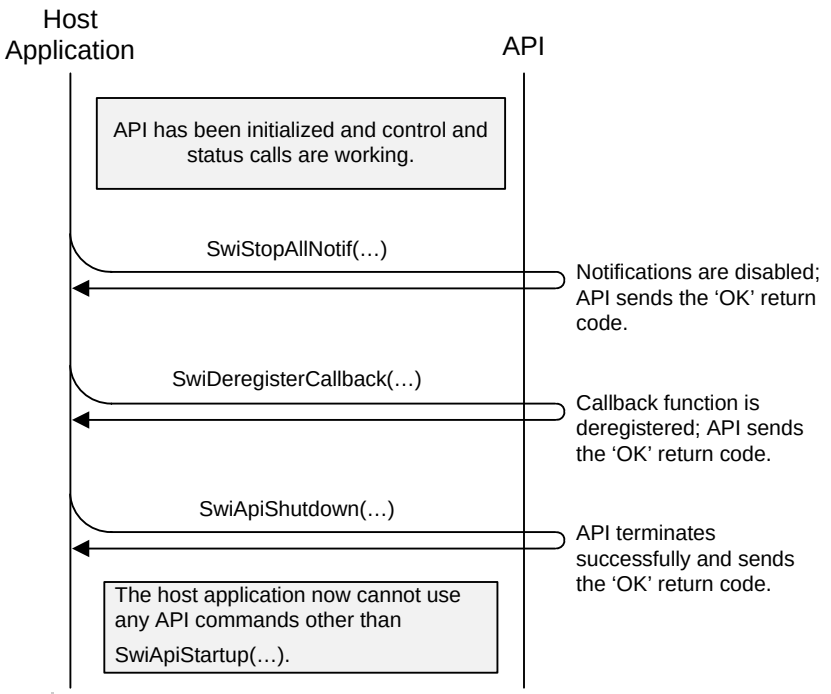


Shutting down the API

When ready to shut down the API, disable notifications, deregister the callback function, and then shutdown the API.

1. Call **SwiStopAllNotif** to disable all notifications.
2. Call **SwiDeRegisterCallback** so the API will stop using the callback function.
3. Call **SwiApiShutdown** to clean up resources, drop communication links, and stop the API.

Figure 6-8: Shutting down the API



Host application API usage

Opening the host application

Typically, on start-up, the host application should:

1. Initialize the API and register with an air server as shown in Figure 6-4 on page 26.
2. Call **SwiSetHostStartup** to ensure the modem is able to register on a network.

Checking that the modem is available

To check if the modem is available (driver has been loaded), call **GetAvailAirServers**.

To make sure the modem is ready to use (not in boot and hold mode, for example), call any API function that returns a modem parameter. If the function fails to return a value within a reasonable number of attempts, the modem is not available.

Powering the modem down and up

Use the following functions to power the modem down and up:

- **SwiGetModemDisable** — Indicates the current modem state (in low power mode or not).
- **SwiSetModemDisable** — Enables or disables the modem. If you disable the modem with this function, you must also use the function to re-enable it — resetting the modem does not re-enable it. (Modem disable setting is persistent across modem power cycles.)
- **SwiSetHostStartup** — Enables or disables the modem. (Powered-down state is not-persistent across modem power cycles.)

*Note: Refer to the API documentation to determine whether your host application should use **SwiSetModemDisable** or **SwiSetHostStartup**.*

Handling notifications

Enabling notifications

Call **SwiEnableNotif**, specifying the notifications required for your application.

You must have a registered callback function before you can receive any of these notifications.

Disabling notifications

Call function **SwiStopNotify** to disable individual notifications, or **SwiStopAllNotif** to disable all notifications.

Processing notifications

When an enabled event notification occurs:

1. API calls the registered callback function, passing the event data in the proper structure to the host application.

Note: When you receive a notification, cache the information passed by the notification and release the thread as soon as possible. Processing the notification in the context of the API notification thread will cause unspecified behavior in the API.

Managing modem resets

The notification **SWI_NOTIFY_Reset** is received when a modem reset occurs.

This notification always remains enabled.

Using the "ready" notifications

The following notifications are received when services become available:

- **SWI_NOTIFY_PlmnReady**—Modem can switch modes between automatic and manual PLMN selection. (See "Registering on a network" on page 41.)
- **SWI_NOTIFY_PhonebookReady**—Phone book services are available. (See Chapter 14.)
- **SWI_NOTIFY_SmsReady**—SMS messaging services are available. (See Chapter 11.)
- **SWI_NOTIFY_SimStatusExp**—Application must wait for this notification before using SIM functions. (See Chapter 7.)

The host application should wait to receive these notifications before calling related functions.

Enabling network registration

Each time the host application starts, call **SwiSetHostStartup** to ensure the modem is able to register on a network.

Setting the TCP window size

The TCP window size controls how much data an application can send before receiving an acknowledgement from the recipient.

See the Microsoft Developer Network article "TCP Receive Window Size and Window Scaling" for details on setting the window size appropriately for your modem and operating system. Call **SwiOptimizeTcp** to modify Windows registry values as appropriate.

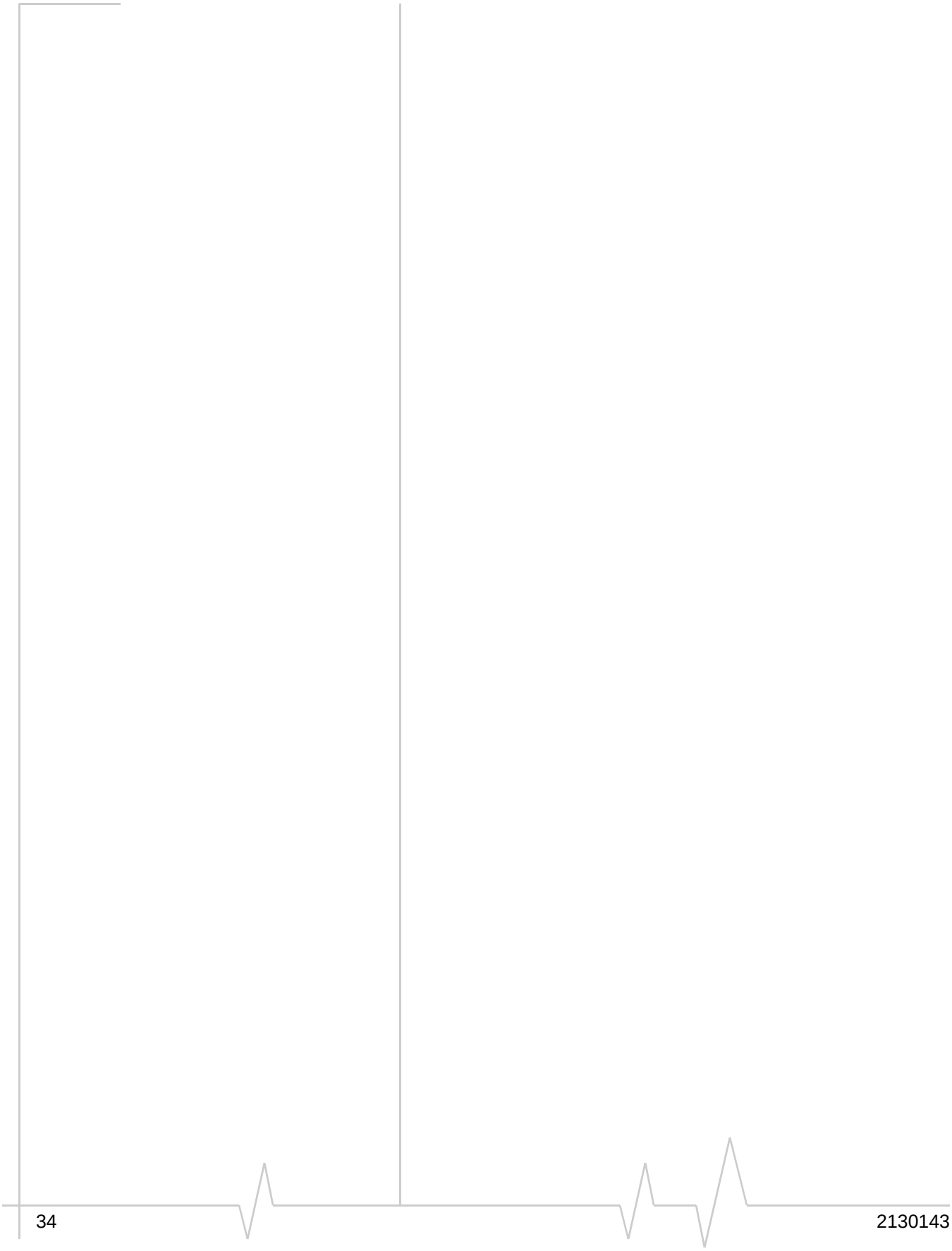
Closing the host application

When the host application closes, it should also shut down the modem and API:

1. Call **SwiApiShutdown** to shut down control and status messages between the modem and the API. (This does not affect data traffic over other interfaces).

Application development for Windows CE-based devices

For information on developing applications for Windows CE-based devices, please contact Sierra Wireless Technical Support. (See "Contact Information" on page 5.)



7: SIM Authentication and Codes

7

This chapter describes how to use API functions to perform the following SIM-related tasks on GSM modules:

- Unblocking the MEP feature
- Managing SIM security (CHV1 and CHV2)
- Unblocking CHV1 and CHV2

Using these API functions

*Note: The functions in this section are found in the **SIM** API module. For additional SIM-related functions, refer to the API documentation.*

The following are key usage notes for functions described in this chapter (see the API documentation for additional details):

- Host must enable notifications before they can be received. (See "Managing notifications" on page 23.)
- Host must receive the **SWI_NOTIFY_SimStatusExp** notification indicating that the SIM is ready before issuing any calls. It is issued after every SIM function call to report the function call result and the SIM's status.
- If a call requires a **SWI_NOTIFY_SimStatusExp** notification, do not repeat the call until you receive it or it times out.

Using a MEP code to unblock the modem

A MEP (Mobile Equipment Personalization) code is used to deactivate the GSM MEP feature, which restricts user equipment to a specific service provider's SIMs.

To deactivate the GSM MEP feature on the modem:

1. Call **SwiSetMEPUnlock** with the correct MEP unlocking code.
2. Wait for the **SWI_NOTIFY_SimStatusExp** notification indicating if the modem was unlocked successfully.
3. If the unlock attempt failed, verify that you are using the correct code and repeat this procedure.

SIM security

Note: In a typical host application interface, the user should have to enter the code twice to make sure it is entered correctly.

The SIM is protected by two levels of security (if enabled) to prevent unauthorized access of the SIM and its features:

- CHV1 — When enabled, a voice-enabled modem can only call emergency numbers unless the correct unlocking code (CHV1) is used.
- CHV2 — Always enabled. The correct unlocking code (CHV2) is required to use special features such as the FDN phonebook.

Checking / setting CHV1 enabled status

To determine if CHV1 is enabled, check the **SWI_NOTIFY_SimStatusExp** notification, or call **SwiGetSimLock**.

To enable or disable CHV1, call **SwiSetSimLock**.

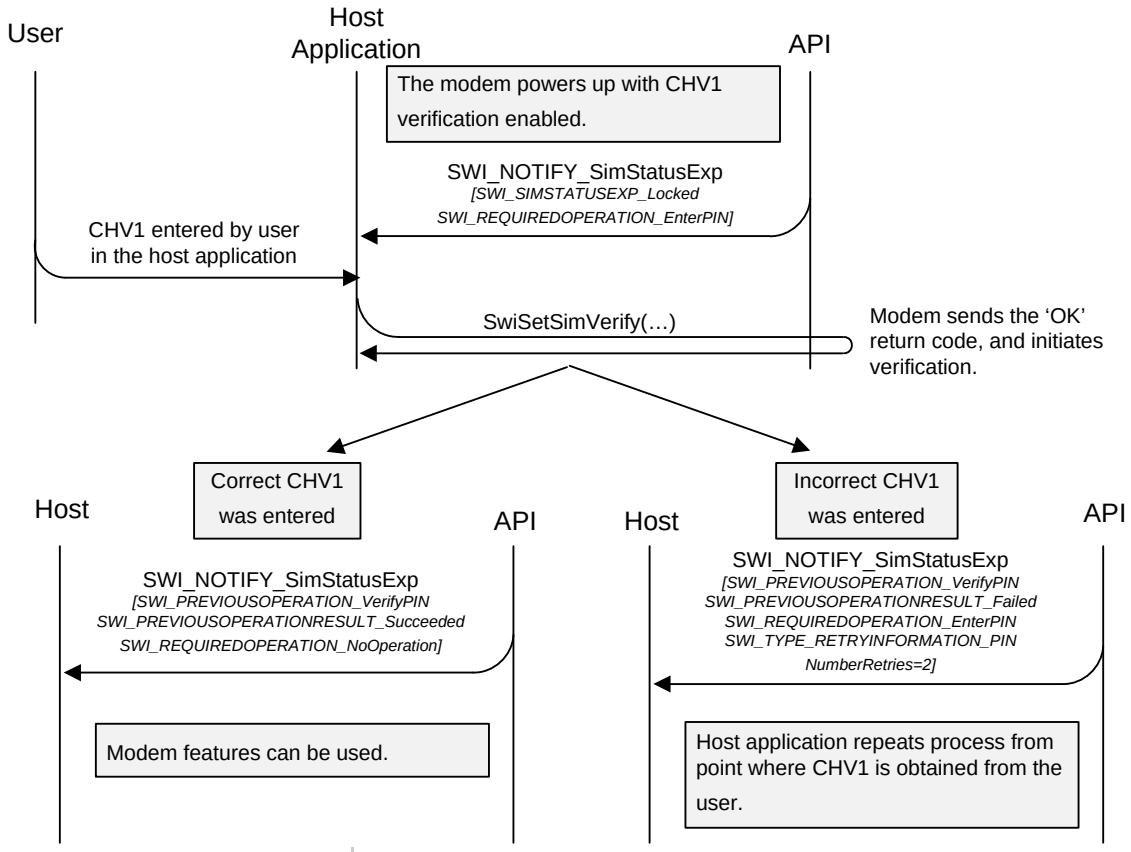
CHV1 verification

If CHV1 security is enabled, the CHV1 code must be entered:

1. When the modem is restarted or reset, the **SWI_NOTIFY_SimStatusExp** notification is sent to the host.
2. The host gets the CHV1 PIN and calls **SwiSetSimVerify** to compare the entered PIN with the CHV1 code on the SIM.
3. The modem sends the **SWI_NOTIFY_SimStatusExp** notification to the host, indicating success (correct PIN) or failure (incorrect PIN). If the PIN was wrong, the notification also indicates that the PIN has to be re-entered, and reports the number of retries remaining before the SIM will be blocked. (See "Unblocking CHV1 or CHV2" on page 38 for details.)

Note: Do not hard code the number of allowed attempts—use the information in the notification. (The number of allowed attempts is network-dependent.)

Figure 7-9: CHV1 verification process



CHV2 verification

CHV2 is always enabled — the CHV2 code must be sent to the modem to access special features such as the FDN phonebook and the ACM (Accumulated Call charge Meter) feature.

1. Call **SwiChv2StatusKick** to tell modem to request CHV2 verification is required.
2. Receive **SWI_NOTIFY_SimStatusExp**.
3. Call **SwiSetSimVerify** with the CHV2 code.
4. Receive **SWI_NOTIFY_SimStatusExp** indicating success.

Note: Just like CHV1, if an incorrect CHV2 code is entered repeatedly, CHV2-restricted functionality becomes blocked. See "Unblocking CHV1 or CHV2" on page 38 for details on unblocking CHV2.

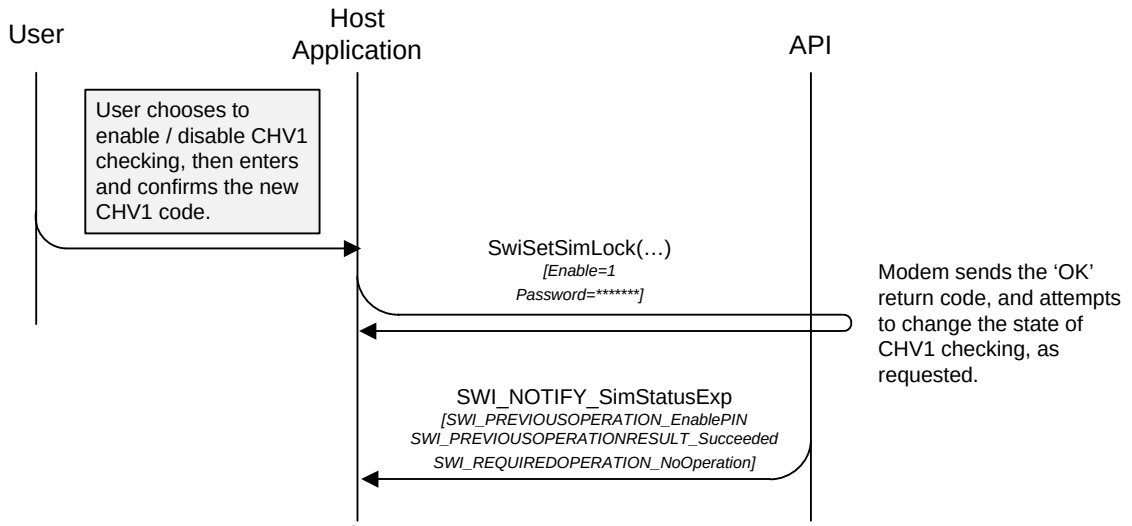
Enabling / disabling CHV1

Note: Only CHV1 can be disabled—CHV2 is always enabled.

To enable or disable CHV1:

1. After getting the CHV1 code from the user, call **SwiSetSimLock**, passing the CHV1 code and setting the Enable flag to enable or disable.
2. Wait for the modem to send the **SWI_NOTIFY_SimStatusExp** notification indicating success or failure (likely because the wrong code was entered).

Figure 7-10: Changing CHV1 verification state



Changing CHV1 or CHV2

To change the CHV1 code:

1. Calls **SwiSetSimPassword** with the new CHV1 code.

To change the CHV2 code:

2. Call **SwiChv2StatusKick** with the parameter **SWI_TYPE_CHV2KICKTYPE_Change**. This triggers the **SWI_NOTIFY_SimStatusExp** notification.
3. Receive the **SWI_NOTIFY_SimStatusExp** notification.
4. Call **SwiSetSimPassword** with the new CHV2 code.

Note: You cannot use an emergency number as the CHV1 code (or as the beginning of the CHV1 code).

Unblocking CHV1 or CHV2

When either CHV1 or CHV2 is blocked because the wrong code was entered too many times in a row, a PUK (Pin Unblocking Code) can be used to unblock it. (PUK1 is used for CHV1, and PUK2 is used for CHV2.)

Note: If CHV1 is permanently blocked, voice-enabled modems can only dial emergency numbers.

PUK codes are obtained from the service provider. If an incorrect PUK code is used too many times in a row, CHV1 (for PUK1) or CHV2 (for PUK2) becomes permanently blocked.

To unblock either CHV code, use the process described for CHV1 verification, replacing the CHV code with the appropriate PUK code.

8: Account Profile Management

8

This chapter describes how to use API functions to perform the following profile-related tasks:

- Read, create, edit, and delete profiles
- Assign a default profile
- Activate profiles

Using these API functions

*Note: The functions in this section are found in the **Connection Profiles** API module. For additional profile-related functions, refer to the API documentation.*

The following are key usage notes for functions described in this chapter (see the API documentation for additional details):

- Host must enable notifications before they can be received. (See "Managing notifications" on page 23.)
- **SWI_NOTIFY_GsmProfileChange** reports any changes made to profiles.

Account profile overview

Account profiles contain information used by the network to verify access to network services. The profile information is obtained from the service provider (usually with the SIM) and may contain:

- A username
- A password
- An APN (Access Point Name)
- An IP address (if not automatically assigned by the network)
- Indication as to whether IP header compression is used
- DNS address(es)

Number of supported profiles

UMTS modules support several profiles (labeled 1, 2, ...). Refer to your modem's Product Specification Document for the actual number supported.

Profile maintenance functions

Identifying account profiles

To get a list of all account profiles, call **SwiGetGsmProfileSummary**. The modem returns a profile list, which includes the status of each profile, and identifies the default profile and active profile.

Reading profiles

To read the full details for a specific profile:

1. Call **SwiGetGsmProfileBasic** to read basic details.
2. Call **SwiGetGsmProfileDns** to read DNS details.

Creating and editing profiles

Creating profiles

Note: After each call, the modem sends the SWI_NOTIFY_GsmProfileChange notification. You do not have to wait for this notification to arrive before calling the next function.

To create a new profile, call the following functions (and pass the new profile's index number in each call):

1. Call **SwiSetGsmProfileBasic** to set basic details.
2. Call **SwiSetGsmProfileDns** to set DNS details.

Setting a default profile to autoactivate

If the profile designated as the default profile is set to autoactivate, the modem initiates a connection using it as soon as the modem is reset.

To set the default profile, call **SwiSetDefaultProfile** using the profile's index number.

Activating a profile

Activating a profile initiates a packet data connection. To activate a profile, call **SwiActivateProfile**. (See "Establishing a data connection" on page 45 for details.)

Deleting profiles

To delete a profile and set it back to factory default values, call **SwiEraseProfile**.

9: Network registration

9

This chapter describes how to use API functions to perform the following tasks:

- Register on a network
- Select frequency bands
- Manually select a network

Using these API functions

*Note: The functions in this section are found in the **Modem Information and Management** and **Network Management and Status** API modules. For additional network registration-related functions, refer to the API documentation.*

The following are key usage notes for functions described in this chapter (see the API documentation for additional details):

- Host must enable notifications before they can be received. (See "Managing notifications" on page 23.)
- To register, an unlocked SIM with a valid account (with no restrictions affecting registration) must be in the modem, the modem must be configured to work on appropriate bands/networks, and it must be in range of a network with adequate signal strength.

Registering on a network

The modem must be registered on a network before data, voice, sms, or other connections can be established.

To register the modem on a network:

1. Make sure the modem is powered up. Call one of the following functions (see the API to determine which is appropriate for your application):
   - **SwiSetHostStartup** with Startup = True.
   - **SwiSetModemDisable** with ModemDisable = False
2. Set the frequency band(s), if necessary, on which the modem will operate. (See your modem's Product Specification Document for a list of supported bands.). See Setting the frequency band(s) below.
3. Select the network on which to register the modem, if necessary.
4. Once the network has been selected, the modem registers automatically.
5. If you want to change the network manually, see Selecting and registering on a network below.

Setting the frequency band(s)

To get a list of supported bands for your modem:

1. Call **SwiGetBandInfo**.

To set the bands that the modem should use (or to 'autoband'):

1. Call **SwiSetBandInfo**.
2. Wait for the **SWI_NOTIFY_BandWrite** notification, which indicates if the band change is successful.

*Note: **SWI_NOTIFY_Band** reports the new frequency band. It is received whenever the modem switches between bands (after a function call, if the modem is set to 'autoband', or after a non-API action such as an AT command).*

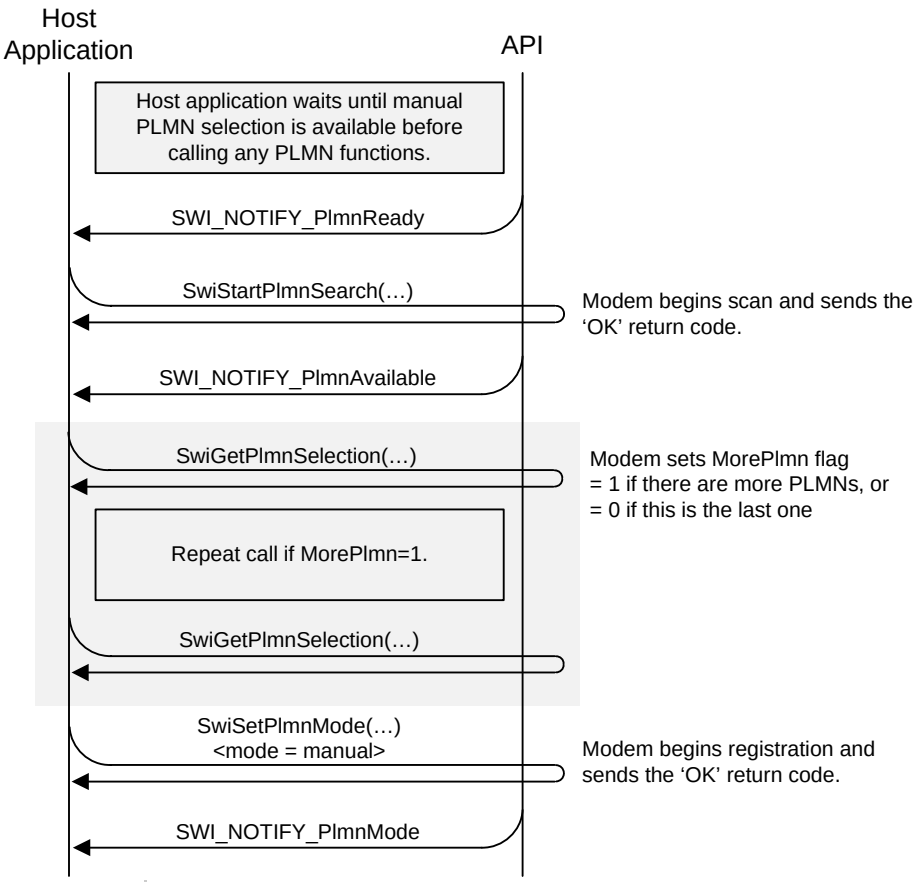
Selecting and registering on a network

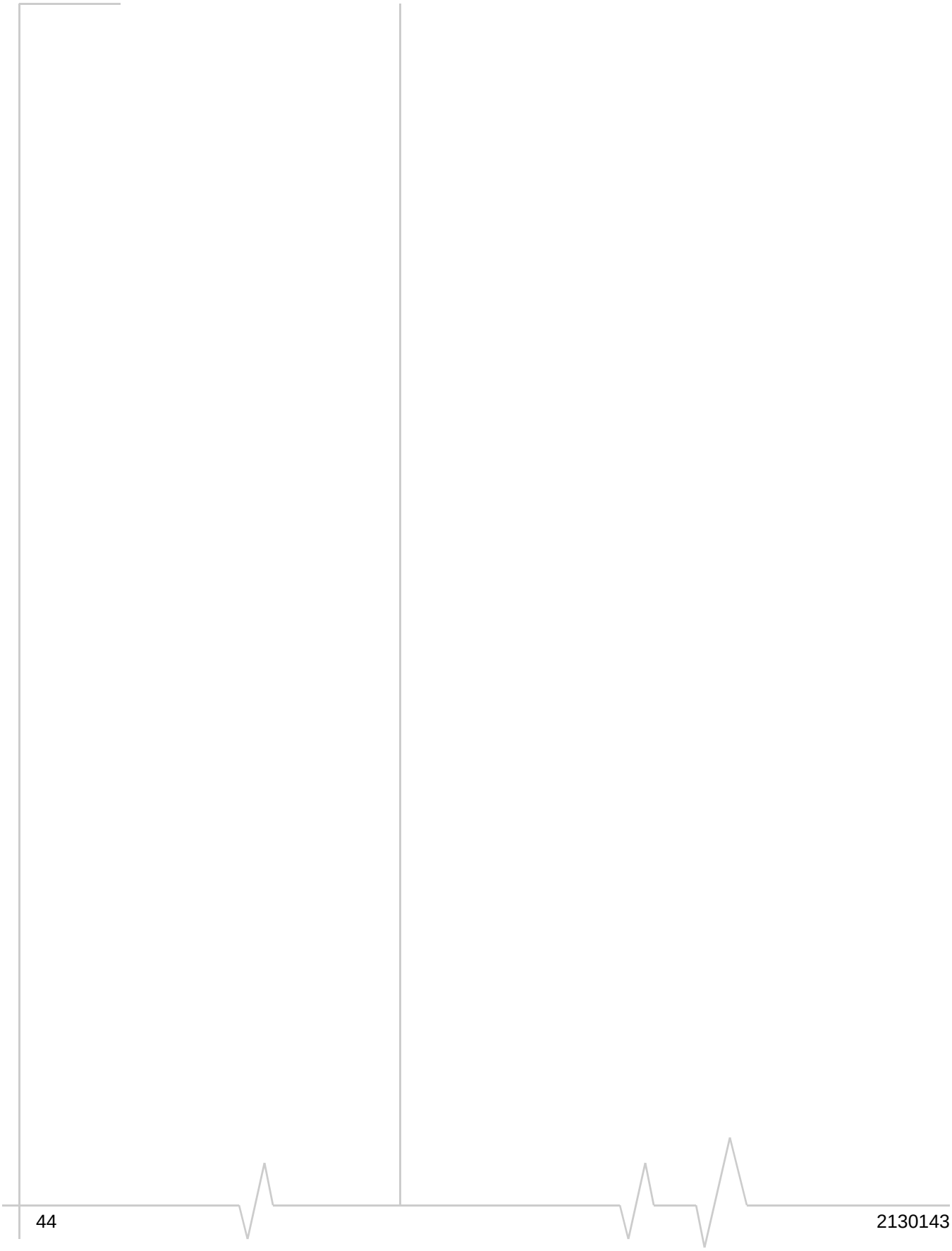
The modem can be set to select a network automatically or manually by calling **SwiSetPLMNMode**. To see what the current setting is, call **SwiGetPLMNMode**.

To manually select a network on which to register:

1. Wait for the **SWI_NOTIFY_PlmnReady** notification. This indicates that manual selection is available and can begin.
2. Call **SwiStartPLMNSearch** to identify available PLMNs.
3. Wait for the **SWI_NOTIFY_PlmnAvailable** notification, indicating that a list of PLMNs can be read from the modem.
4. Call **SwiGetPLMNSelection** to read a PLMN from the modem. Repeat until all PLMNs are read. (The PLMN data structure sets the flag MorePlmn = 1 if there are more PLMNs to read.)
5. Call **SwiSetPLMNMode**, set the mode to 'manual', and identify the PLMN on which to register.
6. Wait for the **SWI_NOTIFY_PlmnMode** notification indicating if the registration attempt succeeded.

Figure 9-11: Manual PLMN selection





10: Data Connections

10

This chapter describes how to use API functions to perform the following task:

- Establish a data connection

Using these API functions

*Note: The functions in this section are found in the **Connection Profiles** and **Network Management and Status** API modules. For additional connection-related functions, refer to the API documentation.*

The following are key usage notes for functions described in this chapter (see the API documentation for additional details):

- Host must enable notifications before they can be received. (See "Managing notifications" on page 23.)
- The modem must be registered on a network before you try to establish a data (GPRS / EDGE / UMTS) connection. See Chapter 9 for details.
- A data connection cannot be initiated if a voice call is in progress.
- (Voice-enabled modems only) If a voice call occurs during a data connection, the data connection is maintained, but suspended (the IP address is not lost). The data connection resumes when the voice call completes. (Note that your host application may have timed out by this point.)

Establishing a data connection

The modem can establish packet-switched connections on GPRS, EDGE, and UMTS networks. See the modem's PSD for maximum data rates.

To establish a data connection with a modem that is already registered on a network:

1. If not already enabled, enable notifications that are received when conditions that can affect the connection occur. Some of these notifications include:
   - **SWI_NOTIFY_RegistrationExp** — Modem is registered on a network (and identifies the PLMN and/or SPN)
   - **SWI_NOTIFY_Band** — Modem switched bands (due to function call or autoband)
   - **SWI_NOTIFY_SimStatusExp** — SIM status changes.
   - **SWI_NOTIFY_NetworkStatus** — Network status changes.
   - **SWI_NOTIFY_ServiceIcon** — Available services change (GPRS, EDGE, UMTS).
   - **SWI_NOTIFY_Rssi** — RSSI value changes.
   - **SWI_NOTIFY_Temperature** — Modem is overheating. Data transmission is suspended until the temperature drops.
   - **SWI_NOTIFY_TransmitAlert** — Problem with the antenna.

Note: If the default profile is set to auto-activate, a data connection is initiated as soon as the modem is reset.

2. Call **SwiActivateProfile** to establish the fastest available connection using a valid profile.
3. Wait for a **SWI_NOTIFY_PktSessionCall** notification, which indicates if the connection was successful.
4. An IP address is assigned to the modem when the connection is established. To get the IP address of an active profile, call **SwiGetIPAddress**.

11: SMS Messaging

This chapter describes how to use API functions to perform the following SMS-related tasks:

- Read and delete incoming messages from the SIM
- Store and send outgoing messages from the SIM
- Enable / disable SMS status reports
- Configure SMS parameters

Using these API functions

*Note: The functions in this section are found in the **SMS** API module. For additional SMS-related functions, refer to the API documentation.*

The following are key usage notes for functions described in this chapter (see the API documentation for additional details):

- Host must enable notifications before they can be received. (See "Managing notifications" on page 23.)
- When the host application starts up, it must receive **SWI_NOTIFY_SmsReady** before calling SMS functions, and receive **SWI_NOTIFY_PhonebookReady** before calling and phonebook-related SMS functions.
- The host must explicitly remove Mobile-terminated (MT) SMS messages from the SIM.

SMS message types

The API supports the following SMS message types:

- MO-SMS (Mobile-Originated SMS)—Outgoing messages stored on the host until sent to the modem, then deleted from the modem after being sent to the network.
  - Message sent from modem to network service center (see SMS parameters for center number)
  - Message remains at service center until downloaded by the recipient or the validity period expires (see sms parameters for period). If period expires, messages are deleted.
- MT-SMS (Mobile-Terminated SMS)—Incoming messages stored on the SIM until deleted by the host.

Reading SMS messages

SMS messages are stored on the SIM in a circular list. Use the procedure below to read all messages from the SIM. The messages remain on the SIM (marked as 'read') until they are explicitly deleted (see "Deleting SMS messages" on page 50).

Determining if messages are on the SIM

1. Wait for SMS service to become available — the modem sends the **SWI_NOTIFY_SmsReady** notification.
2. Call **SwiGetSMSMessageStatus** to get the number of unread and previously read messages on the SIM, and the number of messages waiting at the service center. (You can then wait for **SWI_NOTIFY_SmsStatus** notifications that indicate when new messages are received.)

Note: The SWI_NOTIFY_SmsSimFull notification is sent when the SIM is full. If the SIM is full, messages will queue at the service center.

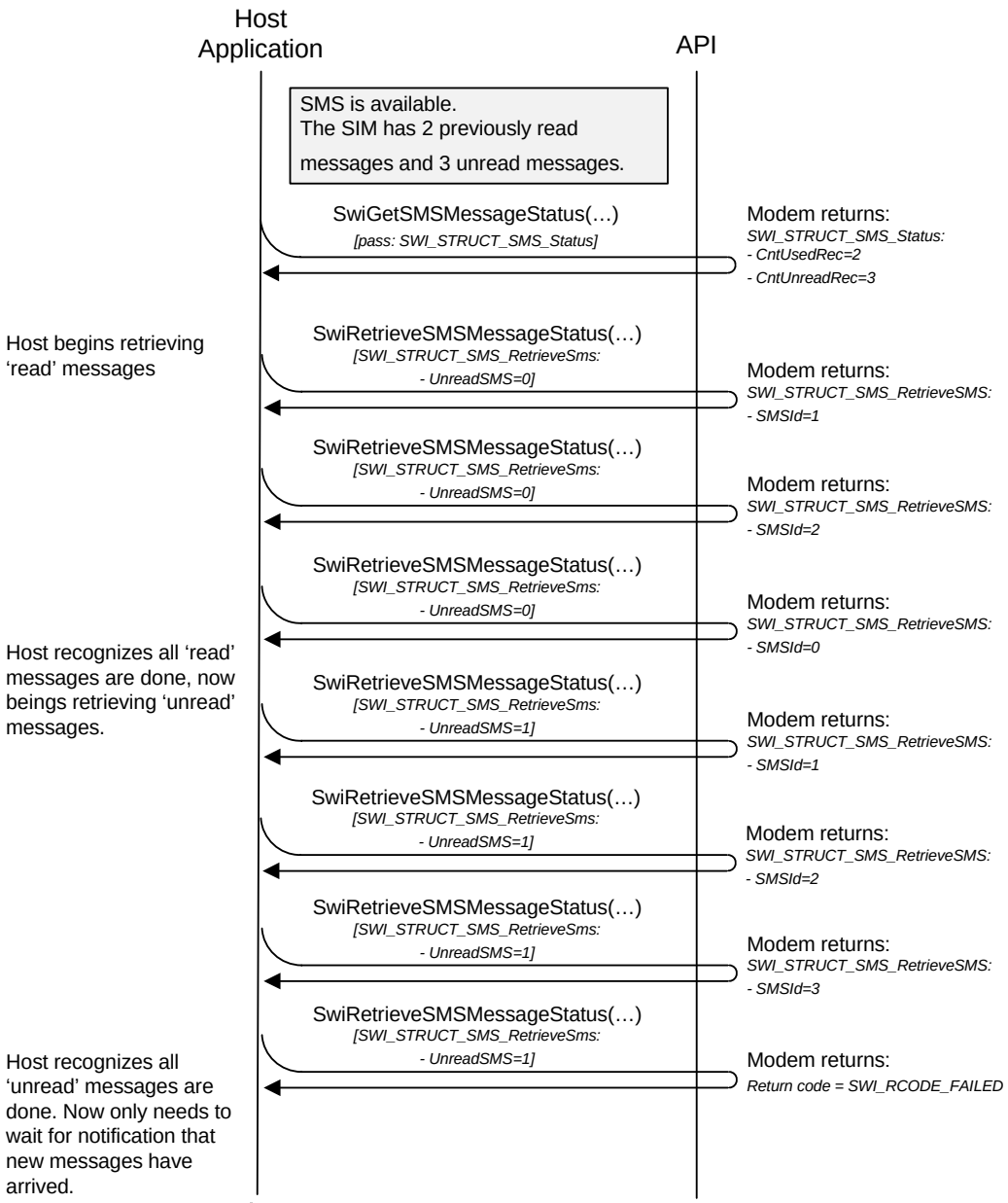
Reading messages from the SIM

To read all of the messages from the SIM, read all of the previously read messages first, then all of the unread messages:

1. Call **SwiRetrieveSMSmessage** with UnreadSMS=0 to get the next available read message.
2. The modem returns the message.
   Note: Make a note of the message ID of the first message received.
3. Repeat steps 1–2 (do not change UnreadSMS) until all messages have been read (you will know this has happened when the message ID of the latest message matches the ID from the first message received).
4. Set UnreadSMS=1 to get unread messages, and repeat steps 1–3 until the function call returns **SWI_RCODE_FAILED**, which indicates there are no more unread messages on the SIM.

Note: Don't rely on the number of messages returned from SwiGetSMSMessageStatus—new messages could be received while you are reading existing messages from the SIM.

Figure 11-12: Read messages from the SIM



Deleting SMS messages

Incoming SMS messages on the SIM must be explicitly deleted — they are not removed by reading them.

To delete an incoming SMS message from the SIM:

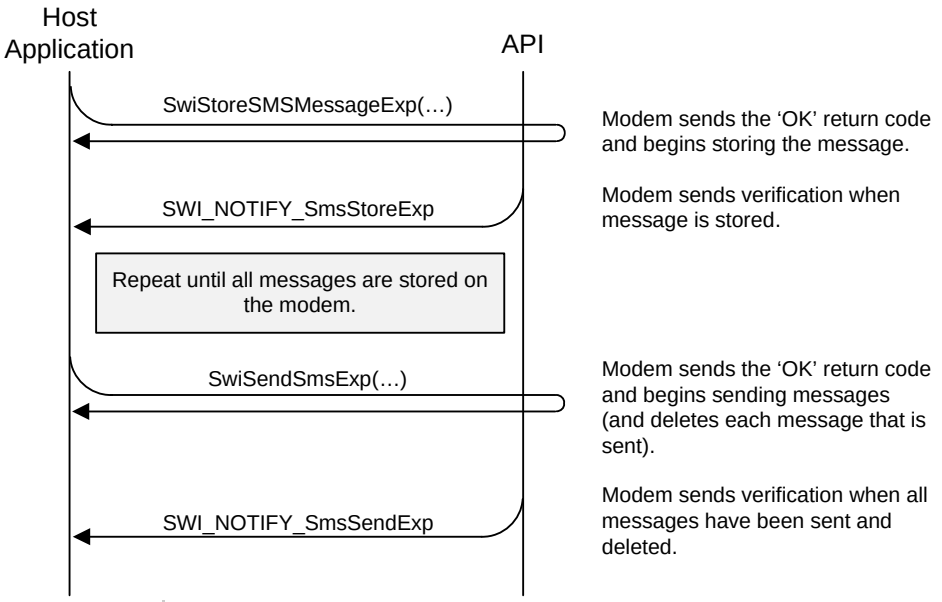
1. Call SwiDeleteSms using the message ID (SmsId) returned from the SwiRetrieveSMSMessage function.

Sending SMS messages

To send SMS messages, each message must be stored on the modem, and then all queued messages can be sent from the modem to the network.

1. Call **SwiStoreSMSMessageExp** with the first (or only) part of the SMS message begin sent (message lengths are network-dependent).
2. Wait until the modem sends **SWI_NOTIFY_SmsStoreExp** before trying to send another message/message portion.
3. Repeat until all messages have been stored.
4. Call **SwiSendSMSExp** to send *all* the messages from the modem to the network. (When a message is sent, it is deleted from the modem.)
5. Wait until the modem sends **SWI_NOTIFY_SmsSendExp** before trying to store or send any additional messages.

Figure 11-13: Send SMS messages



SMS status reports

To determine if SMS status reports are enabled or disabled, and if the user can change the status, call **SwiGetSmsStatusReportCfg**.

To change the status report settings, or to enable or disable status reports, call **SwiSetSmsStatusReportCfg**.

Configuring SMS parameters

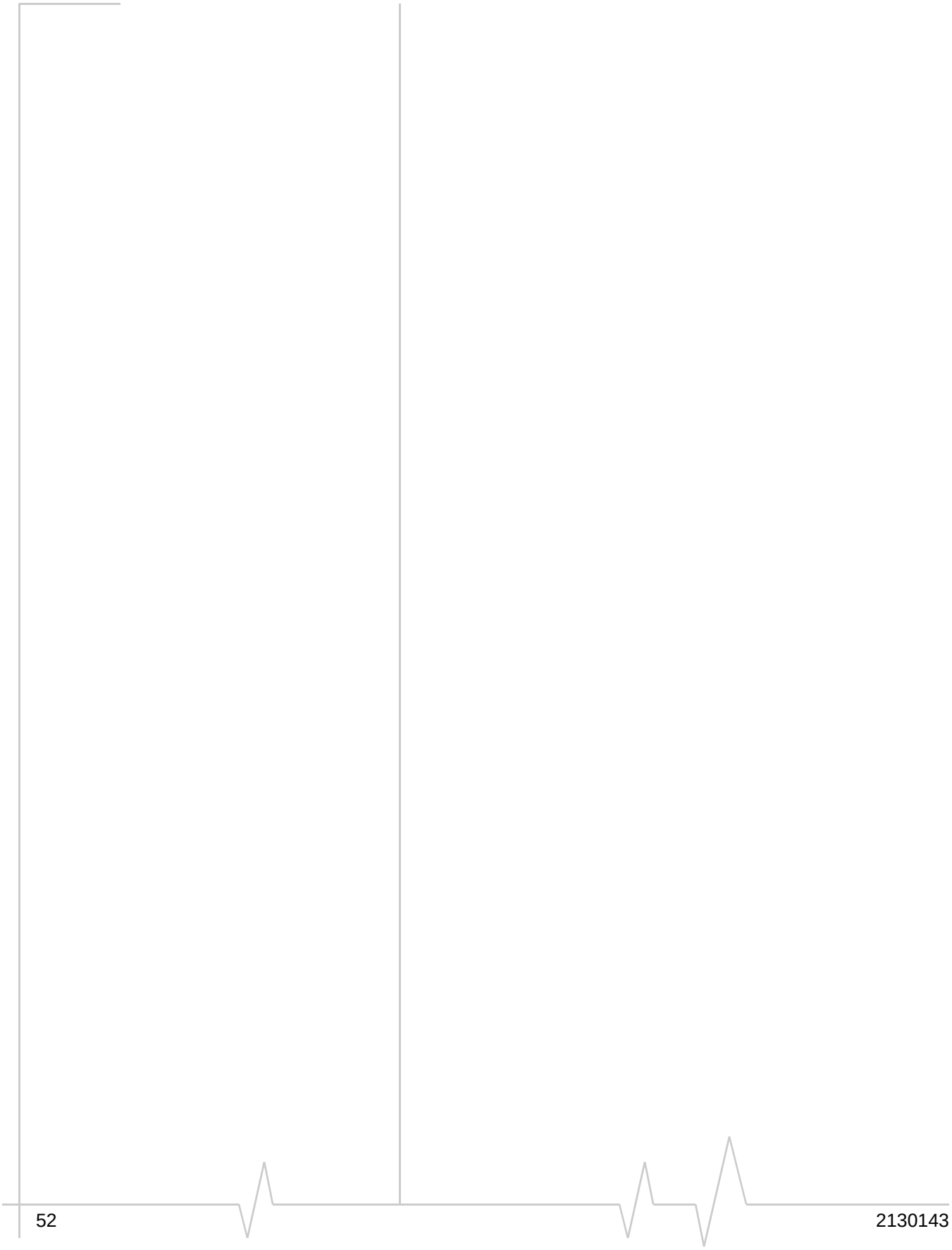
Several SMS parameters can be customized (for a complete list, see the API documentation).

To get the current parameters, call **SwiGetSmsParam**.

To customize parameters, call **SwiSetSmsParam**.

The following customizations are the only ones that should normally be exposed for the user to change:

- **Default Destination Address**—allows the user to specify an SMS address to be used as the default recipient of messages.
- **Service Center**—The service provider likely pre-configured the SIM with a service center number but the interface should allow this to be changed should the service provider change the number.
- **Validity Period**—This is typically user-defined.
- **Routing Option**—This specifies which service is used to send SMS messages.



12: Location-based services

12

This chapter describes how to use API functions to perform the following tasks:

- Get and set modem parameters and statuses
- Perform single location fixes
- Manage a tracking session
- Keep almanac / ephemeris data up to date

Using these API functions

*Note: The functions in this section are found in the **Location Based Services** API module. For additional LBS-related functions, refer to the API documentation.Managing LBS component settings (modem/base station/ satellite/user options)*

The following are key usage notes for functions described in this chapter (see the API documentation for additional details):

- Host must enable notifications before they can be received. (See "Managing notifications" on page 23.)

Retrieving operational settings

Get/set modem default and current operational parameters

To report the modem's default operational parameters, and to get and set the current values of these parameters, call the following functions:

- **SwiGetLbsPaParam** — Get the modem's default operational parameters.
- **SwiGetLbsPaPortId**/**SwiSetLbsPaPortId** — Get and set the SUPL Server port ID
- **SwiGetLbsPaIpAddr**/**SwiSetLbsPaIpAddr** -- Get and set the SUPL Server IP address

Get satellite details

To read satellite information for all satellites in view (azimuth, elevation, SNR, etc.):

1. Call **SwiGetLbsSatInfo**.

Get LBS status

To determine the status of the most recent fix session:

1. Call **SwiGetLbsPdStatus**.

Get/set user-selected LBS fix settings

To get or set LBS fix settings (fix type, performance, accuracy, etc.):

1. Call **SwiGetLbsFixSettings** or **SwiSetLbsFixSettings**.

Position fix / tracking sessions

You can use LBS functions to initiate single position fixes and tracking sessions (multiple position fixes).

Report modem's last known location

To get the result of the modem's most recent position fix:

1. Call **SwiGetLbsPdData**.

Get modem's current location (Initiate single position fix)

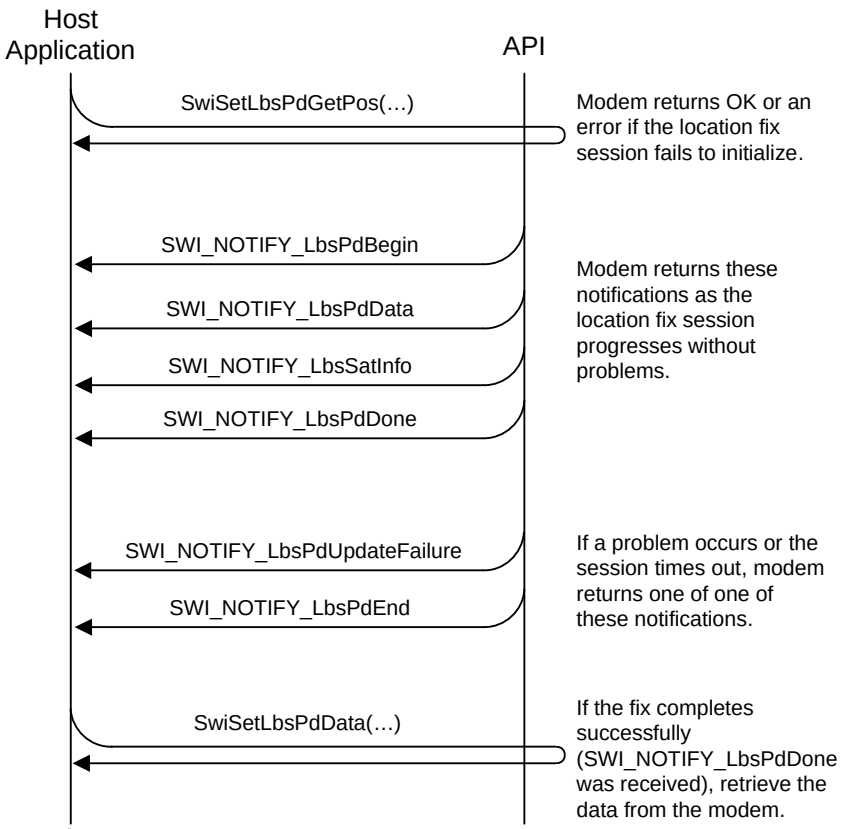
To get the modem's current location:

1. Call **SwiSetLbsPdGetPos** to initiate a position fix. The return code indicates if the fix is initiated or if an error occurred.
2. Wait for notifications reporting the progress of the fix:

   The following notifications are received when a fix is successful:
   - **SWI_NOTIFY_LbsPdBegin**.
   - **SWI_NOTIFY_LbsPdData** (data is available).
   - **SWI_NOTIFY_LbsSatInfo** (data is avaialble).
   - **SWI_NOTIFY_LbsPdDone** (fix is complete).

   The fix may not complete successfully, in which case other notifications are received:
   - If the fix fails due to an error, or if it is stopped via the API or another interface (AT, CnS, etc.), receive **SWI_NOTIFY_LbsPdUpdateFailure**.
   - If the fix times out, receive **SWI_NOTIFY_LbsPdEnd**
3. If the fix was successful, call **SwiGetLbsPdData** to get the fix results.

Figure 12-14: Initiating a single position fix



Initiate tracking session

A tracking session consists of a number of individual position fixes repeated a number of times (the fix count) at a specific rate (the fix rate). If the fix count is 1000, the session keeps running until it is explicitly stopped (see "End tracking session" on page 56).

To initiate and process a tracking session:

1. Call **SwiSetLbsPdTrack** to initiate the session. The return code indicates if the session is initiated or if an error occurred.
2. Wait for notifications reporting the progress of each location fix. (See "Get modem's current location (Initiate single position fix)" on page 54—note that the SWI_NOTIFY_LbsPdDone notification is only sent after the final fix).
3. When the final fix is finished (based on the fix count), the host receives the SWI_NOTIFY_LbsPdDone notification.

End tracking session

To end the tracking session prematurely (before it reaches the fix count):

1. Call **SwiSetLbsPdEndSession**.
2. Receive the **SWI_NOTIFY_LbsPdEnd** notification (indicating why the session ended) and the **SWI_NOTIFY_LbsPdDone** notification (indicating the session is finished).

Respond to network-initiated fix request

If the network requests the modem's location (using the **SWI_NOTIFY_LbsNiReq** notification):

1. The host receives the **SWI_NOTIFY_LbsNiReq** notification. If the notification's NotifType value is LBSNIREQNOTIF_UserRespReq, the host must respond, otherwise the modem's location is returned to the network automatically.
2. If the host must respond, it calls **SwiSetLbsNiReq** indicating if the request is accepted or rejected.
3. If the request is accepted, the modem's location is returned to the network.

Ephemeris / almanac data

The API includes commands that affect the downloading of assistance data.

Enabling / disabling 'Keep Warm' processing

While GPS is required by the host, the GPS assistance data can be kept current by enabling 'Keep Warm' processing. When enabled, the modem periodically downloads GPS assistance data.

To enable Keep Warm processing:

1. Call **SwiSetLbsPaKeepWarmStart**. The modem determines how often to download assistance data.
2. The host receives the **SWI_NOTIFY_LbsPaWarmBegin** notification indicating that Keep Warm has begun. The host also receives (periodically) the **SWI_NOTIFY_LbsPaWarmStatus** notification indicating the current status of Keep Warm processing.

To disable Keep Warm processing:

1. Call **SwiSetlbsPaKeepWarmStop**.
2. The host receives the **SWI_NOTIFY_LbsPaWarmDone** notification indicating that Keep Warm is finished.

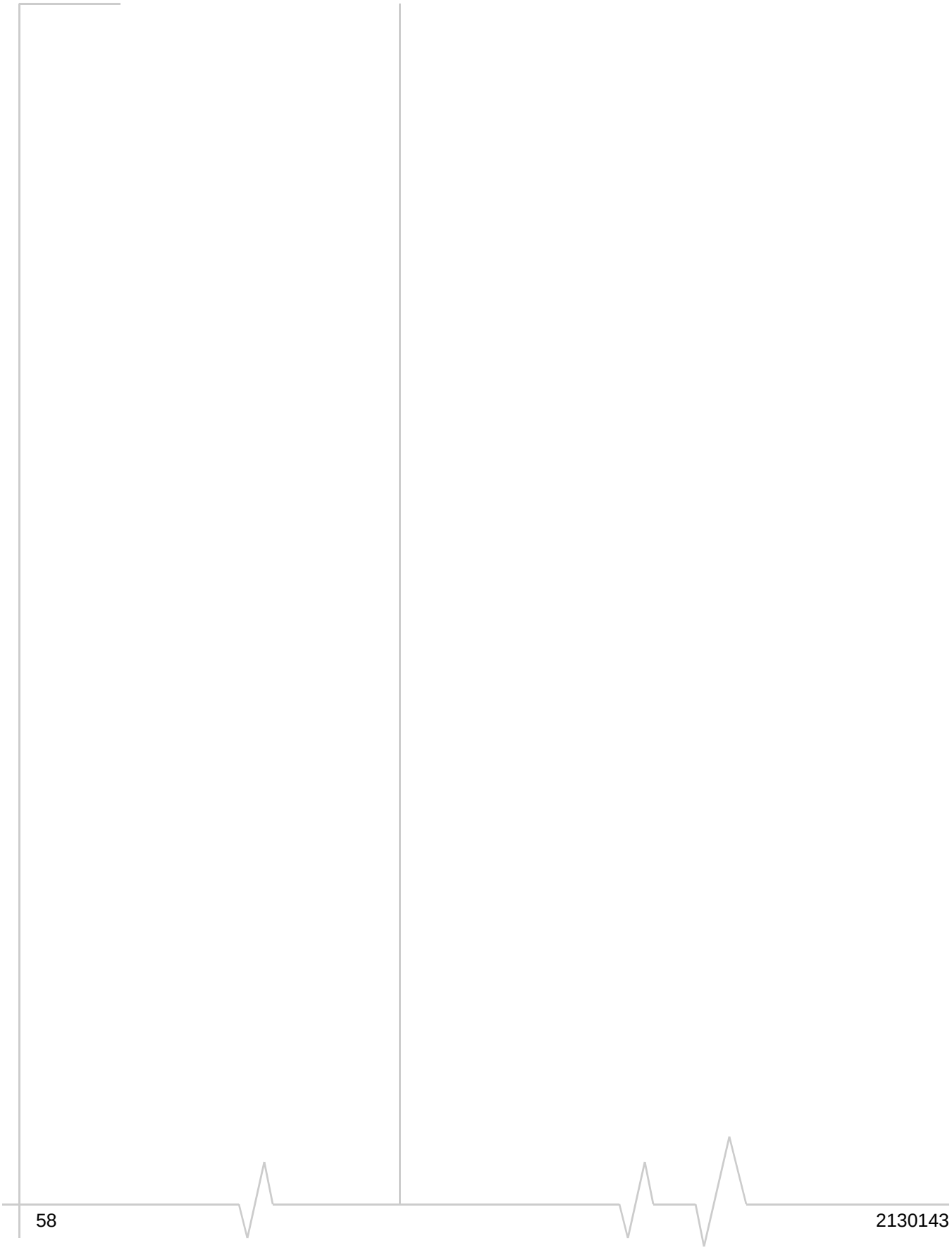
To check the status of Keep Warm processing (enabled/disabled):

1. Call **SwiGetLbsPaWarmStatus**.

Simulating a coldstart to force assistance data download

To simulate a coldstart:

1. Call **SetLbsClearAssistance** to clear the location parameters.



13: Supplementary services

13

Supplementary services are voice services such as call forwarding, call waiting, call barring, etc.

This chapter describes how to use API functions to perform four common supplementary service transactions on voice-enabled GSM modules. These examples illustrate the patterns of expected function calls and notifications for these and other supplementary service transactions:

- Basic supplementary service transaction

Using these API functions

*Note: The functions in this section are found in the **Supplementary Services** API module. For additional supplementary service-related functions, refer to the API documentation.*

The following are key usage notes for functions described in this chapter (see the API documentation for additional details):

- Host must enable notifications before they can be received. (See "Managing notifications" on page 23.)
- Any time the host submits a supplementary service transaction request, it can receive any of a number of notifications for different transaction types. Each supplementary service request has a unique handle that is referred to in associated notifications.

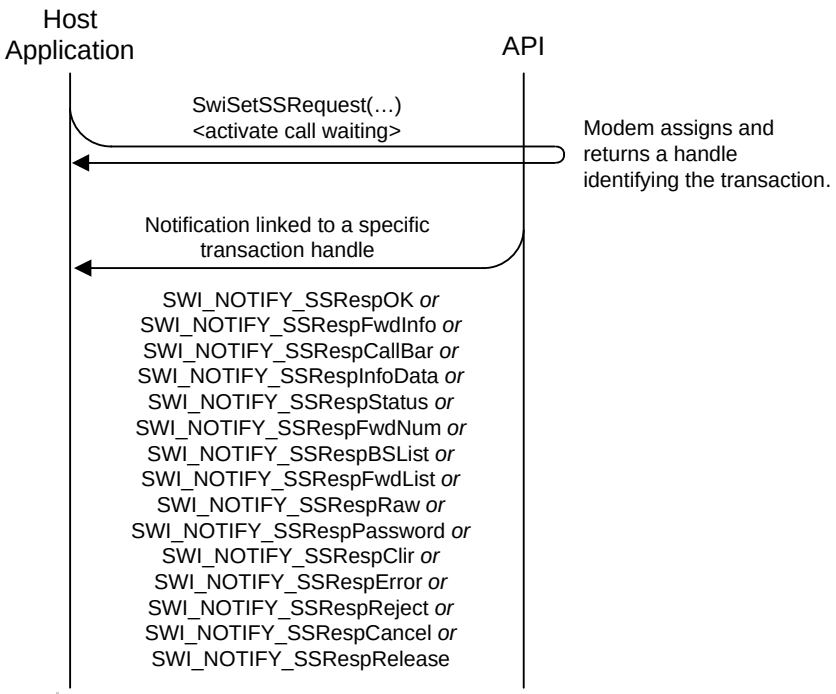
Basic supplementary service transaction

Five general request types can be used to initiate a supplementary service operation — activate, deactivate, interrogate (get details), register, or erase.

To process one of these requests:

1. Call **SwiSetSSRequest**. A unique handle is returned to identify the requested operation.
2. The host waits for notifications for this (and other) supplementary service requests. Each notification includes the handle of the request it refers to.

Figure 13-15: Basic supplementary services transaction



Supplementary service transaction requiring password

Certain supplementary services require a password (call barring, or general supplementary services password). This example uses Call Barring. The basic supplementary service transaction is extended with a modem-initiated password request and submission of the password by the host.

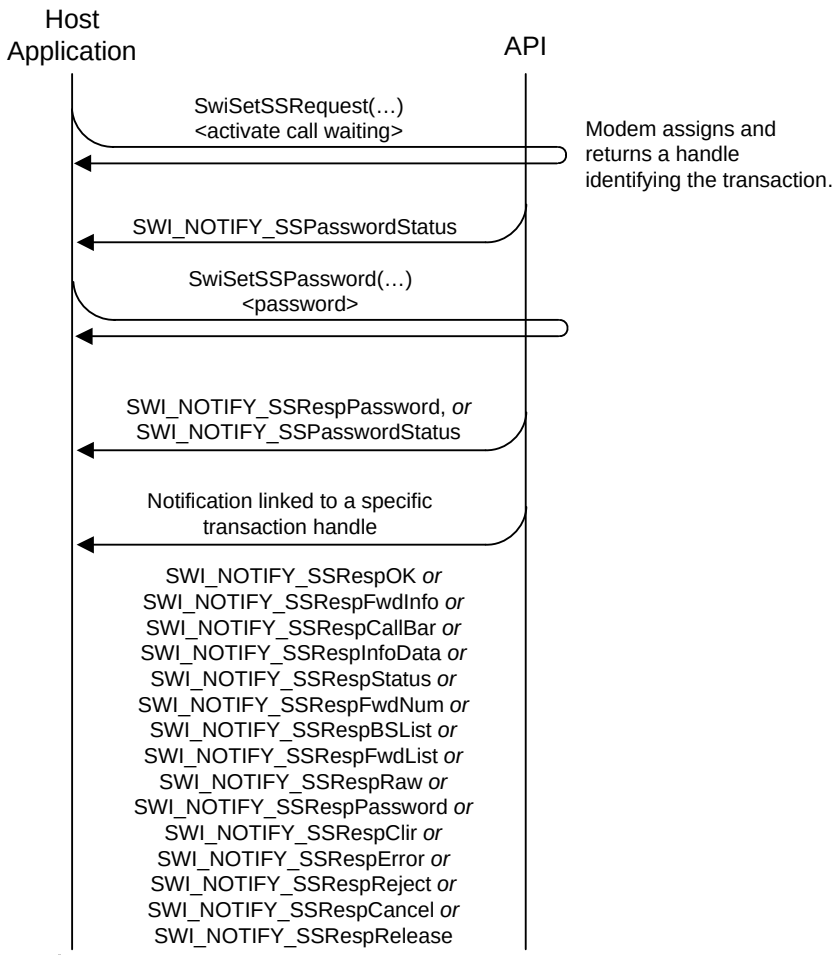
To process one of these requests:

Note: Process begins as for a basic transaction.

1. Call **SwiSetSSRequest**. A unique handle is returned to identify the requested operation.
2. Host receives a **SWI_NOTIFY_SSPasswordStatus** notification, which indicates the password type required.
3. Call **SwiSetSSPassword**, passing the required password.
4. Host receives one of the following notifications:
   - **SWI_NOTIFY_SSRespPassword**, if the password was verified.
   - **SWI_NOTIFY_SSPasswordStatus**, if the password was incorrect or invalid.

5. The host waits for notifications for this (and other) supplementary service requests. Each notification includes the handle of the request it refers to.

Figure 13-16: Supplementary service transaction requiring password



Changing a new supplementary services password

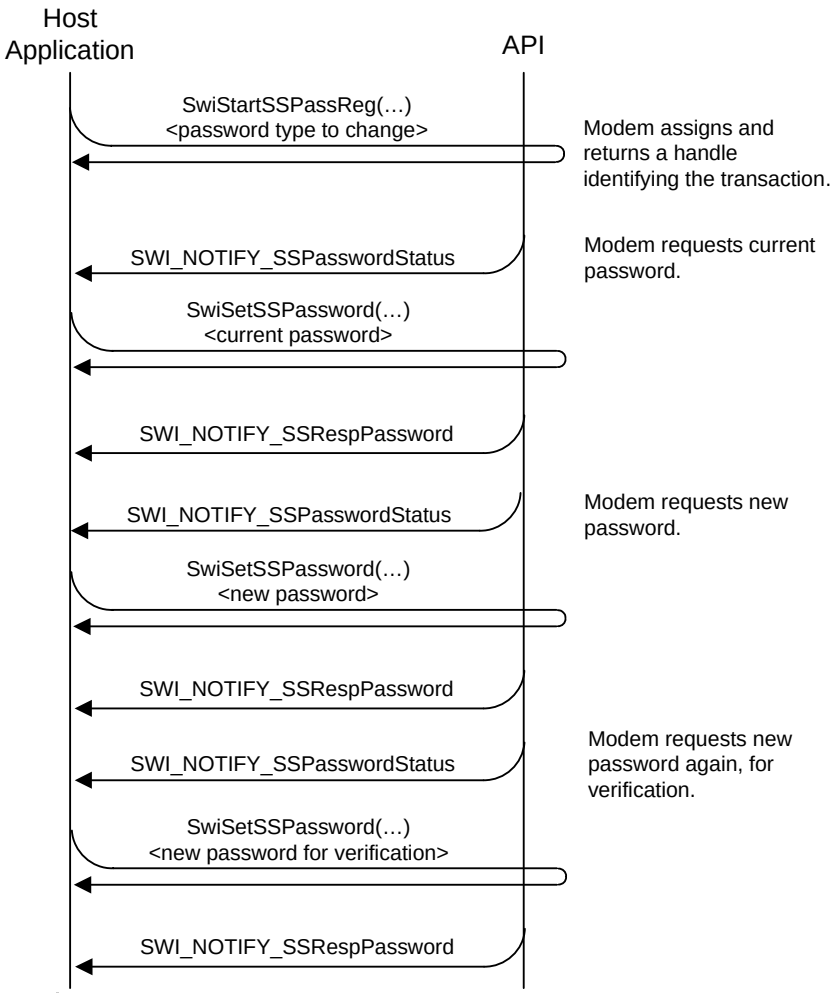
To change a supplementary services password:

1. Call **SwiStartSSPassReg** and indicate the password type to change.
2. Host receives a **SWI_NOTIFY_SSPasswordStatus** notification, requesting the current password.
3. Call **SwiSetSSPassword**, passing the current password.

Note: This example assumes the correct password is entered.

4. Host receives a **SWI_NOTIFY_SSRespPassword** notification. (Password was verified.)
5. Host receives another **SWI_NOTIFY_SSPasswordStatus** notification, requesting the new password.
6. Call **SwiSetSSPassword**, passing the new password.
7. Host receives a **SWI_NOTIFY_SSRespPassword** notification.
8. Host receives another **SWI_NOTIFY_SSPasswordStatus** notification, requesting the new password again for validation.
9. Call **SwiSetSSPassword**, passing the new password again.
10. Host receives **SWI_NOTIFY_SSRespPassword**.

Figure 13-17: Changing supplementary services password



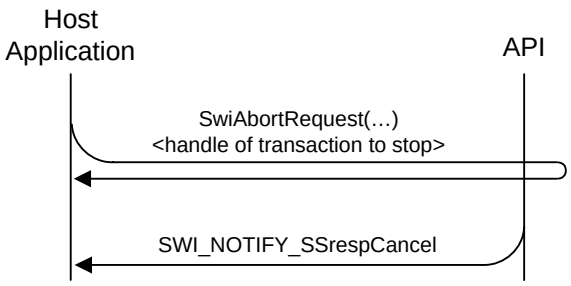
Stopping a supplementary service transaction

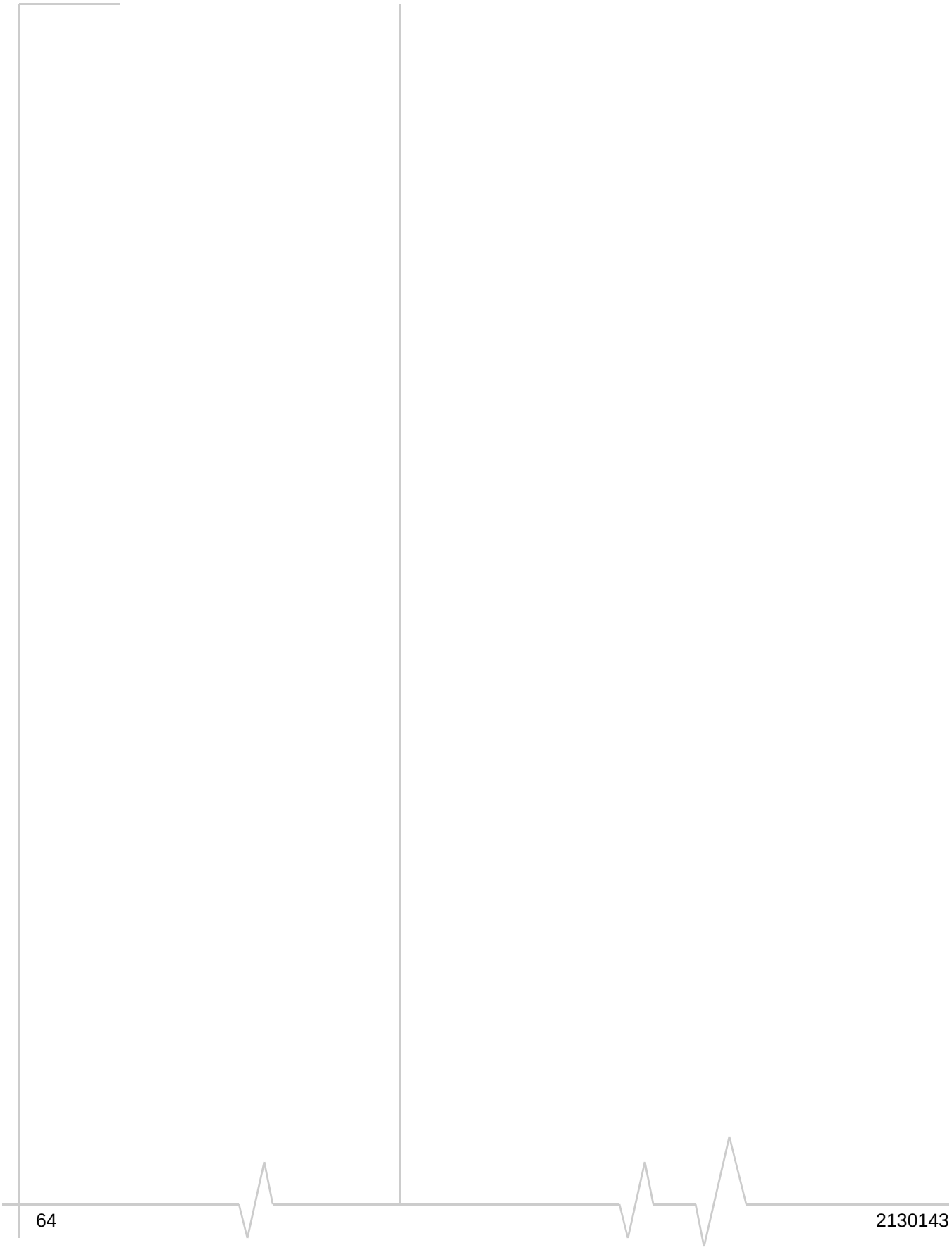
An outstanding supplementary service transaction can be aborted before completion by submitting an abort request.

To stop a transaction:

1. Call **SwiAbortRequest**, passing the handle of the transaction to stop.
2. Host receives a **SWI_NOTIFY_SSRespCancel** notification.

Figure 13-18: Stopping a supplementary service transaction





14: Phone Book Maintenance

14

This chapter describes how to use API functions to perform the following tasks:

- Use overdial numbers
- Maintain phone books (add, edit, delete entries)
- Use the FDN phone book
- Retrieve emergency phone numbers
- Retrieve phone numbers from any phone book

Using these API functions

*Note: The functions in this section are found in the **Phonebook** API module. For additional network registration-related functions, refer to the API documentation.*

The following are key usage notes for functions described in this chapter (see the API documentation for additional details):

- Host must enable notifications before they can be received. (See "Managing notifications" on page 23.)
- Host must receive the **SWI_NOTIFY_PhonebookReady** notification before calling any phone book-related functions.
- Host must receive the **SWI_NOTIFY_SimStatusExp** notification before forcing the modem to request CHV2 verification.

Supported phone books

The API supports the phone books listed in Table 14-6. (Availability of phone books is carrier-dependent.)

Table 14-6: Supported phone books

Abbreviation	Name	Description	Storage	Actions
ADN	Abbreviated Dialing Numbers	• Stores names / numbers for sending phone calls or SMS messages • Number of entries: carrier-dependent, typically 255 max. • Supports overdial numbers. See "Using overdial numbers" on page 66.	SIM	Add Edit Delete
CPHS *(Voice modems only)*	CPHS Mailing Numbers	• Stores up to four voice mailbox numbers; carrier-dependent • Example: Could be used to call a voice mailbox if messages are waiting. • Read-only	SIM	Edit

Table 14-6: Supported phone books (Continued)

Abbreviation	Name	Description	Storage	Actions
FDN	Fixed Dialing Numbers	• Stores numbers that user is restricted to using for dialing / SMS messages (when FDN is enabled). • Number of entries: carrier depend, typically 100 max. • Supports overdial numbers. See "Using overdial numbers" on page 66. See "Using the FDN phone book" on page 68 for additional details.	SIM	Add Edit Delete
LND *(Voice modems only)*	Last Numbers Dialed	• Stores most recent numbers dialed (typically 10). • Example: Could be used to implement a redial feature, or be added to ADN • Read-only phone book	SIM	Add Edit Delete
LNM *(Voice modems only)*	Last Numbers Missed	• Stores numbers of most recent missed incoming calls (typically 10). • Example: Could be used to implement a call-back feature, or be added to ADN. • Read-only phone book	Modem (NVRAM)	Delete (entire book)
LNR *(Voice modems only)*	Last Numbers Received	• Stores numbers of most recent answered incoming calls (typically 10). • Example: Could be used to implement a call-back feature, or be added to ADN. • Read-only phone book	Modem (NVRAM)	Delete (entire book)
MSISDN	Mobile Subscriber International Subscriber Identity Number	• Stores the account's phone number(s) • SIM may be pre-configured by carrier	SIM	Edit
SDN	Service Dialing Numbers	• Carrier-provisioned numbers • Examples: billing enquiries, emergency numbers, technical support, etc. • Read-only phone book	SIM	View only

Using overdial numbers

Note: The ADN and FDN phone books support overdial numbers.

Overdial numbers are numbers and symbols dialed after establishing a connection. They are part of the stored phone number, and begin with a comma.

Valid numbers and symbols include:

- '0'–'9'
- '*'
- '#'
- ',' (first occurrence in phone number)
  Indicates the beginning of the overdial phone number, and forces a three-second pause.
- ',' (subsequent appearances in phone number)
  Three-second pause.
- '?'
  Wildcard character. Indicates a missing digit that the user needs to enter.

For example, if the phone book number is 6045551212,,112,4:

- 6045551212. is dialed.
- Six-second pause after the connection is established
- 112 is dialed
- Three-second pause
- 4 is dialed

Using the phone book functions

Application start-up

When the host application starts up, it should:

1. Wait for **SWI_NOTIFY_PhonebookReady** before calling any phone book functions.
2. Call **SwiGetPhonebookAvailable** to see which phone books can be used.

Maintaining ADN, FDN, MSISDN phone books

Entries in the ADN, FDN, and MSISDN phone books can be updated using API functions.

Note: CHV2 must be verified before adding or editing FDN phone book numbers as shown in "Using the FDN phone book" on page 68.

To add a phone book entry:

1. Call **SwiAddPhonebookEntry**.

To edit a phone book entry:

1. Call **SwiEditPhonebookEntry**.

To delete a phone book entry:

1. Call **SwiDeletePhonebookEntry**.

Using the FDN phone book

Phone book entries:

When enabled, users can only call numbers that match the format of the entries in the phone book, as shown in the following examples of FDN phone book numbers.

- 60" — User can call any number beginning with "604" (for example, 6045551234)
- 2505558989 — User can only call 2505558989.

Enabling/disabling FDN

To check if FDN is enabled or disabled:

1. Call **SwiGetFdnMode**.

To enable or disable the FDN phone book:

2. Wait for the **SWI_NOTIFY_SimStatusExp** notification indicating that the SIM phone books are available.

Note: For information about CHV2 codes, see "CHV2 verification" on page 37.

3. Call **SwiChv2StatusKick** to tell the modem that CHV2 verification is required. This will trigger a notification to enter the CHV2 code.
4. Wait for the **SWI_NOTIFY_SimStatusExp** requesting the code.
5. Call **SwiSetFdnMode**, passing inthe CHV2 code.

SIM phone book statistics

To get the current sizes and remaining space for all SIM-based phone books:

1. Call **SwiGetPhonebookSize**.

Retrieving phone numbers

Retrieving emergency numbers

Emergency phone numbers are stored on both the modem and the SIM.

To obtain all of these numbers:

Note: Duplicate numbers are returned if a number is stored on both the SIM and the modem.

1. Call **SwiGetEmergencyEntry** to retrieve a single phone number.
2. Repeat until the **MoreEntries** flag in the returned data structure is 0 (no more entries remain).

Phone book retrieval

To read the first entry from any phone book:

1. Call **SwiGetPhonebookEntry** with **ReadFromStart** = true.

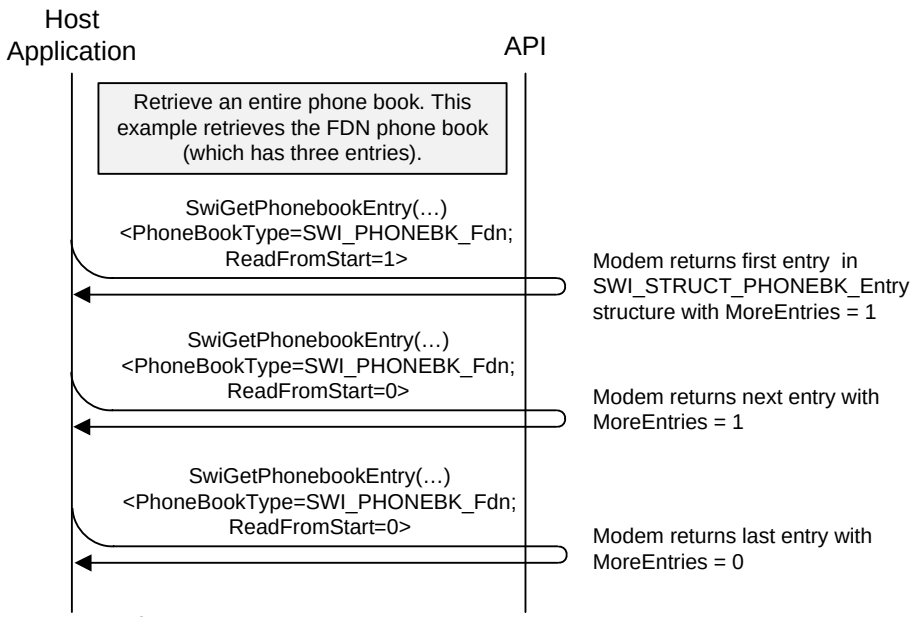
To read the next entry from any phone book (based on the most recent entry read):

1. Call **SwiGetPhonebookEntry** with **ReadFromStart** = false.

To read all entries from a phone book:

1. Call **SwiGetPhonebookEntry** with **ReadFromStart** = true.
2. Repeat (with **ReadFromStart** = false) until **MoreEntries** = false (no more entries remain).

Figure 14-19: Retrieving an entire phone book



Retrieving all entries in the ADN phone book

As soon as phone book service is available, the modem begins returning every entry in the ADN phone book if:

- The ADN phone book is available
- FDN is disabled (If FDN is enabled, only FDN entries can be used. This effectively disables the ADN phone book.)
- The **SWI_NOTIFY_PhonebookEntry** notification is enabled.

One entry is returned with each **SWI_NOTIFY_PhonebookEntry** notification, and the last entry returns with **MoreEntries** = false.

If there are no entries in the ADN phone book, a single **SWI_NOTIFY_PhonebookEntry** notification is returned with **MoreEntries** = false and **Valid** = false.

15: Modem and SIM characteristics

This chapter details several functions that allow the host application to identify account parameters, modem component details, and SIM details.

The following table provides an overview of these elements — for detailed explanations, refer to the API documentation.

Table 15-7: Component characteristics functions

Function	Module	Description
Firmware details		
SwiGetBootVersion	Modem Information and Management	Returns the version of the bootloader (a component of the firmware)
SwiGetBootloaderBuildDate		Returns the date the bootloader was built
SwiGetFirmwareVersion		Returns the version of the modem firmware
SwiGetFirmwareBuildDate		Returns the date the firmware was built
SwiGetFlashImgInfo		Returns firmware flash image details
Modem details		
SwiGetHardwareVersion	Modem Information and Management	Returns the version of the modem hardware
SwiGetUsbdInfo		Returns USB descriptor build details.
SwiGetDeviceID		Returns the device's unique identify number (the ESN or EID)
SwiGetPriInfo		Returns the device's PRI details
SwiGetDriverVersion	Driver	Returns the version of the NDIS driver
SwiGetFPGAVersion	GSM Modem Information and Management	Returns the version of the FPGA (Field Programmable Gate Array), a programmable logic chip in the modem
SwiGetIMEI		Returns the International Mobile Equipment Identity (a number that uniquely identifies every GSM device) from the modem
SwiGetSerialNumber		Returns the factory serial number (FSN) of the modem

Table 15-7: Component characteristics functions

Function	Module	Description
Modem features		
SwiGetAvailableFeatures	Modem Information and Management	Returns the modem's available features, including the PDP context type, and support for voice, tri-band
SwiGetFeatureCustomizations		Returns the modem's customizable features.
SIM identification numbers		
SwiGetGsmIMSI	GSM SIM	Returns the International Mobile Subscriber Identity (a number used to identify the account holder) from the SIM
SwiGetIccId		Returns the Integrated Circuit Card ID (a unique number used to identify the SIM) from the SIM
Available air servers		
SwiGetAvailAirServers	API and Device Management	Returns a list of available air servers
SwiSelectAirServer		Binds the modem to a specific air server

16: Error handling

This chapter describes how to handle error codes returned from API function calls.

Error codes

Refer to the API documentation (**SWI_RCODE** enumeration) for a complete list of possible error codes returned from API function calls.

Handling errors

To retrieve information about the last error returned from a control and status API function, call **SwiGetLastError**, passing a buffer to hold the transaction error.

If your application encounters errors that cannot be resolved, you can contact Sierra Wireless Technical Support for assistance. (See "Contact Information" on page 5.)