Sierra Wireless GOBI RIL Integration Guide for Windows Mobile 6.5 and later

Document #:	
Revision:	**0.5**

Table of Contents

1 General

1.1 Purpose

This document describes all the details and tips that OEM customer needs to know to smoothly and successfully integrate the “RIL” into their Pocket PC Phone Edition. In this document, RIL or RILGOBI refers to Sierra Wireless GOBI RIL and RIL UI Extensions provided with the RIL Toolkit.

1.2 Scope

This document serves as an integration guideline/specification for an OEM software development team that is integrating the Sierra Wireless GOBI RIL and RIL UI Extensions into a Windows Mobile target platform.

1.3 Modems

RILGOBI works with Sierra Wireless Gobi3K, MC9090, SL9090 and MC73xx
The last version of RILGOBI that support Gobi2K is 1.5.1

1.4 Glossary of terms

Acronyms and Definitions

Term	Definition
EM	Embedded Module
FW	Firmware of the Embedded Module
RIL	Radio Interface Layer
OEM	Original Equipment Manufacturer
PCS	Sprint PCS
VZW	Verizon Wireless
ATT	AT&T Wireless
VSP	Virtual Serial Port
NVM	Non-Volatile Memory of the EM
CDMA	Code Division Multiple Access

EVDO	Evolution Data Optimised
HSPD	High-Speed Packet Data
CEPC	x86 version of CE running on a PC
UMTS	Universal Mobile Telecommunication System
QMI	Qualcomm Messaging Interface
VID	Vendor Identification Number
PID	Product Identification Number
AKU	Adaptation Kit Update

1.5 References

Ref. #	Doc. #	Document title
R-1		Qualcomm WWAN Connection Manager Application Programming Interface (API) for Gobi 2000
R-2	2160120	RIL and USB power management v. 0.4
R-3		Gobi Activation Changes v0.1
R-4	2160098	Module Integration Guide for WinCE SDK_v1.6.pdf
R-5	2130114	CDMA and GSM / UMTS Mini Card Hardware Integration Guide v.1.9

1.6 Revision control & history

Revision	Date	Summary of changes	ECO #, if applicable
0.1	11/01/10	Original document creation	
0.2	12/20/10	Added few more registry key information. Integrated feedback from team members.	
0.5	11/07/14	Updated registry settings • Audio provisioning for MC9090 • Sprint Roaming guard • CDMA data dormancy polling • Supplementary service • Emergency number provisioning	

1.7 OEM Important Activities

☑ Throughout this document, there are critical actions or decisions for the OEM to do or make in order to successfully integrate the RIL. These are highlighted in this document with the checkbox icon shown to the left of this paragraph.

2 Components

This section describes the components that make up the RIL and any requirements associated with them.

2.1 Mandatory vs. Optional

Unless otherwise specified, each of these components must be included in the target OS image for the RIL to function properly.

2.2 Component Placement and Attributes

The four applications of the RIL described in Sections 2.3.2.1 through 2.3.2.3 are built using Win32. They can be run from ROM.

All RIL components should be run from the windows root directory (\Windows).

Unless otherwise specified, all files that make up the RIL should have their file attribute set to system and hidden. This should be done to prevent a user (or the Microsoft logo test) from accidentally launching the different applications of the RIL.

2.3 Component Overview

The section describes the components that make up the RIL and describes any special integration requirements necessary for each of the components.

2.3.1 Dynamic Link Library files

Each DLL comes with a corresponding .rel file that specifies the relocation information to be used by the platform building process. The existence of this file during the platform building process will help to reduce the image size used by each DLL in the target OS image.

2.3.1.1 Sw_utilityG.dll

This component provides a set of common utilities that are used by the other components of the RIL. The API exposed in this utility includes functions for accessing the functionality of the RIL from standalone applications and functions for RIL logging (see Section 8).

2.3.1.2 SwiGobi.dll (streaming device driver DLL)

This is the Sierra Wireless USB device driver. This driver provides support for the Windows Mobile OS mandated Cellular Line VSP, which is required for making data connections with the EM on CDMA(see Section 5.1). The driver also implements the NDIS support needed to establish data call on UMTS. RIL driver does not use the NDIS miniport interface provided by this driver.

2.3.1.3 Gobiapi.dll

This dll implements the Gobi APIs as per the Qualcomm Gobi API specification. The GobiAPIs talk to the modem via QMI (Qualcomm Messaging Interface).

2.3.1.4 SwiQDL.dll

This component is a firmware download driver. On Gobi2k, when the modem is powered up, the modem is in boot mode. It presents itself as a SwiQDL VID/PID. Hence on modem power up, SwiQDL will be loaded. SwiQDL then downloads the carrier firmware on the modem. The selection of the carrier firmware is based on the carrier preference specified under the "CarrierIndex" registry key (refer to section A.14).

2.3.1.5 rilgobi.dll

The rilgobi.dll is the main state machine engine of the RIL. It handles requests from the standard and extended UI components, sends and receives requests to and from the EM, and processes events from the EM.

2.3.1.6 SWIxxAux.dll

The SWIxxAux module handles all power control operations and synchronization between the target host system and the EM, including EM power on/off control and notification, system suspend and resume coordination with the EM.

2.3.1.7 SWDialParserG.dll

An optional component designed to receive all button press and TALK press notifications from the OS phone dialler for "special" button press processing (refer to Appendix B). This component implements many of the of special button press processing requirements of the CDMA carriers that are not implemented as part of the standard Phone Edition UI provided by Microsoft.

This component does not need to be included if the OEM wants to implement these special CDMA carrier button press requirements on their own or the OEM can get these requirements waived by the carriers.

☑ If the OEM does not need to implement their own dial parser extension and the OEM wants to include the RIL dial parser extension, then the OS registry key *DLL-OEM* must be populated with the default registry data specified in Section A.10.

☑ If the OEM needs to add their own dial parser extension to the RIL dial parser extension, the OEM dial parser extension should be registered with the OS **instead** of the RIL dial parser extension. The implementation of the OEM dial parser extension **must** include calls to the RIL dial parser extension for button press and TALK press notifications from the phone dialler UI so that the RIL may also process these events. The APIs exposed in the RIL dial parser extension for the OEM dial parser extension to call are specified in Section 11.20.

2.3.1.8 sw_emextG.dll (COM DLL)

An optional component designed to implement various UI requirements of the CDMA carriers that are not implemented as part of the standard Phone Edition UI provided by Microsoft. These UI features include support for CDMA specific Phone Setting menus, support for displaying phone status indicators in the phone dialer (roaming text, location privacy icon).

If the OEM wants to implement these UI requirements on their own or they will attempt to get these requirements waived by the CDMA carrier, this DLL does not need to be included.

☑ If the OEM wants to include the roaming call guard features of SW_EMExt.dll, the OEM must also include the SWDialParserG.dll, specified in Section 2.3.1.7.

☑ If the OEM decides to include this component, the OEM must include the contents of the corresponding *phone.changedirect.txt* optional registry file, specified in Section 4.1, as part of the OS image.

2.3.1.9 OEMAuxG.dll

The RIL requires methods to get and set various "hard" and "soft" signals of the target device in order for the RIL to communicate and control the operation of the EM as well as react to radio specific events and requests of the target device.

In order to provide a standard mechanism for the RIL to gain access to these signals, RIL defines a set of APIs defined in the header file *oemaux.h* to be implemented as part of the OEMAux.dll. This header file is included in each RIL release. See Sections 9 and 10 for the OEMAUX library feature requirements. The APIs for OEMAux library are defined in Section 11.3.

☑ The OEM must implement all APIs specified in *oemaux.h* as part of an OEMAux library for proper operation of the RIL. It is strongly recommended that this DLL be implemented as re-entrant as future APIs for this DLL may need to be called from additional components outside the RIL. The OEMAUX library name should be reflected in the registry specified in A.6.

Without the presence and full support of OEMAux library, the RIL will not be able to successfully load on an ARM target device platform.

2.3.2 Target Executables

Unless otherwise specified, all RIL executable files should have their hidden attribute enabled. A user should NOT be able to browse to one of these hidden executables and launch manually.

2.3.2.1 AAMG.exe

The AAM application is the main application of the RIL and handles coordination between the core RIL functionality and the RIL extension functionality (GobiActivation.exe, UI extensions, etc.).

☑ In order for the RIL to start up in the system, the OEM must launch AAM during system boot up. However, in order to pass logo certification, the AAM should not be launched until after the welcome screen has been displayed.

2.3.2.2 Module Activation Executables

2.3.2.2.1 GobiActivation.exe

The GobiActivation application is a standalone application that is launched through automatic triggers in the RIL or through a menu option extension in the phone dialer. GobiActivation

application is used for configuring CDMA modems only. This application allows the user to activate a modem or update the PRL on the modem. During activation or PRL update process, the application will update the UI with messages indicating the progress of the modem configuration process. There is also a silent mode of the GobiActivation. In the silent mode, GobiActivation application will run, but will not display any UI. There will be no UI updates or messages. To support silent mode, GobiActivation accepts a command line argument. Following is list of command line argument values and their purpose:

a. If 0, default behavior. UI will show up. User can chose to activate the module or just do a PRL update
b. If 1, do activation in silent mode.
c. If 2, do PRL update in silent mode.
d. If 3, GobiActivation application will query and return the activation state of the modem (i.e. active or in-active).
e. If no command line parameter is provided, it will be same as default behavior.

GobiActivation application returns a success or error codes to the calling application. Refer to [R - 3] for the list of return codes.

2.3.2.2.2 Auto-launch of GobiActivation.exe

The RIL is designed to launch the EM activation process automatically whenever it detects an EM that has not been activated. This is to make the out-of-box experience for the user of the device smooth in terms of helping them get through the activation process so that they can quickly begin using the EM to make voice and/or data connections.

☑ If the OEM needs to disable the Auto-launch of GobiActivation.exe when an inactive EM is detected, the registry key *DisableAutoActWiz,* Specified in Section A.7, needs to be created and set to enabled by setting the data field to 1. By default, this registry key is not included in the RIL image, thus allowing the auto-launch of GobiActivation application. Setting the registry key *DisableAutoActWiz* to 0, will also allow the auto-launch of GobiActivation application.

2.3.2.3 PhoneUtilityG.exe

The Phone Utilities is an optional standalone application that supports many of the hidden menu UI requirements of the CDMA carriers. Refer to Appendix B for the set of hidden menus supported by the Phone Utilities. The Phone Utilities application is triggered by the RIL Dial Parser Extension, described in Section 2.3.1.7, and thus requires the RIL Dial Parser Extension to receive button press events to work properly.

2.3.3 Test and Utility Executables

2.3.3.1 RilTest.exe

This is a utility for testing of all RIL API functionality with the capability to capture RIL notifications during debugging. Typically the OEM does not use this utility but could in the case an issue related to RIL notifications needs to be debugged remotely by the OEM. This utility **must not** be included in any final release images.

2.3.4 Images

Microsoft UI supports the replacement or disabling of the default carrier logo image in the phone dial pad. In order to support multiple carriers, two carrier specific gifs are provided with the RIL, swcarrier1.gif and swcarrier2.gif. These images are the images for Sprint PCS and Verizon Wireless, respectively. In the future, if additional carriers require display of their own carrier logo, support can be added to the RIL for this. Refer to Section 6.4 for details on the usage of these images.

3 Builds

Each major release of the RIL will include two ARM builds, debug and release, targeted for a specific Windows Mobile platform.

3.1 Platform

The RIL is currently built to work with Windows Mobile 6.1 AKU1.5.2(build 20963) and Windows Mobile 6.5 AKU 0.6.0 (build 20954).

3.2 ARM

There are two ARM builds, debug and release, to be used on the ARM based target device.

3.2.1 Debug vs. Release

Both the debug and release builds have the capability to log RIL messages to a file (see Section 8). However, the release build only logs critical level messages to the file. In addition, the sw_utility ARM debug build also uses RETAILMSG when logging RIL messages. Given this, the OEM should only use the ARM debug build for debugging purposes. The ARM release build has a smaller footprint and is optimised since it is not sending excessive logs into the system and log file.

All RIL components are signed with the default certificate that comes with the AKU. The OEM might need to resign them with their desired certificate.

☑ It is strongly recommended that the OEM also release an ARM debug build, with DEBUGMSG routed to the console window, with each ARM release build in order to allow Sierra to investigate and debug any problems that may arise.

4 Registry Keys

The RIL requires a number of registry keys for configuration and runtime operation. The OS defines some of these registry keys, the Microsoft standard RIL defines some of these, and many are defined for Sierra specific RIL functionality.

4.1 Registry Image Files

To ease the integration of the RIL registry keys into the target OS image, five registry files, *SwiGobi.reg, swisdk.reg, sw.reg, SwiQdl* and *phnext.reg* are provided in the root directory of the RIL release package that contain all of the default registry keys, values, and data required for default operation of the RIL.

The *sw.reg* file contains mandatory registry keys for RIL operation. However, the *phnext.reg* file contains the registry settings required for the UI extensions of the RIL and does not need to be included in the target OS image if the sw_emextG.dll and SWDialParserG.dll, described in Section 2.3.1.8 and 2.3.1.7 respectively, are both not included.

A CAB package with all the required registry keys is also provided for quick and easy installation of the driver.

4.2 OEM Usage

All of the registry keys and values defined for the RIL can be broken into three groups in terms of how they are used by the OEM. A particular registry key may have registry values that fall under more than just one of these groups.

4.2.1 Don't Care

The majority of the RIL registry keys and values are only used internally by the RIL. For these keys and values, there is no interaction required by the OEM, other than just including these in the target OS image, as described in the beginning of this section. This group of registry value and keys are, for the most part, not described in this document and should never be modified by the OEM.

4.2.2 Read-only

Some of the RIL registry keys and values contain useful information for the OEM (e.g. port number of the "Cellular Line" VSP), but should never be modified by the OEM during creation of the OS image or runtime. These registry keys and values are described in this document and are listed as part of Appendix A.

4.2.3 Configurable

Some of the RIL registry keys and values ship with default data, but require the OEM to review and configure this data if the default values do not match up with the requirements of the target system. Some of these settings include critical information like the communication port numbers and names used by the RIL to talk to the EM. Some of these settings allow for the enabling, disabling, or configuring the various features of the RIL. Each of these configurable registry keys and values are described in this document and are listed as part of Appendix A.

Since many of these configurable registry values and keys are read by the RIL at system boot up, the OEM should make sure they are not modified during runtime.

☑ The OEM should review and determine the target device requirements for the configurable registry keys and values of the RIL and modify as needed to generate the OS image of the target device.

4.2.3.1 Disabling RIL UI Extension Features

For a particular target device, it may be desired to enable certain RIL UI Extension features, while disabling others. The registry value *DisableUIExt*, specified in Appendix A.7, allows the OEM to pick and choose which UI Extensions are disabled by the RIL and which are not. If this registry value is not present, all UI extension will be enabled as long as their required DLLs and applications are included in the OS image.

☑ The OEM can disable specific RIL UI extension features when building the target OS image by creating and configuring the *DisableUIExt* registry value.

5 Communication Ports

There are several communication ports associated with RIL operation that require some integration by the OEM with the target platform.

5.1 Cellular Line VSP (RIL VSP Data Port in CDMA mode)

Windows Mobile OS requires the RIL to present the modem name of the EM as “Cellular Line” (so called RIL VSP feature). The OS determines the mapping for the Cellular Line modem to the RIL VSP through the registry keys specified in Section A.3 and Section A.2.

The default registry for RIL VSP Data Port is SWI9, which is EM’s ATPPP Port. SWI9 port will disappear when EM is powered off or going through a modem reset or USB bus reset (see further detail in [R-4]). In RILGOBI, the VSP Data Port SWI9 is used to establish PPP data connections only in CDMA mode. In UMTS mode, SWI9 is used just as a AT port.

- ☑ For the OS to talk to the Cellular Line exposed by the RIL VSP, the OEM needs to include the registry keys, specified in Sections A.3 and A.2, when building the OS image for the target device.

5.1.1 How does Cellular Line modem work in CDMA mode

When MS OS receives a request to establish a data call via a connectoid that uses Cellular Line modem, MS will first call RIL_Dial (<phone number>, type = Data Call). The <phone number> should be specified upon the creation of the connectoid and should be #777 for Gobi CDMA technology

Upon receiving this request, RIL will open the EM ATPPP port (determined from the Section A.3) and send the ATD<phone number> to it. RIL will then have to wait infinitely (until RIL_Hangup or RIL_Manage is called) for a EM’s response on the EM ATPPP port with either “CONNECT” or “NO CARRIER” or INVALID port handle event (as a response to RIL_Hangup or RIL_Manage function) before returning the RIL RESULT to the RIL_Dial function. Before the RIL composes the RIL RESULT to the upper layer, the RIL has to close the EM ATPPP port. Before closing the EM ATPPP port, the RIL should call `EscapeCommFunction(m_hPortHandle,CLRDTR)` for all cases except for the case where the waiting result is “CONNECT”. Clearing DTR on the “CONNECT” case will cause the EM to disconnect the “just established” data call which we should avoid. This is the reason why the RIL is using SWI9 instead of WMP9 when talking to the EM ATPPP port because closing the WMP9 handle implicitly triggers an automatic DTR de-assertion to the modem.

The port handle becomes invalid when MS OS sends the request to release the data call via the RIL_Manage function.

Upon receiving the RIL_RESULT_OK from the RIL, MS OS will open the RIL VSP data port (determined from the Section A.2) to start PPP streaming. This is the reason why the RIL should close the EM ATPPP port before returning the RIL_RESULT.

5.2 NDIS Port

This port is used by RIL to establish NDIS data connections in UMTS mode.

5.3 EM ATPPP Port

All PPP data connections using the EM are established through the EM ATPPP port. On Gobi, this port is used to establish a PPP data call in CDMA mode only. AT port is consumed by RIL in UMTS mode. RIL has to send certain specific AT commands in order to retrieve information which is not available via QMI in UMTS mode.

- ☑ For the RIL driver to talk to the EM ATPPP port, the OEM needs to include and configure the registry key specified in Section A.3, when building the OS image for the target device.

5.3.1 Port Name (Prefix) and Port Number (Index)

In order to communicate to the EM ATPPP port, the RIL needs to know the communication port name and number of the EM. The RIL determines this from the registry values *Index* and *Prefix*, Specified in Section A.3. The default registry data shipped with the RIL might be repopulated by the Sierra Wireless USB client driver to reflect the actual port name and number supported by the Sierra Wireless USB client driver.

5.3.2 Serial Baudrate (not applicable to Sierra Wireless USB client driver)

If the OEM needs to talk directly with the EM ATPPP port, this should be done through the RIL VSP, specified in the *DataPort* registry value shown in Appendix A.2. By using this VSP value, the OEM does not need to be concerned with the baudrate and handshaking on the port, as the VSP will automatically handle control of these features.

5.4 EM QMI Port

The RIL receives status and events notifications from the EM on the EM QMI port. The RIL also uses this channel to issues commands like sending SMS, querying RSSI, MDN, etc. This port is used by GobiAPI.dll. RIL driver does not directly talk to QMI port.

5.5 Diagnostic Port

Provisioning and debugging tools running on a PC need to talk directly to the EM using the Pass Thru Mode functionality of the RIL. To support this, the RIL will call the OEMAUXG (section 11.3.7) library to engage/disengage the Diagnostic mode. Upon receiving this request, OEMAUX library should connect the modem's UART1 to its external serial port so that a diagnostic tool can talk through it. When diagnostic mode is disengaged, the RIL will reset the modem.

Depending on the modem's configuration, diagnostic command and response can be served EITHER via the modem's UART1 OR via the modem's USB endpoint exclusively. If the modem is configured to serve diagnostic mode over the USB's endpoint then the OEM should route the virtual serial port (SWI5) to the external addressable COM port instead. On Gobi, there is no UART1 to capture DM data. The diagnostic command and response can be served only via modem's USB endpoint exclusively.

5.6 NMEA Port

NMEA data from the modem can be captured by listening to the virtual serial port SWI6.

6 Carrier Configuration

This section is applicable to CDMA mode only. The RIL package supports multiple carrier configurations within a single release in order to meet certain carrier specific requirements. The three carrier configurations currently support by the RIL are Sprint PCS (PCS), Verizon Wireless (VZW), and Generic.

Whenever the RIL is initialised, the *CarrierSuffix* registry value, specified in Section A.7, is used by the RIL to determine what carrier configuration the RIL should be initialised with. As a result, the behavior and appearance of the RIL will behave according to any specific requirements of the current carrier configuration. For example, the default packet data Connection Profile will be updated to reflect the attributes required for a packet data connection with the current carrier (see Section 7).

☑ ***To support the carrier configuration functionality of the RIL, the OEM needs to include and configure the registry key, specified in Section A.7, when building the OS image for the target device.***

6.1 Dynamic vs. Static Mode

The current carrier configuration of the RIL can be set to dynamic or static mode via the *DynamicCarrierSupported* registry value, specified in Section A.7. Setting this mode to static will allow the OEM to manually set the carrier configuration of the RIL. Setting this value to dynamic will allow the RIL to dynamically set the carrier configuration based on the configuration detected in the EM at power up.

☑ The RIL ships with the dynamic carrier configuration feature enabled. The OEM will need to overwrite the *DynamicCarrierSupported* registry value if this feature needs to be disabled. Sierra recommends to have dynamic carrier configuration feature enabled.

6.2 Dynamic Carrier Configuration

If the carrier configuration is set to dynamic, the RIL will dynamically determine and set the *CarrierSuffix* registry data to the correct carrier configuration when the EM is powered up by the RIL.

If RIL detects an EM that is configured specifically for PCS or VZW, then the RIL will automatically update the *CarrierSuffix* registry value to "PCS" or "VZW" respectively, thus allowing the RIL to initialize with the matching carrier configuration supported by the EM. If the RIL detects an EM that is not specific for PCS or VZW, then RIL will update the *CarrierSuffix* registry value to "SW" thus allowing the RIL to initialize with the Generic carrier configuration.

☑ The RIL ships with the current carrier configuration set to UNDEFINED. If the OEM changes the carrier configuration mode to static mode, the OEM will need to manually set the carrier configuration through the *CarrierSuffix* registry value.

6.3 Static Carrier Configuration

If the carrier configuration is set to static, the RIL will initialize with the carrier configuration specified in the *CarrierSuffix* registry value without checking the FW load in the EM. In this mode, the carrier configuration must be configured manually by changing the data value of the *CarrierSuffix* registry value.

6.4 Carrier Logo Image

A carrier logo image is displayed in the main phone dialer pad. This image is displayed by the OS and is initialised in the system at cold/warm boot time. The OS expects this image to be called *carrierlogo.gif* and to be located in the root Windows directory. A default image is provided with the OS that can be replaced by the OEM with an actual carrier image. However, in order to support multiple non-generic carriers, the RIL contains an image for each non-generic carrier configuration (i.e. Sprint and Verizon) and will attempt to overwrite the default image file with a carrier specific image when the carrier configuration is set to a non-generic value. If the current carrier configuration is set to Generic, the default carrier logo image file from the system will be displayed.

☑ In order for the RIL to be able to overwrite the default carrier logo image, the OEM must change the attribute of the default *carrierlogo.gif* file in the root Windows directory from a hidden system file to an unhidden system file. This will allow the RIL to overwrite the file when needed. Another option is for the OEM to not include the default *carrierlogo.gif* file in the target OS image.

6.5 OS Limitation to Runtime Carrier Configuration change

Unfortunately, there are some limitations with the OS adjusting to a carrier configuration change on the fly. As a result, if the carrier configuration of the RIL is changed (by changing the carrier suffix in static mode, or changing the EM carrier configuration in dynamic mode) it is required that a warm or cold boot is performed on the target device after this carrier configuration change is made, in order for the OS to fully adapt to the change.

7 Connection Profiles

The EM can be used to make two different types of data connections: A High-Speed Packet Data (HSPD) connection and a Circuit Switched Data (CSD) connection. With RILGobi, CSD is only supported in CDMA mode. In UMTS this is not supported because the AT port is consumed by RIL driver for its own use. Both of these connection types require the use of a connection profile to be stored and configured in the connection manager of the OS.

- ☑ In RILGOBI, the connection profiles used for HSPD and CSD are **not** automatically created by the RIL driver. They have to be created and maintained by the OEM or the end customer.
- ☑ The "Cellular Line" modem (see Section 5.1) must always be used for all connections profiles that are to use the EM to make the data connection in CDMA mode. "Cellular Line (GPRS)" modem must be used in UMTS mode.

7.1 High-speed Packet Data (HSPD)

For all known current CDMA carriers, there is only one HSPD connection per device/account to be used by the end customer for connecting to the Internet.

The HSPD connection will not connect successfully until the EM has been successfully activated (i.e. successfully ran GobiActivation for CDMA or using active SIM for UMTS) and data provisioned (i.e. successful IOTA or OMADM session for Sprint) for the network associated with the current carrier configuration.

Due to OS limitations, there is no protection in place for a user to manually corrupt the HSPD connection profile from the connection settings menu of the Windows Mobile UI. Given this, the RIL provides a phone settings menu option button to allow a user to request the RIL to repair/reset the HSPD connection profile at anytime. This option (Repair Connectoid) is a feature of the SW_EMExtG.dll, specified in Section 2.3.1.8. This option is only available in CDMA mode.

7.2 Circuit Switched Data (CSD)

- ☑ This feature is not supported by RILGOBI driver.

8 RIL Logging

For debugging purposes during development and in the field, the RIL provides a mechanism for logging RIL debugging messages into a text file. The logging functionality is controlled by the registry key specified in Section A.12.

☑ To support the logging functionality of the RIL, the OEM needs to include and configure the registry key, specified in Section A.12, when building the OS image for the target device.

There are a few different configuration options for RIL logging.

8.1 Log File Name and Path

The path and the name of the log file are specified in the registry value *Default Log File*, specified in Section A.12. The OEM could change this path and name if needed, but it is recommended that the default path and name, provided with the RIL, be used for consistency.

8.2 Log File Size

The log file size is limited, in bytes, by the registry value *Size - Log File Size*, specified in Section A.12. Once the log file reaches this maximum size, SS_Log.log is closed and it is renamed as SS_LogOld.log. Any new log messages are logged into SS_Log.log file which is created again.

OEM: The default registry data for the *Size - Log File Size* registry value can be overwritten by the OEM, when building the OS image for the target device, if it needs to be changed.

8.3 Log Level

The level of logging can be controlled via a registry key “Filter - Status LogLevel” described in Section A.12. The default value of “Filter - Status LogLevel” is 14 (0xe).
There are four types of log messages:
Version – The version of the DLL or an application is logged in the “SS_Log.log”.
Critical – Very critical messages which help in the initial debugging.
Warning – These are messages which are less important than the Critical messages but help in debugging at a very detailed level.
Trace – These provide very detailed information about all the activity happening in any application or a DLL. Enabling Trace messages will slow down the system.

☑ Trace and Warning messages are not supported in the ARM RETAIL release. To log these messages from any application or a DLL in the “SS_Log.log” file, an OEM should use the debug version of that particular application or a DLL. OEM should also use the appropriate mask for the “Filter - Status LogLevel” registry key described in Section A.12.

☑ In order to route any messages to the UART port, OEMs should use debug version of sw_utility.dll.

8.4 OMALog.log

The log file is used by OMA and FUMO activation procedures of non-RUIM CDMA carriers. This file contain useful information for trouble-shooting.

9 Power Control

☑ For detailed information about the Host device power state and USB bus power state implementation, please refer to [R-2]. The current section has more details about the modem power state implementation.

9.1 MIO Signals

There are a number of hardware signals on the EM that need to be accessed by the RIL in order to support the requirements of power control. The power control hardware signals of the EM are summarised in following table.

Table 9-1: EM Signal Summary

Signal Name (OEMAux.h name)	*Signal Direction	Interrupt	Between RIL and AUX API	Between AUX API & HW
Power Signals				
W_DISABLE_N (nON_OFF)	Out	N/A	Active LOW - SetMIO( ) called with SignalState = LOW to to turn ON EM - SetMIO( ) called with SignalState = HIGH to turn OFF EM	To turn OFF EM, set W_DISABLE_N to 0 and then removing host supply power on VCC_3.3V. To turn ON EM, apply the host supply power on VCC_3.3V and set W_DISABLE_N to floating.
(POWER_ON_BUTTON)	In	N/A	Soft signal. - GetMIO returns HIGH if OEMAUX is enabling power to the EM	N/A

***** Signal direction is from the perspective of the target device.

Please refer to reference [R-5] for more details of these hardware signals. In addition to these "hard" signals, there are "soft" signals that have been defined by the RIL that requires OEMAUX support.

A common set of APIs, defined in Section 11.3, are defined by the RIL to access both theses "hard" and "soft" MIO signals.

9.1.1 nON_OFF (hard signal)

This is an output signal to the EM for the RIL to power ON/OFF the EM. This physical output signal to the EM is active LOW. This output signal will be controlled by the RIL through the SetMIO API, specified in Section 11.3.1.

☑ The OEM must support the SetMIO API to turn ON/OFF the EM with the physical W_DISABLE_N output signal.

9.1.2 POWER_ON_BUTTON (soft signal)

This is a "soft" output signal that the RIL can use to query the OEMAUX's GetMIO API, specified in 11.3.2, to determine if OEMAUX is enabling the power to the EM.

9.2 RIL API method to control the modem's power state

As MS specifies, a RIL application should always call RIL_Initialize first to get a handle to the RIL (hRIL) before calling any RIL API.

9.2.1 To turn on the modem

Call RIL_SetEquipmentState (hRil, RIL_EQSTATE_FULL) to turn on the modem.

9.2.2 To turn off the modem

Call RIL_SetEquipmentState (hRil, RIL_EQSTATE_MINIMUM) to turn off the modem gracefully.

Graceful power down will help to prevent any risk of lost of data in the EM and allow the EM to deregister with the network (when registered). The typical power down time of the EM using this method is 5 to 10 seconds.

A forced shutdown implies calling OEMAUX's SetMIO API directly. The RIL has its internal logic to perform a forced shutdown when OEMAUX's GetMIO (POWER_ON_BUTTON) returns power off state while its internal RIL power state is recorded as power on.

9.2.3 To reset the modem

Call RIL_SetEquipmentState (hRil, RIL_EQSTATE_UNKNOWN) to soft reset the modem gracefully.

Graceful soft reset will help to prevent any risk of lost of data in the EM.

A forced reset implies calling OEMAUX's SetMIO (nON_OFF) API directly to force a power down and power up immediately. The RIL has its internal logic to perform a forced reset when it detects a possible communication lost with the modem.

9.2.4 Sample code for power off EM via RIL

Below is sample code for a graceful power off of the modem utilising the standard RIL APIs.

Quickly register to the RIL without any callback or notification, if not already registered, and call RIL_SetEquipmentState.

```
// Need to link with the RIL.lib
#include "ril.h"

HANDLE hResult;
hResult = RIL_Initialize(1, NULL, NULL,  0, 1, &hApi);
if (hResult == 0)
{
        RIL_SetEquipmentState (hApi, RIL_EQSTATE_MINIMUM);
        RIL_Deinitialize (hApi);
}
```

9.3 Handshaking

In order to properly synchronise the data flow between the EM data and control ports and allow for maximum power consumption optimisation, the target device must implement the handshaking protocol specified in reference [R-4], which is fully supported by the EM.

9.3.1 Serial Handshaking

No longer supported by the EM module.

9.3.2 USB Handshaking

USB port suspend/resume is fully supported by the EM and Sierra Wireless USB client driver. The registry name “IdleTimeOut” specified in [R-4] is the key to turn on USB port suspend/resume. By default, this registry key is set to 0 which means to turn off the feature. Once the OEM confirms that the USBD’s API SuspendDevice and ResumeDevice (under ohci2 layer) function correctly, then the OEM can set IdleTimeOut to the value 3000 (0xBB8) so that the USB client driver will call SuspendDevice after IdleTimeOut millisecond of USB bus inactivity to cause a USB bus suspend and call ResumeDevice to resume the USB bus when there are data to be transmitted to the modem.

☑ The OEM must review the MS’ SuspendDevice and ResumeDevice implementation carefully before turning on IdleTimeOut registry as these reference codes seem to not function correctly (causing the bus to NOT suspend or resume as expected).

10 System Time Auto-Update with CDMA Time

The RIL has the capability to auto-update system time of the target device with the CDMA time received from the CDMA network. There are two ways to auto-update the system time.

10.1 RIL auto-updates the system time by itself

This is a legacy way of updating the system time. Earlier Microsoft did not have a well-defined notification to update the system time. Hence we came up with our way of updating the system time based on the time received from the CDMA network. RIL allows this feature to be configured in three different ways through the registry key *AutoUpdateTimeUI*, specified in Section A.11.

a. RIL never auto-updates system time with CDMA time changes
b. RIL always auto-updates system time with CDMA time changes
c. Enable a user option in the UI settings menu to allow the user to enable or disable this feature (requires SW_EMExt.dll, specified in 2.3.1.8).

This option is defaulted in the RIL to never auto-update system time with CDMA network time (option a.).

☑ The OEM should configure the auto-update of system time with CDMA time option preference, via the *AutoUpdateTimeUI* registry key, when building the OS image for the target device.

10.2 RIL supports RIL_NOTIFY_ NITZ

Windows mobile 6.0 onwards, Microsoft introduced Network Identity and Time Zone (NITZ) Support. Though RIL has the capability to support NITZ, currently on Gobi2K, the modem does not provide correct time information. Hence RIL_NOTIFY_NITZ notification is not sent by the ril driver.

☑ Currently none of these methods to auto-update the time will work because on Gobi2K, the firmware does not report correct time information.

11 RIL APIs

11.1 Device Specific IOCTLs

Sierra RIL provides many pre-defined device specific IOCTLs. OEMs may find them very useful. The list of device specific IOCTLs implemented by Sierra RIL can be found in file "RILExt.h". This file is released to all OEMs. They can write applications to call any of these IOCTLs into the RIL driver and use the information returned by these IOCTLs. Below is an explanation of how to read/understand the "RILExt.h" file.

11.1.1 Understanding the declaration of a device specific IOCTL:

The notation followed to declare an IOCTL is "DS_IOCTLNAME" where DS stands for Device Specific and IOCTLNAME is a self explanatory name which explains the purpose of the IOCTL. For Example: DS_GET_VERNO_INFO – It is a device specific IOCTL which is called to Get the Version Information.

Every IOCTL is defined as a constant integer value.
For Example: #define DS_GET_VERNO_INFO 45
To implement a new device specific IOCTL, it should be declared at the end of the list of IOCTLs in "RILExt.h" . The integer constant assigned to the new IOCTL sould be one greater than the previous IOCTL in the list .i.e. if the previous IOCTL's constant integer value is X, then the new IOCTL in the list will have a value of X+1.

A typical declaration of the IOCTL looks like:

```
#define DS_IOCTLNAME  X
// Command:
// ========
// No Param ---- This means there is no input parameter.

// Response:
// =========
// pointer to a return structure or standard RIL_RESULT_ERROR or
// RIL_RESULT_OK
```

For example:

```
#define DS_GET_VERNO_INFO                45
// Command:
// ========
// No Param

// Response:
// =========
// pointer to RILVERNOINFO structure or standard RIL_RESULT_ERROR
```

The definition of RILVERNOINFO can be found in the "RILExt.h" file itself. Note that some IOCTLs may have input parameter(s).

11.1.2 How to call a device specific IOCTL:

To call a device specific IOCTL into the RIL, your application must be a client of RIL driver. Then a device specific IOCTL can be called by using the function RIL_DevSpecific(). More information on this function can be found on MSDN.

11.2 RIL Dial Parser Extension APIs

The APIs for the RIL Dial Parser Extension are a one-to-one mapping of the MS APIs so that the OEM can handle the MS APIs for OEM specific processing and still allow RIL Dial Parser Extension to perform its own processing in parallel.

11.2.1 SW_IsImmediateDialRequestRequired

Purpose:
Give Dial Parser Extensibility a chance to decide whether the framework should regard the digits collected so far as an actionable dial string.

Parameters:
BOOL SW_IsImmediateDialRequestRequired (
LPCTSTR lpszDialString
)

[in] lpszDialString - The digits collected so far

Return Value:
- TRUE if lpszDialString is a complete dial request, meaning that no additional digits need to be collected and the Talk button does not need to be pressed in order for a meaningful dial action to take place.
- FALSE otherwise

11.2.2 SW_PreprocessDialRequest

Purpose:
Give Dial Parser Extensibility a chance to intercept the Dial Request. If the DLL chooses not to, the framework will send the Dial Request to RIL. The implementer can take whatever action is appropriate, including nothing at all, before returning. The implementer may wish to display a Message Box, or may wish to use tapiRequestMakeCall to issue a call to a number other than the one that was entered, etc.

Parameters:
```
DWORD SW_PreprocessDialRequest (
        HWND hWnd,
        LPCTSTR lpszDialString,
        BOOL fSecure
        )
```

[in] hWnd - A window handle, typically used as a parent window for MessageBoxes
[in] lpszDialString - The digits collected so far
[in] fSecure - Whether the string represents a secure call request

Return Value:
- DPEF_ALLOW_DIAL_REQUEST - the framework should do whatever it would normally do
- DPEF_ABORT_DIAL_REQUEST - the framework should assume that the dial request has been handled and stop processing it further

11.3 OEM Auxiliary APIs

11.3.1 SetMIO

Purpose:
Used by the RIL to set the state of the specified MIO "hard" output pin of the EM.

Parameters:
```
RetStatus SetMIO (
        OutputSignal eSignal,
        SignalState estate
        )
```

[in] OutputSignal eSignal - the output pin to set state for

```
enum {
        nON_OFF,                    // Power ON/OFF EM (1=OFF, 0=ON)
} OutputSignal;
```

[in] SignalState eState - the new state

```
enum {
        HIGH,       // a change from LOW to HIGH state
        LOW,        // a change from HIGH to LOW state
} SignalState;
```

Return Value:
- OK_STATUS if the specified output pin was successfully set to the specified state
- INVALID_PRM_ERR otherwise

```
enum {
        OK_STATUS,              // no errors
        TIMEOUT,                // timeout
        INVALID_PRM_ERR         // error in parameters
} RetStatus;
```

11.3.2 GetMIO

Purpose:
Used by the RIL to get the current state of the specified MIO “soft” input signal from the EM or target device.

Parameters:

```
RetStatus GetMIO (
        InputSignal eSignal,
        SignalState* estate
        )
```

[in] InputSignal eSignal - the input pin to be read from the EM; this API is only used to get POWER_ON_BUTTON state)

```
enum {
        POWER_ON_BUTTON
} InputSignal;
```

[in, out] SignalState* eState - pointer to SignalState, which is allocated and freed by caller

```
enum {
        HIGH,           // EM should have power
        LOW,            // EM should not have power
} SignalState;
```

Return Value:
- OK_STATUS if the specified input pin current state was successfully read
- INVALID_PRM_ERR otherwise

```
enum {
        OK_STATUS,              // no errors
        TIMEOUT,                // timeout
        INVALID_PRM_ERR         // error in parameters
} RetStatus;
```

11.3.3 GetAudioPath

Purpose:
Used by the RIL to get the current status of the audio path. This function is called every time RIL_SetAudioDevices is called. If voice call feature is not supported on the product, RIL driver will return error.

Parameters:
RetStatus GetAudioPath (
    eAudioPath * pAudioPath
)

[Out] pAudioPath - the current audio path;

```
enum {
    HANDSET_PATH,
    HEADSET_PATH,
    BLUETOOTH_PATH,
    SPEAKER_PATH,
    TTY_PATH,
    HAC_PATH
} eAudioPath;
```

Return Value:
- OK_STATUS if the specified output pin was successfully set to the specified state
- INVALID_PRM_ERR otherwise

```
enum {
    OK_STATUS,          // no errors
    TIMEOUT,            // timeout
    INVALID_PRM_ERR     // error in parameters
} RetStatus;
```

11.3.4 EnableAudio (previously called CDMAPhoneComeIn)

Purpose:
This API is called by the RIL whenever the state of the audio path from the EM needs to be changed. If voice call feature is not supported on the product, RIL driver will return error.

Parameters:

```
void EnableAudio (
        BOOL bEnable
        )
```

[In] bEnable – Set to TRUE to enable the audio path from the EM, FALSE to disable.

Return Value:
None

11.3.5 GetVolumeLevel

Purpose:
This API is called by the RIL whenever RIL_GetAudioGain is called. The OEMAUX should return the TX and RX audio level for the current active audio path. If voice call feature is not supported on the product, RIL driver will return error.

Parameters:

```
RetStatus GetVolumeLevel (
        DWORD *dwTXAudioLevel,
        DWORD *dwRXAudioLevel
        )
```

[Out] dwTXAudioLevel – TX (speaker) audio level of the current active audio path.
[Out] dwRXAudioLevel – RX (micro) audio level of the current active audio path.

Return Value:
- OK_STATUS if the specified output pin was successfully set to the specified state
- INVALID_PRM_ERR otherwise

```
enum {
        OK_STATUS,              // no errors
        TIMEOUT,                // timeout
        INVALID_PRM_ERR         // error in parameters
} RetStatus;
```

11.3.6 SetVolumeLevel

Purpose:

This API is called by the RIL whenever RIL_SetAudioGain is called. The OEMAUX should apply the TX and RX audio level to the current active audio path. If voice call feature is not supported on the product, RIL driver will return error.

Parameters:
RetStatus SetVolumeLevel (
    DWORD dwTXAudioLevel,
    DWORD dwRXAudioLevel
)

[In] dwTXAudioLevel – TX (speaker) audio level of the current active audio path.
[In] dwRXAudioLevel – RX (micro) audio level of the current active audio path.

Return Value:
- OK_STATUS if the specified output pin was successfully set to the specified state
- INVALID_PRM_ERR otherwise

```
enum {
    OK_STATUS,          // no errors
    TIMEOUT,            // timeout
    INVALID_PRM_ERR     // error in parameters
} RetStatus;
```

11.3.7 SetTetheredMode

Purpose:
This API is called by the RIL whenever `DS_START_DM_MODE_CID` is called. The purpose of this API is to route the modem's UART1 port to external port.

Parameters:
RetStatus SetTetheredMode (
    BOOL isEnable,
    void *pDeviceHandle
)

[In] isEnable – TRUE to start routing the EM's UART1 hardware pins to an external UART for PC DM logging purpose. System auto-suspend is automatically disabled. FALSE to stop routing the EM's UART1 hardware pins to an external UART for PC DM logging purpose. System auto-suspend is automatically reengaged.

[In/Out] pDeviceHandle – Pointer is NULL if the caller doesn't expect the OEMAUX to open any external port and pass it up to the caller for routing data. Instead, OEMAUX is responsible to route the DM from the EM to the external port. Pointer is not NULL if the caller is responsible for routing DM data. In this case, OEMAUX should either return a NULL content of the pointer along with TIMEOUT for the RetStatus if OEMAUX cannot comply to this feature. Otherwise,

OEMAUX should return the handle of the opened port and let the caller to entirely manage the port (Read/Write and Close).

Return Value:
- OK_STATUS if the specified output pin was successfully set to the specified state
- INVALID_PRM_ERR otherwise

```
enum {
        OK_STATUS,              // no errors
        TIMEOUT,                // timeout
        INVALID_PRM_ERR         // error in parameters
} RetStatus;
```

11.3.8 AUX_GetVersion

Purpose:
This API is called by the RIL whenever OEMAUX library is loaded (so that the OEMAUX's version can be fetched and displayed in the RIL log). It also can be queried via the RIL_DevSpecific (DS_GET_VERNO_INFO) command.

Parameters:

```
BOOL AUX_GetVersion (
        LPAUX_VERSION_INFO pVersionInfo
        )
```

[Out] pVersionInfo – Point to a structure with one DWORD field where HIWORD is the major version and LOWORD is the minor version.

Return Value:
- TRUE if successful

12 Carrier Switching

Gobi modem can be used in CDMA or a UMTS mode. It can support multiple CDMA and UMTS carriers. Qualcomm provides pre-certified firmware for each of the following carriers: Vodafone, Verizon, Sprint, ATT, TMobile, Generic UMTS, Telefonica, Telecom Italia, Orange and Alltel. To switch between network technologies (CDMA and UMTS) or to switch between different carriers, the firmware image on the modem needs to be updated. This can be achieved by using following methods:

12.1 Manually changing the registry key

a. Change the "CarrierIndex" registry key listed in section A.14 by using a remote registry editor or any other tool.
b. Reset the modem. After the reset, SwiQDL driver will read the CarrierIndex registry key and download the corresponding carrier firmware image onto the modem. The carrier

firmware location is specified in the “FirmwareLocation” registry key listed in section A.14.

12.2 Using CarrierSwitch application

a. Sierra provides a “CarrierSwitch” application. This application will allow the user to select the carrier from a drop-down menu list. User then has to click on the “Switch” option.
b. This application will then reset the modem. On modem arrival, SwiQDL driver is loaded and it downloads the firmware image of the user selected carrier on the modem.

12.3 Using FirmwareUpgrade2K API

Refer to [R-1] for the details about FirmwareUpgrade2K GobiAPI.

A. PLEASE REFER TO THE GOBIAPI DOCUMENTATION FOR THIS API.REGISTRY KEYS

The following tables list the RIL registry keys of interest to the OEM.

NOTE: HKLM is short for HKEY_LOCAL_MACHINE

OEM Column Key:
- DC (Don't Care) – The OEM should not be concerned with this registry value and must not change its value (see Section 4.2.1). OEM can assume all RIL registry keys not included in this Appendix are don't cares.
- RO (Read-Only) – The OEM should not change this registry value but may need to read it for certain features (see Section 4.2.2).
- CFG (Configurable) – The OEM can configure this registry value in the target OS image as needed to support the feature requirements of the target device (see Section 4.2.3).

A.1 HKLM\Driver\BuiltIn\RIL

OEM	Field	Type	Default value	Description
DC	Dll	String	Rilgobi.dll	The name of the DLL the OS uses to support RIL GOBI Driver
DC	Prefix	String	RIL	The port name of RIL driver
RW	Order	DWORD	2	The loading order of the DLL from the 'dll' field
DC	Index	DWORD	1	The port number of RIL driver

A.2 HKLM\SOFTWARE\Microsoft\RIL

OEM	Field	Type	Default value	Description
DC	DataPort	String	SWI9:	The OS reads this key to determine that Cellular Line is supported over SWI9 port.

A.3 HKCU\Software\Sierra Wireless Inc\RIL

OEM	Field	Type	Default value	Description
DC	DataDormancyPollingInterval	String	0x1388	Interval (in ms) for RIL to check for CDMA data dormancy status. Not recommended for production environment.

A.4 HKCU\Software\Sierra Wireless Inc\RIL\CDMAData

OEM	Field	Type	Default value	Description
CFG	Index	DWORD	9	Communication port index for the EM ATPPP

				port
RO	Baudrate	DWORD	0x38400 (230.4K)	Current baudrate of EM ATPPP port (stored in hexadecimal format)
CFG	Prefix	String	SWI	Communication port prefix name for the EM ATPPP port. This port should be only used by the RIL to initiate the atd command.

A.5 HKCU\ Software\Sierra Wireless Inc\RIL\Audio\<one of RIL_AUDIO_... in ril.h and rilext.h>

OEM	Field	Type	Default value	Description
CFG	ForceUpdate	DWORD	0	Force a module audio provisioning data update from registry values, it reset itself to 0 on update complete.
CFG	TxPcmFilterStageNum RxPcmFilterStageNum	DWORD	OEM choices	Number of filter stages specified
CFG	RxPcmFilterStage*X* TxPcmFilterStage*X*	String	OEM choices	Actual filter stage parameter. Contact Sierra Wireless AE support for audio calibration
CFG	AutoInputGain AutoOutputGain	DWORD	OEM choices	1 to enable, 0 to disable
	TxVolumeLevel VoiceVolumeLevel	DWORD	OEM choices	Volume level for the two directions
	RxGainLevel*1..7*	DWORD	OEM choices	Actual volume gain for each volume steps
	SideToneLevel	DWORD	OEM choices	Audio side tone gain

A.6 HKCU\Software\Sierra Wireless Inc\RIL\PowerControl

OEM	Field	Type	Default value	Description
CFG	Get	DWORD	1	1- Call OEMAUX to get/set EM power state 2- OEMAUX not ready. RIL just fake the desired successful response for the power on/off command
CFG	Set	DWORD	1	Should be set to the same value as “Get” registry
CFG	AuxAPIDllName	String	OEMAux.dll	The library (dll) name that the OEM chooses to implement the OEMAUX functionality.
RO	SwiAuxAPIDllName	String	SwiCEAux.dll	The wrapper library of the OEMAUX.dll that the RIL driver is interfacing with.

A.7 HKCU\Software\Sierra Wireless Inc\RIL\Emergency

OEM	Field	Type	Default value	Description
CFG	RilEmergencyNum	String	911,112….	List of phone numbers that are considered emergency. The EM will use any carrier in coverage to make the call.
CFG	ForceUpdate	DWORD	0	Force an update of the list to the EM’s NV

				during system boot up. Auto reset to 0 on a successful update.

A.8 HKCU\SOFTWARE\Sierra Wireless Inc\RIL\Supsvc

OEM	Field	Type	Default value	Description
CFG	CLIRStatus	DWORD	none	Local cache of the CLIR status
CFG	CallFwRetry	DWORD	1	Allow a retry for Call forward related operations. Required by some carrier (e.g. HK Smartone)

A.9 HKCU\SOFTWARE\Sierra Wireless Inc

OEM	Field	Type	Default value	Description
DC	ModemName	String	Cellular Line	AAM uses this to disconnect RAS before an IOTA session
CFG	CarrierSuffix	String	UNDEFINED	"PCS" - Sprint PCS "VZW" - Verizon Wireless "SW" – Generic "UNDEFINED" – helps generic carrier to correctly populate the system registry since UNDEFINED corresponds to a clean boot state system
DC	ConnectionProfileName	String	3G Connection	Used by IOTA Thin Client to make RAS connection
CFG	DynamicCarrierSupported	DWORD	1	0 or Absent - Dynamic carrier configuration of the RIL disabled 1 - Dynamic carrier configuration of the RIL enabled
CFG	DisableAutoActWiz **(absent by default)**	DWORD	N/A	0 or absent - Activation Wizard Auto-launch enabled 1 – GobiActibation, Activation Wizard, IOTA or OMADM Auto-launch disabled
CFG	KillActiveSync4IOTA **(absent by default)**	DWORD	N/A	1 or Absent - ActiveSync is killed before launching an IOTA session 0 - ActiveSync is not killed before launching an IOTA session
CFG	DisableUIExt **(absent by default)**	DWORD	N/A	Bit Mask options: 0 (or key value absent): enable all UI Extensions 1 - Disable phone settings location tab 2 - Disable phone settings data tab 4 - Disable phone settings system tab
CFG	CarrierProduct	DWORD	1	0 or absent - RIL UI Extensions do not reference carrier support information where appropriate for a product sold/supported by OEM or 3rd party 1 - RIL UI Extensions reference carrier support information where appropriate for a carrier product sold/supported by carrier

CFG	FeaturesSupported	DWORD	0187A200	This registry key is modified dynamically by the RIL driver after the modem is powered ON. The registry key value is modified based on whether the modem is in a UMTS or a CDMA mode.
CFG	PollingForRSSI **(absent by default)**	DWORD	1	Absent = 0 = RIL will report RSSI via notification. 1 = RIL will report RSSI via polling mechanism. Sierra recommends this value.
CFG	NetworkPreference **(absent by default)**	DWORD	0	0 = auto 1 = 2G 2 = 3G Absent = Sierra recommended This registry key should be used ONLY by an advanced user. If this registry key is set, then on modem power up, RIL will call a GobiAPI SetNetworkPreference() to set the network preference (auto, 2G, 3G) in the modem. For CDMA, 2G = 1X, 3G = EVDO. For UMTS, 2G = GSM, 3G = UMTS.

A.10 HKLM\SOFTWARE\Microsoft\DialParser

OEM	Field	Type	Default value	Description
RW	DLL-OEM	String	SWDialParserG.dll	DLL to be used by the OS for notifications of button press and TALK press events from the phone dialer for "special" button press processing.

A.11 HKCU\SOFTWARE\Sierra Wireless Inc\System

OEM	Field	Type	Default value	Description
DC	AutoUpdateTime	DWORD	0	0 - currently disabled 1 - currently enabled
CFG	AutoUpdateTimeUI	DWORD	0	0 - permanently disabled auto-update system time with CDMA time feature 1 - permanently enable auto-update system time with CDMA time feature with no UI setting for user to disable 2 - auto-update system time with CDMA time feature controlled by user through Phone Settings Menu option (current setting specified by AutoUpdateTime)

CFG	RoamGuard	DWORD	1	Sprint roaming guard. 1 to enable and 0 to disable.

A.12 HKCU\SOFTWARE\Sierra Wireless Inc\SS_Log

OEM	Field	Type	Default value	Description
RO	Default Log File	String	\\Windows\\SS_Log.log	Name and path of RIL log file location
CFG	Filter - Status LogLevel	DWORD	0000000e	**Mask:** 0x0008 - Version 0x0004 - Critical 0x0002 - Warning 0x0001 - Trace - Trace and Warning log levels are not supported in the ARM RETAIL release. - Mask does not affect CEPC build because all the logs are routed to DEBUGMSG.
DC	Filter - Status CompLevel	DWORD	ffffffff	RIL component log enable mask
CFG	Size - Log File Size	DWORD	80000	The maximum size of the log file in Bytes
DC	Display Mem	DWORD	1	Used to log available memory in system 0 - Do not display memory 1 - display memory info

A.13 HKEY_CURRENT_USER\Software\Sierra Wireless Inc\IOTA

OEM	Field	Type	Default value	Description
RO	IOTA Log File	String	\\Windows\\IOTALog.log	Name and path of IOTA log file location
CFG	EnableIOTALog	DWORD	00000000	**Mask:** 00000000 - Do NOT store IOTA session log 00000000 - Store IOTA session log

A.14 HKEY_LOCAL_MACHINE\Software\Sierra Wireless Inc\QDL

OEM	Field	Type	Default value	Description
CFG	FirmwareLocation	String	\\Hard Disk\\Images\\Sierra	Firmware location. This is where all firmware images will be stored on the host device.

CFG	CarrierIndex	DWORD	00000002	**Mask:** **One of the carrier index from the list below:** **; 0 - Vodafone** **; 1 - Verizon** **; 2 - ATT** **; 3 - Sprint** **; 4 - T-Mobile** **; 6 - Generic UMTS** **; 7 - Telefonica** **; 8 - Telecom Italia** **; 9 - Orange** **; 11 – Alltel** **The carrier index will be used to download the corresponding carrier firmware image on the module.**

A.15 HKEY_CURRENT_USER\Software\Sierra Wireless Inc\USB

OEM	Field	Type	Default value	Description
CFG	UseNdisConfiguration	DWORD	1	Ndis data through: 0 – Virtual serial ports (preferred for windows mobile devices using RILGOBI solution) 1 – NDIS miniport

A.16 Additional registry keys of interest

Following registry keys may be of interest to the OEMs. More information on these registry keys can be found on MSDN. We list here the registry keys which are provided by RIL either by default or are created dynamically during boot-up.

1. Sending an SMS to an email address
   [HKEY_LOCAL_MACHINE\System\Inbox\Settings]
   "AllowSMStoSMTPConcatenation"=dword:1
   "AllowSMStoSMTPAddress"=dword:1
   "SMStoSMTPShortCode"= "6245"
   These registry keys are provided by default in SW.reg file.

2. Pressing END key should not disconnect the data call.
   [HKEY_LOCAL_MACHINE\ControlPanel\Phone]
   "Flags2"=dword:10

3. RIL populates the registry key “IS637MaxAsciiCharacters” dynamically during boot-up. RIL updates this registry key with the maximum SMS length (allowed by a carrier) read from the EM.
   Microsoft SMS application UI uses the same registry key to display the maximum length of one SMS to the user.
   [HKCU\Software\Microsoft\SMS\TextShared]

“ IS637MaxAsciiCharacters"

B. PHONE UTILITIES MENU ACCESS

The following figure lists all of the hidden menu tabs that are supported in the Phone Utilities application of the RIL and how each menu is accessed for each carrier configuration of the RIL through ##codes entered from the standard MS Phone Dialer or by typing the keys on the hardware keypad.

Figure 1. Access Methods for Phone Utilities Menus

Carrier / Tabs	VZW	PCS	UMTS
Debug	##DEBUGTST	##DEBUG	##DEBUG
RTN	Not supported	##RTN, ##DEBUG	Not supported
System	##DEBUGTST	##DEBUG	Not supported
Activation	##DEBUGTST	Not Supported	Not Supported
Packet Data Profile	##DEBUGTST	##DATA	Not supported
Version Info	Not supported	Not supported	##DEBUG

C. THINGS NOT SUPPORTED BY RILGOBI

1. In CDMA mode, even if the modem is camped on EvDO Rev A network, the UI will NOT display EvDO Rev A icon. It will display default MSFT EV icon.
2. RILGOBI does not support any SIM Toolkit APIs.
3. RILGOBI only support SIM Phonebook APIs with SL9090 and MC73xx
4. RILGOBI does not support any RIL APIs for GSP Intermediate Driver.