Application Developer's Guide

Linux QMI SDK

SIERRA WIRELESS®

4110914
1.20
Apr 28, 2015

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 cellular modems are not advised without proper device certifications. These areas include environments where cellular radio can interfere such as explosive atmospheres, medical equipment, or any other equipment which may be susceptible to any form of radio interference. 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.

Limitations of Liability

This manual is provided "as is". Sierra Wireless makes no warranties of any kind, either expressed or implied, including any implied warranties of merchantability, fitness for a particular purpose, or noninfringement. The recipient of the manual shall endorse all risks arising from its use.

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.

Customer understands that Sierra Wireless is not providing cellular or GPS (including A-GPS) services. These services are provided by a third party and should be purchased directly by the Customer.

SPECIFIC DISCLAIMERS OF LIABILITY: CUSTOMER RECOGNIZES AND ACKNOWLEDGES SIERRA WIRELESS IS NOT RESPONSIBLE FOR AND SHALL NOT BE HELD LIABLE FOR ANY DEFECT OR DEFICIENCY OF ANY KIND OF CELLULAR OR GPS (INCLUDING A-GPS) SERVICES.

Patents

This product may contain technology developed by or for Sierra Wireless Inc.

This product includes technology licensed from QUALCOMM®.

This product is manufactured or sold by Sierra Wireless Inc. or its affiliates under one or more patents licensed from InterDigital Group and MMP Portfolio Licensing.

Copyright

Trademarks

Sierra Wireless®, AirPrime®, AirLink®, AirVantage®, WISMO® and the Sierra Wireless and Open AT logos are registered trademarks of Sierra Wireless, Inc. or one of its subsidiaries.

Watcher® is a registered trademark of NETGEAR, Inc., used under license.

Windows® and Windows Vista® are registered trademarks of Microsoft Corporation.

Macintosh® and Mac OS X® are registered trademarks of Apple Inc., registered in the U.S. and other countries.

QUALCOMM® is a registered trademark of QUALCOMM Incorporated. Used under license.

Other trademarks are the property of their respective owners.

Contact Information

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

Document History

Version	Date	Updates
1.0	September 09, 2011	Initial Release
1.01	October 05, 2011	Added section 5.4: Connection Manager Sample Application; Updated sections 3.1, 4.2, 4.3, 5.2, 5.3
1.02	November 04, 2011	Added Section 6.2 RAM dump tool, corrected header level for 6.1
1.03	March 15, 2012	Added section 4.2; Updated/revised sections 2 and 3.
1.04	November 07, 2012	Added SL9090 as supported device in Section 2.3; Added PDS & SWIOMA Sample App in Section 5; Update titling on Section 4.4; Update code routine at Section 3.3.1
1.05	February 05, 2013	Updated architecture diagram for multiple application support in Section 1.1; Updated connection APIs in Section 3.4; Added call handling sample application in Section 5;
1.06	June 24, 2013	Updated Section 3.3, User Application Development.
1.07	July 26, 2013	Added new section for AirVantage agent integration
1.08	October 18, 2013	Added new section for AirVantage agent auto start preprocessor section 6.2
1.09	December 04, 2013	Added section 4.5 for one command line firmware downloader
1.10	January 16, 2014	rename folder name from avagent_r8 to avagent_r8m
1.11	March 07, 2014	Added multiple module management section 3.3.1
1.12	May 07, 2014	Updated modules & PID list in Section 2.2 Supported Devices
1.13	June 12, 2014	Removed multiple modems not supported from the unsupport feature section
1.14	July 10, 2014	Updated section 6
1.15	July 22, 2014	Add remote DM log capture section 9
1.16	July 22, 2014	Add List of automatic re-register callback in section 10
1.17	October 17 2014	Add Debug Information Section 12
1.18	February 03 2015	Update /sys/modules path at section 2.3.7
1.19	April 09 2015	Add SQF Filter Editing in tools section
1.20	April 28 2015	Add EM7455 in Supported Device version

Contents

1. SLQS SYSTEM ARCHITECTURE 9

2. SLQS PREREQUISITES 10

2.1. Supported Processors 10

2.2. Supported Devices 10

2.3. Device Drivers 11

2.3.1. Acquiring the Drivers 11

2.3.2. Supported Linux Kernels 11

2.3.3. System Dependencies 11

2.3.4. Building and Installing the Drivers 11

2.3.5. Querying Driver Versions and Supported Devices 11

2.3.6. Unloading the Drivers 11

2.3.7. Enabling and Disabling the Drivers' Diagnostic Messages 12

2.3.8. Verifying Proper Driver Operation 12

3. USER APPLICATION DEVELOPMENT 13

3.1. SDK Process 13

3.1.1. Building the SDK Executable 13

3.1.2. Verifying SDK and Target Platform Interoperability 13

3.2. User Application Process 14

3.2.1. Building the Application Executable 14

3.2.2. Communicating with the Device 14

3.3. User Application Development 14

3.3.1. Multiple Module Management 14

3.3.2. Where to Start 15

3.3.3. QCWWANDisconnect API 17

3.3.4. Terminating the SDK Process 17

3.3.5. Device Resets 18

3.4. UMTS, LTE, and CDMA Data Sessions 18

3.4.1. Profile Configuration 18

3.4.2. Session Initiation and Termination 19

4. SLQS IMAGE MANAGEMENT 20

4.1. Firmware Upgrade Process 20

4.2. QDL Image Download 21

4.3. AR75xx, MC73xx, MC7700/10/50 and WP71xx Modem Image Management 22

4.3.1. MC77xx Image Management Sample Application 22

4.4. MC83xx, MC9090 and SL9090 Image Management 23

4.4.1. Gobi Image Management Sample Application 23

4.5. One Command Line Firmware Downloader Sample Application 24

5. OTHER SAMPLE APPLICATIONS 26

5.1. Call Handling Sample Application 26

5.2. Connection Manager Sample Application 26
5.3. SMS Sample Application 27
5.4. SLQS Tutorial Sample Application 27
5.4.1. Using the SLQS Tutorial 28
5.4.1.1. Execution with Root Privileges 28
5.4.1.2. Execution without Root Privileges 29
5.5. Connection Manager Sample Application 30
5.6. Position Determination Service Sample Application 30
5.7. SWIOMA Sample Application 31
6. AIRVANTAGE AGENT INTEGRATION 32
6.1. Auto Start Preprocessor 32
6.2. Agent Configuration File 32
6.3. Agent Constrains 32
6.4. Agent Source Tree 32
6.5. Start/Stop the AirVantage Agent 32
6.6. AirVantage M2M Cloud 33
7. TOOLS 34
7.1. DM Logging Tool 34
7.2. RAM Dump Tool 34
8. SLQS DOCUMENTATION 37
9. REMOTE DM LOG 38
9.1. Introduction 38
9.1.1. Purpose 38
9.1.2. Overview 38
9.2. Capturing DM log on Windows from the Device connected to Linux host 38
9.2.1. *Using Remserial utility/tool on Linux host* 38
9.3. Connect to QXDM 42
9.3.1. In QXDM menu, select Options then Communications 42
9.3.2. Log Viewer (F1) 43
9.3.3. Select event to log from Options then Log View Config (F5) 45
10. CALLBACK THAT AUTO RE-REGISTER 46
11. REFERENCE DOCUMENTS 47
12. DEBUG INFORMATION 47

List of Figures

Figure 1. SLQS System Architecture 9
Figure 2. SLQS Image Management Sequence Diagram 20
Figure 3. QDL Service Sequence Diagram 21

List of Tables

Table 1. Supported Application-Mode VID/PIDs 10
Table 2. Supported Boot-Mode VID/PIDs 10

1. SLQS System Architecture

The SLQS system architecture is described as follows.

- The Application process communicates with the device by executing SLQS APIs.
- The API calls are translated in QMI request SDUs that are sent to the SDK process over a local IPC datagram socket.
- The SDK writes the QMI PDUs to a device file named /dev/qcqmix, where x is an integer, associated with the QMI interface.
- The QMI PDUs are sent to the device over the USB control channel via the GobiNet.ko driver module
- Notifications are received over the interrupt channel, which prompts the driver to read the responses coming over the USB control channel.
- The SDK reads the QMI response from /dev/qcqmix and sends a response to the application process over a local IPC datagram socket.

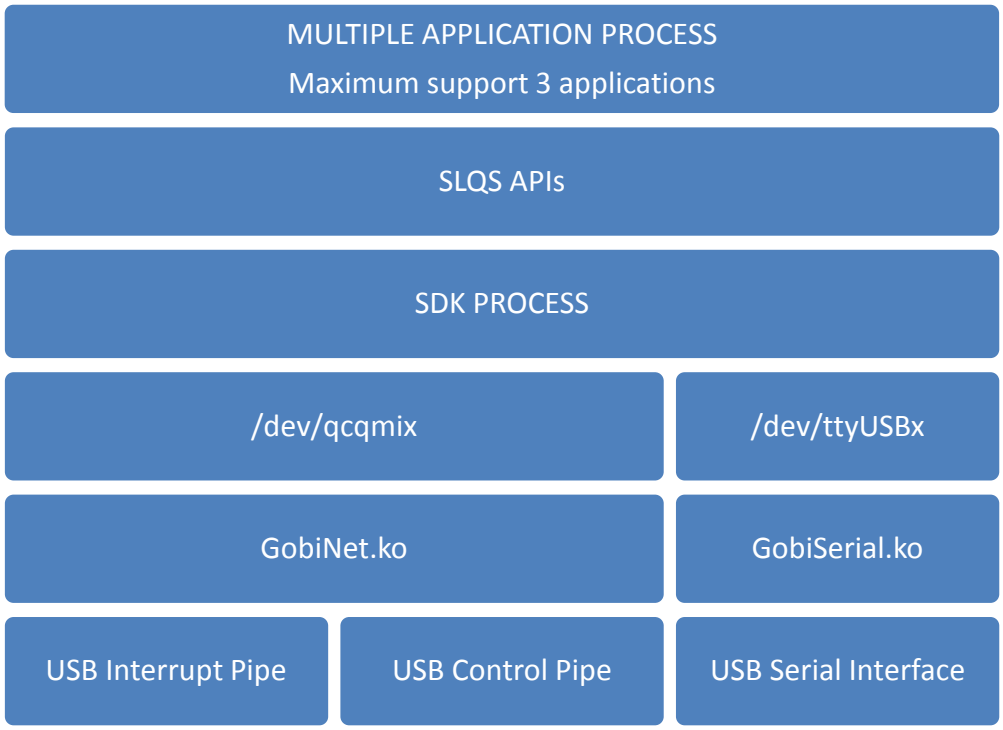


Figure 1. SLQS System Architecture

2. SLQS Prerequisites

2.1. Supported Processors

The following processors are supported:

- x86
- ARM
- PPC
- MIPS

2.2. Supported Devices

The following devices are supported:

- MC77xx
- MC83x5
- SL/MC9090
- WP71xx
- MC73xx/MC78xx
- EM73xx
- AR7554
- EM7455

Note: MC77xx devices must operate in "QMI Mode" and not in "Direct-IP" mode.

The tables below list the hexadecimal values of the Vendor ID (VID) and Product ID (PID) pairs supported by the SLQS.

Table 1. Supported Application-Mode VID/PIDs

VID	1199	1199	1199	1199	1199	1199	3F0	1199	1199
PID	68A2	68C0	9011	9013	9015	9019	371D	9040	9041

Table 2. Supported Boot-Mode VID/PIDs

VID	1199	1199	1199	1199	1199	1199	3F0
PID	68A2	68C0	9010	9012	9014	9018	361D

To check your device's VID/PID, issue the **`lsusb`** command. The output will present a list of USB devices with a column showing each device's manufacturer. The device VID/PID can be read from the row containing the correct device manufacturer. Additionally, on MC77xx devices, you can use the **`AT!UDINFO?`** command to check VID/PID information. If your VID/PID does not match the any of the entries in the tables above, contact your FAE for support.

2.3. Device Drivers

2.3.1. Acquiring the Drivers

Get in touch with your FAE for acquiring drivers for your device if you are operating in QMI mode – the mode of operation required for using the SLQS.

2.3.2. Supported Linux Kernels

Sierra Wireless supports open source kernel version 2.6.32 or newer. It is the customer's responsibility to modify the SDK and drivers for kernels outside the scope of what is supported.

2.3.3. System Dependencies

Make sure you have a network connection and issue the following commands:

```
sudo apt-get install build-essential make gcc
sudo apt-get install linux-headers-`uname -r`
```

2.3.4. Building and Installing the Drivers

```
cd GobiSerial; make; sudo make install
cd GobiNet; make; sudo make install
sudo modprobe GobiSerial [debug=Y]
sudo modprobe GobiNet [debug=Y]
```

2.3.5. Querying Driver Versions and Supported Devices

```
modinfo GobiSerial
modinfo GobiNet
```

2.3.6. Unloading the Drivers

```
sudo rmmod GobiSerial
sudo rmmod GobiNet
```

2.3.7. Enabling and Disabling the Drivers' Diagnostic Messages

Note: Enabling and disabling the driver's diagnostic messages requires root privileges.

Enable diagnostic messages:

```
echo 1 > /sys/module/GobiSerial/parameters/debug
echo 1 > /sys/module/GobiNet/parameters/debug
```

Disable diagnostic messages:

```
echo 0 > /sys/module/GobiSerial/parameters/debug
echo 0 > /sys/module/GobiNet/parameters/debug
```

2.3.8. Verifying Proper Driver Operation

1. Open terminal and type **tailf /var/log/syslog**.
2. Plug in the Sierra Wireless device.
3. Check **/dev/** for existence of the following devices (check syslog in case the device nodes are static i.e. built into the kernel image and not dynamically mounted). Please note that the second QMI interface is available only when the device is in multi-pdn mode (**/dev/qcqmiy**).
   - **/dev/ttyUSB0**
   - **/dev/ttyUSB1**
   - **/dev/ttyUSB2**
   - **/dev/qcqmix** where x is an integer starting at 0
   - **/dev/qcqmiy** where y is an integer starting at 0

3. User Application Development

3.1. SDK Process

3.1.1. Building the SDK Executable

```
navigate to pkgs:  cd pkgs
clean then build:  make –f pkgs.mak complete
clean:             make –f pkgs.mak clean
build:             make –f pkgs.mak
```

3.1.2. Verifying SDK and Target Platform Interoperability

The SDK periodically checks to see if a supported device is connected to the target platform. If you do not see the following message[1] in your logs, then the device has not been detected and the SDK will not be able to communicate with the device. In this case, it is most likely that you are either using an unsupported device or that your drivers need to be updated to support the device.

```
usb 2-1.5: new high speed USB device using ehci_hcd and address 10
usb 2-1.5: config 1 has an invalid interface number: 8 but max is 3
usb 2-1.5: config 1 has no interface number 1
usb 2-1.5: configuration #1 chosen from 1 choice GobiSerial 2-1.5:1.0:
GobiSerial converter detected
usb 2-1.5: GobiSerial converter now attached to ttyUSB0 GobiSerial 2-
1.5:1.2: GobiSerial converter detected
usb 2-1.5: GobiSerial converter now attached to ttyUSB1 GobiSerial 2-
1.5:1.3: GobiSerial converter detected
usb2-1.5: GobiSerial converter now attached to ttyUSB2
usb0: register 'GobiNet' at usb-0000:00:1d.0-1.5, QmiNet Ethernet
Device, 3e:a6:1f:b3:66:62
SWI SDK Process: USDT:Device State Change 0 -> 1
creating qcqmi0
USDT:Device State Change 1 -> 1
```

If you see the message above but do not see the following message in your logs, then the device's interfaces have not been successfully mapped to their respective **/dev/ttyUSBx** and/or **/dev/qcqmix** device special files and the SDK will not be able to communicate with the device.

```
USDT:Device State Change 1 -> 2
USDT:Device ready: VID 1199, PID 68a2, 4 interfaces
QM:qm_ds_handle_app_dev_ready: devstate 1
QM:SDK<-Mdm: ch/QMImsgid/QMImsglen/IPCmsglen: 1/0000/0/25
QM:DS Device Event Notification received 1
```

[1] SDK messages will be displayed in both **/var/log/user.log** and **/var/log/syslog**.

In this case, it is possible that:

1. your drivers don't support the inserted device;
2. you have not added a device node for /dev/qcqmix (usually 0 or 1) with the proper major and minor numbers;
3. the interface configuration of your device is not supported by the SDK; or
4. the SDK's device scanning routine requires custom modifications specific to your platform's **sysfs/sys/bus/usb/devices** entry for the device in question. The major and minor numbers of the device can be determined by issuing **`ls -l /dev/qcqmix`** on the command line. The fourth and fifth columns contain the major and minor numbers, respectively (see **`man ls`** for details).

3.2. User Application Process

3.2.1. Building the Application Executable

Refer to any one of the sample applications' make files as a starting point for writing a script for building your application. Remember to add the "**`strip`**" command to your script in order to remove all symbol information from your libraries and application image if your system is memory constrained.

3.2.2. Communicating with the Device

The application must adhere to the SDK's stop and wait (synchronous execution) protocol; there can be only one outstanding transaction between the application and SDK, at any time. All API function calls are blocking and execute within the context of the application process. When the application executes API function, the corresponding request is constructed and sent to the SDK process over a local IPC. The request (response) is sent to (received from) the device from within the execution context of the SDK process. The response from the device is validated and sent back to the application process over a local IPC socket. After which, the message contents are unpacked and used for populating the user supplied arguments.

Notifications, on the other hand, are asynchronous and therefore, may arrive at any time. The application receives notifications within the execution context of a dedicated notification thread that is created and used by the SDK within the application's process. Thus, it is important that minimal processing be done inside the registered callback functions.

3.3. User Application Development

3.3.1. Multiple Module Management

Since SLQS03.03.00, Sierra Wireless has introduced multiple module management, also known as multi modem support. The SLQSStart is used to select which modem to control. Passing 0, 1 and 2 will select the first, second and third modem detected. It supports a maximum of 12 modems.

The qatest is already updated to support multiple modems as an example. There is a "**`-d`**" command line switch to specify modem index.

The following example controls the first and second modems.

```
sudo ./pkgs/qa/qatesthostx86_64 -r -d0
sudo ./pkgs/qa/qatesthostx86_64 -r -d1
```

3.3.2. Where to Start

The Connection Manager Sample Application is a good place to start. The source code is located at **SampleApps/ Connection_Manager/src/connectionmgr.c**.

The following outlines the recommended method for integrating SLQS initialization code into your application. Note that all variables below are assumed to have been defined.

```
/* Set the SDK executable path for your target platform */
    if( SUCCESS != (resultCode = SetSDKImagePath(sdkbinpath)) )
    {
        rcprint( __func__, resultCode );
        return resultCode;
    }

     /* Launch the SDK process and create IPC sockets over which the APP and SDK
      * will exchange messages.
      */
    if( SUCCESS != (resultCode = SLQSStart(modem_index)) )
    {
        /* first attempt failed, kill SDK process */
        if( SUCCESS != SLQSKillSDKProcess() )
        {
            return resultCode;
        }
        else
        {
            /* start new SDK process */
            if( SUCCESS != (resultCode = SLQSStart(modem_index)) )
            {
                return resultCode;
            }
        }
    }

    /* Enumerate the device */
    while (QCWWAN2kEnumerateDevices(&devicesSize, (BYTE *)pdev) != 0)
    {
        printf ("\nUnable to find device..\n");
        sleep(1);
    }
```

```
    #ifdef DBG
    fprintf( stderr,  "#devices: %d\ndeviceNode: %s\ndeviceKey: %s\n",
            devicesSize,
            pdev->deviceNode,
            pdev->deviceKey );
    #endif

    /* Connect to the SDK */
    resultCode = QCWWANConnect( pdev->deviceNode,
                                pdev->deviceKey );

    /* Subscribe to all the required callbacks */
    SubscribeCallbacks();

/* Graceful SLQS teardown */
void QuitApplication()
{
    free(sdkbinpath);
    fprintf( stderr, "Exiting Application!!!\n" );

    /* Unsubscribe all the callback which was called previously */
    UnSubscribeCallbacks();
    closeLogFile();

     /* If the application is connected to the SDK, then disconnect to (1)
      * terminate threads and free resources that have been created and allocated,
      * respectively, for communicating with the device, and (2) allow other
      * applications to communicate with the device via the SDK.
      */
  QCWWANDisconnect();
    exit( EXIT_SUCCESS );
}

/* macro used in code segments above */
#define rcprint(s, u) syslog(LOG_USER, "%s: rc = 0x%lX, %s", s, u, slqserrstr(u))

/* You can add error code to error string mapping to the table below in order to
 * aid your application debugging.
 */
typedef struct{
    enum eQCWWANError e;
    const char *es;
}slqserr_s;
```

```
static slqserr_s errstr[] =
{
    { eQCWWAN_ERR_INTERNAL,                    "eQCWWAN_ERR_INTERNAL" },
    { eQCWWAN_ERR_MEMORY,                      "eQCWWAN_ERR_MEMORY" },
    { eQCWWAN_ERR_INVALID_ARG,                 "eQCWWAN_ERR_INVALID_ARG" },
    { eQCWWAN_ERR_BUFFER_SZ,                   "eQCWWAN_ERR_BUFFER_SZ" },
    { eQCWWAN_ERR_NO_DEVICE,                   "eQCWWAN_ERR_NO_DEVICE" },
    { eQCWWAN_ERR_SWIDCS_IOCTL_ERR,            "eQCWWAN_ERR_SWIDCS_IOCTL_ERR" },
    { eQCWWAN_ERR_QMI_MISSING_ARG,             "eQCWWAN_ERR_QMI_MISSING_ARG" },
    { eQCWWAN_ERR_SWICM_SOCKET_IN_USE,         "eQCWWAN_ERR_SWICM_SOCKET_IN_USE" },
    { eQCWWAN_ERR_SWIDCS_DEVNODE_NOT_FOUND,    "eQCWWAN_ERR_SWIDCS_DEVNODE_NOT_FOUND" },
    { eQCWWAN_ERR_SWIDCS_IOCTL_ERR,            "eQCWWAN_ERR_SWIDCS_IOCTL_ERR" },
    { eQCWWAN_ERR_SWIDCS_APP_DISCONNECTED,     "eQCWWAN_ERR_SWIDCS_APP_DISCONNECTED" },
    { eQCWWAN_ERR_SWICM_QMI_SVC_NOT_SUPPORTED, "eQCWWAN_ERR_SWICM_QMI_SVC_NOT_SUPPORTED" },
    { 0, "" }
};

static const char *slqserrstr(ULONG er)
{
    int count = 0;
    while( errstr[count].e ){
        if( errstr[count].e == er )
        {
            return errstr[count].es;
        }
        count++;
    }
    return "";
}
```

3.3.3. QCWWANDisconnect API

When your application no longer needs to communicate with the device it should execute the **QCWWANDisconnect API** in order to:

1. free the resources allocated by the SDK for communicating with the device;
2. deregister from all but the device state change callback; and
3. allow other applications to use the services of the SDK. As long as the device is connected to the target and the SDK process is alive, an application can always reconnect at a later time.

3.3.4. Terminating the SDK Process

To kill the SDK process, execute the **SLQSKillSDKProcess API**. Note that this API requires that the SDK image be named **slqssdk**, as is the case for the images located in the build/bin sub-directories of the SDK release.

3.3.5. Device Resets

Assuming the application has registered for the device state change callback, it will be notified whenever a device is disconnected or detected. Following a device reset, once the device has been detected by the SDK, all of the callback functions that the application had registered for will be re-registered by the SDK on the application's behalf. Thus, the application need not take any action on a device reset aside from managing itself.

3.4. UMTS, LTE, and CDMA Data Sessions

This section describes the APIs for configuring profiles for use in a data session call; as well as starting and stopping data session calls. For details of the API parameters, refer to the doxygen documentation of the APIs.

3.4.1. Profile Configuration

Profiles must be set before a data call can be made. Some carriers fix the profiles that can be used on their network. Without the use of SDK APIs, profiles can be created or modified using AT commands. The SDK provides the following APIs for profile configuration:

- GetDefaultProfile
- SetDefaultProfile
- GetDefaultProfileLTE
- SetDefaultProfileLTE

The APIs above write and get the default profile to and from the device, respectively. The default profile will be the one used to establish a data session. The LTE version supports IPV6 in addition to IPV4.

The following APIs perform the same functionality as the APIs above, but allow a profile ID to be specified. Valid profile ID values are 1 to 16.

- SLQSGetProfile
- SLQSSetProfile

The following API deletes a configured profile stored on the device. The deletion of a profile does not affect profile index assignments.

- SLQSDeleteProfile

The following API is used to create a new profile with the specified parameters. Note that some firmware versions do not support the optional Profile ID parameter. In this case an error will be returned and the caller can subsequently create a profile by specifying a NULL pointer for the Profile ID parameter. The Profile ID pertaining to the newly created profile is returned in the response structure parameter.

- SLQSCreateProfile

This API is used to create a new profile with the specified parameters.

- SLQSModifyProfile

3.4.2. Session Initiation and Termination

The API, **SLQSStartStopDataSession**, will use the default profile set up as described above to make a data connection. Some networks may require authentication fields.

To start a data session after a device has been enumerated, the following API may be used. Note that technology should be changed for the appropriate network – UMTS or CDMA; and that the optional parameters below are left as NULL for simplicity. Some of the optional parameters are supplied by the user as preferred information. The network may not be able to assign the preferred values and assign other values instead. In that case, the **SLQSGetRuntimeSettings** API may be used to retrieve some of this information once a data session has been established. This API supports IPV4, IPV6, and IPV4V6 data sessions specified by ipfamily member of struct ssdatasession_params.

```
ULONG technology = 1; //3GPP
ULONG profile_idx = 1;
struct ssdatasession_params session;
session.action = 1; //start data session
session.pTechnology = &technology;
session.pProfileId3GPP = &profile_idx;
session.pProfileId3GPP2 = NULL;
session.ipfamily = 4; //IPv4
rc = SLQSStartStopDataSession( &session );
```

To terminate any currently active data session given the session pointer, the following API is used.

```
session.action = 0; //stop data session
rc = SLQSStartStopDataSession( &session );
```

4. SLQS Image Management

The Gobi Image Management and MC77xx Image Management sample applications can be used to:

1. Query information about the firmware stored on the device
2. Query information about firmware images stored on the host
3. Download firmware to the device

4.1. Firmware Upgrade Process

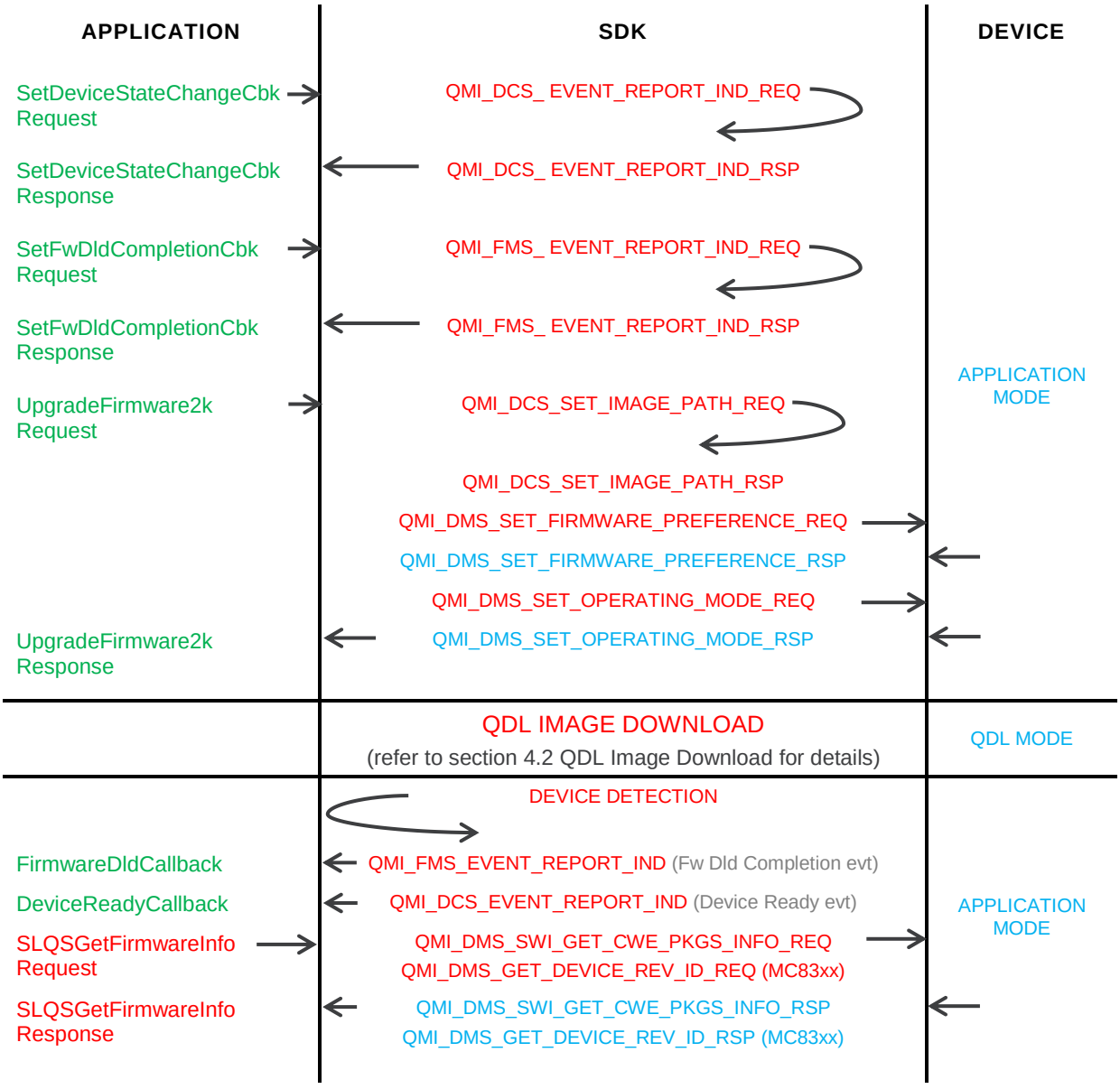


Figure 2. SLQS Image Management Sequence Diagram

Based on the figure above:

1. The Application may choose to register for a firmware download completion callback in order to be notified when the image download process has completed. Additionally, the application should register for the device state change callback in order to be notified of when the device has entered application mode subsequent to the image download, and is ready to communicate with the host.
2. To upgrade the firmware on the device, the application must issue the **UpgradeFirmware2k** API.
3. The Application should not issue any further API requests until the firmware download has completed and the device is ready.
4. Reception of the firmware download completion callback does not guarantee that the download process was successful. Once the device is ready, the application should issue the **GetFirmwareRevisions** API (for MC83xx devices) or the **SLQSGetFirmwareInfo** API (for MC77xx devices) to determine if the upgrade was successful.

4.2. QDL Image Download

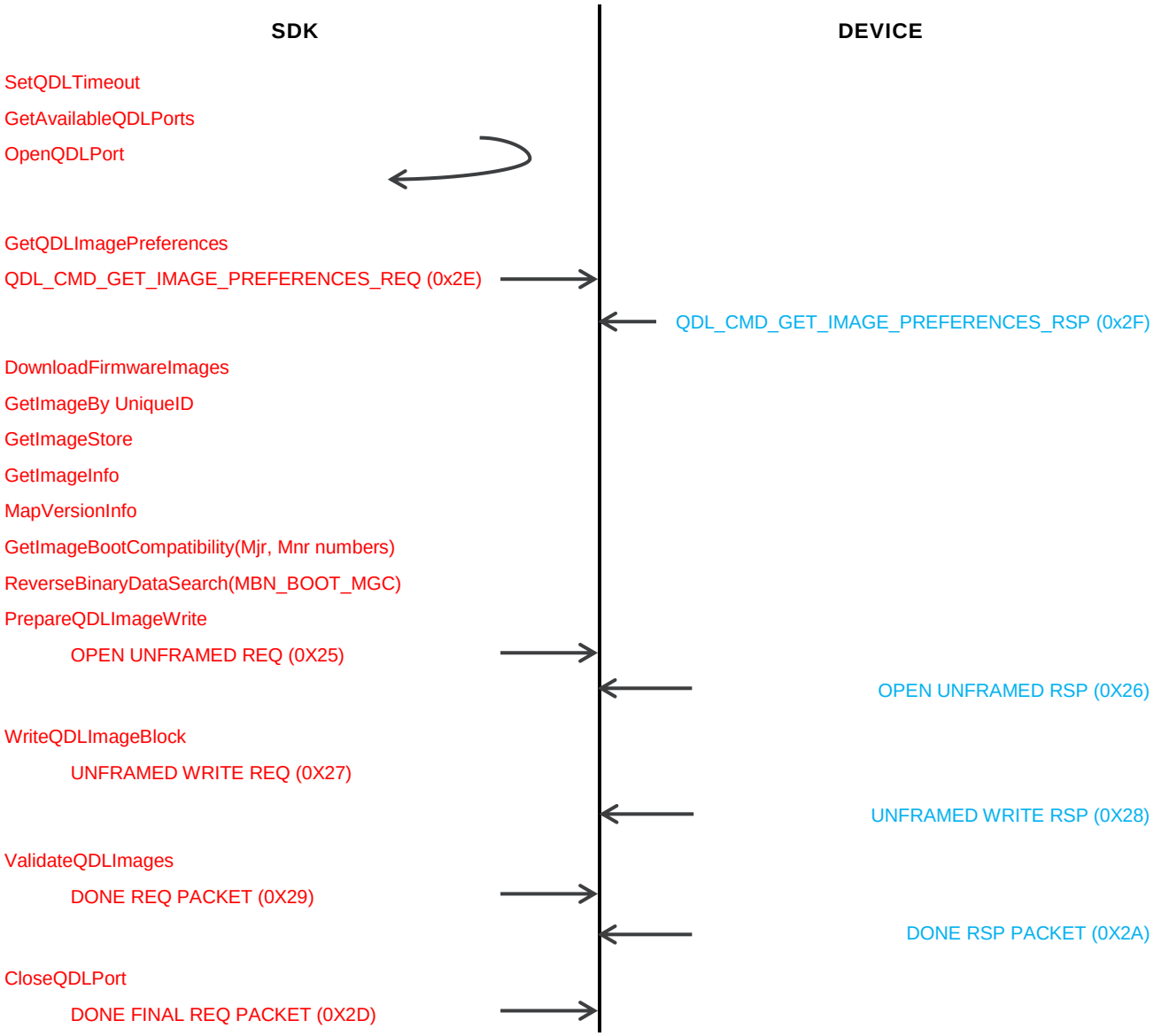


Figure 3. QDL Service Sequence Diagram

4.3. AR75xx, MC73xx, MC7700/10/50 and WP71xx Modem Image Management

4.3.1. MC77xx Image Management Sample Application

Location: SampleApps/MC77xx_Image_Management/

Purpose: Query image information for a MC7xx image located on the host

Query image information for the image running on a MC77xx device

Download firmware to a MC77xx device

Build:

i686:	make
ARM:	make CPU=arm9
Power PC:	make CPU=ppc
MIPS BE:	make CPU=mips
MIPS LE:	make CPU=mipsel

Execute:

i686:	sudo ./bin/mc7ximgmgmthosti686 ../../build/bin/hosti686/slqssdk
ARM:	sudo ./bin/mc7ximgmgmtarm9 ../../build/bin/arm/slqssdk
PPC:	sudo ./bin/mc7ximgmgmtppc ../../build/bin/ppc/slqssdk
MIPS BE:	sudo ./bin/mc7ximgmgmtmips ../../build/bin/mips/slqssdk
MIPS LE:	sudo ./bin/mc7ximgmgmtmipsel ../../build/bin/mipsel/slqssdk

Reference: SampleApps/MC77xx_Image_Management/readme.txt

The only supported file is a *_SPKG.cwe file.

The program must be executed from the SampleApps/MC77xx_Image_Management directory with the instructed execute command above.

There must only be one *.cwe or *.spk file in the path specified for any option which requires the user to specify a path.

If errors are encountered when specifying a relative path, specify the fully qualified path instead. For more details, refer to the readme.txt file

4.4. MC83xx, MC9090 and SL9090 Image Management

4.4.1. Gobi Image Management Sample Application

Location: SLQSab.cd.ef /SampleApps/Gobi_Image_Management/

Purpose: Query carrier image information for Gobi images located on the host

Query carrier image information for the images stored on a device

Download firmware to a device

Build:

i686:	make
ARM:	make CPU=arm9
Power PC:	make CPU=ppc
MIPS BE:	make CPU=mips
MIPS LE:	make CPU=mipsel

Execute:

i686:	sudo ./bin/gobiimgmgmthosti686 ../../build/bin/hosti686/slqssdk
ARM:	sudo ./bin/gobiimgmgmtarm9 ../../build/bin/arm/slqssdk
PPC:	sudo ./bin/gobiimgmgmtppc ../../build/bin/ppc/slqssdk
MIPS BE:	sudo ./bin/gobiimgmgmtmips ../../build/bin/mips/slqssdk
MIPS LE:	sudo ./bin/gobiimgmgmtmipsel ../../build/bin/mipsel/slqssdk

Reference: SLQSab.cd.ef /SampleApps/Gobi_Image_Management/readme.txt

The only supported file types are *.mbn files.

The program must be executed from the SampleApps/Gobi_Image_Management directory with the instructed execute command above.

If errors are encountered when specifying a relative path, specify the fully qualified path instead. For more details, refer to the readme.txt file

4.5. One Command Line Firmware Downloader Sample Application

Location: SLQSab.cd.ef /SampleApps/Firmware_Download/

Purpose: Perform a firmware download for the supported module by one command line.

Build:

i686:	make
ARM:	make CPU=arm9
Power PC:	make CPU=ppc
MIPS BE:	make CPU=mips
MIPS LE:	make CPU=mipsel

Execute:

i686:	sudo ./bin/fwdldhosti686 -s ../../build/bin/hosti686/slqssdk -d [9x00/9x15/g3k] -p [pathname]
ARM:	sudo ./bin/fwdldhostarm -s ../../build/bin/arm/slqssdk -d [9x00/9x15/g3k] -p [pathname]
PPC:	sudo ./bin/fwdldppc -s ../../build/bin/ppc/slqssdk -d [9x00/9x15/g3k] -p [pathname]
MIPS BE:	sudo ./bin/fwdldmips -s ../../build/bin/mips/slqssdk -d [9x00/9x15/g3k] -p [pathname]
MIPS LE:	sudo ./bin/fwdldmipsel -s ../../build/bin/mipsel/slqssdk -d [9x00/9x15/g3k] -p [pathname]

Please note that the command highlighted in red above depends on the module you are trying to perform firmware download on. For example, if it is an MDM9x15 module such as AR7554, MC7304, MC7355 etc. you have to specify the device with –d 9x15, then the path (folder of firmware images) such as –p /tmp/firmware/AR7554.

Inside the firmware folder **/tmp/firmware/AR7554**, there should be two files: one is a .nvu file of the particular firmware version. The other one can be a firmware file either with a .cwe or .spk file extension.

Taking the AR7554 as an example, the whole procedure should be as enumerated below:

1. Prepare the firmware files (.nvu + .cwe or .spk) for the update, for example, by creating a folder **/tmp/firmware/AR7554**.

   tester@Ubuntu12.04:/tmp/firmware/AR7554$ ls

   1101831_9902428_SWI9X15A_06.00.01.00_00_GENEU_006.000_000-field.spk

   NVUP-9999999_9902428_GENEU-4G_006.000_000.nvu

   cd SampleApps/Firmware_Download

2. Type command “sudo ./bin/fwdldhosti686 -s ../../build/bin/hosti686/slqssdk -d 9x15 -p /tmp/firmware/AR7554”
3. Once the firmware download starts, the console log should look like the following:

   tester@Ubuntu12.04:~/projects/Linux_QMI_SDK/tags/SLQS03.02.03/SampleApps/Firmware_Download$ sudo ./bin/fwdldhosti686 -s ../../build/bin/hosti686/slqssdk -d 9x15 -p /tmp/firmware/AR7554

   INFO: Running with device in application mode

   INFO: Path: /tmp/AR7554

```
INFO: Device Type( 0,1,2->9x00,9x15,G3K ): 1

Downloading Firmware.....................................
Firmware Download Completed


INFO: Firmware Download Succeeded
INFO: Device successfully recovered to Application mode
Exiting Application!!!
```

For more details on the usage of this Firmware_Download sample application, please read the readme.txt file under the same directory **SampleApps/Firmware_Download**.

Also, note that:

- The only supported file types are *.nvu, *.cwe and *.spk files.
- The program must be executed from the **SampleApps/Firmware_Download/** directory

5. Other Sample Applications

Information for Call Handling, Connection Manager, SMS and Developer Tutorial sample applications are provided in the following sub-sections.

5.1. Call Handling Sample Application

Location: SampleApps/CallHandling_Application/

Purpose: Voice call testing includes dialing, answering and ending a call.

Build:

i386:	make
ARM:	make CPU=arm9
Power PC:	make CPU=ppc
MIPS BE:	make CPU=mips
MIPS LE:	make CPU=mipsel

Execute:

i386:	sudo ./callhandlinghosti386
ARM:	sudo ./callhandlinghostarm9
PPC:	sudo ./callhandlinghostppc
MIPS BE:	sudo ./callhandlinghostmips
MIPS LE:	sudo ./callhandlingmipsel

Reference: SampleApps/CallHandling_Application/readme.txt

5.2. Connection Manager Sample Application

Location: SampleApps/Connection_Manager/

Purpose: Starting and stopping data session for LTE, UMTS & CDMA

Build:

i386:	make
ARM:	make CPU=arm9
Power PC:	make CPU=ppc
MIPS BE:	make CPU=mips
MIPS LE:	make CPU=mipsel

Execute:

i386:	sudo ./connectionmanagerhosti386
ARM:	sudo ./ connectionmanagerhostarm9

PPC: sudo ./ connectionmanagerhostppc
MIPS BE: sudo ./ connectionmanagerhostmips
MIPS LE: sudo ./ connectionmanagerhostmipsel

Reference: SampleApps/Connection_Manager/readme.txt

5.3. SMS Sample Application

Location: SampleApps/Gobi_Image_Management/
Purpose: Send, read, and delete SMS messages

Build: i386: make
ARM: make CPU=arm9
Power PC: make CPU=ppc
MIPS BE: make CPU=mips
MIPS LE: make CPU=mipsel

Execute:
i386: sudo ./SMSSampleAppi386
ARM: sudo ./SMSSampleApparm9
PPC: sudo ./SMSSampleAppppc
MIPS BE: sudo ./SMSSampleAppmips
MIPS LE: sudo ./SMSSampleAppmipsel

Reference: SampleApps/SMSSampleApp/readme.txt

5.4. SLQS Tutorial Sample Application

Location: SampleApps/SLQS_Tutorial/
Purpose: Familiarize Application Developers with the SDK and provide a starting point for writing an application.

Build: i386: make
ARM: make CPU=arm9
Power PC: make CPU=ppc
MIPS BE: make CPU=mips
MIPS LE: make CPU=mipsel

Execute: i386: sudo ./slqstutoriali386
ARM: sudo ./slqstutorialarm9

PPC: sudo ./slqstutorialppc

MIPS BE: sudo ./slqstutorialmips

MIPS LE: sudo ./slqstutorialmipsel

5.4.1. Using the SLQS Tutorial

Open two terminals, one for running the application, the other for viewing the message log.

In the message log terminal execute **`tailf /var/log/syslog | grep slqstuotrial`**.

In the application terminal execute **`sudo ./slqstutorial`**.

Two example sessions are shown below with interleaved explanations. Messages in green were echoed to **/var/log/syslog** from a third terminal to explain what is being done.

5.4.1.1. Execution with Root Privileges

```
slqstutorial: Run the Application ( sudo ./slqstutorial )
slqstutorial: cigetnumappclients: count: 1
slqstutorial: wSLQSStart: APP<->SDK IPC init successful
slqstutorial: wSLQSStart: APP registered for Device State Change notification
The application has set the SDK image path, registered for the device state change
callback, and started the SDK i.e. called SLQSStart which creates the SDK process
and local IPC sockets.
slqstutorial: Physically Remove the Device
slqstutorial: Device State Change Callback Invoked: rc = 0x0,
slqstutorial: appstatechange: device disconnected, APP disconnected from SDK
slqstutorial: appstatechange: device ready, APP disconnected from SDK
slqstutorial: appstatechange: device ready, APP connected to SDK
The two messages above illustrate that the application will continue to receive
device state change notifications even after calling the QCWWANDisconnect API.
slqstutorial: Attempt to Enumerate the device while it is absent ( Option 1 )
slqstutorial: wQCWWANEnumerateDevices: rc = 0x6, eQCWWAN_ERR_NO_DEVICE
slqstutorial: #devices: 1 deviceNode:  deviceKey:
slqstutorial: wQCWWANConnect: rc = 0x6, eQCWWAN_ERR_NO_DEVICE
Device enumeration has failed as the SDK did not detect a device
slqstutorial: Physically plug in the device
slqstutorial: Device State Change Callback Invoked: rc = 0x1,
slqstutorial: appstatechange: device ready, APP disconnected from SDK
The application is notified of the device state change
slqstutorial: Attempt to Enumerate the device ( Option 1 )
slqstutorial: Enumerate, Connect, Connect/Disconnect device
slqstutorial: wQCWWANEnumerateDevices: rc = 0x0,
slqstutorial: #devices: 1 deviceNode: /dev/qcqmi0 deviceKey: 00000000000000
slqstutorial: appstatechange: device ready, APP disconnected from SDK
Device enumeration is successful but note that the application is still not bound
to the SDK (APP disconnected from SDK.
```

slqstutorial: Attempt to Connect to the enumerated device (Option 2)

slqstutorial: wQCWWANConnect: rc – 0x0

slqstutorial: appstatechange: device ready, APP connected to SDK

The application is now bound to the SDK (APP connected to SDK)and may therefore issue any API function hereon.

slqstutorial: Physically remove the device while the application is bound to the SDK

slqstutorial: Device State Change Callback Invoked: rc = 0x0,

slqstutorial: appstatechange: device disconnected, APP connected to SDK

The application is notified of the device state change

slqstutorial: Plug in the device while the application is still bound to the SDK

slqstutorial: Device State Change Callback Invoked: rc = 0x1,

slqstutorial: appstatechange: device ready, APP connected to SDK

The application is notified of the device state change

slqstutorial: Execute some APIs to confirm that the application is still bound to the SDK

slqstutorial: wGetSessionState: rc = 0x0, **(Option 5)**

slqstutorial: wStartDataSession: rc = 0x0, **(Option 6)**

slqstutorial: wStopDataSession: rc = 0x0, **(Option 7)**

Successful execution of APIs as indicated by a **r**eturn **c**ode of 0x0

slqstutorial: Kill the SDK Process (option 10)

slqstutorial: wSLQSKillSDKProcess: rx = 0x0,

SDK process has been terminated (issue ps –eT | grep slqs to confirm the process is no longer running)

slqstutorial: Restart the SDK process (option 0)

slqstutorial: Device State Change Callback Invoked: rc = 0x1,

slqstutorial: appstatechange: device ready, APP diconnected from SDK

The application has set the SDK image path, registered for the device state change callback, and started the SDK i.e. called SLQSStart which creates the SDK process and local IPC sockets.

slqstutorial: Exit the application (option 11)

slqstutorial: cleanup: Good bye! (0x0)

5.4.1.2. Execution without Root Privileges

Note that there must not be an SDK daemon running with root priveleges or you will not see the same behaviour as described for below. Issue **sudo killall slqssdk** to make sure this is the case.

slqstutorial: Run the application w/o root priveleges

slqstutorial: cigetnumappclients: count: 1

slqstutorial: wSLQSStart: APP<->SDK IPC init successful

slqstutorial: wSLQSStart: APP registered for Device State Change notification

slqstutorial: wSLQSStart: APP<->SDK IPC init successful

slqstutorial: wSLQSStart: APP registered for Device State Change notification

The application has set the SDK image path, registered for the device state change callback, and started the SDK i.e. called SLQSStart which creates the SDK process and local IPC sockets.

```
slqstutorial: Attempt to Enumerate the device ( Option 1 )

slqstutorial: wQCWWANEnumerateDevices: rc = 0xE901, eQCWWAN_ERR_SWIDCS_IOCTL_ERR

Notice that an error is returned because anyone trying to access the /dev/qcqmix
device special file must have root priveleges.

slqstutorial: #devices: 1 deviceNode:  deviceKey:

Since the IOCTL issued by the SDK to the driver fails, the device key is not
returned and the returned values are blank.

slqstutorial: Attempt to Connect to the non-enumerated device  ( Option 2 )

slqstutorial: wQCWWANConnect: rc = 0x6, eQCWWAN_ERR_NO_DEVICE

No device has been enumerated as indicated by the error above

slqstutorial: Attempt to execute other APIs

slqstutorial: wQCWWANGetConnectedDevice: rc = 0x6, eQCWWAN_ERR_NO_DEVICE
( Option 4 )

slqstutorial: wGetSessionState: rc = 0xE903, eQCWWAN_ERR_SWIDCS_APP_DISCONNECTED
( Option 5 )

slqstutorial: wStartDataSession: rc = 0xE903, eQCWWAN_ERR_SWIDCS_APP_DISCONNECTED
( Option 6 )

slqstutorial: wStopDataSession: rc = 0xE903, eQCWWAN_ERR_SWIDCS_APP_DISCONNECTED
( Option 7 )

The application is not bound to the SDK and errors are received as shown above
```

5.5. Connection Manager Sample Application

Location: SampleApps/Connection_Manager/

Purpose: Create, delete, view, and modify profiles. Start/stop data sessions.

Build:

i386:	make
ARM:	make CPU=arm9
Power PC:	make CPU=ppc
MIPS BE:	make CPU=mips
MIPS LE:	make CPU=mipsel

Execute:

i386:	sudo ./connectionmgri386
ARM:	sudo ./connectionmgrarm9
PPC:	sudo ./connectionmgrppc
MIPS:	sudo ./connectionmgrmips
MIPS BE:	sudo ./connectionmgrmipsel

5.6. Position Determination Service Sample Application

Location: SampleApps/PDS_Service/

Purpose: Set and Get GPS Service State. Start/stop tracking session.

Build:	i386:	make
	ARM:	make CPU=arm9
	Power PC:	make CPU=ppc
	MIPS BE:	make CPU=mips
	MIPS LE:	make CPU=mipsel

Execute:

i386:	sudo ./pdsservicehosti386
ARM:	sudo ./pdsservicearm9
PPC:	sudo ./pdsserviceppc
MIPS:	sudo ./pdsservicemips
MIPS BE:	sudo ./pdsservicemipsel

5.7. SWIOMA Sample Application

Location: SampleApps/SWIOMA_Application/

Purpose: Set and Get SWIOMADM setting. Start/cancel SWIOMADM session.

Build:	i386:	make
	ARM:	make CPU=arm9
	Power PC:	make CPU=ppc
	MIPS BE:	make CPU=mips
	MIPS LE:	make CPU=mipsel

Execute:

i386:	sudo ./SWIOMASampleApphosti386
ARM:	sudo ./SWIOMASampleApparm9
PPC:	sudo ./SWIOMASampleAppppc
MIPS:	sudo ./SWIOMASampleAppmips
MIPS BE:	sudo ./SWIOMASampleAppmipsel

6. AirVantage Agent Integration

The AirVantage agent was fully integrated to SLQS starting from version 3.2. It is a default service running in the background when SLQS starts. Users don't need to explicitly start the service.

To disable the service, the user needs to properly setup SLQS through a configuration file. SLQS will not start if AirVantage service is missing or not properly setup.

6.1. Auto Start Preprocessor

AGENT_AUTO_START is by default disabled at **pkgs/slqscompile.mak**. To enable, uncomment command line: **`CFLAGS += -DAGENT_AUTO_START`**.

6.2. Agent Configuration File

Location:	Same directory of slqssdk
Name:	.sdk_config
Syntax:	AVA_PATH=absolute path of runtime folder of agent
	e.g. **AVA_PATH=/home/ SDK/AirVantageAgent/build.arm/runtime**
	To disable the agent, set **`AVA_PATH=NO_AVA`** (only first line of config file will be read)

If the configuration file is missing, SLQS will try to search for "**AirVantageAgent/runtime**" in the slqssdk folder. If it is still not found, SDK will not start.

6.3. Agent Constrains

SLQS supports a maximum of three simultaneous applications and AirVantage service uses two of them. Only one extra application is supported when AirVantage is started.

6.4. Agent Source Tree

Location:	SampleApps/AirVantageAgent/avagent_r8m	
Build:	i386:	build_avagent.sh
	arm:	build_avagent.sh arm
Output:	i386:	build.default
	arm:	build.arm

6.5. Start/Stop the AirVantage Agent

To start, run "**start_sdk_hosti686.sh**" or start any sample application.

To stop, run "**stop_sdk_hosti686.sh**".

6.6. AirVantage M2M Cloud

Address: http://eu.airvantage.net/

Usage: Please refer to document [3] AirVantage Agent SLQS Integration Guide.

7. Tools

7.1. DM Logging Tool

Location: /tools/logging/dm

Purpose: This tool can be used to send DM filters to the device and log raw DM packets for real-time analysis with QPST (remote logging option) or post-hoc analysis (local or remote logging).

Build:

i386:	make
ARM:	make CPU=arm9
Power PC:	make CPU=ppc
MIPS BE:	make CPU=mips
MIPS LE:	make CPU=mipsel

Usage: Navigate to **tools/logging/dm** and execute **`./dmcapture.sh`** in a shell.

7.2. RAM Dump Tool

Location: /tools/logging/ramdump

Purpose: This tool supports the capturing of the device RAM contents when the device is in boot and hold mode. RAM contents are saved in files written to the current working directory.

Build:

i386:	make
ARM:	make CPU=arm9
Power PC:	make CPU=ppc
MIPS BE:	make CPU=mips
MIPS LE:	make CPU=mipsel

Usage:

Prior to Execution:

1. Enter the following AT commands:

   `at!entercnd="A710"`

   `at!eroption=0`

2. Either reproduce a crash you are investigating, or reset the device

Execution:

Within a shell, execute the following (for i386):

```
./ramdumptooli386 /dev/ttyUSB<digit>
```

Where /dev/ttyUSB<digit> = DM interface ttyUSB device file in boot and hold mode (usually /dev/ttyUSB0).

Note: This tool works independent of the SDK.

7.3. SQF Filter Editing

Location: /tools/logging/dm/filter/src

Purpose: This tool supports modifying SQF filter via api

Usage:

1. Include header file : sqf.h
2. Create Gobal variable : sqf_t sqf
3. Create buffer for sqf : sqf_createbuffer()
4. Load sqf file (optional) : sqf_load_file()
5. Edit Filter : sqf_set() / sqf_clear()
6. Save sqf file : sqf_save_file()
7. Free Buffer for sqf : sqf_destroybuffer().

Note: This tool works independent of the SDK.

8. SLQS Documentation

To view the SLQS's API documentation:

1. Navigate to **docs/SwiApiReference** and open index.html.
2. Click on the modules tab.
3. Select the module of interest e.g. "Short Message Service (SMS)" module.
4. Select the header file e.g. "qaGobiApiSms.h".

Note: *An API function header's "Device Supported" section contains a list of devices that have been successfully tested against that API.*

9. Remote DM log

9.1. Introduction

9.1.1. Purpose

This document describes steps to capture DM log on Windows machine from the device connected to Linux host.

9.1.2. Overview

The following are the 3 different tools that can be used on Linux host during the process of capturing DM log.

i. Remserial
ii. RelayAgent
iii. Diagnostic

In case of the first two tools(listed above), it is required to have Qualcomm's QPST & QXDM professional to be installed on Windows machine to capture DM log directly from the Linux host.

Remserial is an open source tool whose source code can be downloaded from the link http://lpccomp.bc.ca/remserial/.

Relayagent & Diagnostic are the tools that are available with CnS SDK of Sierra wireless.

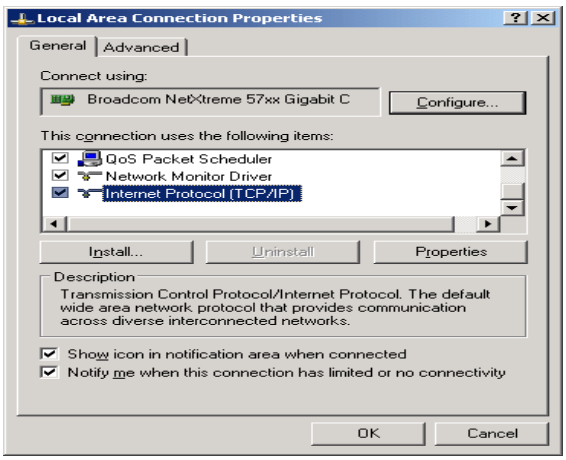
9.2. Capturing DM log on Windows from the Device connected to Linux host

9.2.1. *Using Remserial utility/tool on Linux host*

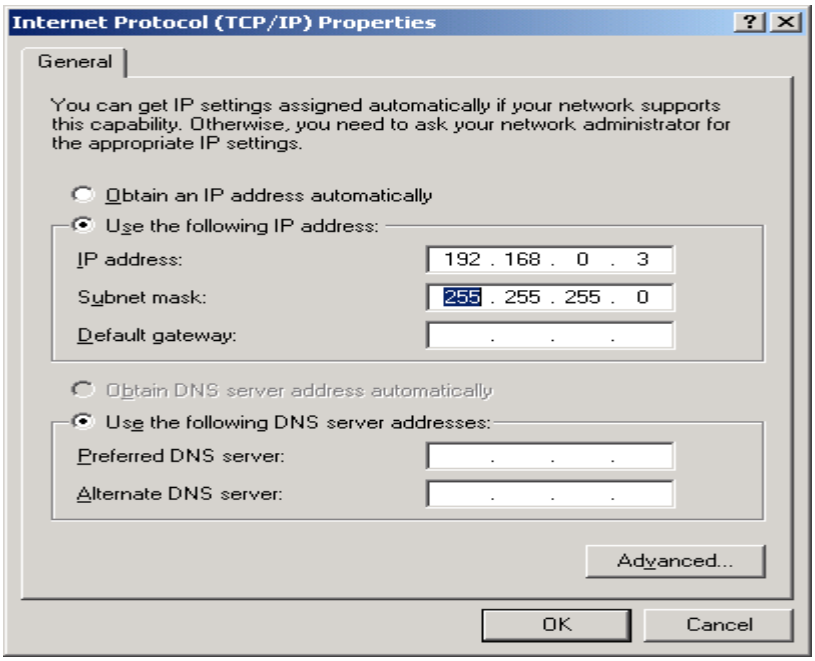
Steps to be followed:

1) Connect Linux and Windows machines using Ethernet cross cable wire.
2) LAN configuration settings on Windows PC:

a) Goto Control Panel -> Network connections ->Right Click on LAN & Select Properties



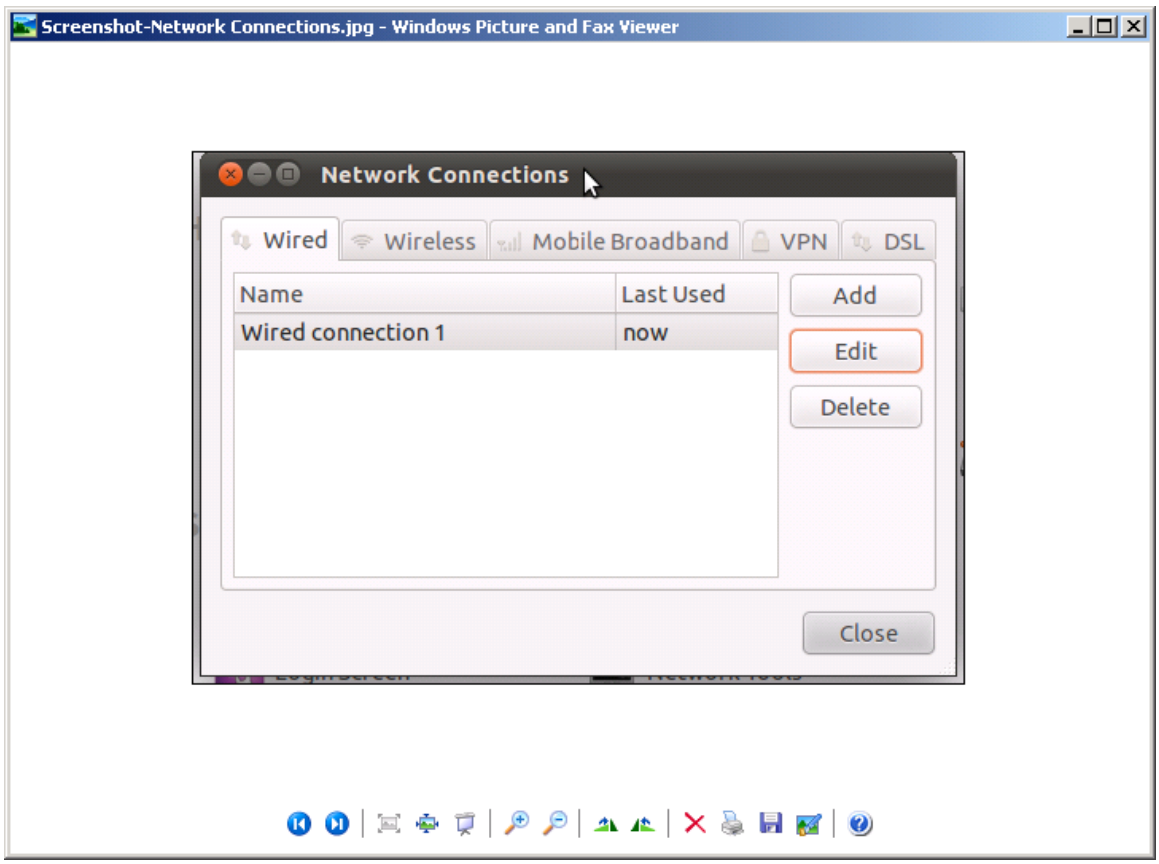
b) Click on properties and configure as follows.



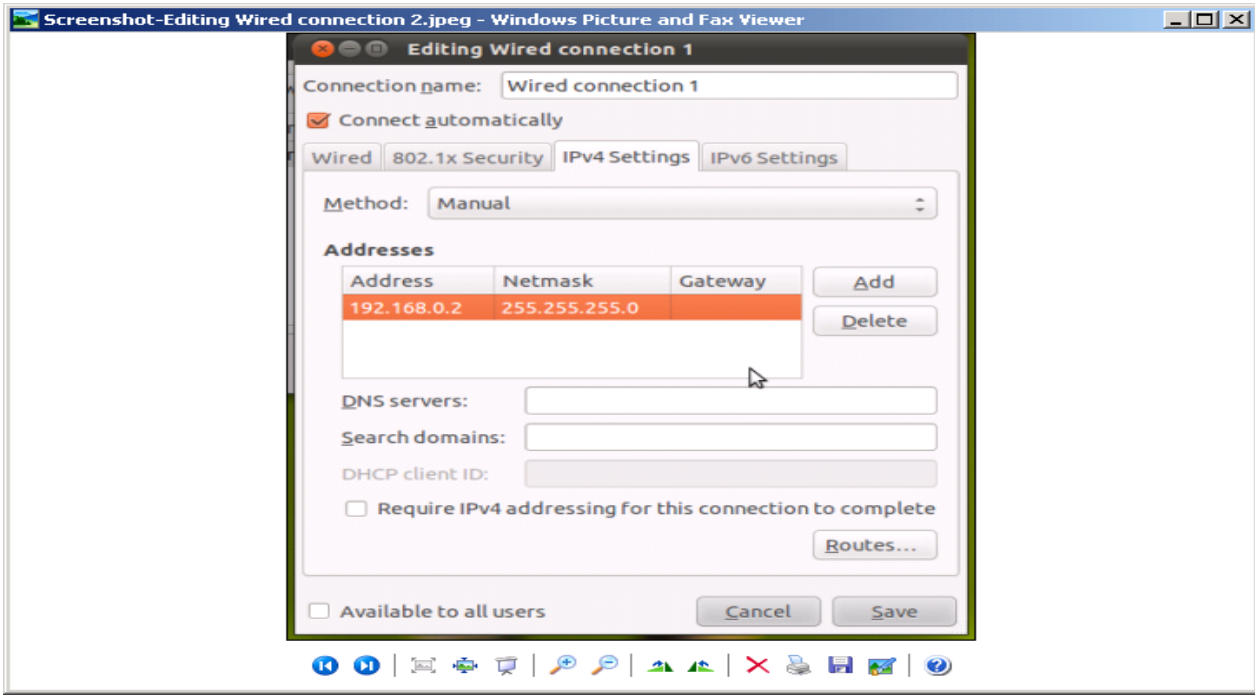
c) Goto command prompt and enter ipconfig which should give the below result
Connection-specific DNS Suffix. :
IP Address . . . . . : 192.168.0.3
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . .:

3) Manual configuration of Wired LAN connection on Linux machine as follows:

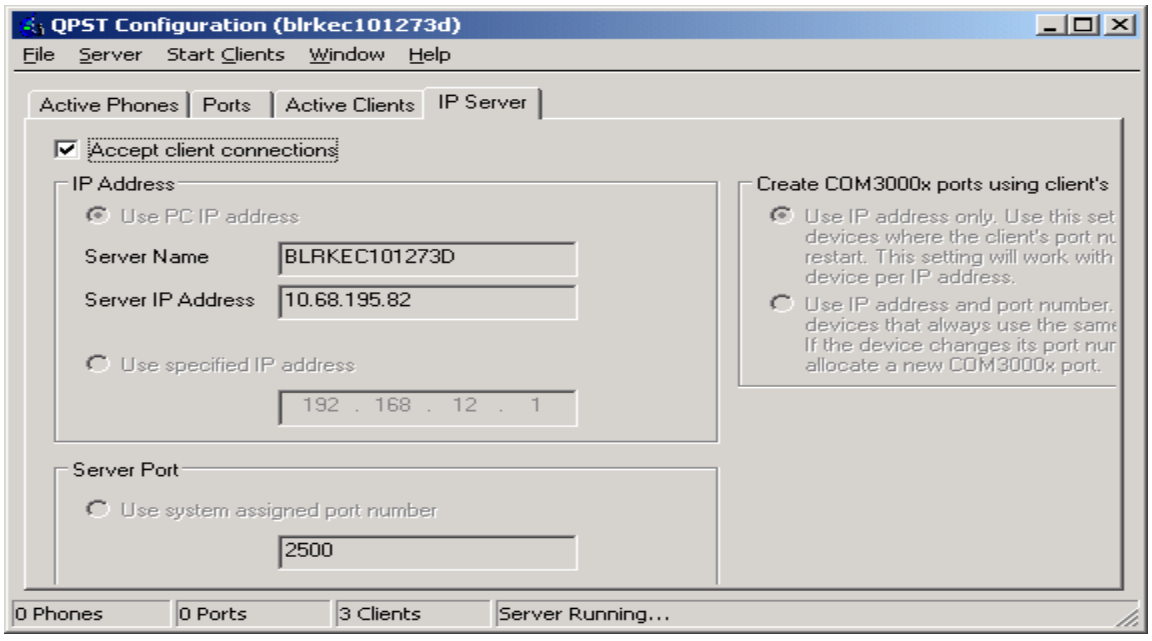
a) Goto Control Center ->Network Connections -> Wired -> Select Wired connection name



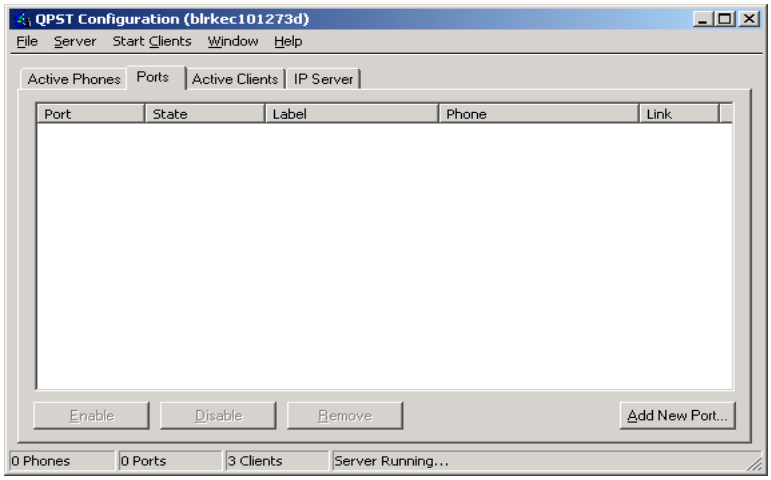
b) Then click on Edit -> IPv4 Settings as follows:



4) QPST Server configuration on Windows machine.



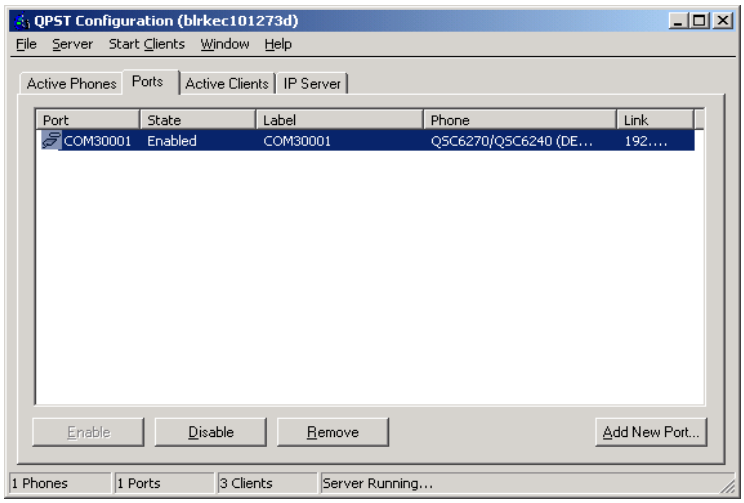
5) Then click on Ports to monitor



6) Navigate to Remserial folder on Linux host.
7) Compile remserial using the command $ sudo make
8) Run the Remserial tool using the following command:

   $sudo ./remserial –r 192.168.0.3 –p 2500 –s “115200 raw” /dev/ttyUSB1

9) If the connection is successful, then the output of QPST configuration Ports will be as follows:

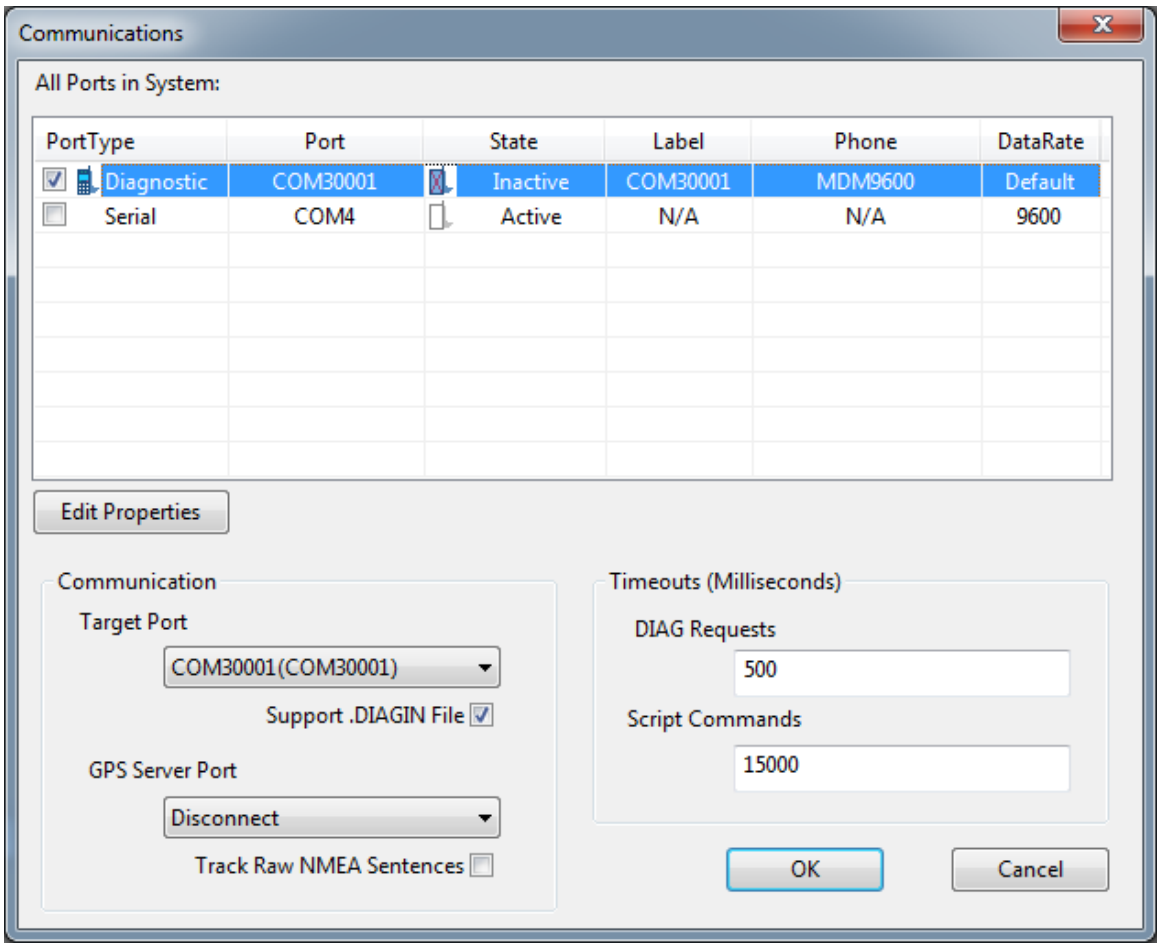


10) Launch QXDM professional on Windows machine and select options -> communications and select the port (COM3001) that is been detected in the above step.

9.3. Connect to QXDM

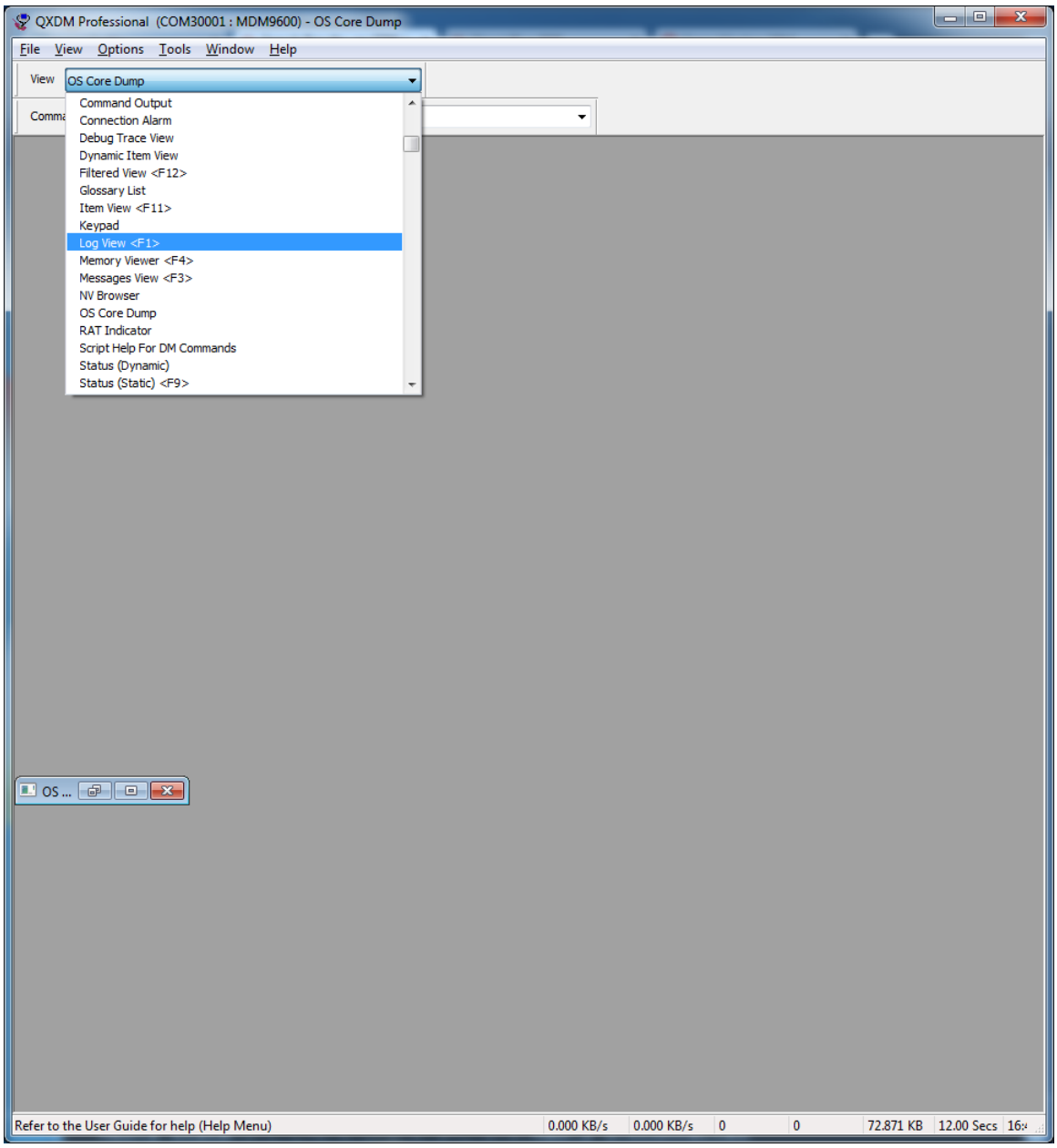
9.3.1. In QXDM menu, select Options then Communications

Select the QPST configured port on the Target Port dropdown box

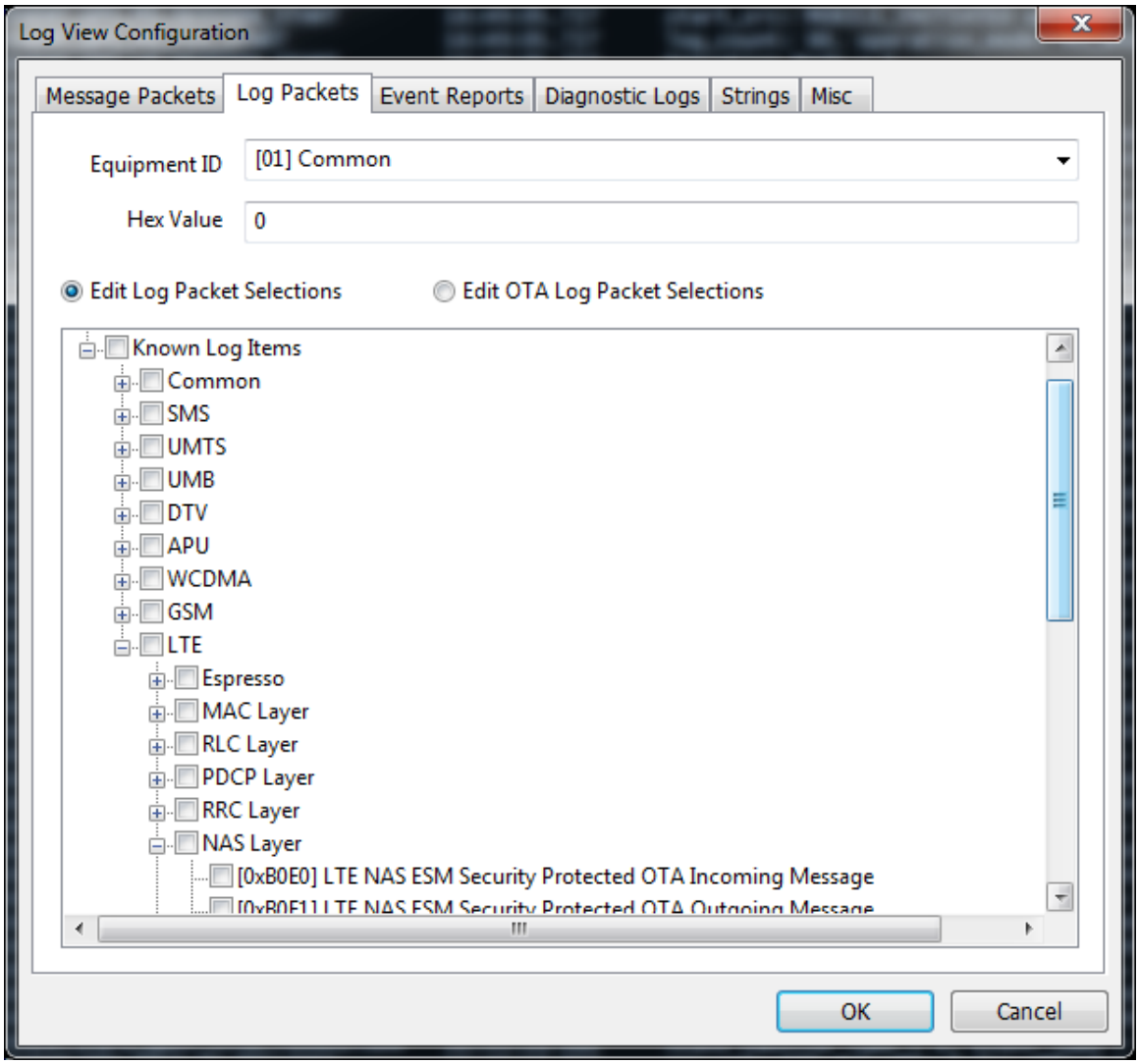


9.3.2. Log Viewer (F1)

Once connected, the title bar will show COM30001: MDM9600 as below


QXDM Professional (COM30001 : MDM9600) - OS Core Dump
File View Options Tools Window Help
View
OS Core Dump
Command Output
Connection Alarm
Debug Trace View
Dynamic Item View
Filtered View <F12>
Glossary List
Item View <F11>
Keypad
Log View <F1>
Memory Viewer <F4>
Messages View <F3>
NV Browser
OS Core Dump
RAT Indicator
Script Help For DM Commands
Status (Dynamic)
Status (Static) <F9>
OS ...
Refer to the User Guide for help (Help Menu)
0.000 KB/s
0.000 KB/s
0
0
72.871 KB
12.00 Secs

9.3.3. Select event to log from Options then Log View Config (F5)



10. Callback that auto re-register

When Modem detach and re-attach, the SDK re-register some callback internally.

Below is the list of automatically re-register callback. For other callback, please manually re-register when modem re-attached.

- SetMobileIPStatusCallback
- SLQSSetWdsEventCallback
- SLQSSetWdsTransferStatisticCallback
- SLQSSetDUNCallInfoCallback
- SLQSSetDataSystemStatusCallback
- SetActivationStatusCallback
- SetPowerCallback
- SLQSSetModemTempCallback
- SetSignalStrengthCallback
- SetRFInfoCallback
- SLQSSetSignalStrengthsCallback
- SetLURejectCallback
- SLQSNasSysInfoCallBack
- SLQSNasNetworkTimeCallBack
- SetNMEACallback
- SetNewSMSCallback
- SLQSSetSMSEventCallback
- SLQSWmsMemoryFullCallBack
- SLQSWmsMessageWaitingCallBack
- SetCATEventCallback
- SetOMADMStateCallback
- SetSLQSOMADMAlertCallback
- SLQSVoiceSetSUPSNotificationCallback
- SLQSVoiceSetAllCallStatusCallBack
- SLQSVoiceSetPrivacyChangeCallBack
- SLQSVoiceSetDTMFEventCallBack
- SLQSVoiceSetSUPSCallBack
- SLQSUIMSetStatusChangeCallBack
- SLQSSetSIPConfigCallback
- SLQSSetRegMgrConfigCallback
- SLQSSetIMSSMSConfigCallback
- SLQSSetIMSUserConfigCallback
- SLQSSetIMSVoIPConfigCallback

11. Reference Documents

[1] SLQS Release Notes

[2] 80-VF459-1 Supplement to Streaming Download Protocol

[3] AirVantage Agent SLQS Integration Guide

Reference: 4115927

12. Debug Information

When SDK is compiled with DEBUG_IPC_MSG_FLAG defined. Additional logs will be captured in syslogs. These logs will print useful information like if a REQ/RESP/NOTIF is sent/received and also prints svc and msg id for that transaction at following points of code flow.

```
1. When request sent from API process to SDK process
Eg:[swi_osapiipcwrite] REQ svc 3 msgid 0x2

2. When request received by SDK process
Eg:[amipcrcvhandler] REQ svc 3 msgid 0x2

3. When response sent from SDK process to API process
Eg:[swi_ossdkipcwrite] RESP svc 3 msgid 0x2

4. When response received by API process
Eg:[amsendnwait] RESP svc 3 msgid 0x2

5. When notification sent from SDK process to API process
Eg:[amapiwaitnotif] NOTIF svc 3 msgid 0x2

6. When notification received by API process
Eg:[swi_osapiipcread] NOTIF svc 3 msgid 0x2

7. A log when a mutex is obtained and released by the API process
Eg:[amgetreqbufp] Mutex Locked
[amrelreqbufp] Mutex Unlocked

8. A log with the timeout value for a particular API
Eg:[SwiQmiMISendnWait]Timeout 2000 seconds
```

For Example:

Sep 8 13:49:29 infy-desktop qatesthostx86_64: SetSignalStrengthCallback - START

Sep 8 13:49:29 infy-desktop qatesthostx86_64: [amgetreqbufp] Mutex Locked

Sep 8 13:49:29 infy-desktop qatesthostx86_64: [SwiQmiMISendnWait]Timeout 2000 seconds

Sep 8 13:49:29 infy-desktop qatesthostx86_64: [swi_osapiipcwrite] REQ svc 3 msgid 0x2

Sep 8 13:49:29 infy-desktop qatesthostx86_64: [swi_osapiipcread] NOTIF svc 0 msgid 0x0

Sep 8 13:49:29 infy-desktop SWI0 SDK Process: [amipcrcvhandler] REQ svc 3 msgid 0x2

Sep 8 13:49:29 infy-desktop SWI0 SDK Process: [swi_ossdkipcread] NOTIF svc 0 msgid 0x0

Sep 8 13:49:29 infy-desktop SWI0 SDK Process: QM:qmqmireq/1390: Request: QMI Instance 0

Sep 8 13:49:29 infy-desktop SWI0 SDK Process: QM:SDK->Mdm: request received : ipcch/svctype/xactionlen/clientnum: 0/0003/14/2

Sep 8 13:49:29 infy-desktop SWI0 SDK Process: QM:SDK->Mdm: request validated : ipcch/svctype/xactionlen/clientnum: 0/0003/14/2

Sep 8 13:49:29 infy-desktop SWI0 SDK Process: QM:Launching QMI DS shell: service 3(NAS)

Sep 8 13:49:29 infy-desktop SWI0 SDK Process: QM:qmqmireq/1501: WDS Request: Active Client 2, WDS Client 0

Sep 8 13:49:29 infy-desktop SWI0 SDK Process: UDIAG:DS Shell launched

Sep 8 13:49:29 infy-desktop SWI0 SDK Process: QMURR1:Endpoint DS shell instance created

Sep 8 13:49:29 infy-desktop SWI0 SDK Process: USB read: bytes2read = 14, read 14 bytes

Sep 8 13:49:29 infy-desktop SWI0 SDK Process: QM:SDK<-Mdm Resp: ch/Msgid/Msglen/IPCmsglen: 0/0002/11/29

Sep 8 13:49:29 infy-desktop SWI0 SDK Process: [swi_ossdkipcwrite] RESP svc 3 msgid 0x2

Sep 8 13:49:29 infy-desktop qatesthostx86_64: [amsendnwait] RESP svc 3 msgid 0x2

Sep 8 13:49:29 infy-desktop qatesthostx86_64: [amrelreqbufp] Mutex Unlocked

Sep 8 13:49:29 infy-desktop qatesthostx86_64: SetSignalStrengthCallback - END

Sep 8 13:52:00 infy-desktop SWI0 SDK Process: USB read: bytes2read = 12, read 12 bytes

Sep 8 13:52:00 infy-desktop SWI0 SDK Process: QM:SDK<-Mdm Notif: ch/Msgid/Msglen/IPCmsglen: 1/0002/9/27

Sep 8 13:52:00 infy-desktop SWI0 SDK Process: [swi_ossdkipcwrite] NOTIF svc 3 msgid 0x2

Sep 8 13:52:00 infy-desktop SWI0 SDK Process: QM:SDK<-Mdm Notif: ch/Msgid/Msglen/IPCmsglen: 3/0002/9/27

Sep 8 13:52:00 infy-desktop kernel: [869620.269915] GobiNet::UpSem 0x0103

Sep 8 13:52:00 infy-desktop kernel: [869620.270156] GobiNet::FindClientMem Found client's 0x103 memory

Sep 8 13:52:00 infy-desktop SWI0 SDK Process: QM:SDK<-Mdm Notif: ch/Msgid/Msglen/IPCmsglen: 5/0002/9/27

Sep 8 13:52:00 infy-desktop qatesthostx86_64: [amapiwaitnotif] NOTIF svc 3 msgid 0x2

Sep 8 13:52:00 infy-desktop qatesthostx86_64: [swi_osapiipcread] NOTIF svc 3 msgid 0x2

Sep 8 13:52:03 infy-desktop SWI0 SDK Process: QM:SDK<-Mdm Notif: ch/Msgid/Msglen/IPCmsglen: 1/0002/9/27

Sep 8 13:52:03 infy-desktop SWI0 SDK Process: [swi_ossdkipcwrite] NOTIF svc 3 msgid 0x2

Sep 8 13:52:03 infy-desktop SWI0 SDK Process: QM:SDK<-Mdm Notif: ch/Msgid/Msglen/IPCmsglen: 3/0002/9/27

Sep 8 13:52:03 infy-desktop SWI0 SDK Process: QM:SDK<-Mdm Notif: ch/Msgid/Msglen/IPCmsglen: 5/0002/9/27

Sep 8 13:52:03 infy-desktop qatesthostx86_64: [amapiwaitnotif] NOTIF svc 3 msgid 0x2

Sep 8 13:52:03 infy-desktop qatesthostx86_64: [swi_osapiipcread] NOTIF svc 3 msgid 0x2

Sep 8 13:52:04 infy-desktop SWI0 SDK Process: USB read: bytes2read = 45, read 45 bytes

Sep 8 13:52:04 infy-desktop SWI0 SDK Process: QM:SDK<-Mdm Notif: ch/Msgid/Msglen/IPCmsglen: 1/0024/42/60

Sep 8 13:52:04 infy-desktop SWI0 SDK Process: [swi_ossdkipcwrite] NOTIF svc 3 msgid 0x36

Sep 8 13:52:04 infy-desktop SWI0 SDK Process: QM:SDK<-Mdm Notif: ch/Msgid/Msglen/IPCmsglen: 3/0024/42/60

Sep 8 13:52:04 infy-desktop SWI0 SDK Process: QM:SDK<-Mdm Notif: ch/Msgid/Msglen/IPCmsglen: 5/0024/42/60

Sep 8 13:52:04 infy-desktop qatesthostx86_64: [amapiwaitnotif] NOTIF svc 3 msgid 0x36

Sep 8 13:52:04 infy-desktop qatesthostx86_64: [swi_osapiipcread] NOTIF svc 3 msgid 0x36

Sep 8 13:52:04 infy-desktop SWI0 SDK Process: USB read: bytes2read = 45, read 45 bytes

Sep 8 13:52:04 infy-desktop SWI0 SDK Process: QM:SDK<-Mdm Notif: ch/Msgid/Msglen/IPCmsglen: 1/0024/42/60

Sep 8 13:52:04 infy-desktop SWI0 SDK Process: [swi_ossdkipcwrite] NOTIF svc 3 msgid 0x36

Sep 8 13:52:04 infy-desktop SWI0 SDK Process: QM:SDK<-Mdm Notif: ch/Msgid/Msglen/IPCmsglen: 3/0024/42/60

Sep 8 13:52:04 infy-desktop SWI0 SDK Process: QM:SDK<-Mdm Notif: ch/Msgid/Msglen/IPCmsglen: 5/0024/42/60

Sep 8 13:52:04 infy-desktop qatesthostx86_64: [amapiwaitnotif] NOTIF svc 3 msgid 0x36

Sep 8 13:52:04 infy-desktop qatesthostx86_64: [swi_osapiipcread] NOTIF svc 3 msgid 0x36

Sep 8 13:52:04 infy-desktop SWI0 SDK Process: USB read: bytes2read = 45, read 45 bytes

Sep 8 13:52:04 infy-desktop SWI0 SDK Process: QM:SDK<-Mdm Notif: ch/Msgid/Msglen/IPCmsglen: 1/0024/42/60

Sep 8 13:52:04 infy-desktop SWI0 SDK Process: [swi_ossdkipcwrite] NOTIF svc 3 msgid 0x36

Sep 8 13:52:04 infy-desktop SWI0 SDK Process: QM:SDK<-Mdm Notif: ch/Msgid/Msglen/IPCmsglen: 3/0024/42/60

Sep 8 13:52:04 infy-desktop SWI0 SDK Process: QM:SDK<-Mdm Notif: ch/Msgid/Msglen/IPCmsglen: 5/0024/42/60

Sep 8 13:52:04 infy-desktop qatesthostx86_64: [amapiwaitnotif] NOTIF svc 3 msgid 0x36

Sep 8 13:52:04 infy-desktop qatesthostx86_64: [swi_osapiipcread] NOTIF svc 3 msgid 0x36

Sep 8 13:52:07 infy-desktop SWI0 SDK Process: USB read: bytes2read = 28, read 28 bytes

Sep 8 13:52:07 infy-desktop SWI0 SDK Process: QM:SDK<-Mdm Notif: ch/Msgid/Msglen/IPCmsglen: 1/0024/25/43

Sep 8 13:52:07 infy-desktop SWI0 SDK Process: [swi_ossdkipcwrite] NOTIF svc 3 msgid 0x36

Sep 8 13:52:07 infy-desktop SWI0 SDK Process: QM:SDK<-Mdm Notif: ch/Msgid/Msglen/IPCmsglen: 3/0024/25/43

Sep 8 13:52:07 infy-desktop SWI0 SDK Process: QM:SDK<-Mdm Notif: ch/Msgid/Msglen/IPCmsglen: 5/0024/25/43

Sep 8 13:52:07 infy-desktop qatesthostx86_64: [amapiwaitnotif] NOTIF svc 3 msgid 0x36

Sep 8 13:52:07 infy-desktop qatesthostx86_64: [swi_osapiipcread] NOTIF svc 3 msgid 0x36