Build a Smarter World

AG35-QuecOpen Developer Guide

LTE Module Series

Rev. AG35-QuecOpen_Developer_Guide_V1.0

Date: 2017-10-18

www.quectel.com

Our aim is to provide customers with timely and comprehensive service. For any assistance, please contact our company headquarters:

Quectel Wireless Solutions Co., Ltd.
7th Floor, Hongye Building, No.1801 Hongmei Road, Xuhui District, Shanghai 200233, China
Tel: +86 21 5108 6236
Email: info@quectel.com

Or our local office. For more information, please visit:
http://quectel.com/support/sales.htm

For technical support, or to report documentation errors, please visit:
http://quectel.com/support/technical.htm
Or email to: support@quectel.com

About the Document

History

Revision	Date	Author	Description
1.0	2017-10-18	Gale GAO/ Gilbert WANG	Initial

Contents

Table Index

Figure Index

1 Introduction

Quectel QuecOpen^{TM} is an open embedded platform that is built on Linux system. It is designed to simplify the development for IoT (Internet of Things) applications.

This guide document is applicable for Quectel AG35 module, and it mainly provides the following information for developers:

- General information about QuecOpen^{TM} platform
- How to start working with QuecOpen^{TM}
- How to build an application
- How to compile an application
- How to download an application
- How to get the application working

2 QuecOpen^{TM} Platform

2.1. System Architecture

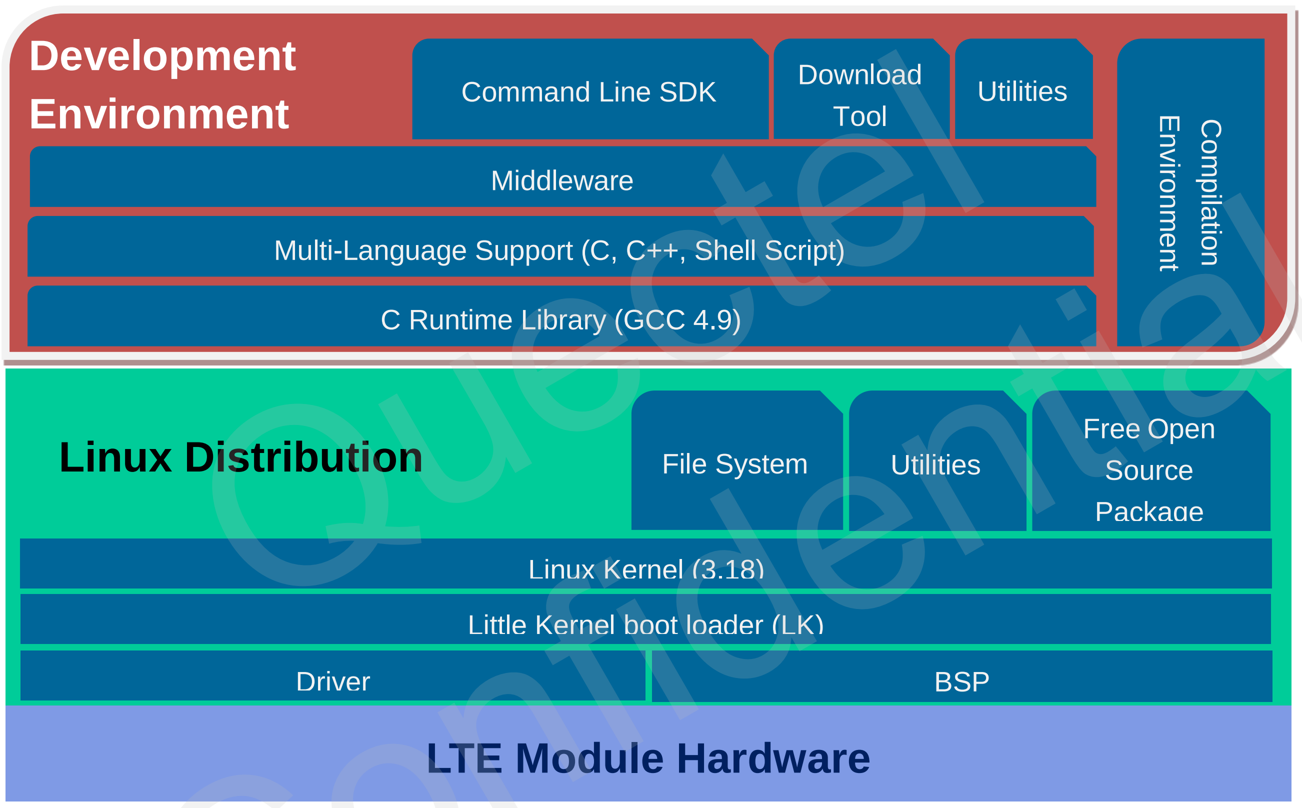


Figure 1: QuecOpen^{TM} Architecture

2.2. Open System Resources

2.2.1. Processor

Cortex-A7 1.2GHz, ARMv7.

2.2.2. Host System

Linux system with kernel 3.18.

2.2.3. Flash Space

In QuecOpen^{TM}, customer applications and other data are stored in Linux file system. The places that can be used by developers are listed in the following table.

Table 1: Data Storage Places for Developers

Filesystem	Type	Size	Used	Available	Use (%)	Mounted on
ubi0:rootfs	ubifs	60.4M	41.6M	18.8M	69%	/
ubi0:usrfs	ubifs	8.7M	284.0K	8.5M	3%	/data
/dev/ubi2_0	ubifs	99.9M	24.0K	99.9M	0%	/usrdata

As defined in the table above, there are three UBI partitions that can be used to store application images or data in Linux file system. They are "rootfs", "usrfs" and "usrdata" partitions.

- *ubi0:rootfs* (/)

The space available is around 18.8MB.

- *ubi0:usrfs* (/data)

The space available is around 8.5MB. And the partition is mounted to */data*. By default, customer applications are put in this partition.

- */dev/ubi2_0* (/usrdata)

The space available is around 99.9MB. And it is mounted to */usrdata* by default. However, this UBI partition is shared with FOTA option, which means FOTA option will erase and write this partition when there is firmware (such as modem image, or Linux image) upgraded by FOTA. So developers may temporarily store some data in */dev/ubi2_0*. However, do remember the data will be cleaned up when doing upgrade by FOTA.

Totally, the fixed flash space (for application and data) is around 27MB (18.8MB+8.5MB), and can be expanded to around 127MB (27MB+100MB). The runtime flash space is around 45MB.

2.2.4. RAM Space

RAM available: 100M bytes

2.3. Open Hardware Resources

Table 2: Multiplexing Pins

Pin Name	Pin No.	Model 1 (Default)	Model 2	Model 3	Reset [1)]	Wake-up Interrupt [2)]	Comment
GPIO1	59	GPIO_38	--	--	B-PD,L	✔	BOOT_ CONFIG_12
GPIO2	61	GPIO_74	--	--	B-PD,L	✔	
GPIO3	62	GPIO_75	--	--	B-PD,L	✔	
GPIO4	144	GPIO_25	--	--	B-PD,L	✔	BOOT_ CONFIG_2
GPIO5	147	GPIO_24	--	--	B-PD,L	✔	BOOT_ CONFIG_1
GPIO6	150	GPIO_42	--	--	B-PD,L	✔	
GPIO7	159	GPIO_58	--	--	B-PD,L	✗	BOOT_ CONFIG_11
BT_EN*	3	BT_EN*	PMU_ GPIO_02	--	--	✗	
PM_ENABLE	5	PM_ENABLE	PMU_ GPIO_03	--	--	✗	
EPHY_RST_N	6	EPHY_ RST_N	GPIO_29	--	B-PD,L	✔	
SGMII_MCLK	7	SGMII_MCLK	GPIO_27	--	B-PD,L	✗	
SGMII_MDATA	8	SGMII_ MDATA	GPIO_28	--	B-PD,L	✔	
EPHY_INT_N	9	EPHY_INT_N	GPIO_30	--	B-PD,L	✔	
SDC1_CMD	18	SDC1_CMD	GPIO_17	--	B-PD,L	✔	

SDC1_CLK	19	SDC1_CLK	GPIO_16	--	B-PD,L	✔	
SDC1_DATA0	20	SDC1_DATA0	GPIO_15	--	B-PD,L	✗	
SDC1_DATA1	21	SDC1_DATA1	GPIO_14	--	B-PD,L	✗	
SDC1_DATA2	22	SDC1_DATA2	GPIO_13	--	B-PD,L	✔	
SDC1_DATA3	23	SDC1_DATA3	GPIO_12	--	B-PD,L	✔	
USIM_ PRESENCE	25	USIM_ PRESENCE	GPIO_34	--	B-PD,L	✔	
I2C1_SDA	42	I2C_SCL_ BLSP4	GPIO_18	--	B-PD,L	✔	
I2C1_SCL	43	I2C_SDA_ BLSP4	GPIO_19	--	B-PD,L	✔	
SDC2_INT_ DET	52	SDC2_INT_ DET	GPIO_26	--	B-PD,L	✔	
UART1_CTS	56	UART_CTS_ BLSP3	GPIO_3	SPI_CLK_ BLSP3	B-PD,L	✔	
UART1_RTS	57	UART_RTS_ BLSP3	GPIO_2	SPI_CS_N _BLSP3	B-PD,L	✗	
UART1_RXD	58	UART_RXD_ BLSP3	GPIO_1	SPI_MISO _BLSP3	B-PD,L	✔	
UART1_TXD	60	UART_TXD_ BLSP3	GPIO_0	SPI_MOSI _BLSP3	B-PD,L	✗	
PCM_SYNC	65	PCM_SYNC	GPIO_79	--	B-PD,L	✔	BOOT_ CONFIG_7
PCM_IN	66	PCM_IN	GPIO_76	--	B-PD,L	✔	
PCM_CLK	67	PCM_CLK	GPIO_78	--	B-PD,L	✗	BOOT_ CONFIG_8
PCM_OUT	68	PCM_OUT	GPIO_77	--	B-PD,L	✗	
I2C_SDA	73	I2C_SDA_ BLSP2	GPIO_6	--	B-PD,L	✗	
I2C_SCL	74	I2C_SCL_ BLSP2	GPIO_7	--	B-PD,L	✗	
SPI_MOSI	77	SPI_MOSI_ BLSP6	GPIO_20	UART_TX D_BLSP6	B-PD,L	✔	
SPI_MISO	78	SPI_MISO_ BLSP6	GPIO_21	UART_RX D_BLSP6	B-PD,L	✔	
SPI_CS_N	79	SPI_CS_N_ BLSP6	GPIO_22	UART_RT S_BLSP6	B-PD,L	✔	

SPI_CLK	80	SPI_CLK_BLSP6	GPIO_23	UART_CTS_BLSP6	B-PU,H	✗	BOOT_CONFIG_4
OTG_PWR_EN*	143	OTG_PWR_EN	GPIO_41	--	B-PD,L	✗	
COEX_UART_TXD	145	COEX_UART_TXD	GPIO_36	--	B-PD,L	✔	BOOT_CONFIG_3
WLAN_EN	149	WLAN_EN	GPIO_54	--	B-PD,L	✔	BOOT_CONFIG_6
WLAN_WAKE	160	WLAN_WAKE	GPIO_59	--	B-PD,L	✔	
UART2_TXD	163	UART_TXD_BLSP5	GPIO_8	SPI_MOSI_BLSP5	B-PD,L	✔	
UART2_CTS	164	UART_CTS_BLSP5	GPIO_11	SPI_CLK_BLSP5	B-PD,L	✔	
UART2_RXD	165	UART_RXD_BLSP5	GPIO_9	SPI_MISO_BLSP5	B-PD,L	✔	
UART2_RTS	166	UART_RTS_BLSP5	GPIO_10	SPI_CS_N_BLSP5	B-PD,L	✗	
WLAN_SLP_CLK	169	WLAN_SLP_CLK	PMU_GPIO_06	--	--	✗	
NET_STATUS	170	NET_STATUS	PMU_GPIO_01	--	--	✗	
STATUS	171	STATUS	PMU_GPIO_04	--	--	✗	

NOTES

1. The pin functions in Model 2 and Model 3 take effect only after software configuration.
1. [1)]“B”: Bidirectional digital with CMOS input
   “BH”: High-voltage tolerant bidirectional digital with CMOS input
   “PD”: Contain an internal pull-down device
   “PU”: Contain an internal pull-up device
   “L”: Low level
   “H”: High level
2. [2)]All GPIOs support interrupt function. But not all interrupts can wake up the sleeping module. The wake-up interrupt function is disabled by default.
3. BOOT_CONFIG_xx and FORCE_USB_BOOT pins are prohibited to be pulled up before the module is powered on.
4. “*” means under development.

2.3.1. GPIOs

There are more than 30 I/O pins that can be configured as general purpose I/O pins. They are multiplexed with other functional pins. All GPIO pins can be accessed by API functions.

Please refer to ***Chapter 4.3.1*** for the API functions for programming GPIO.

2.3.2. Interrupts

All pins that can be multiplexed as GPIO can be interrupt pins. However, not all interrupt pins can wake up the module that is in low-power-consumption mode. Please see the field "Wake-up Interrupt" in ***Table 2***.

Please refer to ***Chapter 4.3.2*** for the API functions for programming interrupt.

2.3.3. UARTs

AG35-QuecOpen module provides four serial ports: one Debug UART and three application UARTs.

- **Debug UART**: Pin 72/71 (RXD/TXD), used to debug the AP system and QuecOpen^{TM} applications.
- **Application UARTs**:
  a) UART1: pin 60/58 (TXD/RXD) + pin 56/57 (CTS/RTS)
  b) UART2: pin 163/165 (TXD/RXD) + pin 164/166 (CTS/ RTS)
  c) UART3: pin 77/78 (TXD/RXD) + pin 80/79 (CTS/RTS)

Please see ***Chapter 4.3.3*** for details about how to program these serial interfaces.

2.3.4. ADC

AG35-QuecOpen module provides three analog-to-digital converter (ADC) interfaces. The voltage value on ADC pins can be read via **AT+QADC=<port>** command, through setting **<port>** into 0, 1 or 2. For more details about the AT command, please refer to ***document [5]***.

- **AT+QADC=0**: read the voltage value on ADC0
- **AT+QADC=1**: read the voltage value on ADC1
- **AT+QADC=2**: read the voltage value on ADC2

In order to improve the accuracy of ADC, the traces of ADC interfaces should be surrounded by ground.

Table 3: Pin Definition of ADC Interfaces

Pin Name	Pin No.	Description
ADC2	172	General purpose analog to digital converter interface
ADC1	175	General purpose analog to digital converter interface
ADC0	173	General purpose analog to digital converter interface

2.3.5. I2C

AG35-QuecOpen module provides two hardware I2C interfaces. Please refer to ***Chapter 4.3.5*** for details.

2.3.6. SPI

AG35-QuecOpen module provides three hardware SPI interfaces. Please refer to ***Chapter 4.3.6*** for details.

2.3.7. PCM

AG35-QuecOpen module provides a PCM interface. It is designed for audio codec by default. Combined with I2C interface, the application can control and manage the codec chip using **AT+QIIC** command or API functions defined in ***Chapter 4.3.5***.

2.3.8. SDIO

AG35-QuecOpen module provides two SDIO interfaces (SDC1 and SDC2). Both of them are 4-bit bidirectional data bus.

SDC2 is designed for SD card or eMMC flash, and SDC1 is designed for Wi-Fi function. For more details, please refer to ***document [2]***.

2.3.9. SGMII

AG35-QuecOpen module provides an SGMII interface.

2.3.10. USB

The USB interface can be mapped into several different functional interfaces, as shown below:

- USB-AT port

- USB-DM port
- USB-NMEA port
- USB-Modem port
- USB-Network adapter

In AG35-QuecOpen module, the GNSS NMEA is outputted to application through a virtual serial port instead of USB-NMEA port by default.

The USB-DM port can be used to download firmware to module, and debug the module system. So, developers MUST design the USB interface for the convenience of downloading and debugging.

Please refer to ***document [1]*** for USB details.

3 Work with QuecOpen^{TM}

This chapter introduces how to start working with QuecOpen^{TM}, For more details about SDK development, please refer to ***document [4]***.

3.1. Set up Host Environment

3.1.1. System Requirements

QuecOpen^{TM} provides the ready-made compilation software package which includes the host operating system, compiler and the other required tools.

- **Operating system**
  Ubuntu 64-bit OS, version 14.04 or later.
- **Compiler**
  arm-oe-linux-gnueabi
- **ADB**
  Android Debug Bridge version 1.0.31.
- **Fastboot**

3.1.2. Install USB Driver

Please refer to ***document [1]*** for details about USB driver installation, and make sure the USB interface can work normally.

3.1.3. Install and Set up ADB Driver on PC

ADB can be used to upload or download files between the host computer and AG35-QuecOpen module, and execute shell commands of the module.

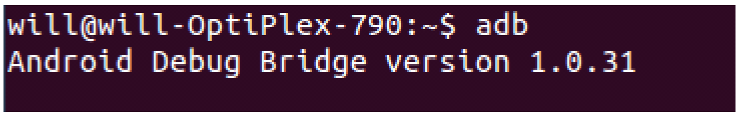
Please follow the commands below to install ADB on your host system.

```
sudo apt-get update
sudo apt-get install android-tools-adb
```

If the above installation command fails, please try again with the series of commands shown below.

```
sudo add-apt-repository ppa:nilarimogard/webupd8
sudo apt-get update
sudo apt-get install android-tools-adb
```

After ADB installation, developers can check whether it is installed successfully by executing adb command.

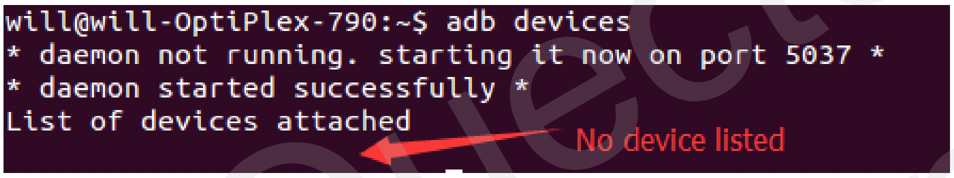


```
will@will-OptiPlex-790:~$ adb
Android Debug Bridge version 1.0.31
```

● **List ADB devices**

```
sudo adb devices
```

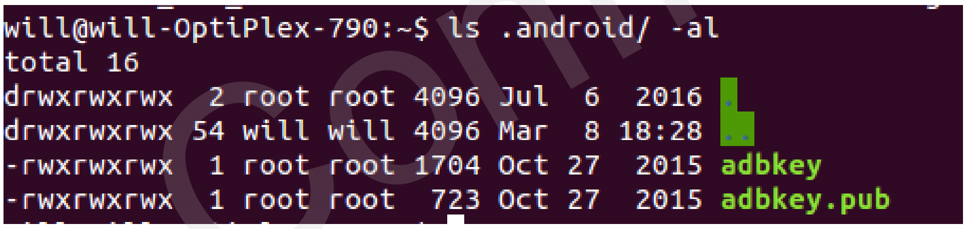
If the USB device cannot be listed, follow the steps below.



```
will@will-OptiPlex-790:~$ adb devices
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
List of devices attached
```

● **Add module USB VID into ADB configuration**

There is a hidden directory name *.android* in your work directory. Please create the file *adb_usb.ini* in *.android* directory, and write into the USB VID in a new line.



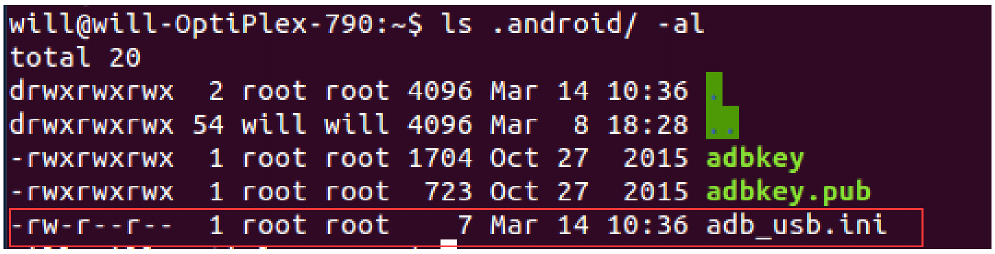
```
will@will-OptiPlex-790:~$ ls .android/ -al
total 16
drwxrwxrwx  2 root root 4096 Jul  6  2016 .
drwxrwxrwx 54 will will 4096 Mar  8 18:28 ..
-rwxrwxrwx  1 root root 1704 Oct 27  2015 adbkey
-rwxrwxrwx  1 root root  723 Oct 27  2015 adbkey.pub
```

```
cd
sudo gedit .android/adb_usb.ini &
```

Or

```
sudo vi .android/adb_usb.ini
```

Input the USB VID in a new line in the file, for example 0x2C7C. Developers can check the USB VID with lsusb command.



```
will@will-OptiPlex-790:~$ ls .android/ -al
total 20
drwxrwxrwx  2 root root 4096 Mar 14 10:36 .
drwxrwxrwx 54 will will 4096 Mar  8 18:28 ..
-rwxrwxrwx  1 root root 1704 Oct 27  2015 adbkey
-rwxrwxrwx  1 root root  723 Oct 27  2015 adbkey.pub
-rw-r--r--  1 root root    7 Mar 14 10:36 adb_usb.ini
```

- **Restart ADB Service**

```
sudo adb kill-server
sudo adb devices
stanley@stanley-OptiPlex-7020:~$ sudo adb devices
List of devices attached
???????????? device  adb device
```

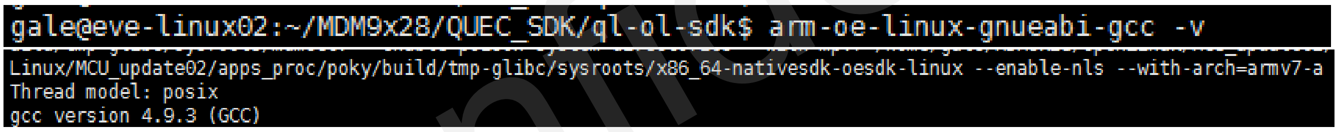
Now the ADB is ready.

3.1.4. Install Cross Compiler

Quectel provides the development environment software package (please consult Quectel Technical Supports for the compiler). Developers just need to decompress the package *ql-ol-sdk.tar.bz2*:

```
tar jxvf ql-ol-sdk.tar.bz2
source ql-ol-crosstool/ql-ol-crosstool-env-init
```

If the environment is properly setup, the information of GCC version will be shown.



```
gale@eve-linux02:~/MDM9x28/QUEC_SDK/ql-ol-sdk$ arm-oe-linux-gnueabi-gcc -v
Linux/MCU_update02/apps_proc/poky/build/tmp-glibc/sysroots/x86_64-nativesdk-oesdk-linux --enable-nls --with-arch=armv7-a
Thread model: posix
gcc version 4.9.3 (GCC)
```

3.2. Compilation

3.2.1. Compiling

In QuecOpen^{TM}, compiling commands are executed in command line. Compiling commands are defined as below.

```
cd ql-ol-extsdk/example/adc
make
```

QuecOpen^{TM} SDK provides some examples for developers' reference. The following illustrates an example for compiling "hello world".

```
ql-ol-extsdk
 ├──example
 |     ├──adc
 |     ├──at
 |     ├──gpio
 |     ├──hello_world
…
```

3.3. Download Application

3.3.1. During Development Phase

3.3.1.1. Download Application with ADB

Developers can download the application executable file to the file system of module Linux system by ADB shell command.

```
sudo adb push <local path> <module path>
```

An example is shown below:

```
adb push ~/ql-ol-sdk/ql-ol-extsdk/example/helloWorld/hellolworld /usrdata
```

3.4. Launch Application

If developers want to run the executable file, they can use adb shell to login the Linux console of AG35-QuecOpen module and run the executable program directly.

```
ql-ol-extsdk/example/helloWorld$ sudo adb shell
/ # ls /usrdata/ -l
total 4
-rwxrwxrwx    1 root     root          3692 Jul  5  2016 helloworld
/ # ./usrdata/helloworld
<Hello QuecOpen !>
atoi("19.7")=19
/ #
```

3.5. Debug Application

Generally, there are two ways to debug application.

- Output log message to standard output device by calling **printf()**, and then developers can catch the log through Debug UART or adb shell.
- Output log message through UART interface.

4 Programming Reference

4.1. System

4.1.1. Time

- **Relevant header files**

```
#include <time.h>
```

- **API functions**

/* Return a string in the form of "Day Mon dd hh:mm:ss yyyy\n" that is the representation of TP in this format. */

```
extern char *asctime (const struct tm *__tp);
```

/* Get resolution of clock CLOCK_ID. */

```
extern int clock_getres (clockid_t __clock_id, struct timespec *__res);
```

/* Get current value of clock CLOCK_ID and store it in TP. */

```
extern int clock_gettime (clockid_t __clock_id, struct timespec *__tp);
```

/* Set clock CLOCK_ID to value TP. */

```
extern int clock_settime (clockid_t __clock_id, const struct timespec *__tp);
```

- **Example**

Please see *example/time/example_time.c* for details.

4.1.2. Timer

- **Relevant header files**

```
#include <sys/timerfd.h>
#include <time.h>
#include <unistd.h>
```

- **API functions**

/* Return file descriptor for new interval timer source. */

```
extern int timerfd_create (clockid_t __clock_id, int __flags);
```

/* Set next expiration time of interval timer source UFD to UTMR. If FLAGS has the TFD_TIMER_ABSTIME flag set the timeout value is absolute. Optionally return the old expiration time in OTMR. */

```
extern int timerfd_settime (int __ufd, int __flags,
                            const struct itimerspec *__utmr,
                            struct itimerspec *__otmr);
```

/* Return the next expiration time of UFD. */

```
extern int timerfd_gettime (int __ufd, struct itimerspec *__otmr);
```

/* Read NBYTES into BUF from FD. Return the number read, -1 for errors or 0 for EOF. */

```
extern ssize_t read (int __fd, void *__buf, size_t __nbytes);
```

/* Create new per-process timer using CLOCK_ID. */

```
extern int timer_create (clockid_t __clock_id,
                         struct sigevent *__restrict __evp,
                         timer_t *__restrict __timerid);
```

/* Set timer TIMERID to VALUE, returning old value in OVALUE. */

```
extern int timer_settime (timer_t __timerid, int __flags,
                          const struct itimerspec *__restrict __value,
                          struct itimerspec *__restrict __ovalue);
```

/* Get current value of timer TIMERID and store it in VALUE. */

```
extern int timer_gettime (timer_t __timerid, struct itimerspec *__value);
```

- **Example**

Please see *example/timer/example_timer.c* for details.

4.1.3. Multitasking

- **Relevant header files**

```
#include <pthread.h>
```

- **API functions**

/* Create a new thread, starting with execution of START-ROUTINE getting passed ARG. Creation attributed come from ATTR. The new handle is stored in *NEWTHREAD. */

```
extern int pthread_create (pthread_t *__restrict __newthread,
                     const pthread_attr_t *__restrict __attr,
                     void *(*__start_routine) (void *),
                     void *__restrict __arg) __THROWNL __nonnull ((1, 3));
```

/* Terminate calling thread. The registered cleanup handlers are called via exception handling so we cannot mark this function with __THROW. */

```
extern void pthread_exit (void *__retval) __attribute__ ((__noreturn__));
```

/* Make calling thread wait for termination of the thread TH. If THREAD_RETURN is not NULL , the exit status of the thread is stored in *THREAD_RETURN. */

```
extern int pthread_join (pthread_t __th, void **__thread_return);
```

- **Example**

Please see *example/pthread/example_pthread.c* for details.

4.2. AT

The tty device */dev/smd8* is designed to be AT port. Developers may simply open this device file and write AT commands through it.

```
#define QUEC_AT_PORT       "/dev/smd8"
smd_fd = open(QUEC_AT_PORT, O_RDWR | O_NONBLOCK | O_NOCTTY);
iRet = write(smd_fd, "AT\r\n", 4);
```

- **Example**

Please see *example/at/example_at.c* for details.

4.3. I/O Interfaces

All kinds of multiplexing interfaces and the quantity of each kind are defined in ***document [3]***.

4.3.1. GPIOs

All programmable GPIO pins are defined in ***Table 2***. Except the dedicated pins, all other pins can be programmed as GPI or GPO.

- **Relevant header files**

```
#include “ql_oe.h”
#include “ql_gpio_def.h”
#include “ql_gpio.h”
#include “gpioSysfs.h”
```

- **API functions**

```
int Ql_GPIO_Init(Enum_PinName           pinName,
                 Enum_PinDirection   dir,
                 Enum_PinLevel          level,
                 Enum_PinPullSel       pullSel
                 );
int Ql_GPIO_SetLevel(Enum_PinName pinName, Enum_PinLevel level);
int Ql_GPIO_GetLevel(Enum_PinName pinName);
int Ql_GPIO_SetDirection(Enum_PinName pinName, Enum_PinDirection dir);
int Ql_GPIO_GetDirection(Enum_PinName pinName);
int Ql_GPIO_SetPullSelection(Enum_PinName pinName, Enum_PinPullSel pullSel);
int Ql_GPIO_GetPullSelection(Enum_PinName pinName);
int Ql_GPIO_Uninit(Enum_PinName pinName);
```

- **Example**

Please see *example/gpio/example_gpio.c* for details.

4.3.2. EINT

All programmable GPIO pins are defined in ***Table 2***. Except the dedicated pins, all other pins can be programmed as GPI and interrupt.

- **Relevant header files**

```
#include "ql_oe.h"
#include "ql_eint.h"
```

- **API functions**

```
int Ql_EINT_Open(Enum_PinName eintPinName);
int Ql_EINT_Enable(Enum_PinName eintPinName, Enum_EintType eintType);
int Ql_EINT_Disable(Enum_PinName eintPinName);
int Ql_EINT_Close(Enum_PinName eintPinName);
```

- **Example**

Please see *example/eint/example_eint.c* for details.

4.3.3. UARTs

AG35-QuecOpen module provides one debug UART and three UART interfaces for application. All of the three application UARTs support hardware handshaking. The three application UART pins are defined in the table below.

Table 4: Application UART Pins

Pin No.	Pin Name	Pin Location	Combined Interface (default)	Pin Multiplexing			Power Domain	Reset	Wake-up Interrupt	Remark
				Primary Function	Alternate Function 1	Alternate Function 2				
79	SPI_CS_N	Edge	SPI Interface	SPI_CS_N_BLSP6	GPIO_22	UART_RTS_BLSP6	1.8V	B-PD,L	✔	
77	SPI_MOSI	Edge	SPI Interface	SPI_MOSI_BLSP6	GPIO_20	UART_TXD_BLSP6	1.8V	B-PD,L	✔	
78	SPI_MISO	Edge	SPI Interface	SPI_MISO_BLSP6	GPIO_21	UART_RXD_BLSP6	1.8V	B-PD,L	✔	
80	SPI_CLK	Edge	SPI Interface	SPI_CLK_BLSP6	GPIO_23	UART_CTS_BLSP6	1.8V	B-PU,H	✗	BOOT_CONFIG_4
163	UART2_TXD	Edge	UART2 interface	UART_TXD_BLSP5	GPIO_8	SPI_MOSI_BLSP5	1.8V	B-PD,L	✔	
164	UART2_CTS	Edge	UART2 interface	UART_CTS_BLSP5	GPIO_11	SPI_CLK_BLSP5	1.8V	B-PD,L	✔	
165	UART2_RXD	Edge	UART2 interface	UART_RXD_BLSP5	GPIO_9	SPI_MISO_BLSP5	1.8V	B-PD,L	✔	
166	UART2_RTS	Edge	UART2 interface	UART_RTS_BLSP5	GPIO_10	SPI_CS_N_BLSP5	1.8V	B-PD,L	✗	
56	UART1_CTS	Edge	UART1 interface	UART_CTS_BLSP3	GPIO_3	SPI_CLK_BLSP3	1.8V	B-PD,L	✓	
57	UART1_RTS	Edge	UART1 interface	UART_RTS_BLSP3	GPIO_2	SPI_CS_N_BLSP3	1.8V	B-PD,L	✗	

60	UART1_TXD	Edge	UART_TXD_BLSP3	GPIO_0	SPI_MOSI_BLSP3	1.8V	B-PD,L	✗
58	UART1_RXD	Edge	UART_RXD_BLSP3	GPIO_1	SPI_MISO_BLSP3	1.8V	B-PD,L	✓

The tty device for UART interfaces are defined as below.

Table 5: UART tty

Pin No.	Pin Name	Signal Direction	tty Device	Remark
163	UART2_TXD	UART TX	/dev/ttyHSL1	Need to configure device tree first.
165	UART2_RXD	UART RX		
77	SPI_MOSI	UART TX	/dev/ttyHSL2	Need to configure device tree first.
78	SPI_MISO	UART RX		
60	UART1_TXD	UART TX	/dev/ttyHS0	
58	UART1_RXD	UART RX		

For pin 163/165 and pin 77/78, developers need to configure the Linux device tree to enable the UART option. The default pin function can be UART or GPIO by configuring Linux device tree. Please refer to the related document to configure device tree.

4.3.3.1. Pin 163/165 as UART

The group of pins generate the tty device */dev/ttyHSL1* in Linux system. Developers may simply open this device to send/receive data.

If hardware handshaking is to be used, pin 164/166 (with CTS/RTS function) should be used together with pin 163/165.

NOTE

By default, the UART interface (DCD/DTR) cannot be tested directly on EVB board. Something more (jump wire) need to be done to test the interface. There are two ways that can be utilized, as illustrated below.

- **Test Solution I: TE-A Module + EVB (×1)**

Step 1: Do not insert the TE-A module into EVB but independently supply power to it.
Step 2: Use jump wires to connect DCD/DTR on TE-A module to TXD_V1.8V/ RXD_V1.8V at J806 on EVB.
Step 3: Connect the Main UART port on EVB to PC for testing.

- **Test Solution II: TE-A Module + EVB (×2)**

Step 1: Prepare two EVB boards (EVB-A and EVB-B). Get rid of the resistors in DCD and DTR lines on EVB-A.
Step 2: Insert TE-A module into EVB-A. And use jump wires to connect DCD/DTR on EVB-A to TXD_V1.8V/RXD_V1.8V at J806 on EVB-B.
Step 3: Supply power to EVB-B.
Step 4: Connect the Main UART port on EVB-B to PC for testing.

4.3.3.2. Pin 77/78 as UART

The two pins can be multiplexed for UART and SPI functions. Developers can configure the multiplexing mode of them by reconfiguring Linux device tree before programming the pins.

The group of pins generate the tty device */dev/ttyHSL2* in Linux system. Developers may simply open this device to send/receive data.

If hardware handshaking is to be used, pin 79/80 (with RTS/CTS function) should be used together with pin 77/78. Pin 79 & 80 are multiplexing pins designed for SPI function by default, and they can offer alternate function of RTS/CTS. Developers can configure the multiplexing mode of pin 79 & 80 by reconfiguring Linux device tree before using them for hardware handshaking.

Test Method:

Use jump wire to connect the pins (pin 77/78) on AG35-QuecOpen module to the UART port on Quectel EVB or your own evaluation board. And the level conversion chip (TTL-to-232) is required.

4.3.3.3. Pin 60/58 as UART

On UMTS & LTE EVB R2.0, there are all pinouts for the full-featured UART1.

The group of pins generate the tty device */dev/ttyHS0* in Linux system. Developers may simply open this device to transfer data.

If hardware handshaking is to be used, pin 56/57 (with primary function of CTS/RTS) should be used together with pin 60/58.

4.3.3.4. Programming Reference

- **Relevant header files**

```
#include "ql_oe.h"
#include "ql_uart.h"
```

- **API functions**

```
int Ql_UART_Open(const char* port, unsigned int baudrate, Enum_FlowCtrl flowCtrl);
int Ql_UART_Read(int fd, char* buf, unsigned int buf_len);
int Ql_UART_Write(int fd, const char* buf, unsigned int buf_len);
int Ql_UART_SetDCB(int fd, ST_UARTDCB *dcb);
int Ql_UART_GetDCBConfig(int fd, ST_UARTDCB *dcb);
int Ql_UART_IoCtl(int fd, unsigned int cmd, void* pValue);
int Ql_UART_Close(int fd);
```

/* Check the first NFDS descriptors each in READFDS (if not NULL) for read readiness, in WRITEFDS (if not NULL) for write readiness, and in EXCEPTFDS (if not NULL) for exceptional conditions. If TIMEOUT is not NULL, time out after waiting the interval specified therein. Returns the number of ready descriptors, or -1 for errors. */

```
extern int select (int __nfds, fd_set *__restrict __readfds,
               fd_set *__restrict __writefds,
               fd_set *__restrict __exceptfds,
               struct timeval *__restrict __timeout);
```

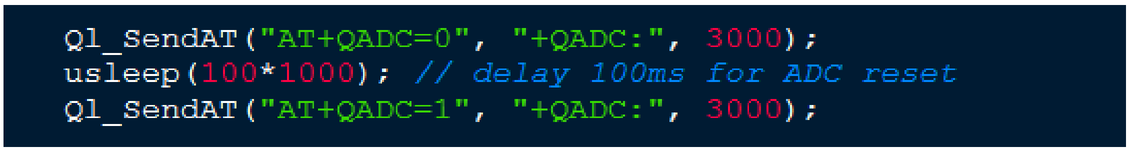
- **Example**

Please see *example/uart/example_uart.c* and *example_uart_at.c* for details.

4.3.4. ADC

In hardware, please refer to ***document [2]*** for the pin definition.

In software, developers may sample ADC value by sending **AT+QADC=0/1/2** through the tty device */dev/smd8*.



```
    Ql_SendAT("AT+QADC=0", "+QADC:", 3000);
    usleep(100*1000); // delay 100ms for ADC reset
    Ql_SendAT("AT+QADC=1", "+QADC:", 3000);
```

- **Example**

Please see *example/adc/example_adc.c* for details.

4.3.5. I2C

AG35-QuecOpen module provides two I2C interfaces. The I2C interfaces can be host only.

Table 6: I2C Pins

Pin No.	Pin Name	Pin Location	Combined Interface (default)	Pin Multiplexing			Power Domain	Reset	Wake-up Interrupt	Remark
				Primary Function	Alternate Function 1	Alternate Function 2				
42	I2C1_SDA	Edge	I2C interface, host only	I2C_SDA_BLSP4	GPIO_18	-	1.8V	B-PD,L	✔	
43	I2C1_SCL	Edge		I2C_SCL_BLSP4	GPIO_19	-	1.8V	B-PD,L	✔	
74	I2C2_SCL	Edge	I2C interface, host only	I2C_SCL_BLSP2	GPIO_7	-	1.8V	B-PD,L	✗	
73	I2C2_SDA	Edge		I2C_SDA_BLSP2	GPIO_6	-	1.8V	B-PD,L	✗	

- **Relevant header files**

```
#include "ql_oe.h"
#include "ql_i2c.h"
```

- **API functions**

```
int Ql_I2C_Init(unsigned char slaveAddr);
int Ql_I2C_Read(int fd, unsigned short slaveAddr, unsigned char ofstAddr,  unsigned char* ptrBuff,
unsigned short length);
int Ql_I2C_Write(int fd, unsigned short slaveAddr, unsigned char ofstAddr,  unsigned char* ptrData,
unsigned short length);
```

- **Example**

Please see *example/i2c/example_i2c.c* for details.

4.3.6. SPI

AG35-QuecOpen module provides three SPI interfaces (two of them multiplexed from UARTs) which support only master mode with a maximum baud rate up to 50MHz, and the default is 19.2MHz.. The interfaces can be host only.

Table 7: SPI Pins

Pin No.	Pin Name	Pin Location	Combined Interface (default)	Pin Multiplexing			Power Domain	Reset	Wake-up Interrupt	Remark
				Primary Function	Alternate Function 1	Alternate Function 2				
79	SPI_CS_N	Edge	SPI Interface	SPI_CS_N_BLSP6	GPIO_22	UART_RTS_BLSP6	1.8V	B-PD,L	✓	
77	SPI_MOSI	Edge		SPI_MOSI_BLSP6	GPIO_20	UART_TXD_BLSP6	1.8V	B-PD,L	✓	
78	SPI_MISO	Edge		SPI_MISO_BLSP6	GPIO_21	UART_RXD_BLSP6	1.8V	B-PD,L	✓	
80	SPI_CLK	Edge		SPI_CLK_BLSP6	GPIO_23	UART_CTS_BLSP6	1.8V	B-PU,H	✗	BOOT_CONFIG_4
56	UART1_CTS	Edge	UART1	UART_CTS_BLSP3	GPIO_3	SPI_CLK_BLSP3	1.8V	B-PD,L	✓	
57	UART1_RTS	Edge		UART_RTS_BLSP3	GPIO_2	SPI_CS_N_BLSP3	1.8V	B-PD,L	✗	
58	UART1_RXD	Edge		UART_RXD_BLSP3	GPIO_1	SPI_MISO_BLSP3	1.8V	B-PD,L	✓	
60	UART1_TXD	Edge		UART_TXD_BLSP3	GPIO_0	SPI_MOSI_BLSP3	1.8V	B-PD,L	✗	
163	UART2_TXD	Edge	UART2	UART_TXD_BLSP5	GPIO_8	SPI_MOSI_BLSP5	1.8V	B-PD,L	✓	
164	UART2_CTS	Edge		UART_CTS_BLSP5	GPIO_11	SPI_CLK_BLSP5	1.8V	B-PU,L	✓	
165	UART2_RXD	Edge		UART_RXD_BLSP5	GPIO_9	SPI_MISO_BLSP5	1.8V	B-PD,L	✓	
166	UART2_RTS	Edge		UART_RTS_BLSP5	GPIO_10	SPI_CS_N_BLSP5	1.8V	B-PD,L	✗	

4.3.6.1. SPI Mode

In QuecOpen system, the SPI device drivers have been compiled to .ko modules. There are two .ko modules existing in the Linux system of AG35-QuecOpen module: spidev.ko, quec_spi_chn.ko.

```
cd /usr/lib/modules/3.18.20/kernel/drivers/spi#
ls
quec_spi_chn.ko    spidev.ko
```

The two SPI driver modules are for different application cases.

- **4-line mode:** spidev.ko

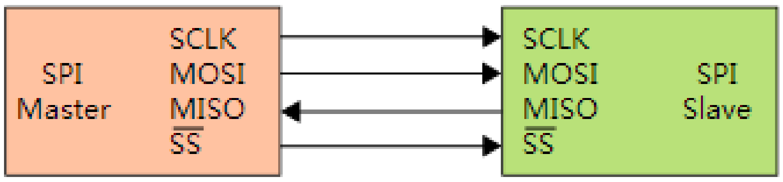


Figure 2: SPI 4-Line Mode

- **6-line mode:** quec_spi_chn.ko

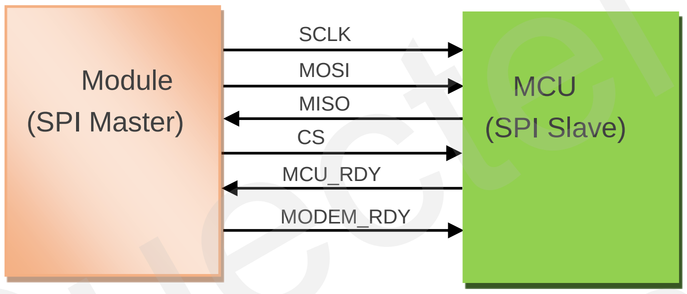


Figure 3: SPI 6-Line Mode

In 6-line mode SPI, from the module side, MCU_RDY should be an interrupt pin that can wake up the module, and MODEM_RDY can be any GPIO. Please refer to ***Table 2*** for details.

4.3.6.2. Install SPI Driver

By default, the SPI device is not present. Developers need to install the SPI driver (.ko module) before accessing to SPI device.

- **4-line mode:** spidev.ko

```
//Uninstall SPI driver module.
rmmod spidev

//Install 4-line mode SPI driver.
insmod spidev.ko busnum=6 chipselect=0

// install 4-line mode SPI driver with speed parameter (the max speed is 19.2MHz by default).
```

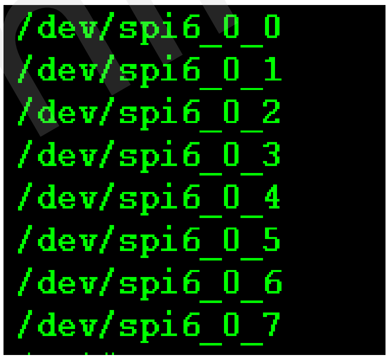
```
insmod spidev.ko busnum=6 chipselect=0 maxspeed=50000000

sleep(1);
```

After the SPI driver module is installed, the SPI device */dev/spidev6.0* will be generated. Developers may simply open this device to send/receive SPI data.

- **6-line mode:** quec_spi_chn.ko

```
//Uninstall SPI driver module.
rmmod quec_spi_chn

//Install 6-line mode SPI driver.
insmod quec_spi_chn.ko busnum=6 chipselect=0 gpiomodemready=78 gpiomcuready=79

//Install 6-line mode SPI driver with speed parameter (the max speed is 19.2MHz by default).
insmod quec_spi_chn.ko busnum=6 chipselect=0 gpiomodemready=78 gpiomcuready=79
maxspeed=50000000

//Install 6-line mode SPI driver with SPI mode and speed parameter.
insmod quec_spi_chn.ko busnum=6 chipselect=0 gpiomodemready=78 gpiomcuready=79 spimode=0
maxspeed=50000000

sleep(1);
```

After the SPI driver module is installed, developers may get 8 SPI devices that indicate 8 SPI channels, which can be used for different business data.



```
/dev/spi6_0_0
/dev/spi6_0_1
/dev/spi6_0_2
/dev/spi6_0_3
/dev/spi6_0_4
/dev/spi6_0_5
/dev/spi6_0_6
/dev/spi6_0_7
```

Figure 4: SPI 8 Channels

Developers may execute the previous commands by system call when initializing SPI in application.

```
system("insmod xxxxx");
sleep(1);
```

4.3.6.3. SPI Parameters

- **SPI customized parameters**

<busnum>	SPI bus number. Fixed to 6.
<chipselect>	0 is active.
<maxspeed>	The default speed is 19.2MHz, and the max speed can be up to 50MHz. The possible values (unit: Hz): {960000, 4800000, 9600000, 16000000, 19200000, 25000000, and 50000000}.
<spimode>	It is decided by SPI device. 0: Mode 0 CPOL=0, CPHA=0 1: Mode 1 CPOL=0, CPHA=1 2: Mode 2 CPOL=1, CPHA=0 3: Mode 3 CPOL=1, CPHA=1
<gpiomodemredy>	A GPIO number which indicates the modem is ready (not in sleep state), and simultaneously wake up the external MCU.
<gpiomcuready>	A GPIO number which indicates the external MCU is ready (not in sleep state), and simultaneously wake up the module.

- **Relevant header files**

```
#include "ql_oe.h"
```

- **API functions**

Developers may directly call the APIs in standard library, such as **open()**, **ioctl()**, to access and control the SPI device */dev/spidev6.0*.

- **Example**

Please see *example/spi/example_spi.c* for details.

4.4. File System

- **Relevant header files**

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
```

- **API functions**

/* Set file access permissions for FILE to MODE. If FILE is a symbolic link, this affects its target instead. */

```
extern int chmod (const char *__file, __mode_t __mode);
```

/* Get file attributes for FILE and put them in BUF. */

```
extern int stat (const char *__restrict __file, struct stat *__restrict __buf);
```

/* Open a file and create a new stream for it. */

```
extern FILE *fopen (const char *__restrict __filename, const char *__restrict __modes);
extern FILE *freopen (const char *__restrict __filename,
                      const char *__restrict __modes,
                      FILE *__restrict __stream) __wur;
```

/* Seek to a certain position on STREAM. */

```
extern int fseek (FILE *__stream, long int __off, int __whence);
```

/* Read chunks of generic data from STREAM. */

```
extern size_t fread (void *__restrict __ptr, size_t __size, size_t __n, FILE *__restrict __stream);
```

/* Read formatted input from STREAM. */

```
extern int fscanf (FILE *__restrict __stream, const char *__restrict __format, ...);
```

/* Write chunks of generic data to STREAM. */

```
extern size_t fwrite (const void *__restrict __ptr, size_t __size,   size_t __n, FILE *__restrict __s);
```

/* Return the current position of STREAM. */

```
extern long int ftell (FILE *__stream);
```

/* Remove file FILENAME. */

```
extern int remove (const char *__filename);
```

/* Rename file OLD to NEW. */

```
extern int rename (const char *__old, const char *__new);
```

/* Close STREAM. */

```
extern int fclose (FILE *__stream);
```

/* Create a new directory named PATH, with permission bits MODE. */

```
extern int mkdir (const char *__path, __mode_t __mode);
```

- **Example**

Please see *example/file/example_file.c* for details.

4.5. SMS

Developers can open the AT port */dev/smd8* to send AT commands to program SMS option. Please refer to ***document [5]*** for the usage of related AT commands.

AG35-QuecOpen module also provides a list of APIs for developers to program SMS. When a specified URC event is received, it will register SMS URC event callback first and then read and parse the SMS.

- **Relevant header files**

```
#include "ql_sms.h"
```

- **API functions**

/* Create an SMS message data structure. Return reference to the new message object.
NOTE
On failure, the process exits, so you do not have to worry about checking the returned reference for validity. */

```
ST_SMS_MsgRef QL_SMS_Create( void );
```

/* Set the timeout to send a SMS message.
NOTE
On failure, the process exits, so you do not have to worry about checking the returned reference for validity. */

```
int QL_SMS_SetTimeout
(
    ST_SMS_MsgRef msgRef,      //< [IN] Reference to the message object.
    uint32_t timeout           //< [IN] Timeout in seconds.
);
```

/* Set the telephone destination number.
NOTE
If telephone destination number is too long (max QL_MDMDEFS_PHONE_NUM_MAX_LEN digits), it is a fatal error. The function will not return.
NOTE
If the caller is passing a bad pointer into this function, it is a fatal error. The function will not return. */

```
int QL_SMS_SetDestination
(
    ST_SMS_MsgRef msgRef,       //< [IN] Reference to the message object.
    const char* dest            //< [IN] Telephone number string.
);
```

/* This function must be called to set the text message content.
NOTE
If message is too long (max QL_SMS_TEXT_MAX_LEN digits), it is a fatal error. The function will not return.
NOTE
If the caller is passing a bad pointer into this function, it is a fatal error. The function will not return. */

```
int QL_SMS_SetText
(
    ST_SMS_MsgRef msgRef,      //< [IN] Reference to the message object.
    const char* text           //< [IN] SMS text.
);
```

/* Set the binary message content.
NOTE
If length of the data is too long (max QL_SMS_BINARY_MAX_BYTES bytes), it is a fatal error. The function will not return.
NOTE
If the caller is passing a bad pointer into this function, it is a fatal error. The function will not return. */

```
int QL_SMS_SetBinary
(
    ST_SMS_MsgRef msgRef,    //< [IN] Reference to the message object.

    const char* binPtr,              //< [IN] Binary data.

    size_t binNumElements       //< [IN]
);
```

/* Set the UCS2 message content (16-bit format).
NOTE
If length of the data is too long (max QL_SMS_UCS2_MAX_CHARS), it is a fatal error. The function will not return.
NOTE
If the caller is passing a bad pointer into this function, it is a fatal error. The function will not return. */

```
int QL_SMS_SetUCS2
(
    ST_SMS_MsgRef msgRef,    //< [IN] Reference to the message object.

    const uint16_t* ucs2Ptr,//< [IN] UCS2 message.
    size_t ucs2NumElements    //< [IN]
);
```

/* Set the PDU message content.
NOTE
If length of the data is too long (max QL_SMS_PDU_MAX_BYTES bytes), it is a fatal error. The function will not return.
NOTE
If the caller is passing a bad pointer into this function, it is a fatal error. The function will not return. */

```
int QL_SMS_SetPDU
(
    ST_SMS_MsgRef msgRef,     //< [IN] Reference to the message object.

    const char* pduPtr,        //< [IN] PDU message.

    size_t pduNumElements     //< [IN]
);
```

/* Send an SMS message.
Verifies first if the parameters are valid, then it checks whether the modem state can support message sending.
@return *QL_FORMAT_ERROR*   Message content is invalid.
@return *E_QL_FAULT*   Function failed to send the message.
@return *E_QL_OK*   Function succeeded.
@return *QL_TIMEOUT*   Timeout before the complete sending.
NOTE
If the caller is passing a bad pointer into this function, it is a fatal error. The function will not return. */

```
int QL_SMS_Send
(
    ST_SMS_MsgRef   msgRef     //< [IN] Reference to the message object.
);
```

/* Create and synchronously send a text message.
@return *QL_SMS_Msg*   Reference to the new Message object pooled.
@return *NULL*   Not possible to pool a new message.
NOTE
If telephone destination number is too long is too long (max QL_MDMDEFS_PHONE_NUM_MAX_LEN digits), it is a fatal error, the function will not return.
NOTE
If message is too long (max QL_SMS_TEXT_MAX_LEN digits), it is a fatal error, the function will not return. */

```
ST_SMS_MsgRef QL_SMS_SendText
(
    const char                          *destStr,     //< [IN] Telephone number string.
    const char                          *textStr,     //< [IN] SMS text.
    QL_SMS_CallbackResultFunc_t handlerPtr, //< [IN]
    void                                 *contextPtr //< [IN]
);
```

/* Delete a message data structure.
It deletes the message data structure and all the allocated memory is freed. If several users own the message object (e.g., several handler functions registered for SMS message reception), the message object will only be deleted if one user owns the message object.
NOTE
If the caller is passing a bad pointer into this function, it is a fatal error. The function will not return. */

```
void QL_SMS_Delete
(
    ST_SMS_MsgRef msgRef     //< [IN] Reference to the message object.
);
```

/* Get the message format.
@return Message format.
NOTE
If the caller is passing a bad pointer into this function, it is a fatal error. The function will not return.
NOTE
For incoming SMS, format returned by QL_SMS_GetFormat is never QL_SMS_FORMAT_PDU. */

```
E_QL_SMS_FORMAT_T QL_SMS_GetFormat
(
    ST_SMS_MsgRef msgRef     //< [IN] Reference to the message object.
);
```

/* Get the message type.
@return Message type.
NOTE
If the caller is passing a bad pointer into this function, it is a fatal error. The function will not return. */

```
E_QL_SMS_TYPE_T QL_SMS_GetType
(
    ST_SMS_MsgRef msgRef     //< [IN] Reference to the message object.
);
```

/* Get the sender telephone number.
Output parameter is updated with the telephone number. If the telephone number string exceeds the value of 'len' parameter, E_QL_OVERFLOW error code is returned and 'tel' is filled until 'len-1' characters and a null-character is implicitly appended at the end of 'tel'.
NOTE
If the caller is passing a bad pointer into this function, it is a fatal error. The function will not return. */

```
int QL_SMS_GetSenderTel
(
      ST_SMS_MsgRef msgRef,       //< [IN] Reference to the message object.

      char* tel,                  //< [OUT] Telephone number string.

      size_t telNumElements       //< [IN]
);
```

/* Get the Service Center Time Stamp string.
Output parameter is updated with the Time Stamp string. If the Time Stamp string exceeds the value of 'len' parameter, an E_QL_OVERFLOW error code is returned and 'timestamp' is filled until 'len-1' characters and a null-character is implicitly appended at the end of 'timestamp'.
NOTE
If the caller is passing a bad pointer into this function, it is a fatal error. The function will not return. */

```
int QL_SMS_GetTimeStamp
(
      ST_SMS_MsgRef msgRef,       //< [IN] Reference to the message object.

      char* timestamp,            //< [OUT] Message time stamp (in text mode).
                                  //<          string format: "yy/MM/dd,hh:mm:ss+/-zz"
                                  //<   (Year/Month/Day,Hour:Min:Seconds+/-TimeZone)

      size_t timestampNumElements    //< [IN]
);
```

/* Get the message length value.
@return Number of characters for text and UCS2 messages, or the length of the data in bytes for raw binary messages.
NOTE
If the caller is passing a bad pointer into this function, it is a fatal error. The function will not return. */

```
size_t QL_SMS_GetUserdataLen
(
      ST_SMS_MsgRef msgRef     //< [IN] Reference to the message object.
);
```

/* Get the text message.
Output parameter is updated with the text string encoded in ASCII format. If the text string exceeds the value of 'len' parameter, E_QL_OVERFLOW error code is returned and 'text' is filled until 'len-1' characters and a null-character is implicitly appended at the end of 'text'.
@return *E_QL_OVERFLOW*   Message length exceeds the maximum length.
@return *E_QL_OK*   Function succeeded.
NOTE

If the caller is passing a bad pointer into this function, it is a fatal error. The function will not return. */

```
int QL_SMS_GetText
(
    ST_SMS_MsgRef msgRef,       //< [IN] Reference to the message object.

    char* text,                 //< [OUT] SMS text.

    size_t textNumElements      //< [IN]
);
```

/* Get the binary message. Output parameters are updated with the binary message content and the length of the raw binary message in bytes. If the binary data exceeds the value of 'len' input parameter, an E_QL_OVERFLOW error code is returned and 'raw' is filled until 'len' bytes.
NOTE
If the caller is passing a bad pointer into this function, it is a fatal error. The function will not return. */

```
int QL_SMS_GetBinary
(
    ST_SMS_MsgRef msgRef,            //< [IN] Reference to the message object.

    char* binPtr,                    //< [OUT] Binary message.

    size_t* binNumElementsPtr        //< [INOUT]
);
```

/* Get the UCS2 Message (16-bit format).
Output parameters are updated with the UCS2 message content and the number of characters. If the UCS2 data exceeds the value of the length input parameter, an *E_QL_OVERFLOW* error code is returned and 'ucs2Ptr' is filled until of the number of characters specified. */

```
int QL_SMS_GetUCS2
(
    ST_SMS_MsgRef msgRef,            //< [IN] Reference to the message object.

    uint16_t* ucs2Ptr,               //< [OUT] UCS2 message.

    size_t* ucs2NumElementsPtr       //< [INOUT]
);
```

/* Get the PDU message.
Output parameters are updated with the PDU message content and the length of the PDU message is in bytes. If the PDU data exceeds the value of 'len' input parameter, an *E_QL_OVERFLOW* error code is returned and 'pdu' is filled until 'len' bytes.
NOTE
If the caller is passing a bad pointer into this function, it is a fatal error. The function will not return. */

```
int QL_SMS_GetPDU
(
    ST_SMS_MsgRef msgRef,           //< [IN] Reference to the message object.

    char* pduPtr,                   //< [OUT] PDU message.

    size_t* pduNumElementsPtr       //< [INOUT]
);
```

/* Get the message Length value.
@return Length of the data in bytes of the PDU message.
NOTE
If the caller is passing a bad pointer into this function, it is a fatal error. The function will not return. */

```
size_t QL_SMS_GetPDULen
(
    ST_SMS_MsgRef msgRef     //< [IN] Reference to the message object.
);
```

/* Delete an SMS message from the storage area.
Verifies first if the parameter is valid, then it checks whether the modem state can support message deleting.
NOTE
If the caller is passing a bad pointer into this function, it is a fatal error. The function will not return. */

```
int QL_SMS_DeleteFromStorage
(
    ST_SMS_MsgRef msgRef     //< [IN] Reference to the message object.
);
```

/* Get the SMS center address.
Output parameter is updated with the SMS service center address. If the telephone number string exceeds the value of 'len' parameter, an E_QL_OVERFLOW error code is returned and 'tel' is filled until 'len-1' characters and a null-character is implicitly appended at the end of 'tel'. */

```
int QL_SMS_GetSmsCenterAddress
(
    ST_SMS_MsgRef      msgRef,        //< [IN] Reference to the message object.
    char* tel,                        //< [OUT] SMS center address number string.
    size_t telNumElements             //< [IN]
);
```

/* Set the SMS center address.
SMS center address number is defined in ITU-T recommendations E.164/E.163. E.164 numbers can have a maximum of fifteen digits and are usually written with a '+' prefix.
NOTE

If the SMS center address number is too long (max QL_MDMDEFS_PHONE_NUM_MAX_LEN digits), it is a fatal error. The function will not return. */

```
int QL_SMS_SetSmsCenterAddress
(
    ST_SMS_MsgRef       msgRef,          //< [IN] Reference to the message object.
    const char* tel                      //< [IN] SMS center address number string.
);
```

/* Set the SMS preferred storage. */

```
int QL_SMS_SetPreferredStorage
(
    ST_SMS_MsgRef                   msgRef,              //< [IN] Reference to the message object.
    E_QL_SMS_STORAGE_TYPE_T e_storage_type     //< [IN] SMS storage type.
);
```

- **Example**

Please see *example/API/api_test_main.c* for details.

4.6. SMS Parser

QuecOpen^{TM} provides a group APIs for converting SMS format between PDU and text.

- **Relevant header files**

```
#include “ql_sms_parser.h”
```

- **API functions**

```
extern int QL_SMS_Decode(const char *s_in_pdu,   sms_info_t   *pt_sms_info);
```

```
extern int QL_SMS_Encode(sms_info_t   *pt_sms_info);
```

- **Example**

Please see *example/API/api_test_main.c* for details.

4.7. Voice Call

Developers can open the AT port */dev/smd8* to send AT commands to program telephony option. Please refer to ***document [5]*** for the usage of related AT commands.

And QuecOpen^{TM} provides another way to program voice call for your reference.

- **Relevant header files**

```
#include “ql_vcall.h”
```

- **API functions**

/* Reference returned by Start function and used by End function. */

```
typedef int ST_VCALL_CallRef;
```

/* SIM identifiers. */

```
typedef enum
{
    E_QL_VCALL_EXTERNAL_SLOT_1,
    E_QL_VCALL_EXTERNAL_SLOT_2,
    E_QL_VCALL_EMBEDDED,
    E_QL_VCALL_REMOTE,
    E_QL_VCALL_ID_MAX
}E_QL_VCALL_ID_T;
```

/* Voice call establishment states. */

```
typedef enum
{
    E_QL_VCALL_EVENT_ALERTING = 0,        //Voice call establishment in progress.
                                          //Far end is now alerting its user (outgoing call).
    E_QL_VCALL_EVENT_CONNECTED = 1,       //Call has been established, and the media is active.
    E_QL_VCALL_EVENT_TERMINATED = 2,      //Call has terminated.
    E_QL_VCALL_EVENT_OFFLINE = 3,         //NO Service available to try to establish a voice call.
    E_QL_VCALL_EVENT_BUSY = 4,            //Remote party (callee) is busy.
    E_QL_VCALL_EVENT_RESOURCE_BUSY = 5, //All local connection resources (lines/channels)
                                              are in use.
    E_QL_VCALL_EVENT_CALL_END_FAILED = 6,           //Call ending failed.
    E_QL_VCALL_EVENT_CALL_ANSWER_FAILED = 7,        //Call answering failed.
    E_QL_VCALL_EVENT_INCOMING = 8                   //Incoming voice call in progress.
}E_QL_VCALL_EVENT_T;
```

/* Voice call termination reason. */

```
typedef enum
{
    E_QL_VCALL_TERM_NETWORK_FAIL = 0,     //Network could not complete the call.
    E_QL_VCALL_TERM_BAD_ADDRESS = 1,      //Remote address could not be resolved.
    E_QL_VCALL_TERM_BUSY = 2,             //Caller is currently busy and cannot take the call.
    E_QL_VCALL_TERM_LOCAL_ENDED = 3,        //Local party ended the call.
    E_QL_VCALL_TERM_REMOTE_ENDED = 4,       //Remote party ended the call.
    E_QL_VCALL_TERM_UNDEFINED = 5           //Undefined reason.
}E_QL_VCALL_TerminationReason_t;
```

/* Reference type used by Add/Remove functions for EVENT 'QL_VCALL_State' */

```
typedef int QL_VCALL_StateHandlerRef_t;
```

```
typedef enum
{
    E_QL_VCALL_DIRECTION_MO = 1,
    E_QL_VCALL_DIRECTION_MT = 2
}E_QL_VCALL_DIRECTION_T;

typedef enum
{
    E_QL_VCALL_STATE_ORIGINATING = 1,       //Origination
    E_QL_VCALL_STATE_INCOMING,              //Incoming
    E_QL_VCALL_STATE_CONVESATION,           //Conversation
    E_QL_VCALL_STATE_CC_IN_PROGRESS,        //Call is originating but waiting for call control to
complete
    E_QL_VCALL_STATE_ALERTING,              //Alerting
    E_QL_VCALL_STATE_HOLD,                  //Hold
    E_QL_VCALL_STATE_WAITING,               //Waiting
    E_QL_VCALL_STATE_DISCONNECTING,        //Disconnecting
    E_QL_VCALL_STATE_END,                   //End
    E_QL_VCALL_STATE_SETUP,                 //MT call is in setup state in 3GPP
}E_QL_VCALL_STATE_T;

typedef enum
{
    E_QL_VCALL_TYPE_VOICE = 0,              //Voice
    E_QL_VCALL_TYPE_VOICE_FORCED,           //Avoid modem call classification
    E_QL_VCALL_TYPE_VOICE_IP,               //Voice over IP
    E_QL_VCALL_TYPE_VT,                     //VideoTelephony call over IP
    E_QL_VCALL_TYPE_VIDEOSHARE ,            //Videoshare
    E_QL_VCALL_TYPE_TEST,                   //Test call
    E_QL_VCALL_TYPE_OTAPA,                  //OTAPA
    E_QL_VCALL_TYPE_STD_OTASP,              //Standard OTASP
    E_QL_VCALL_TYPE_NON_STD_OTASP,          //Nonstandard OTASP
    E_QL_VCALL_TYPE_EMERGENCY,              //Emergency
    E_QL_VCALL_TYPE_SUPS,                   //Supplementary Service
    E_QL_VCALL_TYPE_EMERGENCY_IP,           //Emergency VoIP
    E_QL_VCALL_TYPE_ECALL                   //eCall
}E_QL_VCALL_TYPE_T;

typedef struct
{
    uint8_t                         call_id;
    char                            PhoneNum[20];        //Telephone number string.
    E_QL_VCALL_DIRECTION_T    e_direct;
    E_QL_VCALL_STATE_T          e_state;
    E_QL_VCALL_TYPE_T           e_type;
} vcall_info_t;
```

```
/* Handler for voice call state changes.
  @parameter reference
  Event voice call object reference.
  @parameter identifier
  Identifier of the remote party.
  @parameter event
  Voice call event.
  @parameter contextPtr */
```

```
typedef void (*QL_VCALL_StateHandlerFunc_t)
(
    ST_VCALL_CallRef reference,
    const char*    identifier,
    E_QL_VCALL_STATE_T    event,
    void* contextPtr
);
```

/* Add handler function for EVENT 'QL_VCALL_State'. This event provides information on voice call state changes. */

```
QL_VCALL_StateHandlerRef_t QL_VCALL_AddStateHandler
(
    QL_VCALL_StateHandlerFunc_t handlerPtr,        //< [IN]
    void* contextPtr                                 //< [IN]
);
```

/* Remove handler function for EVENT 'QL_VCALL_State'. */

```
void QL_VCALL_RemoveStateHandler
(
    QL_VCALL_StateHandlerRef_t addHandlerRef    //< [IN]
);
```

```
/* Start a voice call.
  @return
```

- Reference to the voice call (to be used later for releasing the voice call).
- NULL if the voice call could not be processed. */

```
ST_VCALL_CallRef QL_VCALL_Start
(
    E_QL_VCALL_ID_T simId,
    const char* DestinationID        //< [IN] Destination identifier for the voice
);
```

/* Release a voice call.
@return
- *E_QL_OK* if the end of voice call can be processed.
- *E_QL_NOT_FOUND* if the voice call object reference is not found. */

```
int QL_VCALL_End
(
    ST_VCALL_CallRef reference  //< [IN] Voice call object reference to hang-up.
);
```

/* Answer to incoming voice call.
@return
- *E_QL_OK* if the incoming voice call can be answered.
- *E_QL_NOT_FOUND* if the incoming voice call object reference is not found. */

```
int QL_VCALL_Answer
(
    ST_VCALL_CallRef reference        //< [IN] Incoming voice call object reference to answer.
);
```

/* Get the termination reason of a voice call reference.
@return
- *E_QL_OK* if the termination reason is got.
- *E_QL_NOT_FOUND* if the incoming voice call object reference is not found.
- *E_QL_FAULT* if the voice call is not terminated. */

```
int QL_VCALL_GetTerminationReason
(
    ST_VCALL_CallRef reference,        //< [IN] Voice call object reference to read from.

    E_QL_VCALL_TerminationReason_t* reasonPtr  //< [OUT] Termination reason of the voice call.
);
```

- **Example**

Please see *example/API/api_test_main.c* for details.

4.8. Network Service

AG35-QuecOpen module provides a group of APIs to get network register status and signal quality.

- **Relevant header files**

```
#include "ql_nw.h"
```

```
typedef void (* nw_init_cb_func)(void* cb_data);

#define QL_NW_APN_NAME_MAX_LEN 100
#define QL_NW_USER_NAME_MAX_LEN 64
#define QL_NW_PWD_NAME_MAX_LEN 100
#define MAX_NUM_NEIGHBOR_CELLS    10
```

/* Network registration states. */

```
typedef enum
{
    E_QL_WLAN_REG_NOT_REGISTERED = 0,       //Not registered and not currently searching for
                                              new operator.
    E_QL_WLAN_REG_REGISTERED_HOME_NETWORK = 1,          //Registered, home network.
    E_QL_WLAN_REG_NOT_REGISTERED_SEARCHING_NOW = 2, //Not registered but currently
                                                      searching for a new operator.
    E_QL_WLAN_REG_DENIED = 3,                  //Registration was denied, usually because of
                                                invalid access credentials.
    E_QL_WLAN_REG_UNKNOWN    = 4,              //Unknown state.
    E_QL_WLAN_REG_ROAMING      = 5             //Registered to a roaming network.
}E_QL_WLAN_NET_REG_STATE_T;
```

/* Radio Access Technology enum. */

```
typedef enum
{
    E_QL_NW_RAT_GSM                          = 0,
    E_QL_NW_RAT_UTRAN                         = 2,
    E_QL_NW_RAT_GSMW_EGPRS                   = 3,
    E_QL_NW_RAT_UTRANW_HSDPA                 = 4,
    E_QL_NW_RAT_UTRANW_HSUPA                 = 5,
    E_QL_NW_RAT_UTRANW_HSDPA_AND_HSUPA    = 6,
    E_QL_NW_RAT_E_UTRAN                      = 7
}E_QL_NW_RADIO_ACCESS_TYPE_T;

typedef struct
{
    char      rat[16];
    int       mcc;
    int       mnc;
    int       lac;
    int       cid;
}ST_SingleCellInfo;

typedef struct
{
    int                      validCnt;
    ST_SingleCellInfo      cellInfo[MAX_NUM_NEIGHBOR_CELLS];
}ST_CellInfo;
```

/* Handler of network registration state changes.
 @parameter state
   Parameter ready to receive the Network Registration state.
 @parameter contextPtr */

```
typedef void (*QL_NW_RegStateHandlerFunc_t)
(
    E_QL_WWAN_NET_REG_STATE_T     state,
    void                          *contextPtr
);
```

/* Add handler function for EVENT 'le_mrc_NetRegStateEvent
  This event provides information on network registration state changes.
  NOTE
  <b>multi-app safe</b> */

```
extern int QL_NW_AddRegStateEventHandler
(
    QL_NW_RegStateHandlerFunc_t handlerPtr, //< [IN]
    void                        *contextPtr //< [IN]
);
```

/* Get current network information: network name/country code/network code. */

```
extern int QL_NW_GetNetworkNameMccMnc
(
    char        *nameStr,       //[OUT] the home network Name
    size_t      nameLen,        //[IN]
    char        *mccStr,        //[OUT] the mobile country code
    size_t      mccLen,         //[IN]
    char        *mncStr,        //[OUT] the mobile network code
    size_t      mncLen          //[IN]
);
```

/* Get current network register signal information: rat/state/rssi/ber. */

```
extern int QL_NW_GetRegState
(
    E_QL_NW_RADIO_ACCESS_TYPE_T *rat,          //[OUT] The Radio Access Technology.
    E_QL_WWAN_NET_REG_STATE_T    *state,        //[OUT] Network Registration state.
    int                          *rssi,         //[OUT] [OUT] Received signal strength quality.
    int                          *ber           //[OUT] [OUT] Received signal bit error rate.
);
```

/* Get current network location information: rat/mcc/mnc/lac/cid. */

```
int QL_NW_GetServingCell(ST_CellInfo *pt_info);
```

- **Example**

Please see *example/API/api_test_main.c* for details.

4.9. Data Service

QuecOpen^{TM} uses the component "DSI_NetCtrl" to perform data services call. DSI_NetCtrl just controls and manages the data activities and will not provide methods for data transfer explicitly. Developers can adopt BSD sockets or Unix sockets to create data connections.

QuecOpen^{TM} provides a group of wrapper API of dsi_ctrl for data service programming.

- **Relevant header files**

```
#include "ql_oe.h"
```

- **API functions**

```
typedef void (* nw_init_cb_func)(void* cb_data);

/* init dsi_netctrl library, set callback */
typedef void (* dsi_init_cb_func)(void* cb_data);
```

/* Handler of wwan connection state changes.
  @parameter state
  Parameter ready to receive the wwan connection state.
  @parameter contextPtr */

```
typedef void (*QL_WWAN_ConnectionStateHandlerFunc_t)
(
    int        profileIdx, //const char* intfName,
    bool      isConnected,
    void      *contextPtr
);
```

/* Add handler function for EVENT NetRegStateEvent
  This event provides information on network registration state changes.
  NOTE
  <b>multi-app safe</b> */

```
extern int QL_WWAN_AddConnStateEventHandler
(
    QL_WWAN_ConnectionStateHandlerFunc_t      handlerPtr, //< [IN]
    void                                      *contextPtr //< [IN]
);
```

/* Initialize WLAN session, and the result will be callback via cb_func. */

```
extern int QL_WWAN_Initialize(dsi_init_cb_func cb_func, void* cb_data);
```

/* De-initialize WLAN session, release the resource. */

```
extern int QL_WWAN_Deinitialize(void);
```

/* Get data service handle. */

```
dsi_hndl_t QL_WWAN_GetDataSrvHandle(dsi_net_ev_cb cb_func, void * cb_data);
```

/* Release data service handle. */

```
void QL_WWAN_ReleaseDataSrvHandle(dsi_hndl_t hdl);
```

/* Set parameters (such as profile ID, IP type, APN, username, password). */

```
int QL_WWAN_SetProfileId(dsi_hndl_t hdl, int nIndex);
```

/* Set IP Version. */

```
int QL_WWAN_SetIPVersion(dsi_hndl_t hdl, int ip_version);
```

/* Start data call. */

```
int QL_WWAN_StartDataCall(dsi_hndl_t hdl);
```

/* Stop data call. */

```
int QL_WWAN_StopDataCall(dsi_hndl_t hdl);
```

/* Get device name. */

```
int QL_WWAN_GetDeviceName(dsi_hndl_t hdl, char * buf, int len);
```

/* Get IP Addr. */

```
int QL_WWAN_GetIPAddr(/*[in]*/dsi_hndl_t hdl, /*[out]*/dsi_addr_info_t * info_ptr, /*[out]*/int *len);
```

/* Get data bearer technology.*/

```
dsi_data_bearer_tech_t QL_WWAN_GetDataBearerTech(dsi_hndl_t hdl);
```

/* Get data end reason. */

```
int QL_WWAN_GetDataEndReason(dsi_hndl_t hdl, dsi_ce_reason_t *ce_reason, dsi_ip_family_t ipf);
```

/* Get APN related information: APN name/user name/password. */

```
int QL_WWAN_GetAPN
(
    dsi_hndl_t          hdl,            //[IN] DSI handle
    int                 apn_id,         //[IN] id: 0~4, 0 means APN1
    char               *apn,            //[IN] The Access Point Name
    int                 apnLen,
    char               *userName,       //[IN] User name used by authentication
    int                 userLen,
    char               *password,       //[IN] Password used by authentication
    int                 pwdLen
);
```

/* Set APN related information: APN name/user name/password. */

```
int QL_WWAN_SetAPN
(
    dsi_hndl_t          hdl,            //[IN] DSI handle
    int                 apn_id,         //[IN] id: 0~4, 0 means APN1
    const char          *apn,           //[IN] The Access Point Name
    const char          *userName,      //[IN] User name used by authentication
    const char          *password       //[IN] Password used by authentication
);
```

/* Set authorization preference. */

```
int QL_WWAN_SetAuthPref(dsi_hndl_t hdl, dsi_auth_pref_t auth);
```

/* Simple DNS Reslove.
  @parameter host
  A domain name that needs to be resolved.

```
@parameter dns_server_ip
DNS server IP address
@parameter ip_type
ip type (QUERY_IPV4_E or QUERY_IPV6_E)
@parameter resolved_addr
The resolved results */
```

```
void QL_nslookup(char *host, char *dns_server_ip, QUERY_IP_TYPE ip_type, hostaddr_info_u
*resolved_addr);
```

- **Example**

Please see *example/API/api_test_main.c* for details.

- **User Cases**

Case 1: Multi-APN Activation

Step 1: Open the first APN path using the default one (APN1).
1) QL_WWAN_Initialize
2) QL_WWAN_GetDataSrvHandle
3) QL_WWAN_SetProfileId, QL_WWAN_SetIPVersion
4) QL_WWAN_StartDataCall
5) Verify the network access using wget/ping

Step 2: Open the second one.
1) Choose case “set_selected_apn_idx” and input 2
2) QL_WWAN_GetDataSrvHandle
3) QL_WWAN_SetProfileId, QL_WWAN_SetIPVersion
4) QL_WWAN_StartDataCall
5) Verify the network access using wget/ping

Now the two APNs are activated. More APNs can be activated with similar processes as **step 2**.

Case 2: Work as Server using TCP

Step 1: QL_WWAN_Initialize
Step 2: QL_WWAN_GetDataSrvHandle
Step 3: QL_WWAN_SetProfileId, QL_WWAN_SetIPVersion
Step 4: QL_WWAN_StartDataCall
Step 5: "testapi_server(work as server, receive msg and ack msg+ack)"

Case 3: Work as Client using TCP

Step 1: QL_WWAN_Initialize
Step 2: QL_WWAN_GetDataSrvHandle
Step 3: QL_WWAN_SetProfileId, QL_WWAN_SetIPVersion

Step 4: QL_WWAN_StartDataCall
Step 5: "testapi_client(work as client, send msg to server)"

Case 4: Work as Server using UDP

Step 1: QL_WWAN_Initialize
Step 2: QL_WWAN_GetDataSrvHandle
Step 3: QL_WWAN_SetProfileId, QL_WWAN_SetIPVersion
Step 4: QL_WWAN_StartDataCall
Step 5: "test_udp_server"

Case 5: Work as Client using UDP

Step 1: QL_WWAN_Initialize
Step 2: QL_WWAN_GetDataSrvHandle
Step 3: QL_WWAN_SetProfileId, QL_WWAN_SetIPVersion
Step 4: QL_WWAN_StartDataCall
Step 5: "test_udp_client"

The following user case can run in different modules. You can also run it in one module, through using one APN as the server and the other APN as the client.

Case 6: One Module, Using two APNs, Runs as both Server and Client

Step 1: Start the server using APN1
1) QL_WWAN_Initialize
2) QL_WWAN_GetDataSrvHandle
3) QL_WWAN_SetProfileId, QL_WWAN_SetIPVersion
4) QL_WWAN_StartDataCall
5) "testapi_server(work as server, receive msg and ack msg+ack)"

Step 2: Start the client using APN2
1) Choose case “set_selected_apn_idx” and input 2;
2) QL_WWAN_GetDataSrvHandle
3) QL_WWAN_SetProfileId, QL_WWAN_SetIPVersion
4) QL_WWAN_StartDataCall
5) "testapi_client(work as client, send msg to server)" (Here need input server’s IP address)

Case 7: Worked as TCP Client, Send Heart Beat to the Server under Sleep Mode

Step 1: QL_WWAN_Initialize
Step 2: QL_WWAN_GetDataSrvHandle
Step 3: QL_WWAN_SetProfileId, QL_WWAN_SetIPVersion
Step 4: QL_WWAN_StartDataCall
Step 5: "testapi_client(work as client, send msg to server)"
Step 6: "auto sleep onoff", set autosleep on.

4.10. GNSS

The GNSS chip is built in the module. Developers just need to enable the GNSS option to retrieve NMEA statements.

The GNSS NMEA can be outputted through USB-NMEA port, Debug UART port and Linux SMD7 port. In QuecOpen^{TM} application, developers may get NMEA by opening */dev/smd7*.

Please follow the three steps below to get GNSS NMEA:

Step 1: Open */dev/smd7*.

Step 2: Set the output port of GNSS NMEA to SMD port.
Send **AT+QGPSCFG=”outport”,”linuxsmd”** though AT port (*/dev/smd8*).

Step 3: Enable GNSS.
Send **AT+QGPS=1** to enable GNSS option though AT port (*/dev/smd8*).

- **Example**

Please see *example/gnss/example_gps.c* for details.

4.11. (U)SIM

AG35-QuecOpen module provides a group of APIs to get (U)SIM card information.

- **Relevant header files**

```
#include "ql_sim.h"
```

- **API functions**

```
/* Get the (U)SIM card ICCID string.
    -simId: (U)SIM ID
    -iccid: input buffer
    -iccidLen: input buffer length */
```

```
extern int   QL_SIM_GetICCID
(
    E_QL_SIM_ID_T     simId,        //< [IN] The SIM identifier.
    char*             iccid,        //< [OUT] ICCID
    size_t            iccidLen      //< [IN]
);
```

/* Get the (U)SIM card IMSI string.
  -simId: (U)SIM ID
  -imsi: input buffer
  -imsiLen: input buffer length */

```
extern int   QL_SIM_GetIMSI
(
      E_QL_SIM_ID_T     simId,          //< [IN] The SIM identifier.
      char*                  imsi,           //< [OUT] IMSI
      size_t                imsiLen        //< [IN]
);
```

/* This function been called to verify the PIN code.
  @return *E_QL_BAD_PARAMETER* The parameters are invalid.
  @return *E_QL_NOT_FOUND*   The function failed to select the (U)SIM card for this operation.
  @return *QL_UNDERFLOW*   The PIN code is not long enough (min 4 digits).
  @return *E_QL_FAULT*   The function failed to enter the PIN code.
  @return *E_QL_OK*   The function succeeded.
  NOTE
  If PIN code is too long (max 8 digits), it is a fatal error, the function will not return.
  NOTE
  If the caller is passing a bad pointer into this function, it is a fatal error, the function will not return. */

```
int   QL_SIM_VerifyPIN
(
      E_QL_SIM_ID_T     simId,   //< [IN] The SIM identifier.
      const char          *pin       //< [IN] The PIN code.
);
```

/* This function been called to change the PIN code.
 NOTE
 If PIN code is too long (max 8 digits), it is a fatal error, the function will not return.
 NOTE
 If the caller is passing a bad pointer into this function, it is a fatal error, thefunction will not return. */

```
int   QL_SIM_ChangePIN
(
      E_QL_SIM_ID_T     simId,          //< [IN] The SIM identifier.
      const char          *old_pin,     //< [IN] The old PIN code.
      const char          *new_pin       //< [IN] The new PIN code.
);
```

/* Unlock the (U)SIM card: it will disables the request of PIN code.
  NOTE
  If PIN code is too long (max 8 digits), it is a fatal error, the function will not return. *
  NOTE

    If the caller is passing a bad pointer into this function, it is a fatal error, the function will not return.*/

```
int   QL_SIM_Unlock
(
      E_QL_SIM_ID_T     simId,          //< [IN] The SIM identifier.
      const char          *pin              //< [IN] The PIN code.
);
```

```
/* Lock the (U)SIM card: it will enables the request of PIN code.
   NOTE
   If PIN code is too long (max 8 digits), it is a fatal error, the function will not return.
   NOTE
   If the caller is passing a bad pointer into this function, it is a fatal error, the function will not return. */
```

```
int   QL_SIM_Lock
(
      E_QL_SIM_ID_T     simId,    //< [IN] The SIM identifier.
      const char          *pin       //< [IN] The PIN code.
);
```

```
/* Unblock the (U)SIM card: it will use PUK to unblock the (U)SIM card and set new PIN.
   NOTE
   If PIN code is too long (max 8 digits), it is a fatal error, the function will not return.
   NOTE
   If the caller is passing a bad pointer into this function, it is a fatal error, the function will not return. */
```

```
int   QL_SIM_Unblock
(
      E_QL_SIM_ID_T     simId,          //< [IN] The SIM identifier.
      const char          *puk,            //< [IN] The PIN code.
      const char          *newpin         //< [IN] The PIN code.
);
```

```
/* Get the (U)SIM card PIN retry times.
   -simId: (U)SIM ID */
```

```
extern int   QL_SIM_GetPINTriesCnt
(
       E_QL_SIM_ID_T simId       //< [IN] The SIM identifier.
);
```

```
/* Get the (U)SIM card state.
   -simId: (U)SIM ID */
```

```
extern E_QL_SIM_STATES_T   QL_SIM_GetState
(
      E_QL_SIM_ID_T simId       //< [IN] The SIM identifier.
);
```

- **Example**

Please see *example/API/api_test_main.c* for details.

4.12. Basic Device Information

AG35-QuecOpen module provides a group of APIs to get basic device information.

- **Relevant header files**

```
#include “ql_dev.h”
```

- **API functions**

/* Get the IMEI string.
  -imei: input buffer
  -imeiLen: input buffer length */

```
extern int QL_DEV_GetImei(char* imei, size_t imeiLen);
```

/* Get the FirmwareVersion string.
  -version: input buffer
  -versionLen: input buffer length */

```
extern int QL_DEV_GetFirmwareVersion(char* version, size_t versionLen);
```

/* Get the DeviceModel string.
  -model: input buffer
-modelLen: input buffer length */

```
extern int QL_DEV_GetDeviceModel(char* model, size_t modelLen);
```

/* Get the MEID string.
  -meid: input buffer
  -meidLen: input buffer length */

```
extern int QL_DEV_GetMeid(char* meid, size_t meidLen);
```

/* Get the ESN string.
  -esn: input buffer
  -esnLen: input buffer length */

```
extern int QL_DEV_GetEsn(char* esn, size_t esnLen);
```

- **Example**

Please see *example/API/api_test_main.c* for details.

4.13. QMI Timer

AG35-QuecOpen module provides a group of APIs to register timer into modem side, so that modem can wakeup Linux when Linux is suspended.

- **Relevant header files**

```
#include “ql_timer.h”
```

- **API functions**

```
/* Function: QL_Timer_Register
   Description: Register timer
   @timer_id: Timer_ID will be registered
   @exp_cb_fcn: Callback function to call at timer expiry
   @cb_params: Parameters for exp_cb_fcn
   Return:
   RES_OK     success
   Others     fail */
```

```
int QL_Timer_Register(ql_timer_id timer_id, ql_timer_exp_cb   exp_cb_fcn, void *cb_params);
```

```
/* Function: QL_Timer_Start
   Description: Start timer
   @timer_id: Timer ID will be start
   @interval:  timer in ms will be expired
   @auto_repeat:
   0:   not periodic
   1:   periodic
   Return:
   RES_OK         success
   Others         fail */
```

```
int QL_Timer_Start(ql_timer_id timer_id, int interval, int auto_repeat);
```

```
/* Function: QL_Timer_Stop
   Description: Stop timer
   @timer_id: Timer ID will be stop
   Return:
   RES_OK         success
   Others         fail */
```

```
int QL_Timer_Stop(ql_timer_id timer_id);
```

- **Example**

Please see *example/API/api_test_main.c* for details.

4.14. Low Power Consumption

AG35-QuecOpen module provides a way for power saving. If the WakeLock is inactive, and sleep mode is set, AG35-QuecOpen module will enter into sleep mode. When QMI timer has been set up, AG35-QuecOpen module can be woken up periodically.

When a call/SMS/data packet is received, or GPIO pull up/down, it will be woken up.

AG35-QuecOpen module provides a group of APIs to control this mode.

- **Relevant header files**

```
#include “ql_oe.h”
#include “ql_sleep_wakelock.h”
```

- **API functions**

/* Create WakeLock, return the file description of the WakeLock. */

```
extern int Ql_SLP_WakeLock_Create(const char *name, size_t len);
```

/* Lock the WakeLock by the file description of the WakeLock. */

```
extern int Ql_SLP_WakeLock_Lock(int fd);
```

/* Unlock the WakeLock by the file description of the WakeLock. */

```
extern int Ql_SLP_WakeLock_Unlock(int fd);
```

/* Destroy the WakeLock by the file description of the WakeLock. */

```
extern int Ql_SLP_WakeLock_Destroy(int fd);
```

/* Enable/Disable autosleep function. */

```
extern int Ql_Autosleep_Enable(char enable);
```

- **Example**

Please see *example/sleep_wakelock/example_sleep_wakelock.c* for details.

5 Appendix A References

Table 8: Related Documents

SN	Document Name	Remark
[1]	Quectel_WCDMA&LTE_Linux_USB_Driver_User_Guide	USB driver installation guide for UMTS/HSPA/LTE modules
[2]	Quectel_AG35-QuecOpen_Hardware_Design	Hardware design for AG35-QuecOpen modules
[3]	Quectel_AG35-QuecOpen_GPIO_Assignment_ Speadsheet	All multiplexing pins available in AG35-QuecOpen modules
[4]	Quectel_AG35-QuecOpen_SDK_Development_Guide	Compiling and developing with AG35 SDK
[5]	Quectel_AG35_AT_Commands_Manual	AT commands manual for AG35 module
[6]	Qualcomm_DSI_NetCtrl_Library_API_Interface_ Specification	API library for network management