# EC2x&EG9x&EG25-G Series

# QuecOpen Network Information API Reference Manual

**LTE Standard Module Series**

Version: 1.0

Date: 2021-02-19

Status: Released

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.**
Building 5, Shanghai Business Park Phase III (Area B), No.1016 Tianlin Road, Minhang District, Shanghai 200233, China
Tel: +86 21 5108 6236
Email: info@quectel.com

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

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

## General Notes

Quectel offers the information as a service to its customers. The information provided is based upon customers' requirements. Quectel makes every effort to ensure the quality of the information it makes available. Quectel does not make any warranty as to the information contained herein, and does not accept any liability for any injury, loss or damage of any kind incurred by use of or reliance upon the information. All information supplied herein is subject to change without prior notice.

## Disclaimer

While Quectel has made efforts to ensure that the functions and features under development are free from errors, it is possible that these functions and features could contain errors, inaccuracies and omissions. Unless otherwise provided by valid agreement, Quectel makes no warranties of any kind, implied or express, with respect to the use of features and functions under development. To the maximum extent permitted by law, Quectel excludes all liability for any loss or damage suffered in connection with the use of the functions and features under development, regardless of whether such loss or damage may have been foreseeable.

## Duty of Confidentiality

The Receiving Party shall keep confidential all documentation and information provided by Quectel, except when the specific permission has been granted by Quectel. The Receiving Party shall not access or use Quectel's documentation and information for any purpose except as expressly provided herein. Furthermore, the Receiving Party shall not disclose any of the Quectel's documentation and information to any third party without the prior written consent by Quectel. For any noncompliance to the above requirements, unauthorized use, or other illegal or malicious use of the documentation and information, Quectel will reserve the right to take legal action.

## Copyright

The information contained here is proprietary technical information of Quectel. Transmitting, reproducing, disseminating and editing this document as well as using the content without permission are forbidden. Offenders will be held liable for payment of damages. All rights are reserved in the event of a patent grant or registration of a utility model or design.

# About the Document

## Revision History

| Version | Date | Author | Description |
|---|---|---|---|
| - | 2021-01-08 | Tinker SUN | Creation of the document |
| 1.0 | 2021-02-19 | Tinker SUN | First official release |

# Contents

# Table Index

# 1 Introduction

This document introduces the network information API functions supported by Quectel EC2x series, EG9x series and EG25-G modules in QuecOpen® solution. The functions are used for setting the module's network registration parameters, and for obtaining the network registration related information.

Through network information API functions, you can achieve functions listed below.

1. Initialize or deinitialize network registration service
2. Set or get network registration information, including preferred network mode, roaming notification status and network registration status information.
3. Get the network time
4. Set the low power mode
5. Get the signal strength
6. Scan the network
7. Register network events and register the callback function for handling the registered events

NOTE

QuecOpen® is an open-source embedded development platform based on Linux system. It is intended to simplify the design and development of IoT applications. For more information on QuecOpen® solution of EC2x series, EG9x series and EG25-G modules, see ***document [1]***.

## 1.1. Applicable Modules

**Table 1: Applicable Modules**

| Module Series | Module |
|---|---|
| EC2x series | EC25 series |
| | EC21 series |
| | EC20 R2.1 |
| EG9x series | EG95 series |

| | |
|---|---|
| | EG91 series |
| EG25-G | EG25-G |

# 2 Network Registration Related Features

## 2.1. Low Power Mode

In QuecOpen solution, EC2x series, EG9x series and EG25-G modules support sleep wakeup feature. With this feature, the module can enter sleep mode to save power consumption when there is no task to be handled; when there is any new task, the module first wakes up from sleep mode to handle the task and then re-enters sleep mode.

During the sleep mode, the module can be woken up by various events, including SMS, calls, timer interrupts, and GPIO interrupts. In certain application scenarios, you may not care about a specific wakeup event, so you do not want the wakeup event to wake up the module. For instance, the module is woken up by the signal strength event [1)] every 2–3 seconds, while waking up the module in such a frequency greatly influences the power consumption and is not desired in some power-sensitive applications. Therefore, EC2x series, EG9x series and EG25-G QuecOpen modules provide an API function (see ***Chapter 3.4.10***) to set the module to low power mode in which the module will not be woken up by signal strength events.

**NOTE**

[1)] The module reports the signal strength event automatically even when you did not register the event with *QL_MCM_NW_EventRegister()*. If you intend to disable the reporting of the event, you have set the module into low power mode. See ***Chapter 3.4.10*** for details.

## 2.2. Signal Bar for Signal Strength Indication

The signal strength of the module is indicated by parameters such as *rssi*, *rsrp*, *ecio* and *sinr* (see ***Chapter 3.4.11***). These parameters are reported through the Signal Strength Event directly; you can also query the parameter values with *QL_MCM_NW_GetSignalStrength()*. Neither of the ways enables you to intuitively know the signal strength, and the parameters may be different in different network modes. Indicating the signal strength in signal bars is a great solution to allow users intuitively know the signal quality.

EC2x series, EG9x series and EG25-G QuecOpen modules support indicating signal strength in signal bars. Based on the signal strength measurement algorithm defined by Android system, the module converts the signal strength parameter values for different network modes into signal bars, through which you can know the relative signal strength through the number of bars.

There are five signal strength levels (NONE, POOR, MODERATE, GOOD, GREAT) that correspond to 1, 2, 3, 4 and 5 signal bars respectively.

### 2.2.1. Signal Strength Indication in CDMA

In CDMA, the module first acquires signal strength parameters *rssi* and *ecio*, and then converts the parameter values into *rssi_level* and *ecio_level*, respectively.

*rssi_level* and *ecio_level* represent the number of signal bar, ranging from 1 to 5.

- **Criteria for converting *rssi* into *rssi_level***

| | |
|---|---|
| *rssi* < -100 | *rssi_level* = 1 |
| -100 ≤ *rssi* < -95 | *rssi_level* = 2 |
| -95 ≤ *rssi* < -85 | *rssi_level* = 3 |
| -85 ≤ *rssi* < -75 | *rssi_level* = 4 |
| *rssi* ≥ -75 | *rssi_level* = 5 |

- **Criteria for converting *ecio* into *ecio_level***

| | |
|---|---|
| *ecio* < -150 | *ecio_level* = 1 |
| -150 ≤ *ecio* < -130 | *ecio_level* = 2 |
| -130 ≤ *ecio* < -110 | *ecio_level* = 3 |
| -110 ≤ *ecio* < -90 | *ecio_level* = 4 |
| *ecio* ≥ -90 | *ecio_level* = 5 |

The module finally indicates the signal strength level in *rssi_level* or *ecio_level*, whichever is smaller.

### 2.2.2. Signal Strength Indication in HDR

In HDR, the module first acquires signal strength parameters *rssi* and *sinr*, and then converts the parameter values into *rssi_level* and *sinr_level*, respectively.

*rssi_level* and *sinr_level* represent the number of signal bar, ranging from 1 to 5.

- **Criteria for converting *rssi* into *rssi_level***

| | |
|---|---|
| *rssi* < -105 | *rssi_level* = 1 |
| -105 ≤ *rssi* < -90 | *rssi_level* = 2 |
| -90 ≤ *rssi* < -75 | *rssi_level* = 3 |

| | |
|---|---|
| -75 ≤ *rssi* < -65 | *rssi_level* = 4 |
| *rssi* ≥ -65 | *rssi_level* = 5 |

- **Criteria for converting *sinr* into *sinr_level***

| | |
|---|---|
| *sinr* < 1 | *sinr_level* = 1 |
| 1 ≤ *sinr* < 3 | *sinr_level* = 2 |
| 3 ≤ *sinr* < 5 | *sinr_level* = 3 |
| 5 ≤ *sinr* < 7 | *sinr_level* = 4 |
| *sinr* ≥ 7 | *sinr_level* = 5 |

The module finally indicates the signal strength level in *rssi_level* or *sinr_level*, whichever is smaller.

### 2.2.3. Signal Strength Indication in LTE

In LTE, the module first acquires signal strength parameters *rsrp* and *rssi*, and then converts *rsrp* into *rsrp_level*. If *rsrp_level* ranges between 1 and 5, then the module will indicate the signal strength level in *rsrp_level* directly. Otherwise, it indicates the signal strength level in *asu_level* (converted from *rssi*).

*rsrp_level* and *asu_level* represents the number of signal bar.

- **Criteria for converting *rsrp* into *rsrp_level***

| | |
|---|---|
| *rsrp* < -115 | *rsrp_level* = 1 |
| -115 ≤ *rsrp* < -105 | *rsrp_level* = 2 |
| -105 ≤ *rsrp* < -95 | *rsrp_level* = 3 |
| -95 ≤ *rsrp* < -85 | *rsrp_level* = 4 |
| -85 ≤ *rsrp* < -44 | *rsrp_level* = 5 |
| *rsrp* ≥ -44 | *rsrp_level* = 0 |

- **Criteria for converting *rssi* into ASU and then *asu_level***

1. The formula for converting *rssi into* ASU:
   ASU = (*rssi* + 113) / 2

2. The criteria for converting ASU into *asu_level*

| | |
|---|---|
| 0 ≤ ASU < 5 | *asu_level* = 2 |
| 5 ≤ ASU < 8 | *asu_level* = 3 |
| 8 ≤ ASU < 12 | *asu_level* = 4 |
| 12 ≤ ASU ≤ 63 | *asu_level* = 5 |
| ASU > 63 | *asu_level* = 1 |

### 2.2.4. Signal Strength Indication in Other Network Modes

In other network modes such as CDMA2000, WCDMA, TD-SCDMA and GSM, the module indicates the signal strength level in *asu_level*. It converts the signal strength parameter *rssi* into ASU first and then converts ASU into the number of signal bar *asu_level*.

- **Criteria for converting *rssi* into ASU and then *asu_level***

1. The formula for converting *rssi into* ASU:
   ASU = (*rssi* + 113) / 2

2. The criteria for converting ASU into *asu_level*

| | |
|---|---|
| ASU ≤ 2 \|\| ASU == 99 | *asu_level* = 1 |
| 2 < ASU < 5 | *asu_level* = 2 |
| 5 ≤ ASU < 8 | *asu_level* = 3 |
| 8 ≤ ASU < 12 | *asu_level* = 4 |
| ASU ≥ 12 (and unequal to 99) | *asu_level* = 5 |

# 3 Network Information APIs

## 3.1. Header File Location

The header file *ql_mcm_nw.h* is located in the following directory of QuecOpen SDK:
*ql-ol-sdk/ql-ol-crosstool/sysroots/armv7a-vfp-neon-oe-linux-gnueabi/usr/include/quectel-openlinux-sdk*

Unless otherwise specified, the header files mentioned in this document are all located in this directory.

## 3.2. Example Location

The use examples, which demonstrate how the network information API is best used, are located in the QuecOpen SDK directory of *ql-ol-sdk/ql-ol-extsdk/example/test_mcm_api/test_nw.c*.

## 3.3. Overview of API Functions

**Table 2: Overview of API Functions**

| Function | Description |
|---|---|
| *QL_MCM_NW_Client_Init()* | Initializes the network registration service |
| *QL_MCM_NW_Client_Deinit()* | Deinitializes the network registration service |
| *QL_MCM_NW_SetConfig()* | Sets the preferred network mode and roaming notification status |
| *QL_MCM_NW_GetConfig()* | Gets the current setting for preferred network mode and roaming notification status |
| *QL_MCM_NW_GetNitzTimeInfo()* | Gets the network time |
| *QL_MCM_NW_EventRegister()* | Registers network events |

| | |
|---|---|
| *QL_MCM_NW_GetOperatorName()* | Gets the information of a mobile network operator |
| *QL_MCM_NW_PerformScan()* | Triggers a network scan |
| *QL_MCM_NW_GetRegStatus()* | Gets the information about network registration status |
| *QL_MCM_NW_SetLowPowerMode()* | Sets whether to enable the low power mode |
| *QL_MCM_NW_GetSignalStrength()* | Gets the signal strength information |
| *QL_MCM_NW_GetCellAccessState()* | Gets the cell access state |
| *QL_MCM_NW_AddRxMsgHandler()* | Sets the callback function for network events |

NOTE

Unless otherwise specified, all above API functions do not support concurrent calls, and do not call them in any callback function.

## 3.4. Description of API Functions

### 3.4.1. QL_MCM_NW_Client_Init

This function initializes the network registration service.

- **Prototype**

```
E_QL_ERROR_CODE_T QL_MCM_NW_Client_Init(nw_client_handle_type *ph_nw);
```

- **Parameter**

*ph_nw*:
[Out] Network registration service handle.

- **Return Value**

| | |
|---|---|
| *E_QL_SUCCESS* | Initialized the service successfully. |
| Other values | Failed to initialize the service. See *ql_mcm.h* for the error code. |

NOTE

This function must be called prior to any other network information API function.

### 3.4.2. QL_MCM_NW_Client_Deinit

This function deinitializes the network registration service.

- **Prototype**

```
E_QL_ERROR_CODE_T QL_MCM_NW_Client_Deinit(nw_client_handle_type ph_nw);
```

- **Parameter**

*ph_nw*:
[In] Network registration service handle returned by *QL_MCM_NW_Client_Init().*

- **Return Value**

| | |
|---|---|
| *E_QL_SUCCESS* | Deinitialized the service successfully. |
| Other values | Failed to deinitialize the service. See *ql_mcm.h* for the error code. |

### 3.4.3. QL_MCM_NW_SetConfig

This function sets the preferred network mode and roaming notification status while registering to the network.

- **Prototype**

```
E_QL_ERROR_CODE_T QL_MCM_NW_SetConfig(nw_client_handle_type h_nw, QL_MCM_NW_CO
NFIG_INFO_T *pt_info );
```

- **Parameter**

*h_nw*:
[In] Network registration service handle returned by *QL_MCM_NW_Client_Init().*

*pt_info*:
[In] Preferred network mode and roaming notification status. See ***Chapter 3.4.3.1*** for details.

- **Return Value**

| | |
|---|---|
| *E_QL_SUCCESS* | Set the preferred network mode and roaming notifications successfully. |
| Other values | Failed to set the preferred network mode and roaming notifications. See *ql_mcm.h* for the error code. |

NOTES

1. The settings of this function are saved after power off.
2. If the network mode set with this function is not available, the module will search and register on

another network. When roaming is enabled, the module priorities the networks that support roaming.

#### 3.4.3.1. QL_MCM_NW_CONFIG_INFO_T

The preferred network modes and roaming notification status are defined as follows:

```
typedef struct
{
    uint64_t preferred_nw_mode;
    E_QL_MCM_NW_ROAM_STATE_TYPE_T roaming_pref;
}QL_MCM_NW_CONFIG_INFO_T;
```

- **Parameter**

| Type | Parameter | Description |
|---|---|---|
| uint64_t | *preferred_nw_mode* | Preferred network mode.<br>See ***Chapter 3.4.3.2*** for details. |
| *E_QL_MCM_NW_ROAM_STATE_TYPE_T* | *roaming_pref* | Roaming notification setting.<br>See ***Chapter 3.4.3.3*** for details. |

#### 3.4.3.2. Preferred Network Mode Definition

```
#define QL_MCM_NW_MODE_NONE        0x00
#define QL_MCM_NW_MODE_GSM         0x01
#define QL_MCM_NW_MODE_WCDMA       0x02
#define QL_MCM_NW_MODE_CDMA        0x04
#define QL_MCM_NW_MODE_EVDO        0x08
#define QL_MCM_NW_MODE_LTE         0x10
#define QL_MCM_NW_MODE_TDSCDMA     0x20
#define QL_MCM_NW_MODE_PRL         0x10000
```

- **Parameter**

| Parameter | Description |
|---|---|
| *QL_MCM_NW_MODE_NONE* | No preferred network mode. |
| *QL_MCM_NW_MODE_GSM* | Set GSM as the preferred network mode. |
| *QL_MCM_NW_MODE_WCDMA* | Set WCDMA as the preferred network mode. |

| | |
|---|---|
| *QL_MCM_NW_MODE_CDMA* | Set CDMA as the preferred network mode. |
| *QL_MCM_NW_MODE_EVDO* | Set EVDO as the preferred network mode. |
| *QL_MCM_NW_MODE_LTE* | Set LTE as the preferred network mode. |
| *QL_MCM_NW_MODE_TDSCDMA* | Set TD-SCDMA as the preferred network mode. |
| *QL_MCM_NW_MODE_PRL* | Set the PRL networks in the (U)SIM card as the preferred network. |

**NOTES**

1. You can set multiple preferred network modes.
2. PRL stands for Preferred Roaming List and is a database used in a wireless device. It is built and provided by your wireless carrier, and used when your device is connecting to the carrier's network. It indicates which radio bands, sub-bands, and service provider IDs will be scanned and in what priority order. Without a correct and valid PRL, your device will not be able to roam outside your home network, and may not be able to connect at all inside the network. The database consists of an Acquisition Table, which lists which radio frequencies to search for in which areas, and a System Table, which tells the device which networks it is allowed to connect to, and the preferred order.

### 3.4.3.3. Roaming Status and Roaming Notification Status Definition

The roaming status and roaming notification status are defined as follows:

```
typedef enum
{
    E_QL_MCM_NW_ROAM_STATE_OFF   = 0,
    E_QL_MCM_NW_ROAM_STATE_ON    = 1
}E_QL_MCM_NW_ROAM_STATE_TYPE_T;
```

- **Parameter**

| **Parameter** | **Description** |
|---|---|
| *E_QL_MCM_NW_ROAM_STATE_OFF* | Disabled |
| *E_QL_MCM_NW_ROAM_STATE_ON* | Enabled |

### 3.4.4. QL_MCM_NW_GetConfig

This function gets the current setting for preferred network mode and roaming notification status.

- **Prototype**

```
E_QL_ERROR_CODE_T QL_MCM_NW_GetConfig(nw_client_handle_type h_nw, QL_MCM_NW_CONFIG_INFO_T *pt_info );
```

- **Parameter**

*h_nw*:
[In] Network registration service handle returned by *QL_MCM_NW_Client_Init().*

*pt_info*:
[In] Preferred network mode and roaming notification status. See ***Chapter 3.4.3.1*** for details.

- **Return Value**

| | |
|---|---|
| *E_QL_SUCCESS* | Got the current setting successfully. |
| Other values | Failed to get the current setting. See *ql_mcm.h* for the error code. |

### 3.4.5. QL_MCM_NW_GetNitzTimeInfo

This function gets the network time.

- **Prototype**

```
E_QL_ERROR_CODE_T QL_MCM_NW_GetNitzTimeInfo(nw_client_handle_type h_nw, QL_MCM_NW_NITZ_TIME_INFO_T *pt_info);
```

- **Parameter**

*h_nw*:
[In] Network registration service handle returned by *QL_MCM_NW_Client_Init().*

*pt_info*:
[Out] Network time information. See ***Chapter 3.4.5.1*** for details.

- **Return Value**

| | |
|---|---|
| *E_QL_SUCCESS* | Got the network time successfully. |
| Other values | Failed to get the network time. See *ql_mcm.h* for the error code. |

#### 3.4.5.1. QL_MCM_NW_NITZ_TIME_INFO_T

The network time information is defined as follows:

```
typedef struct
{
    char          nitz_time[QL_MCM_NW_NITZ_BUF_LEN + 1];
    uint64_t      abs_time;
    int8_t        leap_sec;
}QL_MCM_NW_NITZ_TIME_INFO_T;
```

- **Parameter**

| Type | Parameter | Description |
|---|---|---|
| char | *nitz_time* | UTC time in the format of: YY/MM/DD,HH:MM:SS+/-TZ |
| uint64_t | *abs_time* | Absolute time, relative to 00:00:00 on January 1, 1970 (UTC) |
| int8_t | *leap_sec* | Leap second (time error adjustment threshold) |

### 3.4.6. QL_MCM_NW_EventRegister

This function registers network events such as the voice-dialing registration event, data-dialing registration event, signal strength event, cell access state change event and the network time update event. The first two events are commonly registered ones.

- **Prototype**

```
E_QL_ERROR_CODE_T QL_MCM_NW_EventRegister(nw_client_handle_type h_nw, uint32_t bit_ma
sk);
```

- **Parameter**

*h_nw*:
[In] Network registration service handle returned by *QL_MCM_NW_Client_Init().*

*bit_mask*:
[In] Network event to be registered. See ***Chapter 3.4.6.1*** for details.

- **Return Value**

| | |
|---|---|
| *E_QL_SUCCESS* | Registered the network event successfully. |
| Other values | Failed to register the network event. See *ql_mcm.h* for the error code. |

#### 3.4.6.1. Network Event Definition

The network events are defined as follows:

```
#define     NW_IND_VOICE_REG_EVENT_IND_FLAG                 (1 << 0)
#define     NW_IND_DATA_REG_EVENT_IND_FLAG                  (1 << 1)
#define     NW_IND_SIGNAL_STRENGTH_EVENT_IND_FLAG           (1 << 2)
#define     NW_IND_CELL_ACCESS_STATE_CHG_EVENT_IND_FLAG     (1 << 3)
#define     NW_IND_NITZ_TIME_UPDATE_EVENT_IND_FLAG          (1 << 4)
```

- **Parameter**

| Parameter | Description |
|---|---|
| *NW_IND_VOICE_REG_EVENT_IND_FLAG* | Voice-dialing registration event |
| *NW_IND_DATA_REG_EVENT_IND_FLAG* | Data-dialing registration event |
| *NW_IND_SIGNAL_STRENGTH_EVENT_IND_FLAG* | Signal strength event |
| *NW_IND_CELL_ACCESS_STATE_CHG_EVENT_IND_FLAG* | Cell access state change event |
| *NW_IND_NITZ_TIME_UPDATE_EVENT_IND_FLAG* | Network time update event |

**NOTE**

You can register multiple network events.

### 3.4.7. QL_MCM_NW_GetOperatorName

This function gets the information of a mobile network operator (wireless carrier).

- **Prototype**

```
E_QL_ERROR_CODE_T QL_MCM_NW_GetOperatorName(nw_client_handle_type h_nw, QL_MCM_
NW_OPERATOR_NAME_INFO_T *pt_info);
```

- **Parameter**

*h_nw*:
[In] Network registration service handle returned by *QL_MCM_NW_Client_Init().*

*pt_info*:
[In] The information of a mobile network operator. See ***Chapter 3.4.7.1*** for details.

- **Return Value**

| | |
|---|---|
| *E_QL_SUCCESS* | Got the operator information successfully. |
| Other values | Failed to get the operator information. See *ql_mcm.h* for the error code. |

#### 3.4.7.1. QL_MCM_NW_OPERATOR_NAME_INFO_T

The operator information is defined as follows:

```
typedef struct
{
    char long_eons[512 + 1];
    char short_eons[512 + 1];
char mcc[3 + 1];
char mnc[3 + 1];
}QL_MCM_NW_OPERATOR_NAME_INFO_T;
```

- **Parameter**

| Type | Parameters | Description |
|---|---|---|
| char | *long_eons* | Full name of the operator |
| char | *short_eons* | Short name of the operator |
| char | *mcc* | Mobile country code |
| char | *mnc* | Mobile network code |

### 3.4.8. QL_MCM_NW_PerformScan

This function triggers a network scan. It may take a long time to complete network scan, so wait for the result (*pt_info*) patiently.

- **Prototype**

```
E_QL_ERROR_CODE_T QL_MCM_NW_PerformScan(nw_client_handle_type h_nw, QL_MCM_NW_SCAN_RESULT_LIST_INFO_T *pt_info);
```

- **Parameter**

*h_nw*:
[In] Network registration service handle returned by *QL_MCM_NW_Client_Init().*

*pt_info*:
[In] Network scan result (network information). See ***Chapter 3.4.8.1*** for details.

- **Return Value**

| | |
|---|---|
| *E_QL_SUCCESS* | Completed network scan successfully. |
| Other values | Failed to scan the network. See *ql_mcm.h* for the error code. |

### 3.4.8.1. QL_MCM_NW_SCAN_RESULT_LIST_INFO_T

The result of network scan is defined as follows:

```
typedef struct
{
uint32_t entry_len;
QL_MCM_NW_SCAN_ENTRY_INFO_T entry[QL_MCM_NW_SCAN_LIST_MAX];
}QL_MCM_NW_SCAN_RESULT_LIST_INFO_T;
```

- **Parameter**

| Type | Parameter | Description |
|---|---|---|
| uint32_t | *entry_len* | The length of the network scan result. |
| *QL_MCM_NW_SCAN_ENTRY_INFO_T* | *entry* | Network scan result.<br>See ***Chapter 3.4.8.2*** for details. |

### 3.4.8.2. QL_MCM_NW_SCAN_RESULT_LIST_INFO_T

The network scan result is defined as follows:

```
typedef struct
{
QL_MCM_NW_OPERATOR_NAME_INFO_T          operator_name;
E_QL_MCM_NW_NETWORK_STATUS_TYPE_T       network_status;
E_QL_MCM_NW_RADIO_TECH_TYPE_T           rat;
}QL_MCM_NW_SCAN_ENTRY_INFO_T;
```

- **Parameter**

| Type | Parameter | Description |
|---|---|---|
| *QL_MCM_NW_OPERATOR_NAME_INFO_T* | *operator_name* | Operator information.<br>See ***Chapter 3.4.7.1*** for details. |

| | | |
|---|---|---|
| *E_QL_MCM_NW_NETWORK_STATUS_TYPE_T* | *network_status* | Network status.<br>See ***Chapter 3.4.8.3*** for details. |
| *E_QL_MCM_NW_RADIO_TECH_TYPE_T* | *rat* | Radio access technologies.<br>See ***Chapter 3.4.8.4*** for details. |

### 3.4.8.3. E_QL_MCM_NW_NETWORK_STATUS_TYPE_T

The network status is defined as follows:

```
typedef  enum
{
      E_QL_MCM_NW_NETWORK_STATUS_NONE                    = 0,
      E_QL_MCM_NW_NETWORK_STATUS_CURRENT_SERVING         = 1,
      E_QL_MCM_NW_NETWORK_STATUS_PREFERRED               = 2,
      E_QL_MCM_NW_NETWORK_STATUS_NOT_PREFERRED           = 3,
      E_QL_MCM_NW_NETWORK_STATUS_AVAILABLE               = 4,
      E_QL_MCM_NW_NETWORK_STATUS_FORBIDDEN               = 5
}E_QL_MCM_NW_NETWORK_STATUS_TYPE_T;
```

- **Parameter**

| **Parameter** | **Description** |
|---|---|
| *E_QL_MCM_NW_NETWORK_STATUS_NONE* | Unknown network status |
| *E_QL_MCM_NW_NETWORK_STATUS_CURRENT_SERVING* | The serving network |
| *E_QL_MCM_NW_NETWORK_STATUS_PREFERRED* | Preferred network |
| *E_QL_MCM_NW_NETWORK_STATUS_NOT_PREFERRED* | Non-preferred network |
| *E_QL_MCM_NW_NETWORK_STATUS_AVAILABLE* | Available network |
| *E_QL_MCM_NW_NETWORK_STATUS_FORBIDDEN* | Forbidden network |

### 3.4.8.4. E_QL_MCM_NW_RADIO_TECH_TYPE_T

The radio access technologies are defined as follows:

```
typedef  enum
{
      E_QL_MCM_NW_RADIO_TECH_TD_SCDMA    = 1,
      E_QL_MCM_NW_RADIO_TECH_GSM         = 2,
      E_QL_MCM_NW_RADIO_TECH_HSPAP       = 3,
      E_QL_MCM_NW_RADIO_TECH_LTE            = 4,
      E_QL_MCM_NW_RADIO_TECH_EHRPD          = 5,
      E_QL_MCM_NW_RADIO_TECH_EVDO_B         = 6,
      E_QL_MCM_NW_RADIO_TECH_HSPA           = 7,
      E_QL_MCM_NW_RADIO_TECH_HSUPA          = 8,
      E_QL_MCM_NW_RADIO_TECH_HSDPA          = 9,
      E_QL_MCM_NW_RADIO_TECH_EVDO_A         = 10,
      E_QL_MCM_NW_RADIO_TECH_EVDO_0         = 11,
      E_QL_MCM_NW_RADIO_TECH_1xRTT          = 12,
      E_QL_MCM_NW_RADIO_TECH_IS95B          = 13,
      E_QL_MCM_NW_RADIO_TECH_IS95A          = 14,
      E_QL_MCM_NW_RADIO_TECH_UMTS           = 15,
      E_QL_MCM_NW_RADIO_TECH_EDGE           = 16,
      E_QL_MCM_NW_RADIO_TECH_GPRS           = 17,
      E_QL_MCM_NW_RADIO_TECH_NONE           = 18
}E_QL_MCM_NW_RADIO_TECH_TYPE_T;
```

- **Parameter**

| Parameter | Description |
|---|---|
| *E_QL_MCM_NW_RADIO_TECH_TD_SCDMA* | TD-SCDMA network |
| *E_QL_MCM_NW_RADIO_TECH_GSM* | GSM network |
| *E_QL_MCM_NW_RADIO_TECH_HSPAP* | HSPA+ network |
| *E_QL_MCM_NW_RADIO_TECH_LTE* | LTE network |
| *E_QL_MCM_NW_RADIO_TECH_EHRPD* | eHRPD network |
| *E_QL_MCM_NW_RADIO_TECH_EVDO_B* | EVDO_B network |
| *E_QL_MCM_NW_RADIO_TECH_HSPA* | HSPA network |
| *E_QL_MCM_NW_RADIO_TECH_HSUPA* | HSUPA network |
| *E_QL_MCM_NW_RADIO_TECH_HSDPA* | HSDPA network |
| *E_QL_MCM_NW_RADIO_TECH_EVDO_A* | EVDO_A network |
| *E_QL_MCM_NW_RADIO_TECH_EVDO_0* | EVDO_0 network |
| *E_QL_MCM_NW_RADIO_TECH_1xRTT* | 1xRTT network |
| *E_QL_MCM_NW_RADIO_TECH_IS95B* | IS-95B network |
| *E_QL_MCM_NW_RADIO_TECH_IS95A* | IS-95A network |

| | |
|---|---|
| *E_QL_MCM_NW_RADIO_TECH_UMTS* | UMTS network |
| *E_QL_MCM_NW_RADIO_TECH_EDGE* | EDGE network |
| *E_QL_MCM_NW_RADIO_TECH_GPRS* | GPRS network |
| *E_QL_MCM_NW_RADIO_TECH_NONE* | Unknown network |

### 3.4.9. QL_MCM_NW_GetRegStatus

This function gets the information about the module's network registration status, including the network registration status for voice dialing and data dialing.

- **Prototype**

```
E_QL_ERROR_CODE_T QL_MCM_NW_GetRegStatus(nw_client_handle_type h_nw, QL_MCM_NW_
REG_STATUS_INFO_T *pt_info);
```

- **Parameter**

*h_nw*:
[In] Network registration service handle returned by *QL_MCM_NW_Client_Init().*

*pt_info*:
[In] Network registration status information. See ***Chapter 3.4.9.1*** for details.

- **Return Value**

*E_QL_SUCCESS* Got the network registration status information successfully.
Other values Failed to get the network registration status information. See *ql_mcm.h* for the error code.

#### 3.4.9.1. QL_MCM_NW_REG_STATUS_INFO_T

The network registration status information is defined as follows:

```
typedef struct
{
    uint8_t                                    voice_registration_valid;
    QL_MCM_NW_COMMON_REG_INFO_T                voice_registration;
    uint8_t                                    data_registration_valid;
    QL_MCM_NW_COMMON_REG_INFO_T                data_registration;
    uint8_t                                    voice_registration_details_3gpp_valid;
    QL_MCM_NW_3GPP_REG_INFO_T                  voice_registration_details_3gpp;
    uint8_t                                    data_registration_details_3gpp_valid;
    QL_MCM_NW_3GPP_REG_INFO_T          data_registration_details_3gpp;
    uint8_t                            voice_registration_details_3gpp2_valid;
    QL_MCM_NW_3GPP2_REG_INFO_T         voice_registration_details_3gpp2;
    uint8_t                            data_registration_details_3gpp2_valid;
    QL_MCM_NW_3GPP2_REG_INFO_T         data_registration_details_3gpp2;
}QL_MCM_NW_REG_STATUS_INFO_T;
```

- **Parameter**

| Type | Parameter | Description |
|---|---|---|
| uint8_t | *voice_registration_valid* | Indicates whether *voice_registration* is valid. |
| *QL_MCM_NW_COMMON_REG_INFO_T* | *voice_registration* | Network registration status information for voice dialing.<br>See ***Chapter 3.4.9.2*** for details. |
| uint8_t | *data_registration_valid* | Indicates whether *data_registration* is valid. |
| *QL_MCM_NW_COMMON_REG_INFO_T* | *data_registration* | Network registration status information for data dialing.<br>See ***Chapter 3.4.9.2*** for details. |
| uint8_t | *voice_registration_details_3gpp_valid* | Indicates whether *voice_registration_details_3gpp* is valid. |
| *QL_MCM_NW_3GPP_REG_INFO_T* | *voice_registration_details_3gpp* | Network registration status information for 3GPP compliant voice dialing.<br>See ***Chapter 3.4.9.3*** for details. |
| uint8_t | *data_registration_details_3gpp_valid* | Indicates whether *data_registration_details_3gpp* is valid. |
| *QL_MCM_NW_3GPP_REG_INFO_T* | *data_registration_details_3gpp* | Network registration status information for 3GPP compliant data dialing.<br>See ***Chapter 3.4.9.3*** for details. |
| uint8_t | *voice_registration_details_3gpp2_valid* | Indicates whether *voice_registration_details_3gpp2* is valid. |
| *QL_MCM_NW_3GPP2_REG_INFO_T* | *voice_registration_details_3gpp2* | Network registration status information for 3GPP2 compliant voice dialing.<br>See ***Chapter 3.4.9.4*** for details. |
| uint8_t | *data_registration_details_3gpp2_valid* | Indicates whether *data_registration_details_3gpp2* is valid. |
| *QL_MCM_NW_3GPP2_REG_INFO_T* | *data_registration_details_3gpp2* | Network registration status information for 3GPP2 compliant data dialing.<br>See ***Chapter 3.4.9.4*** for details. |

### 3.4.9.2. QL_MCM_NW_COMMON_REG_INFO_T

The information about the network registration status for voice and data dialing is defined as follows:

```
typedef struct
{
    E_QL_MCM_NW_TECH_DOMAIN_TYPE_T      tech_domain;
    E_QL_MCM_NW_RADIO_TECH_TYPE_T       radio_tech;
    E_QL_MCM_NW_ROAM_STATE_TYPE_T       roaming;
    E_QL_MCM_NW_DENY_REASON_TYPE_T      deny_reason;
    E_QL_MCM_NW_SERVICE_TYPE_T          registration_state;
}QL_MCM_NW_COMMON_REG_INFO_T;
```

- **Parameter**

| Type | Parameter | Description |
|---|---|---|
| *E_QL_MCM_NW_TECH_DOMAIN_TYPE_T* | *tech_domain* | Technical specification type.<br>See ***Chapter 3.4.9.5*** for details. |
| *E_QL_MCM_NW_RADIO_TECH_TYPE_T* | *radio_tech* | Radio access technologies.<br>See ***Chapter 3.4.8.4*** for details. |
| *E_QL_MCM_NW_ROAM_STATE_TYPE_T* | *roaming* | Roaming status.<br>See ***Chapter 3.4.3.3*** for details. |
| *E_QL_MCM_NW_DENY_REASON_TYPE_T* | *deny_reason* | Network registration rejection causes.<br>See ***Chapter 3.4.9.6*** for details. |
| *E_QL_MCM_NW_SERVICE_TYPE_T* | *registration_state* | Network service type.<br>See ***Chapter 3.4.9.7*** for details. |

### 3.4.9.3. QL_MCM_NW_SCAN_RESULT_LIST_INFO_T

The information about the network registration status for 3GPP compliant voice and data dialing, is defined as follows:

```
typedef struct
{
    E_QL_MCM_NW_TECH_DOMAIN_TYPE_T      tech_domain;
    E_QL_MCM_NW_RADIO_TECH_TYPE_T       radio_tech;
    char                                mcc[3+1];
    char                                mnc[3+1];
    E_QL_MCM_NW_ROAM_STATE_TYPE_T       roaming;
    uint8_t                             forbidden;
    uint32_t                            cid;
    uint16_t                            lac;
    uint16_t                                psc;
    uint16_t                                tac;
}QL_MCM_NW_3GPP_REG_INFO_T;
```

- **Parameter**

| Type | Parameter | Description |
|---|---|---|
| *E_QL_MCM_NW_TECH_DOMAIN_TYPE_T* | *tech_domain* | Technical specification type.<br>See ***Chapter 3.4.9.5*** for details. |
| *E_QL_MCM_NW_RADIO_TECH_TYPE_T* | *radio_tech* | Radio access technologies.<br>See ***Chapter 3.4.8.4*** for details |
| char | *mcc* | Mobile country code. |
| char | *mnc* | Mobile network code. |
| *E_QL_MCM_NW_ROAM_STATE_TYPE_T* | *roaming* | Roaming status.<br>See ***Chapter 3.4.3.3*** for details. |
| uint8_t | *forbidden* | Network forbidden. |
| uint32_t | *cid* | Cell ID. |
| uint16_t | *lac* | Location area code. |
| uint16_t | *psc* | Primary scrambling code. |
| uint16_t | *tac* | Tracking area code. |

### 3.4.9.4. QL_MCM_NW_3GPP2_REG_INFO_T

The information about the network registration status for 3GPP2 compliant voice and data dialing, is defined as follows:

```
typedef struct
{
    E_QL_MCM_NW_TECH_DOMAIN_TYPE_T          tech_domain;
    E_QL_MCM_NW_RADIO_TECH_TYPE_T           radio_tech;
    char                                    mcc[3+1];
    char                                    mnc[3+1];
    E_QL_MCM_NW_ROAM_STATE_TYPE_T           roaming;
    uint8_t                                 forbidden;
    uint8_t                                 inPRL;
    uint8_t                                 css;
    uint16_t                                sid;
    uint16_t                                nid;
    uint16_t                                    bsid;
}QL_MCM_NW_3GPP2_REG_INFO_T;
```

- **Parameter**

| Type | Parameter | Description |
|---|---|---|
| *E_QL_MCM_NW_TECH_DOMAIN_TYPE_T* | *tech_domain* | Technical specification type.<br>See ***Chapter 3.4.9.5*** for details. |
| *E_QL_MCM_NW_RADIO_TECH_TYPE_T* | *radio_tech* | Radio access technologies.<br>See ***Chapter 3.4.8.4*** for details |
| char | *mcc* | Mobile country code. |
| char | *mnc* | Mobile network code. |
| *E_QL_MCM_NW_ROAM_STATE_TYPE_T* | *roaming* | Roaming status.<br>See ***Chapter 3.4.3.3*** for details. |
| uint8_t | *forbidden* | Network forbidden. |
| uint8_t | *inPRL* | PRL networks preferred. |
| uint8_t | *css* | Concurrency support. |
| uint16_t | *sid* | System ID. |
| uint16_t | *nid* | Network ID. |
| uint16_t | *bsid* | Base station ID. |

#### 3.4.9.5. E_QL_MCM_NW_TECH_DOMAIN_TYPE_T

The technical specification type is defined as follows:

```
typedef enum
{
    E_QL_MCM_NW_TECH_DOMAIN_NONE        = 0,
    E_QL_MCM_NW_TECH_DOMAIN_3GPP        = 1,
    E_QL_MCM_NW_TECH_DOMAIN_3GPP2       = 2,
}E_QL_MCM_NW_TECH_DOMAIN_TYPE_T;
```

- **Parameter**

| Parameter | Description |
|---|---|
| *E_QL_MCM_NW_TECH_DOMAIN_NONE* | Unknown technical specification. |

| | |
|---|---|
| *E_QL_MCM_NW_TECH_DOMAIN_3GPP* | 3GPP. |
| *E_QL_MCM_NW_TECH_DOMAIN_3GPP2* | 3GPP2. |

### 3.4.9.6. E_QL_MCM_NW_DENY_REASON_TYPE_T

The rejection reason for network registration is defined as follows:

```
typedef enum
{
    E_QL_MCM_NW_IMSI_UNKNOWN_HLR_DENY_REASON   = 1,
    E_QL_MCM_NW_ILLEGAL_MS_DENY_REASON         = 2,
    E_QL_MCM_NW_IMSI_UNKNOWN_VLR_DENY_REASON   = 3,
    E_QL_MCM_NW_IMEI_NOT_ACCEPTED_DENY_REASON  = 4,
    E_QL_MCM_NW_ILLEGAL_ME_DENY_REASON         = 5,
    E_QL_MCM_NW_PLMN_NOT_ALLOWED_DENY_REASON   = 6,
    E_QL_MCM_NW_LA_NOT_ALLOWED_DENY_REASON     = 7,
    E_QL_MCM_NW_ROAMING_NOT_ALLOWED_LA_DENY_REASON   = 8,
    E_QL_MCM_NW_NO_SUITABLE_CELLS_LA_DENY_REASON     = 9,
    E_QL_MCM_NW_NETWORK_FAILURE_DENY_REASON          = 10,
    E_QL_MCM_NW_MAC_FAILURE_DENY_REASON              = 11,
    E_QL_MCM_NW_SYNCH_FAILURE_DENY_REASON            = 12,
    E_QL_MCM_NW_CONGESTION_DENY_REASON               = 13,
    E_QL_MCM_NW_GSM_AUTHENTICATION_UNACCEPTABLE_DENY_REASON     = 14,
    E_QL_MCM_NW_NOT_AUTHORIZED_CSG_DENY_REASON                  = 15,
    E_QL_MCM_NW_SERVICE_OPTION_NOT_SUPPORTED_DENY_REASON        = 16
    E_QL_MCM_NW_REQ_SERVICE_OPTION_NOT_SUBSCRIBED_DENY_REASON = 17,
    E_QL_MCM_NW_CALL_CANNOT_BE_IDENTIFIED_DENY_REASON           = 18,
    E_QL_MCM_NW_SEMANTICALLY_INCORRECT_MSG_DENY_REASON          = 19,
    E_QL_MCM_NW_INVALID_MANDATORY_INFO_DENY_REASON              = 20,
    E_QL_MCM_NW_MSG_TYPE_NON_EXISTENT_DENY_REASON               = 21,
    E_QL_MCM_NW_INFO_ELEMENT_NON_EXISTENT_DENY_REASON           = 22,
    E_QL_MCM_NW_CONDITIONAL_IE_ERR_DENY_REASON                  = 23,
    E_QL_MCM_NW_MSG_INCOMPATIBLE_PROTOCOL_STATE_DENY_REASON     = 24,
    E_QL_MCM_NW_PROTOCOL_ERROR_DENY_REASON                      = 25,
}E_QL_MCM_NW_DENY_REASON_TYPE_T;
```

- **Parameter**

| Parameter | Description |
|---|---|
| *E_QL_MCM_NW_IMSI_UNKNOWN_HLR_DENY_REASON* | Unknown IMSI in HLR |

| | |
|---|---|
| *E_QL_MCM_NW_ILLEGAL_MS_DENY_REASON* | Illegal mobile station |
| *E_QL_MCM_NW_IMSI_UNKNOWN_VLR_DENY_REASON* | Unknown IMSI in VLR |
| *E_QL_MCM_NW_IMEI_NOT_ACCEPTED_DENY_REASO* | IMEI not recognized |
| *E_QL_MCM_NW_ILLEGAL_ME_DENY_REASON* | Illegal mobile equipment |
| *E_QL_MCM_NW_PLMN_NOT_ALLOWED_DENY_REASON* | PLMN not allowed |
| *E_QL_MCM_NW_LA_NOT_ALLOWED_DENY_REASON* | Location not allowed |
| *E_QL_MCM_NW_ROAMING_NOT_ALLOWED_LA_DENY_REASON* | Roaming not allowed is this location area |
| *E_QL_MCM_NW_NO_SUITABLE_CELLS_LA_DENY_REASON* | No suitable cells in this location area |
| *E_QL_MCM_NW_NETWORK_FAILURE_DENY_REASON* | Network failure |
| *E_QL_MCM_NW_MAC_FAILURE_DENY_REASON* | MAC failure |
| *E_QL_MCM_NW_SYNCH_FAILURE_DENY_REASON* | Sync failure |
| *E_QL_MCM_NW_CONGESTION_DENY_REASON* | Congestion |
| *E_QL_MCM_NW_GSM_AUTHENTICATION_UNACCEPTABLE_DENY_REASON* | GSM authentication unacceptable |
| *E_QL_MCM_NW_NOT_AUTHORIZED_CSG_DENY_REASON* | Not authorized CSG |
| *E_QL_MCM_NW_SERVICE_OPTION_NOT_SUPPORTED_DENY_REASON* | Service option not supported |
| *E_QL_MCM_NW_REQ_SERVICE_OPTION_NOT_SUBSCRIBED_DENY_REASON* | Service option not subscribed |
| *E_QL_MCM_NW_CALL_CANNOT_BE_IDENTIFIED_DENY_REASON* | Call cannot be identified |
| *E_QL_MCM_NW_SEMANTICALLY_INCORRECT_MSG_DENY_REASON* | Semantically incorrect message |
| *E_QL_MCM_NW_INVALID_MANDATORY_INFO_DENY_REASON* | Invalid mandatory information |
| *E_QL_MCM_NW_MSG_TYPE_NON_EXISTENT_DENY_REASON* | Message type non-existent |
| *E_QL_MCM_NW_INFO_ELEMENT_NON_EXISTENT_DENY_REASON* | Information element non-existent |
| *E_QL_MCM_NW_CONDITIONAL_IE_ERR_DENY_REASON* | IE error |
| *E_QL_MCM_NW_MSG_INCOMPATIBLE_PROTOCOL_STATE_DENY_REASON* | Message type not compatible with protocol state |

| | |
|---|---|
| *E_QL_MCM_NW_PROTOCOL_ERROR_DENY_REASON* | Protocol error |

#### 3.4.9.7. E_QL_MCM_NW_SERVICE_TYPE_T

The network service type is defined as follows:

```
typedef  enum
{
     E_QL_MCM_NW_SERVICE_NONE              =  0x0000,
     E_QL_MCM_NW_SERVICE_LIMITED           =  0x0001,
     E_QL_MCM_NW_SERVICE_FULL              =  0x0002,
}E_QL_MCM_NW_SERVICE_TYPE_T;
```

- **Parameter**

| Parameter | Description |
|---|---|
| *E_QL_MCM_NW_SERVICE_NONE* | No service |
| *E_QL_MCM_NW_SERVICE_LIMITED* | Restricted service |
| *E_QL_MCM_NW_SERVICE_FULL* | Normal service |

### 3.4.10. QL_MCM_NW_SetLowPowerMode

This function sets whether to enable the low power mode, that is, whether to disable the reporting of signal strength event.

- **Prototype**

```
E_QL_ERROR_CODE_T  QL_MCM_NW_SetLowPowerMode(nw_client_handle_type  h_nw,  uint32_t  l
ow_power_mode_on);
```

- **Parameter**

*h_nw*:
[In] Network registration service handle returned by *QL_MCM_NW_Client_Init().*

*low_power_mode_on*:
[In] Low power mode.
0 Normal mode. Enable the reporting of signal strength event.
1 Low power mode. Disable the reporting of signal strength event.

- **Return Value**

| | |
|---|---|
| *E_QL_SUCCESS* | Set the power mode successfully. |
| Other values | Failed to set the power mode. See *ql_mcm.h* for the error code. |

NOTES

1. The setting of this function will not be saved after power-off.
2. This setting is valid for all clients.

### 3.4.11. QL_MCM_NW_GetSignalStrength

This function gets the signal strength information. It only returns the signal strength information of the network which the module current registers on.

- **Prototype**

```
E_QL_ERROR_CODE_T QL_MCM_NW_GetSignalStrength(nw_client_handle_type h_nw, QL_MCM_
NW_SIGNAL_STRENGTH_INFO_T *pt_info);
```

- **Parameter**

*h_nw*:
[In] Network registration service handle returned by *QL_MCM_NW_Client_Init().*

*pt_info*:
[Out] Signal strength information. See ***Chapter 3.4.11.1*** for details.

- **Return Value**

| | |
|---|---|
| *E_QL_SUCCESS* | Got the signal strength information successfully. |
| Other values | Failed to get the signal strength information. See *ql_mcm.h* for the error code. |

#### 3.4.11.1. QL_MCM_NW_SIGNAL_STRENGTH_INFO_T

The signal strength information is defined as follows:

```
typedef struct
{
    uint8_t                               gsm_sig_info_valid;
    QL_MCM_NW_GSM_SIGNAL_INFO_T           gsm_sig_info;
    uint8_t                               wcdma_sig_info_valid;
    QL_MCM_NW_WCDMA_SIGNAL_INFO_T         wcdma_sig_info;
    uint8_t                               tdscdma_sig_info_valid;
    QL_MCM_NW_TDSCDMA_SIGNAL_INFO_T     tdscdma_sig_info;
    uint8_t                             lte_sig_info_valid;
    QL_MCM_NW_LTE_SIGNAL_INFO_T         lte_sig_info;
    uint8_t                             cdma_sig_info_valid;
    QL_MCM_NW_CDMA_SIGNAL_INFO_T        cdma_sig_info;
    uint8_t                             hdr_sig_info_valid;
    QL_MCM_NW_HDR_SIGNAL_INFO_T         hdr_sig_info;
}QL_MCM_NW_SIGNAL_STRENGTH_INFO_T;
```

- **Parameter**

| Type | Parameter | Description |
|---|---|---|
| *uint8_t* | *gsm_sig_info_valid* | Indicates whether *gsm_sig_info* is valid. |
| *QL_MCM_NW_GSM_SIGNAL_INFO_T* | *gsm_sig_info* | GSM signal strength information.<br>See ***Chapter 3.4.11.2*** for details. |
| uint8_t | *wcdma_sig_info_valid* | Indicates whether *wcdma_sig_info* is valid. |
| *QL_MCM_NW_WCDMA_SIGNAL_INFO_T* | *wcdma_sig_info* | WCDMA signal strength information.<br>See ***Chapter 3.4.11.3*** for details. |
| uint8_t | *tdscdma_sig_info_valid* | Indicates whether *tdscdma_sig_info* is valid. |
| *QL_MCM_NW_TDSCDMA_SIGNAL_INFO_T* | *tdscdma_sig_info* | TD-SCDMA signal strength information.<br>See ***Chapter 3.4.11.4*** for details. |
| uint8_t | *lte_sig_info_valid* | Indicates whether *lte_sig_info* is valid |
| *QL_MCM_NW_LTE_SIGNAL_INFO_T* | *lte_sig_info* | LTE signal strength information.<br>See ***Chapter 3.4.11.5*** for details. |
| uint8_t | *cdna_sig_info_valid* | Indicates whether *tcdma_sig_info* is valid. |
| *QL_MCM_NW_CDMA_SIGNAL_INFO_T* | *cdma_sig_info* | CDMA signal strength information.<br>See ***Chapter 3.4.11.6*** for details. |
| uint8_t | *hdr_sig_info_valid* | Indicates whether *hdr_sig_info* is valid. |
| *QL_MCM_NW_HDR_SIGNAL_INFO_T* | hdr_sig_info | HDR signal strength information.<br>See ***Chapter 3.4.11.7*** for details. |

### 3.4.11.2. QL_MCM_NW_GSM_SIGNAL_INFO_T

The GSM signal strength information is defined as follows:

```
typedef  struct
{
    int8_t          rssi;
}QL_MCM_NW_GSM_SIGNAL_INFO_T;
```

- **Parameter**

| Type | Parameter | Description |
|---|---|---|
| uint8_t | *rssi* | Received signal strength indicator. Unit: dBm. |

### 3.4.11.3. QL_MCM_NW_WCDMA_SIGNAL_INFO_T

WCDMA signal strength information is defined as follows:

```
typedef  struct
{
      int8_t      rssi;
      int16_t    ecio;
}QL_MCM_NW_WCDMA_SIGNAL_INFO_T;
```

- **Parameter**

| Type | Parameter | Description |
|---|---|---|
| uint8_t | *rssi* | Received signal strength indicator. Unit: dBm. |
| int16_t | *ecio* | Energy per chip to interference power ratio. Unit: -0.5 dB. |

### 3.4.11.4. QL_MCM_NW_TDSCDMA_SIGNAL_INFO_T

TD-SCDMA signal strength information is defined as follows:

```
typedef  struct
{
      int8_t      rssi;
      int8_t      rscp;
      int16_t    ecio;
      int8_t      sinr;
}QL_MCM_NW_TDSCDMA_SIGNAL_INFO_T;
```

- **Parameter**

| Type | Parameter | Description |
|---|---|---|
| uint8_t | *rssi* | Received signal strength indicator. Unit: dBm. |

| | | |
|---|---|---|
| uint8_t | *rscp* | Received signal code power. Unit: dBm. |
| uint16_t | *ecio* | Energy per chip to interference power ratio. Unit: dB. |
| uint8_t | *sinr* | Signal-to-interference-plus-noise ratio. Unit: dB. |

### 3.4.11.5. QL_MCM_NW_LTE_SIGNAL_INFO_T

The LTE signal strength information is defined as follows:

```
typedef  struct
{
      int8_t     rssi;
      int8_t     rsrq;
      int16_t   rsrp;
      int16_t   snr;
}QL_MCM_NW_LTE_SIGNAL_INFO_T;
```

- **Parameter**

| Type | Parameter | Description |
|---|---|---|
| uint8_t | *rssi* | Received signal strength indicator. Unit: dBm. |
| uint8_t | *rsrq* | Reference signal received quality. Unit: dB. |
| int16_t | *rsrp* | Reference signal received power. Unit: dBm. |
| int16_t | *snr* | Signal-to-noise ratio. Unit: 0.1 dB. |

### 3.4.11.6. QL_MCM_NW_CDMA_SIGNAL_INFO_T

The CDMA signal strength information is defined as follows:

```
typedef  struct
{
      int8_t     rssi;
      int16_t   ecio;
}QL_MCM_NW_CDMA_SIGNAL_INFO_T;
```

- **Parameter**

| Type | Parameter | Description |
|---|---|---|
| uint8_t | *rssi* | Received signal strength indicator. Unit: dBm. |
| int16_t | *ecio* | Energy per chip to interference power ratio. Unit: -0.5 dB. |

### 3.4.11.7. QL_MCM_NW_HDR_SIGNAL_INFO_T

HDR signal strength information is defined as follows:

```
typedef  struct
{
     int8_t     rssi;
     int16_t    ecio;
     int8_t     sinr;
     int32_t    io;
}QL_MCM_NW_HDR_SIGNAL_INFO_T;
```

- **Parameter**

| Type | Parameter | Description |
|---|---|---|
| uint8_t | *rssi* | Received signal strength indicator. Unit: dBm. |
| int16_t | *ecio* | Energy per chip to interference power ratio. Unit: -0.5 dB. |
| uint8_t | *sinr* | SINR level. Range: 1–8.<br>SINR level    SINR<br>0    -9 dB<br>1    -6 dB<br>2    -4.5 dB<br>3    -3 dB<br>4    -2 dB<br>5    1 dB<br>6    3 dB<br>7    6 dB<br>8    9 dB |
| int32_t | *io* | Interference of other cells. Unit: dBm.<br>Only applicable for 1x EVDO. |

### 3.4.12. QL_MCM_NW_GetCellAccessState

This function gets the cell access state.

- **Prototype**

```
E_QL_ERROR_CODE_T QL_MCM_NW_GetCellAccessState(nw_client_handle_type h_nw, E_QL_M
CM_NW_CELL_ACCESS_STATE_TYPE_T *pe_state);
```

- **Parameter**

*h_nw*:
[In] Network registration service handle returned by *QL_MCM_NW_Client_Init().*

*pe_state*:
[Out] Cell access state. See ***Chapter 3.4.12.1*** for details.

- **Return Value**

| | |
|---|---|
| *E_QL_SUCCESS* | Got the cell access state successfully. |
| Other values | Failed to get the cell access state. See *ql_mcm.h* for the error code. |

#### 3.4.12.1. E_QL_MCM_NW_CELL_ACCESS_STATE_TYPE_T

The cell access state is defined as follows:

```
typedef enum
{
    E_QL_MCM_NW_CELL_ACCESS_NONE                 = 0x00,
    E_QL_MCM_NW_CELL_ACCESS_NORMAL_ONLY          = 0x01,
    E_QL_MCM_NW_CELL_ACCESS_EMERGENCY_ONLY = 0x02,
    E_QL_MCM_NW_CELL_ACCESS_NO_CALLS             = 0x03,
    E_QL_MCM_NW_CELL_ACCESS_ALL_CALLS            = 0x04,
}E_QL_MCM_NW_CELL_ACCESS_STATE_TYPE_T;
```

- **Parameter**

| **Parameter** | **Description** |
|---|---|
| *QL_NW_CELL_ACCESS_NONE* | Unknown access |
| *QL_NW_CELL_ACCESS_NORMAL_ONLY* | Normal access only |
| *QL_NW_CELL_ACCESS_EMERGENCY_ONLY* | Emergency access only |
| *QL_NW_CELL_ACCESS_NO_CALLS* | No access |

| | |
|---|---|
| *QL_NW_CELL_ACCESS_ALL_CALLS* | All access |

### 3.4.13. QL_MCM_NW_AddRxMsgHandler

This function sets the callback function for network events. When the event registered with *QL_MCM_NW_EventRegister()* occurs, the callback function of the event will be called automatically.

- **Prototype**

```
E_QL_ERROR_CODE_T QL_MCM_NW_AddRxMsgHandler(nw_client_handle_type h_nw, QL_MCM_NW_RxMsgHandlerFunc_t handlerPtr, void* contextPtr);
```

- **Parameter**

*h_nw*:
[In] Network registration service handle returned by *QL_MCM_NW_Client_Init().*

*handlerPtr*:
[In] Event callback function.

*contextPtr*:
[In] Void pointer. (Reserved.)

- **Return Value**

| | |
|---|---|
| *E_QL_SUCCESS* | Set the callback function successfully. |
| Other values | Failed to set the callback function. See *ql_mcm.h* for the error code. |

# 4 Examples

All example codes shown in this chapter are all sourced from *ql-ol-sdk/ql-ol-extsdk/example/test_mcm_api/test_nw.c* where you can view the complete examples of API functions.

## 4.1. Initialize Network Registration Service

```
case 0://"QL_MCM_NW_Client_Init"
{
    ret = QL_MCM_NW_Client_Init(&h_nw);
    printf("QL_MCM_NW_Client_Init ret = %d\n", ret);
    break;
}
```

## 4.2. Deinitialize Network Registration Service

```
case 12://"QL_MCM_NW_Client_Deinit"
{
    ret = QL_MCM_NW_Client_Deinit(h_nw);
    printf("QL_MCM_NW_Client_Deinit ret = %d\n", ret);
    break;
}
```

## 4.3. Set the Preferred Network Mode and Roaming Notification Status

```
case 1://"QL_MCM_NW_SetConfig"
{
    QL_MCM_NW_CONFIG_INFO_T        t_info = {0};

    int mask = 0;
    printf("please input event mask hex(16_PRL | TDSCDMA | LTE | EVDO | CDMA | WCDMA | GSM):
\n");
```

```
    scanf("%x", &mask);
    t_info.preferred_nw_mode = mask;

    printf("please input roaming pref(0:off 1:on): \n");
    scanf("%d", &mask);
    t_info.roaming_pref = mask;

    ret = QL_MCM_NW_SetConfig(h_nw, &t_info);
    printf("QL_MCM_NW_SetConfig ret = %d\n", ret);
    break;
}
```

## 4.4. Get the Preferred Network Mode and Roaming Notification Status

```
case 2://"QL_MCM_NW_GetConfig"
{
    QL_MCM_NW_CONFIG_INFO_T        t_info = {0};
    ret = QL_MCM_NW_GetConfig(h_nw, &t_info);
    printf("QL_MCM_NW_GetConfig ret = %d, preferred_nw_mode=0x%X, roaming=%d\n", ret,
t_info.preferred_nw_mode, t_info.roaming_pref);
    break;
}
```

## 4.5. Get Network Time

```
case 3://"QL_MCM_NW_GetNitzTimeInfo"
{
    QL_MCM_NW_NITZ_TIME_INFO_T t_info;
    ret = QL_MCM_NW_GetNitzTimeInfo(h_nw, &t_info);
    printf("QL_MCM_NW_GetNitzTimeInfo ret = %d, nitz_time=%s, abs_time=%lld, leap_sec=%d, \n",
            ret, t_info.nitz_time, t_info.abs_time, t_info.leap_sec);
    break;
}
```

## 4.6. Register Network Events

```
case 4://"QL_MCM_NW_EventRegister"
{
    int mask = 0;

    printf("please input event mask-hex(NITZ_UPDATE | CELL_ACC_STATE_CHG | SIG_STRENGTH |
DATA | VOICE): \n");
    scanf("%x", &mask);

    ret = QL_MCM_NW_EventRegister(h_nw, mask);
    printf("QL_MCM_NW_EventRegister ret = %d\n", ret);
    break;
}
```

## 4.7. Get Operator Information

```
case 5://"QL_MCM_NW_GetOperatorName"
{
    QL_MCM_NW_OPERATOR_NAME_INFO_T    t_info;
    ret = QL_MCM_NW_GetOperatorName(h_nw, &t_info);
    printf("QL_MCM_NW_GetOperatorName ret = %d, long_eons=%s, short_eons=%s, mcc=%s,
mnc=%s\n", ret, t_info.long_eons, t_info.short_eons, t_info.mcc, t_info.mnc);
    break;
}
```

Scan the Network

```
case 6://"QL_MCM_NW_PerformScan"
{
    QL_MCM_NW_SCAN_RESULT_LIST_INFO_T      t_info;
    ret = QL_MCM_NW_PerformScan(h_nw, &t_info);
    printf("QL_MCM_NW_PerformScan ret = %d, list_len=%d, detail info.....\n", ret, t_info.entry_len);
    display_network_scan_result(t_info);
    break;
}

void display_network_scan_result(QL_MCM_NW_SCAN_RESULT_LIST_INFO_T info)
{
    int i = 0;
    char net_info[16] = {0};
    char radio_info[16] = {0};

    for(i = 0; i < info.entry_len; i++)
    {
        memset(net_info, 0, sizeof(net_info));
        memset(radio_info, 0, sizeof(radio_info));
        printf("\t[%d]: long_eons=%s, short_eons=%s, mcc=%s, mnc=%s, ",
               i,
               info.entry[i].operator_name.long_eons,
               info.entry[i].operator_name.short_eons,
               info.entry[i].operator_name.mcc,
               info.entry[i].operator_name.mnc);


        if(nw_get_net_status(info.entry[i].network_status, net_info, sizeof(net_info)) == 0)
        {
            printf("unrecognized network_status:%d, ", info.entry[i].network_status);
        }
        else
        {
            printf("network_status=%s, ", net_info);
        }

        if(nw_get_radio_tech(info.entry[i].rat, radio_info, sizeof(radio_info)) == 0)
        {
            printf("unrecognized rat:%d\n ", info.entry[i].rat);
        }
        else
        {
            printf("radio_tech=%s\n", radio_info);
        }
    }
}
```

## 4.8. Scan the Network

```
case 6://"QL_MCM_NW_PerformScan"
{
    int i = 0;
    QL_MCM_NW_SCAN_RESULT_LIST_INFO_T    *pt_info = NULL;
    char *net_status[] = {"NONE", "CURRENT_SERVING", "PREFERRED", "NOT_PREFERRED",
"AVAILABLE", "FORBIDDEN"};
```

```
    pt_info                    =                    (QL_MCM_NW_SCAN_RESULT_LIST_INFO_T
*)malloc(sizeof(QL_MCM_NW_SCAN_RESULT_LIST_INFO_T));
    if(pt_info == NULL)
    {
        printf("Out of memory!");
        break;
    }
    memset(pt_info, 0, sizeof(QL_MCM_NW_SCAN_RESULT_LIST_INFO_T));

    ret = QL_MCM_NW_PerformScan(h_nw, pt_info);
    printf("QL_MCM_NW_PerformScan ret = %d, list_len=%d, detail info:\n", ret, pt_info->entry_len);
    for(i=0; i<pt_info->entry_len; i++)
    {
        printf("\t[%d]: long_eons=%s, short_eons=%s, mcc=%s, mnc=%s, network_status=%s, rat=%s
\n",
                    i,
                    pt_info->entry[i].operator_name.long_eons,
                    pt_info->entry[i].operator_name.short_eons,
                    pt_info->entry[i].operator_name.mcc,
                    pt_info->entry[i].operator_name.mnc,
                    net_status[pt_info->entry[i].network_status],
                    radio_tech[pt_info->entry[i].rat]);
    }

    free(pt_info);
    break;
}
```

## 4.9. Get Network Registration Status Information

```
case 7://"QL_MCM_NW_GetRegStatus"
{
    QL_MCM_NW_REG_STATUS_INFO_T          t_info;

    memset(&t_info, 0, sizeof(QL_MCM_NW_REG_STATUS_INFO_T));
    ret = QL_MCM_NW_GetRegStatus(h_nw, &t_info);
    printf("QL_MCM_NW_GetRegStatus ret = %d, detail info:\n", ret);
    if(t_info.voice_registration_valid)
    {
        printf("voice_registration: \ntech_domain=%s, radio_tech=%s, roaming=%d, registration_state
=%d\n",
            tech_domain[t_info.voice_registration.tech_domain],
            radio_tech[t_info.voice_registration.radio_tech],
            t_info.voice_registration.roaming,
            t_info.voice_registration.registration_state);
    }
    if(t_info.data_registration_valid)
    {
        printf("data_registration: \ntech_domain=%s, radio_tech=%s, roaming=%d, registration_state
=%d\n",
            tech_domain[t_info.data_registration.tech_domain],
            radio_tech[t_info.data_registration.radio_tech],
            t_info.data_registration.roaming,
            t_info.data_registration.registration_state);
    }
    if(t_info.voice_registration_details_3gpp_valid)
    {
        printf("voice_registration_details_3gpp: \ntech_domain=%s, radio_tech=%s, mcc=%s, mnc=%s,
roaming=%d, forbidden=%d, cid=0x%X, lac=%d, psc=%d, tac=%d\n",
            tech_domain[t_info.voice_registration_details_3gpp.tech_domain],
            radio_tech[t_info.voice_registration_details_3gpp.radio_tech],
            t_info.voice_registration_details_3gpp.mcc,
            t_info.voice_registration_details_3gpp.mnc,
            t_info.voice_registration_details_3gpp.roaming,
            t_info.voice_registration_details_3gpp.forbidden,
            t_info.voice_registration_details_3gpp.cid,
            t_info.voice_registration_details_3gpp.lac,
            t_info.voice_registration_details_3gpp.psc,
            t_info.voice_registration_details_3gpp.tac);
    }
    if(t_info.data_registration_details_3gpp_valid)
    {
        printf("data_registration_details_3gpp: \ntech_domain=%s, radio_tech=%s, mcc=%s, mnc=%s,
roaming=%d, forbidden=%d, cid=0x%X, lac=%d, psc=%d, tac=%d\n",
            tech_domain[t_info.data_registration_details_3gpp.tech_domain],
            radio_tech[t_info.data_registration_details_3gpp.radio_tech],
            t_info.data_registration_details_3gpp.mcc,
            t_info.data_registration_details_3gpp.mnc,
            t_info.data_registration_details_3gpp.roaming,
            t_info.data_registration_details_3gpp.forbidden,
            t_info.data_registration_details_3gpp.cid,
            t_info.data_registration_details_3gpp.lac,
            t_info.data_registration_details_3gpp.psc,
            t_info.data_registration_details_3gpp.tac);
    }

    if(t_info.voice_registration_details_3gpp2_valid)
    {
        printf("voice_registration_details_3gpp2: \ntech_domain=%s, radio_tech=%s, mcc=%s, mnc=%s, roaming=%d, forbidden=%d, sid=%d, nid=%d, bsid=%d\n",
            tech_domain[t_info.voice_registration_details_3gpp2.tech_domain],
            radio_tech[t_info.voice_registration_details_3gpp2.radio_tech],
            t_info.voice_registration_details_3gpp2.mcc,
            t_info.voice_registration_details_3gpp2.mnc,
            t_info.voice_registration_details_3gpp2.roaming,
            t_info.voice_registration_details_3gpp2.forbidden,
            t_info.voice_registration_details_3gpp2.sid,
            t_info.voice_registration_details_3gpp2.nid,
            t_info.voice_registration_details_3gpp2.bsid);
    }

    if(t_info.data_registration_details_3gpp2_valid)
    {
        printf("data_registration_details_3gpp2: \ntech_domain=%s, radio_tech=%s, mcc=%s, mnc=%s, roaming=%d, forbidden=%d, sid=%d, nid=%d, bsid=%d\n",
            tech_domain[t_info.data_registration_details_3gpp2.tech_domain],
            radio_tech[t_info.data_registration_details_3gpp2.radio_tech],
            t_info.data_registration_details_3gpp2.mcc,
            t_info.data_registration_details_3gpp2.mnc,
            t_info.data_registration_details_3gpp2.roaming,
            t_info.data_registration_details_3gpp2.forbidden,
            t_info.data_registration_details_3gpp2.sid,
            t_info.data_registration_details_3gpp2.nid,
            t_info.data_registration_details_3gpp2.bsid);
    }
    break;
}
```

## 4.10. Set Low Power Mode

```
case 8://"QL_MCM_NW_SetLowPowerMode"
{
    int mode = 0;
    printf("please input low power mode(0: off, other: on): \n");
    scanf("%d", &mode);
    ret = QL_MCM_NW_SetLowPowerMode(h_nw, mode);
    printf("QL_MCM_NW_SetLowPowerMode ret = %d\n", ret);
    break;
}
```

## 4.11. Get Signal Strength Information

```
case 10://"QL_MCM_NW_GetSignalStrength"
{
    QL_MCM_NW_SIGNAL_STRENGTH_INFO_T    t_info;

    memset(&t_info, 0, sizeof(QL_MCM_NW_SIGNAL_STRENGTH_INFO_T));
    ret = QL_MCM_NW_GetSignalStrength(h_nw, &t_info);
    printf("QL_MCM_NW_GetSignalStrength ret = %d, detail info:\n", ret);

    if(t_info.gsm_sig_info_valid)
    {
        printf("gsm_sig_info: rssi=%d\n", t_info.gsm_sig_info.rssi);
    }

    if(t_info.wcdma_sig_info_valid)
    {
        printf("wcdma_sig_info: rssi=%d, ecio=%d\n",
            t_info.wcdma_sig_info.rssi,
            t_info.wcdma_sig_info.ecio);
    }

    if(t_info.tdscdma_sig_info_valid)
    {
        printf("tdscdma_sig_info: rssi=%d, rscp=%d, ecio=%d, sinr=%d\n",
            t_info.tdscdma_sig_info.rssi,
            t_info.tdscdma_sig_info.rscp,
            t_info.tdscdma_sig_info.ecio,
            t_info.tdscdma_sig_info.sinr);
    }

    if(t_info.lte_sig_info_valid)
    {
        printf("lte_sig_info: rssi=%d, rsrq=%d, rsrp=%d, snr=%d\n",
            t_info.lte_sig_info.rssi,
            t_info.lte_sig_info.rsrq,
            t_info.lte_sig_info.rsrp,
            t_info.lte_sig_info.snr);
    }

    if(t_info.cdma_sig_info_valid)
    {
        printf("cdma_sig_info: rssi=%d, ecio=%d\n",
            t_info.cdma_sig_info.rssi,
            t_info.cdma_sig_info.ecio);
    }

    if(t_info.hdr_sig_info_valid)
    {
        printf("hdr_sig_info: rssi=%d, ecio=%d, sinr=%d, io=%d\n",
            t_info.hdr_sig_info.rssi,
            t_info.hdr_sig_info.ecio,
            t_info.hdr_sig_info.sinr,
            t_info.hdr_sig_info.io);
    }

    break;
}
```

## 4.12. Get Cell Access Status

```
case 11://"QL_MCM_NW_GetCellAccessState"
{
    E_QL_MCM_NW_CELL_ACCESS_STATE_TYPE_T    e_state;
    ret = QL_MCM_NW_GetCellAccessState(h_nw, &e_state);
    printf("QL_MCM_NW_GetCellAccessState ret = %d, e_state=%d\n", ret, e_state);
    break;
}
```

## 4.13. Set Callback Function of Network Events

```
case 15 ://"QL_MCM_NW_AddRxMsgHandler"
{
    ret = QL_MCM_NW_AddRxMsgHandler(h_nw, nw_event_ind_handler, NULL);
    printf("QL_MCM_NW_AddRxMsgHandler, ret=%d\n", ret);
    break;
}
```

# 5 Appendix A References

Table 3: Related Document

| SN | Document Name | Description |
|---|---|---|
| [1] | Quectel_EC2x&EG9x&EG25-G_Series_QuecOpen_Quick_Start_Guide | Quick start guide for QuecOpen solution of EC2x series, EG9x series and EG25-G modules |

Table 4: Terms and Abbreviations

| Abbreviation | Description |
|---|---|
| 1xRTT | Single-Carrier Radio Transmission Technology |
| API | Application Programming Interface |
| ARFCN | Absolute Radio-Frequency Channel Number |
| ASU | Arbitrary Strength Unit |
| BSIC | Base Station Identity Code |
| CDMA | Code-Division Multiple Access |
| CSG | Closed Subscriber Group |
| EARFCN | E-UTRA Absolute Radio Frequency Channel Number |
| Ec/Io | Energy per chip to Interference power ratio |
| EDGE | Enhanced Data Rates for GSM Evolution |
| eHRPD | evolved High Rate Package Data |
| GPRS | General Packet Radio Service |
| GSM | Global System for Mobile Communications |
| HDR | High Data Rate |
| HLR | Home Location Register |

| | |
|---|---|
| HSPA | High Speed Packet Access |
| IMSI | International Mobile Subscriber Identity |
| LAC | Location Area Code |
| Long_eons | Long Enhanced Operator Name String |
| LTE | Long Time Evolution |
| MAC | Medium Access Control |
| MCC | Mobile Country Code |
| MNC | Mobile Network Code |
| PLMN | Public Land Mobile Network |
| PRL | Preferred Roaming List |
| PSC | Primary Scrambling Code |
| RSCP | Received Signal Code Power |
| RSRP | Reference Signal Received Power |
| RSRQ | Reference Signal Received Quality |
| RSSI | Received Signal Strength Indicator |
| SDK | Software Development Kit |
| Short_eons | Short Enhanced Operator Name String |
| SINR | Signal-to-Interference-plus-Noise Ratio |
| SNR | Signal-to-Noise Ratio |
| TAC | Tracking Area Code |
| TD-SCDMA | Time Division-Synchronous Code Division Multiple Access |
| UARFCN | UTRA Absolute RF Channel Number |
| UMTS | Universal Mobile Telecommunications System |
| UTRAN | UMTS Terrestrial Radio Access Network |
| VLR | Visitor Location Register |