EC2x&EG9x&EG25-G Series
QuecOpen GNSS API Reference Manual

LTE Standard Module Series

Version: 1.0

Date: 2021-02-08

Status: Released

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

About the Document

Revision History

Version	Date	Author	Description
-	2021-01-07	Navy QIU/ Arno WANG	Creation of the document
1.0	2021-02-08	Navy QIU/ Arno WANG	First official release

Contents

About the Document **3**
Contents **4**
Table Index **6**

1 Introduction **7**
1.1. Applicable Modules 7

2 GNSS Feature Overview **8**
2.1. Functional Characteristics 8
2.2. Performance Indicators 8

3 GNSS API **10**
3.1. Header File Location 10
3.2. Example Location 10
3.3. Enumerations 10
3.3.1. E_QL_LOC_NFY_MSG_ID_T 10
3.3.2. E_QL_LOC_DELETE_AIDING_DATA_TYPE_T 11
3.3.3. E_QL_LOC_AGPS_TYPE_T 13
3.3.4. E_QL_LOC_NI_USER_RESPONSE_TYPE_T 14
3.4. Structures 14
3.4.1. QL_LOC_INJECT_TIME_INTO_T 14
3.4.2. QL_LOC_INJECT_LOCATION_INTO_T 15
3.4.3. QL_LOC_AGPS_DATA_CONN_OPEN_INTO_T 15
3.4.4. QL_LOC_AGPS_SERVER_INTO_T 16
3.4.5. QL_LOC_POS_MODE_INFO_T 16
3.4.6. QL_LOC_LOCATION_INFO_T 17
3.4.7. QL_LOC_NI_RESPONSE_INTO_T 19
3.5. APIs Description 19
3.5.1. QL_LOC_Client_Init 20
3.5.2. QL_LOC_Client_Deinit 21
3.5.3. QL_LOC_AddRxIndMsgHandler 21
3.5.3.1. QL_LOC_RxIndMsgHandlerFunc_t 22
3.5.4. QL_LOC_Set_Indications 22
3.5.5. QL_LOC_Start_Navigation 23
3.5.6. QL_LOC_Stop_Navigation 24
3.5.7. QL_LOC_Set_Position_Mode 24
3.5.8. QL_LOC_Get_Current_Location 25
3.5.9. QL_LOC_Delete_Aiding_Data 25
3.5.10. QL_LOC_InjectTime 26
3.5.11. QL_LOC_InjectLocation 26
3.5.12. QL_LOC_Xtra_InjectData 27
3.5.13. QL_LOC_Xtra_InjectFile 28
3.5.14. QL_LOC_Agps_DataConnOpen 28

3.5.15. QL_LOC_Agps_DataConnClose 29
3.5.16. QL_LOC_Agps_NfyDataConnFailed 29
3.5.17. QL_LOC_Agps_SetServer 30
3.5.18. QL_LOC_NI_SetResponse 30
3.5.19. QL_LOC_Agps_UpdateNWAvailability 31

4 Example 32
4.1. Use Steps of GNSS APIs 32
4.2. Description of Example 32
4.3. Code of Example 34
4.4. Compilation Description 38

5 Appendix A References 39

Table Index

Table 1: Applicable Modules 7
Table 2: Performance Indicators 8
Table 3: APIs Overview 19
Table 4: Related Documents 39
Table 5: Terms and Abbreviations 39

1 Introduction

Quectel LTE Standard EC2x series, EG9x series and EG25-G modules support QuecOpen® solution. 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®, see ***document [1]***.

This document introduces how to realize the GNSS feature on Quectel LTE Standard EC2x series, EG9x series and EG25-G QuecOpen modules via GNSS APIs in SDK provided by Quectel.

GNSS is an abbreviation of Global Navigation Satellite System, which generally refers to all satellite navigation systems including global satellite navigation system (such as GPS, GLONASS, BeiDou, Galileo), area navigation system (such as QZSS from Japan and IRNSS from India) and enhanced satellite navigation system (such as WAAS, SDCM, EGNOS, MSAS, GAGAN).

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 GNSS Feature Overview

The Quectel EC2x series, EG9x series and EG25-G QuecOpen modules integrate a IZat Gen8C GNSS engine which support GPS, BeiDou, GLONASS and Galileo navigation system. Combined with the support of supports SUPL and XTRA assistance technologies, the modules provide a quicker and more accurate fix service.

This chapter offers the functional characteristics and performance indicators of Quectel EC2x series, EG9x series and EG25-G QuecOpen modules.

2.1. Functional Characteristics

- Support SBAS including WAAS, EGNOS, MSAS and GAGAN.
- Support XTRA rapid positioning technology.
- Support AGPS auxiliary positioning technology, such as SUPL.
- Support multi constellation positioning including GPS, GLONASS, BeiDou, Galileo and QZSS.
- Support point positioning and DGPS.
- Support to output positioning information in multiple frequency, such as 1 Hz, 2 Hz, 5 Hz and 10 Hz.

2.2. Performance Indicators

Table 2: Performance Indicators

Indicator	Performance	Notes
2D positioning accuracy (50%, 68%, 95%)	< 2 m, < 2.5 m, < 5 m	Stand-alone mode in open areas.
3D positioning accuracy (50%, 68%, 95%)	< 2.5 m, < 3 m, < 6 m	Stand-alone mode in open areas.
Number of Channels tracked simultaneously	40	-
TTFF after performing cold start	29 s	Stand-alone mode in open areas.

TTFF after performing warm start	27 s	Stand-alone mode in open areas.
TTFF after performing hot start	1 s	Stand-alone mode in open areas.
Signal recapture time after a loss of lock for 30s.	≈ 1 s	In open areas.
Signal recapture time after a loss of lock for 5 mins.	≈ 2 s	In open areas.
Acquisition sensitivity (cold start, 95%)	> -149 dBm	Cold start, the timeout is 300 s.
Tracking sensitivity	> -163 dBm	Stand-alone or MSB mode.
Velocity accuracy (68%, 95%)	0.15 m/s, 0.3 m/s	Straight driving at a speed of 30 m/s
Heading accuracy (68%, 95%)	0.2 deg 0.5 deg	Straight driving at a speed of 30 m/s
Maximum speed	1852 km/h	

NOTE

For details about positioning mode, please refer to ***Chapter 3.4.5***.

3 GNSS API

3.1. Header File Location

The interface header file is *ql-ol-sdk/ql-ol-extsdk/include/ql_mcm_gps.h*.

3.2. Example Location

The interface example is *ql-ol-sdk/ql-ol-extsdk/example/API/api_test_main.c*.

3.3. Enumerations

3.3.1. E_QL_LOC_NFY_MSG_ID_T

The enumeration of message ID and its corresponding message structure is defined as below:

```
typedef enum
{
    E_QL_LOC_NFY_MSG_ID_STATUS_INFO = 0,          /**<  pv_data = &E_QL_LOC_STATUS_VALUE_T  */
    E_QL_LOC_NFY_MSG_ID_LOCATION_INFO,            /**<  pv_data = &QL_LOC_LOCATION_INFO_T  */
    E_QL_LOC_NFY_MSG_ID_SV_INFO,                  /**<  pv_data = &QL_LOC_SV_STATUS_T  */
    E_QL_LOC_NFY_MSG_ID_NMEA_INFO,                /**<  pv_data = &QL_LOC_NMEA_INFO_T  */
    E_QL_LOC_NFY_MSG_ID_CAPABILITIES_INFO,        /**<  pv_data = &E_QL_LOC_CAPABILITIES_T  */
    E_QL_LOC_NFY_MSG_ID_AGPS_STATUS,              /**<  pv_data = &QL_LOC_AGPS_STATUS_T  */
    E_QL_LOC_NFY_MSG_ID_NI_NOTIFICATION,          /**<  pv_data = &QL_LOC_NI_NOTIFICATION_INTO_T  */
    E_QL_LOC_NFY_MSG_ID_XTRA_REPORT_SERVER,       /**<  pv_data = &QL_LOC_XTRA
```

```
_REPORT_SERVER_INTO_T   */
}E_QL_LOC_NFY_MSG_ID_T;
```

- **Parameter**

Parameter	Description
E_QL_LOC_NFY_MSG_ID_STATUS_INFO	GNSS status
E_QL_LOC_NFY_MSG_ID_LOCATION_INFO	Location information
E_QL_LOC_NFY_MSG_ID_SV_INFO	Visible satellites
E_QL_LOC_NFY_MSG_ID_NMEA_INFO	NMEA sentences
E_QL_LOC_NFY_MSG_ID_CAPABILITIES_INFO	Positioning mode
E_QL_LOC_NFY_MSG_ID_AGPS_STATUS	AGPS status
E_QL_LOC_NFY_MSG_ID_NI_NOTIFICATION	NI notification
E_QL_LOC_NFY_MSG_ID_XTRA_REPORT_SERVER	XTRA server

3.3.2. E_QL_LOC_DELETE_AIDING_DATA_TYPE_T

The enumeration of the type of GNSS auxiliary data to be deleted is defined as below:

```
typedef enum
{
    E_QL_LOC_DELETE_EPHEMERIS        = (1 <<  0),    /**<  Delete ephemeris data.  */
    E_QL_LOC_DELETE_ALMANAC          = (1 <<  1),    /**<  Delete almanac data.  */
    E_QL_LOC_DELETE_POSITION         = (1 <<  2),    /**<  Delete position data.  */
    E_QL_LOC_DELETE_TIME             = (1 <<  3),    /**<  Delete time data.  */
    E_QL_LOC_DELETE_IONO             = (1 <<  4),    /**<  Delete IONO data.  */
    E_QL_LOC_DELETE_UTC              = (1 <<  5),    /**<  Delete UTC data.  */
    E_QL_LOC_DELETE_HEALTH           = (1 <<  6),    /**<  Delete health data.  */
    E_QL_LOC_DELETE_SVDIR            = (1 <<  7),    /**<  Delete SVDIR data.  */
    E_QL_LOC_DELETE_SVSTEER          = (1 <<  8),    /**<  Delete SVSTEER data.  */
    E_QL_LOC_DELETE_SADATA           = (1 <<  9),    /**<  Delete SA data.  */
    E_QL_LOC_DELETE_RTI              = (1 << 10),    /**<  Delete RTI data.  */
    E_QL_LOC_DELETE_CELLDB_INFO      = (1 << 11),    /**<  Delete cell DB information.  */
    E_QL_LOC_DELETE_ALMANAC_CORR     = (1 << 12),    /**<  Delete almanac correction
data.  */
    E_QL_LOC_DELETE_FREQ_BIAS_EST    = (1 << 13),    /**<  Delete frequency bias estimate.
*/
    E_QL_LOC_DELETE_EPHEMERIS_GLO    = (1 << 14),    /**<  Delete ephemeris GLO data.
```

```
*/
    E_QL_LOC_DELETE_ALMANAC_GLO     = (1 << 15),    /**<  Delete almanac GLO data.  */
    E_QL_LOC_DELETE_SVDIR_GLO       = (1 << 16),    /**<  Delete SVDIR GLO data.  */
    E_QL_LOC_DELETE_SVSTEER_GLO     = (1 << 17),    /**<  Delete SVSTEER GLO data.
*/
    E_QL_LOC_DELETE_ALMANAC_CORR_GLO= (1 << 18),    /**<  Delete almanac correction
GLO data.  */
    E_QL_LOC_DELETE_TIME_GPS        = (1 << 19),    /**<  Delete time GPS data.  */
    E_QL_LOC_DELETE_TIME_GLO        = (1 << 20),    /**<  Delete time GLO data.  */
    E_QL_LOC_DELETE_ALL             = 0xFFFFFFFF,   /**<  Delete all location data.  */
}E_QL_LOC_DELETE_AIDING_DATA_TYPE_T;
```

- **Parameter**

Parameter	Description
E_QL_LOC_DELETE_EPHEMERIS	Delete ephemeris data
E_QL_LOC_DELETE_ALMANAC	Delete almanac data
E_QL_LOC_DELETE_POSITION	Delete location data
E_QL_LOC_DELETE_TIME	Delete time information
E_QL_LOC_DELETE_IONO	Delete IONO data
E_QL_LOC_DELETE_UTC	Delete UTC data
E_QL_LOC_DELETE_HEALTH	Delete health data
E_QL_LOC_DELETE_SVDIR	Delete SVDIR data
E_QL_LOC_DELETE_SVSTEER	Delete SVSTEER data
E_QL_LOC_DELETE_SADATA	Delete satellite data
E_QL_LOC_DELETE_RTI	Delete RTI data
E_QL_LOC_DELETE_CELLDB_INFO	Delete cell database data
E_QL_LOC_DELETE_ALMANAC_CORR	Delete almanac correction data
E_QL_LOC_DELETE_FREQ_BIAS_EST	Delete frequency bias estimate data
E_QL_LOC_DELETE_EPHEMERIS_GLO	Delete GLONASS ephemeris data
E_QL_LOC_DELETE_ALMANAC_GLO	Delete GLONASS almanac data
E_QL_LOC_DELETE_SVDIR_GLO	Delete GLONASS SVDIR data

E_QL_LOC_DELETE_SVSTEER_GLO	Delete GLONASS SVSTEER data
E_QL_LOC_DELETE_ALMANAC_CORR_GLO	Delete GLONASS almanac correction data
E_QL_LOC_DELETE_TIME_GPS	Delete GPS time data
E_QL_LOC_DELETE_TIME_GLO	Delete GLONASS time data
E_QL_LOC_DELETE_ALL	Delete all data

3.3.3. E_QL_LOC_AGPS_TYPE_T

The enumeration of AGPS protocol type is defined as below:

```
typedef enum
{
    E_QL_LOC_AGPS_TYPE_INVALID          = -1,    /**<  Invalid.  */
    E_QL_LOC_AGPS_TYPE_ANY              = 0,     /**<  Any.  */
    E_QL_LOC_AGPS_TYPE_SUPL             = 1,     /**<  SUPL.  */
    E_QL_LOC_AGPS_TYPE_C2K              = 2,     /**<  C2K.  */
    E_QL_LOC_AGPS_TYPE_WWAN_ANY         = 3,     /**<  WWAN any.  */
    E_QL_LOC_AGPS_TYPE_WIFI             = 4,     /**<  Wi-Fi.  */
    E_QL_LOC_AGPS_TYPE_SUPL_ES          = 5,     /**<  SUPL_ES.  */
}E_QL_LOC_AGPS_TYPE_T;
```

- **Parameter**

Parameter	Description
E_QL_LOC_AGPS_TYPE_INVALID	Invalid index
E_QL_LOC_AGPS_TYPE_ANY	Any AGPS protocol type
E_QL_LOC_AGPS_TYPE_SUPL	SUPL protocol
E_QL_LOC_AGPS_TYPE_C2K	C2K
E_QL_LOC_AGPS_TYPE_WWAN_ANY	Any WWAN
E_QL_LOC_AGPS_TYPE_WIFI	Wi-Fi
E_QL_LOC_AGPS_TYPE_SUPL_ES	SUPL_ES

3.3.4. E_QL_LOC_NI_USER_RESPONSE_TYPE_T

The enumeration of NI user response type is defined as below:

```
typedef enum
{
E_QL_LOC_NI_RESPONSE_ACCEPT       = 1,          /**<  Accept.  */
E_QL_LOC_NI_RESPONSE_DENY         = 2,          /**<  Deny.  */
E_QL_LOC_NI_RESPONSE_NORESP       = 3,          /**<  No response.  */
}E_QL_LOC_NI_USER_RESPONSE_TYPE_T;
```

- **Parameter**

Parameter	Description
E_QL_LOC_NI_RESPONSE_ACCEPT	Accept response
E_QL_LOC_NI_RESPONSE_DENY	Deny response
E_QL_LOC_NI_RESPONSE_NORESP	No response

3.4. Structures

3.4.1. QL_LOC_INJECT_TIME_INTO_T

The structure of the injected UTC time is defined as below:

```
typedef struct
{
    int64_t time;                 /**<  Inject time.*/
    int64_t time_reference;       /**<  Time reference.*/
    int32_t uncertainty;          /**<  Uncertainty.*/
}QL_LOC_INJECT_TIME_INTO_T;
```

- **Parameter**

Type	Parameter	Description
int64_t	*time*	Injected time. Unit: millisecond.
int64_t	*time_reference*	Time reference. Keep it as 0.
int32_t	*uncertainty*	Time accuracy. Keep it as 3500.

3.4.2. QL_LOC_INJECT_LOCATION_INTO_T

The structure of the location data to be injected is defined as below:

```
typedef struct
{
    double latitude;    /**<    Latitude.*/
    double longitude;   /**<    Longitude.*/
    float accuracy;     /**<    Accuracy.*/
}QL_LOC_INJECT_LOCATION_INTO_T;
```

- **Parameter**

Type	Parameter	Description
double	*latitude*	Latitude. Unit: degree.
double	*longitude*	Longitude. Unit: degree.
float	*accuracy*	Accuracy. Unit: meter.

3.4.3. QL_LOC_AGPS_DATA_CONN_OPEN_INTO_T

The structure of AGPS data connection is defined as below:

```
#define QL_LOC_APN_NAME_LENGTH_MAX 100
typedef struct
{
E_QL_LOC_AGPS_TYPE_T                e_agps_type;                /**<    AGPS type.*/
char                                apn[QL_LOC_APN_NAME_LENGTH_MAX + 1];/**<
    APN.*/
E_QL_LOC_AGPS_APN_BEARER_TYPE_T     e_bearer_type;              /**<    Bearer type.*/
}QL_LOC_AGPS_DATA_CONN_OPEN_INTO_T;
```

- **Parameter**

Type	Parameter	Description
E_QL_LOC_AGPS_TYPE_T	*e_agps_type*	AGPS protocol type
char	*apn*	APN
E_QL_LOC_AGPS_APN_BEARER_TYPE_T	*e_bearer_type*	Bearer type

3.4.4. QL_LOC_AGPS_SERVER_INTO_T

The structure of AGPS protocol type, server address and port is defined as below:

```
typedef struct
{
    E_QL_LOC_AGPS_TYPE_T    e_agps_type;                                    /**<   AGPS type.*/
    char                    host_name[QL_LOC_SEVER_ADDR_LENGTH_MAX + 1];/**<   Host name.*/
    uint32_t                port;                                           /**<   Port.*/
}QL_LOC_AGPS_SERVER_INTO_T;
```

- **Parameter**

Type	Parameter	Description
E_QL_LOC_AGPS_TYPE_T	*e_agps_type*	AGPS protocol type
char	*host_name*	Server address
uint32_t	*port*	Port

3.4.5. QL_LOC_POS_MODE_INFO_T

The structure of position configuration items is defined as below:

```
typedef struct
{
  E_QL_LOC_POS_MODE_T mode;     //Position mode. Only Stand-alone and MSB are supported.
E_QL_LOC_POS_RECURRENCE_T recurrence;      //Position recurrence mode.
  uint32_t                  min_interval;          //Positioning interval. Unit: millisecond.
uint32_t                preferred_accuracy;    //Horizontal positioning accuracy. Unit: meter.
uint32_t                preferred_time;        //Positioning timeout. Unit: millisecond.
}QL_LOC_POS_MODE_INFO_T;
```

- **Parameter**

Type	Parameter	Description
E_QL_LOC_POS_MODE_T	*mode*	Position mode. *E_QL_LOC_POS_MODE_STANDALONE*: Stand-alone mode *E_QL_LOC_POS_MODE_MS_BASED*: MSB mode (speed up positioning) *E_QL_LOC_POS_MODE_MS_ASSISTED*: MSA mode (not supported currently)
E_QL_LOC_POS_RECUR	*recurrence*	Position recurrence mode. *E_QL_LOC_POS_RECURRENCE_PERIODIC*: Periodic positioning

RENCE_T		*E_QL_LOC_POS_RECURRENCE_SINGLE*: Single positioning
uint32_t	*min_interval*	Positioning interval. Unit: millisecond. Valid values are 100, 200, 500, 1000, > 1000.
uint32_t	*preferred_accuracy*	Horizontal positioning accuracy. Unit: meter.
uint32_t	*preferred_time*	Positioning timeout. Unit: millisecond.

3.4.6. QL_LOC_LOCATION_INFO_T

The structure of location data is defined as below:

```
typedef struct
{
uint32_t       size;
E_QL_LOC_LOCATION_VALID_FLAG flags;
E_QL_LOC_ULP_LOCATION_SOURCE position_source;
double          latitude;
double          longitude;
double          altitude;
float           speed;
float           bearing;
float           accuracy;
int64_t         timestamp;
int32_t         is_indoor;
float           floor_number;
uint32_t        raw_data_len;
uint8_t         raw_data[QL_LOC_GPS_RAW_DATA_LEN_MAX];
char            map_url[QL_LOC_GPS_LOCATION_MAP_URL_SIZE + 1];
uint8_t         map_index[QL_LOC_GPS_LOCATION_MAP_IDX_SIZE];
}QL_LOC_LOCATION_INFO_T;
```

- **Parameter**

Type	Parameter	Description
uint32_t	*size*	Size of this structure.
E_QL_LOC_LOCATION_VALID_FLAG	*flags*	Data validity indication. If the mask is 1, it means the data in this structure is valid. The value is the combination of the following mask. E_QL_LOC_LOCATION_LAT_LONG_VALID: Longitude and latitude value is valid. E_QL_LOC_LOCATION_ALTITUDE_VALID: Altitude value is valid. E_QL_LOC_LOCATION_SPEED_VALID: Speed value is valid. E_QL_LOC_LOCATION_BEARING_VALID: Bearing value is valid. E_QL_LOC_LOCATION_ACCURACY_VALID: Positioning accuracy value is valid. E_QL_LOC_LOCATION_SOURCE_INFO_VALID: Source information is valid. E_QL_LOC_LOCATION_IS_INDOOR_VALID: Indoor mode is valid. E_QL_LOC_LOCATION_FLOOR_NUMBE_VALID: Floor number is valid. E_QL_LOC_LOCATION_MAP_URL_VALID: Map URL is valid. E_QL_LOC_LOCATION_MAP_INDEX_VALID: Map index is valid.
E_QL_LOC_ULP_LOCATION_SOURCE	*position_source*	Location data source.
double	*latitude*	Latitude. Range: -90–90. Unit: degree.
double	*longitude*	Longitude. Range: 0–180. Unit: degree.
double	*altitude*	Altitude. Unit: meter.
float	*speed*	Speed. Range: 0–540. Unit: m/s.
float	*bearing*	Bearing. Range: 0–360. Unit: degree.
float	*accuracy*	Horizontal accuracy. Unit: meter.
int64_t	*timestamp*	UTC time. Unit: millisecond.
int32_t	*Is_indoor*	Indoor or not. (Depends on the indication of E_QL_LOC_LOCATION_IS_INDOOR_VALID)

float	*floor_number*	Floor number. (Depends on the indication of E_QL_LOC_LOCATION_IS_INDOOR_VALID)
int32_t	*raw_data_len*	Length of raw data. Range: 0–256.
uint8_t	*raw_data*	Raw data. (Invalid data)
char	*map_url*	Map URL. (Depends on the indication of E_QL_LOC_LOCATION_MAP_URL_VALID)
uint8_t	*map_index*	Map index. (Depends on the indication of E_QL_LOC_LOCATION_MAP_INDEX_VALID)

3.4.7. QL_LOC_NI_RESPONSE_INTO_T

The structure of NI response information is defined as below:

```
typedef struct
{
int32_t                                notify_id;  /**<   Notification ID.*/
E_QL_LOC_NI_USER_RESPONSE_TYPE_T       user_resp;  /**<   User response.*/
}QL_LOC_NI_RESPONSE_INTO_T;
```

- **Parameter**

Type	Parameter	Description
int32_t	*notify_id*	Notification ID.
E_QL_LOC_NI_USER_RESPONSE_TYPE_T	*user_resp*	User response. See ***Chapter 3.3.4***.

3.5. APIs Description

Table 3: APIs Overview

Function	Description
QL_LOC_Client_Init	Initialize a GNSS client
QL_LOC_Client_Deinit	Deregister a GNSS client
QL_LOC_AddRxIndMsgHandler	Register a callback function to process GNSS data
QL_LOC_Set_Indications	Set indications of callback data

QL_LOC_Start_Navigation	Start GNSS
QL_LOC_Stop_Navigation	Stop GNSS
QL_LOC_Set_Position_Mode	Set position configuration items
QL_LOC_Get_Current_Location	Get current location data
QL_LOC_Delete_Aiding_Data	Delete GNSS auxiliary data
QL_LOC_InjectTime	Inject UTC time to GNSS
QL_LOC_InjectLocation	Inject location data to GNSS
QL_LOC_Xtra_InjectData	Inject XTRA auxiliary data to GNSS
QL_LOC_Xtra_InjectFile	Inject XTRA file to GNSS
QL_LOC_Agps_DataConnOpen	Notify that AGPS data connection is opened
QL_LOC_Agps_DataConnClose	Notify that AGPS data connection is closed
QL_LOC_Agps_NfyDataConnFailed	Notify the failure of starting a AGPS data connection
QL_LOC_Agps_SetServer	Set AGPS server address and port
QL_LOC_NI_SetResponse	Send NI user responses
QL_LOC_Agps_UpdateNWAvailability	Update network availability

NOTES

1. The interfaces introduced in this document are not applicable to the module that supports QDR & PPE. Please contact Quectel Technical Support for details.
2. Calling other interfaces in a callback function is not allowed.

3.5.1. QL_LOC_Client_Init

This function initializes a GNSS client to create a GNSS session.

- **Prototype**

```
int QL_LOC_Client_Init(loc_client_handle_type    *ph_loc);
```

- **Parameter**

ph_loc:
[Out] Handle that is returned after initializing a GNSS client and creating a GNSS session. This parameter is used in subsequent GNSS interfaces.

- **Return Value**

0 Create a GNSS session successfully.
Others Fail to create a GNSS session.

3.5.2. QL_LOC_Client_Deinit

This function deregisters a GNSS client to release the GNSS session.

- **Prototype**

```
int QL_LOC_Client_Deinit(loc_client_handle_type    h_loc);
```

- **Parameter**

h_loc:
[In] Handle that is returned after initializing a GNSS client and creating a GNSS session.

- **Return Value**

0 Release the session successfully.
Others Fail to release the session.

3.5.3. QL_LOC_AddRxIndMsgHandler

This function registers a callback function to process GNSS data. The receivable messages of the registered callback function are determined by *QL_LOC_Set_Indications*.

- **Prototype**

```
int QL_LOC_AddRxIndMsgHandler(QL_LOC_RxIndMsgHandlerFunc_t handlerPtr, void* contextPtr);
```

- **Parameter**

handlerPtr:
[In] The callback function used to process GNSS data.

contextPtr:
[In] The parameters required by the callback function. See ***Chapter 3.5.3.1***.

- **Return Value**

0 Register the callback function successfully.
Others Failed to register the callback function.

3.5.3.1. QL_LOC_RxIndMsgHandlerFunc_t

This callback function processes GNSS data. It reports the corresponding data when the event is occurred according to the configurations of *QL_LOC_Set_Indications* and *QL_LOC_Set_Position_Mode*.

- **Prototype**

```
typedef void (*QL_LOC_RxIndMsgHandlerFunc_t)
(
    loc_client_handle_type    h_loc,
    E_QL_LOC_NFY_MSG_ID_T     e_msg_id,
    void                      *pv_data,
    void                      *contextPtr
);
```

- **Parameter**

h_loc:
[Out] Handle that is returned after initializing a GNSS client and creating a GNSS session.

e_msg_id:
[Out] Message ID. See ***Chapter 3.3.1***.

pv_data:
[Out] Callback data. See ***Chapter 3.3.1***.

contextPtr:
[Out] Customized data. Callback tag.

- **Return Value**

None.

3.5.4. QL_LOC_Set_Indications

This function sets callback data.

- **Prototype**

```
int QL_LOC_Set_Indications(loc_client_handle_type h_loc, int bit_mask);
```

- **Parameter**

h_loc:
[In] Handle that is returned after initializing a GNSS client and creating a GNSS session.

bit_mask:
[In] Bit mask of callback data, and it is defined as below:

```
LOC_IND_LOCATION_INFO_ON                (1 << 0)      //Location data
LOC_IND_STATUS_INFO_ON                  (1 << 1)      //GNSS engine status data
LOC_IND_SV_INFO_ON                      (1 << 2)      //Satellites related data
LOC_IND_NMEA_INFO_ON                    (1 << 3)      //NMEA sentences
LOC_IND_CAP_INFO_ON                     (1 << 4)      //Calibration mode
LOC_IND_UTC_TIME_REQ_ON                 (1 << 5)      //Request of UTC time injection
LOC_IND_XTRA_DATA_REQ_ON                (1 << 6)      //Request of XTRA data injection
LOC_IND_AGPS_DATA_CONN_CMD_REQ_ON   (1 << 7)   //Enable data connection request
LOC_IND_NI_NFY_USER_RESP_REQ_ON     (1 << 8)      //NI notifies user to response the request
```

- **Return Value**

0 Set callback data successfully.
Others Fail to set callback data.

3.5.5. QL_LOC_Start_Navigation

This function turns on GNSS and starts obtaining navigation data.

- **Prototype**

```
int QL_LOC_Start_Navigation(loc_client_handle_type h_loc);
```

- **Parameter**

h_loc:
[In] Handle that is returned after initializing a GNSS client and creating a GNSS session.

- **Return Value**

0 Turn on GNSS successfully.
Others Fail to turn on GNSS.

3.5.6. QL_LOC_Stop_Navigation

This function turns off GNSS and stops obtaining navigation data.

- **Prototype**

```
int QL_LOC_Stop_Navigation(loc_client_handle_type h_loc);
```

- **Parameter**

h_loc:
[In] Handle that is returned after initializing a GNSS client and creating a GNSS session.

- **Return Value**

0 Turn off GNSS successfully.
Others Fail to turn off GNSS.

3.5.7. QL_LOC_Set_Position_Mode

This function sets position configuration items, such as navigation mode, data acquisition interval and accuracy.

- **Prototype**

```
int QL_LOC_Set_Position_Mode(loc_client_handle_type h_loc, QL_LOC_POS_MODE_INFO_T *pt_mode);
```

- **Parameter**

h_loc:
[In] Handle that is returned after initializing a GNSS client and creating a GNSS session.

pt_mode:
[In] Position configuration items. See ***Chapter 3.4.5***.

- **Return Value**

0 Set position configuration items successfully.
Others Fail to set position configuration items.

3.5.8. QL_LOC_Get_Current_Location

This function gets current location data. Timeout returns if the location data is not obtained within a specified time.

- **Prototype**

```
int QL_LOC_Get_Current_Location(loc_client_handle_type h_loc, QL_LOC_LOCATION_INFO_T *pt_loc_info, int timeout_sec);
```

- **Parameter**

h_loc:
[In] Handle that is returned after initializing a GNSS client and creating a GNSS session.

pt_loc_info:
[Out] Location data. See ***Chapter 3.4.6***.

timeout_sec:
[In] Positioning timeout. Unit: millisecond.

- **Return Value**

0 Get location data successfully.
-2 Timeout.

3.5.9. QL_LOC_Delete_Aiding_Data

This function deletes GNSS auxiliary data.

- **Prototype**

```
int QL_LOC_Delete_Aiding_Data( loc_client_handle_type h_loc, E_QL_LOC_DELETE_AIDING_DATA_TYPE_T flags);
```

- **Parameter**

h_loc:
[In] Handle that is returned after initializing a GNSS client and creating a GNSS session.

flags:
[In] Specific type of data to be deleted. See ***Chapter 3.3.2***.

- **Return Value**

0 Delete GNSS auxiliary data successfully.
Others Fail to delete GNSS auxiliary data.

NOTE

Please wait 1 to 3 seconds to execute other function after calling this function as it takes a certain time to delete data.

3.5.10. QL_LOC_InjectTime

This function injects UTC time to GNSS engine to determine whether the injected XTRA data is valid. Please inject UTC time before starting GNSS with *QL_LOC_Start_Navigation* to speed up positioning.

- **Prototype**

```
int QL_LOC_InjectTime( loc_client_handle_type h_loc, QL_LOC_INJECT_TIME_INTO_T *pt_info);
```

- **Parameter**

h_loc:
[In] Handle that is returned after initializing a GNSS client and creating a GNSS session.

pt_info:
[In] UTC time data to be injected. Unit: millisecond. See ***Chapter 3.4.1***.

- **Return Value**

0 Inject UTC time successfully.
Others Fail to inject UTC time.

NOTE

The difference between the injected time and current UTC time should be less than 10 seconds, otherwise, the positioning time will be increased.

3.5.11. QL_LOC_InjectLocation

This function injects location data to GNSS engine to speed up positioning.

- **Prototype**

```
int QL_LOC_InjectLocation( loc_client_handle_type h_loc, QL_LOC_INJECT_LOCATION_INTO_T *pt_info);
```

- **Parameter**

h_loc:
[In] Handle that is returned after initializing a GNSS client and creating a GNSS session.

pt_info:
[In] Location data to be injected. See ***Chapter 3.4.2***.

- **Return Value**

0 Inject location data successfully.
Others Fail to inject location data.

3.5.12. QL_LOC_Xtra_InjectData

This function injects XTRA auxiliary data to GNSS engine.

- **Prototype**

```
Int QL_LOC_Xtra_InjectData(loc_client_handle_type h_loc, char *data, int length);
```

- **Parameter**

h_loc:
[In] Handle that is returned after initializing a GNSS client and creating a GNSS session.

data:
[In] XTRA data.

length:
[In] Length of XTRA data.

- **Return Value**

0 Inject XTRA auxiliary data successfully.
Others Fail to inject XTRA auxiliary data.

NOTE

The module supports IPC mechanism. Due to the restriction on IPC mechanism, currently the maximum length of the XTRA data supported to be injected is 0xFC00.

3.5.13. QL_LOC_Xtra_InjectFile

This function injects XTRA file to GNSS engine.

- **Prototype**

```
int QL_LOC_Xtra_InjectFile( loc_client_handle_type  h_loc, char *filename);
```

- **Parameter**

h_loc:
[In] Handle that is returned after initializing a GNSS client and creating a GNSS session.

filename:
[In] Complete path of XTRA file.

- **Return Value**

0 Inject XTRA file successfully.
Others Fail to inject XTRA file.

NOTE

The module supports IPC mechanism. Due to the restriction on IPC mechanism, currently the maximum length of the XTRA file supported to be injected is 0xFC00.

3.5.14. QL_LOC_Agps_DataConnOpen

This function notifies that AGPS data connection is opened.

- **Prototype**

```
int QL_LOC_Agps_DataConnOpen(loc_client_handle_type h_loc, QL_LOC_AGPS_DATA_CONN_OPEN_INTO_T *pt_info);
```

- **Parameter**

h_loc:
[In] Handle that is returned after initializing a GNSS client and creating a GNSS session.

pt_info:
[In] Data connection information. See ***Chapter 3.4.3***.

- **Return Value**

0 Notify that the data connection is opened successfully.
Others Fail to notify that the data connection is opened.

3.5.15. QL_LOC_Agps_DataConnClose

This function notifies that AGPS data connection is closed.

- **Prototype**

```
int QL_LOC_Agps_DataConnClose(loc_client_handle_type h_loc, E_QL_LOC_AGPS_TYPE_T
    atype);
```

- **Parameter**

h_loc:
[In] Handle that is returned after initializing a GNSS client and creating a GNSS session.

atype:
[In] AGPS protocol type. See ***Chapter 3.3.3***.

- **Return Value**

0 Notify that the data connection is closed successfully.
Others Fail to notify that the data connection is closed.

3.5.16. QL_LOC_Agps_NfyDataConnFailed

This function notifies the failure of starting a AGPS data connection.

- **Prototype**

```
int QL_LOC_Agps_NfyDataConnFailed(loc_client_handle_type h_loc, E_QL_LOC_AGPS_TYPE_
T     atype);
```

- **Parameter**

h_loc:
[In] Handle that is returned after initializing a GNSS client and creating a GNSS session.

atype:
[In] AGPS protocol type. See ***Chapter 3.3.3***.

- **Return Value**

0 Notify the failure successfully.
Others Fail to notify the failure.

3.5.17. QL_LOC_Agps_SetServer

This function sets AGPS server address and port.

- **Prototype**

```
int QL_LOC_Agps_SetServer(loc_client_handle_type h_loc, QL_LOC_AGPS_SERVER_INTO_T *pt_info);
```

- **Parameter**

h_loc:
[In] Handle that is returned after initializing a GNSS client and creating a GNSS session.

pt_info:
[In] AGPS server address and port. See ***Chapter 3.4.4***.

- **Return Value**

0 Set AGPS server address and port successfully.
Others Fail to set AGPS server address and port.

3.5.18. QL_LOC_NI_SetResponse

This function sends NI user response.

- **Prototype**

```
int QL_LOC_NI_SetResponse(loc_client_handle_type h_loc, QL_LOC_NI_RESPONSE_INTO_T *pt_info);
```

- **Parameter**

h_loc:
[In] Handle that is returned after initializing a GNSS client and creating a GNSS session.

pt_info:
[In] AGPS NI response information. See ***Chapter 3.4.7***.

- **Return Value**

0 Send response successfully.
Others Fail to send response.

3.5.19. QL_LOC_Agps_UpdateNWAvailability

This function updates network availability.

- **Prototype**

```
int QL_LOC_Agps_UpdateNWAvailability(loc_client_handle_type h_loc, int available, const char *apn);
```

- **Parameter**

h_loc:
[In] Handle that is returned after initializing a GNSS client and creating a GNSS session.

available:
[In] Whether the network is available.

apn:
[In] Access point name.

- **Return Value**

0 Update network availability successfully.
Others Fail to update network availability.

4 Example

4.1. Use Steps of GNSS APIs

QuecOpen SDK provides examples (*example/API/api_test_main.c*) for reference. The following introduces how to use GNSS APIs.

- Case A: Report the related information to the application through the callback function.

1. Call *QL_LOC_Client_Init* to initialize GNSS client and create a GNSS session.
2. Call *QL_LOC_AddRxIndMsgHandler(pf_cb)* to register a callback function for processing GNSS data.
3. Call *QL_LOC_Set_Indications* to set callback data.
4. Call *QL_LOC_Set_Position_Mode* to set position configuration items.
5. Call *QL_LOC_Start_Navigation* to turn on GNSS and start obtaining navigation data.
6. Handle the events returned by the callback function *QL_LOC_RxIndMsgHandlerFunc_t*.
7. Call *QL_LOC_Stop_Navigation* to turn off GNSS and stops obtaining navigation data.
8. Call *QL_LOC_Client_Deinit* to deregister a GNSS client and release the GNSS session.

- Case B: Actively obtain the location data once.

1. Call *QL_LOC_Client_Init* to initialize GNSS client and create a GNSS session.
2. Call *QL_LOC_AddRxIndMsgHandler(pf_cb)* to register a callback function for processing GNSS data. (Optional)
3. Call *QL_LOC_Set_Indications* to set callback data. Set bit_mask=LOC_IND_LOCATION_INFO_ON.
4. Call *QL_LOC_Set_Position_Mode* to set recurrence mode to single positioning.
5. Call *QL_LOC_Get_Current_Location* to obtain current location data. If the function times out, the module returns current location data or the location data previously stored.
6. Call *QL_LOC_Client_Deinit* to deregister a GNSS client and release the GNSS session.

4.2. Description of Example

Execute the following command to run the example program *example_gps*.

```
root@mdm9607-perf:/#  ./example_gps
```

After the example program runs successfully, the module prints the following:

```
root@mdm9607-perf:/# ./example_gps
============== gps test start =============
please input test mode(0: sync_get_position_once, other:get_gps_info_by_cb): 1
Starting MCM RIL Services: done
[QL_MCM_Client_Init  529]:  mcm_client_init ret=0x2 with h_mcm=0x0 ==> Sleep 2s and Retry !
[QL_MCM_Client_Init  529]:  mcm_client_init ret=0x2 with h_mcm=0x0 ==> Sleep 2s and Retry !
[QL_MCM_Client_Init  536]: Client initialized successfully 0x3
[QL_MCM_Client_Init  546]:  mcm_client_init start up required service!
[ql_mcm_async_cb  252]: ####h_mcm=0x3  msg_id=0x800
[ql_mcm_client_srv_updown_async_cb  33]: ####h_mcm=0x3  msg_id=0x800
[loc_ind_cb  22]:
 ===== mcmlocservice UP ! =========
QL_LOC_Client_Init ret 0 with h_loc=3
QL_LOC_AddRxIndMsgHandler ret 0
Please input indication bitmask(NiNfy|AGPS|XTRA|UTC|CAP|NMEA|SV|Status|Location):
511       //511=0x1FF=01 1111 1111 indicates all bits are enabled
[ql_mcm_ind_cb  133]: ####h_mcm=0x3  msg_id=0x312
[ql_loc_rx_ind_msg_cb  8]: e_msg_id=4
[ql_mcm_ind_cb  133]: ####h_mcm=0x3  msg_id=0x312
[ql_loc_rx_ind_msg_cb  8]: e_msg_id=4
QL_LOC_Set_Indications ret 0
QL_LOC_Set_Position_Mode ret 0
QL_LOC_Start_Navigation ret=0
Wait and handle event ! You can input -1 to exit): [ql_mcm_ind_cb  133]: ####h_mcm=0x3  ms
g_id=0x30f
[ql_loc_rx_ind_msg_cb  8]: e_msg_id=0
[ql_mcm_ind_cb  133]: ####h_mcm=0x3  msg_id=0x30f
[ql_loc_rx_ind_msg_cb  8]: e_msg_id=0
[ql_mcm_ind_cb  133]: ####h_mcm=0x3  msg_id=0x30f
[ql_loc_rx_ind_msg_cb  8]: e_msg_id=0
[ql_mcm_ind_cb  133]: ####h_mcm=0x3  msg_id=0x310
[ql_loc_rx_ind_msg_cb  8]: e_msg_id=2
[ql_mcm_ind_cb  133]: ####h_mcm=0x3  msg_id=0x311
[ql_loc_rx_ind_msg_cb  8]: e_msg_id=3
NMEA info: timestamp=315964862708, length=17, nmea=$GPGSV,1,1,0,*65
[ql_mcm_ind_cb  133]: ####h_mcm=0x3  msg_id=0x311
[ql_loc_rx_ind_msg_cb  8]: e_msg_id=3
NMEA info: timestamp=315964862709, length=17, nmea=$GLGSV,1,1,0,*79
[ql_mcm_ind_cb  133]: ####h_mcm=0x3  msg_id=0x311
[ql_loc_rx_ind_msg_cb  8]: e_msg_id=3
NMEA info: timestamp=315964862709, length=29, nmea=$GPGSA,A,1,,,,,,,,,,,,,,,*1E
[ql_mcm_ind_cb  133]: ####h_mcm=0x3  msg_id=0x311
```

```
[ql_loc_rx_ind_msg_cb  8]: e_msg_id=3
NMEA info: timestamp=315964862709, length=24, nmea=$GPVTG,,T,,M,,N,,K,N*2C
[ql_mcm_ind_cb  133]: ####h_mcm=0x3  msg_id=0x311
[ql_loc_rx_ind_msg_cb  8]: e_msg_id=3
NMEA info: timestamp=315964862710, length=24, nmea=$GPRMC,,V,,,,,,,,,N*53
[ql_mcm_ind_cb  133]: ####h_mcm=0x3  msg_id=0x311
[ql_loc_rx_ind_msg_cb  8]: e_msg_id=3
NMEA info: timestamp=315964862710, length=25, nmea=$GPGGA,,,,,,0,,,,,,,,*66
//Make sure all information above can be printed correctly.
```

4.3. Code of Example

```
#include <ql_oe.h>

static void ql_loc_rx_ind_msg_cb(loc_client_handle_type  h_loc,
                                 E_QL_LOC_NFY_MSG_ID_T    e_msg_id,
                                 void                     *pv_data,
                                 void                     *contextPtr)
{
    QL_USER_LOG("e_msg_id=%d\n", e_msg_id);
    switch(e_msg_id)
    {//Obtain and handle the information corresponding to the message ID.
        case E_QL_LOC_NFY_MSG_ID_STATUS_INFO:
            break;
        case E_QL_LOC_NFY_MSG_ID_LOCATION_INFO:
        {
            QL_LOC_LOCATION_INFO_T *pt_location = (QL_LOC_LOCATION_INFO_T *)pv_dat
a;
            printf("**** flag=0x%X, Latitude = %f, Longitude=%f, accuracy = %f ****\n",
                    pt_location->flags,
                    pt_location->latitude,
                    pt_location->longitude,
                    pt_location->accuracy);
            break;
        }
        case E_QL_LOC_NFY_MSG_ID_SV_INFO:
            break;
        case E_QL_LOC_NFY_MSG_ID_NMEA_INFO:
        {
            QL_LOC_NMEA_INFO_T  *pt_nmea = (QL_LOC_NMEA_INFO_T  *)pv_data;
```

```
                printf("NMEA info: timestamp=%lld, length=%d, nmea=%s\n",
                        pt_nmea->timestamp, pt_nmea->length, pt_nmea->nmea);
                break;
            }
            case E_QL_LOC_NFY_MSG_ID_CAPABILITIES_INFO:
                break;
            case E_QL_LOC_NFY_MSG_ID_AGPS_STATUS:
                break;
            case E_QL_LOC_NFY_MSG_ID_NI_NOTIFICATION:
                break;
            case E_QL_LOC_NFY_MSG_ID_XTRA_REPORT_SERVER:
                break;
        }
    }

    void sync_get_position_once(void)
    {
        int                         ret             = E_QL_OK;
        int                         h_loc           = 0;
        int                         bitmask         = 0;
        QL_LOC_POS_MODE_INFO_T   t_mode          = {0};
        QL_LOC_LOCATION_INFO_T   t_loc_info    = {0};
        int                         timeout_sec = 60;

        ret = QL_LOC_Client_Init(&h_loc);
        printf("QL_LOC_Client_Init ret %d with h_loc=%d\n", ret, h_loc);

        ret = QL_LOC_AddRxIndMsgHandler(ql_loc_rx_ind_msg_cb, (void*)h_loc);
        printf("QL_LOC_AddRxIndMsgHandler ret %d\n", ret);

        bitmask = 1; //force set to 1 to get location only.

        ret = QL_LOC_Set_Indications(h_loc, bitmask);
        printf("QL_LOC_Set_Indications ret %d\n", ret);

        t_mode.mode                     = E_QL_LOC_POS_MODE_STANDALONE;
        t_mode.recurrence               = E_QL_LOC_POS_RECURRENCE_SINGLE;
        t_mode.min_interval             = 1000;
        t_mode.preferred_accuracy     = 50;
        t_mode.preferred_time           = 90;   //The parameter is adjustable.
        ret = QL_LOC_Set_Position_Mode(h_loc, &t_mode);
        printf("QL_LOC_Set_Position_Mode ret %d\n", ret);

        ret = QL_LOC_Get_Current_Location(h_loc, &t_loc_info, timeout_sec);
```

```
    printf(" QL_LOC_Get_Current_Location ret %d\n", ret);
    if(ret < 0)
    {
        if(ret == -2)
        {// -2: timeout, may need try again
            printf("QL_LOC_Get_Current_Location timeout, try again!\n");
        }
        else
        {
            printf("QL_LOC_Get_Current_Location Fail, ret %d\n", ret);
        }
    }
    else
    {
        printf("**** Latitude = %lf, Longitude=%lf, altitude=%lf, accuracy = %f ****\n",
                t_loc_info.latitude, t_loc_info.longitude, t_loc_info.altitude, t_loc_info.accuracy);
    }

    ret = QL_LOC_Client_Deinit(h_loc);
    printf("QL_LOC_Client_Deinit ret=%d\n", ret);

    return ;
}

void get_gps_info_by_cb(void)
{
    int                         ret             = E_QL_OK;
    int                         h_loc           = 0;
    int                         bitmask         = 0;
    QL_LOC_POS_MODE_INFO_T   t_mode         = {0};
    QL_LOC_LOCATION_INFO_T   t_loc_info   = {0};

    ret = QL_LOC_Client_Init(&h_loc);
    printf("QL_LOC_Client_Init ret %d with h_loc=%d\n", ret, h_loc);

    ret = QL_LOC_AddRxIndMsgHandler(ql_loc_rx_ind_msg_cb, (void*)h_loc);
    printf("QL_LOC_AddRxIndMsgHandler ret %d\n", ret);

    printf("Please input indication bitmask(NiNfy|AGPS|XTRA|UTC|CAP|NMEA|SV|Status|Location):\
n", ret);
    scanf("%d", &bitmask);  //You can set bitmask to enable corresponding callback message as
 required.

    /* Set what we want callbacks for */
```

```
    ret = QL_LOC_Set_Indications(h_loc, bitmask);
    printf("QL_LOC_Set_Indications ret %d\n", ret);

    t_mode.mode                 = E_QL_LOC_POS_MODE_STANDALONE;
    t_mode.recurrence           = E_QL_LOC_POS_RECURRENCE_PERIODIC;
    t_mode.min_interval         = 1000;
    t_mode.preferred_accuracy   = 50;
    t_mode.preferred_time       = 90;  //The parameter is adjustable.
    ret = QL_LOC_Set_Position_Mode(h_loc, &t_mode);
    printf("QL_LOC_Set_Position_Mode ret %d\n", ret);

    ret = QL_LOC_Start_Navigation(h_loc);
    printf("QL_LOC_Start_Navigation ret=%d\n", ret);

    while(1)
    {
        int finish_flag = 0;//Wait for the message and handle it in the callback function.
        printf("Wait and handle event ! You can input -1 to exit): ");
        scanf("%d", &finish_flag);
        if(finish_flag == -1)
        {
            break;
        }
    }
    ret = QL_LOC_Stop_Navigation(h_loc);
    printf("QL_LOC_Stop_Navigation ret=%d\n", ret);

    ret = QL_LOC_Client_Deinit(h_loc);
    printf("QL_LOC_Client_Deinit ret=%d\n", ret);
}


int main(int argc, char *argv[])
{
    int mode;

    printf("============== gps test start ============\r\n");
    printf("please input test mode(0: sync_get_position_once, other:get_gps_info_by_cb): ");
    scanf("%d", &mode);

    if(mode == 0)
    {
        sync_get_position_once();
    }
```

```
    else
    {
        get_gps_info_by_cb();
    }
    printf("================ gps test end =============\r\n");
}
```

4.4. Compilation Description

This section describes how to compile a single *example_voice.c*.

1. Execute the following command to unzip QuecOpen SDK.

```
tar -jxvf ql-ol-sdk.tar.bz2
```

2. Execute the following command to enter directory *ql-ol-sdk*.

```
cd ql-ol-sdk
```

3. Execute the following command to configure environment.

```
source ql-ol-crosstool/ql-ol-crosstool-env-init
```

NOTE

Please make sure that the version of QuecOpen SDK and the module's firmware is the same. Otherwise, an error may occur.

4. Execute the following command to enter the directory of the example program *example_gps*.

```
cd ql-ol-extsdk/example/example_gps
```

5. Execute the following commands to start compilation.

```
make clean;
make
```

6. After the successful compilation, the file is generated and stored under the directory of *example_gps*.

5 Appendix A References

Table 4: Related Documents

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

Table 5: Terms and Abbreviations

Abbreviation	Description
AGPS	Assisted Global Positioning System
API	Application Programming Interface
APN	Access Point Name
BeiDou	BeiDou Navigation Satellite System
DB	Data Base
DGPS	Differential Global Position System
EGNOS	European Geostationary Navigation Overlay Service
GAGAN	GPS Aided Geo Augmented Navigation
GLONASS	Global Navigation Satellite System in Russia
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
IRNSS	Indian Regional Navigation Satellite System
MSA	Mobile Station Assisted
MSAS	Multi-Functional Satellite Augmentation System

MSB	Mobile Station Based
NI	Network Initialed
NMEA	NMEA (National Marine Electronics Association) 0183 Interface Standard
PPE	Precise Positioning Engine
QDR	Qualcomm Dead Reckoning
QZSS	Quasi-Zenith Satellite System
RTI	Real Time Integration
SA	Satellite
SBAS	Satellite-Based Augmentation System
SDCM	System of Differential Correction and Monitoring
SDK	Software Development Kit
SUPL	Secure User Plane Location
SVDIR	Satellites Available Direction
SVSTEER	Satellites Available Steer
TTFF	Time To First Fix
UTC	Coordinated Universal Time
WAAS	Wide Area Augmentation System
WWAN	Wireless Wide Area Network
XTRA	eXTended Receiver Assistance