AG35 Series QuecOpen SMS API Reference Manual

Automotive Module Series

Version: 1.0

Date: 2020-09-18

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
1.0	2020-09-18	Solomon CUI	Initial

Contents

Table Index

1 Introduction

SMS (short message service) is a text messaging service component of most telephone, Internet, and mobile device systems. Quectel AG35 series module in QuecOpen® solution supports SMS in both Text and PDU modes.

This document introduces how to use the SMS APIs provided in the QuecOpen® SDK of Quectel AG35 series module to achieve the following features:

1. Getting the SMS center number and type.
2. Setting the SMS center number and type.
3. Sending a text message (including long text message).
4. Receiving a text message (including long text message).
5. Sending an SMS message in PDU mode.

2 SMS APIs

2.1. Header File Location

The interface header file *ql_mcm_sms.h* is located in *ql-ol-sdk/ql-ol-extsdk/include* directory. Unless otherwise specified, the header files mentioned in this document are in this directory by default.

2.2. API Overview

Table 1: API Overview

Function	Description
QL_SMS_Client_Init()	Initializes SMS to obtain the handle
QL_SMS_Client_Deinit()	Deregisters SMS
QL_SMS_GetSmsCenterAddress()	Gets the SMS center number and type
QL_SMS_SetSmsCenterAddress()	Sets the SMS center number and type
QL_SMS_Send_Sms()	Sends a text message
QL_SMS_AddRxMsgHandler()	Sets the callback function for text message receiving
QL_SMS_Send_SmsPdu()	Sends an SMS message in PDU mode

NOTE

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

2.3. API Description

2.3.1. QL_SMS_Client_Init

This function initializes SMS to obtain the handle.

- **Prototype**

```
int QL_SMS_Client_Init(sms_client_handle_type    *ph_sms);
```

- **Parameter**

ph_sms:
[Out] SMS handle pointer.

- **Return Value**

E_QL_SUCCESS	Initialized SMS to obtain the handle successfully.
Other values	Failed to initialize SMS. See *ql_mcm.h* for the error code.

NOTE

Before using any other SMS API function, call this function first to initialize SMS to obtain the handle.

2.3.2. QL_SMS_Client_Deinit

This function deregisters SMS.

- **Prototype**

```
int QL_SMS_Client_Deinit(sms_client_handle_type h_sms);
```

- **Parameter**

h_sms:
[In] SMS handle returned by *QL_SMS_Client_Init()*.

- **Return Value**

E_QL_SUCCESS	Deregistered SMS successfully.
Other values	Failed to deregister SMS. See *ql_mcm.h* for the error code.

NOTE

Call this function to deregister SMS and release the resource when SMS is no longer needed.

2.3.3. QL_SMS_GetSmsCenterAddress

This function gets the SMS center number and type.

- **Prototype**

```
int QL_SMS_GetSmsCenterAddress( sms_client_handle_type h_sms, ql_sms_service_center_cfg_t *get_sca_cfg);
```

- **Parameter**

h_sms:
[In] SMS handle returned by *QL_SMS_Client_Init()*.

get_sca_cfg:
[Out] SMS center number and type. See ***Chapter 2.3.3.1*** for details.

- **Return Value**

E_QL_SUCCESS	Obtained the SMS center number and type successfully.
Other values	Failed to obtain the SMS center number and type. See *ql_mcm.h* for the error code.

2.3.3.1. ql_sms_service_center_cfg_t

The SMS center number and type are defined as follows:

```
typedef struct
 {
   char service_center_addr[QL_SMS_MAX_ADDR_LENGTH + 1];              /**<    Address of the service center.*/
   uint8_t service_center_addr_type_valid;
   char service_center_addr_type[QL_SMS_MAX_SCA_TYPE_LENGTH + 1];     /**<    129 if the SMSC address does not start with a "+" character;
                                                                             145 if the SMSC address starts with a "+" character*/
} ql_sms_service_center_cfg_t;
```

- **Parameter**

Type	Parameters	Description
char	*service_center_addr*	SMS center number.
uint8_t	*service_center_addr_type_valid*	Whether *service_center_addr_type* is valid. 0 Invalid 1 Valid
char	*service_center_addr_type*	If the SMS center number starts with "+", enter "145", otherwise enter "129".

2.3.4. QL_SMS_SetSmsCenterAddress

This function sets the SMS center number and type.

- **Prototype**

```
int QL_SMS_SetSmsCenterAddress( sms_client_handle_type h_sms, ql_sms_service_center_cfg_t *set_sca_cfg);
```

- **Parameter**

h_sms:
[In] SMS handle returned by *QL_SMS_Client_Init()*.

set_sca_cfg:
[In] SMS center number and type. See ***Chapter 2.3.3.1*** for details.

- **Return Value**

E_QL_SUCCESS Set the SMS center number and type successfully.
Other values Failed to set the SMS center number and type. See *ql_mcm.h* for the error code.

NOTE

It is not recommended to use this function, as if the SMS center number is set incorrectly, SMS messages may not be sent successfully.

2.3.5. QL_SMS_Send_Sms

This function sends a text message.

- **Prototype**

```
int QL_SMS_Send_Sms(sms_client_handle_type h_sms, ql_sms_info_t *pt_sms_info);
```

- **Parameter**

h_sms:
[In] SMS handle returned by *QL_SMS_Client_Init()*.

pt_sms_info:
[In] SMS content and destination number. See ***Chapter 2.3.5.1*** for details.

- **Return Value**

E_QL_SUCCESS	Sent the text message successfully.
Other values	Failed to send the text message. See *ql_mcm.h* for the error code.

2.3.5.1. ql_sms_info_t

The text message information to be sent or received, including message content, destination number, etc. Details are defined as follows:

```
typedef struct
{
    /* If SMS is stored, it won't be parsed. You need to read it by yourself */
    E_QL_SMS_STORAGE_TYPE_T e_storage;          //Specify where to store this message
    E_QL_SMS_FORMAT_T format;
    E_QL_SMS_TYPE_T type;
    char src_addr[QL_SMS_MAX_ADDR_LENGTH];     //SMS center number string.
    int sms_data_len;
    char sms_data[QL_SMS_MAX_MT_MSG_LENGTH]; //SMS content, data format depends on
                                                format
    char timestamp[21];                        //Message time stamp (in text mode). String
                                                format: "yy/MM/dd,hh:mm:ss+/-TimeZone"
    uint8_t user_data_head_valid;              //Indicates whether long SMS message is valid.
                                                TRUE-long message; FALSE-short message;
    ql_sms_user_data_head_t   user_data_head;  //Long SMS user data head info.
    E_QL_SMS_MODE_TYPE_T e_mode;               //Specify the SMS message mode.
    uint32_t storage_index;                    //Storage index. -1 means not store.
} ql_sms_info_t;
```

- **Parameter**

Type	Parameters	Description
E_QL_SMS_STORAGE_TYPE_T	*e_storage*	SMS message storage type. *E_QL_SMS_STORAGE_TYPE_NONE*: Not stored *E_QL_SMS_STORAGE_TYPE_UIM*: Stored in UIM *E_QL_SMS_STORAGE_TYPE_NV*: Stored in NVM
E_QL_SMS_FORMAT_T	*format*	SMS message format. *E_QL_SMS_FORMAT_GSM_7BIT*: GSM 7-bit *E_QL_SMS_FORMAT_BINARY_DATA*: Binary SMS *E_QL_SMS_FORMAT_UCS2*: UCS-2 encoding *E_QL_SMS_FORMAT_IRA*: Not supported
E_QL_SMS_TYPE_T	*type*	SMS type. *E_QL_SMS_TYPE_RX*: Received SMS message *E_QL_SMS_TYPE_TX*: Sent SMS message *E_QL_SMS_TYPE_BROADCAST_RX*: Received broadcast SMS message
char	*src_addr*	SMS center number.
int	*sms_data_len*	SMS message length.
char	*sms_data*	SMS message content.
char	*timestamp*	SMS message timestamp. Format: yy/MM/dd,hh:mm:ss+/-TimeZone
uint8_t	*user_data_head_valid*	Whether *user_data_head* is valid. TRUE Valid FALSE Invalid
ql_sms_user_data_head_t	*user_data_head*	Header information of a long text message. See ***Chapter 2.3.5.2*** for details.
E_QL_SMS_MODE_TYPE_T	*e_mode*	SMS message mode. *E_QL_SMS_MESSAGE_MODE_UNKNOWN*: Unknown *E_QL_SMS_MESSAGE_MODE_CDMA*: CDMA *E_QL_SMS_MESSAGE_MODE_GW*: GSM & WCDMA
uint32_t	*storage_index*	Storage index. *-1* indicates not stored.

2.3.5.2. ql_sms_user_data_head_t

The header information of a long text message is defined as follows:

```
typedef struct
 {
```

```
    uint8_t total_segments;      /**<      The number of long short message*/
    uint8_t seg_number;          /**<      Current number.*/
    uint8_t reference_number;    /**< reference_number.*/
}ql_sms_user_data_head_t;
```

- **Parameter**

Type	Parameter	Description
uint8_t	*total_segments*	The total number of segments of the long text message.
uint8_t	*seg_number*	The segment number of the current message.
uint8_t	*reference_number*	The receiving number of the current message, which uniquely identifies this message.

2.3.6. QL_SMS_AddRxMsgHandler

This function sets the callback function for text message receiving.

- **Prototype**

```
int QL_SMS_AddRxMsgHandler(QL_SMS_RxMsgHandlerFunc_t handlerPtr, void* contextPtr);
```

- **Parameter**

handlerPtr:
[In] Callback function of text message receiving. See ***Chapter 2.3.6.1*** for details.

contextPtr:
[In] Context information. Passed as a parameter to the callback function.

- **Return Value**

E_QL_SUCCESS Set the callback function for text message receiving successfully.
Other values Failed to set the callback function. See *ql_mcm.h* for the error code.

2.3.6.1. QL_SMS_RxMsgHandlerFunc_t

The callback function for text message receiving is defined as follows:

```
typedef void (*QL_SMS_RxMsgHandlerFunc_t)
(
    QL_SMS_MsgRef          msgRef,
```

```
    void*                       contextPtr
);
```

- **Parameter**

msgRef:
[In] Text message. See ***Chapter 2.3.5.1*** for details.

contextPtr:
[In] Context information. Pass it in when you set the callback function for text message receiving.

2.3.7. QL_SMS_Send_SmsPdu

This function sends an SMS message in PDU mode.

- **Prototype**

```
int QL_SMS_Send_SmsPdu( sms_client_handle_type h_sms, ql_wms_send_raw_message_data_t *raw_message_data, ql_wms_raw_send_resp_t *rawresp);
```

- **Parameter**

h_sms:
[In] SMS handle returned by *QL_SMS_Client_Init()*.

raw_message_data:
[In] SMS message in PDU mode. See ***Chapter 2.3.7.1*** for details.

rawresp:
[In] Result of sending the SMS message in PDU mode. See ***Chapter 2.3.7.2*** for details.

- **Return Value**

E_QL_SUCCESS	Sent the SMS message in PDU mode successfully.
Other values	Failed to send the SMS message in PDU mode. See *ql_mcm.h* for the error code.

2.3.7.1. ql_wms_send_raw_message_data_t

The SMS message in PDU mode, including the message format, length and content, is defined as follows:

```
typedef struct
 {
   E_QL_WMS_MESSAGE_FORMAT_TYPE format;
```

```
    uint32_t raw_message_len;                   /**< Must be set to # of elements in raw_message */
    uint8_t raw_message[QL_WMS_MESSAGE_LENGTH_MAX];          /**< Raw message data*/
}ql_wms_send_raw_message_data_t;
```

- **Parameter**

Type	Parameter	Description
E_QL_WMS_MESSAGE_FORMAT_TYPE	*format*	Format of the SMS message in PDU mode. *E_QL_WMS_MESSAGE_FORMAT_CDMA*: CDMA *E_QL_WMS_MESSAGE_FORMAT_GW_PP*: GSM & WCDMA – Point-to-point
uint32_t	*raw_message_len*	Length of the SMS message in PDU mode.
uint8_t	*raw_message*	Content of the SMS message in PDU mode.

2.3.7.2. ql_wms_raw_send_resp_t

The result of sending the SMS message in PDU mode is defined as follows:

```
typedef struct
 {
   uint16_t         message_id;              /*    Message ID */
   uint8_t          cause_code_valid;        /**< Must be set to true if cause_code is being passed */
   E_QL_WMS_TL_CAUSE_CODE_TYPE                cause_code;
}ql_wms_raw_send_resp_t;
```

- **Parameter**

Type	Parameter	Description
uint16_t	*message_id*	ID of this message in PDU mode.
uint8_t	*cause_code_valid*	Whether *cause_code* is valid. 0 Invalid 1 Valid
E_QL_WMS_TL_CAUSE_CODE_TYPE	*cause_code*	For details of *cause_code*, see the definition in *ql_mcm.h*.

3 SMS API Usage Examples

All examples in this chapter are available in *ql-ol-sdk/ql-ol-extsdk/example/test_mcm_api/test_sms.c*. You can view the complete examples of APIs anytime in the directory listed.

1. After the program starts, *QL_SMS_Client_Init()* must be called to initialize SMS to obtain the handle.
2. Before the program exits or the SMS is no longer used, *QL_SMS_Client_Deinit()* must be called to release resources.

3.1. Get the SMS Center Number and Type

```
case 7://"QL_SMS_GetSmsCenterAddress"
        {
            ql_sms_service_center_cfg_t get_sca_cfg;

            memset(&get_sca_cfg, 0, sizeof(get_sca_cfg));

            ret = QL_SMS_GetSmsCenterAddress(h_sms, &get_sca_cfg);
            printf("QL_SMS_GetSmsCenterAddress ret=%d \n", ret);
            if (E_QL_SUCCESS == ret)
            {
                printf("SCA: %s\n", get_sca_cfg.service_center_addr);
                if (get_sca_cfg.service_center_addr_type_valid)
                {
                    printf("SCA type: %s\n", get_sca_cfg.service_center_addr_type);
                }
            }

            break;
        }
```

3.2. Send Text Messages

The code below shows how to send text messages in GSM-7 bit, binary and UCS-2 formats.

```
case 1://"QL_SMS_Send_Sms"
        {
            int                     i          = 0;
            int                     len        = 0;
            E_QL_SMS_FORMAT_T     e_format= 0;
            char                    sms_buf[QL_SMS_MAX_MT_MSG_LENGTH] = {0};
            ql_sms_info_t           *pt_sms_info = NULL;

            pt_sms_info = (ql_sms_info_t*)malloc(sizeof(ql_sms_info_t));
            if(pt_sms_info == NULL)
            {
                printf("Malloc fail!\n");
                break;
            }
            memset(pt_sms_info, 0, sizeof(ql_sms_info_t));

            printf("please input dest phone number: \n");
            scanf("%s", pt_sms_info->src_addr);

            printf("please input sms encoding type(0:GSM-7, 1:Binary, 2:UCS2): \n");
            scanf("%d", &e_format);
            e_format = e_format & 0x03;

            if((e_format == E_QL_SMS_FORMAT_GSM_7BIT) || (e_format ==
E_QL_SMS_FORMAT_UCS2))
            {
                printf("please input message content: \n");
                getchar();//Add this on purpose or the following fgets will be skipped.
                fgets(sms_buf, QL_SMS_MAX_MT_MSG_LENGTH, stdin);
                len= strlen(sms_buf); //to skip 0x0A
                sms_buf[len-1] = '\0';
                printf("textData[%d]:%s\n", len, sms_buf);

                printf("input %d byte data:", len);
                for(i=0; i<len; i++)
                {
                    printf("%.2X ", sms_buf[i]);
                }
                printf("\n");
            }
            else
            {
                printf("please input binary data counts in bytes: \n");
                do
```

```
            {
                i = scanf("%d", &len);
            }while(i != 1);

            printf("please input binary data: \n");
            for(i=0; i<len; i++)
            {
                printf("Byte[%d]=", i);
                scanf("%2X", &sms_buf[i]);
            }
        }

        if(e_format == E_QL_SMS_FORMAT_GSM_7BIT)
        {
            memcpy(pt_sms_info->sms_data, sms_buf, QL_SMS_MAX_MT_MSG_LENGTH);
            pt_sms_info->sms_data_len = strlen(sms_buf);
        }
        else if(e_format == E_QL_SMS_FORMAT_BINARY_DATA)
        {
            memcpy(pt_sms_info->sms_data, sms_buf, len);
            pt_sms_info->sms_data_len = len;
        }
        else
        {
            len = UTF8StrToUnicodeStr(sms_buf,
                                      (uint16_t*)pt_sms_info->sms_data,
                                      len);// Got BE data
            pt_sms_info->sms_data_len = len * 2;//in bytes

            printf("raw UTF-16 len=%d, data:", pt_sms_info->sms_data_len);
            for(i=0; i<pt_sms_info->sms_data_len; i++)
            {
                printf("%.2X ", pt_sms_info->sms_data[i]);
            }
            printf("\n");
        }

        pt_sms_info->format = e_format;

        ret = QL_SMS_Send_Sms(h_sms, pt_sms_info);
        printf("#QL_SMS_Send_Sms ret=%d \n", ret);
        free(pt_sms_info);
        break;
    }
```

3.3. Receive Text Messages

The code below shows the use case of callback function for text message receiving:

```
static void ql_sms_cb_func( QL_SMS_MsgRef     msgRef,
                            void*             contextPtr)
{
    int i;
    if(msgRef->e_storage != E_QL_SMS_STORAGE_TYPE_NONE)
    {
        char *msg_format[]    = {"CDMA",    "GW"};
        char *storage_type[]= {"UIM",     "NV"};
        printf("###You've got one new %s message, stored to %s index=%d\n",
                    msg_format[msgRef->e_mode & 1],
                    storage_type[msgRef->e_storage & 1],
                    msgRef->storage_index);
    }
    else if(msgRef->format == E_QL_SMS_FORMAT_UCS2)
    {
        unsigned char* smsbuf = NULL;

        smsbuf = (char*)malloc(sizeof(char)*QL_SMS_MAX_MT_MSG_LENGTH);
        if(smsbuf == NULL)
        {
            printf("Out of memory");
            return;
        }
        memset(smsbuf, 0, QL_SMS_MAX_MT_MSG_LENGTH);
        UnicodeStrToUTF8Str((unsigned short*)(&msgRef->sms_data[0]),
                            smsbuf,
                            QL_SMS_MAX_MT_MSG_LENGTH) ;
        if (msgRef->user_data_head_valid)
        {
            printf("\n###You've got one new UCS2 msg from %s at %s, total_segments:%d,
seg_number:%d, reference_number:%02x, len=%d, content=%s\n",
                            msgRef->src_addr,
                            msgRef->timestamp,
                            msgRef->user_data_head.total_segments,
                            msgRef->user_data_head.seg_number,
                            msgRef->user_data_head.reference_number,
                            msgRef->sms_data_len,
                            smsbuf);
        }
```

```
        else
        {
            printf("\n###You've got one new UCS2 msg from %s at %s, len=%d, content=%s\n",
                                msgRef->src_addr,
                                msgRef->timestamp,
                                msgRef->sms_data_len,
                                smsbuf);
        }
        printf("Received UCS raw data:");
        for(i=0; i<msgRef->sms_data_len; i++)
        {
            printf("%.2X ", msgRef->sms_data[i]);
        }

        printf("\nAfter convert to UTF8, len=%d, data:", strlen(smsbuf));
        for(i=0; i<strlen(smsbuf); i++)
        {
            printf("%.2X ", smsbuf[i]);
        }
        printf("\n");
        free(smsbuf);
    }
    else if(msgRef->format == E_QL_SMS_FORMAT_BINARY_DATA)
    {
        if (msgRef->user_data_head_valid)
        {
            printf("###You've got one new BINARY msg from %s at %s , total_segments:%d,
seg_number:%d, reference_number:%02x, len=%d, content=",
                                msgRef->src_addr,
                                msgRef->timestamp,
                                msgRef->user_data_head.total_segments,
                                msgRef->user_data_head.seg_number,
                                msgRef->user_data_head.reference_number,
                                msgRef->sms_data_len);
        }
        else
        {
            printf("###You've got one new BINARY msg from %s at %s , len=%d, content=",
                                msgRef->src_addr,
                                msgRef->timestamp,
                                msgRef->sms_data_len);
        }
        for(i=0; i<msgRef->sms_data_len; i++)
        {
```

```
            printf("%.2X ", msgRef->sms_data[i]);
        }
        printf("\n");
    }
    else //default is GSM-7
    {
        if (msgRef->user_data_head_valid)
        {
            printf("###You've got one new GSM-7 msg from %s at %s, total_segments:%d,
seg_number:%d, reference_number:%02x, content=%s\n",
                                msgRef->src_addr,
                                msgRef->timestamp,
                                msgRef->user_data_head.total_segments,
                                msgRef->user_data_head.seg_number,
                                msgRef->user_data_head.reference_number,
                                msgRef->sms_data);
        }
        else
        {
            printf("###You've got one new GSM-7 msg from %s at %s, content=%s\n",
                                msgRef->src_addr,
                                msgRef->timestamp,
                                msgRef->sms_data);
        }
        for(i=0; i<msgRef->sms_data_len; i++)
        {
            printf("%.2X ", msgRef->sms_data[i]);
        }
        printf("\n\n");
    }
}
```

The code below shows how to set the callback function for text message receiving:

```
case 3://"QL_SMS_AddRxMsgHandler"
        {
            ret = QL_SMS_AddRxMsgHandler(ql_sms_cb_func, (void*)h_sms);
            printf("QL_SMS_AddRxMsgHandler ret=%d \n", ret);
            break;
        }
```

3.4. Send PDU SMS

```
case 8://"QL_SMS_Send_SmsPdu"
        {
            ql_wms_send_raw_message_data_t raw_message_data;
            uint8_t raw_message[] = {
                0x00, 0x01, 0x00, 0x0b, 0x81, 0x81, 0x52, 0x06, 0x22, 0x61,
                0xf5, 0x00, 0x00, 0x06, 0x79, 0x74, 0x3e, 0x26, 0x9b, 0x01};
            ql_wms_raw_send_resp_t rawresp;


            memset(&raw_message_data, 0, sizeof(raw_message_data));
            memset(&rawresp, 0, sizeof(rawresp));

            raw_message_data.format = E_QL_WMS_MESSAGE_FORMAT_GW_PP;
            raw_message_data.raw_message_len = sizeof(raw_message);
            memcpy(raw_message_data.raw_message, raw_message, sizeof(raw_message));

            ret = QL_SMS_Send_SmsPdu(h_sms, &raw_message_data, &rawresp);
            printf("QL_SMS_Send_SmsPdu ret=%d \n", ret);
            printf("message id: %d\n", rawresp.message_id);
            if (rawresp.cause_code_valid)
            {
                printf("cause code: %#x\n", rawresp.cause_code);
            }
            break;
        }
```

4 Appendix A References

Table 2: Terms and Abbreviations

Abbreviation	Description
ASCII	American Standard Code for Information Interchange
CDMA	Code Division Multiple Access
GSM	Global System for Mobile Communications
ISO	International Standard Organization
NVM	Non-Volatile Memory
PDU	Protocol Data Unit
SDK	Software Development Kit
SMS	Short Message Service
UCS	Universal Character Set
UIM	User Identity Module
WCDMA	Wideband CDMA