FN980M Appzone Linux

SDK User Guide

1VV0301722 Rev. 2 – 2022-06-13

Telit Technical Documentation

APPLICABILITY TABLE

Products	SW Version	Module
FN980M Series	38.02.xx1	5G

CONTENTS

1. INTRODUCTION

1.1. Scope

This document describes the Appzone Linux API, also referred to as the Mobile Connection Manager (MCM) API. The MCM API allows a subset of services provided by Qualcomm's MDM chipsets to be accessible to Linux® applications.

Telit also add proprietary APIs for features which are not provided by Qualcomm's MCM.

1.2. Audience

This document is intended for software developers who will be using the MCM API.

This document provides the public interfaces necessary to use the features provided by the MCM API.

A functional overview and information on leveraging the interface functionality are also provided. This document assumes that the user is familiar with Linux programming.

1.3. Contact Information, Support

For technical support and general questions please e-mail:

- *TS-EMEA@telit.com*
- *TS-AMERICAS@telit.com*
- *TS-APAC@telit.com*
- *TS-SRD@telit.com*
- *TS-ONEEDGE@telit.com*

Alternatively, use:

https://www.telit.com/contact-us

Product information and technical documents are accessible 24/7 on our website:

https://www.telit.com

1.4. Symbol Conventions

Danger: This information MUST be followed or catastrophic equipment failure or personal injury may occur.

Warning: Alerts the user on important steps about the module integration.

Note/Tip: Provides advice and suggestions that may be useful when integrating the module.

Electro-static Discharge: Notifies the user to take proper grounding precautions before handling the product.

All dates are in ISO 8601 format, that is YYYY-MM-DD.

2. REQUIREMENTS

The following are the requirements for working with the Appzone Linux environment:

- Ubuntu 14.04 OS

Ubuntu 14.04 is a Linux-based computer operating system. Ubuntu will be used for developing the Appzone Linux Applications

- GCC 4.6.3 compiler

The GNU Compiler Collection (GCC) is a compiler system produced by the GNU Project. The GCC compiler will be used for compiling the connection manager applications.

Note: GCC 4.6.3 is built in the Ubuntu 12.04 OS.

- GDB

The GNU Debugger (GDB) is the standard debugger for the GNU operating system. The GDB debugger will be used for debugging the connection manager applications.

- ADB

The Android Debug Bridge (ADB) is a command-line tool to assist in debugging Android-powered devices. The ADB will be used for loading and running the Appzone Linux applications into the Fn980m module.

3. STANDALONE SDK

3.1. Initial Configuration

3.1.1. Installing ADB

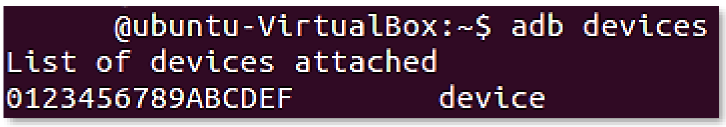
1. To install ADB on your system, use the following commands:
   - sudo add-apt-repository ppa:phablet-team/tools && sudo apt-get update
   - sudo apt-get install android-tools-adb android-tools-fastboot
   - Stop the adb server using the command:

     ```
     adb kill-server
     ```
   - Start the adb server using the command:

     ```
     adb start-server
     ```
2. To ensure that ADB is installed on your computer, connect your SDX55 module to the computer and run the following command:

   ```
   adb devices
   ```

   Example:

   ```
         @ubuntu-VirtualBox:~$ adb devices
   List of devices attached
   0123456789ABCDEF          device
   ```

   Ensure that your ADB device is recognized by the Linux system.

3.2. SDK Overview and Content

The Stand-alone SDK does not include a complete application development environment. It includes only the core SDK, which you can access from a command line interface (CLI) or with a plugin of your favorite IDE (if available).

The Stand-alone SDK consists of the following:

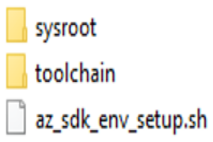
- **Toolchain:** Contains the set of tools that compiles source code into executables that can run on target devices. This also includes a compiler, a linker, and run-time libraries.
- **Sysroot:** Contains header files and libraries required for the build.
- **Setup script:** *az_sdk_env_setup.sh* file when executed, sets the development environment for the target device.

3.3. Installing the SDK

1. To install the SDK, unzip **<SDK version>_AZSDK .tar.gz** standalone package in the specific directory on your host development machine.
2. To unzip the SDK package, use the commands below.

```
user@host:~$ mkdir AZ_SDK
user@host:~$ cd AZ_SDK
user@host:~/AZ_SDK$ tar –zxf ../FN980M_38.02.XXX_AZSDK.tar.gzgz
user@host:~/AZ_SDK$ ls
```

3. After unzipping, the following files will be present in the directory as explained above.



sysroot
toolchain
az_sdk_env_setup.sh

Warning: AZ SDK version must match with target SW version. Because the sysroot image contains libraries, headers, and symbols specific to the target SW version. Please refer to Applicability Table for the latest AZ SDK and target SW version information.

3.4. Setting up the SDK

1. Run the SDK environment setup script (az_sdk_env_setup.sh).
2. Setup the SDK with the Source command:

```
user@host:~/AZ_SDK$ source az_sdk_env_setup.sh
-------------------------------------------------------------------
AZ SDK ENVIORNMENT
AZ SDK VERSION   : 38.00.X00-B025
AZ SDK PATH      : /home/user/AZ_SDK
AZ SDK SYSROOT   : /home/user/AZ_SDK/sysroot
AZ SDK TOOLCHAIN : /home/user/AZ_SDK/toolchain
-------------------------------------------------------------------
```

3. When you run the setup script, the following environment variables are defined:

Environment variables	Description
AZ_SDK_SYSROOT	The path to the sysroot contains target-specific libraries, header files
AZ_SDK_TOOLCHAIN	The path to the cross-development toolchain
AZ_M2MB_LIBS	The list to M2MB API library name
CC	The command and arguments to run the C compiler
CXX	The command and arguments to run the C++ compiler
CPP	The command and arguments to run the C preprocessor
AS	The command and arguments to run the assembler
LD	The command and arguments to run the linker
GDB	The command and arguments to run the GNU Debugger
STRIP	The command and arguments to run 'strip', which strips symbols
RANLIB	The command and arguments to run 'ranlib'
OBJCOPY	The command and arguments to run 'objcopy'
OBJDUMP	The command and arguments to run 'objdump'
AR	The command and arguments to run 'ar'
M4	GNU M4 is an implementation of the traditional Unix macro processor
NM	The command and arguments to run 'nm'
TARGET_PREFIX	The toolchain binary prefix for the target tools
CROSS_COMPILE	The toolchain binary prefix for the target tools
CONFIGURE_FLAGS	The arguments for GNU configure
CFLAGS	Suggested C flags
CXXFLAGS	Suggested C++ flags
LDFLAGS	Suggested linker flags when you use CC to link
CPPFLAGS	Suggested preprocessor flags
ARCH	Architecture

Table 1: Environment Variables

3.5. Creating and Building Applications

Now that the SDK environment is set up, the next step is to develop an application on your host machine.

3.5.1. Make File-Based Applications

This section describes a simple application development using makefile for demonstration purposes.

1. Prepare the following application files for the application

```
user@host:~/CA$ ls
main.c  Makefile
```

2. Write the helloworld application in **main.c** as shown below:

```
#include <stdio.h>
int main()
{
        printf("Hello, world!\n");
        return 0;
}
```

3. For Make-file based applications, the cross-toolchain environment variables established by running the az_sdk_env script are subject to general make rules. Prepare the Make File as below:

```
CFLAGS = -Wall -g
OBJECTS = main.o
TARGET = helloworld
all : $(TARGET)
$(TARGET) : $(OBJECTS)
        $(CC) $(LDFLAGS) -o $@ $^
clean :
        rm -f $(OBJECTS) $(TARGET)
```

4. Build the application as below:

```
user@host:~/CA$ ls
main.c  Makefile
user@host:~/CA$ make
arm-oe-linux-gnueabi-gcc  -march=armv7-a -mthumb -mfpu=neon -
mfloat-abi=hard --sysroot=/home/user/AZ_SDK/sysroot -Wall -
Werror -Wextra -g   -c -o main.o main.c
arm-oe-linux-gnueabi-gcc  -march=armv7-a -mthumb -mfpu=neon -
mfloat-abi=hard --sysroot=/home/user/AZ_SDK/sysroot -Wl,--
hash-style=gnu -Wl,--as-needed -o helloworld main.o
user@host:~/CA$ ls
helloworld  main.c  main.o  Makefile
```

3.6. Downloading and Running Applications

1. To begin, ensure that the ADB is enabled. You can configure ADB using the modem console.

```
at#enadb=1
OK
```

2. Load the application to the target device using ADB.

```
C:\Users\user>adb push Z:\CA\helloworld /data
Z:\CA\helloworld: 1 file pushed, 0 skipped. 1.4 MB/s (10268
bytes in 0.007s)
```

3. Run the application as below:

```
asdxprairie login: root
Password: oelinux123
root@sdxprairie:~# cd /data/
root@sdxprairie:/data# ls helloworld
helloworld
root@sdxprairie:/data# chmod 755 helloworld
root@sdxprairie:/data# ./helloworld
Hello, world!
root@sdxprairie:/data
```

3.7. Run on Bootup

If the user wants to run the application whenever the target device boots up, then the application can be registered to the startup script as below:

```
root@sdxprairie:~# mkdir /cache/oem_initscript
root@sdxprairie:~# vi /cache/oem_initscript/oemhp_start.sh
root@sdxprairie:~# cat /cache/oem_initscript/oemhp_start.sh
#!/bin/bash
/data/helloworld
```

4. USING THE MCM API'S

The MCM API is a callback-oriented API for accessing and manipulating communications for the device. The main method of accessing any functionality provided by the MCM framework is to create a request message structure, fill it with relevant parameters, and then pass it to the MCM framework via a synchronous or asynchronous call, which will then return a response message corresponding to the request. In addition, indication events can be received corresponding to system messages or changes.

The following sections provide the steps for development using the IoE MCM framework.

4.1. Initializing the MCM client

The MCM client must be initialized with the following code before any other calls are sent:

```
mcm_client_handle_type      hndl;
mcm_client_init (&hndl, ind_cb, async_cb);
```

Where:

ind_cb = Indication callback

async_cb = Asynchronous callback

The function returns 0 if successful.

4.2. Creating a Request Object with Parameters

To create a request object, use the code below and fill it with relevant parameters. For example, to create a voice call request object:

```
mcm_voice_dial_req_msg_v01req;
```

The following parameters are optional:

```
req.address_valid=1;
strlcpy(req.address,phone_number, MCM_MAX_PHONE_NUMBER_V01 + 1);
req.call_type_valid = 1;
req.call_type = MCM_VOICE_CALL_TYPE_VOICE_V01;
req.uusdata_valid = 0;
```

4.3. Creating a Response Object and Allocate Memory

To create a response object, use the code below and dynamically allocate memory to the object. For example, to create a voice call request object:

```
mcm_voice_dial_req_msg_v01req;
rsp = malloc(sizeof(mcm_voice_dial_resp_msg_v01));
memset(rsp, 0, sizeof(mcm_voice_dial_resp_msg_v01));
```

Note: The release of memory allocated with the malloc function at this stage is the user responsibility.

4.4. Making a Call

This API supports both synchronous and asynchronous calls.

- To dial an asynchronous voice call:

```
MCM_CLIENT_EXECUTE_COMMAND_ASYNC(hndl, MCM_VOICE_DIAL_REQ_V01, &req,
 rsp, async_cb, &token_id);
```

- To dial a synchronous call:

```
MCM_CLIENT_EXECUTE_COMMAND_SYNC(hndl, MCM_VOICE_DIAL_REQ_V01, &req,
rsp);
```

Where:

hndl = MCM client handle

MCM_VOICE_DIAL_REQ_V01 = Message ID for the request to identify the different requests req = Request object

rsp = Response object async_cb = Asynchronous callback function

token_id = Token ID returned from the request; used to verify whether a future callback is for the same async request

4.5. Defining an Asynchronous Callback Function

This function is used to receive a response from the async cal (see section 4.4 Making a Call). For example, to define a callback function for dialing a voice call:

```
void async_cb(mcm_client_handle_type   hndl, uint32_t  msg_id,
              void   *resp_c_struct, uint32_t  resp_len, void
                     *token_id)

 {
  switch(msg_id)
 {
   case MCM_VOICE_DIAL_RESP_V01:
    rsp =
    (mcm_voice_dial_resp_msg_v01*)resp_c_struct;
    if(!rsp->call_id_valid)
    {
     printf("Invalid Valid Call ID");
    }
// Can add more error checks here depending on the structure of the
// response
```

Where:

msg_id = Message ID for the response to identify different response types resp_c_struct = Response object returned by the framework

token_id = Toden ID returned from the callback; this is the same value as the value that was returned from the prior async request

4.6. Defining an Indication Callback Function (Optional)

This type of callback generally provides information concerning a change of state in the system:

```
void ind_cb(mcm_client_handle_type hndl,uint32_t   msg_id, void
   *ind_c_struct,uint32_t     ind_len);
```

```
To register for these types of callbacks, an event register call
must be used. For example:
```

```
MCM_CLIENT_EXECUTE_COMMAND_SYNC(hndl,
MCM_VOICE_EVENT_REGISTER_REQ_V01,

&ind_req, &ind_rsp);
```

4.7. Releasing a Client Handle

To release the client handle, use the following code:

```
mcm_client_release(hndl);
```

The function returns 0 if successful.

4.8. Compiling the Code

To compile the code, use the steps below:

1. Obtain the header files in the API folder and the mcm_client_stubs.c in the stubs folder.
2. Create a shared library (libmcm.so) using the mcm_client_stubs.c file. For example:

   ```
   arm-none-linux-gnueabi-gcc -I ../api -shared -Wl,-
   soname,libmcm.so.0 -o libmcm.so -fPIC mcm_client_stubs.c
   ```
   Where:
   `arm-none-linux-gnueabi-gcc` = A cross compiler `api` = A folder containing all the MCM header files
3. Link to the above shared library while generating the executable program for the C code. For example:

`arm-none-linux-gnueabi-gcc sample_code.c -I ../api -L. -lmcm -o sample_code Where: sample_code.c` = C code with MCM-related functions lmcm = Shared library libmcm.so sample_code = Name of the executable that was generated.

5. TEST EXAMPLE USING MCM API'S

This chapter describes a basic example that demonstrates how to use MCM APIs.

5.1. ATCOP Usage Source

5.1.1. Header Files and Definitions

```
#ifdef USE_GLIB

#include <glib.h>

#define strlcpy g_strlcpy

#endif


#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/shm.h>

#include <semaphore.h>


#include "mcm_client.h"

#include "mcm_common_v01.h"

#include "mcm_atcop_v01.h"


#define ATCOP    1

#define DATA     2

#define DM       3

#define LOC      4

#define MOBILEAP 5

#define NW       6

#define SIM      7
```

```
#define SMS      8
#define VOICE    9
#define EXIT     0
mcm_client_handle_type hndl;

/*DM*/
mcm_dm_get_radio_mode_req_msg_v01 mode_req;
mcm_dm_get_radio_mode_resp_msg_v01* mode_rsp;

/*ATCOP*/
mcm_atcop_req_msg_v01 at_req;
mcm_atcop_resp_msg_v01* at_rsp;

/*VOICE*/
mcm_voice_dial_req_msg_v01 dial_req;
mcm_voice_dial_resp_msg_v01* dial_rsp;

/* Is the call active? */
int callActive = FALSE;
uint32_t call_id;

/*SIM*/
mcm_sim_get_card_status_req_msg_v01 sim_req;
mcm_sim_get_card_status_resp_msg_v01* sim_rsp;

/*NW*/
mcm_nw_get_config_resp_msg_v01 config_req;

/*SMS*/
```

```
mcm_sms_get_msg_config_req_msg_v01 sms_req;
mcm_sms_get_msg_config_resp_msg_v01* sms_rsp;

int token_id = 0;
sem_t sem_wait_for_callback;
```

5.1.2. Release the Handler

```
/*  Release the handler */
void tear_down()
{
int release_result=mcm_client_release(hndl);
if(release_result!=0)
{
printf ("releasing client handle\n");
}
printf("MCM client hndl released\n");
}
```

5.1.3. Asynchronous Callback

```
/*asynchronous callback function*/
void async_cb(
mcm_client_handle_type hndl,
uint32_t  msg_id,
void *resp_c_struct,
uint32_t resp_len,
  void *token_id)
{
printf("==== ASYNC CALL BACK ENTER ====\n");
witch(msg_id)
{
case MCM_VOICE_DIAL_RESP_V01:
dial_rsp = (mcm_voice_dial_resp_msg_v01*)resp_c_struct;

if(dial_rsp->response.result != MCM_RESULT_SUCCESS_V01)
{
printf("Voice call failed.Error code: %d\n", dial_rsp->response.error);
                callActive = FALSE;
}
if(dial_rsp->call_id_valid)
{
printf("Valid Call ID = %d \n", dial_rsp->call_id);
                call_id = dial_rsp->call_id;
                callActive = TRUE;
}
else
{
                    printf("Invalid Call ID = %d \n", dial_rsp->call_id);
                    printf("Call ID Validity = %d \n", dial_rsp->call_id_valid);
                    callActive = FALSE;
}
break;

default:
printf("**** Unknown callback response **** \n");
break;
}
}
```

5.1.4. AT Test Function

This function executes the AT command.

```
void AT_test()

{

char atcom[MCM_ATCOP_MAX_REQ_MSG_SIZE_V01];

printf("input AT command : ");

scanf("%s",&atcom);


strcpy(at_req.cmd_req, atcom);

at_req.cmd_len=sizeof(at_req.cmd_req);


at_rsp = malloc(sizeof(mcm_atcop_resp_msg_v01));

if(at_rsp!=0)

{

memset(at_rsp, 0, sizeof(mcm_atcop_resp_msg_v01));

}


printf("\n**** AT COMMAND Test **** \n");

MCM_CLIENT_EXECUTE_COMMAND_SYNC(hndl,    MCM_ATCOP_REQ_V01,    &at_req,
at_rsp);


if(at_rsp->resp.result != MCM_RESULT_SUCCESS_V01)

printf("AT command response FAILED\n");


else

{

printf("Result %s\n",at_rsp->cmd_resp);

}
free(at_rsp);
}
```

5.1.5. Main Function

```
int main()
{


int input;
printf("Telit IoE MCM TEST!!!\n");
memset(&hndl, 0, sizeof(hndl));
int init_result = mcm_client_init(&hndl, ind_cb, async_cb);


printf("MCM_client_init result == %d\n",init_result);


if(init_result == MCM_SUCCESS_V01 || init_result == MCM_SUCCESS_CONDITIONAL_SUCCESS_V01)  /* mcm_client_init returns 0 on success */
{       // MCM CLIENT is SUCCESSFULLY INITIALIZED HERE
            while(1)
            {
             printf("\nTest IoE Manager\n");
             //printf("0 : EXIT\n1 : ATCOP\n2 : DATA\n3 : DM\n4 : LOC\n5 : MOBILEAP\n6 : NW\n7 : SIM\n8 : SMS\n9 : VOICE\n");
             printf("0 : EXIT\n1 : ATCOP\n3 : DM\n7 : SIM\n8 : SMS\n9 : VOICE\n");
             printf(" Input number : ");
             scanf("%d",&input);
             switch(input)
             {
              case ATCOP:
                AT_test();      // AT test function is called here
```

```
  break;

case DATA:
  break;

case DM:
  break;

case LOC:
  break;

case MOBILEAP:
  break;

case NW:
  break;

case SIM:
  break;

case SMS:
  break;

case VOICE:
  break;

case EXIT:
  printf("Done!!!\n");
  tear_down();
```

```
                return 0;

              default:
                printf("please input Right Value!!!\n");
                tear_down();
                return 0;
            }
            }

}
else
{
printf("MCM client init failed\n");
}

return 0;
}
```

5.2. Compilation

For compilation, use the steps below:

1. Set up the SDK (see section 3.4 Setting up the SDK)
2. Make your application directory inside AZ_SDK. Make a C source file (see section 5.1 ATCOP Usage Source)
3. Make a shared library using mcm_client_stubs.c.
   this source is available in API. (see section 4.8 Compiling the Code))

   ```
   arm-oe-linux-gnueabi-gcc  -march=armv7-a -mthumb -mfpu=neon -
   mfloat-abi=hard -sysroot=
   /home/TMT/user/Repositories/SDX55/AZ_SDK/sysroot -shared -Wl,-
   soname,libmcm.so.0 -o libmcm.so -fPIC mcm_client_stubs.c
   ```

   where:

   arm-oe-linux-gnueabi-gcc = cross compiler

--sysroot=/home/TMT/user/Repositories/SDX55/AZ_SDK/sysroot = API path where all the MCM api's are present

mcm_client_stubs.c = the source need to present in same folder while using which the shared library is created

```
libmcm.so
```

libmcm.so will be created

4. Compile the source file created in Step 2 using the shared library created in step 3.

   ```
   arm-oe-linux-gnueabi-gcc source-name.c -march=armv7-a -mthumb -
   mfpu=neon -mfloat-abi=hard –
   sysroot=/home/TMT/user/Repositories/SDX55/AZ_SDK/sysroot -L. -
   lmcm -o source-name
   ```

   where:

   source-name.c = application source file

   -lmcm = shared library

6. PRODUCT AND SAFETY INFORMATION

6.1. Copyrights and Other Notices

SPECIFICATIONS ARE SUBJECT TO CHANGE WITHOUT NOTICE

Although reasonable efforts have been made to ensure the accuracy of this document, Telit assumes no liability resulting from any inaccuracies or omissions in this document, or from the use of the information contained herein. The information contained in this document has been carefully checked and is believed to be reliable. Telit reserves the right to make changes to any of the products described herein, to revise it and to make changes from time to time without any obligation to notify anyone of such revisions or changes. Telit does not assume any liability arising from the application or use of any product, software, or circuit described herein; neither does it convey license under its patent rights or the rights of others.

This document may contain references or information about Telit's products (machines and programs), or services that are not announced in your country. Such references or information do not necessarily mean that Telit intends to announce such Telit products, programming, or services in your country.

6.1.1. Copyrights

This instruction manual and the Telit products described herein may include or describe Telit copyrighted material, such as computer programs stored in semiconductor memories or other media. The laws in Italy and in other countries reserve to Telit and its licensors certain exclusive rights for copyrighted material, including the exclusive right to copy, reproduce in any form, distribute and make derivative works of the copyrighted material. Accordingly, any of Telit's or its licensors' copyrighted material contained herein or described in this instruction manual, shall not be copied, reproduced, distributed, merged or modified in any way without the express written permission of the owner. Furthermore, the purchase of Telit products shall not be deemed to grant in any way, neither directly nor by implication, or estoppel, any license.

6.1.2. Computer Software Copyrights

Telit and the Third-Party supplied Software (SW) products, described in this instruction manual may include Telit's and other Third-Party's copyrighted computer programs stored in semiconductor memories or other media. The laws in Italy and in other countries reserve to Telit and other Third-Party, SW exclusive rights for copyrighted computer programs, including – but not limited to - the exclusive right to copy or reproduce in any form the copyrighted products. Accordingly, any copyrighted computer programs contained in Telit's products described in this instruction manual shall not be copied (reverse engineered) or reproduced in any manner without the express written permission of the copyright owner, being Telit or the Third-Party software supplier. Furthermore, the purchase of Telit products shall not be deemed to grant either directly or by implication, estoppel, or in any other way, any license under the copyrights, patents or patent applications of Telit or other Third-Party supplied SW, except for the normal non-exclusive, royalty free license to use arising by operation of law in the sale of a product.

6.2. Usage and Disclosure Restrictions

6.2.1. License Agreements

The software described in this document is owned by Telit and its licensors. It is furnished by express license agreement only and shall be used exclusively in accordance with the terms of such agreement.

6.2.2. Copyrighted Materials

The Software and the documentation are copyrighted materials. Making unauthorized copies is prohibited by the law. The software or the documentation shall not be reproduced, transmitted, transcribed, even partially, nor stored in a retrieval system, nor translated into any language or computer language, in any form or by any means, without prior written permission of Telit.

6.2.3. High-Risk Materials

Components, units, or third-party goods used in the making of the product described herein are NOT fault-tolerant and are NOT designed, manufactured, or intended for use as on-line control equipment in the following hazardous environments requiring fail-safe controls: operations of Nuclear Facilities, Aircraft Navigation or Aircraft Communication Systems, Air Traffic Control, Life Support, or Weapons Systems ("High-Risk Activities"). Telit and its supplier(s) specifically disclaim any expressed or implied warranty of fitness eligibility for such High-Risk Activities.

6.2.4. Trademarks

TELIT and the Stylized T-Logo are registered in the Trademark Office. All other product or service names are property of their respective owners.

6.2.5. Third-Party Rights

The software may include Third-Party's software Rights. In this case the user agrees to comply with all terms and conditions imposed in respect of such separate software rights. In addition to Third-Party Terms, the disclaimer of warranty and limitation of liability provisions in this License, shall apply to the Third-Party Rights software as well.

TELIT HEREBY DISCLAIMS ANY AND ALL WARRANTIES EXPRESSED OR IMPLIED FROM ANY THIRD-PARTY REGARDING ANY SEPARATE FILES, ANY THIRD-PARTY MATERIALS INCLUDED IN THE SOFTWARE, ANY THIRD-PARTY MATERIALS FROM WHICH THE SOFTWARE IS DERIVED (COLLECTIVELY "OTHER CODES"), AND THE USE OF ANY OR ALL OTHER CODES IN CONNECTION WITH THE SOFTWARE, INCLUDING (WITHOUT LIMITATION) ANY WARRANTIES OF SATISFACTORY QUALITY OR FITNESS FOR A PARTICULAR PURPOSE.

NO THIRD-PARTY LICENSORS OF OTHER CODES MUST BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST OF PROFITS), HOWEVER CAUSED AND WHETHER MADE UNDER CONTRACT, TORT OR OTHER LEGAL THEORY, ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE OTHER CODES OR THE EXERCISE OF ANY RIGHTS GRANTED UNDER EITHER OR BOTH THIS LICENSE AND THE LEGAL TERMS APPLICABLE TO ANY SEPARATE FILES, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

6.2.6. Waiver of Liability

IN NO EVENT WILL TELIT AND ITS AFFILIATES BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, GENERAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY INDIRECT DAMAGE OF ANY KIND WHATSOEVER, INCLUDING BUT NOT LIMITED TO REIMBURSEMENT OF COSTS, COMPENSATION OF ANY DAMAGE, LOSS OF PRODUCTION, LOSS OF PROFIT, LOSS OF USE, LOSS OF BUSINESS, LOSS OF DATA OR REVENUE, WHETHER OR NOT THE POSSIBILITY OF SUCH DAMAGES COULD HAVE BEEN REASONABLY FORESEEN,CONNECTED IN ANY WAY TO THE USE OF THE PRODUCT/S OR TO THE INFORMATION CONTAINED IN THE PRESENT DOCUMENTATION, EVEN IF TELIT AND/OR ITS AFFILIATES HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES OR THEY ARE FORESEEABLE OR FOR CLAIMS BY ANY THIRD PARTY.

6.3. Safety Recommendations

Make sure the use of this product is allowed in your country and in the environment required. The use of this product may be dangerous and has to be avoided in areas where:

- it can interfere with other electronic devices, particularly in environments such as hospitals, airports, aircrafts, etc.
- there is a risk of explosion such as gasoline stations, oil refineries, etc. It is the responsibility of the user to enforce the country regulation and the specific environment regulation.

Do not disassemble the product; any mark of tampering will compromise the warranty validity. We recommend following the instructions of the hardware user guides for correct wiring of the product. The product has to be supplied with a stabilized voltage source and the wiring has to be conformed to the security and fire prevention regulations. The product has to be handled with care, avoiding any contact with the pins because electrostatic discharges may damage the product itself. Same cautions have to be taken for the SIM, checking carefully the instruction for its use. Do not insert or remove the SIM when the product is in power saving mode.

The system integrator is responsible for the functioning of the final product. Therefore, the external components of the module, as well as any project or installation issue, have to be handled with care. Any interference may cause the risk of disturbing the GSM network or external devices or having an impact on the security system. Should there be any doubt, please refer to the technical documentation and the regulations in force. Every module has to be equipped with a proper antenna with specific characteristics. The antenna has to be installed carefully in order to avoid any interference with other electronic devices and has to guarantee a minimum distance from the body (20 cm). In case this requirement cannot be satisfied, the system integrator has to assess the final product against the SAR regulation.

The equipment is intended to be installed in a restricted area location.

The equipment must be supplied by an external specific limited power source in compliance with the standard EN 62368-1.

The European Community provides some Directives for the electronic equipment introduced on the market. All of the relevant information is available on the European Community website:

https://ec.europa.eu/growth/sectors/electrical-engineering_en

7. GLOSSARY

ADB	Android Debug Bridge
API	Application programming interface
APN	Access Point Name
GCC	Gnu Compiler Collection
GDB	Gnu Debugger
JSON	JavaScript Object Notation
LwM2M	Lightweight Machine To Machine
M2MB	Machine to Machine optimized Telit APIs
MCM	Mobile Connection Manager
PC	Personal Computer
PDP	Packed Data Protocol
SDK	Software Development Kit
USB	Universal Serial Bus
URC	Unsolicited Result Code
XML	eXtensible Markup File

8. RELATED DOCUMENTS

[1]	80624ST10996A	FN980m AT Command Reference Guide Preliminary
[2]	80624ST11005A	FN980m QMI Command Reference Guide Preliminary Draft
[3]	1VV0301615	FN980m SW Guide Preliminary

9. DOCUMENT HISTORY

Revision	Date	Changes
2	2020-06-13	Template updated Editorial changes
1	2021-05-07	Initial Revision

From Mod.0809 rev.5

Connect to our site and contact our technical support team for any question

www.telit.com