FN980m Appzone Linux SDK User Guide

1VV0301722 Rev. 1 – 2021-05-07

TELIT TECHNICAL DOCUMENTATION

SPECIFICATIONS ARE SUBJECT TO CHANGE WITHOUT NOTICE

NOTICES LIST

While reasonable efforts have been made to assure the accuracy of this document, Telit assumes no liability resulting from any inaccuracies or omissions in this document, or from use of the information obtained herein. The information in this document has been carefully checked and is believed to be reliable. However, no responsibility is assumed for inaccuracies or omissions. Telit reserves the right to make changes to any products described herein and reserves the right to revise this document and to make changes from time to time in content hereof with no obligation to notify any person of revisions or changes. Telit does not assume any liability arising out of 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.

It is possible that this publication may contain references to, or information about Telit products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that Telit intends to announce such Telit products, programming, or services in your country.

COPYRIGHTS

This instruction manual and the Telit products described in this instruction manual may be, include or describe copyrighted Telit material, such as computer programs stored in semiconductor memories or other media. Laws in the Italy and other countries preserve for 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 copyrighted material of Telit and its licensors contained herein or in the Telit products described in this instruction manual may not be copied, reproduced, distributed, merged or modified in any manner without the express written permission of Telit. Furthermore, the purchase of Telit products shall not be deemed to grant either directly or by implication, estoppel, or otherwise, any license under the copyrights, patents or patent applications of Telit, as arises by operation of law in the sale of a product.

COMPUTER SOFTWARE COPYRIGHTS

The Telit and 3rd Party supplied Software (SW) products described in this instruction manual may include copyrighted Telit and other 3rd Party supplied computer programs stored in semiconductor memories or other media. Laws in the Italy and other countries preserve for Telit and other 3rd Party supplied SW certain exclusive rights for copyrighted computer programs, including the exclusive right to copy or reproduce in any form the copyrighted computer program. Accordingly, any copyrighted Telit or other 3rd Party supplied SW computer programs contained in the Telit products described in this instruction manual may not be copied (reverse engineered) or reproduced in any manner without the express written permission of Telit or the 3rd Party SW supplier. Furthermore, the purchase of Telit products shall not be deemed to grant either directly or by implication, estoppel, or otherwise, any license under the copyrights, patents or patent applications of Telit or other 3rd Party supplied SW, except for the normal non-exclusive, royalty free license to use that arises by operation of law in the sale of a product.

USAGE AND DISCLOSURE RESTRICTIONS

I. License Agreements

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

II. Copyrighted Materials

Software and documentation are copyrighted materials. Making unauthorized copies is prohibited by law. No part of the software or documentation may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, without prior written permission of Telit

III. High Risk Materials

Components, units, or third-party products used in 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: the operation 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 for such High Risk Activities.

IV. Trademarks

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

V. Third Party Rights

The software may include Third Party Right software. In this case you agree to comply with all terms and conditions imposed on you in respect of such separate software. In addition to Third Party Terms, the disclaimer of warranty and limitation of liability provisions in this License shall apply to the Third Party Right software.

TELIT HEREBY DISCLAIMS ANY AND ALL WARRANTIES EXPRESS OR IMPLIED FROM ANY THIRD PARTIES REGARDING ANY SEPARATE FILES, ANY THIRD PARTY MATERIALS INCLUDED IN THE SOFTWARE, ANY THIRD PARTY MATERIALS FROM WHICH THE SOFTWARE IS DERIVED (COLLECTIVELY “OTHER CODE”), AND THE USE OF ANY OR ALL THE OTHER CODE 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 CODE SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST 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 CODE 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.

APPLICABILITY TABLE

PRODUCTS

	SW Versions	Modules
🟦 🟩 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 general contact, technical support services, technical questions and report documentation errors contact Telit Technical Support at:

- TS-EMEA@telit.com
- TS-AMERICAS@telit.com
- TS-APAC@telit.com

Alternatively, use:

http://www.telit.com/support

For detailed information about where you can buy the Telit modules or for recommendations on accessories and components visit:

http://www.telit.com

Our aim is to make this guide as helpful as possible. Keep us informed of your comments and suggestions for improvements.

Telit appreciates feedback from the users of our information.

1.4. Text Conventions

STOP Danger – This information MUST be followed, or catastrophic equipment failure or bodily injury may occur.

⚠ Caution or Warning – Alerts the user to important points about integrating the module, if these points are not followed, the module and end user equipment may fail or malfunction.

ℹ Tip or Information – Provides advice and suggestions that may be useful when integrating the module.

All dates are in ISO 8601 format, i.e. YYYY-MM-DD.

1.5. 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

2. REQUIREMENTS

The requirements needed in order to work with the Appzone Linux environment are as follows:

- **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. STAND ALONE SDK

3.1. Initial Configuration

3.1.1. Install ADB

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:

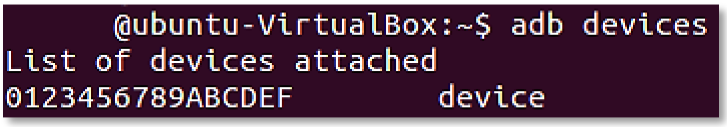
  adb kill-server
- Start the adb server using the command:

  adb start-server

To confirm that ADB was installed on your computer, connect your SDX55 module to the computer and use the following command:

- adb devices

Example:



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

Make sure 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 device. This also includes a compiler, a linker, and run-time libraries.
- **Sysroot:** Contains header files and libraries required for build.
- **Setup script:** *az_sdk_env_setup.sh* file when executed, sets the development environment for the target device.

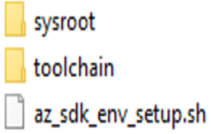
3.3. Installing the SDK

The first thing to do for installing the SDK is to unzip “<SDK version>_AZSDK .tar.gz” stand alone package in the specific directory on your host development machine.

Follow below commands to unzip the SDK.

```
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
```

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



sysroot
toolchain
az_sdk_env_setup.sh

⚠ AZ SDK version must match with target SW version. Because sysroot image contain libraries, headers and symbols specific to target SW version. Please, refer to APPLICABILITY TABLE for the available version information of AZ SDK and target SW version.

3.4. Setting up the SDK

After unzipping the SDK, The SDK environment setup script should be run. The set up script (az_sdk_env_setup.sh) resides in the unzipped directory.

Setup the SDK with 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
-----------------------------------------------------------------
```

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

ENVIRONMENT VARIABLES	DESCRIPTION
AZ_SDK_SYSROOT	The path to the sysroot contain 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

3.5. Creating and Building Applications

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

3.5.1. Make File Based Applications

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

Step 1: Prepare the following application files for the application:

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

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

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

Step 3: For Make-file based application, the cross-toolchain environment variables established by running 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)
```

Step 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

Step 1: First of all, the user need to check if the **ADB** is enabled. ADB setting can be done using the modem console.

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

Step 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)
```

Step 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 is bootup, 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. INSTRUCTIONS FOR 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. Initialize 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. Create a Request Object with Parameters

Use the code below to create a request object and fill it with relevant parameter. 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. Create a Response Object and Allocate Memory

Use the code below to create a response object 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 malloc function at this stage is the user responsibility.

4.4. Make a Call

This API supports both synchronous and asynchronous calls. For example, 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. Define an Asynchronous Callback Function

This function is used to receive a response from the async call made in Section 6.4. 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. Define 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. Release a Client Handle

Use the following code to release the client handle:

```
mcm_client_release(hndl);
```

The function returns 0 if successful.

4.8. Compile the Code

Use the following steps to compile the code:

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

In this section a simple example is described which will explain how to use MCM API's which is almost a simple practical implementation of section 6.

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");
  switch(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;
```

```
            callAction = 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 is the function responsible for executing 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

Step 1:

Step 1 is setting up the SDK which is same as section 5.4.

Step 2:

Make your application directory inside AZ_SDK. Make a C source file as explained in section 7.1

Step 3:

Make a shared library using mcm_client_stubs.c.
this source is available in API. (Refer to section 6.8)

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

Step 4:

Compile the source file created in step 2 using 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. GLOSSARY AND ACRONYMS

	Description
API	Application programming interface
ADB	Android Debug Bridge
API	Application programming interface
APN	Access Point Name – it is the name of a gateway between a GSM, GPRS, 3G or 4G mobile network and another computer network (usually the Internet)
GCC	Gnu Compiler Collection
GDB	Gnu Debugger
JSON	JavaScript Object Notation. It is a text-based data interchange format designed for transmitting and storing structured data, both human readable and machine readable.
LwM2M	Lightweight Machine To Machine – IoT Application Protocol designed for bidirectional communication between devices and a central server
M2MB	Machine to Machine optimized Telit APIs
MCM	Mobile Connection Manager
PC	Personal Computer
PDP	Packed Data Protocol – Often used in conjunction with “context” to define a specific data structure that allows the device to communicate using the Internet Protocol
SDK	Software Development Kit
USB	Universal Serial Bus
URC	Unsolicited Result Code – it is the message returned by the mobile equipment (the modem) that is not a direct result of an AT command. It could be a soft interrupt or the response of an AT asynchronous command
XML	eXtensible Markup File. It is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable

7. DOCUMENT HISTORY

Revision	Date	Changes
1	2021-05-07	Initial Revision

SUPPORT INQUIRIES

Link to **www.telit.com** and contact our technical support team for any questions related to technical issues.

www.telit.com

Telit

Telit Communications S.p.A.
Via Stazione di Prosecco, 5/B
I-34010 Sgonico (Trieste), Italy

Telit IoT Platforms LLC
5300 Broken Sound Blvd, Suite 150
Boca Raton, FL 33487, USA

Telit Wireless Solutions Inc.
3131 RDU Center Drive, Suite 135
Morrisville, NC 27560, USA

Telit Wireless Solutions Co., Ltd.
8th Fl., Shinyoung Securities Bld.
6, Gukjegeumyung-ro8-gil, Yeongdeungpo-gu
Seoul, 150-884, Korea

Telit Wireless Solutions Ltd.
10 Habarzel St.
Tel Aviv 69710, Israel

Telit Wireless Solutions
Technologia e Servicos Ltda
Avenida Paulista, 1776, Room 10.C
01310-921 São Paulo, Brazil