UM2426
User manual

X-CUBE-CELLULAR cellular connectivity Expansion Package for STM32Cube

Introduction

This user manual describes the content and use of the X-CUBE-CELLULAR cellular connectivity Expansion Package for STM32Cube.

The X-CUBE-CELLULAR Expansion Package enables connectivity over cellular networks. The network access technology depends on the cellular modem used: NB-IoT, LTE Cat M or 2G. The cellular connectivity framework exposes standard APIs for easy integration of any application interacting with a remote host using the TCP-over-IP or UDP-over-IP protocol.

The X-CUBE-CELLULAR Expansion Package for STM32Cube provides also an application to demonstrate and verify end-to-end connection over the Internet.

X-CUBE-CELLULAR is available for multiple STM32 and modem setups. Refer to the main release note available at the root of the X-CUBE-CELLULAR Expansion Package for details.

The main features of the X-CUBE-CELLULAR Expansion Package are:

- Ready-to-run firmware examples using the NB-IoT, LTE Cat M or 2G protocols
- Cellular Service and connection management with remote SIM provisioning capability
- Network radio level reporting
- Testing of access to a remote machine
- Connection and data exchange using the TCP or UDP socket protocols

Contents

List of tables

List of figures

1 General information

This user manual describes the X-CUBE-CELLULAR Expansion Package and its use. It explains neither the cellular networks nor the cellular protocol stacks, the descriptions of which being available on the Internet.

Information on STMicroelectronics cellular solutions is available from STMicroelectronics MCU wiki *[7]*.

X-CUBE-CELLULAR is available for the hardware setups described in *[5]*.

The X-CUBE-CELLULAR Expansion Package runs on STM32L4 32-bit microcontrollers based on the Arm®(a) Cortex®-M4 processor. It can be easily adapted to run on other STM32 microcontroller series

1.1 Terms and definitions

Table 1 presents the definition of acronyms that are relevant for a better understanding of this document.

Table 1. List of acronyms

Term	Definition
API	Application programming interface / Access point identifier
APN	Access point name
BSD	Berkeley software distribution
BSP	Board support package
CID	Context ID (context identifier of a cellular connection)
CLI	Command-line interface
COM	Cellular communication
DC	Data Cache
HAL	Hardware abstraction layer
IDE	Integrated development environment
IF	Interface
IoT	Internet of things (refer to *[4]*)
IPC	Inter-processor channel
LED	Light-emitting diode
NAT	Network address translation
NFMC	Network-friendly management configuration (refer to *[4]*)
MNO	Mobile network operator

a. Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

Table 1. List of acronyms (continued)

Term	Definition
MVNO	Mobile virtual network operator
PDN	Packet data network
PDP	Packet data protocol
PPP	Point-to-point protocol
PPPoSIF	PPP over serial IF
PS	Packet switching
RAM	Random-access memory
RSSI	Received-signal strength indication
RTC	Real-time clock
TCP	Transmission control protocol
UDP	User datagram protocol
URC	Unsolicited result code

1.2 References

1. *Development guidelines for STM32Cube Expansion Packages* user manual (UM2285)
2. *Development checklist for STM32Cube Expansion Packages* user manual (UM2312)
3. *Getting started with STM32CubeL4 for STM32L4 Series and STM32L4+ Series* user manual (UM1860)
4. *IoT Device Connection Efficiency Guidelines* (TSG.34/TS.34) from the GSM Association
5. *Release note* available at the root of the X-CUBE-CELLULAR Expansion Package. The main release note provides a link to the *Cellular* release note, which contains all information about the current delivery and a regularly updated FAQ
6. *X-CUBE-CELLULAR Application Programming Interface* available as file *EmbSw_application_programming_interface_Cellular_Package.chm* in X-CUBE-CELLULAR Expansion Package
7. *Cellular LTE CatM / NBIoT overview* wiki
   This page contains application examples, document references, and links to regularly updated STM32 cellular material. It is available from STMicroelectronics MCU wiki at wiki.st.com/stm32mcu

2 Important note regarding the safety

Caution requirements about the RF exposition must be fulfilled by the end-user as defined by the regulatory standards.

Warning: Use the hardware only with the antenna connected. With no antenna connected, there is a risk of damage to the modem.

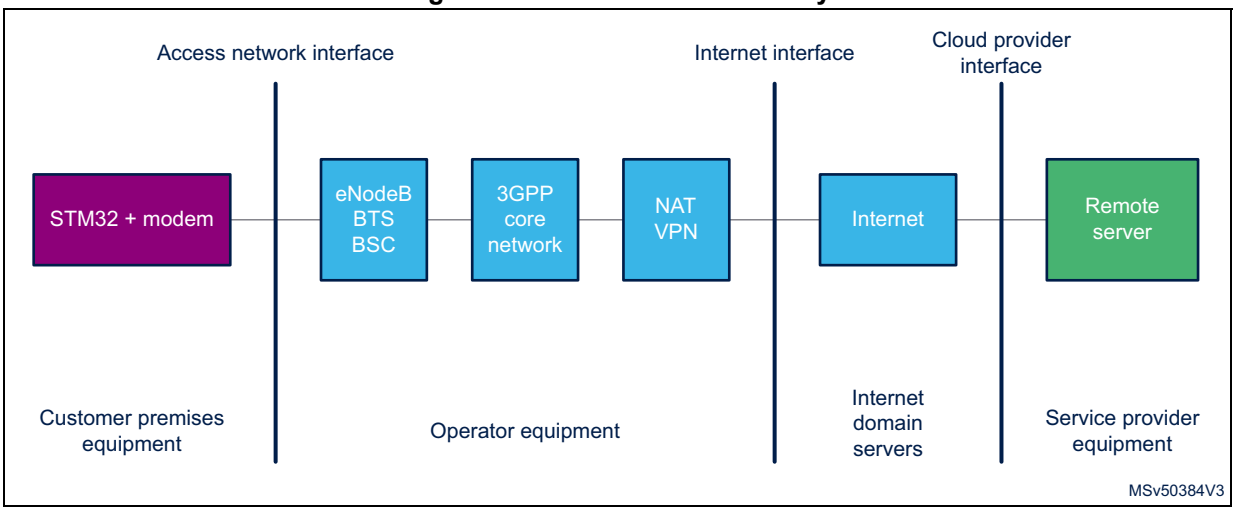
3 Service connectivity description

The X-CUBE-CELLULAR Expansion Package offers out-of-the-box connectivity for communication to the Internet. It implements a complete middleware-level and application-level stack in C language.

The example provided in X-CUBE-CELLULAR demonstrates TCP and UDP over IP, and also allows the use of the ping command. It connects to an echo server that returns to the device the data it sends. This example also provides the throughput at the application level.

Figure 1 presents the cellular IoT connectivity handled by the X-CUBE-CELLULAR Expansion Package.

Figure 1. Cellular IoT connectivity



4 Package description

This chapter details the content and use of the X-CUBE-CELLULAR Expansion Package.

4.1 General description

The X-CUBE-CELLULAR Expansion Package only provides software components running on the host STM32 MCU. Cellular modem firmware is not in the scope of this document.

The following integrated development environments are supported:

- STMicroelectronics integrated development environment for STM32 products (STM32CubeIDE)
- IAR Systems® IAR Embedded Workbench® for Arm® (EWARM)
- Keil® Microcontroller Development Kit (MDK-ARM)

Note: Refer to [5] for information about the IDE versions supported.

IAR™ binaries are provided in the package.

4.2 Modem socket versus LwIP

Either modem socket or LwIP can be used for the IP stack:

- Modem socket: the IP stack runs in modem FW
- LwIP: the LwIP stack runs on the STM32 side

This option is selected in software through a #define used during the compilation process.

Note: Only IPV4 is supported.

If LwIP is selected, the communication between host and modem is done through the PPP layer. There is a PPP client on the host side, and a PPP server on the modem side. PPPoSIF adapts the LwIP stack to a serial IF, while LwIP usually uses Ethernet interfacing.

4.3 Architecture

X-CUBE-CELLULAR runs on STM32 boards and allows sending or receiving IP packets to or from the Internet via an add-on cellular module.

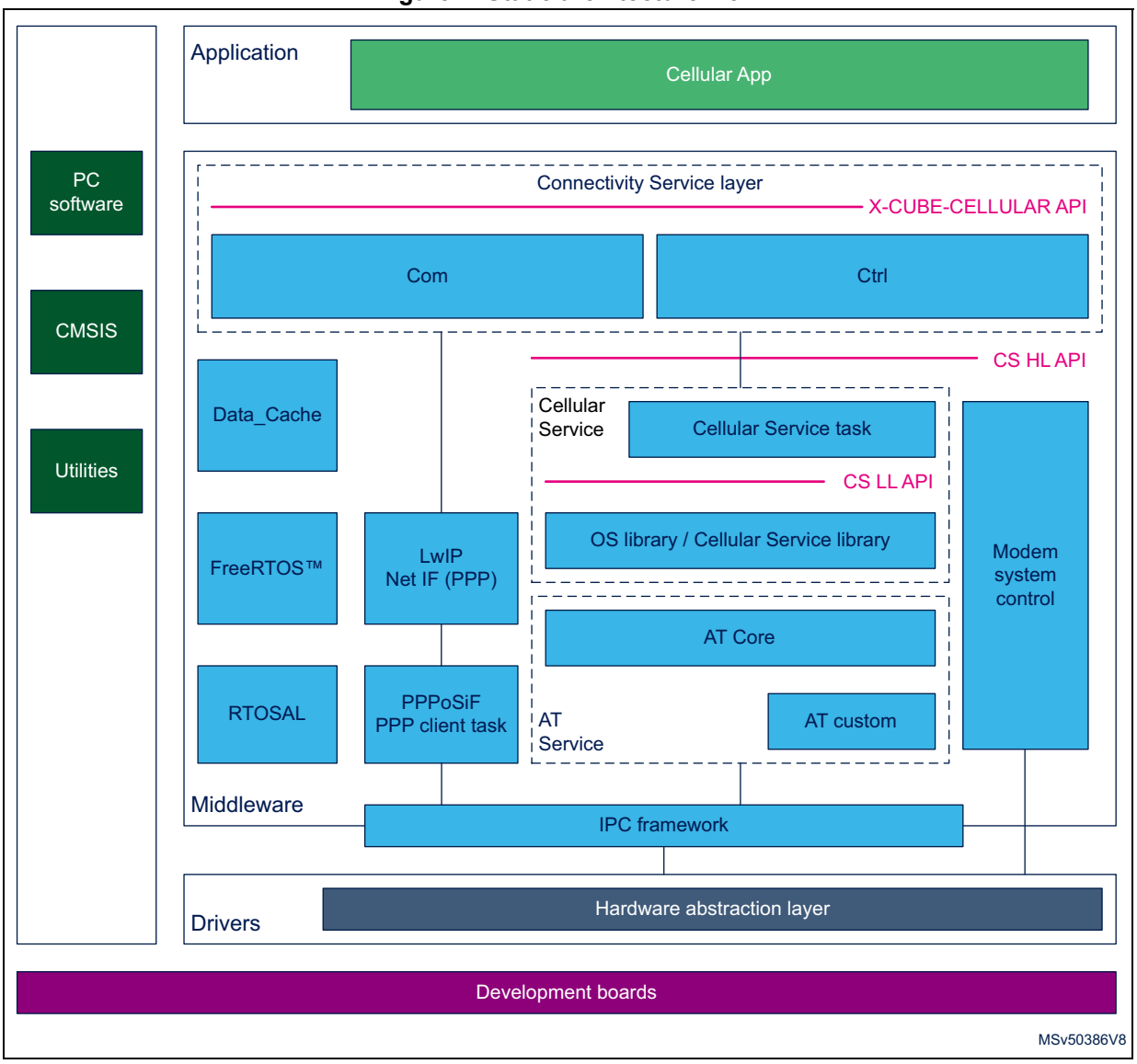
The package is split into the following components:

- STM32L4 Series HAL
- RTOSAL / CMSIS / FreeRTOS™
- LwIP / PPP
- AT Service
- Cellular Service
- Ctrl
- Com
- Data_Cache
- IPC

4.3.1 Static architecture view

X-CUBE-CELLULAR static architecture is presented in *Figure 2*.

Figure 2. Static architecture view



API description

The X-CUBE-CELLULAR API of the cellular package is defined by two components.

- Cellular management (**Ctrl**): initialization/starting of cellular features. It also represents the control plane.
- **Com**: data communication API based on the BSD API. It represents the data plane.

Component description

- **Cellular App**: implements a client application, which sends data to a remote server and waits in return the same packet. It is used to provide a simple example using the TCP or UDP protocol socket. It also implements ping service and provides the throughput at application level. Two instances can be started at the same time to send and receive data.
- **PPP client task / PPPoSIF**: optional component. It is only present when LwIP is used. It is in charge of establishing the PPP link with the modem.
- **LwIP / Net IF**: the LwIP component and its adaptation to PPP.
- **Cellular Service**:
  - **Cellular Service task** controls modem power-on and initialization, instructs the modem to perform network registration, activates the PDN (PDP context), and enters data transfer mode. It uses AT service to send AT commands to the modem.

    It implements a generic finite state machine to maintain consistent service state based on modem internal state change events (such as FOTA or reset), network registration state change events, and events related to the and PDP context status. It implements the network friendly features (NFMC) as defined in *[4]*. The configuration for example encompasses APN and CID settings, enabling and disabling NFMC, and setting the back-off timers.

    For system robustness, the Cellular Service task ensures that the modem is always operational by regularly polling the modem RSSI.
  - **Cellular Service OS**: is a library that offers a collection of functions. The library serializes the access to the single AT channel interface that is used to communicate with the modem. The functions are called by the COM service and the Cellular Service task.
  - **Cellular Service library**: offers a collection of blocking function calls to interact with a modem. Cellular Service is in charge of translating the request from the Cellular Service task or COM service to a sequence of AT commands that must be sent to the modem. It finally calls a callback function (from the Cellular Service task or COM service) when an asynchronous event (URC) is received from the modem.
- **AT Service**: provides a framework to send or receive AT commands to or from the modem over IPC. The AT Core task is in charge of processing the Cellular Service requests and translate them into AT commands. It is also in charge of processing AT commands response and URCs from the modem and forward them to Cellular Service. AT is split into two parts:

  1. a generic part, "Core" (AT framework and manage standard AT commands)

  2. a specific part, "custom" (implements specific modem behavior and AT commands)
- **Modem system control**: supports modem HW system control signaling (power on/off, reset). It is split into a generic and a specific part. The generic part exposes the generic API to the application (Cellular Service) while the specific part controls the GPIO dedicated to the modem.

- **IPC**: provides function to send data to Modem or to read data from the Rx FIFO buffer. Supports two channels: character mode and stream mode:
  - Stream mode is used for data transfer (PPP).
  - Character mode is used to send AT commands.
- **Data_Cache**: allows the sharing of data between components.
- **FreeRTOS™**: provides RTOS services to create the resources and scheduler needed by the software to run, such as threads and tasks, dynamic memory allocation, mutexes, and semaphores. A default task (*freertos.c*) is in charge of system initialization, creation of all application tasks.
- **RTOSAL**: provides the CMSIS V1/V2 abstraction. The application uses the RTOSAL API; according to a flag, CMSIS V1 or CMSIS V2 is used.

4.3.2 Dynamic architecture view

The X-CUBE-CELLULAR dynamic architecture is further illustrated with sequence charts that illustrate the interactions between components from initialization to socket. These sequence charts are available from STMicroelectronics MCU wiki *[7]*.

4.4 Folder structure

Figure 3 and *Figure 4* present the folder structure of the X-CUBE-CELLULAR Expansion Package with middleware folder break down.

Figure 3. X-CUBE-CELLULAR folder structure (1 of 2)

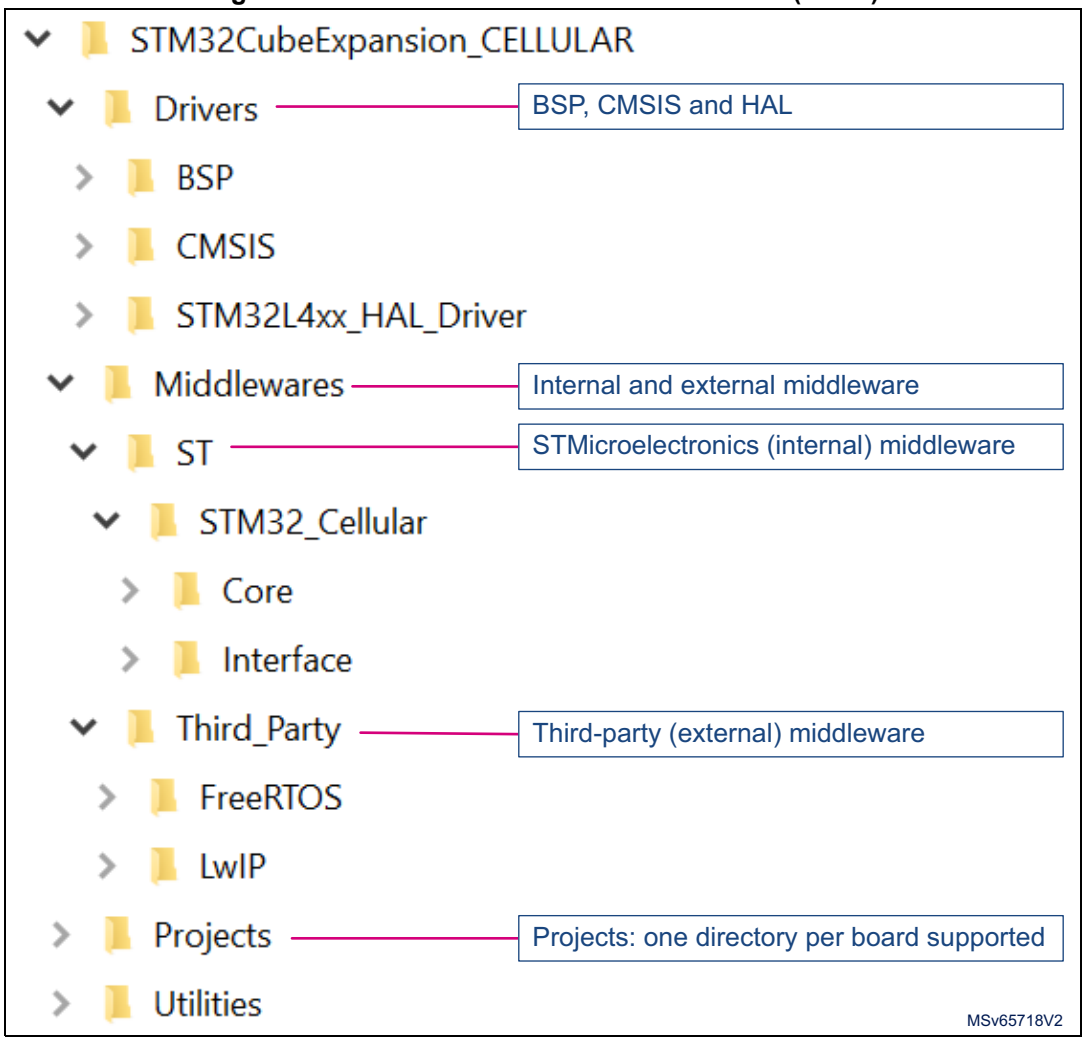
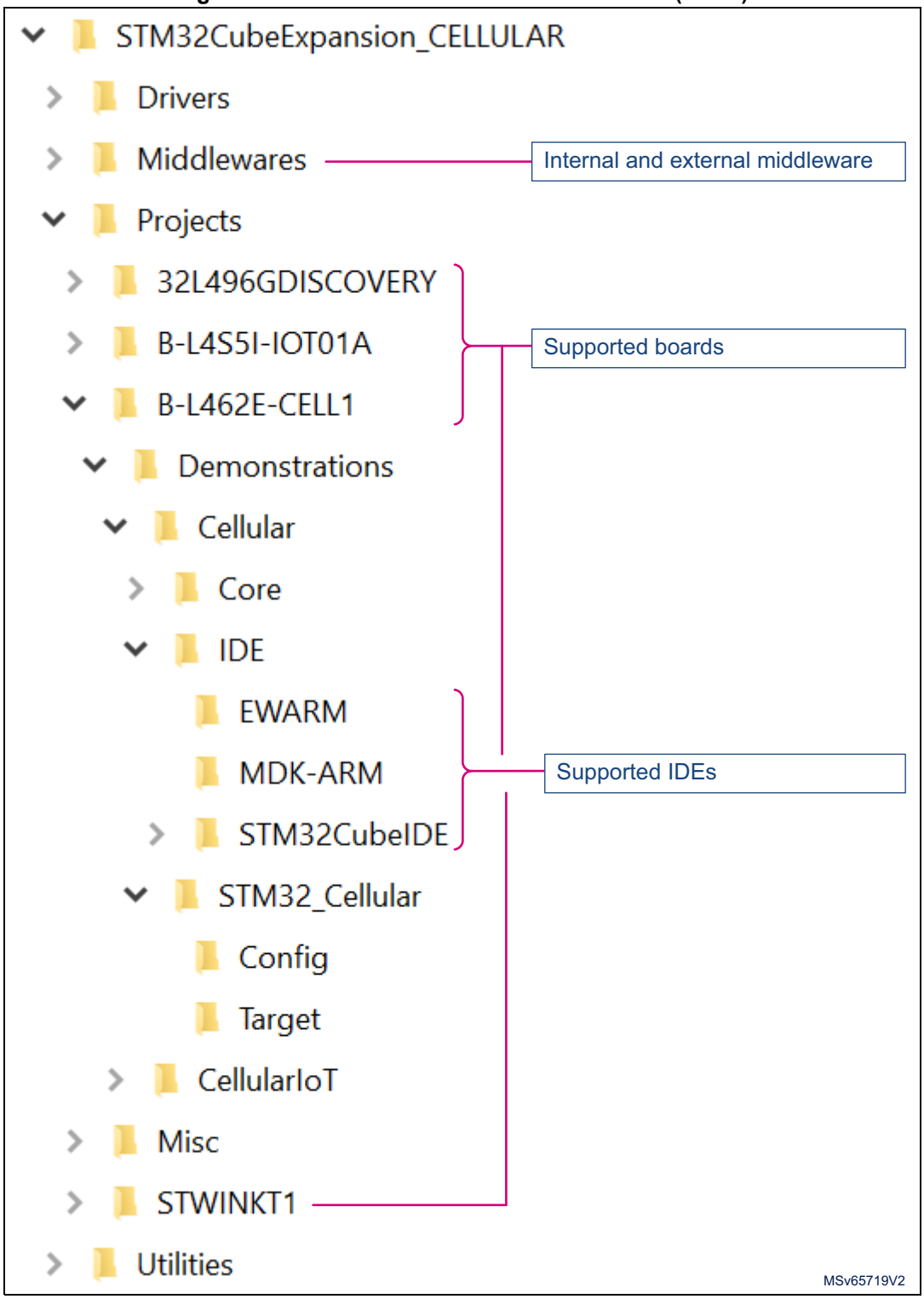


Figure 4. X-CUBE-CELLULAR folder structure (2 of 2)



4.5 Reset push-button

The reset push-button (black) is used to reset the board at any time. This action forces the reboot of the platform.

4.6 Real-time clock

System date and time are managed by the RTC.

The system date is updated by the application and therefore the hour values depend on the remote server time zone.

5 Cellular connectivity example

This chapter describes the cellular connectivity available application example, the Cellular App that exchanges data over TCP or UDP with an echo server, allows ping command, and provides the throughput at application level.

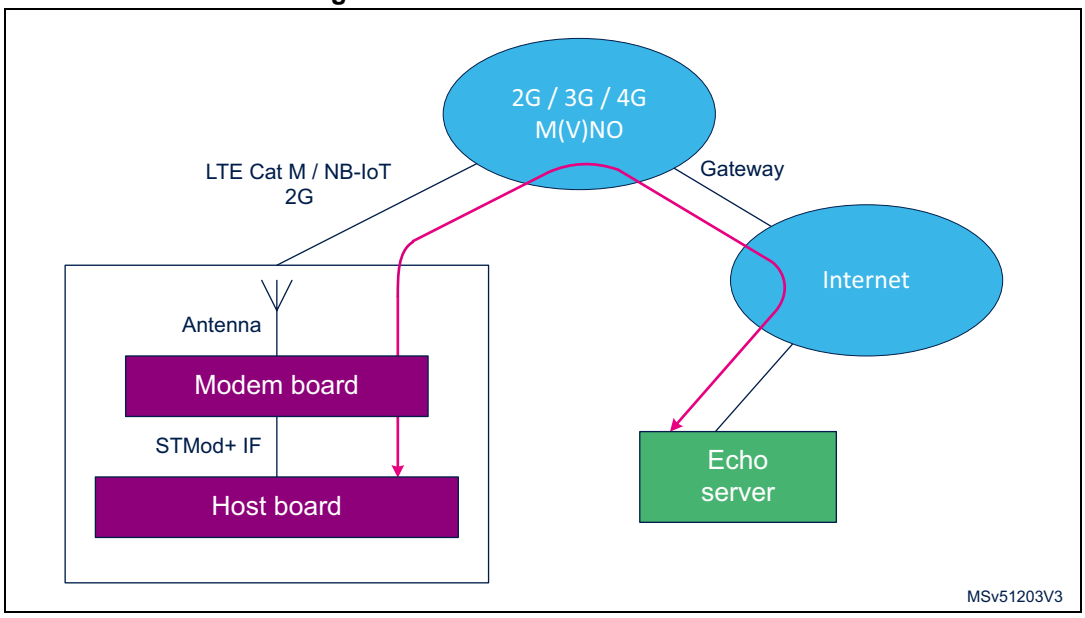
5.1 Real network or simulator

Users must ensure that network coverage with the target technology is available at their location. Indeed, 2G, LTE Cat M and NB-IoT are not available worldwide. If the live network is not available for the target technology, the users can use a network simulator such as Amarisoft's. To run the example, it is assumed that a network connection is possible with the device.

5.2 Connection overview

The server connection overview is presented in *Figure 5*.

Figure 5. Server connection overview



5.3 Cellular App example

The cellular application is composed of several parts:

- The first part is the ECHO client. The ECHO client sends data to an echo server, which replies with what it has received. By default, there is one ECHO client instance activated but a second one can be started.
- The second part of the cellular application is PING; It allows the user to send a ping command to test the link with another IP.
- The last part of cellular application is the performance; it displays the throughput at application (not air interface) level.

Enter the `cellularapp help` command to display all the commands provided with the cellular application.

6 Hardware and software environment setup

Multiple STM32 boards / STMod+ modems configurations are possible. Refer to *[5]* and *[7]* for detailed information.

Power on the board by plugging its USB connector to a PC, USB power supply, or USB power bank. If traces must be displayed, the USB connector must be connected to a PC with an open console application.

Warning: Use the hardware only with the antenna connected. With no antenna connected, there is a risk of damage to the modem.

Figure 6 depicts the hardware setup.

Figure 6. Hardware setup

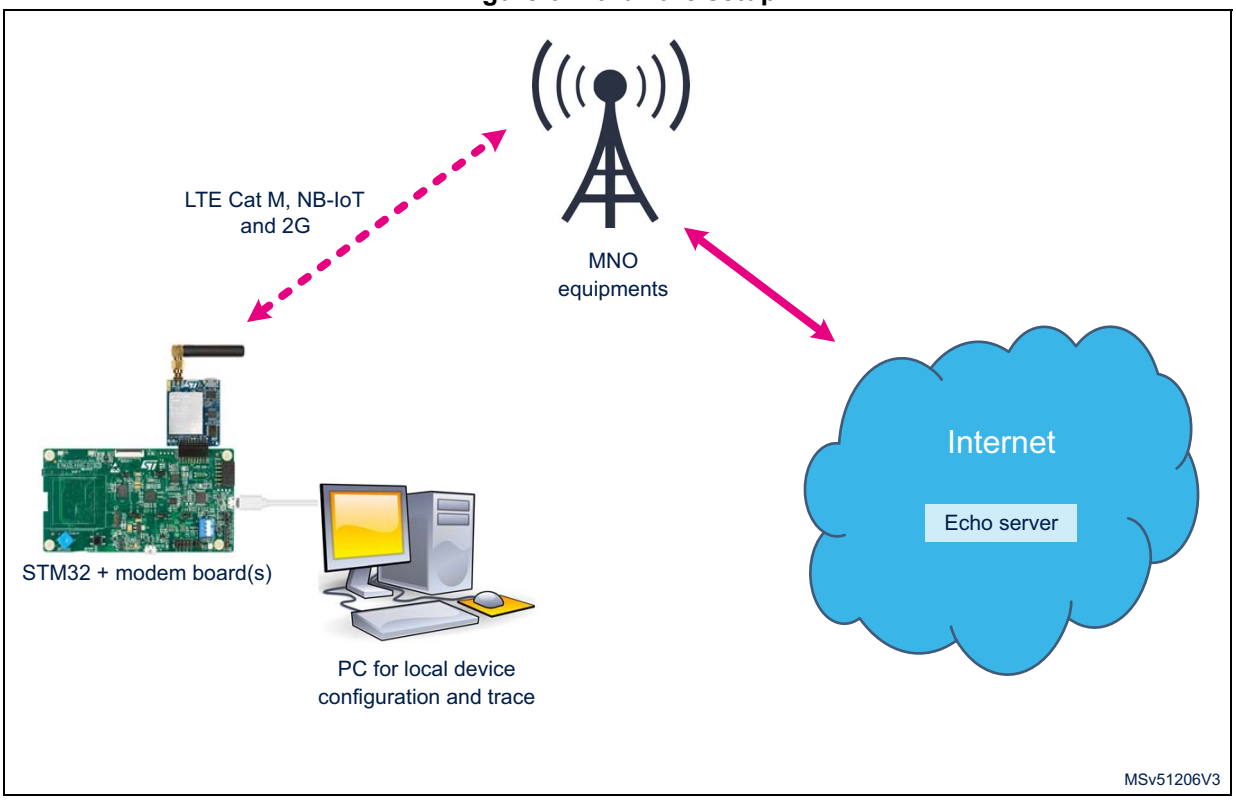
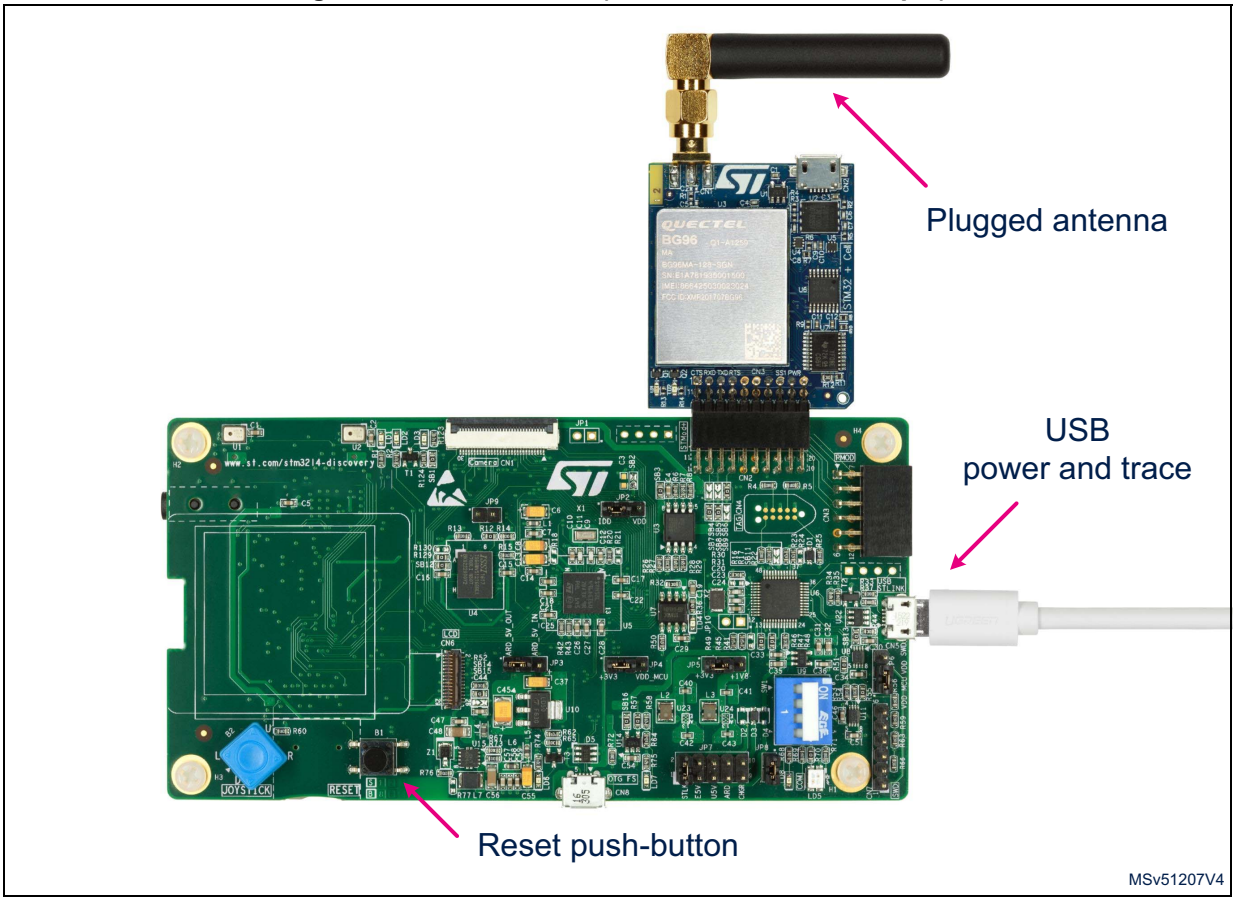


Figure 7 and *Figure 8* present two hardware setup examples. Refer to *[5]* for the updated list of supported hardware setups.

Figure 7 shows the P-L496G-CELL02 with the USB cable in place for power supply and optional trace / boot menu.

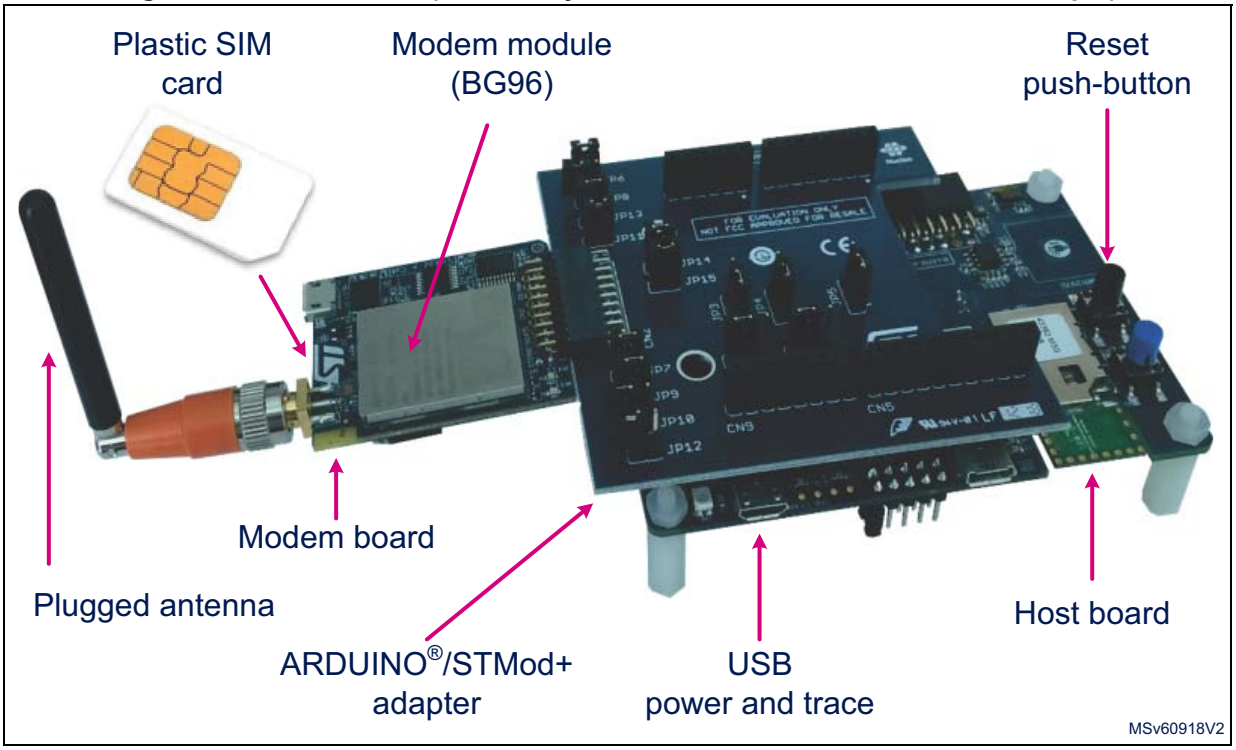
Figure 7. Hardware view (P-L496G-CELL02 example)



Pictures are not contractual.

Figure 8 shows the “Discovery IoT node cellular with BG96” set with the B-L475E-IOT01A host, X-NUCLEO-STMODA1 adapter and MB1329 modem boards connected together.

Figure 8. Hardware view (“Discovery IoT node cellular with BG96” set example)



7 Interacting with the host board

To interact with the Host board a serial console is used (Virtual COM port over USB). With the Windows® operating system, the use of the Tera Term software is recommended.

Serial port settings for communicating with the host board are illustrated in *Figure 9*. The menu is reached through *Setup* > *Serial port*. Set *Baud rate* to 115200 to get Tera Term up and running. For the boot setup configuration, a *Transmit delay* of 10 ms must be applied. Local echo must be enabled.

Figure 9. Serial port settings to interact with the host board

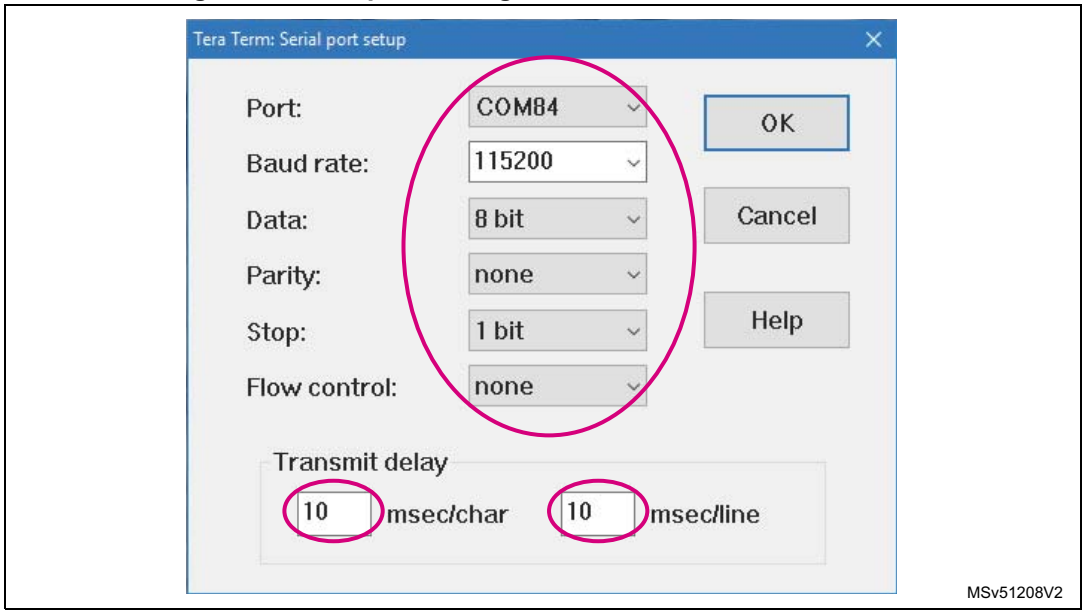
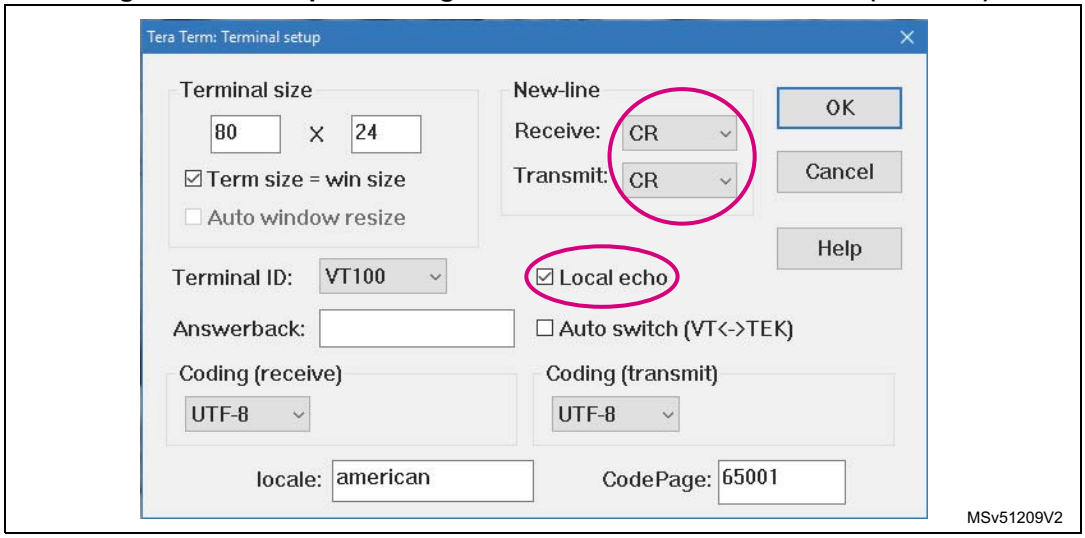


Figure 10 illustrates the menu for setting the terminal parameters. It is reached through *Setup* > *Terminal*. Set both *Receive* and *Transmit New-line* parameters to CR. Set *Receive* to AUTO instead of CR for clearer traces.

Figure 10. Serial port settings to interact with the host board (new-line)



Once all terminal and serial port parameters are properly set, press the board reset button (black).

7.1 Debug

The debug trace is viewed through the Virtual COM port of the PC that is connected to the board and powers it.

Debug levels are customized in file *plf_sw_config.h*.

7.2 Console command

The user can enter command lines (CLI) over a terminal to configure some parameters needed to setup the cellular connectivity such as SIM card, APN, Access Technology and others. CLI can also be used to read the actual parameters used or some status. By default, the CLI is enabled. However if CLI is not needed, it can be disabled by setting the `USE_CMD_CONSOLE` compilation flag to 0 in file *plf_features.h*.

8 How to customize the software?

There are three possible software customization levels applicable to X-CUBE-CELLULAR, which are presented in this chapter: user customization, advanced user customization, and developer customization.

8.1 First customization level: user customization

The first customization level is the use of PC commands. Note that the `help` command displays the list of available commands.

At this level, firmware is not modified. No customization-induced compilation is needed.

8.2 Second customization level: advanced user customization

At this level, firmware configuration modification is possible. Specific features can be added or removed and firmware configuration parameters can be modified.

Customization-induced recompilation is needed.

8.3 Third customization level: developer customization

8.3.1 Boot

The boot is the first part of the device initialization. It is included in file *main.c* (main function). This part concerns the HW initialization done by HAL. The file is generated by STM32CubeMX with the *.ioc* file provided. It must be updated only if a new HW is used or if the user configures peripherals on the host board (such as GPIO, I^2C, or others).

8.3.2 Initialization of software components

The application is initialized in file *freertos.c*.

The application initializes (`cellular_init`) and starts (`cellular_start`) the Cellular middleware.

8.3.3 Software customization

Firmware configuration parameters are included in files:

- *FreeRTOSConfig.h*: includes FreeRTOS™ parameters
- *lwipopts.h*: includes LwIP parameters
- *plf_hw_config.h*: includes HW parameters (such as UART configuration, GPIO used, and others). A change is usually needed to adapt the software to a new board.
- *plf_sw_config.h*: includes SW parameters (such as trace activations and others)
- *plf_thread_config.h*: includes thread stack sizes and task priorities. The stack sizes included in this file are used to calculate the FreeRTOS™ heap size (contained in file *FreeRTOSConfig.h*).
- *plf_features.h*: includes the selected applications

Revision history

Table 2. Document revision history

Date	Revision	Changes
28-Jun-2018	1	Initial release.
2-Nov-2018	2	X-CUBE-CELLULAR support extended to the “Discovery IoT node cellular” set: – Updated *Introduction*, *Chapter 1: General information*, *Section 4.2: Modem socket versus LwIP*, *Chapter 6: Hardware and software environment setup*, *Section 5.4: Grovestreams (HTTP) access example*, and *A.2 How to activate the soldered SIM card?* Augmented X-CUBE-CELLULAR feature description: – Added *Section 7.3: Console command*, *Section 8.4: Data cache*, *Section 8.2.3: Different kinds of available traces*, *Section 8.2.4: How to configure traces?*, and *A.6 How to select BG96 modem configuration bands?* Updated X-CUBE-CELLULAR feature description: – NIFMAN in *Section 4.3.2: Static architecture view* – PING in *Section 5.3: PING example*, *Section 7.2.3: “Setup Menu” option*, and *Section 7.2.8: Boot menu and configuration complete example* – Menu in *Section 7.2: Boot menu* – Compilation variables in *Table 2: Compilation variables for applications in firmware*
12-Feb-2019	3	Extended support to B-L475E-IOT01A with X-NUCLEO-STMODA1 and GM01Q-STMOD, and 32L496GDISCOVERY with GM01Q-STMOD: – Updated *Introduction* and *Chapter 6: Hardware and software environment setup* – Updated *Figure 1*, *Figure 19*, and *Figure 20* Added UDP support and ECHO client application: – Updated *Chapter 1: General information* and *Chapter 3: Service connectivity description* – Updated *Section 4.2: Modem socket versus LwIP* and *Section 4.3.2: Static architecture view* – Added *Section 5.4: ECHO example* – Updated *Table 2: Compilation variables for applications in firmware* and *Table 5: Number of project threads setting example* Simplified *Section 7.2: Boot menu* and *Section 7.3: Console command*. Reorganized *A.4.1 COM API*. Updated *Section 8.4: Data cache* and *A.4.2 Data Cache API*.

Table 2. Document revision history (continued)

Date	Revision	Changes
22-May-2019	4	Updated the document title and simplified the document structure: – Updated *Introduction* and *Chapter 1: General information* – Updated, *Figure 2*, *Figure 3*, and *Figure 7*; added *Figure 8* – Simplified *Section 4.3.3: Dynamic architecture view* – Removed *Section 4.7:Application LED* – Simplified the description of Data Cache in *Section 8.4.3: Main Data Cache entries* – Removed *A.3 Frequently asked questions* – Removed *A.4 X-CUBE-CELLULAR API descriptions*
11-Oct-2019	5	Updated *Introduction* and *Chapter 1: General information*. Updated *Section 4.3.1: Architecture concept*, *Section 4.3.2: Static architecture view*, *Section 4.3.3: Dynamic architecture view*, and *Section 4.5: Folder structure* as per middleware evolution. Updated file names, file paths, and API descriptions in *Chapter 6: Hardware and software environment setup*, *Chapter 7: Interacting with the host board*, and *Chapter 8: How to customize the software?* Removed *A.3 How to measure cellular throughput?* and added *A.4 How to configure the Network library API?*
9-Apr-2020	6	Introduced the MQTT, COM and CUSTOM applications and reflected the evolution of the Expansion Package structure: – Added *Section 5.6: MQTT example* and *Section 5.7: COM example* – Updated *Chapter 1: General information* and *Chapter 3: Service connectivity description* – Updated *Section 4.3.2: Static architecture view*, *Figure 4: Dynamic architecture - X-CUBE-CELLULAR API - Platform initialization and start*, *Section 4.4: X-CUBE-CELLULAR Expansion Package description*, *Section 4.5: Folder structure*, *Section 8.3.5: Adding a new component*, and *Section 8.4: Data cache* – Updated file names, file paths, and API descriptions in *Chapter 8: How to customize the software?* including *Table 2: Compilation variables for applications in firmware*, *Table 4: New thread registration example*, and *Table 5: Number of project threads setting example* – Removed *Section 8.4.2: Data Cache API*, *Section 8.4.3: Main Data Cache entries*, and *Table 6: Code for thread stack consumption monitoring example*

Table 2. Document revision history (continued)

Date	Revision	Changes
2-May-2021	7	Focused X-CUBE-CELLULAR on the LTE Cat M or NB-IoT cellular communication protocols with 2G fallback through the Cellular App application: – Updated *Introduction*, *General information*, *Service connectivity description* and *Package description* – Removed *Figure 7: Dynamic architecture - Network library API - Initialization*, *Figure 8: Dynamic architecture - Network library API - Connection*, *Figure 9: Dynamic architecture - Network library API - Disconnection*, and *Figure 10: Dynamic architecture - Network library API - Socket service* – Removed *Section 5.3: PING example*, *Section 5.5: GroveStreams (HTTP) access example*, *Section 5.6: MQTT example*, *Section 5.7: COM example*, *Section 7.2: Boot menu,* and *Section 7.3: Console command* – Added *Cellular App example* Simplified the document: – Removed *Appendix A Support material* and *Section 4.3.1: Architecture concept* – Removed *Section 8.2.1: Adding/removing an application in firmware*, *Section 8.2.2: IP stack on MCU side or on modem side*, *Section 8.2.3: Different kinds of available traces* and *Section 8.2.4: How to configure traces?* – Removed *Section 8.3.4: Firmware adaptation to a new HW configuration*, *Section 8.3.5: Adding a new component*, *Section 8.4: Data cache* and *Section 8.5: Thread stack consumption monitoring* – Removed *Figure 4: Dynamic architecture - X-CUBE-CELLULAR API - Platform initialization and start*, *Figure 5: Dynamic architecture - X-CUBE-CELLULAR API - Up to PDN creation*, and *Figure 6: Dynamic architecture - X-CUBE-CELLULAR API - Socket* – Removed *Section 4.4: Expansion Package description* – Updated *Important note regarding the safety*, *Cellular connectivity example* and *Hardware and software environment setup*