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: 2G, 3G, LTE Cat M1, or NB-IoT (also known as NB1). The cellular connectivity framework exposes standard APIs for easy integration of any application interacting with a remote host using the TCP- or UDP-over-IP protocol.

The X-CUBE-CELLULAR Expansion Package for STM32Cube provides an application example that connects and subscribes to cloud services using the HTTP protocol in order to report data from the device to the server, as well as to receive commands from the remote server. Two additional applications are provided as examples: ECHO (exchanges data with a remote server using TCP or UDP), and PING (tests the access to a remote machine).

X-CUBE-CELLULAR is available for the following hardware (refer to Chapter 1 for details):

- P-L496G-CELL01 cellular-to-cloud pack with an STM32L496AG-based Discovery host board and an add-on board based on Quectel UG96 modem (2G / 3G)
- P-L496G-CELL02 cellular-to-cloud pack with an STM32L496AG-based Discovery host board and an add-on board based on Quectel BG96 modem (LTE Cat M / NB-IoT / 2G fallback)
- B-L475E-IOT01A IoT Discovery board and add-on board with Quectel BG96 modem
- B-L475E-IOT01A IoT Discovery board and add-on board with Sequans® GM01Q modem
- 32L496GDISCOVERY Discovery board and modem board based on Sequans® GM01Q, or Quectel UG96 or BG96 modem

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

- Ready-to-run firmware examples using the 2G, 3G, LTE Cat M1, or NB-IoT protocols
- Configuration of the HTTP and MQTT connections to the IoT platform and cellular connectivity
- Cellular connection
- Reporting of such values as temperature, humidity, and pressure
- Network radio level reporting
- Testing of access to a remote machine
- Connection and data exchange using the TCP or UDP socket protocols
# Contents

1  General information .................................................. 6  
   1.1 Terms and definitions ............................................. 7  
   1.2 References .......................................................... 8  

2  Important note regarding the security .............................. 9  

3  Service connectivity description .................................... 10  

4  Package description .................................................. 11  
   4.1 General description ................................................ 11  
   4.2 Modem socket versus LwIP ........................................... 11  
   4.3 Architecture .......................................................... 11  
     4.3.1 Architecture concept ......................................... 12  
     4.3.2 Static architecture view .................................... 14  
     4.3.3 Dynamic architecture view ................................ .. 17  
   4.4 X-CUBE-CELLULAR Expansion Package description .............. 25  
   4.5 Folder structure .................................................... 26  
   4.6 Reset push-button .................................................. 28  
   4.7 Real-time clock ...................................................... 28  

5  Cellular connectivity examples ....................................... 29  
   5.1 Real network or simulator ......................................... 29  
   5.2 Connection overview ................................................ 29  
   5.3 PING example ........................................................ 30  
   5.4 ECHO example ......................................................... 31  
   5.5 GroveStreams (HTTP) access example ............................... 31  
   5.6 MQTT example ........................................................ 33  
   5.7 COM example .......................................................... 35  

6  Hardware and software environment setup .......................... 36  

7  Interacting with the host board ...................................... 41  
   7.1 Debug ................................................................. 42
8 How to customize the software? ................................................... 43
  8.1 First customization level: user customization ............................ 43
  8.2 Second customization level: advanced user customization ............ 43
     8.2.1 Adding/removing an application in firmware .................... 43
     8.2.2 IP stack on MCU side or on modem side ......................... 44
     8.2.3 Different kinds of available traces ............................... 45
     8.2.4 How to configure traces? ...................................... 46
  8.3 Third customization level: developer customization .................... 46
     8.3.1 Boot .................................................. 46
     8.3.2 Initialization of software components .......................... 46
     8.3.3 Software customization ....................................... 47
     8.3.4 Firmware adaptation to a new HW configuration ............... 47
     8.3.5 Adding a new component ...................................... 47
  8.4 Data cache ......................................................... 49
  8.5 Thread stack consumption monitoring .................................. 50

Appendix A Support material ....................................................... 52
  A.1 How to configure a GroveStreams account? ............................ 52
  A.2 How to activate the soldered SIM card? ............................... 54
  A.3 How to select Quectel BG96 modem configuration bands? ........... 55
  A.4 How to configure the Network library API? ........................... 55

Revision history ................................................................. 56
List of tables

Table 1. List of acronyms ................................................................. 7
Table 2. Compilation variables for applications in firmware ......................... 43
Table 3. Compilation variable for IP stack selection .................................. 45
Table 4. New thread registration example ............................................. 50
Table 5. Number of project threads setting example .................................. 50
Table 6. Document revision history ...................................................... 56
List of figures

Figure 1. Cellular IoT connectivity ................................................................. 10
Figure 2. Architecture concept ................................................................. 12
Figure 3. Static architecture view ............................................................... 14
Figure 4. Dynamic architecture - X-CUBE-CELLULAR API - Platform initialization and start 18
Figure 5. Dynamic architecture - X-CUBE-CELLULAR API - Up to PDN creation 19
Figure 6. Dynamic architecture - X-CUBE-CELLULAR API - Socket .......................... 20
Figure 7. Dynamic architecture - Network library API - Initialization .................. 21
Figure 8. Dynamic architecture - Network library API - Connection ..................... 22
Figure 9. Dynamic architecture - Network library API - Disconnection .................. 23
Figure 10. Dynamic architecture - Network library API - Socket service ............... 24
Figure 11. X-CUBE-CELLULAR folder structure (1 of 2) .................................... 26
Figure 12. X-CUBE-CELLULAR folder structure (2 of 2) .................................... 27
Figure 13. GroveStreams connection overview ............................................... 29
Figure 14. GroveStreams web interface, component view .................................. 32
Figure 15. GroveStreams web interface, dashboard view .................................. 32
Figure 16. STMicroelectronics MQTT dashboard ........................................... 34
Figure 17. Node-RED® example ................................................................ 35
Figure 18. Hardware setup (P-L496G-CELL02 example) ................................. 37
Figure 19. Hardware view (P-L496G-CELL02 example) ...................................... 38
Figure 20. Hardware view ("Discovery IoT node cellular with BG96" set example) .... 39
Figure 21. Serial port settings to interact with the host board ......................... 41
Figure 22. Serial port settings to interact with the host board (new-line) ............. 41
Figure 23. GroveStreams organization creation acceptance screen ................... 52
Figure 24. GroveStreams organization creation screen .................................... 52
Figure 25. GroveStreams organization access screen ....................................... 53
Figure 26. GroveStreams organization administration menu ............................ 53
Figure 27. GroveStreams API key selection screen ......................................... 54
Figure 28. GroveStreams API key display screen ........................................... 54
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.

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

- Ready-to-run firmware examples using the 2G, 3G, LTE Cat M1, or NB-IoT protocols to support quick evaluation and development of IoT cloud applications
- Menu and command line through Virtual COM UART over USB ST-LINK to configure the cellular connectivity parameters (technology selection, bands, APN, and others), and connection to the GroveStreams cloud IoT platform (HTTP)
- Cellular connection
- Reporting of such values as temperature, humidity, and pressure. The values are real if the MEMS add-on board (X-NUCLEO-IKS01A2) is connected, otherwise they are simulated. The sensors in the “Discovery IoT node cellular” set are always used
- Network radio level reporting
- PING application to test the access to a remote machine
- ECHO application to provide an example of connection and data exchanges using the TCP or UDP (connected or not-connected mode) socket protocols
- MQTT application to exchange data between the device and the cloud. On the cloud side, a dashboard is the interface with the end user
- COM application to test the access to other services than sockets provided by the COM library (such as read information from ICC)
- CUSTOM application, an empty application already pre-integrated in X-CUBE-CELLULAR. It is provided to simply add application code and observe its execution

X-CUBE-CELLULAR is available for the following hardware:

- both the P-L496G-CELL01 and P-L496G-CELL02 cellular-to-cloud packs. Each pack is composed of an STM32L496AG-based Discovery host board connected to an add-on cellular modem through the STMod+ connector:
  - The add-on board of P-L496G-CELL01 is equipped with the UG96 modem (2G / 3G) from Quectel
  - The add-on board of P-L496G-CELL02 is equipped with the BG96 modem (LTE Cat M / NB-IoT / 2G fallback) from Quectel
- the “Discovery IoT node cellular” set, which is a combination of the B-L475E-IOT01A IoT Discovery board, X-NUCLEO-STMODA1 ARDUINO® / STMod+ adapter, and:
  - either STMicroelectronics MB1329 modem board with Quectel BG96 modem
  - or Sequans® GM01Q-STMOD modem board with the GM01Q modem (referenced as B-CELL-GM01Q in STMicroelectronics)
- 32L496GDISCOVERY Discovery board connected to one of these modem boards:
  - Sequans® GM01Q-STMOD with the GM01Q modem
  - Add-on board with the Quectel UG96 or BG96 modem

Refer to the X-CUBE-CELLULAR cellular connectivity Expansion Package porting on other hardware application note (AN5249) for adaptation to other hardware such as the “Discovery IoT node cellular” set.
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

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application programming interface</td>
</tr>
<tr>
<td>APN</td>
<td>Access point name</td>
</tr>
<tr>
<td>BSD</td>
<td>Berkeley software distribution</td>
</tr>
<tr>
<td>BSP</td>
<td>Board support package</td>
</tr>
<tr>
<td>C2C</td>
<td>Cellular to cloud</td>
</tr>
<tr>
<td>CID</td>
<td>Context ID (context identifier of a cellular connection)</td>
</tr>
<tr>
<td>COM</td>
<td>Cellular communication</td>
</tr>
<tr>
<td>DC</td>
<td>Data Cache</td>
</tr>
<tr>
<td>eUICC</td>
<td>Embedded UICC (UICC with remote profile feature)</td>
</tr>
<tr>
<td>eSIM</td>
<td>Embedded SIM</td>
</tr>
<tr>
<td>FEEPROM</td>
<td>Represents the embedded Flash memory of the STM32 MCU</td>
</tr>
<tr>
<td>HAL</td>
<td>Hardware abstraction layer</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext transfer protocol</td>
</tr>
<tr>
<td>ICC</td>
<td>International circuit card</td>
</tr>
<tr>
<td>ICCID</td>
<td>International circuit card identifier</td>
</tr>
<tr>
<td>ICMP</td>
<td>Internet control message protocol</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated development environment</td>
</tr>
<tr>
<td>IF</td>
<td>Interface</td>
</tr>
<tr>
<td>IoT</td>
<td>Internet of things (refer to [4])</td>
</tr>
<tr>
<td>IPC</td>
<td>Inter-processor channel</td>
</tr>
<tr>
<td>ITM</td>
<td>Instruction trace module</td>
</tr>
<tr>
<td>LED</td>
<td>Light-emitting diode</td>
</tr>
<tr>
<td>M2M</td>
<td>Machine to machine</td>
</tr>
<tr>
<td>MQTT</td>
<td>Message queuing telemetry transport</td>
</tr>
<tr>
<td>NAT</td>
<td>Network address translation</td>
</tr>
<tr>
<td>NFMC</td>
<td>Network-friendly management configuration (refer to [4])</td>
</tr>
<tr>
<td>NIFMAN</td>
<td>Network IF manager</td>
</tr>
<tr>
<td>MNO</td>
<td>Mobile network operator</td>
</tr>
<tr>
<td>MVNO</td>
<td>Mobile virtual network operator</td>
</tr>
<tr>
<td>PDN</td>
<td>Packet data network</td>
</tr>
<tr>
<td>PDU</td>
<td>Protocol data unit</td>
</tr>
</tbody>
</table>
The X-CUBE-CELLULAR Expansion Package runs on STM32L4 32-bit microcontrollers based on the Arm® Cortex®-M4 processor.

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. X-CUBE-CELLULAR cellular connectivity Expansion Package porting on other hardware application note (AN5249)
6. X-CUBE-CELLULAR Application Programming Interface available as file EmbSw_application_programming_interface_Cellular_Package.chm in X-CUBE-CELLULAR Expansion Package

---

a. Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.
2 Important note regarding the security

Caution: Application developers must take care of security aspects, and put mechanisms in place to protect the tokens and secrets used for the connections.

The application example provided in the X-CUBE-CELLULAR Expansion Package does not implement such protection mechanisms. It only presents a basic implementation for an easy understanding of the stack interface.

Warning: Use the HW only with the antenna connected. With no antenna connected, there is a risk of damage to the modem because of the power reflected from the antenna connector to the modem RF output.
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- and application-level stack in C language, which allows the connection of the C2C kit to a web site.

The first connectivity example provided connects to the GroveStreams web site. In this example, the board reports notifications to the GroveStreams web browser.

The second example provided implements the ping network-testing feature.

The third example provided is the ECHO application. The application connects or not to the remote server (according to the socket protocol used), sends a buffer, and expects in return the same buffer as the response.

The fourth example provided is the MQTT application. The application sends data from the device to the cloud and receives data from the end user by means of a dashboard.

An other example, called the CUSTOM application, provides an empty skeleton already integrated in X-CUBE-CELLULAR. Add code inside the main task, activate custom rebuild, and observe the code running.

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

![Figure 1. Cellular IoT connectivity](MSv50384V2)

The P-L496G-CELL01 and P-L496G-CELL02 are used to represent all the compliant hardware.

The cellular-to-cloud kit comprises an STM32-based main board, a cellular add-on modem board, and a prepaid SIM, which enables the registration to a cellular PLMN. The global roaming provided by the SIM provider allows device attachment from any country. The SIM only offers IP connectivity (meaning that SMS is not supported). The volume of data available in the prepaid offer depends on where it is used.

A private IP address is allocated to the device by the MNO or MVNO. Any client application running on device using TCP transaction request/response can reach a server located in the Internet by means of IP address translation (NAT) on the MNO/MVNO router.
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 Embedded Workbench® for Arm® (EWARM)
- Keil® Microcontroller Development Kit (MDK-ARM)

Note: Refer to the release note available in the root folder of the delivery package 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. The generated FW is either for modem socket or for LwIP use. It is not possible to further change this setting through the boot menu.

Note: If the modem socket is used, the software described in this user manual limits data plane support to TCP or UDP IPv4 Client application only. TCP or UDP server modes and IPv6 are not supported.

If LwIP is used, TCP and UDP (both server and client) are fully 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.

Note: The LwIP mode is not supported for the “Discovery IoT node cellular” set.

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.

Note: Some parts of X-CUBE-CELLULAR can be used in a bare OS environment. The complete stack only runs with FreeRTOS™.
The package is split into the following components:

- STM32L4 Series HAL
- CMSIS/FreeRTOS™
- Network
- LwIP
- AT Service
- Cellular Service
- Cellular_Mngt
- Com
- Data_Cache
- IPC
- NIFMAN
- Common

### 4.3.1 Architecture concept

This section provides a high-level view of the software architecture supporting cellular connectivity, which is illustrated in *Figure 2.*
The cellular connectivity stack exposes two main interfaces to the application:

- The control plane interface: there are two interfaces for control. The Cellular_Mngt library provides an API to initialize SW components and starts the Cellular Service. The Data Cache interface is used to read information related to cellular network like Signal Strength (RSSI), and to get event notification like network registration state changes and network interface readiness.

- The data plane interface: also referred to as the Com interface, it is used to send and receive TCP or UDP segments to and from a remote client or server. The interface is based on standard BSD socket API in order to ease the integration of the application.

The IPC layer abstracts the actual HW bus interface used with the modem. The IPC supports two logical channels, each composed of one Tx (to the IPC) and one Rx (from the IPC). One is used for exchanging AT commands with the modem while the other one is used to carry PPP frames when LwIP is used. The selection of the active channel is controlled by Cellular Service.
4.3.2 Static architecture view

X-CUBE-CELLULAR static architecture is presented in Figure 3.

Figure 3. Static architecture view
API description

The application has access to two different API levels:
- X-CUBE-CELLULAR API
- Network library API

X-CUBE-CELLULAR API

The X-CUBE-CELLULAR API of the cellular package is defined by three components.
- Cellular management (Cellular_Mngt): initialization/starting of cellular features
- Com: data communication API based on the BSD API
- Data_Cache: this component allows the sharing of data and information between components. Particularly it allows application to be notified of each Data_Cache update and to get network information and status

Network library API

Another level of API is supplied by the Network library component. This API is above the X-CUBE-CELLULAR API.

The goal of this component is to standardize network API between Wi-Fi®, Ethernet and cellular media.

This component supplies:
- The management of the network initialization
- The data communication API based on the BSD API
- Network notifications, done by a callback recorded by the application at initialization

Component description

- HTTP client: implements an HTTP client, which sends requests to the www.grovestreams.com cloud in the application. The HTTP client uses the Data Cache to monitor the network interface state changes (from NIFMAN), and the COM socket interface to send or receive HTTP packets over TCP. The HTTP client also implements recovery as defined by GSMA TS 34 when a remote HTTP server is not reachable.
- ECHO client: 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.
- PING client: implements a ping application that tests the access to a remote machine.
- MQTT client: implements the usage of MQTT client with both publisher and subscriber roles. Publisher is used to send data to the cloud while subscriber is used to be notified from the cloud.
- COM client: implements an example application for Com library services usage other that sockets services (such as ICC communication).
- CUSTOM client: implements a pre-integrated and empty client where application code can be easily added and executed inside X-CUBE-CELLULAR.
- Connectivity Service layer, API for Client applications:
  - Data_Cache: framework that decouples the management of producer and consumer data (resource). Any resource state updated by a producer is pushed to the Data Cache (in RAM), which in turn informs the final consumer(s) to process the updated resource state via the callback provided by the consumer application. Data Cache is used by Cellular Service tasks to publish the Cellular network.
information like RSSI. It is also used by NIFMAN to publish the network interface readiness.

- **Com**: a library that provides a collection of BSD-like socket functions to open, configure, and send or receive application PDU to remote TCP or UDP applications. A high-level ping service is also provided.

- **Cellular_Mngt**: a library that exposes a basic function to initialize and start Cellular Service components.

- **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 informs NIFMAN to setup the network interface (PPP link). 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]. For example, when PDP activation fails because of a wrong APN, the Cellular Service task performs a new attempt after expiration of a back-off timer. The Cellular Service task stores the cellular configuration and network access parameters into the Flash memory, and configure the modem as per need. 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 to low-level Cellular Service. 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**: support 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.

- **NIFMAN**: the network interface manager task controls the network interface activation.
  When LwIP is used, NIFMAN monitors the PPP server status (on the modem side), and
  starts or stops the PPP client accordingly. The application can then monitor the network
  interface status before opening a socket for data transfer.

- **IPC**: abstracts the actual physical interface (UART) to the upper layer. Supports the
  logical channel handler (FIFO) that is mapped to a physical channel. Supports two
  channels: character mode and stream mode:
  - Stream mode is used for data transfer (PPP).
  - Character mode is used to send AT commands.

- **Common**: provides tools such as debug and trace. Also provides the setup menu (over
  any terminal through a serial interface) to change the default configuration, which is
  hard coded during compilation and image creation.

- **FreeRTOS™ (and CMSIS)**: 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. It finally initializes and starts the Cellular
  Service components by calling `cellular_init()` and `cellular_start()`.

### 4.3.3 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:

- **X-CUBE-CELLULAR API**
  - *Figure 4: Dynamic architecture - X-CUBE-CELLULAR API - Platform initialization
    and start*
  - *Figure 5: Dynamic architecture - X-CUBE-CELLULAR API - Up to PDN creation*
  - *Figure 6: Dynamic architecture - X-CUBE-CELLULAR API - Socket*

- **Network library API**
  - *Figure 7: Dynamic architecture - Network library API - Initialization*
  - *Figure 8: Dynamic architecture - Network library API - Connection*
  - *Figure 9: Dynamic architecture - Network library API - Disconnection*
  - *Figure 10: Dynamic architecture - Network library API - Socket service*
Figure 4. Dynamic architecture - X-CUBE-CELLULAR API - Platform initialization and start

- Initializes the Com socket descriptor tables
- Initializes the Cellular SW resources (such as mutex, default settings values and others)
- Initializes the cellular data caches content
- Initializes the network interface manager

<app>_init()

- Configures the Cellular Service settings: SIM card slot, RSSI polling period, Network registration timer, PDN activation retry timers (NFMC), Modem DFOTA monitoring timer
- Starts the Cellular Service task
- Starts the radio interface and tasks creation

<app>_start()

Optional call, making it possible to overwrite the cellular platform default settings (such as SIM slot, APN and others)

Legend:
Function call
Function return
Once started, the Cellular Service task performs actions to power the modem and any action to register to Network and activate PDN.

app_callback(event_id=DC_COM_RADIO_LTE_INFO)

dc_com_write(RADIO_LTE_INFO: ON)

Checks the SIM PIN code; reads IMSI and others information.

app_callback(event_id=DC_COM_SIM_INFO)

dc_com_write(SIM_INFO: IMSI)

Once the modem is initialized, CST forces the modem to perform “automatic” Network search and monitor the signal strength. If the signal quality is good enough, it performs network registration for CS and PS domains and finally activates the PDN.

app_callback(event_id=DC_COM_CELLULAR)

dc_com_write(CELLULAR_INFO: RSSI)

When PDN is ready (IP add available), CST informs NIFMAN via Data_Cache. NIFMAN in turn updates the Network Interface State to ON.

app_callback(event_id=DC_COM_NIFMAN)

dc_com_read(DC_COM_NIFMAN_INFO)

Legend:
- Function call
- Function return
Figure 6. Dynamic architecture - X-CUBE-CELLULAR API - Socket

Legend:
- Function call
- Function return

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com_socket()</td>
<td>Optional</td>
</tr>
<tr>
<td>com_bind()</td>
<td>Optional</td>
</tr>
<tr>
<td>com_setopt()</td>
<td>Optional</td>
</tr>
<tr>
<td>com_connect()</td>
<td>Depends on the protocol used (connected or not)</td>
</tr>
<tr>
<td>com_send()</td>
<td>Send data</td>
</tr>
<tr>
<td>com_recv()</td>
<td>Receive data</td>
</tr>
<tr>
<td>com_close()</td>
<td>Optional</td>
</tr>
</tbody>
</table>
When the modem is powered on and the SIM ready, an event is reported via callback, which generates a message to Cellular input.

Power on the modem and check that the SIM card is ready: target_state = SIM ONLY

When the modem is powered on and the SIM ready, an event is reported via callback, which generates a message to Cellular input.
Figure 8. Dynamic architecture - Network library API - Connection

```
net_if_connect() -> net_cellular_if_connect()

@app> event_handler_cb
net_if_notify(NET_STATE_CONNECTING)

NET_OK
Cellularnotif_cb (cst_state)
If NIFMAN_INFO state == SERVICE_ON, then inform that the net interface state changed to CONNECTED.

@app> event_handler_cb
net_if_notify(NET_STATE_CONNECTED)
```

Legend:
- Function call
- Function return

Set target_state = FULL to data cache.
Figure 9. Dynamic architecture - Network library API - Disconnection

```
If modem == SIM CONNECTED

net_if_disconnect() \rightarrow net_cellular_if_disconnect()

<app>event_handler_cb

net_if_notify(NET_STATE_CONNECTING)

<app>event_handler_cb

net_if_disconnect() \rightarrow net_cellular_if_disconnect()

Set target_state = SIM ONLY.

<app>event_handler_cb

net_if_notify(NET_STATE_DISCONNECTED)
```

Legend:
- Function call
- Function return
Figure 10. Dynamic architecture - Network library API - Socket service

Legend:
- Function call
- Function return

Net OK
4.4 X-CUBE-CELLULAR Expansion Package description

This section describes the software components of the X-CUBE-CELLULAR package. X-CUBE-CELLULAR is an Expansion Package for STM32Cube. Its main features are:

- Fully compliant with STM32Cube architecture
- Expands STM32Cube in order to enable the development of applications accessing and using various cloud platforms
- Based on the STM32CubeHAL, which is the hardware abstraction layer for STM32 microcontrollers

The software components used by the application are:

- STM32Cube HAL
  The HAL driver layer provides a generic multi-instance simple set of APIs (application programming interfaces) to interact with the upper layers (application, libraries and stacks).
  It is composed of generic and extension APIs. It is directly built around a generic architecture and allows the layers that are built upon, such as the middleware layer, implementing their functionalities without dependencies on the specific hardware configuration for a given microcontroller unit (MCU).
  This structure improves the library code reusability and guarantees an easy portability onto other devices.

- Board support package (BSP)
  The software package needs to support the peripherals on the STM32 boards apart from the MCU. This software is included in the board support package (BSP). This is a limited set of APIs which provides a programming interface for certain board specific peripherals such as the LED and the User button.

- Application
  HTTP, ECHO, PING, MQTT, COM, and CUSTOM clients.

- Middleware
  Optionally LwIP.

- FreeRTOS™
  FreeRTOS™ is mandatory to run the tasks for X-CUBE-CELLULAR components.

- Configuration files
  component and platform configuration file are provided at project repository
  - `plf_features.h` defines the feature list to include in firmware.
  - `plf_hw_config.h` defines the mapping of the GPIO and HW interface specific to logical names to ease SW porting to another board. It also provides the HW bus interface configuration (such as the UART) used for communicating with the modem.
  - `plf_sw_config.h` provides platform SW configuration such as trace, application behaviors, which can differ from one platform to another, such as modem polling timer value.
  - `plf_thread_config.h` defines the thread configuration (priority, stack size) and calculates the global heap needed.

Other parameters can be customized. Additional details are provided in Chapter 8: How to customize the software? on page 43.

Some parameters can also be defined dynamically at runtime. They are stored in the Flash memory and re-used at the next platform boot if needed.
4.5 Folder structure

*Figure 11* and *Figure 12* present the folder structure of the X-CUBE-CELLULAR Expansion Package with middleware folder break down.

*Figure 11. X-CUBE-CELLULAR folder structure (1 of 2)*
Figure 12. X-CUBE-CELLULAR folder structure (2 of 2)

- Internal and external middleware
- STMicroelectronics (internal) middleware
- Middleware core
- Middleware interface for applications
- Client applications for cellular connectivity examples
- Third-party (external) middleware
4.6 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.

4.7 Real-time clock

System date and time are managed by the RTC.

At boot time, it is possible to update the system date manually at boot setup (if available). If no such update is performed, the system date is updated by the ECHO or HTTP client using the information sent by the remote host.

Note: If the system date is updated by ECHO or HTTP, the hour value depends on the remote server time zone.
5 Cellular connectivity examples

This chapter describes the cellular connectivity available examples. Several examples that can run in parallel are provided, such as PING, ECHO, HTTP, MQTT, COM, and CUSTOM clients.

5.1 Real network or simulator

The worldwide coverage of 2G and 3G networks makes it possible to systematically run the examples for these technologies on real networks. The LTE cat M1 and NB1 technologies do not yet offer a similar global coverage. If such networks are not available, the user must use a Cat M1 / NB1 compliant network simulator.

In this document, assumption is made that a real network is available whatever the network technology used.

The add-on modem boards (in P-L496G-CELL01 and P-L496G-CELL02) embed an eSIM, provisioned with the EMnify MVNO profile:

- For Quectel UG96 modem, 2G or 3G real networks are used
- For Quectel BG96 modem, with the EMnify eSIM, 2G fallback is possible and sometimes, depending on the country, LTE Cat M1 is also available. To be sure to be able to use the BG96 modem in Cat M1 or NB1 mode, the user must insert a plastic SIM card compliant with this network technology from an MNO that has already deployed it.

5.2 Connection overview

The GroveStreams connection overview is presented in Figure 13.

Figure 13. GroveStreams connection overview

The P-L496G-CELL01 and P-L496G-CELL02 are used to represent all the compliant hardware.
5.3 PING example

The console command allows the generation of ping requests (refer to Section 7.3: Console command on page 42). The simple ping command generates 10 ping requests using default IP address 8.8.8.8. To specify another IP address, use the ping <ddd.ddd.ddd.ddd> command.

Ping command example:

ping 8.8.4.4

Ping result example:

<<< Ping Activated 8.8.4.4>>>

>Ping: 32 bytes from 8.8.4.4: seq=01 time= 73ms ttl=50
Ping: 32 bytes from 8.8.4.4: seq=02 time= 77ms ttl=50
Ping: 32 bytes from 8.8.4.4: seq=03 time= 48ms ttl=50
Ping: 32 bytes from 8.8.4.4: seq=04 time= 59ms ttl=50
Ping: 32 bytes from 8.8.4.4: seq=05 time= 78ms ttl=50
Ping: 32 bytes from 8.8.4.4: seq=06 time= 74ms ttl=50
Ping: 32 bytes from 8.8.4.4: seq=07 time= 56ms ttl=50
Ping: 32 bytes from 8.8.4.4: seq=08 time= 70ms ttl=50
Ping: 32 bytes from 8.8.4.4: seq=09 time= 66ms ttl=50
Ping: 32 bytes from 8.8.4.4: seq=10 time= 66ms ttl=50

--- 8.8.4.4 Ping Statistics ---
Ping: min/avg/max = 48/66/78 ms ok = 10/10
<<< Ping Completed >>>
5.4 ECHO example

The ECHO application is a client that exchanges data with a remote server using the TCP or UDP (connected or not-connected mode) socket. ECHO is the only example activated by default in the generated binaries (meaning that data exchange with the pre-configured remote server starts automatically when software finishes to boot):

- The default data exchanged with the server is the current "date and time". The response expected from the server is the exact same buffer (an echo).
  - Format of the default data exchanged (size 21 bytes):
    dd/mm/yyyy - hh:mm:ss
  - Prompt command for changing the size of the data exchanged:
    echoclient size <n>
    where n is the total data size (minimum size is 21 bytes)

In addition to the default data, a loop of numbers from 0 to 9 is used to reach the expected size.

- The default protocol used to create the socket is UDP not-connected mode if the modem supports this mode, or UDP connected otherwise.
  The protocol used is changed with the prompt command
  echoclient protocol <value>
  - <value> = TCP: TCP
  - <value> = UDP: UDP connected mode protocol
  - <value> = UDPSERVICE: UDP not-connected mode protocol
    (only proposed if the modem supports it)

Note: if the client is already active (exchange of data with a server) while the echo client protocol command is typed, the socket is closed before a new one is created with the new protocol requested.

- The echoclient on command activates the ECHO client application.
- The echoclient off command deactivates the ECHO client application.

5.5 GroveStreams (HTTP) access example

The cellular connectivity demonstration for GroveStreams consists in two use cases:

- The device periodically reports sensor data to the GroveStreams cloud IoT platform. The end-user connects to the GroveStreams web server dashboard.
- The end-user controls the LED application state from the dashboard.

The HTTP client on the MCU sets up a connection to the GroveStreams server through a dedicated account. It pushes data (temperature, humidity, pressure and cellular radio signal strength) to the GroveStreams server through the HTTP PUT command. An end-user connected to the GroveStreams platform with a standard web browser has access to the GroveStreams dashboard web page. The device polls the GroveStreams web server every two seconds by sending HTTP GET commands to retrieve any request from the end-user. By this mechanism, the end-user can switch the application LED on the host board ON and OFF.
For temperature, humidity, and pressure, if the X-NUCLEO-IKS01A2 board with sensors is plugged, real values are sent to the cloud server. Otherwise, simulated data are reported. With the “Discovery IoT node cellular” set, real values are always sent. 

*Figure 14* and *Figure 15* present the related GroveStreams interfaces.

*Figure 14.* GroveStreams web interface, component view

*Figure 15.* GroveStreams web interface, dashboard view
5.6 MQTT example

The MQTT example is an application using the MQTT-C stack. The MQTT-C uses the Network library API (ST middleware) to send and receive data.

When the device sends data, it has the publisher role; when it receives data, it has the subscriber role. Both publish and subscribe are performed in MQTT topics, which can be considered as pipes.

To achieve this way of working, an MQTT broker (server) on the Internet is needed. Additionally, the Node-RED® application is used to provide a user dashboard.

In the Node-RED® description, some nodes are MQTT oriented, meaning that they can get data sent on topics and put data into topics. Some other nodes are dedicated to the dashboard, such as buttons, curves, and gauges.

When the device publishes data in a topic, the MQTT broker and the associated Node-RED® node are informed, and the dashboard is updated with the new value.

When the user needs to manage the device, he can send data to the MQTT broker using the dashboard so that the device is informed.

To set up such a configuration, the user must start an Internet reachable server with MQTT (such as ECLIPSE Mosquitto™) and have Node-RED® installed. For even more simplicity, the MQTT example uses the Stackhero.io build-in service.

In less than ten minutes, the MQTT/Node-RED® configuration is up and running in a secure way (TLS is used).

Once the MQTT/Node-RED® is up, a few credentials must be reported into STM32 firmware, such as URL, user, and password. This can be done either by updating the .src files or using the setup menu at boot.

Whatever the MQTT/Node-RED® solution used, STMicroelectronics provides the .json file with all the sources needed to get the same dashboard as in the present example.
Figure 16 shows the STMicroelectronics dashboard used in this example.

Figure 16. STMicroelectronics MQTT dashboard

The pressure, humidity, temperature, operator name, IMSI, IMEI, and LEDs status are sent from the device to the cloud.

The LEDs and reboot button enable the user to manage the device.
Figure 17 shows the NodeRED code in the MQTT example:

Figure 17. Node-RED® example

5.7 COM example

The COM application implements an example of Com library services usage other than sockets services (such as ICC communication).

When activated in the build, interaction with this application is only through the serial terminal.

In this version of X-CUBE-CELLULAR, the COM application uses the Com library to interact with the ICC (using the CSIM protocol: APDU sends to the SIM through the modem using AT command AT+CSIM).

Enter the following commands in the serial terminal:
- `comclt icc imsi` to request and display the IMSI
- `comclt icc iccid` to request and display the ICCID
- `comclt icc` to test all predefined ICC commands
6  Hardware and software environment setup

STM32 MCU firmware must always be programmed. Modem firmware can be upgraded too if needed.

Note: For P-L496G-CELL01 and P-L496G-CELL02, before programming with new firmware, connect the system to a PC with Tera Term and note the voucher number; this number is needed to activate the eSIM on the modem board. If the board is flashed before getting the voucher, re-flash the original image and get the voucher.

The embedded SIM in the “Discovery IoT node cellular” set is not provisioned with a M(V)NO profile: No eSIM activation is needed; A plastic SIM card must be used instead.

For the P-L496G-CELL01 and P-L496G-CELL02 Discovery packs, connect the host board to the modem board by means of the STMod+ connector (refer to Figure 19). For the “Discovery IoT node cellular” set, first connect the MB1329 modem board to the X-NUCLEO-STMODA1 adapter board, then connect the X-NUCLEO-STMODA1 to the B-L475E-IOT01A host board (refer to Figure 20).

The GM01Q-STMOD modem board can be connected by means of the STMod+ connector either to the host board contained in P-L496G-CELL01 or P-L496G-CELL02, to the 32L496GDISCOVERY board, or to the B-L475E-IOT01A board through the X-NUCLEO-STMODA1 adapter. For the first three possibilities, refer to Figure 19; for the last one, refer to Figure 20.

When the UICC chip soldered on the add-on board is not used, insert a UICC compliant with the real network used into the UICC socket.

If the soldered UICC chip is used, it must be activated beforehand and selected at boot by means of the boot menu.

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.

Power bands are needed for instance when 2G is selected with particular radio conditions.

______________________________
Warning: Use the HW only with the antenna connected. With no antenna connected, there is a risk of damage to the modem because of the power reflected from the antenna connector to the modem RF output.
Figure 18 depicts the hardware setup.

Figure 18. Hardware setup (P-L496G-CELL02 example)
Figure 19 shows the P-L496G-CELL02 with the USB cable in place for power supply and optional trace / boot menu.

Figure 19. Hardware view (P-L496G-CELL02 example)
Summary of environment steps to setup the GroveStreams demonstration:

- Create the GroveStreams account (refer to A.1: How to configure a GroveStreams account? on page 52)
- Create a GroveStreams organization in the GroveStreams account (refer to A.1)
- Get the 2 needed API keys in the organization (refer to A.1)
- Update the provided GS_setup.txt file with the user API keys previously read
- Connect the modem add-on board with its antenna to the host STM32 board (refer to Figure 19)
- Connect the STM32 board to a PC via a USB cable (refer to Figure 19)
- Program the STM32 board by dragging and dropping the STM32 FW image onto the newly appeared drive
- Start the terminal and set the parameters (refer to Chapter 7: Interacting with the host board)
- Reboot the board (press the black button)
- Select item “2” in the boot menu (refer to Chapter 7: Interacting with the host board for a complete description), select the SIM card to be used and the APN if needed, set the GroveStreams parameters, and save in Flash. The APN name is provided by the M(V)NO.
- Reboot the board (the trace is displayed in the terminal)
- Connect to the GroveStreams account, select the dashboard, double click on sensor, observe the data displayed and the toggling of the LED induced by the GroveStreams dashboard.

There are 2 important txt files for GroveStreams:
- **Grovestreams_Blueprint.txt**: used during organization creation (use as such, no modification is needed)
- **Grovestreams_Setup.txt**: used to configure the generic FW, to access the GroveStreams account. This file must be updated with the 2 API keys replaced by the API keys of the organization (refer to A.1: How to configure a GroveStreams account? on page 52)

**Note:** File “Grovestreams_Setup.txt” is not mandatory. It is only an user-friendly method to enter personal GroveStreams parameters into the FW, avoiding to enter each parameter individually by using “File” > “Send file …” and selecting “Grovestreams_Setup.txt”.

Refer to Chapter 7: Interacting with the host board for entering GroveStreams parameters with file “Grovestreams_Setup.txt” by using the menu at boot (Setup Menu > GroveStreams).

Refer to Chapter 7: Interacting with the host board for selecting the UICC or SIM socket, and the APN, by using the menu at boot with (Setup Menu > Cellular Service).
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 21. 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 21. Serial port settings to interact with the host board

![Tera Term Serial port setup](MSv51208V2)

Figure 22 illustrates the menu for setting the terminal parameters. It is reached through Setup > Terminal. Set both Receive and Transmit New-line parameters to CR.

Figure 22. Serial port settings to interact with the host board (new-line)

![Tera Term Terminal setup](MSv51209V2)
Once all terminal and serial port parameters are properly set, press the board reset button (black).

The HTTP service is active as soon as STM32 SW is running, sending data to the GroveStreams cloud. The GroveStreams dashboard displays the data values and switches the device LED ON / OFF.

Parameter setting is possible at boot time through the console. Refer to **Section 7.2: Boot menu** for details.

### 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 `Projects\STM32L496G_Discovery\Demonstrations\Cellular\STM32_Cellular\App\plf_sw_config.h`. Details are provided in **Section 8.2.3: Different kinds of available traces**, **Section 8.2.4: How to configure traces?** and **Section 8.5: Thread stack consumption monitoring**.

### 7.2 Boot menu

The boot menu is enabled by setting the `USE_DEFAULT_SETUP` variable to 0, which is the default value, in configuration file `Projects\STM32L496G_Discovery\Demonstrations\Cellular\STM32_Cellular\App\plf_features.h`.

At boot stage, a menu is displayed via the serial console to select the firmware use:

- Launch configuration setup
- Modem boot only without starting firmware applications

If no character is entered after 3 seconds, firmware starts in normal mode.

### 7.3 Console command

After target boot, the commands presented in this section are available to get and set software component status.

- To activate the console command feature, the `USE_CMD_CONSOLE` compilation variable must set to 1 in file `Projects\STM32L496G_Discovery\Demonstrations\Cellular\STM32_Cellular\App\plf_features.h`.
  ```c
  #define USE_CMD_CONSOLE       (1)
  ```
- The board must be connected to a serial console as described in the introduction of **Chapter 7: Interacting with the host board on page 41**.
- Type `<Return>` after boot to display the help
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. This chapter also presents how to monitor thread stack consumption.

8.1 First customization level: user customization

The first customization level is the setup configuration at boot time (refer to Chapter 7: Interacting with the host board on page 41). 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 as presented in sections 8.2.1, 8.2.2, 8.2.3, and 8.2.4. Customization-induced recompilation is needed.

8.2.1 Adding/removing an application in firmware

The Projects\STM32L496G_Discovery\Demonstrations\Cellular\STM32_Cellular\App\plf_features.h configuration file allows the selection of the applications to be included in firmware.

Table 2 presents the compilation variables that can be defined or undefined as a function of the applications needed.

<table>
<thead>
<tr>
<th>Compilation variable(1)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#define USE_HTTP_CLIENT</td>
<td>Includes the HTTP client (GroveStreams) application.</td>
</tr>
<tr>
<td>#define USE_PING_CLIENT</td>
<td>Includes ping utilities. The PING application uses COM PING API functionalities. If USE_PING_CLIENT is defined, USE_COM_PING must be defined too (see the corresponding entry in this table).</td>
</tr>
<tr>
<td>#define USE_ECHO_CLIENT</td>
<td>Includes the ECHO client application.</td>
</tr>
<tr>
<td>#define USE_MQTT_CLIENT</td>
<td>Includes the MQTT client application.</td>
</tr>
<tr>
<td>#define USE_COM_CLIENT</td>
<td>Includes the COM client application.</td>
</tr>
<tr>
<td>#define USE_DC_MEMS</td>
<td>Includes sensor management Middlewares\ST\STM32_Cellular\Modules\DataCache\Supplier\Src\dc_mems.c. Note: this option can be set only on 32L496GDISCOVERY.</td>
</tr>
</tbody>
</table>
8.2.2 IP stack on MCU side or on modem side

The IP stack runs either on the MCU side or on the modem side. The default configuration is: modem side.

# define USE_SIMU_MEMS
Includes the simulation of sensors (no physical sensor Middlewares\STM32_Cellular\Modules\DataCache_Supplier\Src\dc_mems.c. This option can be used also with USE_DC_MEMS defined. It is useful if no sensor shield is plugged.

# define USE_DEFAULT_SETUP
Defines if the boot setup menu is used or not:
- 0: boot setup menu used
- 1: boot setup menu not used. In this case, default parameters are set. These parameters are defined in file <component>config.h for each component using the setup menu Middlewares\STM32_Cellular\Samples\HTTP\Inc\http_client_config.h Middlewares\STM32_Cellular\Samples\Ping\Inc\ping_client_config.h Middlewares\STM32_Cellular\Cellular_Service\Inc\cellular_service_config.h

# define USE_COM_PING
Includes PING functionalities in module COM. Because few memory is used with PING functionalities, this define provides the possibility not to include PING functions if PING is not used on platform. If USE_PING_CLIENT is defined, then USE_COM_PING has to be defined.

# define USE_COM_ICC
Includes ICC functionalities in module COM.

# define COM_SOCKETS_ERRNO_COMPAT
If activated, then when USE_SOCKETS_TYPE is set to USE_SOCKETS_MODEM, com_getsockopt with COM_SO_ERROR parameter returns a value compatible with errno.h (refer to file com_sockets_err_compat.c for the conversion).

# define USE_CMD_CONSOLE
Includes console command (refer to Section 7.3: Console command on page 42).

# define USE_STACK_ANALYSIS
Includes thread stack monitoring facilities.

# define USE_CELPERF
Includes cellular throughput performance measurements.

# define USE_DC_GENERIC
Manages the generic values to be sent to the GroveStreams server.

# define USE_LINK_UART
Gets commands from another board via the UART.

# define USE_NETWORK_LIBRARY
Activates the Network library layer.

1. All defines are independent: several applications can be selected together.
The `plf_features.h` configuration file includes the definition of the location of the IP stack used.

*Table 3* presents the compilation variable that defines the IP stack used.

<table>
<thead>
<tr>
<th>Compilation variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>#define USE_SOCKETS_TYPE</code></td>
<td>Defines the IP stack used:</td>
</tr>
<tr>
<td></td>
<td>– USE_SOCKETS_LWIP:</td>
</tr>
<tr>
<td></td>
<td>IP stack on the MCU side (LwIP stack)</td>
</tr>
<tr>
<td></td>
<td>– USE_SOCKETS_MODEM:</td>
</tr>
<tr>
<td></td>
<td>IP stack on the modem side(1)</td>
</tr>
</tbody>
</table>

1. If the IP stack used is on the modem side, only the TCP/UDP IPv4 Clients are supported.

### 8.2.3 Different kinds of available traces

Traces are centralized in the TraceInterface module.

C macros are defined in each module of X-CUBE-CELLULAR to manage the traces.

The user can easily modify or enrich the implementation according to his needs.

**Enhanced UART traces**

The traces are sent to the UART connected to the ST-LINK, which uses the TraceInterface module (for fine trace selection, pretty buffer display, and others).

This is the recommended trace option. It is activated by default.

**ITM traces**

The Instrumentation Trace Macrocell activates traces that are less intrusive than UART traces. The TraceInterface module offers a basic implementation of ITM traces, which the user can enrich according to his needs.

It is activated by default.

*Note: Enhanced UART traces and ITM traces are activated by default and provide the same trace at the same time on different communication channels.*

To visualize the traces, the STM32 ST-LINK Utility software must be installed.

The ITM trace is visualized after the following series of operation is performed:

1. Connect the USB ST-LINK connector of the board to the computer
2. Connect the ST-LINK Utility to the board (Menu Target>Connect)
3. Open the serial viewer (Menu STLINK>Printf via the SWO viewer)
4. Set the system clock to the correct value (usually 80 000 000 Hz)
5. Activate all stimulus ports
6. Press start to display the traces

*Note: ST-LINK Utility must be connected to the board. It is therefore not possible to debug via the IDE and visualize the ITM trace at the same time.*
Standard `printf` traces

It is possible to select standard `printf` traces instead of enhanced UART traces. It uses also the UART connected to the ST-LINK but offer less options and buffers are not displayed.

8.2.4 How to configure traces?

The configuration of traces is done in file `Projects\STM32L496G_Discovery\Demonstrations\Cellular\STM32_Cellular\App\plf_sw_config.h` by means of the following flags:

- **SW_DEBUG_VERSION**
  - Set this flag to 1 (default value) to enable debug traces
  - Set this flag to 0 to disable debug traces (only setup menu traces will be displayed)
- **TRACE_IF_TRACES_UART**
  - Set this flag to 1 to enable enhanced UART traces (default value)
  - Set this flag to 0 to disable enhanced UART traces
- **TRACE_IF_TRACES_ITM**
  - Set this flag to 1 to enable ITM traces (default value)
  - Set this flag to 0 to disable ITM traces
- **USE_PRINTF**
  - Set this flag to 1 to enable standard `printf` UART traces
  - Set this flag to 0 to disable standard `printf` UART traces (default value)

**Note:** When `printf` traces are activated, enhanced UART traces and ITM traces are disabled.

Each software module of X-CUBE-CELLULAR can be enabled or disabled using the corresponding flag beginning by "USE_TRACE_", such as `USE_TRACE_HTTP_CLIENT` or `USE_TRACE_PING_CLIENT` for instance.

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, I²C, or others).

8.3.2 Initialization of software components

The SW components (application and middleware) are initialized in file `freertos.c`. Each component comprises a static initialization to initialize its data structure `<Component>_Init`, and a real time initialization to start the component thread `<Component>_Start`. Both are called from the `StartDefaultTask` function in the right order.
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

8.3.4 Firmware adaptation to a new HW configuration

To adapt the firmware to a new board or new HW configuration, follow these steps:
1. Create an STM32CubeMX project based on the new board
2. Configure the HW IPs as configured for the 32L496GDISCOVERY board
3. Generate the software configuration files from STM32CubeMX
4. Update file `plf_hw_config.h` to match the GPIO and HW handler names generated in the configuration files

8.3.5 Adding a new component

The CUSTOM Client provides an empty skeleton application that enables the user to:
- simply add and easily integrate his own application code inside X-CUBE-CELLULAR
- reduce X-CUBE-CELLULAR to a minimum set of features for his own project
- easily use optional features like trace debug, command, and stack analysis

To use the CUSTOM Client, follow the steps 1 to 5 below.

**Step 1 - Activate the custom configuration**

Activate the pre-defined symbol `USE_CUSTOM_CONFIG` in the project options:

change line

```
USE_CUSTOM_CONFIG=0
```

to

```
USE_CUSTOM_CONFIG=1
```

This change causes the defines in file `Middlewares\ST\STM32_Cellular\Samples\Custom\Inc\plf_custom_config.h` to overwrite the configuration of X-CUBE-CELLULAR. As a result, `plf_custom_config.h` sets the X-CUBE-CELLULAR platform to a minimal set of features, with for instance all applications and optional features deactivated.
Step 2 - Activate the custom application and customize features

In Middlewares\STM32_Cellular\Samples\Custom\Inc\plf_custom_config.h:

- Activate the custom_client application:
  
  if not already done, change line
  
  USE_CUSTOM_CLIENT=0
  
  to
  
  USE_CUSTOM_CLIENT=1

- Activate the optional features needed.
  For instance:
  
  – For Ping Comlib access, activate define USE_COM_PING
  – For command console module access, activate define USE_CMD_CONSOLE

- Configure the FreeRTOS™ memory needs
  
  – According to the number of threads requested by the user’s own code. Configure
    define CUSTOMCLIENT_THREAD to this number
    (a minimum of one thread is requested for the CUSTOM Client task body)
  
  – According to the heap requested by your main task body code, configure define
    CUSTOMCLIENT_THREAD_STACK_SIZE to this number

If the user’s code needs more than one thread:

- For each extra thread, declare a specific define XXX_THREAD_STACK_SIZE and
  XXX_THREAD_PRIO

- Then define APPLICATION_HEAP_SIZE to the sum of all XXX_THREAD_STACK_SIZE
  defines

- If a stack analysis module is used to trace the heap consumption, add for each extra
  thread, its declaration to stack analysis following the instruction in Table 4: New thread
  registration example

Step 3 - Add user’s own application code

The CUSTOM Client offers all pre-integrated functions needed to add an application: init,
start, and thread body are already available and integrated in X-CUBE-CELLULAR.
How to customize the software?

In file Middlewares\ST\STM32_Cellular\Samples\Custom\Src\custom_client.c:

- In init function `custom_client_init()`
  - Add the initialization of internal static variables of the application
  - Add calls to init of extra modules if any
- In start function `custom_client_start()`
  - See that a custom client thread is already pre-defined (the custom client thread is meant to contain the main task body code of user’s application)
  - See that registration to the Data Cache module is already done; the Data Cache module provides information about the X-CUBE-CELLULAR platform to the application (modem is on, registration to network done, and others)
  - See that registration to command module (under define) is already pre-integrated
  - See that the declaration to the stack analysis module (under define) is already pre-integrated
  - Add request to other services (most often registration) and calls to start extra modules if any
- In thread task body function `custom_client_thread()`
  - Add the main code of the user’s application

Step 4 - Customize extra functionalities

Still in file Middlewares\ST\STM32_Cellular\Samples\Custom\Src\custom_client.c:

- If the monitoring of some X-CUBE-CELLULAR platform parameters is needed, customize callback: `custom_client_notif_cb()`
  This callback is called by the Data Cache each time a status of cellular platform changes.
- If the use of the command module is needed, customize functions
  `custom_client_cmd_help()` and `custom_client_cmd()`

Step 5 - Build, program and test the new piece of software

8.4 Data cache

The Data Cache allows the sharing of data and events by software components. A software component (producer) creates a data entry and writes data in it. Each data entry is associated to an identifier.

The other components (consumers) can read the data by means of the identifier.

A component can subscribe to a callback to be informed when Data Cache data entry has been updated.

The Data Cache structure includes the `rt_state` field. This field contains the state of service and the validity of entry data.

- DC_SERVICE_UNAVAIL: field values of structure not significant
- DC_SERVICE_ON: service started (field values of structure are significant)
- Other value are entry dependent

For more information, refer to [6].
8.5 Thread stack consumption monitoring

The Stack Analysis module enables the monitoring of the thread stack consumption. Each time a thread is created, its registration must be added to Stack Analysis to provide its stack size allocation (this is already done for all the threads declared in the project).

Table 4 shows, as for any new thread creation, the addition of a call to service stackAnalysis_addStackSizeByHandle().

Table 4. New thread registration example

```
#define THREAD_NUMBER                     
 (uint8_t)(USED_TCPIP_THREAD 
  +USED_DEFAULT_THREAD  
  +USED_FREERTOS_TIMER_THREAD 
  +USED_FREERTOS_IDLE_THREAD 
  +USED_PPPOSIF_CLIENT_THREAD 
  +USED_BOARD-buttonsTHREAD 
  +USED_ATCORE_THREAD 
  +USED_CELLULAR_SERVICE_THREAD 
  +USED_NIFMAN_THREAD 
  +USED_DC_MEMS_THREAD 
  +USED_CMD_THREAD 
  +USED_CUSTOMCLIENT_THREAD 
  +USED_ECHOCLEINT_THREAD 
  +USED_HTTPCLEDENT_THREAD 
  +USED_PINGCLIENT_THREAD 
  +USED_COMCLIENT_THREAD 
  +USED_MQTCLEINT_THREAD 
  +USED_NET_CELLULAR_THREAD 
  +USED_STACK_ANALYSIS_THREAD)
```

The number of project threads is calculated according to feature activation (refer to file plf_thread_config.h) as shown in Table 5.

Table 5. Number of project threads setting example

```
#define USE_STACK_ANALYSIS (0)
```

The monitoring is not activated by default. To activate it (printf of all thread stack size evolutions), change in file plf_features.h

```
#define USE_STACK_ANALYSIS (0)
```
#define USE_STACK_ANALYSIS (1)  
If an automatic monitoring is needed, update  
#define STACK_ANALYSIS_TIMER (0)  
with the expected timer value (expressed in ms).  
For instance, if STACK_ANALYSIS_TIMER is set to 300000U, the thread stacks status is displayed every 300000 milliseconds (meaning every 5 minutes).  
If a display on demand is preferred, call the stack analysis service called:  
stackAnalysis_trace();  
For more options regarding the configuration monitoring refer to file stack_analysis.h.
A.1 How to configure a GroveStreams account?

The different steps are:
1. Create an email account if needed (a valid email account is needed to create a GroveStreams account)
2. Create a GroveStreams account (free sign up)
3. Setup the GroveStreams account (create the organization with the blueprint)
4. Get the needed API keys (used in STM32 MCU firmware)

The series of figures from Figure 23 to Figure 28 illustrate the creation and configuration of a GroveStreams account.

Figure 23 shows the login screen for the creation of a GroveStreams organization.

The blueprint (template) is used for the creation of the organization as shown in Figure 24:
- Enter the organization name in field Organization Name
- Tick the Create with a custom blueprint option
- Select the Blueprint File
- Validate with Create Organization

Figure 24. GroveStreams organization creation screen
Get access to the organization as shown in Figure 25.

Figure 25. GroveStreams organization access screen

Prepare to copy the API keys by selecting Admin > API Keys as shown in Figure 26.

Figure 26. GroveStreams organization administration menu
Select *Feed Put API Key* as shown in *Figure 27*.

**Figure 27. GroveStreams API key selection screen**

![API Keys](image)

Select *View Secret Key* to display the key as illustrated in *Figure 28*.

**Figure 28. GroveStreams API key display screen**

![API Secret Key](image)

The API key can be copied and pasted as per need. Proceed similarly for the dashboard API key, by un-selecting *Feed Put API Key* before selecting *Dashboard API Key*.

### A.2 How to activate the soldered SIM card?

The C2C kits are provided with a UICC chip pre-provisioned by the EMnify MVNO, which allows worldwide 2G and 3G roaming. This is not an eUICC but an embedded SIM. It is a classical SIM profile provisioned in a chip during the factory process. It is not an eUICC that allows remote profile update.

*Note:* This section does not apply to the “Discovery IoT node cellular” set, which features no embedded SIM provisioned by an M(V)NO, neither to the GM01Q-STMOD modem board.

LTE Cat M1 and NB1 technologies are not supported by EMnify. In case of UICC activation on a board with Quectel BG96 modem, only 2G fallback is possible.

To use Quectel BG96 modem in Cat M1 or NB1 mode, the user must operate with the microSIM plastic card.

There is no restriction for add-on boards with Quectel UG96 modem.
The steps to activate EMnify on the UICC chip are:
1. Boot the board with a terminal connected so that a text can be read
2. Get the voucher from the displayed text
3. Connect with a PC to https://stm32-c2c.com
4. Register the board using the voucher
5. Select the EMnify item in the displayed board
6. Follow the activation procedure
7. The EMnify profile is immediately activated (double click on the SIM to display the SIM status)

A.3 How to select Quectel BG96 modem configuration bands?
The configuration of Quectel BG96 modem bands can be done using console commands (refer to Chapter 7: Interacting with the host board on page 41).

A.4 How to configure the Network library API?
By default, the cellular package is configured to use the X-CUBE-CELLULAR API.
To activate the Network library component, follow these steps:
1. Modify the IDE properties of the Network_Library component to compile it (by default the component is not compiled).
2. Modify the plf_feature.h file of the project to include the Network_Library component in cellular firmware.

The USE_NETWORK_LIBRARY compilation variable must be set to 1:
#define USE_NETWORK_LIBRARY (1) /* 0: not activated, 1: activated */

Example:
The ECHO application supplies an example of the two kinds of API (X-CUBE-CELLULAR and Network library). According to the USE_NETWORK_LIBRARY setting, this application uses either the X-CUBE-CELLULAR API or Network library API.

Notes:
1. Only the ECHO and MQTT applications offer a Network Library API compatibility.
2. As only one application can register a callback to Network library to be receive network notifications, only one application can be used at a time with the Network library API.
3. In case of the X-CUBE-CELLULAR API, each application can register to Data Cache to receive network notifications, and then can be used at the same time as the other applications.
## Revision history

### Table 6. Document revision history

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

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>Changes</th>
</tr>
</thead>
</table>
| 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 |
- 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 |
IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, please refer to www.st.com/trademarks. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2020 STMicroelectronics – All rights reserved