UM2347
User manual

Getting started with X-CUBE-CLD-GEN IoT cloud generic software expansion for STM32Cube

Introduction

This user manual describes the content and use of the X-CUBE-CLD-GEN IoT cloud generic Expansion Package for STM32Cube.

The X-CUBE-CLD-GEN Expansion Package consists of a set of libraries and application examples for STM32L4 Series, STM32F4 Series, and STM32F7 Series microcontrollers acting as end devices.

These examples implement IoT cloud applications using the MQTT and HTTP protocols, over Wi-Fi®, Ethernet, or cellular connections.

X-CUBE-CLD-GEN runs on five platforms:

- The B-L475E-IOT01A and 32F413HDISCOVERY boards support Wi-Fi® connectivity with an on-board Inventek ISM43362 module
- The 32F769IDISCOVERY board provides a native Ethernet interface
- The P-L496G-CELL01 and P-L496G-CELL02 packs, with the 2G/3G Quectel UG96 and LTE Quectel BG96 cellular modem daughterboards respectively, support cellular connectivity

For the five platforms, the sample applications configure the network connectivity parameters, and illustrate the various ways for a device to interact with the cloud.

An application shows how an MQTT client can connect to an MQTT broker in order to publish data and receive parameter updates or commands from the cloud.

Device authentication through MQTT login and password is supported. TLS encryption, server authentication and device authentication are supported.

The MQTT broker can be a self-administrated server like Eclipse Mosquitto™, or the Ubidots or Litmus Loop cloud platforms.

Other applications also demonstrate how a simple HTTP client can connect to either the Exosite, Grovestreams or Ubidots cloud platforms using the HTTP or HTTPS protocol.

The B-L475E-IOT01A board reports telemetry data such as measurements of humidity, temperature, 3-axis magnetic, acceleration, and gyroscope data, atmospheric pressure and distance.

Contents

List of tables

List of figures

1 General information

The X-CUBE-CLD-GEN Expansion Package runs on STM32 32-bit microcontrollers based on the Arm®(a) Cortex®-M processor. *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
APN	Access point number
BSP	Board support package
C2C	Cellular to cloud
CA	Certification authority
CLI	Command-line interface
DHCP	Dynamic host configuration protocol
DNS	Domain name server
ECDSA	Elliptic curve digital signature algorithm
HAL	Hardware abstraction layer
HTTP	Hypertext transfer protocol
HTTPS	Hypertext transfer protocol over SSL
IDE	Integrated development environment
IoT	Internet of things
IP	Internet protocol
JSON	JavaScript object notation
LED	Light-emitting diode
MQTT	Message queuing telemetry transport
MVNO	Mobile virtual network operator
RSA	Rivest–Shamir–Adleman cryptosystem
NAT	Network address translation
RTC	Real-time clock
SSL	Secure sockets layer
TLS	Transport layer security
UART	Universal asynchronous receiver/transmitter
VPN	Virtual private network

arm

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

2 X-CUBE-CLD-GEN IoT ecosystem

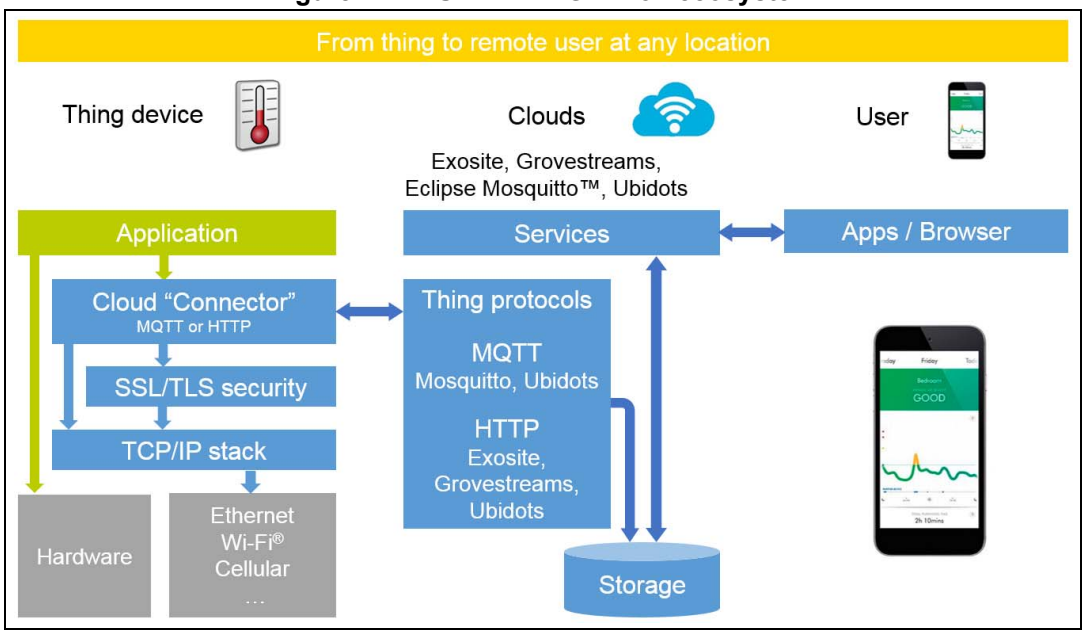
This chapter introduces the environment in which the X-CUBE-CLD-GEN Expansion Package can be used.

The X-CUBE-CLD-GEN Expansion Package implements connections to various cloud platforms using either a MQTT or a HTTP connection

A user can connect to the cloud with a smartphone or personal computer and have access to the information provided by the board at any time and from any location.

Figure 1 presents the IoT ecosystem targeted by the X-CUBE-CLD-GEN Expansion Package.

Figure 1. X-CUBE-CLD-GEN IoT ecosystem



3 Package description

This chapter details the X-CUBE-CLD-GEN package content.

3.1 General description

The X-CUBE-CLD-GEN Expansion Package consists of a set of libraries and application examples for STM32L4 Series, STM32F4 Series, and STM32F7 Series microcontrollers acting as end devices.

X-CUBE-CLD-GEN runs on five platforms:

- The B-L475E-IOT01A and 32F413HDISCOVERY support Wi-Fi® connectivity with an on-board Inventek ISM43362 module.
- The 32F769IDISCOVERY board provides a native Ethernet interface.
- The P-L496G-CELL01 and P-L496G-CELL02 packs, with the 2G/3G Quectel UG96 and LTE Quectel BG96 cellular modem daughterboards respectively, support cellular connectivity. The P-L496G-CELL01 STM32 Discovery pack for 2G/3G cellular to cloud (STM32-C2C/2G-3G) and P-L496G-CELL02 STM32 Discovery pack for LTE cellular to cloud (STM32-C2C/2G-LTE) are turnkey development platforms for cellular- and cloud-technology-based solutions. The packs are composed of an STM32L496AGI6-based low-power Discovery mother board with preloaded firmware, and an STMod+ cellular expansion board with antenna.

The X-CUBE-CLD-GEN Expansion Package contains the following components:

- Paho MQTT embedded C
- HTTP client library
- mbedTLS
- Inventek ISM43362 Wi-Fi® driver for the B-L475E-IOT01A and 32F413HDISCOVERY boards
- Ethernet driver, FreeRTOS™, and LwIP for the 32F769IDISCOVERY board
- Sensor drivers for the B-L475E-IOT01A board
- Cellular driver for the P-L496G-CELL01 and P-L496G-CELL02 packs
- BSPs for the four boards
- STM32L4 Series, STM32F4 Series, and STM32F7 Series HAL
- Application examples

The software is provided as a zip archive containing source code.

The following integrated development environments are supported:

- IAR Embedded Workbench® for Arm® (EWARM)
- Keil® Microcontroller Development Kit (MDK-ARM)
- System Workbench for STM32

Refer to the release notes available in the package root folder for information about the IDE versions supported.

3.2 Architecture

This section describes the software components of the X-CUBE-CLD-GEN package.

X-CUBE-CLD-GEN is an Expansion Package for the 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 software are the following:

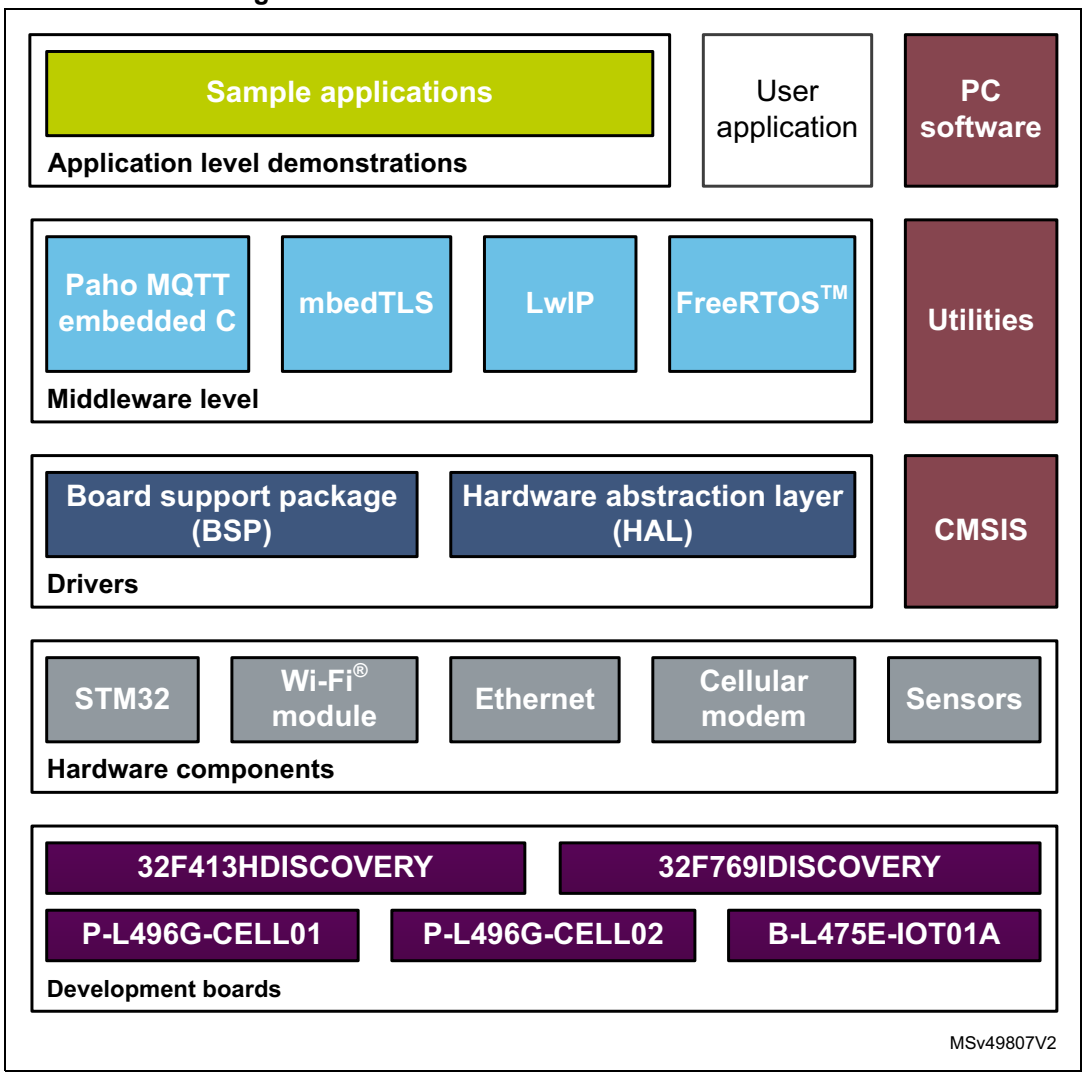
- **STM32Cube HAL**
  The HAL driver layer provides a generic multi-instance simple set of APIs (application programming interfaces) to interact with the upper layers (application, libraries and stacks).

  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.
- **MQTT client middleware**
  It is composed of the Paho MQTT embedded C client library (used as a transport layer by the MQTT applications), and JSON parser.
- **mbedTLS**
  MQTT and HTTPS use a TLS connection, which is managed by the mbedTLS library.
- **TCP/IP**
  The TCP/IP connection can be handled either by the Wi-Fi® module, the cellular module, or the LwIP middleware (when the Ethernet connection is used). In the X-CUBE-CLD-GEN package, only the 32F769IDISCOVERY board can connect via Ethernet.
- **FreeRTOS™**
  It is a real-time operating system required by LwIP for providing a socket-based interface to the user.

Figure 2 outlines X-CUBE-CLD-GEN software architecture.

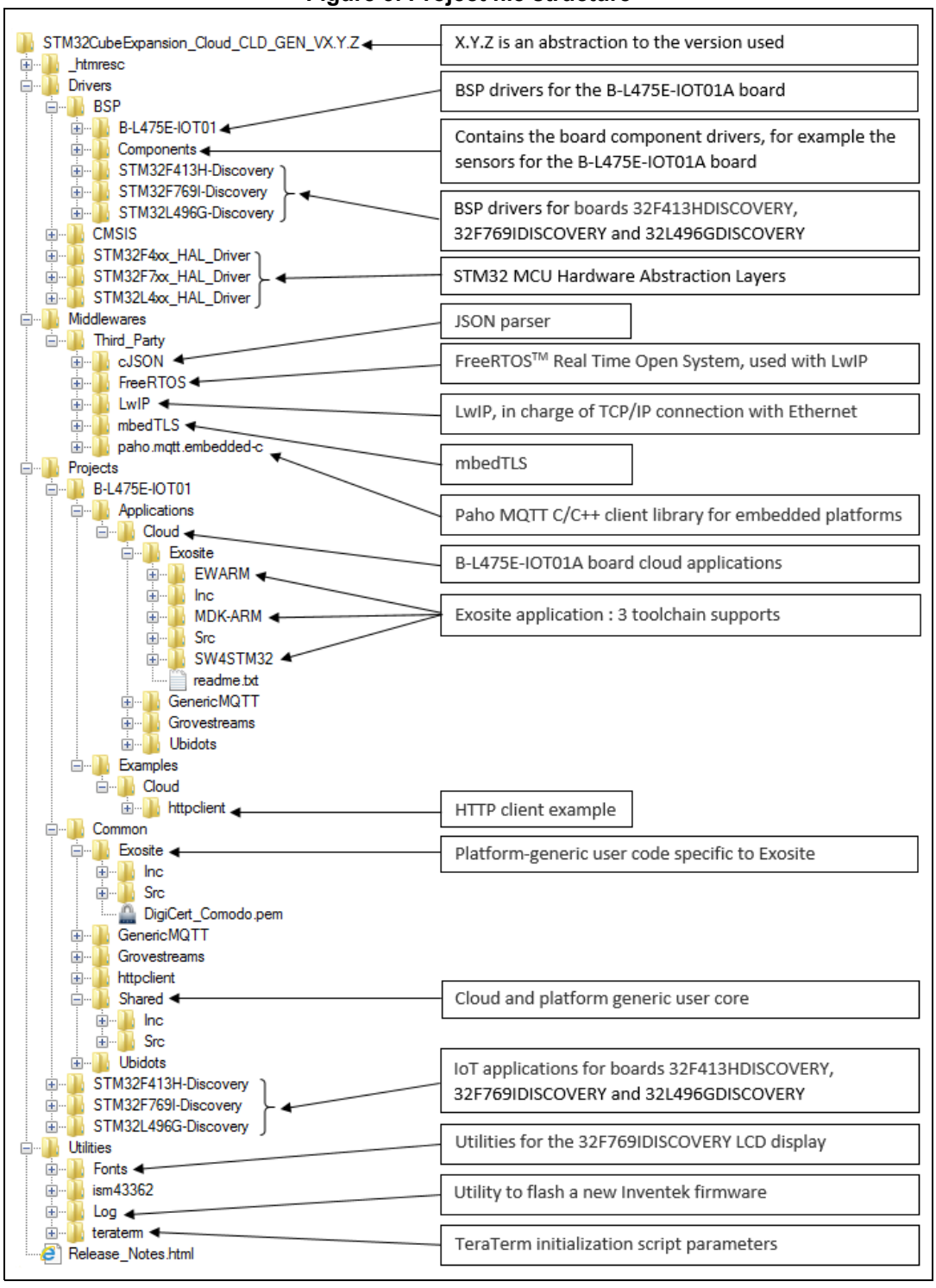
Figure 2. X-CUBE-CLD-GEN software architecture



3.3 Folder structure

Figure 3 presents the folder structure of the X-CUBE-CLD-GEN package.

Figure 3. Project file structure



3.4 B-L475E-IOT01A board sensors

The sensors that are present on the board and used by the sample application are:

- Capacitive digital sensor for relative humidity and temperature (HTS221)
- High-performance 3-axis magnetometer (LIS3MDL)
- 3D accelerometer and 3D gyroscope (LSM6DSL)
- 260-1260 hPa absolute digital output barometer (LPS22HB)
- Proximity sensor (VL53L0X)

3.5 Wi-Fi components

The Wi-Fi® software is split over *Drivers/BSP/Components* for the module specific software and over *Projects/<board>/WiFi* for I/O operations and for the Wi-Fi® module abstraction.

3.6 Cellular components

The *Cellular* software is split over *Drivers/BSP/Components* for the module specific software and *Projects/STM32L496G-Discovery* for I/O operations and for the *Cellular* module abstraction.

The *BSP\Components\ug96* directory contains the driver supporting by default the UG96 module. This software is also able to drive the BG96 module.

3.7 Reset push-button

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

3.8 User push-button

The User push-button (blue) is used:

- To enter a menu that allows configuring the Wi-Fi® or cellular, and security credentials. This can be done from the time the board starts up to until five seconds after.
- When the board has been initialized, the use of this button is application specific. It is documented in the runtime section of the corresponding application.

The application configures and manages the User button via the board support package (BSP) functions.

The BSP functions are in the *Drivers\BSP\<board name>* directory.

When using the BSP button functions with the `BUTTON_USER` value, the application does not take into account the way this button is connected from a hardware standpoint for a given platform. The mapping is handled by the BSP.

3.9 User LED

The configuration of the user LED that is used by the applications is done via the board support package (BSP) functions.

The BSP functions are under the *Drivers\BSP\<board name>* directory.

Using the BSP button functions with the `LED_GREEN` value, the application does not take into account the way the LED is mapped for a given platform. The mapping is handled by the BSP.

3.10 Real-time clock

The STM32 RTC is updated at startup from the www.gandi.net web server. The `HAL_RTC_GetTime()` function provides the time value to the user, for instance to timestamp messages.

The *libC* time function is implemented based on the RTC.

3.11 mbedTLS configuration

The mbedTLS middleware support is fully configurable by means of a `#include` configuration file.

The name of the configuration file can be overridden by means of the `MBEDTLS_CONFIG_FILE` compilation switch.

The X-CUBE-CLD-GEN package uses file *<application name>_mbedtls_config.h* for project configuration.

This is implemented by having the following `#` directives at the beginning of the *mbedTLS.c* and *mbedTLS.h* files:

```
#if !defined(MBEDTLS_CONFIG_FILE)
#include "mbedtls/config.h"
#else
#include MBEDTLS_CONFIG_FILE
#endif
```

The configuration file specifies for instance the ciphers to integrate, the size of the internal TLS buffers, or the platform porting callbacks.

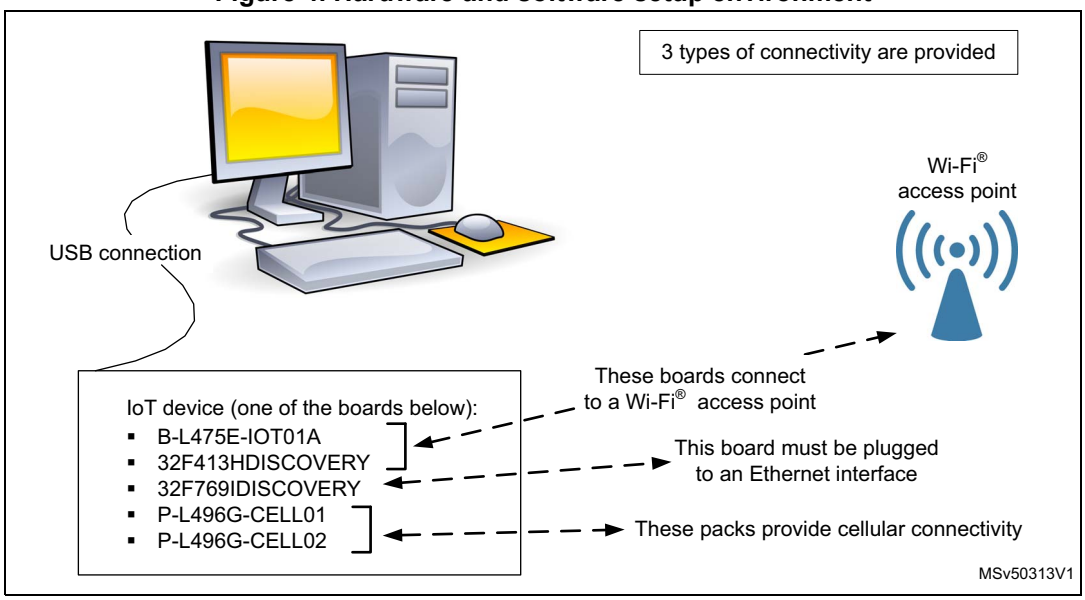
4 Hardware and software environment setup

To set up the hardware and software environment, one of the five supported platforms must be plugged to a personal computer via a USB cable. This connection with the PC allows:

- Flashing the board
- Storing the Wi-Fi® or cellular, and security credentials
- Interacting with the board via a UART console
- Debugging

The B-L475E-IOT01A or 32F413HDISCOVERY boards must be connected to a Wi-Fi® access point while the 32F769IDISCOVERY board must be connected to an Ethernet interface. The P-L496G-CELL01 and P-L496G-CELL02 packs provide cellular connectivity. The various setups are illustrated in *Figure 4*.

Figure 4. Hardware and software setup environment



The prerequisites for running the examples are:

- One of the following connectivity solution:
  - A Wi-Fi® access point, with a transparent Internet connectivity meaning that neither a proxy, nor a firewall are blocking the outgoing traffic. It has to run a DHCP server delivering the IP and DNS configuration to the board.
  - An Ethernet connection with a transparent Internet connectivity meaning that neither a proxy, nor a firewall are blocking the outgoing traffic.
  - A valid cellular subscription, with activated wireless communications data services, either from the embedded SIM integrating the MVNO profile or from an external Micro-SIM.
- A development PC for building the application, programming through ST-LINK, and running the virtual console.
- A computer for running:
  - An MQTT broker (server) that the user can configure. If any, the computer firewall must let the MQTT connections in (typically ports 1883 and 8883).

    Preferably use Eclipse Mosquitto™, for which the package provides example configuration files.

    In order to activate the TLS server authentication, the broker must run on a host that the user board can have access to through its host name. If the host is part of a private LAN, a local DNS server is needed.
  - An MQTT client in order to interact with the device. A Node-RED instance provides dashboards. Otherwise, a plain Eclipse Mosquitto™ MQTT client is simpler to install.
  - This can for instance be the development PC, a virtual private server, or a single-board computer.
- The OpenSSL™ toolset, to build the user root CA, device certificate, and trust chain.

5 Application build and flash

Caution: Before opening the project with any tool chain, make sure that the folder installation path is not too deep since the tool chain may report errors after the build otherwise.

Open and build the project with one of the supported development tool chains (see the release notes for detailed information about the version requirements).

Program the firmware on the STM32 board: copy (or drag and drop) the binary file under *Projects*\<*board name*>*Applications**Cloud*\<*...*>*Binary* to the USB mass storage location created when the STM32 board is plugged to the PC. Alternatively, you can program the STM32 board directly through one of the supported development tool chains.

6 Interacting with the boards

A serial terminal is required to:

- Configure the board
- Display locally the received application IoT cloud-to-device messages

The example in this document is illustrated with the use of Tera Term. Any other similar tool can be used instead.

When the board is used for the first time, it must be programmed with the cloud device IoT identification data. The Cloud device IoT identification data are specific to each IoT cloud provider and detailed in the application related sections of this document:

- Determine the STM32 ST-LINK Virtual COM port used on the PC for the Discovery board. On a Windows® PC, open the *Device Manager*.
- Open a virtual terminal on the PC and connect it to the above virtual COM port.

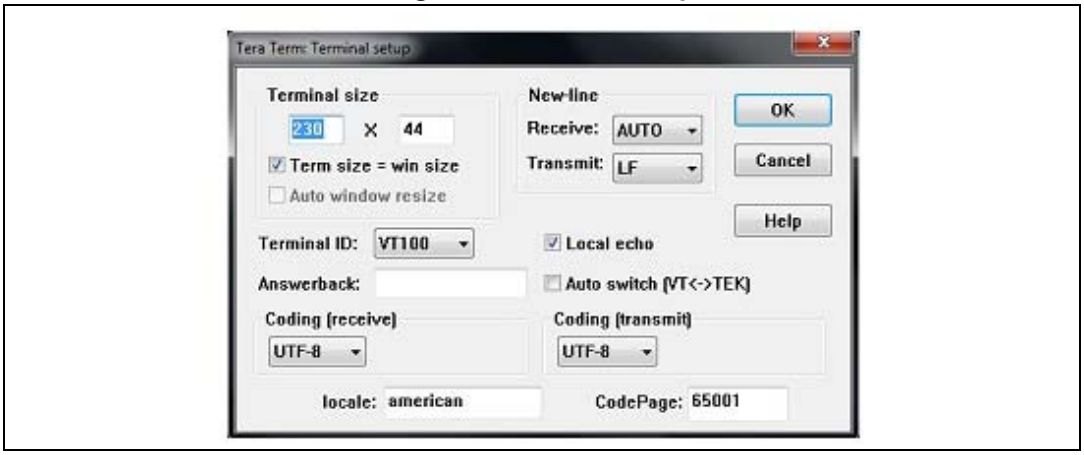
A Tera Term initialization script is provided in the package utility directory (refer to *Figure 3*); this script sets the correct parameters. To use it, open Tera Term, select *Setup* and then *Restore setup*.

The information provided below in this chapter can be used to configure the UART terminal as an alternative to using the Tera Term initialization script.

The terminal setup is illustrated in *Figure 5*, which shows the terminal setup and the *New-line* recommended parameters.

The virtual terminal *New-line* transmit configuration must be set to LineFeed (\n or LF) in order to allow copy-paste from Unix® type text files. The *Local echo* option makes copy-paste visible on the console.

Figure 5. Terminal setup

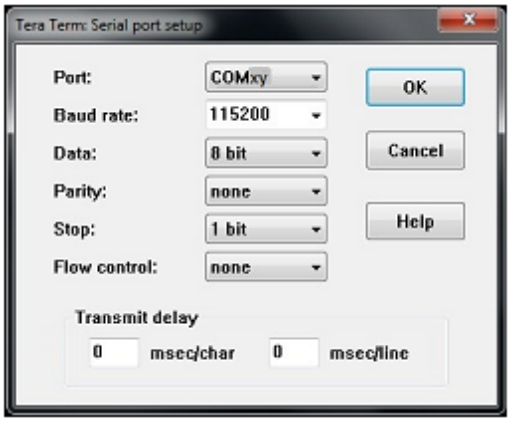


The serial port must be configured with:

- COM port number
- 115200 baud rate
- 8-bit data
- Parity none
- 1 stop bit
- No flow control

The serial port setup is illustrated in *Figure 6*.

Figure 6. Serial port setup



Once the UART terminal and the serial port are set up, press the board reset button (black).

Follow the indications on the UART terminal to upload Wi-Fi® or cellular, and cloud data. Those data remain in Flash and are reused the next time the board boots.

7 B-L475E-IOT01A board

This section describes how the B-L475E-IOT01A board is used in this package.

7.1 Board capabilities

The sensors that are present on the board and used by the sample applications are:

- Capacitive digital sensor for relative humidity and temperature (HTS221)
- High-performance 3-axis magnetometer (LIS3MDL)
- 3D accelerometer and 3D gyroscope (LSM6DSL)
- 260-1260 hPa absolute digital output barometer (LPS22HB)
- Proximity sensor (VL53L0X)

7.2 Inventek module hardware interface

The Inventek module is connected to the MCU and controlled by it as described in *Table 2*.

Table 2. Inventek module hardware interface

Name	Pin	Type	Comment
ISM43362_RST	PE8	Output GPIO, open drain mode	Active low. The Inventek module after power-up or reset raises the CMD/DATA READY pin to signal that the first Data Phase has started. The CMD/DATA READY pin is mapped to the ISM43362_DRDY_EXTI1 STM32 MCU pin.
ISM43362_BOOT0	PB12	Output GPIO, push pull mode	Enable Inventek micro boot loader.
ISM43362_WAKEUP	PB13	Output GPIO, open drain mode	Seen from the Inventek module, the wakeup pin is an external interrupt pin that on the rising edge causes the module to exit stop mode. It is an edge trigged input.
ISM43362_SPI3_CSN	PE0	Output GPIO, open drain mode	The STM32 host must set this output to *Low* to initiate a communication with the Wi-Fi® module.
ISM43362_DRDY_EXTI1	PE1	Input GPIO, interrupt mode when rising	The Inventek module sets this pin to *High* when ready to communicate.
INTERNAL_SPI3_SCK	PC10	Mode SPI3 Alternate Function, no pull	SPI interface to read and write data to the Inventek Wi-Fi® module.
INTERNAL_SPI3_MISO	PC11		
INTERNAL_SPI3_MOSI	PC12		

7.3 Published data

Here is an example of a published sensor message:

```
publication topic: /sensors/Device001        payload: {
 "state": {
  "reported": {
   "temperature": 28.27,
   "humidity": 27.92,
   "pressure": 971.21,
   "proximity": 2040,
   "acc_x": -30, "acc_y": -5, "acc_z": 1034,
   "gyr_x": 910, "gyr_y": -2520, "gyr_z": 1120,
   "mag_x": 96, "mag_y": -45, "mag_z": 527,
   "ts": 1519913214, "mac": "C47F5101186"
  }
 }
}
```

Table 3 presents the units for the values reported by the sensors of the B-L475E-IOT01A board.

Table 3. Units for the values reported by the sensors of the B-L475E-IOT01A board

Data	Unit
Temperature	degree Celsius (°C)
Humidity	relative humidity (%)
Pressure	hectopascal (hPa)
Proximity	millimeter (mm)
Acceleration	milli g-force (mgforce)
Angular velocity	millidegree per second (mdps)
Magnetic induction	milligauss (mG)

7.4 Running the application

The steps to run the application are listed below:

1. Set-up the infrastructure if you administrate your own server and cloud devices as explained in the section related to the application that you use.
2. Make sure that the liner (which is a very thin film placed on the proximity sensor) has been removed, otherwise value 8190 is reported for the proximity measurement.
3. Ensure that JP8 is open, JP5, JP6 and JP7 are closed, JP4 is set to 5V_ST_LINK.
4. Connect a Type-A to Micro-B USB cable from the B-L475E-IOT01A1 or B-L475E-IOT01A2 IoT Discovery board (connector USB ST-LINK CN7) to a PC.
5. LED6 (ST-LINK COM - bi color) must be lit (red) and LED5 (5 V power) must also be lit (LED5 is green).
6. Under the directory *Projects\ B-L475E-IOT01\Applications\Cloud\<Cloud provider>\<tool chain>*, select the project, build and flash the binary with the IDE.

8 32F413HDISCOVERY board

The 32F413HDISCOVERY Discovery kit allows users developing applications with the STM32F4 Series microcontrollers based on the Arm® Cortex®-M4 core. It embeds a STM32F413ZHT6 microcontroller featuring 1.5 Mbytes of Flash memory and 320 Kbytes of SRAM, in an LQFP144 package.

The ISM43362 Inventek Wi-Fi® module 802.11 b/g/n is integrated on this board.

When needed, more information is available to the web page of *www.st.com* dedicated to this board.

9 32F769IDISCOVERY board

The 32F413HDISCOVERY Discovery kit allows users developing and share applications with the STM32F4 Series microcontrollers based on the Arm® Cortex®-M7 core. It embeds a STM32F769NIH6 microcontroller featuring 2 Mbytes of Flash memory and 512 + 16 + 4 Kbytes of SRAM, in a BGA216 package.

The 32F413HDISCOVERY is equipped with an Ethernet connector.

When needed, more information is available to the web page of *www.st.com* dedicated to this board.

10 P-L496G-CELL01 and P-L496G-CELL02 packs

10.1 Board capabilities

The P-L496G-CELL01 and P-L496G-CELL02 packs allow users developing their applications with the STM32L4 Series microcontrollers based on the Arm® Cortex®-M4 core.

Both packs embed a display-less variant of the 32L496GDISCOVERY Discovery board with the low-power STM32L496AGI6 microcontroller, featuring 1 Mbyte of Flash memory and 320 Kbytes of SRAM, in an UFBGA169 package. They also come with an add-on board connected through the STMod+ connector.

The add-on board features:

- A 2G/3G modem (UG96 module, for the P-L496G-CELL01 pack) or an LTE-IoT Cat M1/NB1/2G modem (BG96 module, for the P-L496G-CELL02 pack)
- An embedded ST Incard™ SIM with an MVNO profile, ready to use
- A 32-Kbyte EEPROM containing instructions, which have been preloaded at manufacturing. Among others, the preloaded FW allows the activation of the embedded SIM by using a console application.
- A USB port, which can be used to have access to the Quectel module (FW upgrade, AT commands). It can also be used as an additional power source.

Both modules (UG96 and BG96) are addressed with a set of AT-commands that are for most of them compatible over the 2 packs.

10.2 Module hardware interface

The daughter board is connected to the MCU through an ST-Mod+ connector. The Quectel modem is controlled by means of a series of pins as described in *Table 4*.

Table 4. Quectel module hardware control interface

Name	Pin	Type	Comment
RST	PB2	Output GPIO, push-pull mode	The STM32 host uses these pins to power on or reset the cellular module.
PWR	PD3	Output GPIO, push-pull mode	
TX	PB6	Mode UART1 Alternate Function, pull up	The UART interface is used to read and write data from and to the cellular module.
RX	PG10	Mode UART1 Alternate Function, pull up	

The SIM is selected through two pins as described in *Table 5*.

Table 5. Quectel module SIM selection interface

Name	Pin	Type	Comment
SIM_SEL0	PC2	Output GPIO, push-pull mode	The STM32 host selects the SIM (embedded or external slot). – Assuming SIM_SEL1 is low: – Set SIM_SEL0 high to select the embedded SIM – Set SIM_SEL0 low to select the external SIM
SIM_SEL1	PI3	Output GPIO, push-pull mode	

10.3 Running the application

The steps to run the application are listed below:

1. Connect the Quectel UG96 or BG96 daughterboard to the MCU board IO expander keeping the modem package upside and the SIM card slot downside.

   UG96 is the modem used by default. In case BG96 is used, the user needs to define the `USE_BG96` compilation switch in the project.

2. Activate the UG96 or BG96 embedded SIM card

   The kit comes with a 3-month free-of-charge subscription prepaid with 3 euros. The amount of data it represents depends on the geographical zone; it varies between 15 Mbytes and 0.85 Mbyte.

   The user must enter a voucher on the https://stm32-c2c.com portal to activate the subscription. The voucher is obtained by connecting the board to an hyperterminal as specified in the blister.

   If the board needs to be reinitialized with its original firmware, it is available at https://stm32-c2c.com under *Restore factory firmware*.

   Once the board is registered on the https://stm32-c2c.com portal with its voucher, proceed with the *EMnify* button available on the portal.

3. Retrieve the APN, username and password from your SIM provider.

   In the case of the Embedded SIM integrating the MVNO profile:
   For EMnify: set APN to *em*. No need to set a username/password combination when later prompted for it in the console.

4. Ensure that JP7 is closed (ST_LINK as power source) and that SW1 (blue switch) is *ON*.

5. Connect a Type-A to Micro-B USB cable from the STM32L496AGI6-based Discovery board (connector USB ST-LINK CN5) to a PC.

6. On the MCU side:

   a) LED5 (ST-LINK COM - bi color) must be lit (red). LED8 (5 V power) must also be lit (green).

   b) LED2 (green), then LED1 (red) must be consecutively lit.

7. On the cellular daughter board side:

   – LED1 (red) must be lit (module is powered on), then LED2 (green) must blink (the radio is running).

8. Under directory *Projects\STM32L496G-Discovery\Applications\Cloud\<Cloud provider>\<tool chain>*, select the project, then build and flash the binary with the IDE.

11 MQTT generic application

11.1 Application description

The *GenericMQTTXcubeSample* application implements an MQTT client, which connects to an MQTT broker in order to publish telemetry data and receive parameter updates or commands from the cloud.

The quickest way to get started is to use a clear connection without any authentication to test.mosquitto.org. It hosts a publicly available Eclipse Mosquitto™ MQTT server/broker. In such a case, *Section 11.3: Hosting an own MQTT broker* is not applicable.

When a secure network connection is needed, the use of TLS is mandatory. This implies more dependencies, like downloading the root Certification Authority certificate of the MQTT broker, or even creating the device x509 private key and certificate if a mutual authentication is needed.

11.2 User configuration

The application user configuration is formatted as a "configuration string". It is entered by the user on the serial console and stored in the MCU Flash memory.

It can be updated at startup upon user prompt.

The user configuration string consists of:

1. The server location: host name and port number
2. The network connection security level:
   a) Clear connection
   b) TLS encrypted (insecure mode, for test purpose only)
   c) TLS encrypted, with x509 server authentication
   d) TLS encrypted, with mutual client-server x509 authentication
3. The MQTT Client ID
4. [Optional] The MQTT user name and password

After the configuration string is entered, the application requests the user to paste the needed root CA certificate(s) and possibly the device certificate and key, according to the specified configuration security level. Refer to *Section 11.3: Hosting an own MQTT broker* for more details on TLS provisioning.

The client configuration must match the server configuration.

Table 6 and *Table 7* describe the supported modes, and which server configuration matches a given client configuration. Mosquitto client/server command line parameters are provided. Refer to the *mosquitto_sub* and *mosquitto.conf* man pages for more details.

Table 6. Configurations without MQTT authentication

Configuration	Device configuration string	Mosquitto client command line	Mosquitto server configuration file
Clear connection	HostName=<mqtt_server_hostname>;HostPort=1883;ConnSecurity=0;MQClientId=mySTM32_1;	mosquitto_sub -v -h <mqtt_server_hostname> -t '#'	listener 1883
Encrypted connection without server authentication	HostName=<mqtt_server_hostname>;HostPort=8883;ConnSecurity=1;MQClientId=mySTM32_1;	mosquitto_sub -v -h <mqtt_server_hostname> -t '#' -p 8883 --cafile ca.crt --insecure	listener 8883 cafile ca.crt certfile server.crt keyfile server.key
Encrypted connection with server authentication	HostName=<mqtt_server_hostname>;HostPort=8883;ConnSecurity=2;MQClientId=mySTM32_1;	mosquitto_sub -v -h <mqtt_server_hostname> -t '#' -p 8883 --cafile ca.crt	listener 8883 cafile ca.crt certfile server.crt keyfile server.key
Encrypted connection with mutual authentication	HostName=<mqtt_server_hostname>;HostPort=8883;ConnSecurity=3;MQClientId=mySTM32_1;	mosquitto_sub -v -h <mqtt_server_hostname> -t '#' -p 8883 --cafile ca.crt --cert client.crt --key client.key	listener 8883 cafile ca.crt certfile server.crt keyfile server.key require_certificate true

Table 7. Configurations with MQTT authentication

Configuration	Device configuration string	Mosquitto client command line	Mosquitto server configuration file
Clear connection	HostName=<mqtt_server_hostname>;HostPort=1883;ConnSecurity=0;MQClientId=mySTM32_1;MQUserName=my_user;MQUserPwd=my_pwd;	mosquitto_sub -v -h <mqtt_server_hostname> -t '#' -u my_user -P my_pwd	listener 1883 allow_anonymous false password_file auth.txt
Encrypted connection without server authentication	HostName=<mqtt_server_hostname>;HostPort=8883;ConnSecurity=1;MQClientId=mySTM32_1;MQUserName=my_user;MQUserPwd=my_pwd;	mosquitto_sub -v -h <mqtt_server_hostname> -t '#' -p 8883 --cafile ca.crt –insecure -u my_user -P my_pwd	listener 8883 allow_anonymous false password_file auth.txt cafile ca.crt certfile server.crt keyfile server.key

Table 7. Configurations with MQTT authentication (continued)

Configuration	Device configuration string	Mosquitto client command line	Mosquitto server configuration file

Encrypted connection with server authentication	`HostName=<mqtt_server_hostname>;HostPort=8883;ConnSecurity=2;MQClientId=mySTM32_1;MQUserName=my_user;MQUserPwd=my__pwd;`	`mosquitto_sub -v -h <mqtt_server_hostname> -t '#' -p 8883 --cafile ca.crt -u my_user -P my_pwd`	`listener 8883`
			`allow_anonymous false`
			`password_file auth.txt`
			`cafile ca.crt`
			`certfile server.crt`
			`keyfile server.key`

Encrypted connection with mutual authentication	`HostName=<mqtt_server_hostname>;HostPort=8883;ConnSecurity=3;MQClientId=mySTM32_1;MQUserName=my_user;MQUserPwd=my__pwd;`	`mosquitto_sub -v -h <mqtt_server_hostname> -t '#' -p 8883 --cafile ca.crt --cert client.crt --key client.key -u my_user -P my_pwd`	`listener 8883`
			`allow_anonymous false`
			`password_file auth.txt`
			`cafile ca.crt`
			`certfile server.crt`
			`keyfile server.key`
			`require_certificate true`

Notes:

- To be configured by the user:
  - Replace <mqtt_server_hostname> by the fully qualified name of the host of the MQTT broker.

    It must match the CommonName of the server certificate.

    With security modes 0 and 1 (and only with those), if the server fully qualified name cannot be resolved by the client, the IP address may be used instead.
  - `my_user` and `my_pwd` must match the content of the *auth.txt* password file.
- The *auth.txt* file is generated by the `mosquitto_passwd` command. Refer to the `mosquitto_passwd` man page for details.
- If several clients may connect to the same broker, each must have a different `MQClientId`.
- An Eclipse Mosquitto™ broker can support several configurations simultaneously, provided that:
  - Each resides on a different port
  - All share the same authentication file

11.3 Hosting an own MQTT broker

Hosting an MQTT broker requires:

1. A host, which can be reached by the device over TCP/IP
   - Sufficient privileges are required to install applications and open TCP ports
   - On a Debian® or Ubuntu® host: `> apt-get install mosquitto`
2. Generating keys and certificates, for TLS use

   As shown in *Table 6* and *Table 7*, several key and certificate files must be provisioned on server and client side, depending on the selected network connection security level:
   - *ca.crt* The shared root Certification Authority certificate
   - *server.key* The server private key
   - *server.crt* The server certificate, signed by the root CA
   - *client.key* The client key
   - *client.crt* The client certificate, signed by the root CA

The first three files are needed as soon as TLS is to be enabled. The last two files are needed only if the chosen security level requires mutual authentication.

Those files can be created by the OpenSSL™ utility. OpenSSL™ is available on multiple platforms, including Linux® and Cygwin™. The commands listed in the sequence presented hereafter work with OpenSSL™ 1.0.2:

1. Create a root certification authority, which signs the certificates of the server and device:

   ```
   openssl req -new -x509 -days 1000 -extensions v3_ca -keyout
   ca.key -out ca.crt
   ```

   This produces files *ca.key* and *ca.crt*.
2. Create the server files (for Eclipse Mosquitto™):

   a) Create a private key for the server:

   For an ECDSA key:

   ```
   openssl ecparam -name secp384r1 -out server.key -genkey
   ```

   For an RSA key:

   ```
   openssl genrsa -out server.key 2048
   ```

   This produces file *server.key*.

   b) Create a certificate signature request from the server key:

   ```
   openssl req -out server.csr -key server.key -new
   ```

   This produces file *server.csr*.

Caution: It is important that the Common Name (CN) that is set in the certificate request matches the host name of the host where Eclipse Mosquitto™ runs. Otherwise, the server certificate verification fails and the MQTT client does not connect. For instance, if Eclipse Mosquitto™ runs at myiothost.st.com, CN must be set to myiothost.st.com, or *.st.com.

Alternatively, if the name of the server cannot be resolved through DNS by the device, the host verification feature must be disabled by setting option *ConnSecurity* to 0 or 1 in the connection string when prompted on the console.

   c) Create the server certificate, signed by the root certification authority:

   ```
   openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key  -
   CAcreateserial -out server.crt -days 1000
   ```

   This produces files *server.crt* and *ca.srl*.

3. Create the client files (for the device)
   a) Create a private key for the client:
      `openssl genrsa -out client.key 2048`
      This produces file *client.key*.
   b) Create a certificate signature request from the client key:
      `openssl req -out client.csr -key client.key -new`
      This produces file *client.csr*.
   c) Create the client certificate for the device, signed by the root certification authority:
      `openssl x509 -req -in client.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out client.crt -days 1000`
      This produces files *client.crt* and *ca.srl*.

At this point, on the server side, the server credentials must be passed to Eclipse Mosquitto™ through configuration file *mosquitto.conf* (absolute paths are recommended):

- `listener 8883`
- `cafile /home/john/ca.crt`
- `certfile /home/john/server.crt`
- `keyfile /home/john/server.key`

The server is launched on the host with `mosquitto -v -c mosquitto.conf`.

Note: In order to check that the server is reachable from the network, try to join it through telnet from a different host: `telnet <hostname> <port number>`.
If the host can be resolved and the port is not blocked by a firewall, the server displays "New connection from xxx on port yyy" on the standard error output.

Finally, on the device side, set the security credentials when prompted on the console at application startup, by pasting the contents of *ca.crt* (concatenated with the other root CA certificates required by the application; this is described in *Section 11.4* at step *8*), *client.crt* and *client.key*.

11.4 Application first launch

The first launch of the application is described by the following sequence from step *1* to step *9*. Steps *1* and *2* are skipped when connecting to test.mosquitto.org:

1. Prepare the TLS credentials matching the desired user configuration as detailed in *Section 11.2: User configuration* and *Section 11.3: Hosting an own MQTT broker*.
2. Configure and start the Eclipse Mosquitto™ MQTT broker.
3. Choose a DeviceID for the MQTT client to be used as:
   – ClientID in the device configuration string
   – DeviceID to build the topic names in case subscription or sending commands manually to the device is needed
   – DeviceID in the Node-RED sample dashboard
4. Connect the board to the development PC through USB (ST-LINK USB port). For more details, refer to *Chapter 6: Interacting with the boards on page 18*.
5. Open the console through a serial terminal emulator such as Tera Term, select the ST-LINK COM port of the board, and configure it with:
   – 8N1, 115200 bauds, no HW flow control
   – Line endings set to LF or CR-LF (Transmit) and LF (Receive)
6. For the Wi-Fi®-enabled boards, enter the Wi-Fi® SSID, encryption mode and password via the console.

   For the board with cellular connectivity, enter the C2C network configuration (SIM operator access point code, user name and password).

   Examples:

   - with EMnify SIM: access point: "em", username: "", password: ""
   - with Eseye SIM: access point: "ESEYE1", username: "", password: ""
7. Set the device configuration string, without the enclosing quotes ("), or leading/trailing spaces. Apply the connection string template printed in the console.

   When connecting to test.mosquitto.org, the configuration string is:

   ```
   HostName=test.mosquitto.org;HostPort=1883;ConnSecurity=0;MQClientId=<DeviceID>;
   ```

   Make sure that `DeviceID` is set and unique.
8. Set the TLS root CA certificates, which the device uses to authenticate the remote hosts through TLS.

   The application requires that a concatenation of 2 CA certificates is provided:

   a) For the HTTPS server, which is used to retrieve the current time and date at boot time (the *Comodo* certificate).

      It is located in *Projects/Common/GenericMQTT/Comodo.crt*

   b) For the MQTT server, which can also be connect through TLS.

      Depending on the broker chosen, it can be the root CA certificate that has been created for the MQTT broker, or the server CA provided by the cloud provider such as Litmus Loop, Ubidots, or test.mosquitto.org.

   The concatenated string must end with an empty line.

   If test.mosquitto.org is used in an unencrypted way, as set by the connection string at step 7, only the *Comodo* certificate is needed.
9. After the parameters are configured, it is possible to change them by restarting the board and pushing the User button (blue button) when prompted.

11.5 Application runtime

1. The application makes an HTTPS request to retrieve the current time and date, and configure the RTC. Once OK, the RTC is updated.

   Note: HTTPS has the advantage over NTP that the server can be authenticated by the board, preventing a possible man-in-the-middle attack. However, the first time the board is switched on (and each time it is powered down and up, if the RTC is not backed up), the verification of the server certificate fails as its validity period does not match the RTC value. The following log is then printed on the console:

   `x509_verify_cert() returned -9984 (-0x2700)`

   If the `CLOUD_TIMEDATE_TLS_VERIFICATION_IGNORE` *switch is defined in "cloud.c", which is the case by default, this error is ignored and the RTC is updated from the "Date:" field of the HTTP response header. Otherwise, a few more error log lines are printed, and the application tries to connect again without verifying the server certificate.*

2. The application connects to the MQTT broker by:
   a) Sending the status of the device (LED status and `TelemetryInterval`) to the status topic (*/devices/<DeviceID>/status* in the generic case)
   b) Staying idle, pending on local user, or hub-initiated events

The possible local user actions are:

- Single push on the User button:

  Trigs a sampling of the sensor values and their publication to the broker through a publish MQTT message on the telemetry topic (*/sensors/<DeviceID>* in the generic case).

- Double push on the User button:

  Starts or stops the publication loop of the sensor values. When the loop is running, the sensor values are published every `TelemetryInterval` seconds. Each sensor values publication is signaled by the user LED blinking quickly for half a second.

  Issues possible remotely-initiated events by publishing specific payloads to the topic the device has registered to, from another MQTT client (this topic is displayed on the console */devices/<DeviceID/control* in the generic case):

  – LED control command:

    Publish the payload `{"LedOn": true}` or `{"LedOn": false}`

  – Change of the `TelemetryInterval` telemetry publication interval:

    Publish the payload `{"TelemetryInterval": <number of seconds>}`

  – Reboot command:

    Publish the payload `{"Reboot": true}`

  Upon update, the LED status and the `TelemetryInterval` value are published to the device status topic.

If the MQTT broker is part of a managed cloud service such as Litmus Loop, or Ubidots, the name of the MQTT topics and the format of the MQTT messages must generally be customized in the application code. There are already such customizations under compilation switches in the sample application.

12 Node-RED as a dashboard for the MQTT generic application

The Node-RED *node.js* application is a convenient tool to build cross-protocol network use cases based on graphical data flow descriptions and Javascript plug-ins.

For instance, the Node-RED application can host MQTT clients, which run alongside an MQTT broker, in order to build a use case involving several MCU boards and a graphical web user interface.

Node-RED is a graphical alternative to the Eclipse Mosquitto™ command line pub/sub client.

For new users, it is recommended to start with the mosquitto CLI, and setup a Node-RED flow only once advanced visualization is needed.

Node-RED is most useful on the devices that embed various sensors to be monitored, thanks to its dashboard extension.

This chapter shows how to create a graphical dashboard for the STM32L4-Series-based B-L475E-IOT01A IoT board and its embedded sensors.

12.1 Installation

Node-RED can be installed on Linux®, Windows® or macOS®. The installation example shows Linux® command lines:

1. Install a recent node package from *nodejs.org* and get the node command in the `PATH`
2. Install Node-RED, for instance in a local *X-CUBE-node-red* directory:
   - `mkdir X-CUBE-node-red && cd X-CUBE-node-red`
   - `npm install --unsafe-perm node-red`
3. Install the node-red-dashboard extension:
   - `npm install node-red-dashboard`
4. Launch the Node-RED platform:
   - `node node-modules/node-red/red.js`
5. Connect to the platform with a web browser at http://127.0.0.1:1880

Note: If the Node-RED server is installed on a headless host, remote connection to the platform is possible with the host name or IP address. If needed, port 1880 must be opened in the host firewall.

Caution: By default, access to the Node-RED server is public. For setting Node-RED access rights, and to use HTTPS for the connection, refer to the security section of the Node-RED user guide.

12.2 Flow configuration

Import *Projects/Common/GenericMQTT/flow.nodered* through a copy-paste to the clipboard as shown in *Figure 7* and *Figure 8*.

Figure 7. Node-RED dashboard: flow import (1/2)

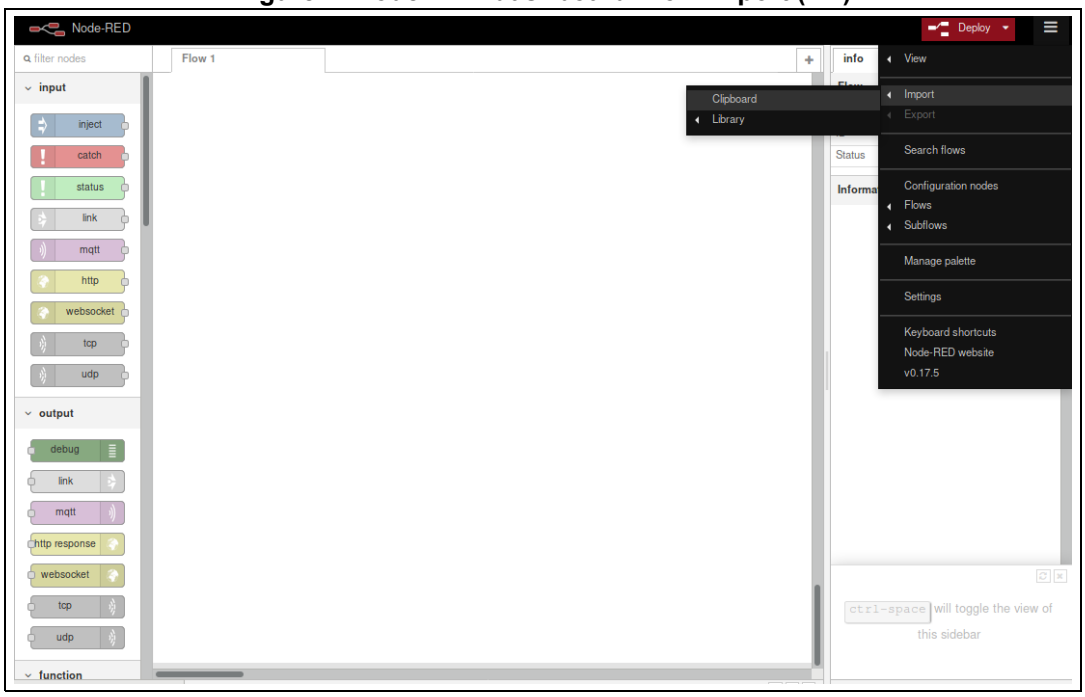
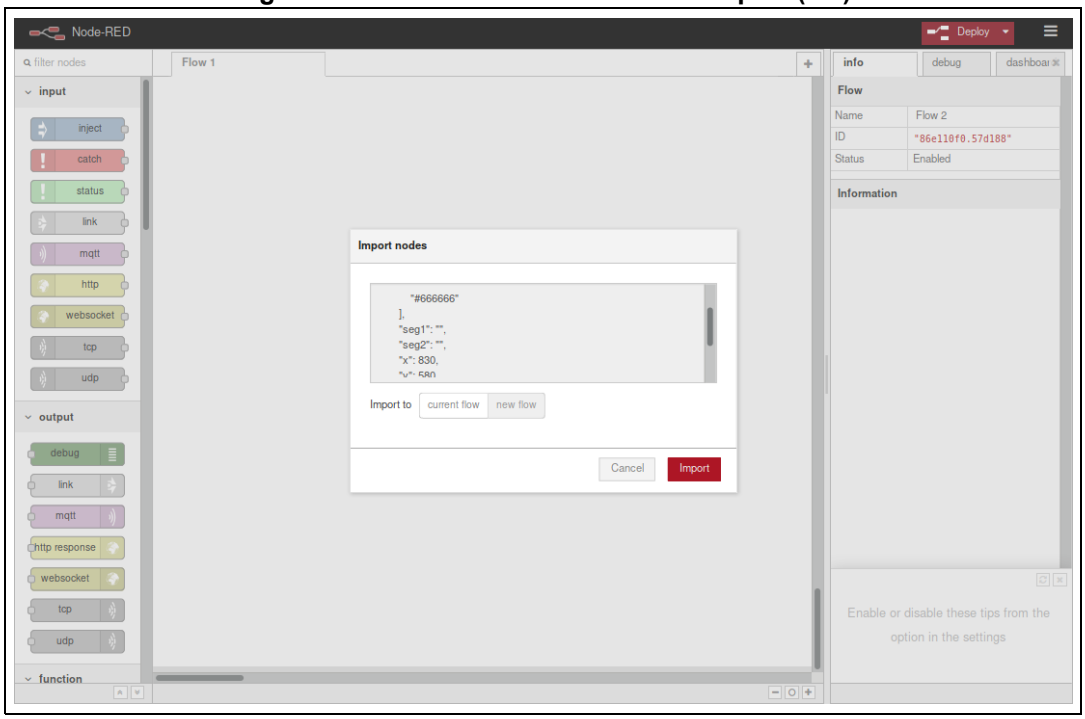


Figure 8. Node-RED dashboard: flow import (2/2)



As a result, the MQTT flow is displayed as illustrated in *Figure 9* and *Figure 10*.

Figure 9. Node-RED dashboard flow anatomy: network connectors

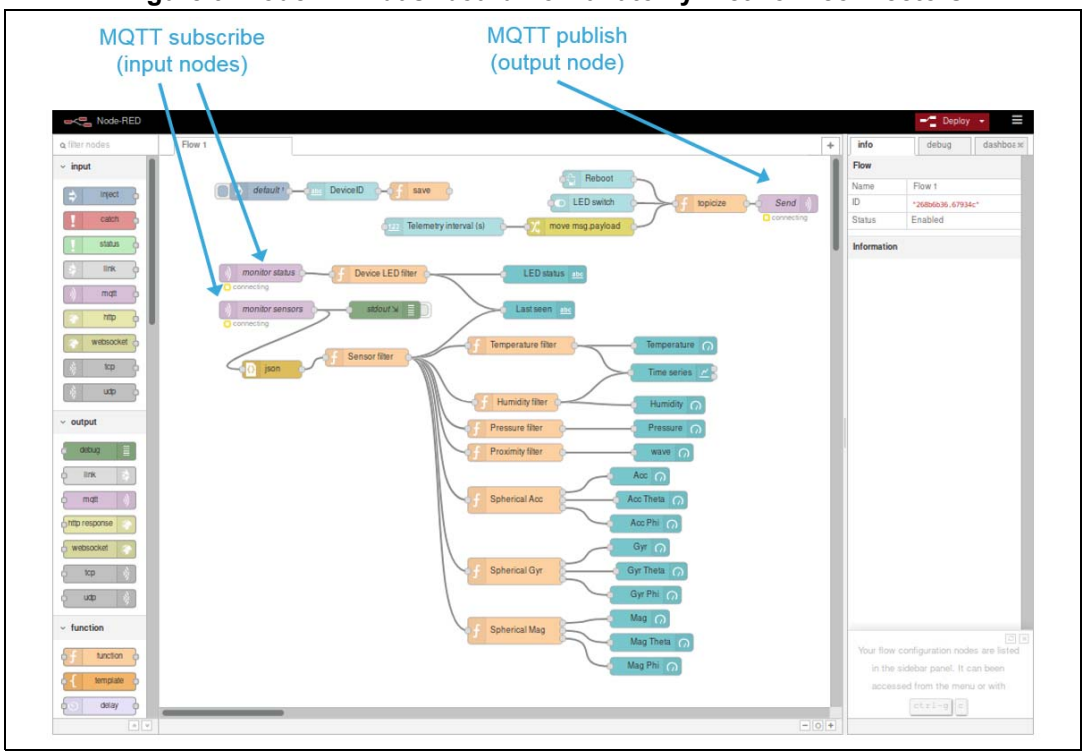
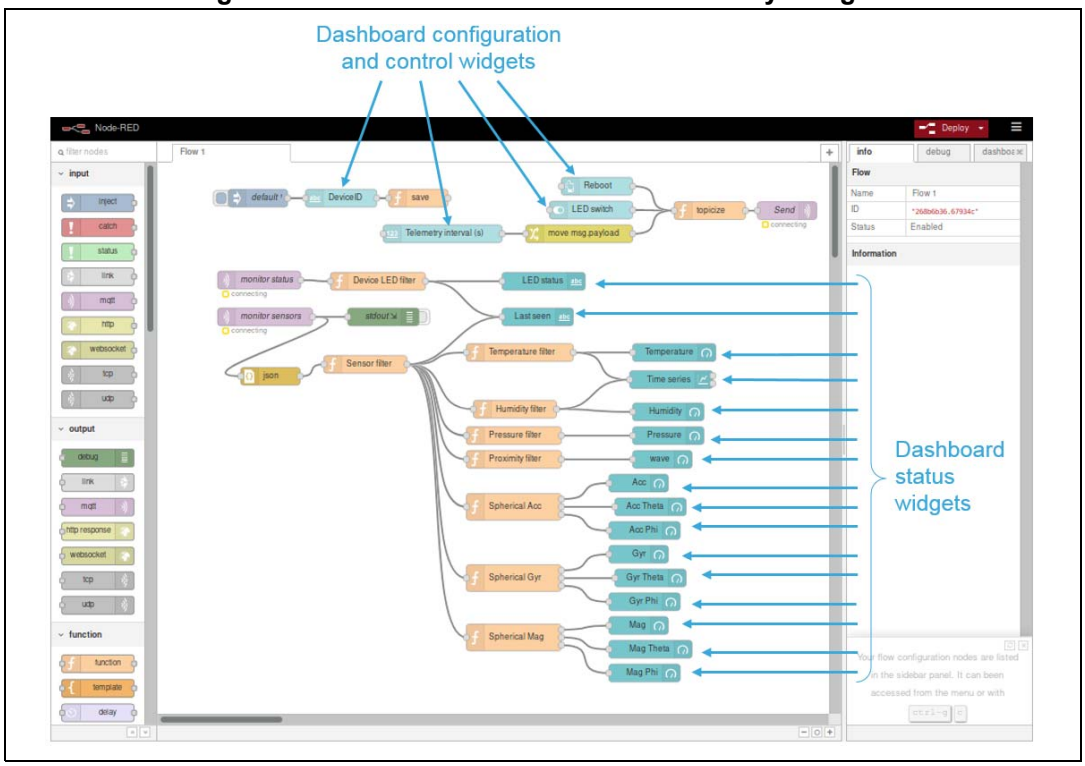


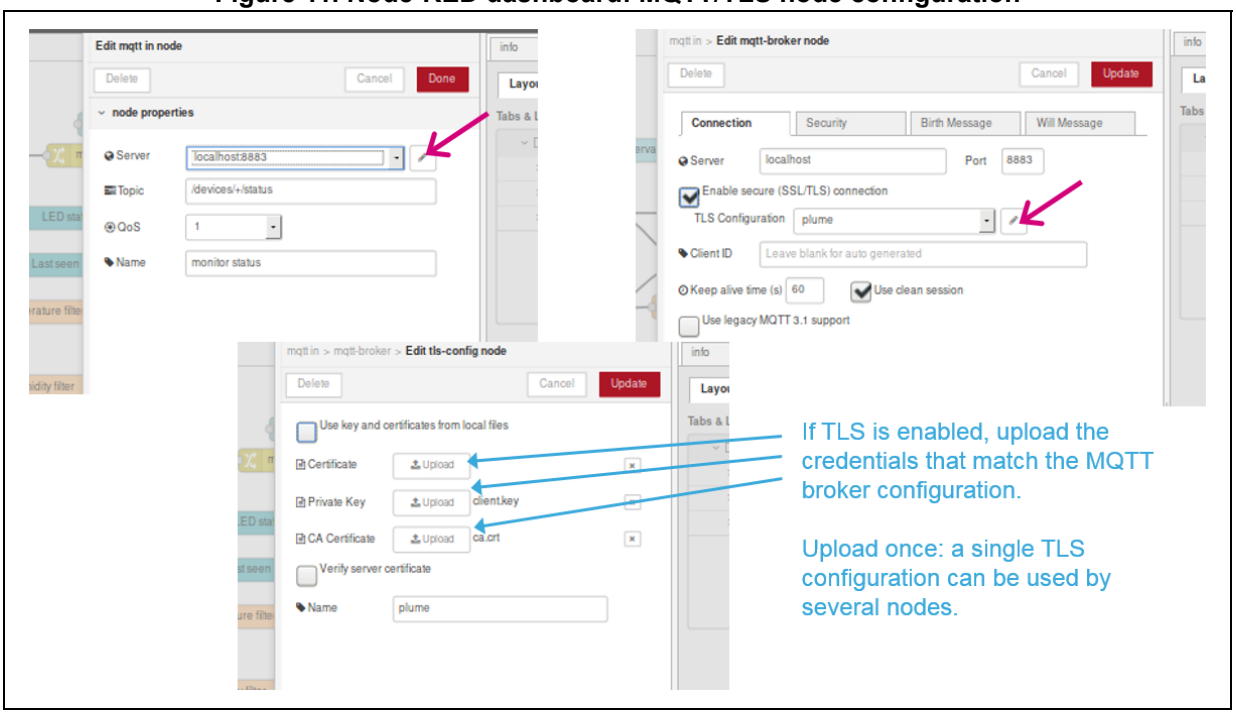
Figure 10. Node-RED dashboard flow anatomy: widgets



12.3 Flow customization

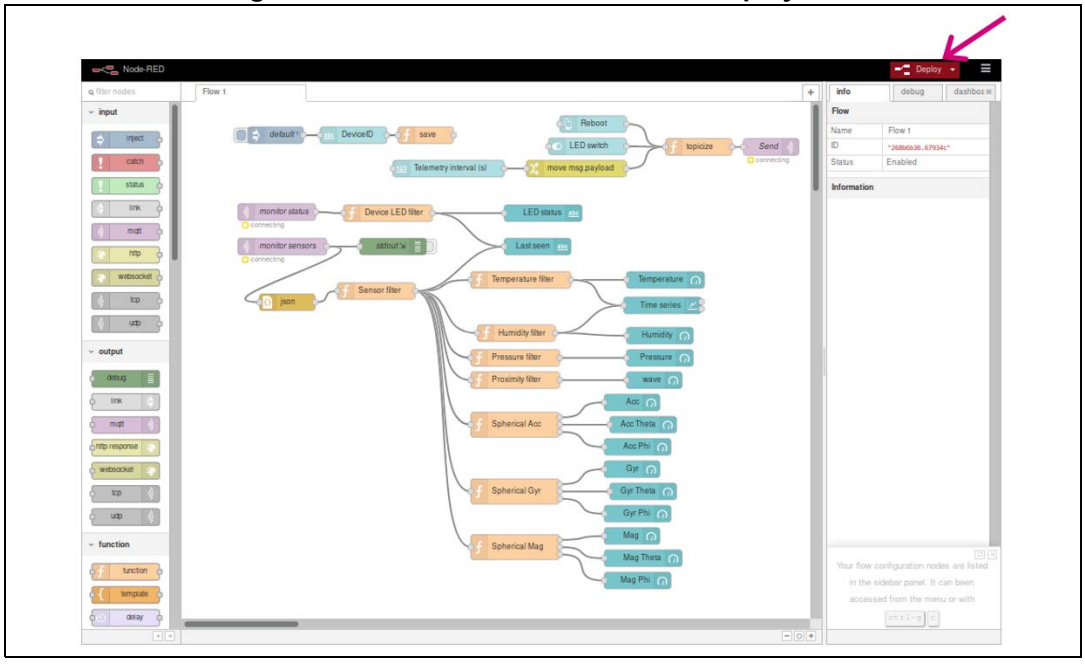
Double-click and configure each MQTT node. Both clear and TLS connections are supported as illustrated in *Figure 11*.

Figure 11. Node-RED dashboard: MQTT/TLS node configuration



After the confirmation of all the changes by clicking *Update / Done* on the configuration blade, click on the *Deploy* button as shown in *Figure 12*.

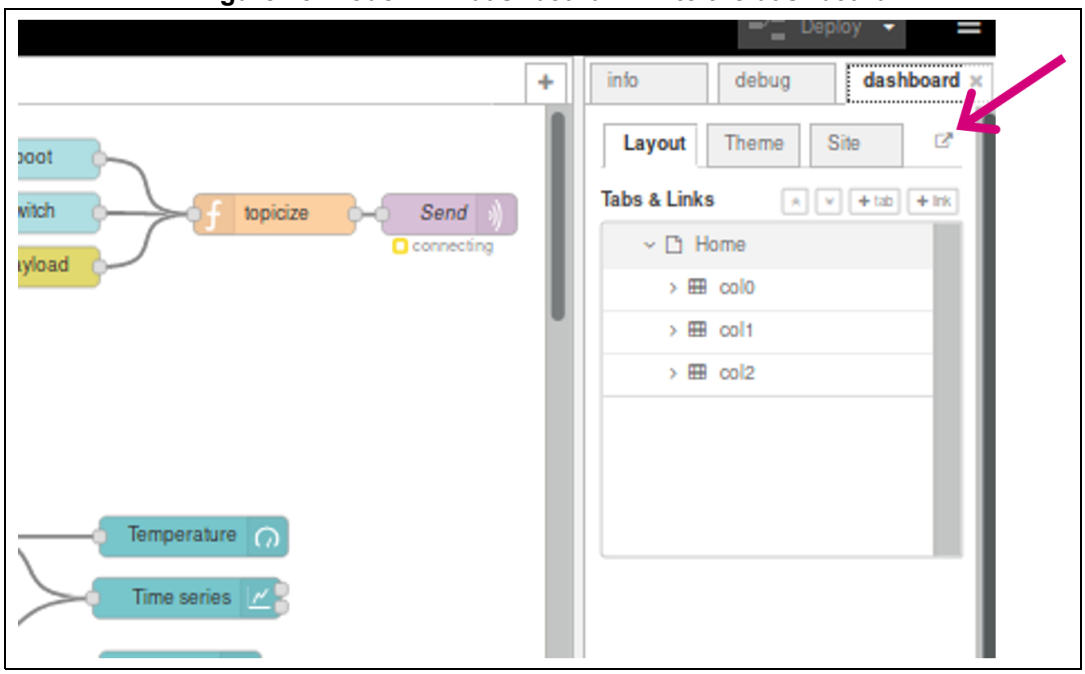
Figure 12. Node-RED dashboard: flow deployment



12.4 Dashboard display

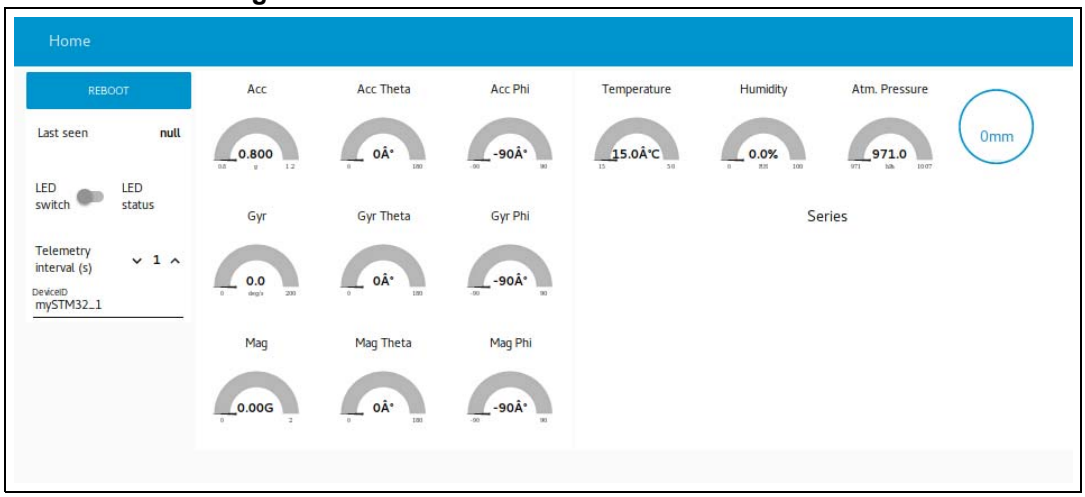
Go to http://127.0.0.1:1880/ui, or click on the shortcut in the *dashboard* tab of the flow web page as shown in *Figure 13*.

Figure 13. Node-RED dashboard: link to the dashboard



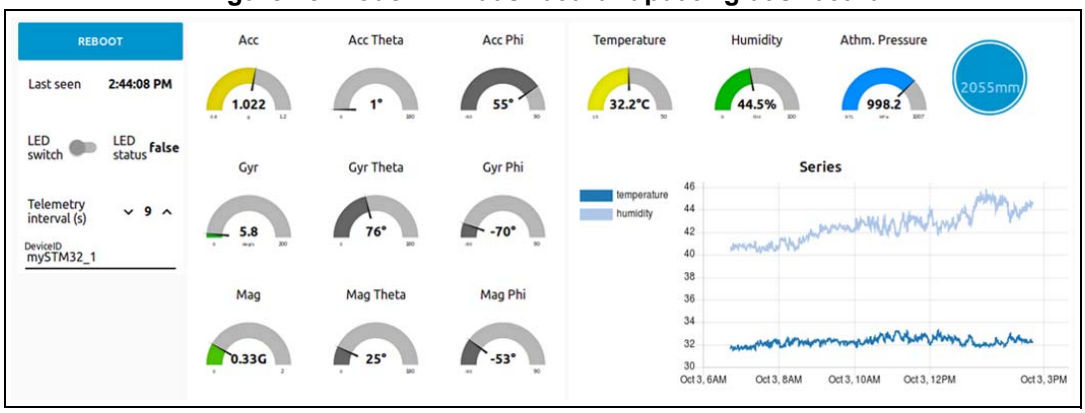
The IoT board dashboard is displayed as in *Figure 14*. There is no data at this stage.

Figure 14. Node-RED dashboard: clear dashboard



Set the *DeviceID* field according to the device configuration and press *ENTER* to validate the update. Start the *GenericMQTT* application on the device, activate the publication loop by a double-push on the User button, and see the live update of the widgets as illustrated in *Figure 15*.

Figure 15. Node-RED dashboard: updating dashboard



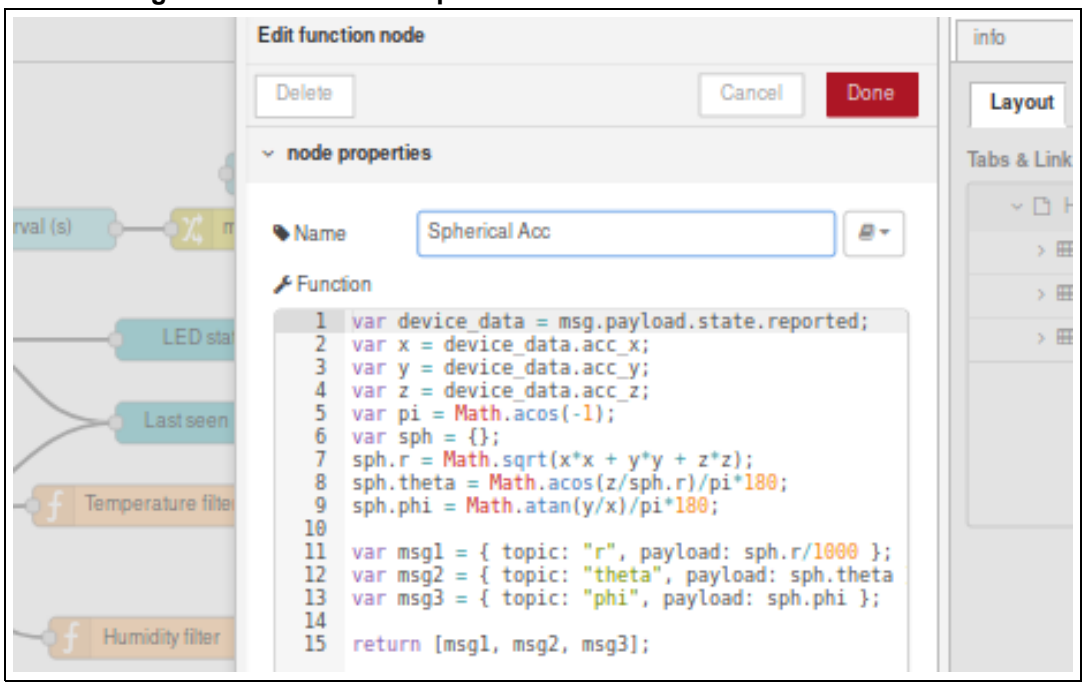
It is then possible to control the board by:

- Toggling the LED switch to drive the LED status
- Changing the telemetry interval
- Sending a reboot command
- Switching to the monitoring of another board by changing the *DeviceID* field

12.5 Under the hood

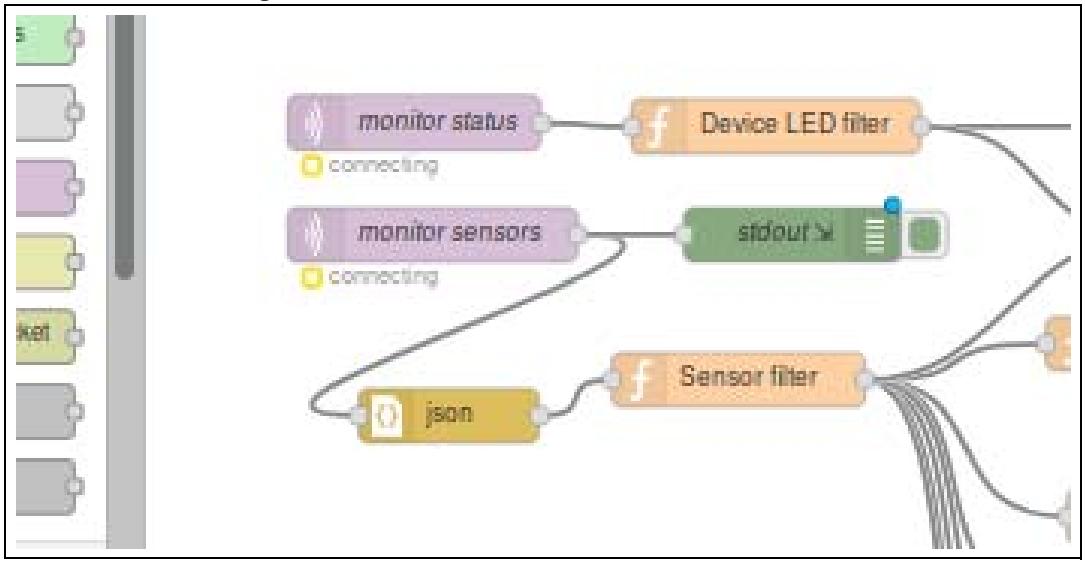
The 3-axis measurements are displayed in spherical coordinates. They are translated from the Cartesian coordinates by simple function nodes. *Figure 16* shows how this is done in the *Spherical Acc* node.

Figure 16. Cartesian to spherical coordinates conversion function



A trace of the */sensors/#* topic is activated by clicking on the right of the *stdout* debug node and further clicking on the flow button as shown in *Figure 17*.

Figure 17. Node-RED dashboard: trace activation



The trace is displayed on the Node-RED server standard output, and on the debug tab of the flow. It is also possible to connect any node output to the *stdout* node, and get any data traced in the debug tab as presented in *Figure 18*.

Figure 18. Node-RED dashboard: debug tab

12.6 Network architecture examples

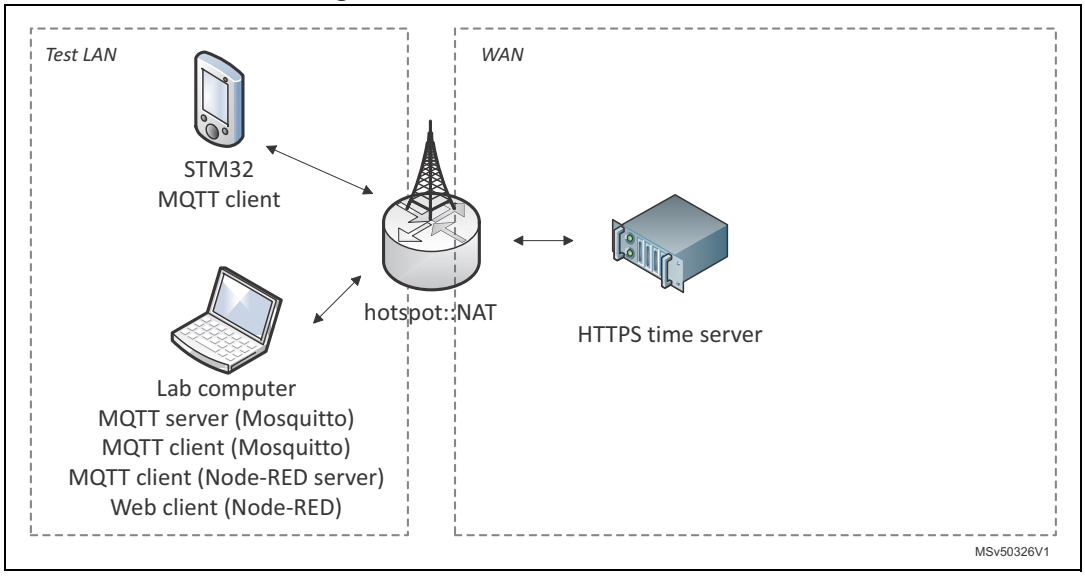
The security rules of the local network infrastructure can impact the location of the MQTT and HTTP applications that implement the test case.

This section presents various possible network architecture mappings.

The user only manages the test LAN, a computer on this LAN, and the servers.

Figure 19 shows an example of a related network architecture.

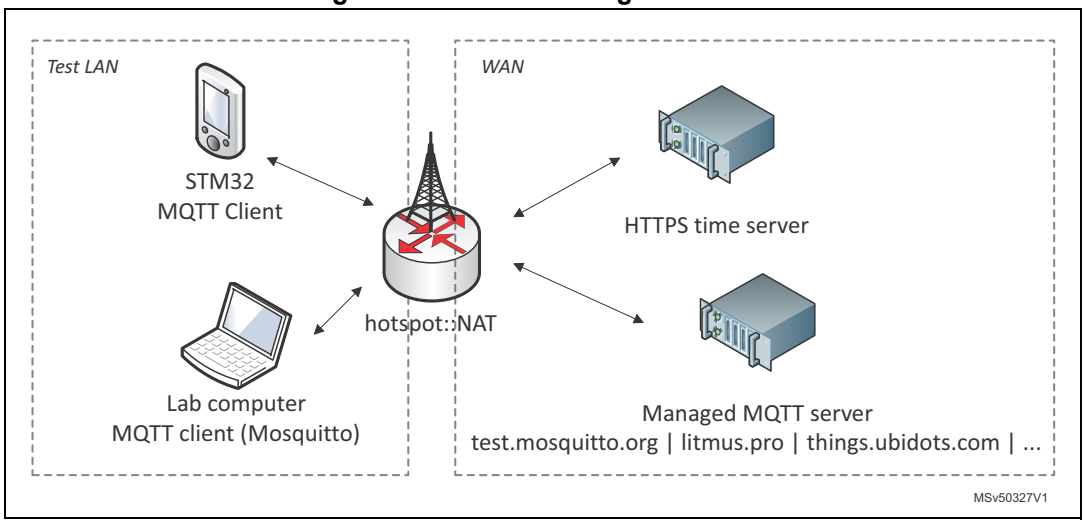
Figure 19. Local self-hosted services



The user only manages the test LAN and a computer on this LAN.

In this case, the user is not willing to manage the servers. *Figure 20* shows an example of a related network architecture.

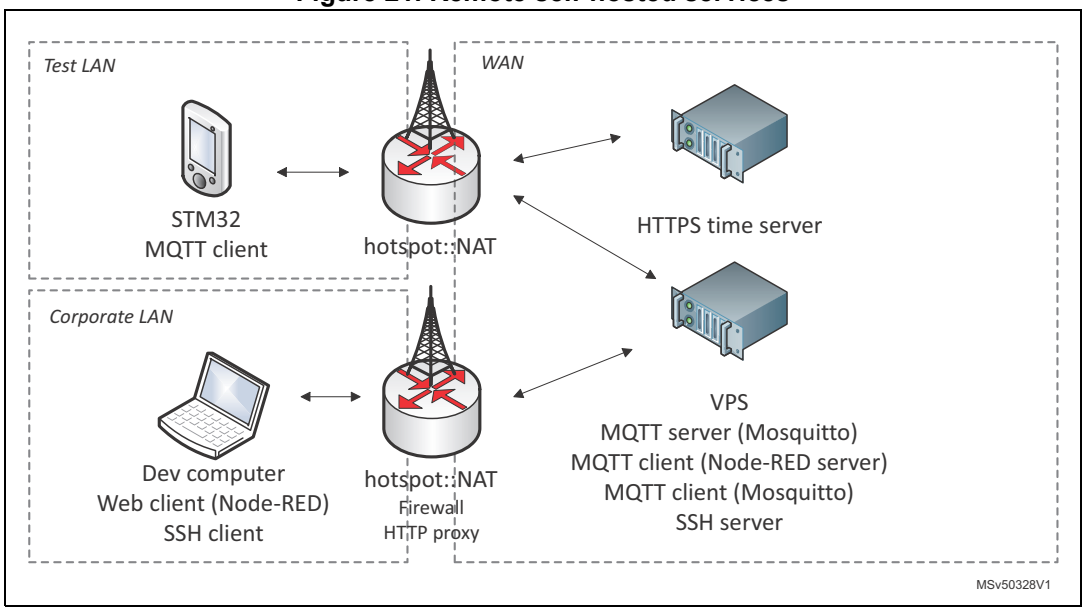
Figure 20. Remote managed services



The user only manages the test LAN.

In this case, the user does not manage a computer on the LAN. A Virtual Private Server must be leased. *Figure 21* shows an example of a related network architecture.

Figure 21. Remote self-hosted services



12.7 References

Useful Node-RED information for the use as an MQTT generic application dashboard are available at:

- https://nodered.org/
- http://noderedguide.com/

13 Using the EMnify VPN

EMnify customers can create their own virtual private network for their mobile IoT/M2M devices fitted with EMnify SIMs. Data traffic is exchanged between the devices and the application server through an OpenVPN® tunnel, enabling direct communication with the IPs of the mobile devices (no NAT applied).

The tunnel is established between the EMnify core network and the customer VPN gateway or server.

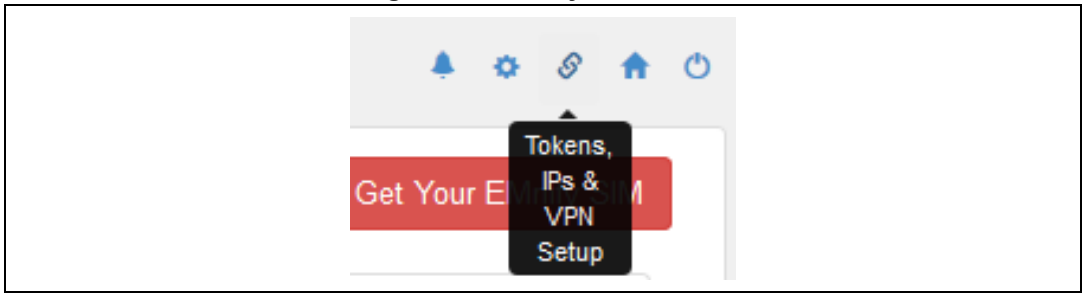
This chapter describes how a user can connect an EMnify endpoint to a lab computer[(a)] using a VPN connection. The lab computer can act as an application server or a relay that redirects messages to any IoT cloud server.

EMnify allocates one private address space to each *organization*, for instance 10.193.184.0/22.

- All the cellular endpoints with SIM cards associated to the *organization* belong to the subnet
- Each *organization* is provided with one OpenVPN® client configuration file

A *tun0* virtual network interface is created in the 10.0.0.0/8 subnet on the lab PC. Refer to the installation details on the top-right corner of the EMnify account start page as illustrated in *Figure 22*.

Figure 22. EMnify VPN menu



The *Technical Resources* section of the EMnify web page provides additional documentation.

Two topologies are possible:

- The application server runs on the host of the OpenVPN® client
- The application server runs on another host than the OpenVPN® client

The application server runs on the host of the OpenVPN® client

In such a case, an HTTP or MQTT server runs on the lab PC (inside the test LAN), which is now part of the VPN.

The traffic reaching the application server can be analyzed by running a network protocol analyzer on the lab PC.

a. In this chapter, the lab computer or lab PC is the computer used by the user for experimenting with the VPN.

The application server runs on another host than the OpenVPN® client

In such a case, the application server runs on a host located in the public Internet.

Thanks to a port redirection, the lab PC routes and possibly captures the traffic between the client and the server.

For using the lab PC as a relay to the Ubidots MQTT server, the following function is an example that redirects the network packets:

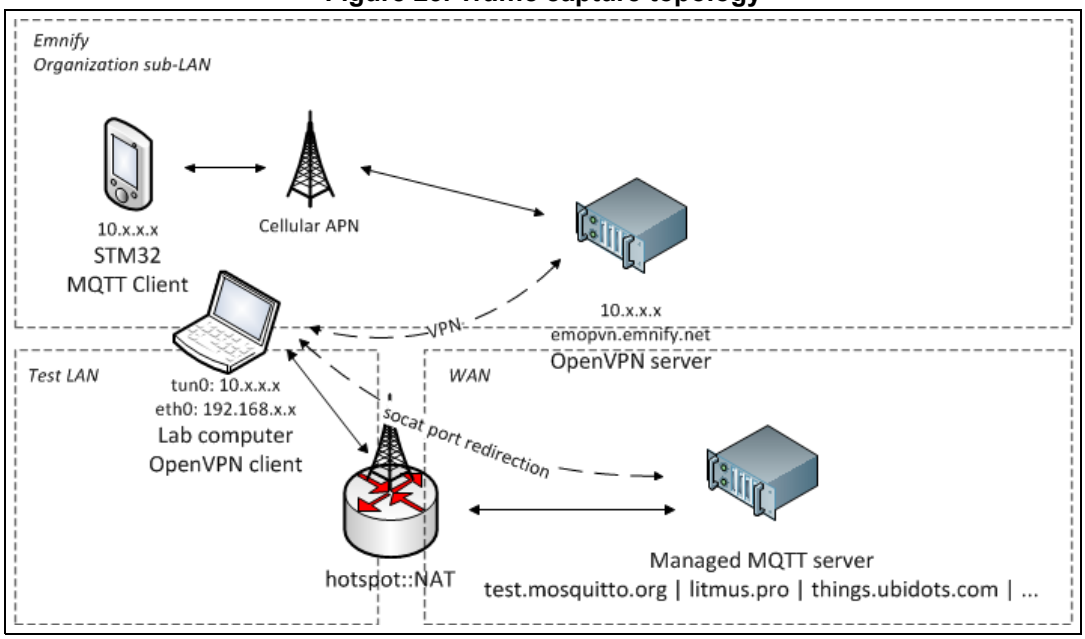
```
$ socat tcp4-listen:8000,reuseaddr,fork tcp:things.ubidots.com:8883
```

The MQTT client must point to `<IP of the tun0 interface>:8000`.

The lab PC is not resolved by the DNS of the cellular device, meaning that the TLS server authentication is not possible. As a result, the server certificate verification by mbedTLS must be skipped by setting the `tls_server_noverification` socket option.

Wireshark® can capture the traffic that *socat* redirects to the remote application server as illustrated in *Figure 23*.

Figure 23. Traffic capture topology



14 MQTT Ubidots application

14.1 Application description

The MQTT Ubidots application is derived from the Generic MQTT application with a few changes to allow the connection to the Ubidots cloud MQTT server.

It connects to Ubidots IoT cloud through the MQTT protocol with the credentials provided by the user. When the User button is pushed, it sends a LED toggle command to the IoT Cloud endpoint, which returns the message to the board and triggers the LED toggle.

The C pre-processor `define UBIDOTS_MQTT` must be defined before compiling the Generic MQTT application. This changes some topic strings and payload content format when posting data to the Ubidots MQTT server.

In addition, the application configuration string is specific to Ubidots MQTT (refer to *Section 14.4*).

14.2 Account creation

The account for MQTT Ubidots is the same as for HTTPS Ubidots:

- If an account has already been created for HTTPS Ubidots, use it for MQTT Ubidots. Refer to *Chapter 18: HTTPS Ubidots application* for details.
- If no account exists already, connect with a web browser to the *Ubidots for Education* web site and create an account by providing an email and a password.

Make sure to avoid a possible confusion with Ubidots for industry web site:

- Ubidots offers different services:
  - *Ubidots*, which is for industry and business
  - Ubidots for Education
- The application works with the *Ubidots for Education* service
- To get access to the *Ubidots for Education* web site, open a web browser on the Ubidots site, and select *Industries / Education*.

Once the account is created, login to the *Ubidots for Education* web site.

14.3 Device creation

For the device creation, apply the following:

1. In the top-right menu, click on *Devices* and add a device using the + button.
2. Give the device a name, for instance *iotboard*.
3. Click on *API Label* and enter a label, for instance *1234*; Take note of the device API label.

For the security token, apply the following:

1. On the top-right of the screen, click on the account name to open the account menu.
2. In the account menu, click on *API credentials;* Take note of the *Default token* on the right.
3. Copy-paste the default token in a text file. Do not copy the *API key* on the left.

14.4 Application first launch

1. Compile the application with `#define UBIDOTS_MQTT` activated: uncomment it in file *GenericMQTTXcubeSample.c.*
2. Flash the board with the compiled application and start it. Connect to the USB COM port of the board with a serial terminal.
3. When asked, enter the device security parameters or credentials:
   - Enter the GenericMQTT configuration string of the MQTT generic application presented in *Chapter 11 on page 27*, making sure that `device_API_label` and `Default_Token` are replaced with the appropriate values):

     ```
     HostName=things.ubidots.com;HostPort=8883;ConnSecurity=2;MQC
     lientId=device_API_label;MQUserName=Default_Token;MQUserPwd=
     ```

   - Enter the TLS security certificate available in *Projects\Common\Ubidots\ DSTRootCAX3_comodo.pem*.

     The certificate is the same as for the Ubidots HTTPs application presented in *Chapter 18 on page 56*.

14.5 Application runtime

By default, the Ubidots application automatically tries to connect to the Ubidots cloud through the MQTT protocol by using the parameters set in *Section 14.4*.

Once connected, a single push on the User button (blue button) triggers a publish of the sensor values and timestamp to the cloud.

A double push on the User button starts the publication loop mode. Values are sent at regular intervals. Double push again the User button to exit from the publication loop mode.

14.6 Dashboard use

On the Ubidots web site, in a device window, it is possible to see the graphs of the variables when they change over time. It is also possible to create a dashboard by clicking on *Dashboards* top-right in the main menu.

Click on the + icon to add a widget. For example, select *Chart type / Line chart*, then add a variable to be monitored.

Note: It is easier to configure a dashboard when using the B-L475E-IOT01A board because of the multiple variables coming from the various sensors.

15 MQTT Litmus Loop application

15.1 Application description

The MQTT Litmus Loop application is derived from the generic MQTT application with a few changes to allow the connection to the Litmus Loop platform.

The application connects to the Litmus Loop platform through the MQTT protocol with the credentials provided by the user.

Both the MQTT plain TCP and MQTT TLS/SSL device models are supported by the platform and the client application.

15.2 Configuration at server level

Connect to https://litmusautomation.com/ and select *Try Loop*.

To create the Litmus Loop environment, at server level, refer to the *Getting Started Guide* from Litmus at https://docs.litmusautomation.com.

15.3 Configuration at client level

Before building, define the `LITMUS_LOOP` preprocessor variable in file *GenericMQTTXcubeSample.c* by uncommenting it. This changes some topic strings and payload content formats for data posted to the Litmus Loop platform, making the data publication format compatible with the Litmus Loop platform so that the Litmus Loop dashboard can be fully exploited.

The Litmus Loop platform uses IPSO objects format. The IPSO Alliance is an organization promoting the Internet Protocol (IP) for what it calls *smart object* communications.

The client subscribes to topic *loop/req/<LoopTopicId>/json* in order to receive commands.

The client publishes to topic *loop/data/<LoopTopicId>/json* to publish its data.

`< LoopTopicId >` is a term defined for the Litmus Loop STMicroelectronics application (it does not exist on the Limus Loop server side), representing the concatenation of two strings provided by Litmus at device creation. It is built as `<client ID>/<another Litmus Loop key>`.

For example:

- If Litmus has defined the device with:

  ```
  "mqttReqTopicName":
  "loop/req/1ntalsl12345q4r5rr06x0ya5/ea9wkabcdekotufrzkef7su2j/json"
  ```

- `<LoopTopicId>` must be:

  ```
  1ntalsl12345q4r5rr06x0ya5/ea9wkabcdekotufrzkef7su2j
  ```

The application configuration string is specific to Litmus Loop as it adds the `<LoopTopicId>` at the end.

15.4 Application first launch

Except for the `<LoopTopicId>` parameter added at the end of the connection string, the steps are the same as in *Section 11.4: Application first launch on page 31* in *Chapter 11: MQTT generic application*.

15.5 Application runtime launch

The steps are the same as in *Section 11.5: Application runtime on page 33* in *Chapter 11: MQTT generic application*. The commands are also the same while topic names are different as mentioned in *Section 15.3: Configuration at client level*.

15.6 Web Litmus Loop interface

The web Litmus Loop interface can be used to publish commands to the client, subscribe to response from the client, or using a dashboard to visualize the published data.

16 HTTPS Exosite application

16.1 Application description

The application shows a basic use of Exosite IoT cloud using the HTTP REST API.

16.2 Account creation

Create an account on the Exosite web site by clicking on *Sign up* and providing an email address and a password.

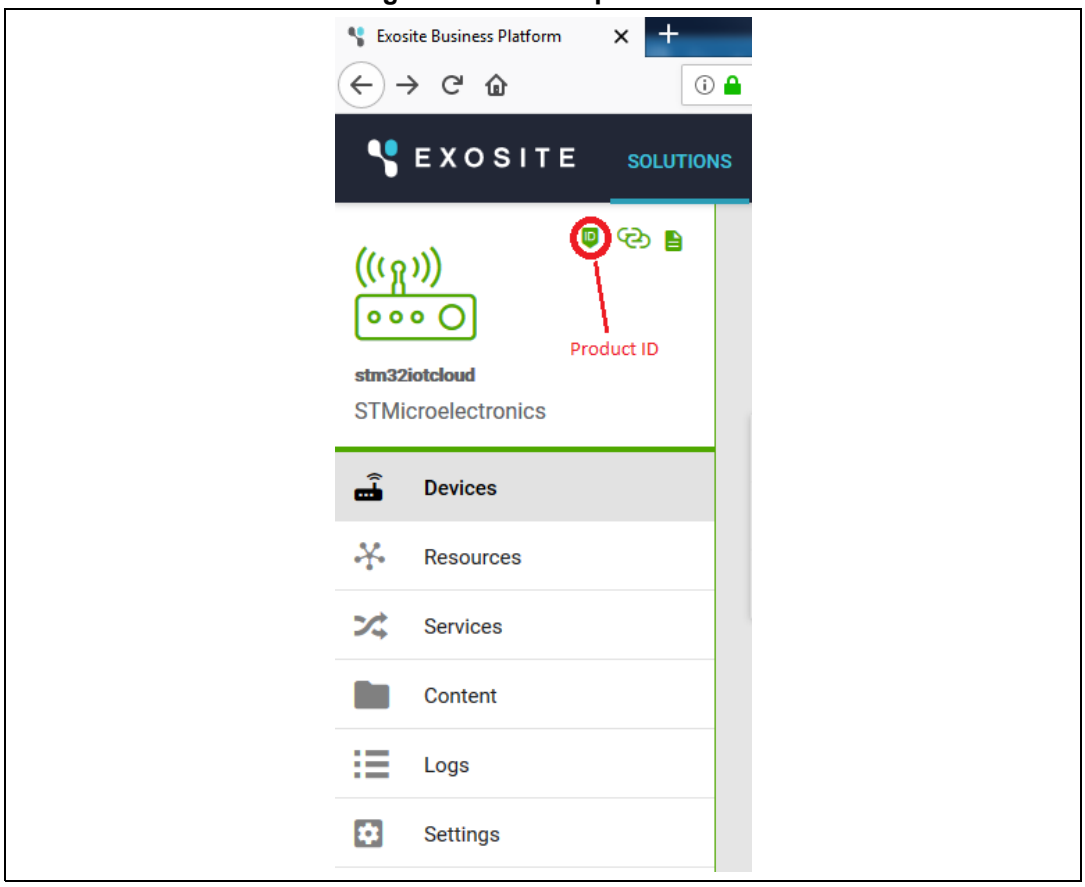
16.3 Device creation

1. Log on the Exosite web site.
2. Create a product, for example *iotcloud*.
3. Configure the product so that the device connects through HTTPS with a token.

   For *Device Identity Format*, don't enforce a device prefix.
4. Select the product.

   Take note of the product/solution ID. It is obtained by clicking on the small green *ID* icon on the top-left of the product page and selecting *Copy solution ID*. The *ID* icon is shown in *Figure 24*. The ID is later needed for the configuration of the device. Beware of a possible confusion with the product name that must be avoided.

Figure 24. Exosite product ID



5. Click first on *Devices*, then on *New Device(s) / Add one device*.
   a) Fill *Identity* with a device name
   b) No prefix is needed. It is not needed either to restrict the activation period
   c) Confirm with the *Add* button
6. In the device list, click on the new device menu represented by three vertical dots on the right. Select *Set Authentication key*. Take note of the randomly-generated token. The token is later needed for device configuration.
7. LED resource: in the resources tab, create a resource named *Led* with the boolean type.

16.4 Application first launch

1. Flash the board with the compiled application. Connect to the USB COM port of the board with a serial terminal.
2. If needed, enter the network settings.
3. Enter the device security parameters and credentials:
   – Enter the Product ID as indicated in the Exosite web site
   – Enter the device token/CIK noted previously (refer to *Section 16.3*)
   – Enter the TLS security certificate available in *Projects\Common\Exosite\DigiCert_Comodo.pem*

16.5 Application runtime

Once correctly configured, the application connects to the Exosite cloud. It requests the value of the *Led* resource every few seconds.

Press the User button (blue button) once to toggle the *Led* status and send the new value to the Exosite Cloud.

Press the User button twice to end the application.

On the B-L475E-IOT01A board, the application collects the board sensor values every ten to fifteen seconds and sends them to the Exosite cloud. This is needed to create the corresponding resources in the Exosite cloud so that the values are visible afterwards.

The resources sent by the B-L475E-IOT01A board to the Exosite cloud are the following:

- Temperature
- Humidity
- Pressure
- Proximity
- Acc_x
- Acc_y
- Acc_z
- Gyr_x
- Gyr_y
- Gyr_z
- Mag_x
- Mag_y
- Mag_z

16.6 Dashboard use

The resource values are visible on the Exosite cloud by clicking on the device name.

The log of resource changes is visible in *Logs*.

17 HTTPS Grovestreams application

17.1 Application description

The application shows the basic use of the Grovestreams IoT cloud with the HTTP REST API.

The application regularly gets the LED status from the cloud. If the User button is pressed, the LED status is changed between *On* and *Off*. The application reads the LED status from the cloud and changes the LED status on the board.

17.2 Account creation

1. Open a web browser window on the Grovestreams web site and create an account by selecting *Sign up*
2. Sign-in with the created account
3. Create an *Organization* and enter it

17.3 Device creation

Component creation:

1. In *Observation Studio*, create a *Component* by right-clicking on the *Components* folder and selecting *New Component*
2. Give the component a name, for example *iotboard*
3. Enter a component ID, for instance *1234*. Take note of the component ID, which is needed afterwards during board configuration. Beware of a possible confusion with the component name.

Led stream creation:

1. Right-click on the component and select *Edit component*
2. In *Streams*, create a *Led* stream with:
   – Name: *Led*
   – ID: *Led*
   – Type: *Boolean*
   – No specific Unit

The application uses *Stream ID Led* to publish the LED status in the *Led* stream.

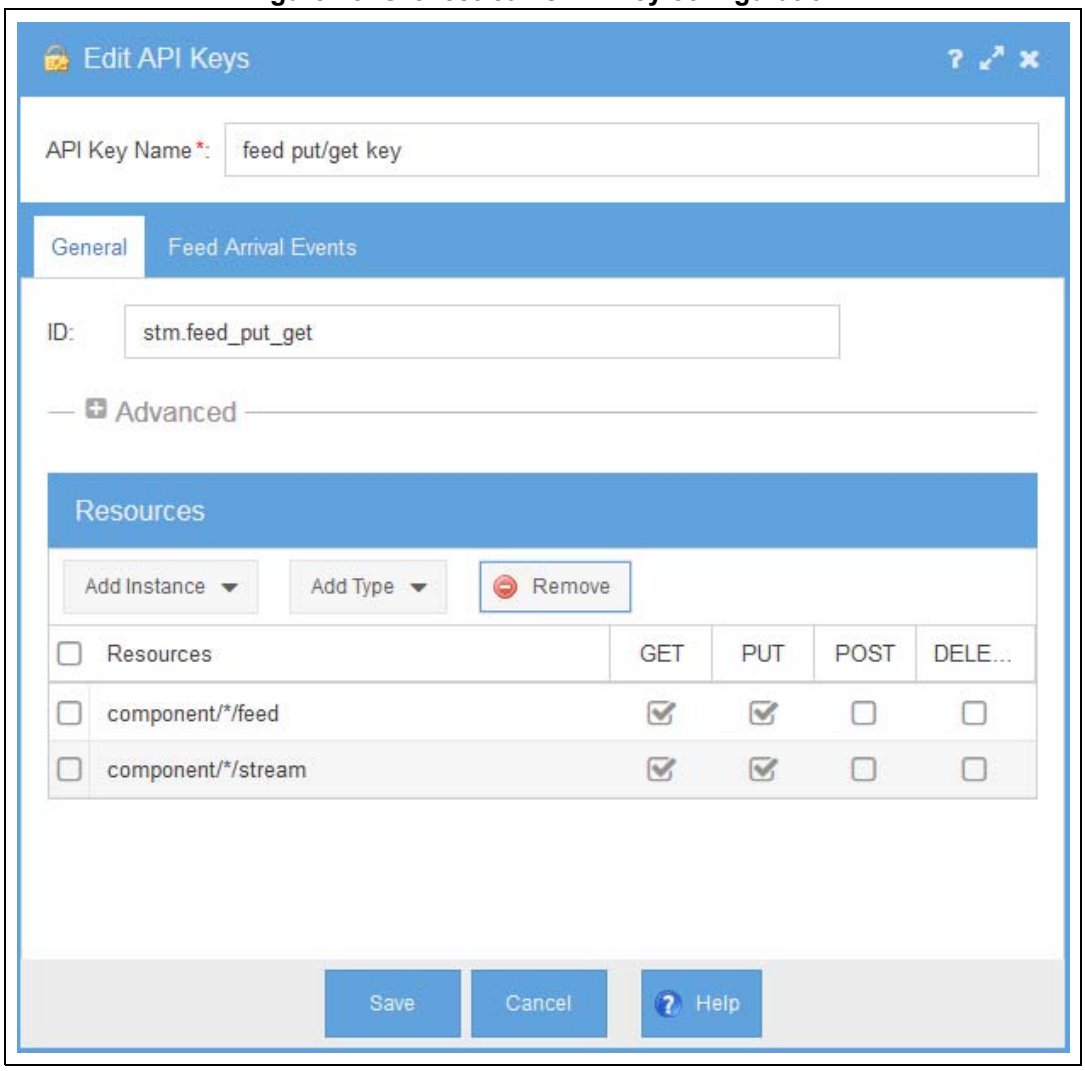
API key:

1. In the top menu, select *Admin / API key*
2. In the *API keys* box, click on + to create a new key
3. Give the key a name such as *Put/get key*, and an ID such as *PUT_GET_KEY*
4. In *Resources*, click on *Add Type* and add *Component/*/feed* as well as type *Component/*/stream*

   For both types, thick the check-boxes to allow *PUT* and *GET* actions.
5. Click on *Save* to exit the *API key* box.

   API key steps *1* to *5* are illustrated in *Figure 25*.

Figure 25. Grovestreams API key configuration



6. Select the API key just created, and click on the *View secret key* button
7. Note the API secret key that is displayed for later use (refer to *Section 17.4*) and close the box

17.4 Application first launch

1. Flash the board with the compiled application. Connect to the USB COM port of the board with a serial terminal.
2. When asked, enter the device security parameters or credentials:
   – Enter the API key noted previously (refer to *Section 17.3*)
   – Enter the component ID as noted previously (refer to *Section 17.3*). Beware of a possible confusion with the component name.
   – Enter the TLS security certificate available in *Projects\Common\Grovestreams\godaddy_comodo.pem*

17.5 Application runtime

By default, the Grovestreams application tries to connect to the Grovestreams cloud using the parameters set as described in *Section 17.4*.

Push the User button (blue button) to change the LED value and send it to the cloud. After a few seconds, the application reads it back from the cloud and makes the LED change accordingly.

In the case of the B-L475E-IOT01A board, the board sensor values are sent every ten to twenty seconds to the cloud.

17.6 Dashboard use

1. In *Observation Studio*, click on *Dashboards*
2. In the *Dashboards* view, click on *contents*, then right-click on *New / Dashboard*
3. Enter *Dashboard settings* and add a stream to view.

Note: It is easier to configure a dashboard when using the B-L475E-IOT01A board because of the multiple variables coming from the various sensors.

18 HTTPS Ubidots application

18.1 Application description

The application connects to Ubidots IoT Cloud through the HTTP REST API with the credentials provided by the user. When the User button is pushed, it sends a LED toggle command to the IoT Cloud endpoint, which returns the message to the board and triggers the LED toggle.

Ubidots offers different services: *Ubidots* and *Ubidots for Education*. The application works with *Ubidots for Education*.

To get access to the *Ubidots for Education* web site, open a web browser on the Ubidots site and select *Industries / Education*.

18.2 Account creation

The account for HTTPS Ubidots is the same as for MQTT Ubidots:

- If an account has already been created for MQTT Ubidots, use it for HTTPS Ubidots. Refer to *Chapter 14: MQTT Ubidots application* for details.
- If no account exists already, connect with a web browser to the *Ubidots for Education* web site and create an account by providing an email and a password.

Make sure to avoid a possible confusion with Ubidots for industry web site:

- Ubidots offers different services:
  - *Ubidots*, which is for industry and business
  - *Ubidots for Education*
- The application works with the *Ubidots for Education* service
- To get access to the *Ubidots for Education* web site, open a web browser on the Ubidots site, and select *Industries / Education*.

Once the account is created, login to the *Ubidots for Education* web site.

18.3 Device creation

For the device creation, apply the following:

1. In the top-right menu, click on *Devices* and add a device using the + button.
2. Give the device a name, for instance *iotboard*.
3. Click on *API Label* and enter a label, for instance *1234*; Take note of the device API label.

For the security token, apply the following:

1. On the top-right of the screen, click on the account name to open the account menu.
2. In the account menu, click on *API credentials;* Take note of the *Default token* on the right.
3. Copy-paste the default token in a text file. Do not copy the *API key* on the left.

18.4 Application first launch

1. Flash the board with the compiled application
2. Connect to the board USB COM port with a serial terminal
3. Enter the device security parameters or credentials:
   a) Enter the API Label of the device
   b) Enter the *default token* noted previously (refer to *Section 18.3*).
   Do not enter the *API key*
   c) Enter the TLS security certificate available in
   Projects\Common\Ubidots\ DSTRootCAX3_comodo.pem

18.5 Application runtime

By default, the Ubidots application automatically tries to connect to Ubidots cloud using the parameters set in *Section 18.4*.

Push the User button (blue button) to toggle the LED value and send it to the cloud. After a few seconds, the application reads it back from the cloud and changes the LED status accordingly.

In the case of B-L475E-IOT01A board, the sensor values are sent every ten to twenty seconds to the cloud.

18.6 Dashboard use

On the Ubidots web site, in a device window, it is possible to see the graphs of the variables when they change over time. It is also possible to create a dashboard by clicking on *Dashboards* top-right in the main menu.

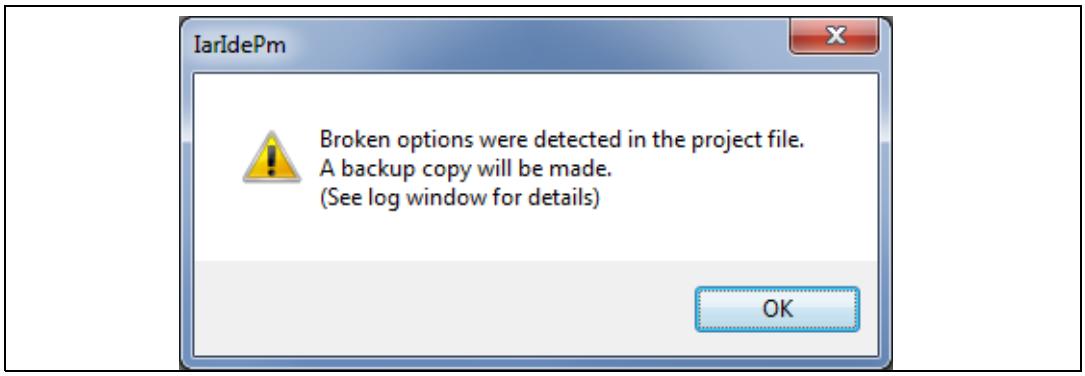
Click on the + icon to add a widget. For example, select *Chart type / Line chart*, then add a variable to be monitored.

Note: It is easier to configure a dashboard when using the B-L475E-IOT01A board because of the multiple variables coming from the various sensors.

19 Frequently asked questions

Q: Why do I get this pop-up (refer to *Figure 26*) when I open the project with IAR™?

Figure 26. Pop-up when the IAR™ IDE version is not compatible with the one used for X-CUBE-CLD-GEN



A: It is very likely that the IAR™ IDE version is older than the one used to develop the package (refer to the release notes available in the package root folder for the IDE versions supported), hence the compatibility is not ensured. In this case, the IAR™ IDE version needs to be updated.

Q: My device does not connect to the Wi-Fi® access point. How shall I proceed?

A: Make sure that another device can connect to the Wi-Fi® access point. If it can, enter the Wi-Fi® credentials by pressing the User button (blue) up to five seconds after board reset.

Q: The proximity sensor always reports "8190" even if I place an obstacle close to it.

A: Make sure that the liner (which is a very thin film placed on the proximity sensor) has been removed. Its color is orange and it is not very visible.

20 Revision history

Table 8. Document revision history

Date	Revision	Changes
5-Apr-2018	1	Initial release.
19-Apr-2018	2	Updated: – *Section 10.3: Running the application*, item *2* for EMnify conditions and item *3* for EMnify API naming. – *Section 11.4: Application first launch*, item *6* for EMnify API naming. – *Section 14.5: Application runtime*, for LED behavior.