User manual

Getting started with Google Cloud Platform™ Expansion Package for STM32Cube

Introduction

This user manual describes the content of the STM32Cube™ Expansion Package for the Google Cloud IoT Core service of Google Cloud Platform™ (GCP).

The STM32Cube Expansion Package (X-CUBE-GCP) for GCP provides application examples that connect STMicroelectronics boards to the Google Cloud IoT Core of Google Cloud Platform™. It uses an MQTT client library ported onto the corresponding STM32 devices to enable the connection to the cloud platform.

X-CUBE-GCP 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 support cellular connectivity, with the 2G/3G (UG96) and LTE (BG96) Quectel cellular modem daughterboards respectively

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

Implementation examples are included for device-to-cloud telemetry reporting, and cloud-to-device messages for secure connection to the cloud, sending commands to the cloud and receive notifications from the connected devices.
The B-L475E-IOT01A board reports telemetry data such as measurements of humidity, temperature, and atmospheric pressure.

1 General information

The X-CUBE-GCP Expansion Package for the Google Cloud IoT Core of Google Cloud Platform™ runs on STM32 32-bit microcontrollers based on the Arm® Cortex®-M processor. The definitions of the acronyms that are relevant for a better understanding of this document are presented in Table 1.

Table 1. List of acronyms

Term	Definition
API	Application programming interface
BSP	Board support package
CA	Certification authority
DHCP	Dynamic host configuration protocol
DNS	Domain name server
ECDSA	Elliptic curve digital signature algorithm
GCP	Google Cloud Platform™
HAL	Hardware abstraction layer
HTTP	Hypertext Transfer Protocol
IDE	Integrated development environment
IoT	Internet of things
IP	Internet protocol
JSON	JavaScript object notation
JWT	JSON web token
LED	Light-emitting diode
MQTT	Message queuing telemetry transport
RSA	Rivest, Shamir, and Adleman
RTC	Real-time clock
UART	Universal asynchronous receiver/transmitter

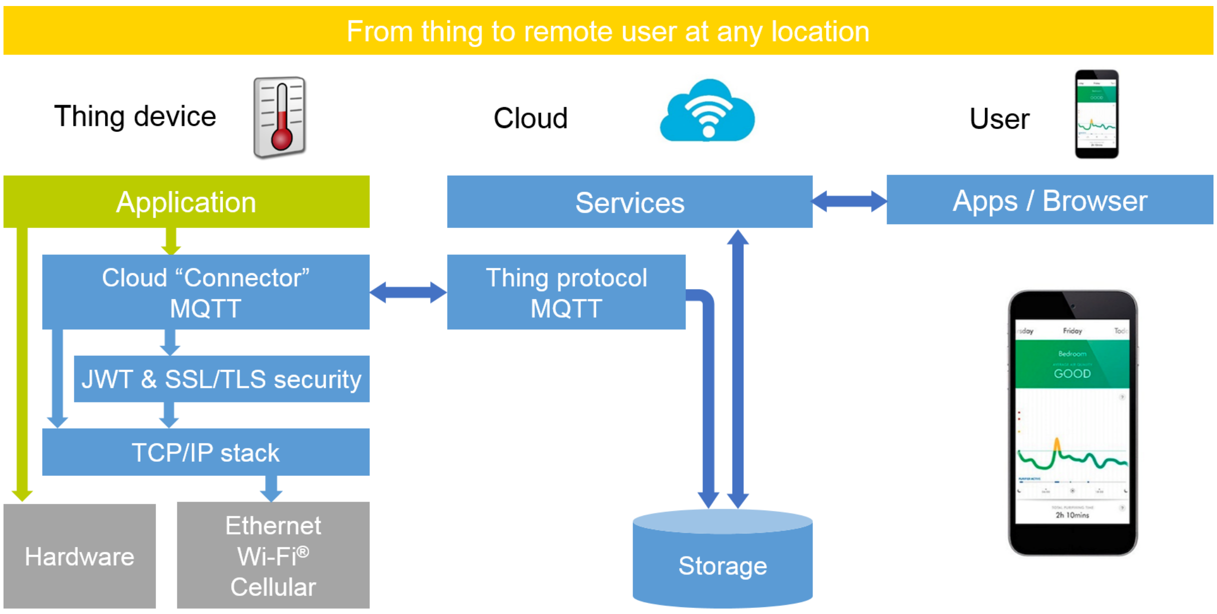
Note: Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and or elsewhere.

2 Google Cloud Platform™

This chapter introduces the Google Cloud IoT Core of Google Cloud Platform™. Detailed information related to Google Cloud Platform™ (GCP) is available from Google Cloud™ dedicated website at cloud.google.com.

X-CUBE-GCP implements an embedded C client that allows the boards to securely connect to the Google Cloud IoT Core of GCP. The GCP ecosystem targeted by X-CUBE-GCP is presented in Figure 1.

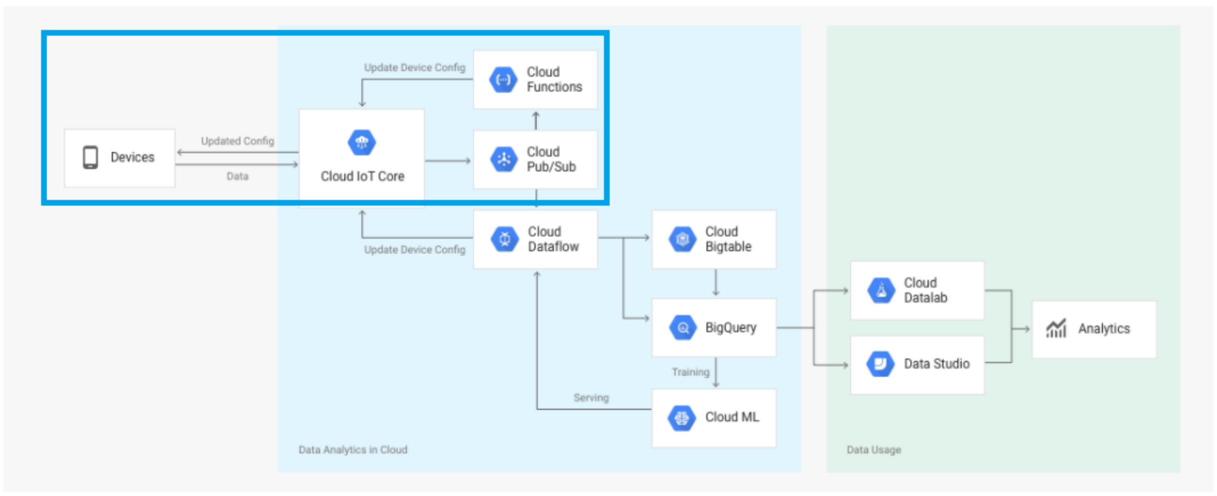
Figure 1. Google Cloud IoT Core ecosystem



The user connects to the cloud with a smartphone or personal computer and has access to the information provided by the board at any time from any location.

Google Cloud™ presents the Google Cloud IoT Core in the GCP online documentation as shown in Figure 2.

Figure 2. Google Cloud IoT Core diagram



Note: *©2018 Google LLC, used with permission. Google and the Google logo are registered trademarks of Google LLC.*

X-CUBE-GCP implements the required services to connect the device to the part of the ecosystem highlighted in blue in Figure 2.

In particular, the package demonstrates how to configure the IoT Core services of GCP in order to connect a device securely to the Google Cloud IoT Core service.

The device uses a private key (ECDSA or RSA) to sign a JWT (JSON Web Token) that is required during the connection to the cloud. The cloud IoT Core authenticates the device’s JWT by means of the corresponding public key that is uploaded to the cloud service during the device creation process.

X-CUBE-GCP also features an example showing how to format and exchange data using the MQTT protocol between the device and the IoT Core component, send telemetry data to the cloud, and receive configuration data from the cloud. Data sent to the cloud IoT Core is then published via the cloud Pub/Sub (Publish/Subscribe) service and can be accessed by a web application for further use. The development of web applications for data analytics and use is not covered in X-CUBE-GCP.

3 Package description

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

3.1 General description

The X-CUBE-GCP 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-GCP 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 Quectel's 2G/3G UG96 and LTE 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-GCP Expansion Package contains the following components:

- Eclipse™ Paho MQTT embedded C client
- 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 all MCU boards
- STM32L4 Series, STM32F4 Series, and STM32F7 Series HAL
- Application examples

Software is provided as a zip archive containing the 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 in X-CUBE-GCP. X-CUBE-GCP is an expansion software for STM32Cube™. Its main features and characteristics are:

- Fully compliant with STM32Cube™ architecture
- Expands STM32Cube™ in order to enable the development of applications accessing and using the Google Cloud IoT Core of Google Cloud Platform™
- Based on the STM32CubeHAL, which is the hardware abstraction layer for STM32 microcontrollers

The software components used by the application software to access and use the Google Cloud IoT Core of Google Cloud Platform™ 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, to implement their functionalities without dependencies on the specific hardware configuration for a given MCU.

This structure improves the library code reusability and guarantees an easy portability onto other devices.

- **Board support package (BSP)**

  The board software package needs to support the peripherals on the STM32 boards apart from the MCU. This software is included in the board support package. 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.
- **mbedTLS**

  The MQTT middleware uses a TLS connection that is managed by the mbedTLS library.
- **MQTT Client Middleware**

  It is composed of the Eclipse™ Paho MQTT embedded C client library (used as a transport layer by the MQTT applications), and JSON parser.
- **TCP/IP**

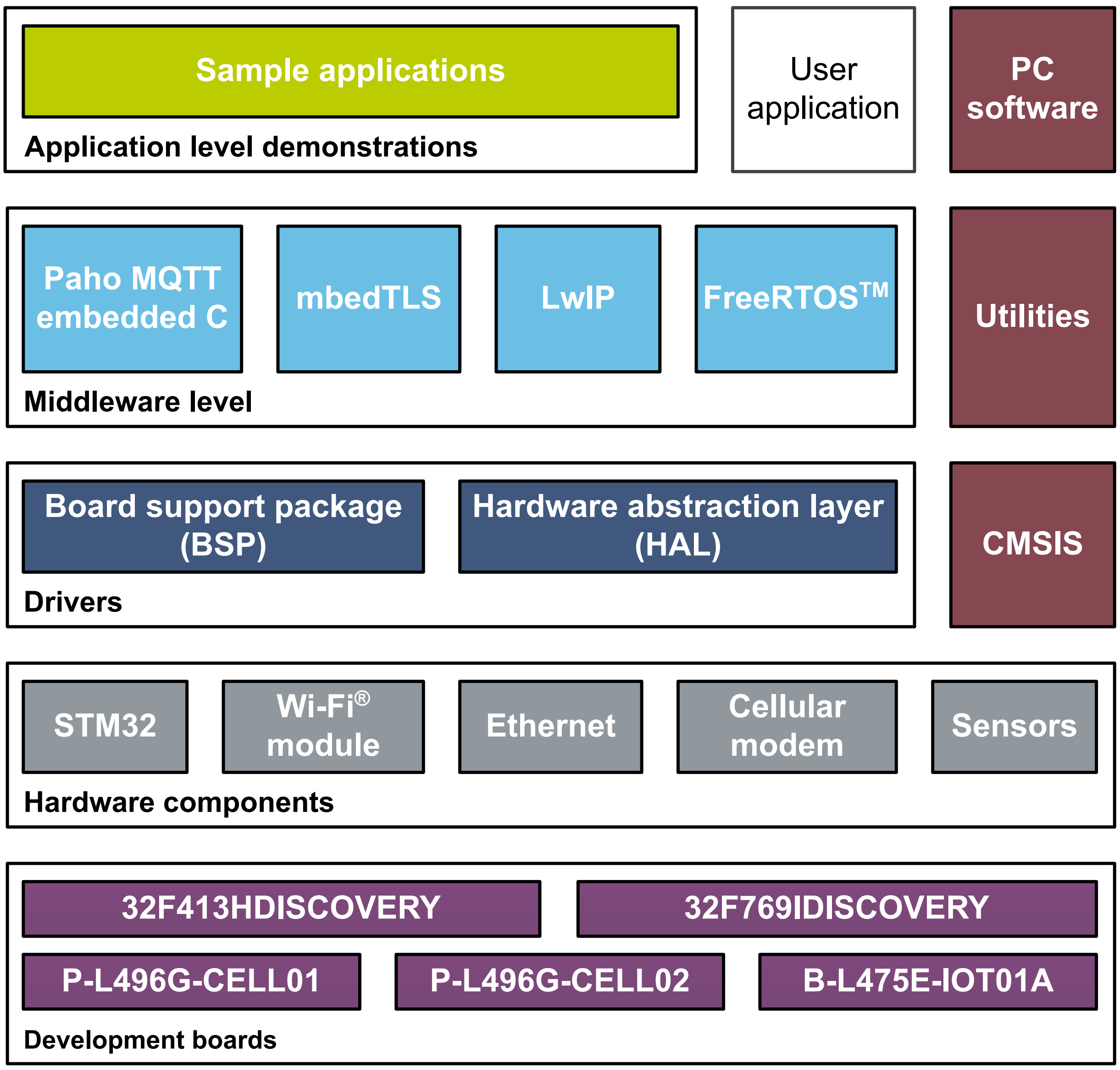
  The TCP/IP connection can be handled either by the Wi-Fi® module (when a Wi-Fi® connection is being used), by the LwIP TCP-IP stack (when an Ethernet connection is being used), or from the cellular connectivity (when a cellular expansion board is used).
- **FreeRTOS™**

  It is a real-time operating system required by LwIP for providing a socket-based interface to the user.
- **Google Cloud™** IoT sample application

  A sample application that implements the required device services for connecting to the Google Cloud IoT Core component of GCP. This includes connectivity management, JWT secure connection handling, and proper message formatting for interaction with the cloud.

The architecture of X-CUBE-GCP software is shown in Figure 3.

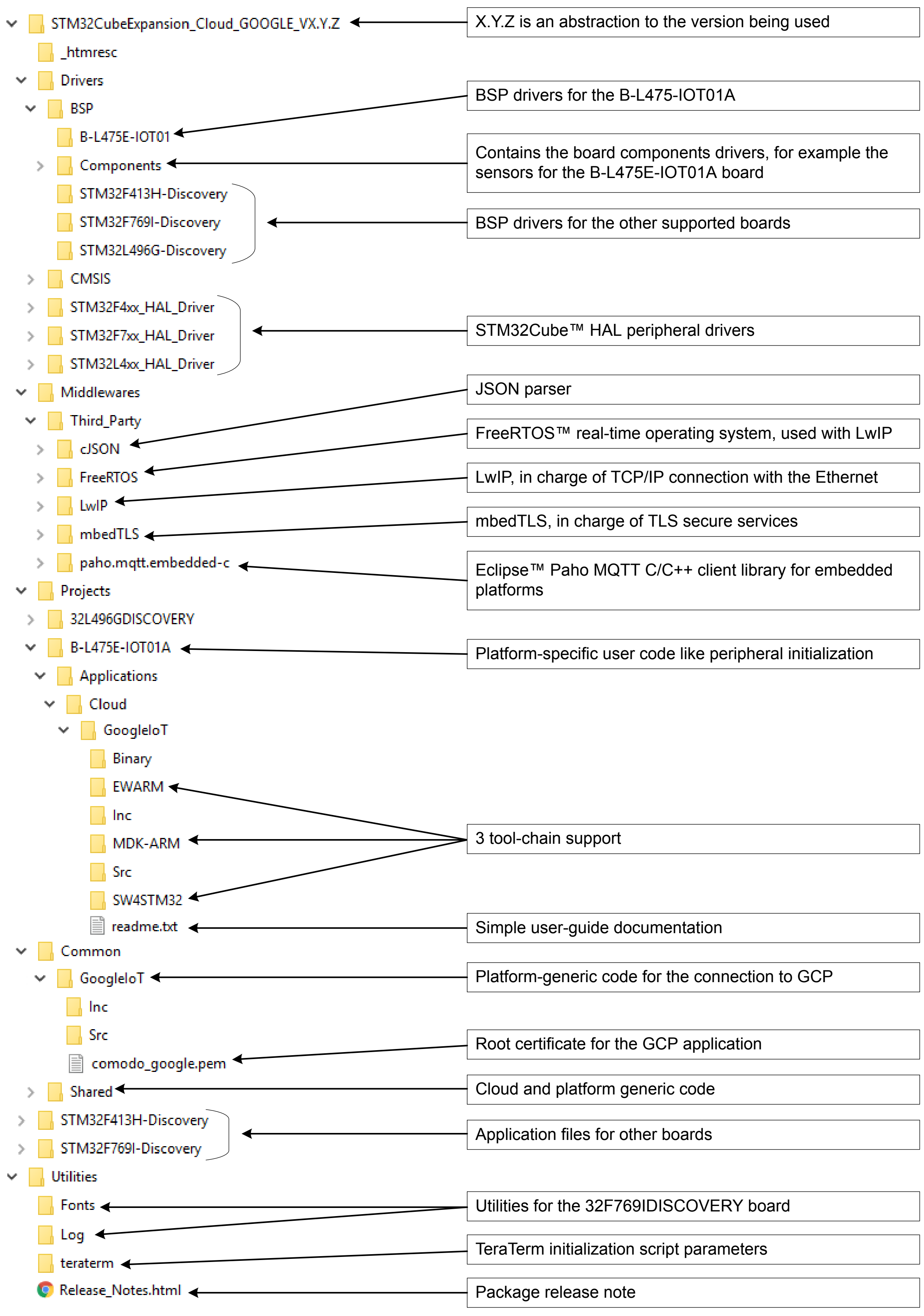
Figure 3. X-CUBE-GCP software architecture



3.3 Folder structure

The folder structure of the X-CUBE-GCP Expansion Package is presented in Figure 4.

Figure 4. Project file structure


STM32CubeExpansion_Cloud_GOOGLE_VX.Y.Z
_htmresc
Drivers
BSP
B-L475E-IOT01
Components
STM32F413H-Discovery
STM32F769I-Discovery
STM32L496G-Discovery
CMSIS
STM32F4xx_HAL_Driver
STM32F7xx_HAL_Driver
STM32L4xx_HAL_Driver
Middlewares
Third_Party
cJSON
FreeRTOS
LwIP
mbedTLS
paho.mqtt.embedded-c
Projects
32L496GDISCOVERY
B-L475E-IOT01A
Applications
Cloud
GoogleIoT
Binary
EWARM
Inc
MDK-ARM
Src
SW4STM32
readme.txt
Common
GoogleIoT
Inc
Src
comodo_google.pem
Shared
STM32F413H-Discovery
STM32F769I-Discovery
Utilities
Fonts
Log
teraterm
Release_Notes.html
X.Y.Z is an abstraction to the version being used
BSP drivers for the B-L475-IOT01A
Contains the board components drivers, for example the sensors for the B-L475E-IOT01A board
BSP drivers for the other supported boards
STM32Cube™ HAL peripheral drivers
JSON parser
FreeRTOS™ real-time operating system, used with LwIP
LwIP, in charge of TCP/IP connection with the Ethernet
mbedTLS, in charge of TLS secure services
Eclipse™ Paho MQTT C/C++ client library for embedded platforms
Platform-specific user code like peripheral initialization
3 tool-chain support
Simple user-guide documentation
Platform-generic code for the connection to GCP
Root certificate for the GCP application
Cloud and platform generic code
Application files for other boards
Utilities for the 32F769IDISCOVERY board
TeraTerm initialization script parameters
Package release note

3.4 B-L475E-IOT01A board sensors

The on-board sensors 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)

The units for the values reported by the sensors of the B-L475E-IOT01A board are presented in Table 2.

Table 2. 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)

3.5 Wi-Fi® components

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

3.6 Reset push button

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

3.7 User push button

The user push button (blue) is used in the following cases:

- To configure the Wi-Fi® and Google Cloud™ security credentials. This can be done from the moment the board starts up until five seconds after.
- When the board is initialized, if the user push button is pressed shortly, the application publishes the sensor values for the B-L475E-IOT01A board only, a 0/1 toggle value (the green LED switches accordingly), and a timestamp. On button double press, the application enters a loop and publishes automatically every second. The next double press restores the previous mode. The application configures and manages the user button via the board support package 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.

3.8 User LED

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

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

Using the BSP 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.

3.9 Real-time clock

The RTC of the STM32 device is updated at startup from the www.gandi.net web server. The user can use the HAL_RTC_GetTime() function to get the time value. This function is used, for instance, to time stamp messages.

3.10 mbedTLS configuration

The mbedTLS middleware support is entirely configurable by means of a *#include* configuration file. The name of the configuration file can be overridden by means of the MBEDTLS_CONFIG_FILE #define directive. TheX-CUBE-GCP package uses file *googleiot_mbedtls_config.h* for project configuration. This is implemented by setting 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 the ciphers to integrate.

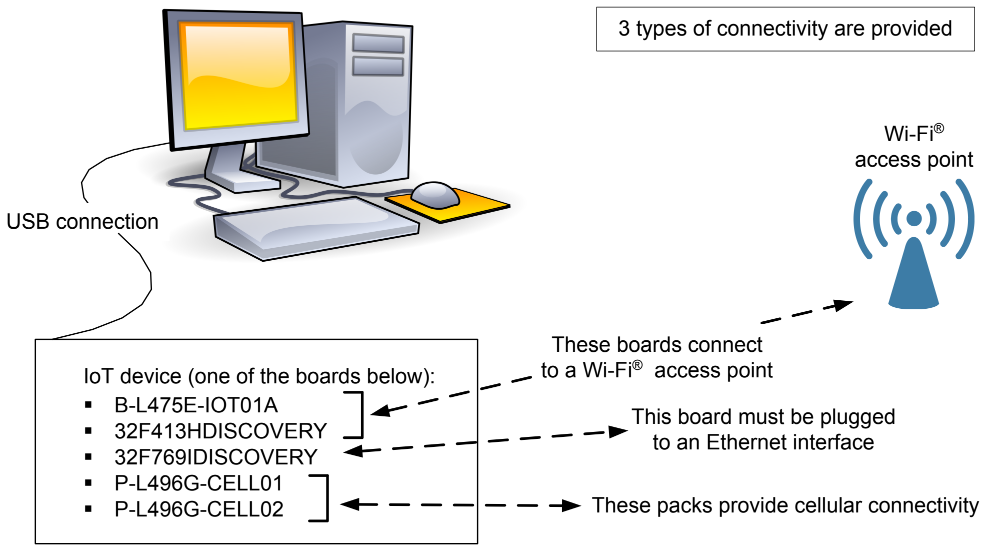
4 Hardware and software environment setup

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

- Flash the board
- Store the Wi-Fi® and the Google Cloud™ security credentials
- Interact with the board via a UART console
- Debug

The B-L475E-IOT01A and 32F413HDISCOVERY boards must be connected to a Wi-Fi® access point, the P-L496G-CELL01 and P-L496G-CELL02 packs must have their antenna plugged in to connect to a surrounding cellular network, and the 32F769IDISCOVERY board must be connected to an Ethernet interface as illustrated in Figure 5.

Figure 5. Hardware and software setup environment


3 types of connectivity are provided
Wi-Fi® access point
USB connection
These boards connect to a Wi-Fi® access point
IoT device (one of the boards below):
B-L475E-IOT01A
32F413HDISCOVERY
32F769IDISCOVERY
P-L496G-CELL01
P-L496G-CELL02
This board must be plugged to an Ethernet interface
These packs provide cellular connectivity

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 it through ST-LINK, and running the virtual console.
- A computer running a web browser for the connection to the Google Cloud™ console, or the GCLOUD SDK command line tool to administrate the Google Cloud Platform™. If any, the computer firewall must let the MQTT connections in (typically the ports 1883 and 8883).

  This can be a development PC, a virtual private server, or a single-board computer for instance.
- The OpenSSL toolset, to build a new device public / private key pair.

5 Application example

This section introduces how to register and log on the Google Cloud IoT Core of Google Cloud Platform™, and how to use the GCP IoT Core application from the X-CUBE-GCP Expansion Package. It shows how to configure GCP from the Google Cloud™ console, but the same configuration can be applied by using the command line tool from Google Cloud™ SDK. Refer to Google Cloud™ SDK documentation for more details about the command-line tool.

5.1 Application description

The application offers the services listed below:

- **Connectivity management**

  Application first configures the board connectivity (Wi-Fi®, Ethernet or Cellular) and ensures that the connectivity is up and running so as to provide a valid IP address and IP connectivity.
- **Secure connection through JWT**

  Once IP connectivity is available, the application securely connects to the GCP IoT Core service by means of a JSON Web Token (JWT) creation. There are several key formats supported in GCP IoT core. Two key formats are supported in this application: ES256 and RS256.
- **Sending / getting Device State**

  Once securely connected, the application sends an updated Device State to GCP IoT Core. This allows getting device information from GCP console. The Device State is an internal representation of the device.

  In X-CUBE-GCP application, the Device State is a simple JSON string including the board LED state, telemetry interval, local timestamp, MAC address, and current firmware version. Here is an example of a published Device State:

```
{
  "LedOn": false,
  "TelemetryInterval": 60,
  "ts": 1530170497,
  "mac": "C47F5104341",
  "FW version": "V1.1.0"
}
```

- **Configuring devices**

  GCP provides the ability to modify device configuration. As defined in GCP documentation, a device configuration is an arbitrary, user-defined blob of data. The X-CUBE-GCP application uses configuration to control the LED state from the cloud, change the telemetry interval, and force reboot. The application accepts configurations that are in simple JSON format as shown below:

```
{
"LedOn": true,
"TelemetryInterval":60,
"Reboot": false
}
```

  X-CUBE-GCP registers for configuration at each connection. The GCP IoT Core service always sends the latest configuration at each new registration. Therefore, if the reboot command `"Reboot": true` is sent in a configuration, the user must update the confirmation back to `"Reboot": false` consequently in order to avoid endless reboot at each new connection.
- **Sending telemetry data through the MQTT Bridge**

  The application only supports the MQTT Bridge service of GCP IoT Core. It does not support the HTTP bridge.

  The application sends telemetry data once if the user makes a single press on the user button. If the user makes a double press on the user button, the device enters a publication loop from which telemetry data is sent periodically. The period of this loop is configured during the initial device configuration step with parameter `TelemetryInterval`.

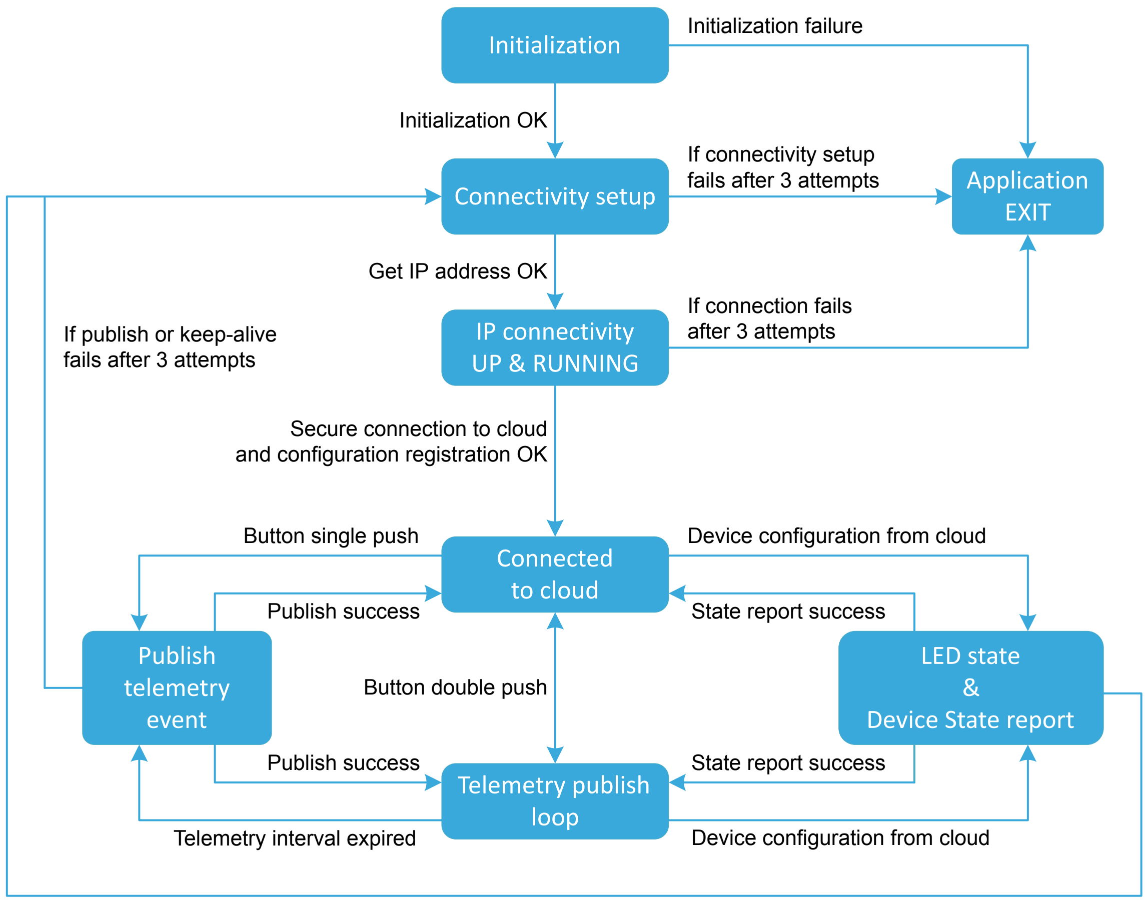
As explained in GCP online documentation, a device can publish a telemetry event by issuing a PUBLISH message through the MQTT connection. The Quality of Service used in the application example is QOS1 (delivered at least once). Messages must be published to an MQTT topic in the following format: `/devices/{device-id}/events`

An example of a published message on a board with sensors is shown below:

```
{
"data":
[{"dataType":"temp","value":"27.91"},
{"dataType":"hum","value":"48.72"},
{"dataType":"pres","value":"1015.17"}]
}
```

The application behavior is illustrated in Figure 6.

Figure 6. Application state machine


Initialization
Initialization failure
Initialization OK
Connectivity setup
If connectivity setup fails after 3 attempts
Application EXIT
Get IP address OK
IP connectivity UP & RUNNING
If connection fails after 3 attempts
If publish or keep-alive fails after 3 attempts
Secure connection to cloud and configuration registration OK
Button single push
Connected to cloud
Device configuration from cloud
Publish success
State report success
Publish telemetry event
Button double push
LED state & Device State report
Publish success
State report success
Telemetry publish loop
Telemetry interval expired
Device configuration from cloud

5.2 GCP and IoT Core account setup

A Google Cloud Platform™ account is needed for using Google Cloud IoT Core services. The user must create an account before proceeding further and create or select a GCP project.

Note: For registration, project creation, and API enabling, visit cloud.google.com.

User account creation is performed by means of the following steps:

1. Enable billing for the project as shown in Figure 7. If billing is not enabled for his project, the user cannot proceed further with the account creation remaining steps.

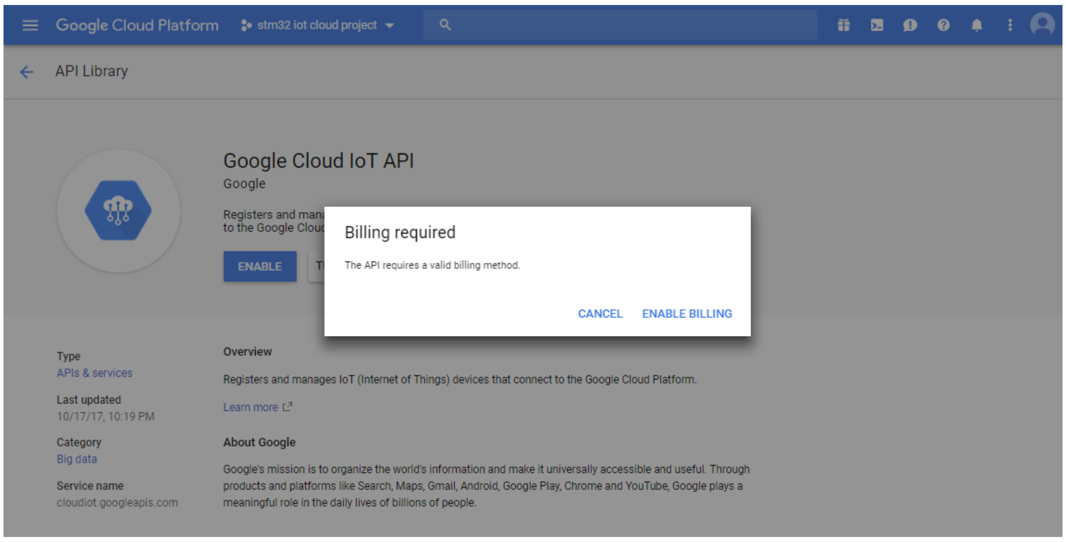


Figure 7. **Billing required**

Google Cloud IoT API
Google
Registers and mana... to the Google Clou...

Billing required
The API requires a valid billing method.

Type
APIs & services
Last updated
10/17/17, 10:19 PM
Category
Big data
Service name
cloudiot.googleapis.com

Overview
Registers and manages IoT (Internet of Things) devices that connect to the Google Cloud Platform.
Learn more

About Google
Google's mission is to organize the world's information and make it universally accessible and useful. Through products and platforms like Search, Maps, Gmail, Android, Google Play, Chrome and YouTube, Google plays a meaningful role in the daily lives of billions of people.

2. Enable the *"Google Cloud IoT Core"* and *"Pub/Sub"* APIs, which are being used by the application. In the GCP console, go to *"API & Services dashboard"* as shown in Figure 8.

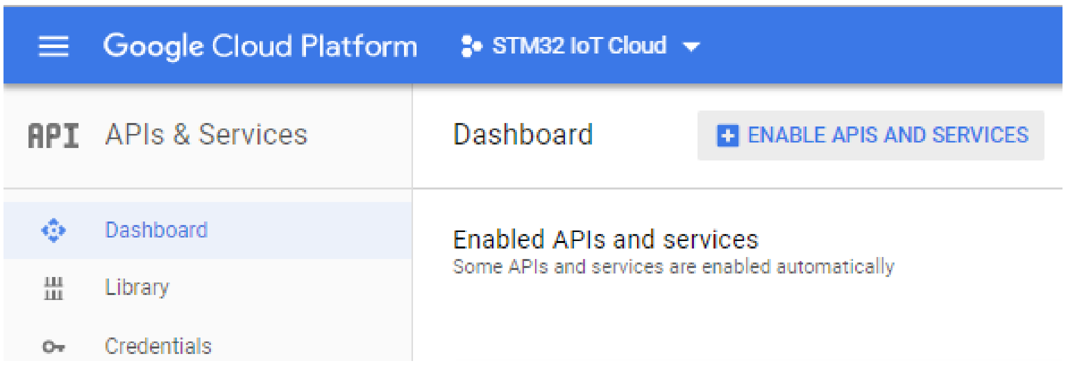


Figure 8. **Enable APIs menu**

Enabled APIs and services
Some APIs and services are enabled automatically

3. use "ENABLE APIS AND SERVICES" and enable at least Google Cloud IoT API and Cloud Pub/Sub API as shown in Figure 9. Enable Google Cloud IoT API and Figure 10. Enable Cloud Pub/Sub API.

Figure 9. **Enable Google Cloud IoT API**

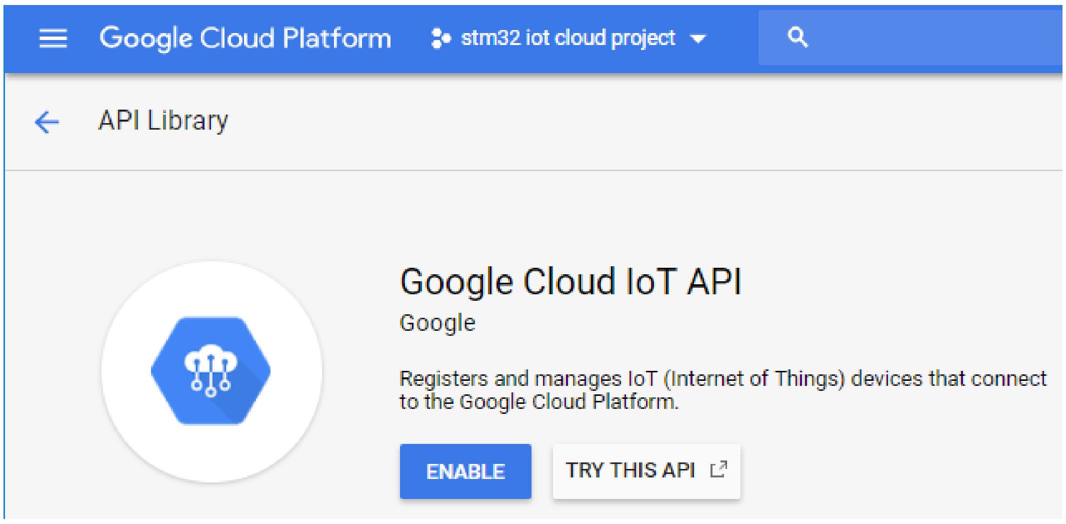
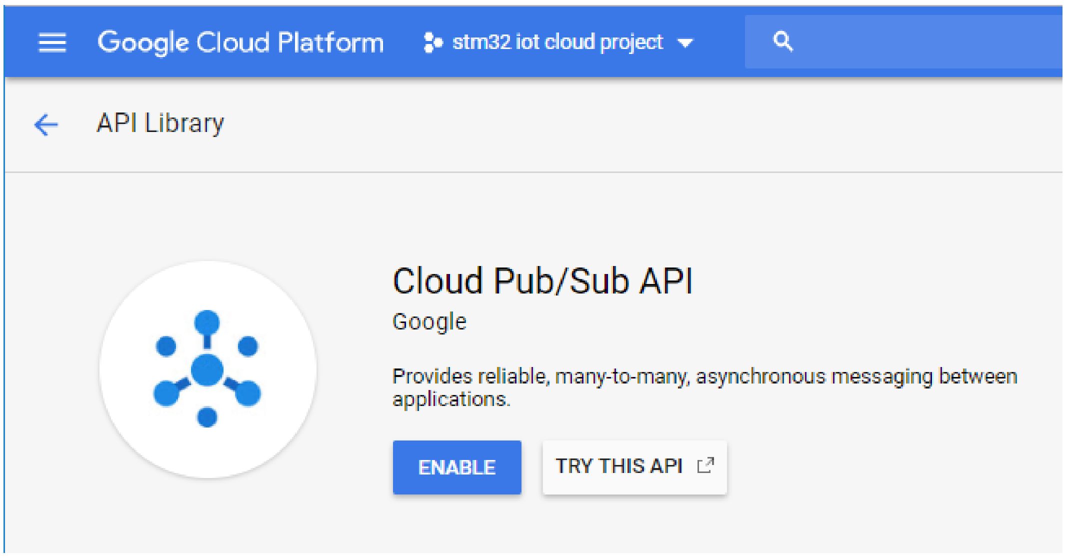


Figure 10. **Enable Cloud Pub/Sub API**



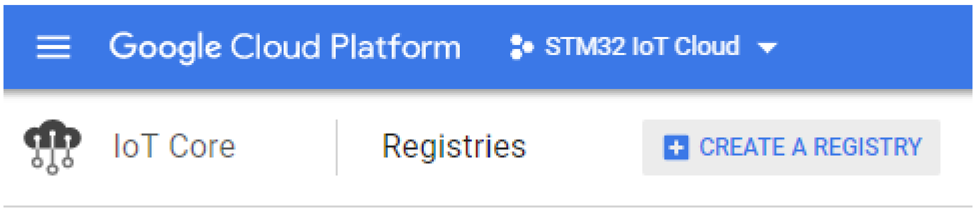
5.3 Device creation on GCP IoT core

For a complete documentation on Google Cloud IoT Core and how to create registries, devices, and telemetry topics, visit cloud.google.com/iot/docs.

Follow the steps presented below to create and register a new device:

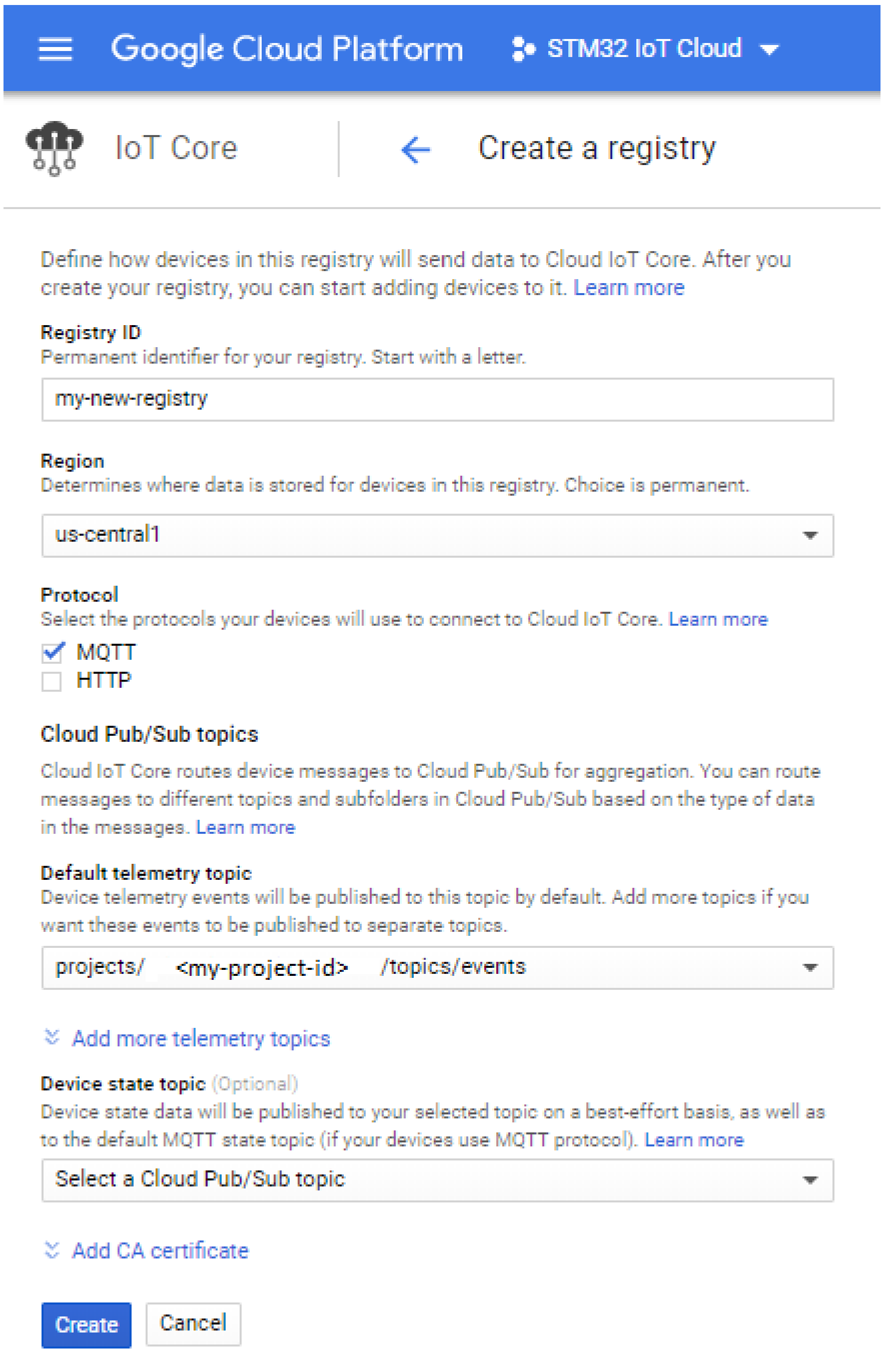
1. In Google Cloud IoT Core, create a new device registry, which is defined as `{registry-id}` in the connection string, as shown in Figure 11.

Figure 11. **GCP 'Create a registry' button**



2. In the registry creation panel, select a region (by default us-central1), which can be later configured as `{cloud-region}` in the connection string. If it does not exist, create a telemetry topic named *"events"* and maintain the default state topic. The topic must be named *"events"* since this is the name used by the application. The application only supports the MQTT protocol; HTTP is not supported and can be unselected. The user does not need to add a certificate. The window dedicated to the registry creation settings is shown in Figure 12.

Figure 12. GCP 'Create a registry' settings



Define how devices in this registry will send data to Cloud IoT Core. After you create your registry, you can start adding devices to it. Learn more

Registry ID
Permanent identifier for your registry. Start with a letter.
my-new-registry

Region
Determines where data is stored for devices in this registry. Choice is permanent.
us-central1

Protocol
Select the protocols your devices will use to connect to Cloud IoT Core. Learn more
[✓] MQTT
[ ] HTTP

Cloud Pub/Sub topics
Cloud IoT Core routes device messages to Cloud Pub/Sub for aggregation. You can route messages to different topics and subfolders in Cloud Pub/Sub based on the type of data in the messages. Learn more

Default telemetry topic
Device telemetry events will be published to this topic by default. Add more topics if you want these events to be published to separate topics.
projects/ <my-project-id> /topics/events

Add more telemetry topics

Device state topic (Optional)
Device state data will be published to your selected topic on a best-effort basis, as well as to the default MQTT state topic (if your devices use MQTT protocol). Learn more
Select a Cloud Pub/Sub topic

Add CA certificate

3. Add a new device in the newly created registry
   Define a Device ID, which is `{device-id}` in the connection string.
   Use *"openssl"* for creating the public and private keys of type RS256 or RS256 as described in the *"Creating Public/Private Key Pairs"* documentation of GCP.

– In case of RS256:

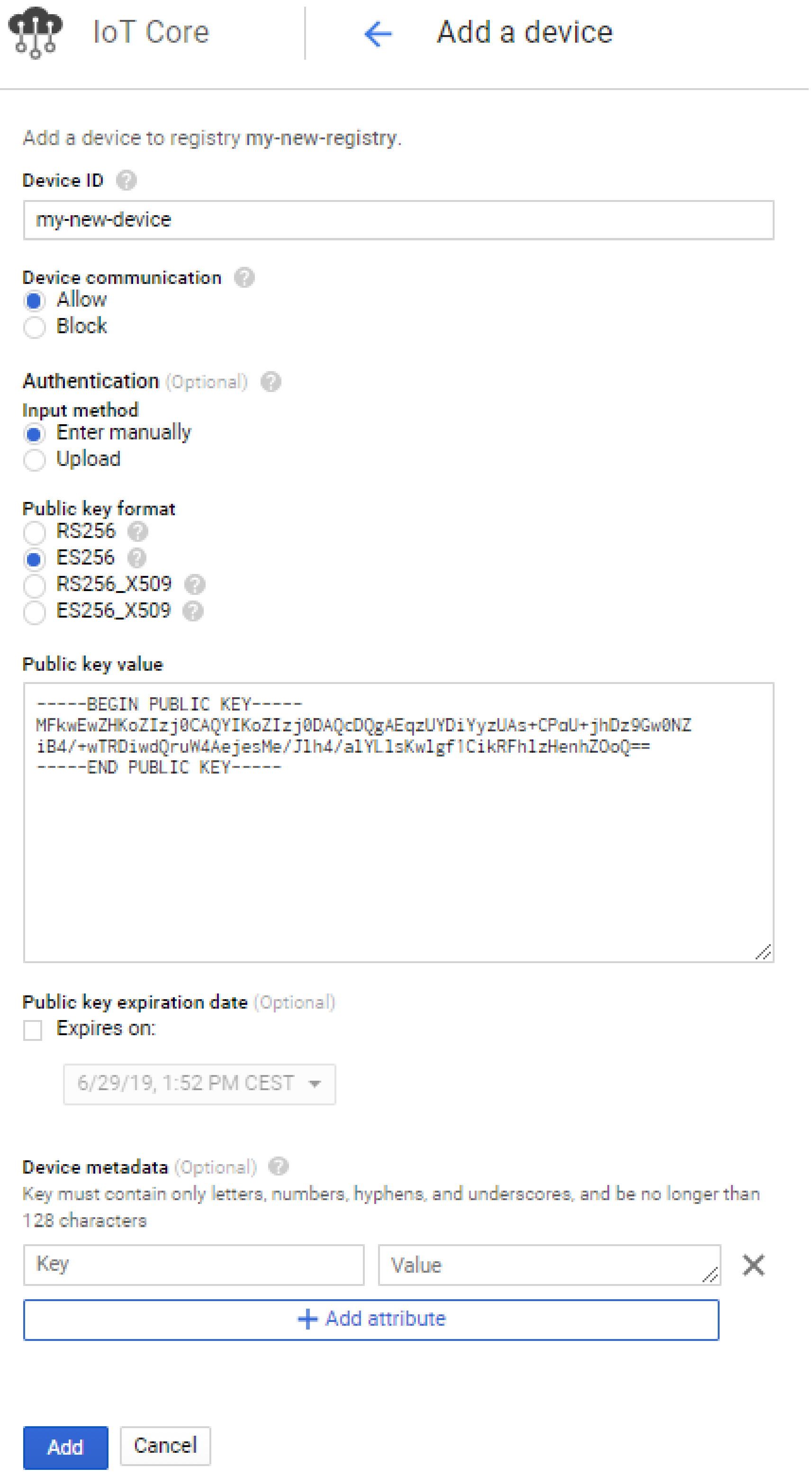
```
> openssl genrsa -out rsa_private.pem 2048
> openssl rsa -in rsa_private.pem -pubout -out rsa_public.pem
```

– In case of ES256

```
> openssl ecparam -genkey -name prime256v1 -noout -out ec_private.pem
> openssl ec -in ec_private.pem -pubout -out ec_public.pem
```

Add the public key pem content to the device and keep the private key which will be passed to the device through console interface. The device creation is illustrated in Figure 13. GCP 'Create a device' settings.

Figure 13. GCP ‘Create a device’ settings



Add a device to registry my-new-registry.

Device ID

my-new-device

Device communication

[✓ Allow] Block

Authentication (Optional)

Input method

[✓ Enter manually] Upload

Public key format

RS256 [✓ ES256] RS256_X509 ES256_X509

Public key value

```
-----BEGIN PUBLIC KEY-----
MFkwEwZHKoZIzj0CAQYIKoZIzj0DAQcDQgAEqzUYDiYyzUAs+CPaU+jhDz9Gw0NZ
iB4/+wTRDiwdQruW4AejesMe/Jlh4/alYLlsKwlgf1CikRFhlzHenhZOoQ==
-----END PUBLIC KEY-----
```

Public key expiration date (Optional)

Expires on:

6/29/19, 1:52 PM CEST

Device metadata (Optional)

Key must contain only letters, numbers, hyphens, and underscores, and be no longer than 128 characters

Key | Value

Once the steps above are performed, the cloud is set up; the device an be used and connected to the GCP.

5.4 Application build and flash

Note: 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 note 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\GoogleIoT\Binary* to the USB mass storage location created when the STM32 board is plugged to the PC. Alternatively, the user can program the STM32 board directly through one of the supported development tool chains.

5.5 Application first launch

1. The board must be connected to a PC through the USB (ST-LINK USB port). 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 hardware flow control
   – Line endings set to LF
   – *“echo on”* option to be able to read user-entered parameters
   – if available, enabling the *“automode”* option is also advised

   For more details, refer to Section 6 Interacting with the boards.

   For the Wi-Fi®-enabled boards, enter the Wi-Fi® SSID, encryption mode, and password via the console. For cellular connectivity, enter the access point and password provided by the selected operator.
2. User is prompted to enter a connection string that contains the required information related to the Project, Registry Device that have been created in GCP user account.

   Enter the Google Cloud IoT Core connection string for the device: (template: project-id=xxx;registry-id=xxx;device-id=xxx;cloud-region=xxx)

   An example connection string is:

   ```
   project-id=my-project-123456;registry-id=stm32registry;device-
   id=mystm32device;cloud-region=us-central1
   ```
3. Set the TLS root CA certificates:

   Copy-paste the content of *Projects\ Common\GoogleIoT\ comodo_google.pem*. The device uses it to authenticate the remote hosts through TLS.

   Note: the sample 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)
   b. For GCP, in order to authenticate the Cloud server. Depending on the server, the *comodo_google.pem* may need to be updated based on Google Cloud™ list of supported CAs, which is available at pki.google.com/roots.pem.
4. Set the device Private Key

   Copy-paste the content of the private key *rsa_private.pem* or *ec_private.pem* corresponding to the public key that has been created during Device Creation step in GCP console.
5. After the parameters are configured, it is possible to change them by restarting the board and pushing the user button (blue button) when prompted.

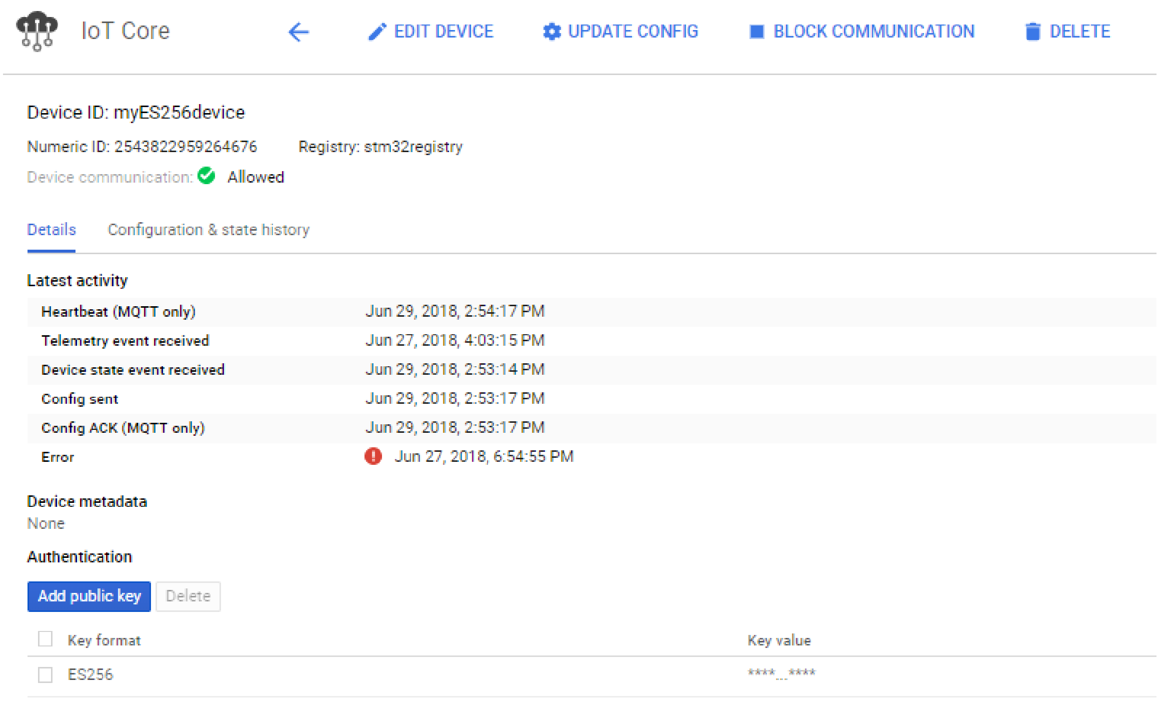
5.6 Application runtime

1. **RTC configuration**

   The application makes an HTTPS request to retrieve the current time and date, and configures the RTC.
2. **Device-to-cloud connection**

   After correct configuration of settings, the device connects to the GCP IoT core. For checking proper connection, user can check the device details from the GCP IoT core console as shown in Figure 14. IoT core device console.

Figure 14. **IoT core device console**



Device ID: myES256device

Numeric ID: 2543822959264676 Registry: stm32registry

Device communication: ✓ Allowed

Details Configuration & state history

Latest activity

Heartbeat (MQTT only)	Jun 29, 2018, 2:54:17 PM
Telemetry event received	Jun 27, 2018, 4:03:15 PM
Device state event received	Jun 29, 2018, 2:53:14 PM
Config sent	Jun 29, 2018, 2:53:17 PM
Config ACK (MQTT only)	Jun 29, 2018, 2:53:17 PM
Error	Jun 27, 2018, 6:54:55 PM

Device metadata
None

Authentication

Add public key Delete

Key format	Key value
ES256	****...****

From the same console menu, the user can:
- Check the latest error status. A typical error is *“mqtt: SERVER: The connection was closed because MQTT keep-alive check failed”* when the device is unplugged and the connection to the cloud is interrupted.
- Check the configuration and state history in the corresponding menu. The latest published Device State and Configurations are available for user reference.

3. **Telemetry published data**

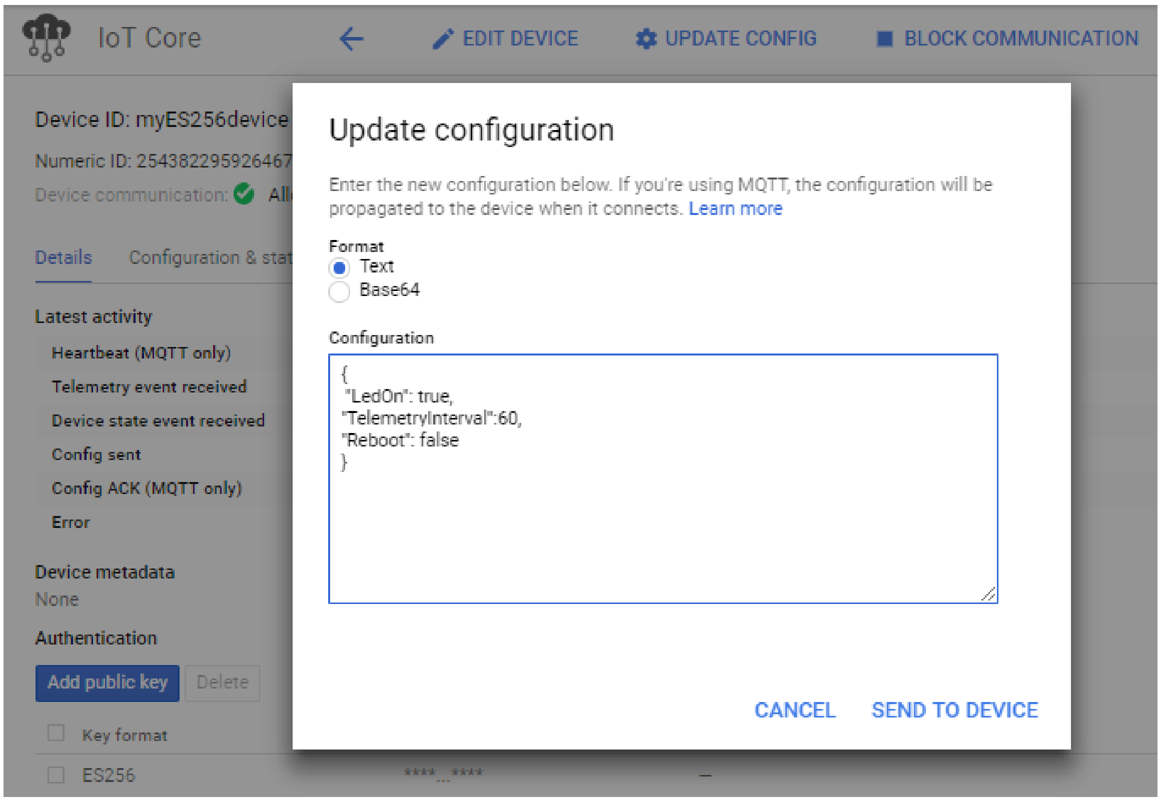
   The application publishes the sensor values if the board is the B-L475E-IOT01, a 0/1 toggle value (green LED switches accordingly), and a timestamp.
   - Once if the user button is pressed shortly once
   - At a periodic user-defined telemetry intervals if the user button is double pressed. The telemetry interval can be modified through the Device configuration.

   The IoT Core Device panel shown in the previous *“Device to Cloud connection”* step shows the date of the latest received telemetry data. GCP does not provide a ready-to-use access to the data. The user must rely on other GCP services or can alternatively use the CLOUD SDK toolset to visualize the data.

4. **Cloud-to-device connection**

   This interaction is available from the GCP console using the *“UPDATE CONFIG”* menu where commands encoded in simple JSON format can be sent to the device (refer to Figure 15. IoT core device configuration). If the device is not connected, the configuration is received at the next connection.

Figure 15. IoT core device configuration



Device ID: myES256device
Numeric ID: 254382295926467
Device communication: All

Latest activity
Heartbeat (MQTT only)
Telemetry event received
Device state event received
Config sent
Config ACK (MQTT only)
Error

Device metadata
None

Authentication

	Key format	
	ES256	**** ... ****

Update configuration

Enter the new configuration below. If you're using MQTT, the configuration will be propagated to the device when it connects. Learn more

Format
[✓ Text]
Base64

Configuration

```
{
 "LedOn": true,
"TelemetryInterval":60,
"Reboot": false
}
```

Reminder: The X-CUBE-GCP registers to configuration at each connection and GCP IoT Core service always sends the latest configuration at each new registration. Therefore, if the reboot command `"Reboot": true` is sent in a configuration, the user must consequently update the confirmation back to `"Reboot": false` in order to avoid endless reboot at each new connection.

5.7 Dashboard and plotting values

There is no default dashboard for visualizing data in GCP. It is up to the user to create a web application by means of extra GCP services. An alternative is to install GCLOUD SDK from Google Cloud™ and use the toolset to subscribe to the Pub/Sub topic.

Refer to GCLOUD SDK documentation and `gcloud pubsub` commands like: `$ gcloud pubsub subscriptions pull`

6 Interacting with the boards

A serial terminal is required to:

- Configure the board
- Monitor application status
- Display locally the 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 connection string data:

- 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 serial 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 4. Project file structure); this script sets the correct parameters. To use it, open Tera Term, select Setup and then Restore setup.

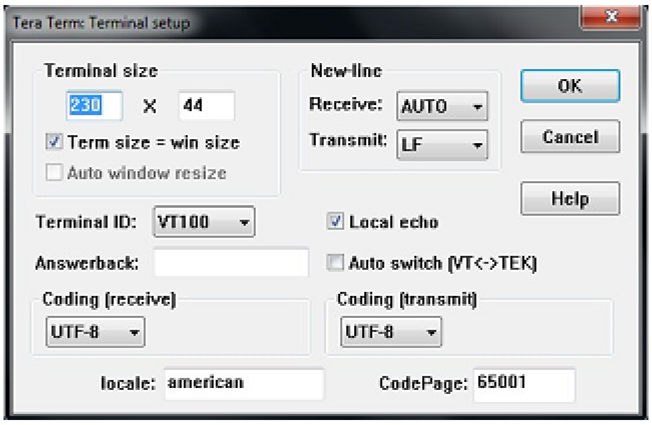
Note: *The information provided below can be used to configure the UART terminal as an alternative to using the Tera Term initialization script.*

Terminal setup

Terminal setup is illustrated in Figure 16. The screenshot shows the *Terminal setup* and the *New-line* recommended parameters.

The serial 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 selection of the *Local echo* option makes copy-paste results visible on the console.

Figure 16. Terminal setup



Terminal size: 230 X 44
[✓] Term size = win size
[ ] Auto window resize
New-line
Receive: AUTO
Transmit: LF
Terminal ID: VT100
[✓] Local echo
Answerback:
[ ] Auto switch (VT<->TEK)
Coding (receive): UTF-8
Coding (transmit): UTF-8
locale: american
CodePage: 65001

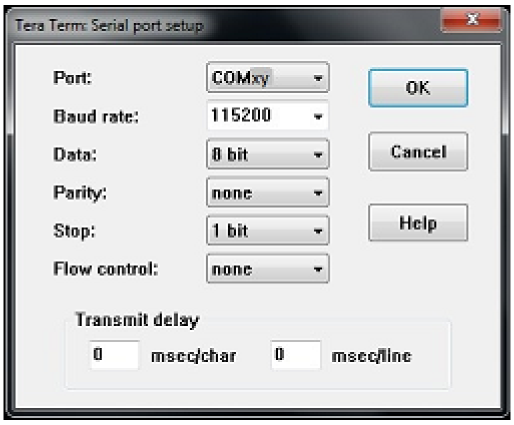
Serial port setup

The serial port must be configured with:

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

Serial port setup is illustrated in Figure 17.

Figure 17. Serial port setup


Tera Term: Serial port setup

Port: COMxy
Baud rate: 115200
Data: 8 bit
Parity: none
Stop: 1 bit
Flow control: none

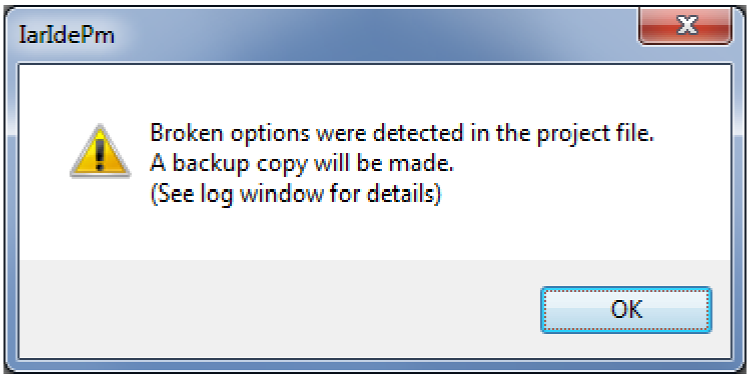
Transmit delay
0 msec/char 0 msec/line

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® and cloud configuration data. Those data remain in Flash memory and are reused the next time the board boots.

7 Frequently asked questions

Q: Why do I get this pop-up when I open the project with IAR™?

Figure 18. **Pop-up when the IAR™ IDE version is not compatible with the one used for X-CUBE-GCP**


IarIdePm

Broken options were detected in the project file.
A backup copy will be made.
(See log window for details)

A: It is very likely that the IAR™ IDE version is older than the one used to develop the package (refer to the release note 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: How shall I modify the application to publish other messages?

A: Depending whether B-L475E-IOT01A or another board is used, an update of the function `GoogleIoT_publishTelemetry ()` or `GoogleIoT_publishDeviceState ()` respectively is needed in the in file *googleiot.c*.

Q: My device does not connect to GCP. How shall I proceed?

A: Things that need to be checked are:

1. Verify the project-id, registry-id and device-id are correctly spelled and submitted in connection string. Typically, GCP often creates a uniquely defined project-id that contains automatically generated name and/or number, like <google-defined-project-123456>. So make sure to use the project-id displayed in the registry console.
2. If not successful, the initial configuration must be done again carefully.
3. If there is still no connection, check that the device public key defined for the device in the GCP console is set with the correct corresponding private key stored onto the device.

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: On the B-L475E-IOT01A board, the proximity sensor always reports *“8190”* even if I place an obstacle close to it.

A: The proximity sensor is delivered with a temporary protection film. Make sure the film is removed before using the sensor. The protection film is orange and hardly visible. On B-L475E-IOT01A, the proximity sensor is located near the bottom left corner of the front side of the board where the MCU is soldered.

Revision history

Table 3. **Document revision history**

Date	Version	Changes
6-Sep-2018	1	Initial release.

Contents

List of tables

List of figures