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 the Google Cloud Platform™ (GCP).

The STM32Cube Expansion Package (X-CUBE-GCP) for GCP provides an application example that connects an STMicroelectronics board to the Google Cloud™ IoT Core service of the Google Cloud Platform™. It is based on the Google Cloud™ IoT Device embedded C SDK ported onto STM32L4 Discovery kit IoT node to enable the connection to the cloud platform.

X-CUBE-GCP version 2.0.0 runs on the B-L475E-IOT01A Discovery kit.

For the B-L475E-IOT01A platform, the sample application configures the network connectivity parameters, and illustrates various ways for a device to interact with GCP.

An implementation example is included for device-to-cloud telemetry reporting, and cloud-to-device messages for secure connection to the cloud, sending telemetry and events data to the cloud, and receiving commands and configurations from the Google Cloud™ IoT Core service.

1 General information

The X-CUBE-GCP Expansion Package for the Google Cloud™ IoT Core of the 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
RTC	Real-time clock
SBSFU	Secure Boot and Secure Firmware Update
SDK	Software development kit
UART	Universal asynchronous receiver/transmitter

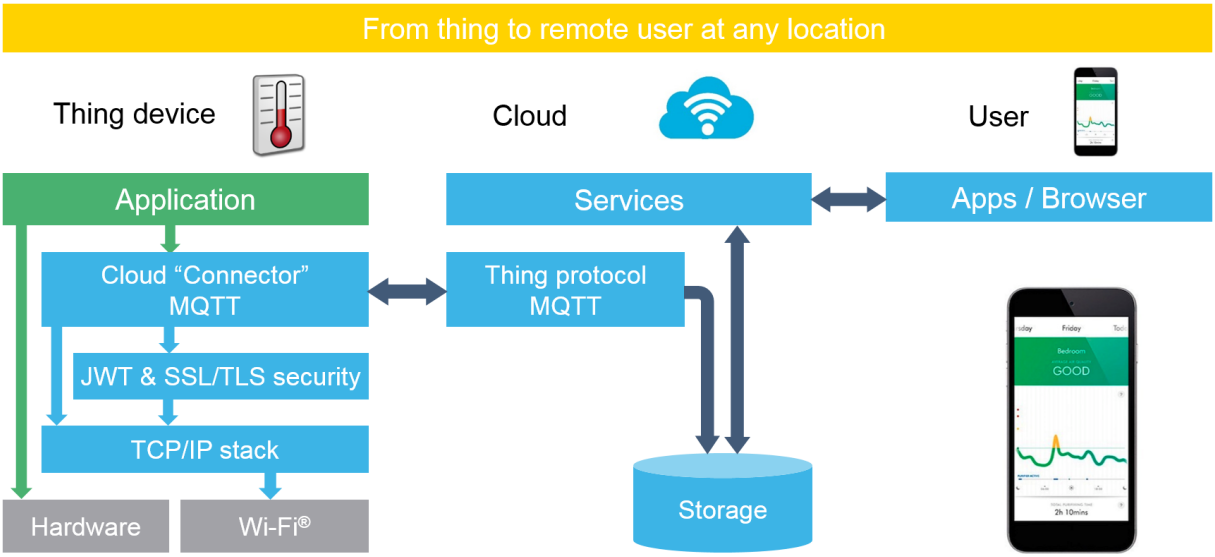
Note: Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

2 Google Cloud Platform™

This chapter introduces the Google Cloud™ IoT Core service of the Google Cloud Platform™. Detailed information related to Google Cloud Platform™ (GCP) is available from the Google Cloud™ dedicated website at cloud.google.com, and detailed information about the Google Cloud™ IoT Device embedded C SDK is available at github.com/GoogleCloudPlatform/iot-device-sdk-embedded-c.

X-CUBE-GCP is based on the Google Cloud™ IoT Device embedded C SDK ported to the B-L475E-IOT01A platform that allows the board to securely connect to the Google Cloud™ IoT Core of GCP. The GCP ecosystem targeted by X-CUBE-GCP is presented in Figure 1.

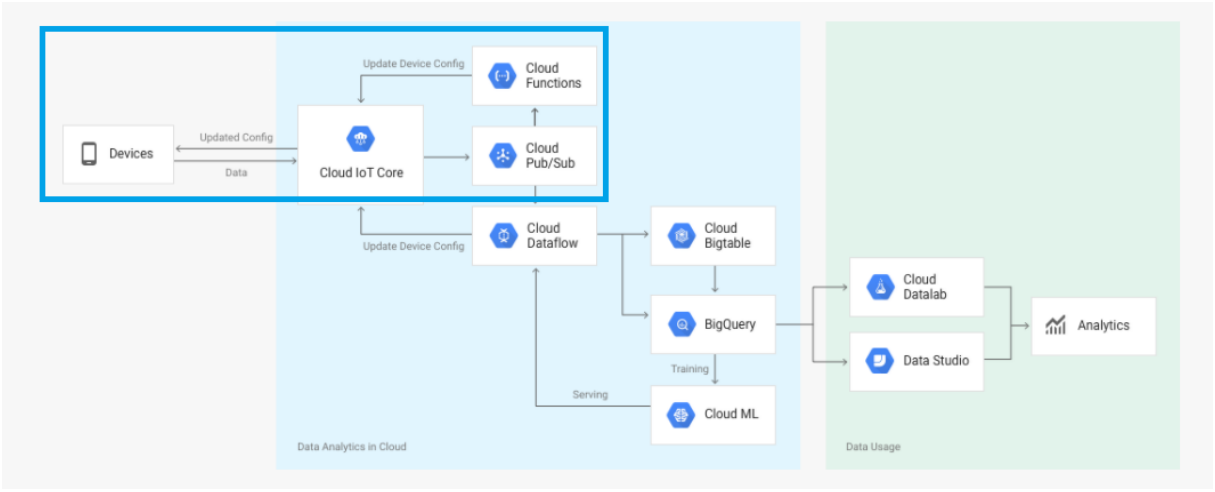
Figure 1. Google Cloud™ IoT Core ecosystem



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 on-line documentation as shown in Figure 2.

Figure 2. Google Cloud™ IoT Core diagram



Note: ©2020 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 Expansion Package demonstrates how to configure the IoT Core service of GCP in order to connect a device securely to the Google Cloud™ IoT Core service.

The device uses a private key (ECDSA) 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 the STM32L4 Series microcontrollers acting as end devices.

X-CUBE-GCP version 2.0.0 runs on the B-L475E-IOT01A Discovery kit. This platform supports Wi-Fi® connectivity with an on-board Inventek ISM43362 module

The X-CUBE-GCP Expansion Package contains the following components:

- Google Cloud™ IoT Device SDK for Embedded C
- mbedTLS
- Secure bootloader derived from the X-CUBE-SBSFU Expansion Package
- Network library, which is an abstraction interface for all TCP/IP connectivity
- Inventek ISM43362 Wi-Fi® driver for the B-L475E-IOT01A board
- Sensor drivers for the B-L475E-IOT01A board
- BSPs for the B-L475E-IOT01A board
- STM32L4 Series HAL
- Application examples

Software is provided as a zip archive containing the source code. The following integrated development environments are supported:

- IAR Systems - IAR Embedded Workbench® for Arm® (EWARM)
- Keil® - Microcontroller Development Kit (MDK-ARM)
- STMicroelectronics - STM32CubeIDE

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 Package for STM32Cube. Its main features and characteristics are:

- Fully compliant with the STM32Cube architecture
- Expands STM32Cube in order to enable the development of applications accessing and using the Google Cloud™ IoT Core of the 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 the 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.

- **Google Cloud™ IoT Device embedded C SDK**

  The Google Cloud™ IoT Device embedded C SDK is an open-source C library that connects IoT devices to Google Cloud™ IoT Core.

- **TCP/IP**

  The TCP/IP connection is handled by the Wi-Fi® module.

- **Secure Boot and Secure Firmware Update**
  - Secure bootloader: manages application verification and authentication at boot time
  - Secure Firmware Update: manages firmware update in a trusted secure way

  The secure bootloader and Secure Firmware Update solution is derived from the X-CUBE-SBSFU Expansion Package. Some elementary components of the X-CUBE-SBSFU Expansion Package are pre-integrated in the X-CUBE-GCP Expansion Package.

- **STM32 Network library**

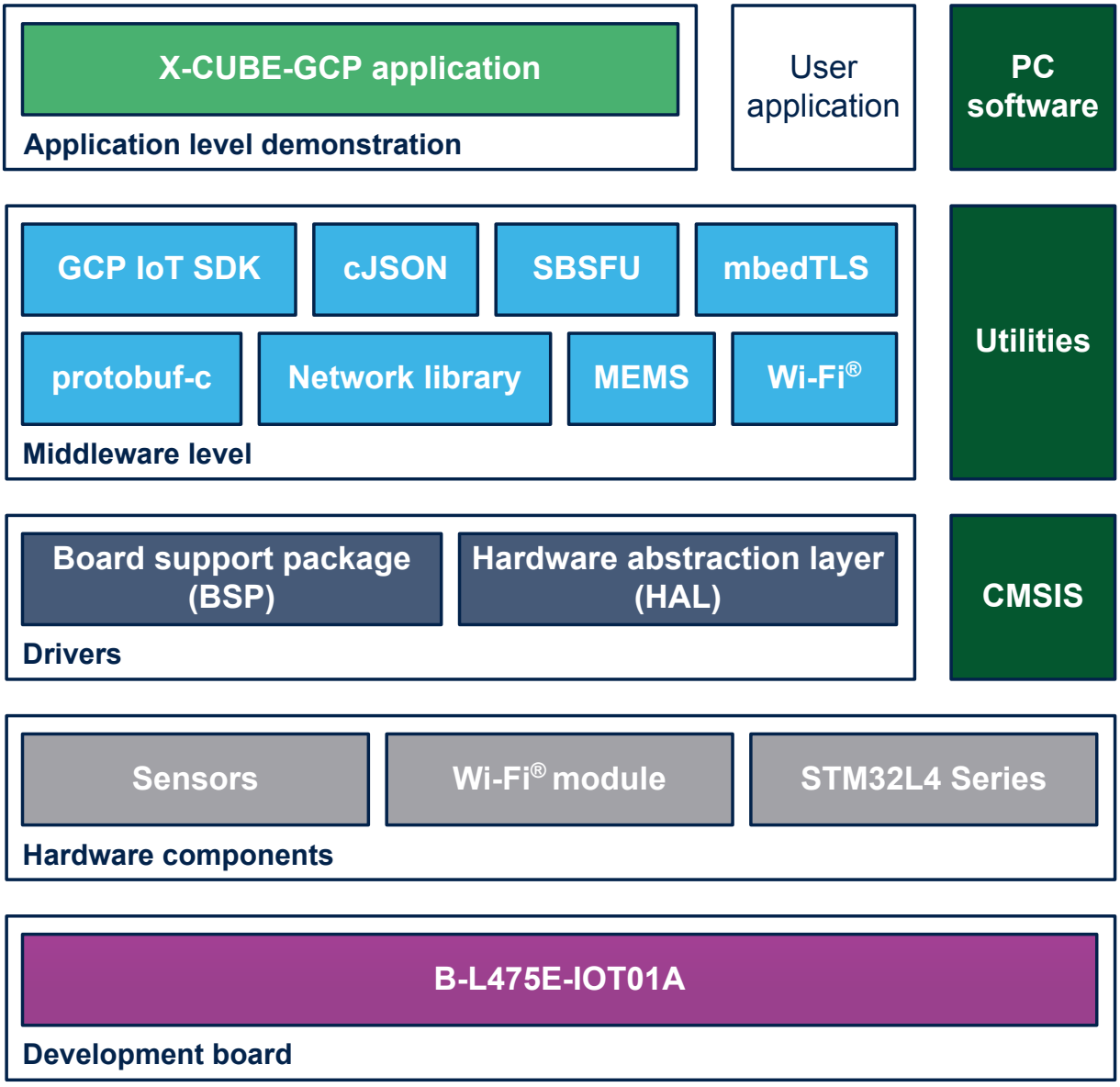
  The STM32 Network library provides an API to access network services on STM32 devices. It supports several network adapters and several protocols. This API is intended to be used in any STM32Cube application requiring network services.

- **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.

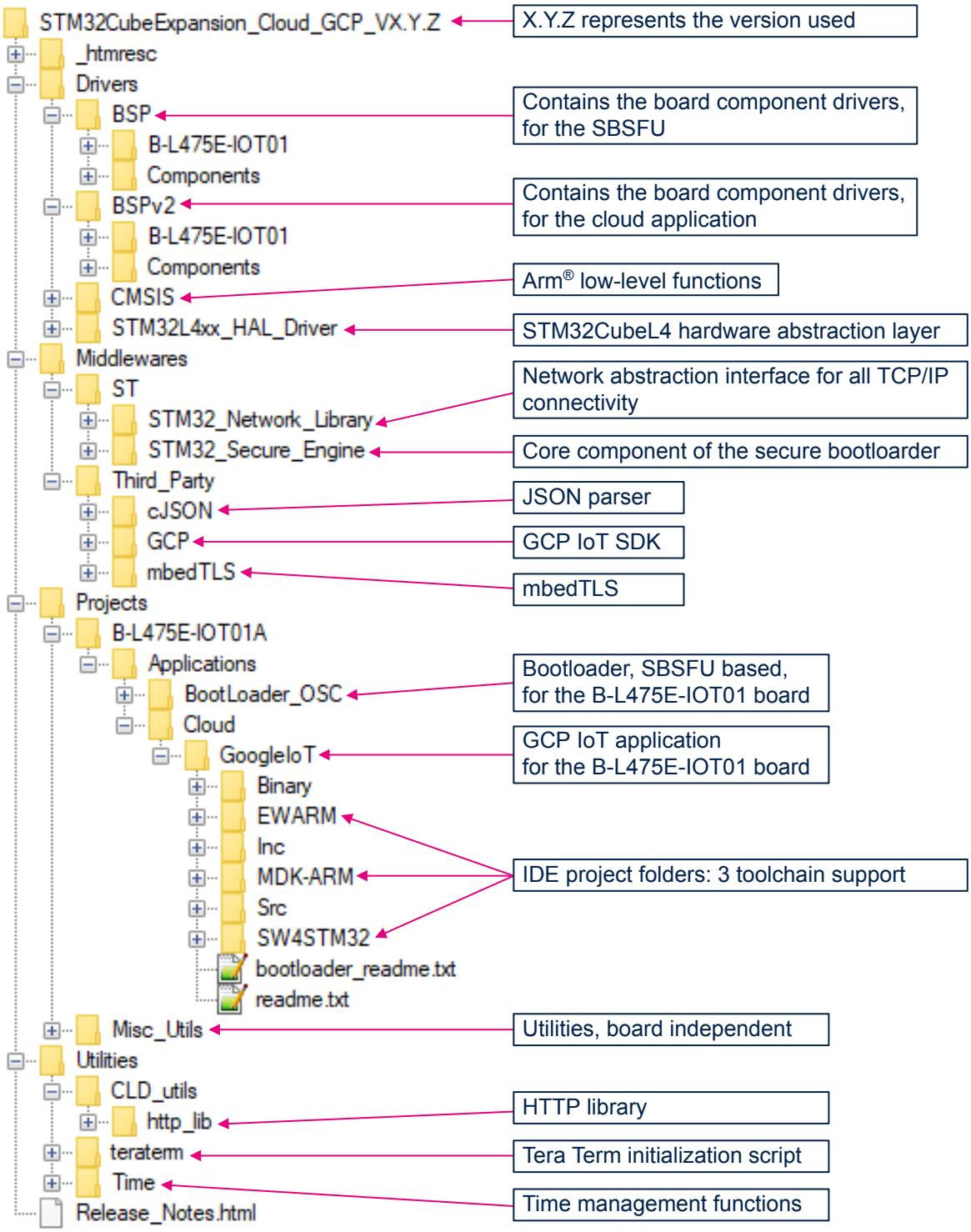
Figure 3. X-CUBE-GCP software architecture



3.3 Folder structure

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

Figure 4. **Project file structure**



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 the Network library.

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 the 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\BSPv2\<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. The X-CUBE-GCP Expansion Package uses file `mbedtls_config.h` for project configuration. This is implemented by setting the following `#` directives at the beginning of the source files:

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

The configuration file specifies the ciphers to integrate.

3.11 Secure bootloader

3.11.1 Overview

The pre-integrated bootloader allows the update of the user application (initially, the Google Cloud™ IoT sample application), adding new features, and correcting potential issues. This is performed in a secure way, which prevents any unauthorized update and access to confidential embedded data such as secret code and firmware encryption key.

The secure bootloader enforces hardware security mechanisms on STM32. It guarantees a unique entry point after reset, ensures immutable boot code, enforces security check and performs firmware image verification (authenticity and integrity) before execution.

It takes advantage of hardware protection mechanisms present in STM32 devices:

- RDP-L2: read data protection (disables external access, protects boot options)
- WRP/PCROP: write data protection (protects code and keys from Flash dump, protects trusted code)
- MPU: execution allowed in a chain of trust
- Firewall: protects Flash memory and RAM at runtime (secure enclave for critical operations)

The secure bootloader is a standalone executable. It is delivered in the package both as a pre-built executable file, and as source files, allowing advanced users to rebuild it with different settings.

Caution: The delivered pre-compiled bootloader does not enforce security, so that the user can still plug a debugger and debug the application. It must not be used in a production context.

Pre-compiled bootloader settings:

- Dual-image mode
- Enable local loader
- Secure IP turned off
- AES-GCM symmetric cryptography

The dual-image mode enables safe image update, with firmware image backup and rollback capabilities. The Flash memory is split in two areas named Slot #0 and Slot #1. The Slot #0 area contains the active firmware (firmware header + firmware). The Slot #1 area can be written a downloaded firmware (firmware header + encrypted firmware) to be decrypted and installed at next reboot. A specific Flash memory area is used to swap the content of Slot #0 and Slot #1 during the installation process at boot time. Using those two slots, firmware update supports a rollback mechanism if an error happens at first execution of a new installed firmware.

The bootloader integration is done in a seamless way, so that the user can keep using the IDE as used to, when programming the board or debugging.

By contrast to the usual build flow (compile/link/program/debug), specific pre- and post-build phases are added to the sample projects:

- The post-build phase combines the bootloader with the application, and overwrites the application executable so that it can be programmed and run as usual.
- The pre-build phase deletes the `.elf` file to force the link and post-build, even if the user project sources have not changed. It prevents post-build dependencies issues with STM32CubeIDE.

3.11.2 Application boot

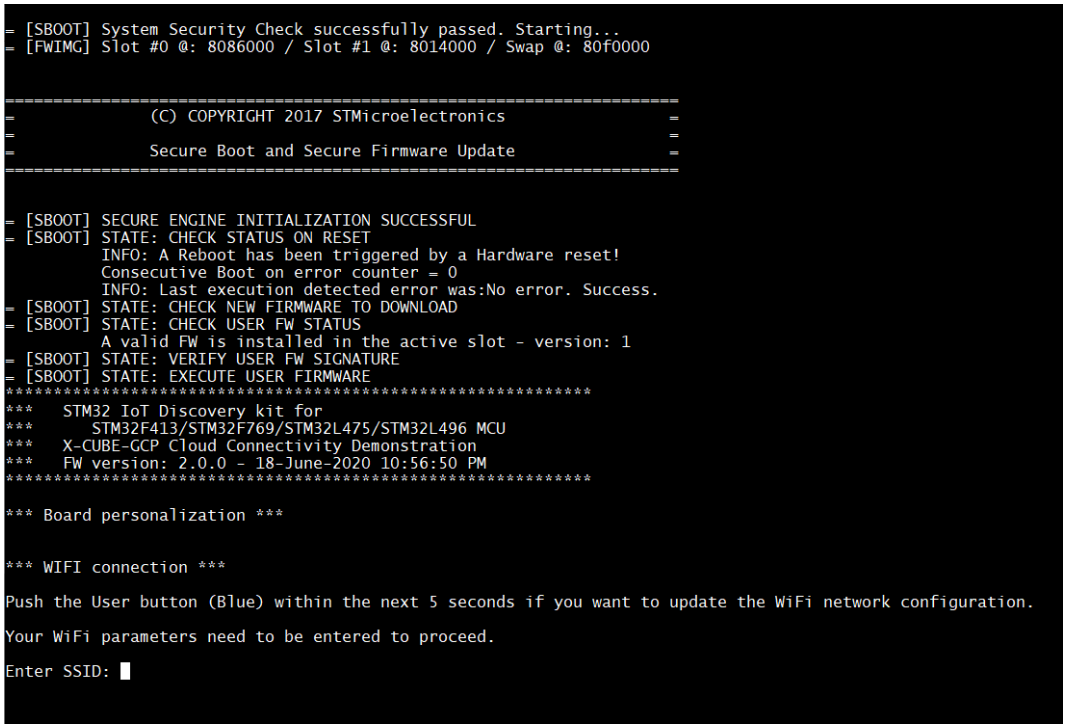
The bootloader runs first. It enforces security and checks in Slot #1 if new firmware must be installed.

- If no new firmware must be installed, it starts the already installed application
- If new firmware must be installed, it decrypts if needed, and checks it before installing and running it

Bootloader messages are prefixed with `[SBOOT]`.

In the screenshot in Figure 5, the bootloader does not find an application to install from Slot #1. Therefore, it checks the firmware image in Slot #0, and runs it.

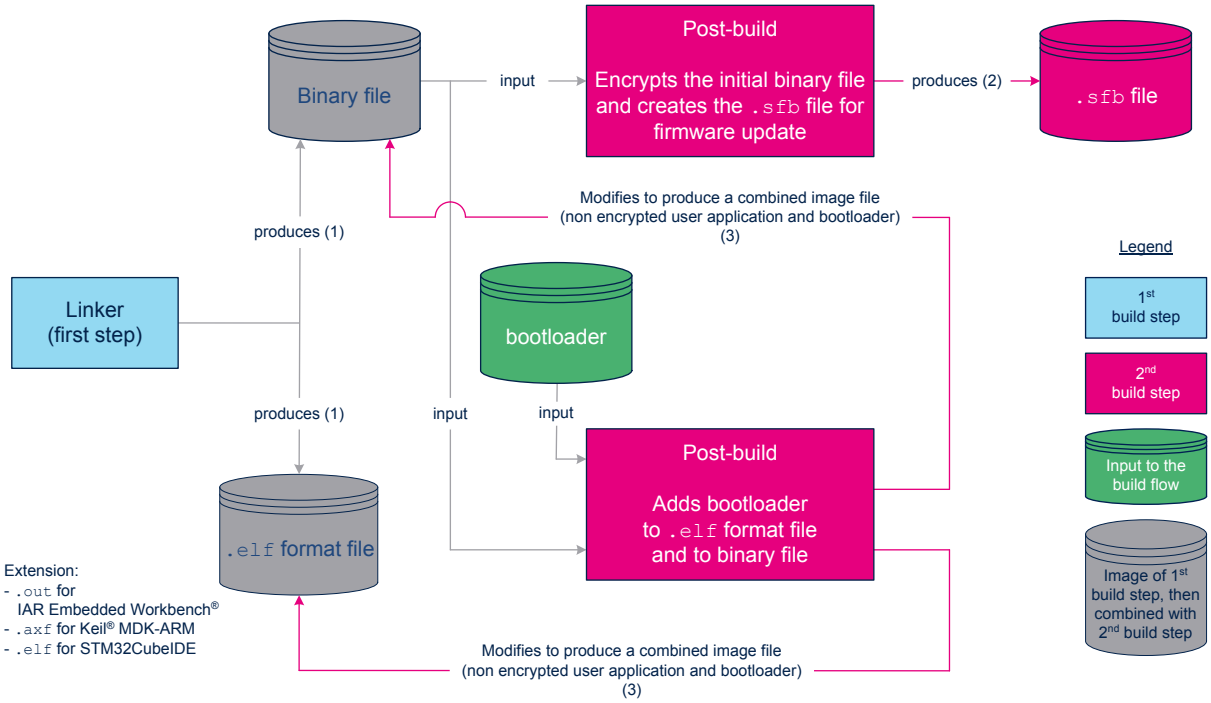
Figure 5. Application boot - console feedback



3.11.3 Building the whole firmware image

Figure 6 describes the flow for building binary images.

Figure 6. Image build flow



The source files are compiled and linked as usual, which produces a binary file and an `.elf` format file.

After the link, a post-build script is run.

A file with the `.sfb` extension is created, which contains the standalone user application, encrypted, and prefixed with meta data, ready for being downloaded and written at runtime to the firmware update slot.

The script overwrites the `.elf` and binary output files. It combines the bootloader and the user application.

Detail of output files built

The combined image file (binary and `.elf` format) contains the bootloader combined with the initial user application. The user application is placed in the executable slot in a non-encrypted form. The file location depends on the IDE:

- IAR Embedded Workbench®

  `Project/<board_name>/Applications/Cloud/GoogleIoT/EWARM/<board_name>/Exe/<board_name>.out`

  `Project/<board_name>/Applications/Cloud/GoogleIoT/EWARM/<board_name>/<board_name>_GoogleIoT.bin`

- Keil® MDK-ARM

  `Project/<board_name>/Applications/Cloud/GoogleIoT/MDK-ARM/<board_name>/Exe/<board_name>.axf`

  `Project/<board_name>/Applications/Cloud/GoogleIoT/MDK-ARM/<board_name>/Exe/<board_name>_GoogleIoT.bin`

- STM32CubeIDE

  `Project/<board_name>/Applications/Cloud/GoogleIoT/STM32CubeIDE/Release/<board_name>_GoogleIoT.elf`

  `Project/<board_name>/Applications/Cloud/GoogleIoT/STM32CubeIDE/Release/<board_name>_GoogleIoT.bin`

The binary `.sfb` file packs the firmware header and firmware image of the user application. This file is the one used when performing firmware update. Its location depends on the IDE:

- IAR Embedded Workbench®

  `Project/<board_name>/Applications/Cloud/GoogleIoT/EWARM/PostBuild/<board_name>_GoogleIoT.sfb`

- Keil® MDK-ARM

  `Project/<board_name>/Applications/Cloud/GoogleIoT/MDK-ARM/PostBuild/<board_name>.sfb`

- STM32CubeIDE

  `Project/<board_name>/Applications/Cloud/GoogleIoT/STM32CubeIDE/PostBuild/<board_name>_GoogleIoT.sfb`

The post-processing log file is useful to analyze the possible post-build issues. The most common issue is due to an incorrect path to the STM32CubeProgrammer (STM32CubeProg) tool. The file location depends on the IDE:

- IAR Embedded Workbench®

  `Project/<board_name>/Applications/Cloud/GoogleIoT/EWARM/output.txt`

- Keil® MDK-ARM

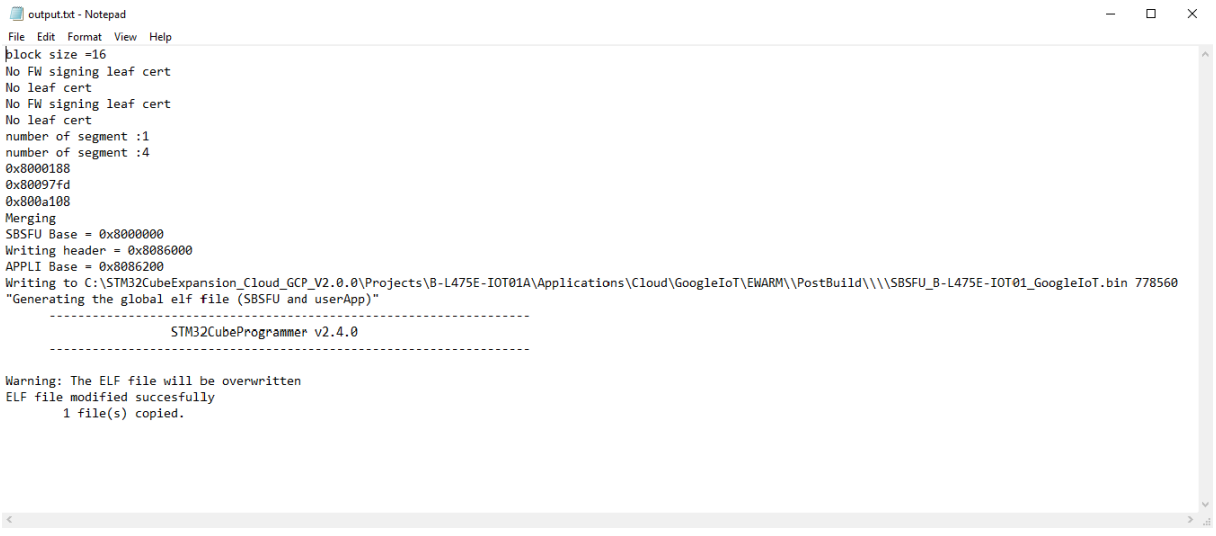
  `Project/<board_name>/Applications/Cloud/GoogleIoT/MDK-ARM/output.txt`

- STM32CubeIDE

  `Project/<board_name>/Applications/Cloud/GoogleIoT/STM32CubeIDE/output.txt`

Figure 7 shows the example of a post-processing log file.

Figure 7. Post-processing log file example



3.11.4 Rebuilding the bootloader

The bootloader is based on the technology of the X-CUBE-SBSFU Expansion Package. X-CUBE-GCP contains some sub-modules of X-CUBE-SBSFU:

- `Project/<board_name>/Applications/BootLoader_OSC/2_Images_SBSFU`
- `Project/<board_name>/Applications/BootLoader_OSC/2_Images_SECoreBin`
- `Project/<board_name>/Applications/BootLoader_OSC/Linker_Common`

Refer to the release note in the X-CUBE-AWS Expansion Package for information about the related X-CUBE-SBSFU version.

The X-CUBE-SBSFU sub-module usage is as follows:

SECorebin contains the Secure Engine Core sources, a protected environment, where all critical data and operations can be managed in a secure way.

SBSFU implements the Secure Boot and Secure Firmware Update with the support of the Secure Engine Core.

Linker_Common contains the linker files with the definition of the different Slot #0 and Slot #1 areas. The user application is linked against Slot #0 definition.

The original X-CUBE-SBSFU package is slightly modified to support the Google Cloud™ IoT sample application and to better integrate with the IDEs. Post-script templates are adapted to help with IDE integration.

Updating the bootloader implies to rebuild, **in this order**:

1. The Secure Engine library. The project is located in the `2_Images_SE_CoreBin` folder.
2. The Secure Boot / Secure Firmware Update executable. The result is called *"bootloader"* in this document. Depending on the IDE, it is located in:
   - IAR Embedded Workbench®
     `Project/<board_name>/Applications/Bootloader_OSC2_Images_SBSFU/EWARM/<boardname>Exe/Project.out`
   - Keil® MDK-ARM
     `Project/<board_name>/Applications/Bootloader_OSC2_Images_SBSFU/MDK-ARM/<boardname>_2_Images_SBSFU/Exe/SBSFU.axf`
   - STM32CubeIDE
     `Project/<board_name>/Applications/Bootloader_OSC2_Images_SBSFU/STM32CubeIDE/<board_name>_2_Images_SBSFU/Debug/SBSFU.elf`

Rebuilding the bootloader is required if the bootloader configuration or some linker script files are changed.

The configuration is defined by several files and is based on C preprocessor statements:

- `2_Images_SBSFU/SBSFU/App/app_sfu.h` for SBSFU settings
- `2_Images_SECoreBin/Inc/se_crypto_config.h` for the cryptography settings

The current bootloader is configured as follows:

- `SECBOOT_ECCDSA_WITH_AES128_CBC_SHA256`
- `SFU_DEBUG_MODE`
- `SECBOOT_DISABLE_SECURITY_IPS`

The secure firmware image is encrypted, but the hardware protections are not enforced. For instance, the JTAG link is on, so that debugging remains possible. Read the documentation of the X-CUBE-SBSFU Expansion Package for more information.

Programming the user application

From a user perspective, the application gets programmed as usual. Nevertheless, when security is enforced by the bootloader, it may be necessary to reprogram the Option Bytes to clear some protections and revert to RDP-L0 before reprogramming the Flash memory. This can be achieved with the STM32CubeProgrammer (STM32CubeProg) or STM32 ST-LINK Utility (STSW-LINK004) tools.

Debugging the application

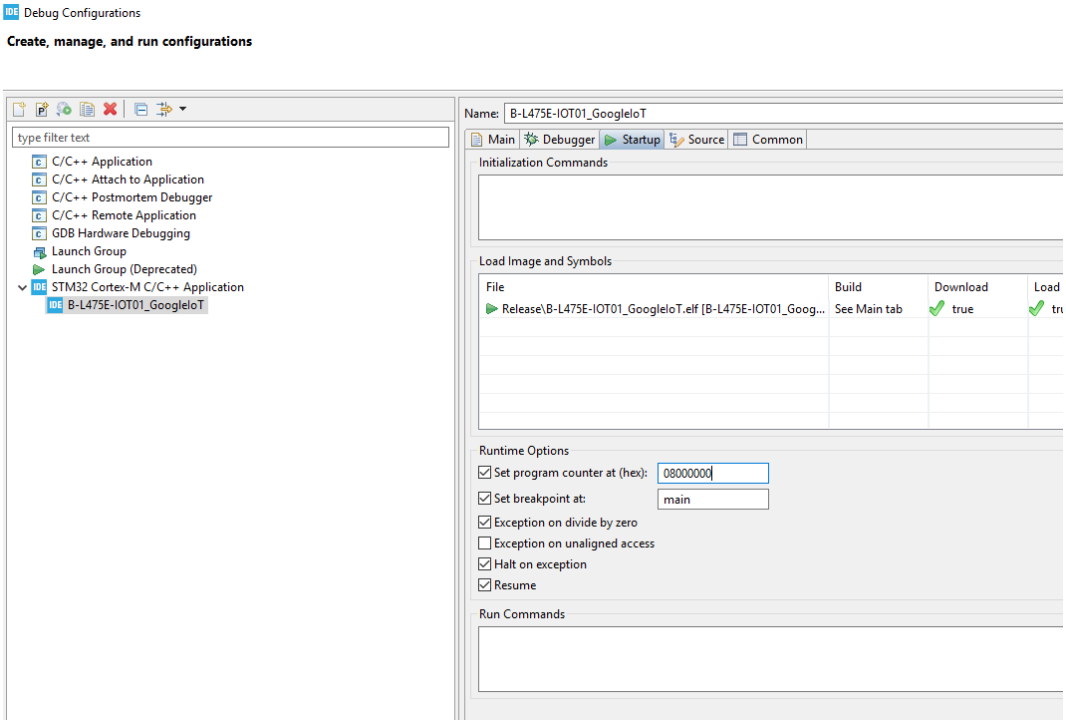
The debugger retrieves the debug information from the user application `.elf` file (the Google Cloud™ IoT sample application in the present case). It has no access to the bootloader symbols. The debugger places a breakpoint on the `main()` function of the user application.

Upon reset:

- In IAR Embedded Workbench® and Keil® MDK-ARM, the debugger starts the application at address 0x0800 0000 so that the bootloader is executed. For this purpose, IAR Embedded Workbench® is given an extra option: [**Debugger**]>[**extra options**]>[**command line option**] `--drv_vector_table_base=0x08000000`

- The STM32CubeIDE debugger does not start the application at address 0x0800 0000 but finds the address of the `vector_table` symbol in the debug information. As a result, the bootloader execution is skipped and the user application is run directly. A workaround is to push the reset button on the board and force a HW reset. The bootloader is then executed prior to the user application. The user application eventually reaches the`main()` function and stops at the breakpoint. The correct fix consists in changing the debug configuration option as shown in Figure 8 below. The program counter must be set to 0x0800 0000 for the application to boot correctly. This option is unfortunately saved in the Eclipse® workspace (and not in the project files) and cannot therefore be delivered as part of the present Expansion Package. This is described in Figure 8.

Figure 8. STM32CubeIDE debug configuration



3.11.5 Firmware update

The application offers to update firmware by pressing the blue user button (blue) and entering the URL for the firmware `.sfb` file. The `.sfb` file is generated during compilation by post-build tools. It is available under `Projects/<board>/Applications/Cloud/GoogleIoT/<Toolchain Directory>/PostBuild/`. The file must be hosted on an HTTP or HTTPS server with public access for reading.

In the case of Google Cloud™, it is possible to use a Google® storage account. Make sure the file is public (readable from the Internet without password).

Note that Google Cloud™ suggests to use `https://storage.cloud.google.com/` URLs. Such URLs require to log in with a Google® account. Prefer instead URLs as `https://storage.googleapis.com/`.

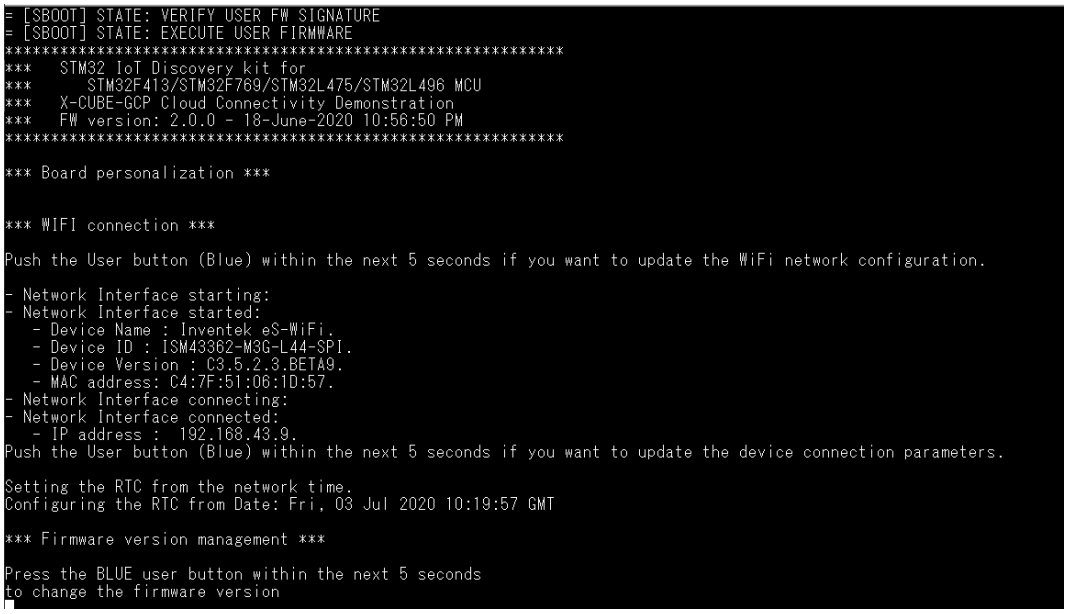
Example:

`https://storage.googleapis.com/project-name-999999999.appspot.com/firmware.sfb`

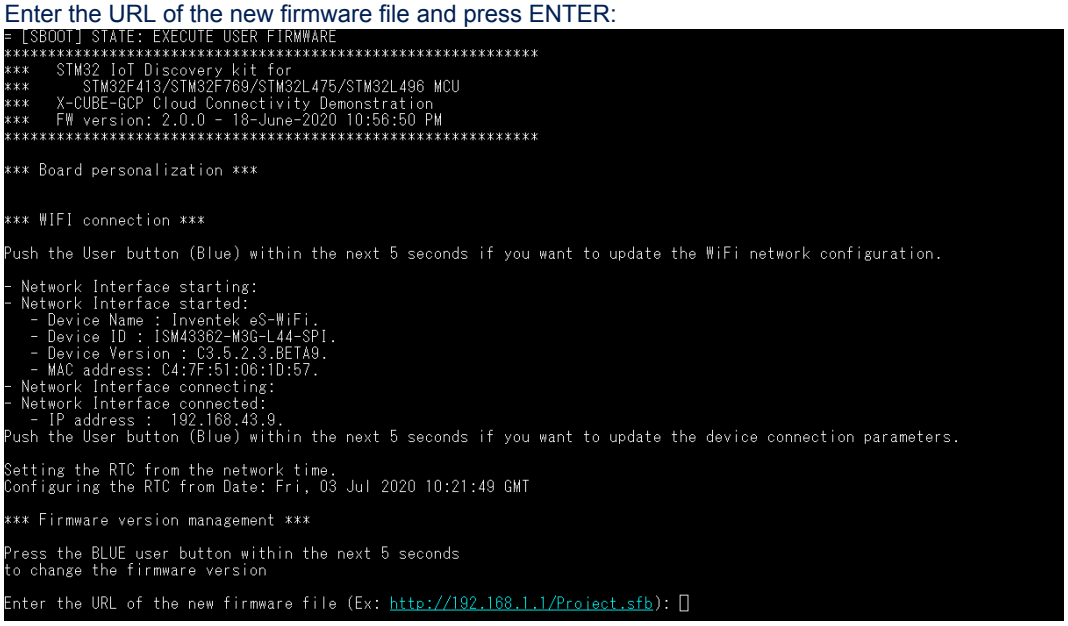
If using an HTTPS server, its certificate or Root CA must be configured in the application during the configuration phase (root CA config). If the file is downloaded successfully, it is written in the STM32 Flash memory. Then the board reboots and the bootloader updates firmware.

The step-by-step procedure below with screenshots shows how to perform a firmware update:

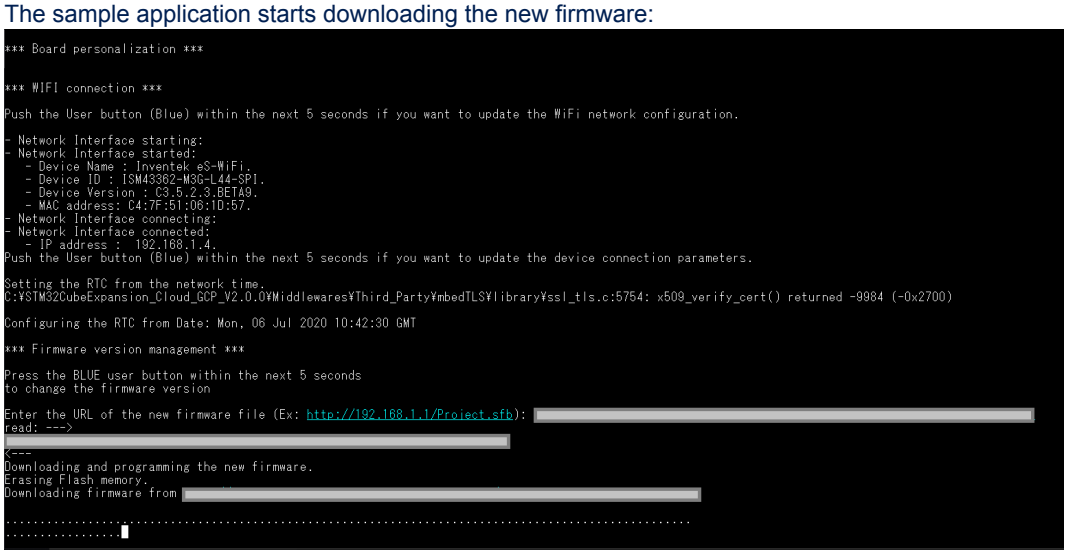
1. At init, wait until the sample application requests the user to "Press the Blue button […] to change the firmware version" as shown below:



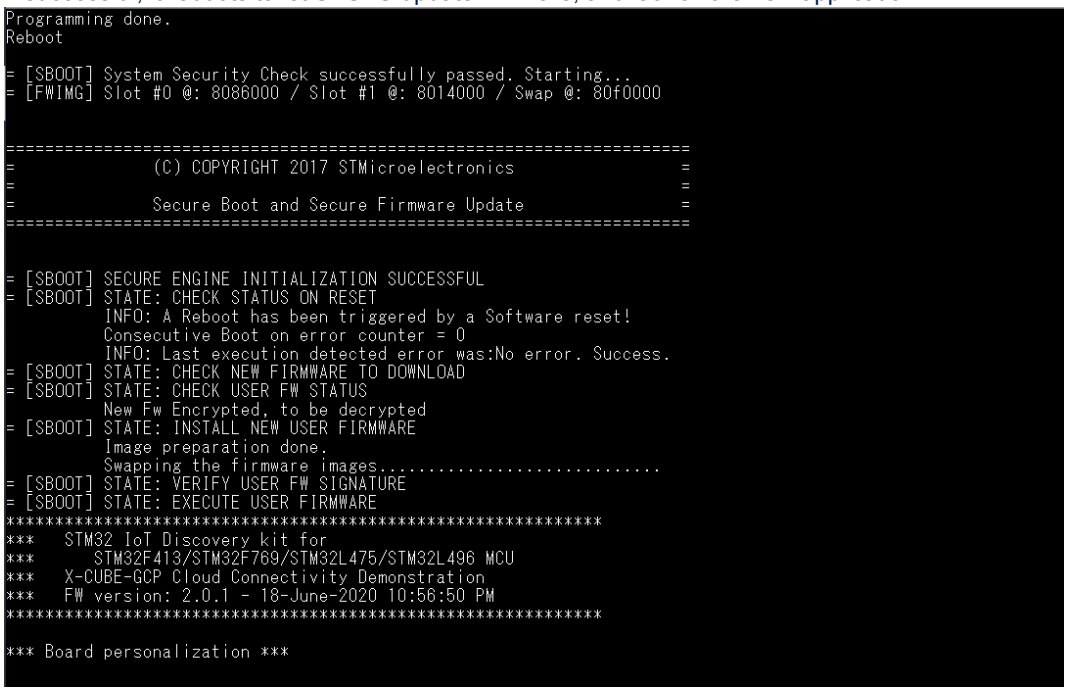
2. Enter the URL of the new firmware file and press ENTER:



3. The sample application starts downloading the new firmware:



4. If successful, it reboots to let SBSFU update firmware, and launch the new application:



Note about HTTP download

To minimize the RAM footprint, the firmware file is downloaded through ranged HTTP requests by chunks of 1 Kbyte that are immediately written to their destination Flash page.

As a result, the HTTP server must support ranged requests, as required by HTTP/1.1.

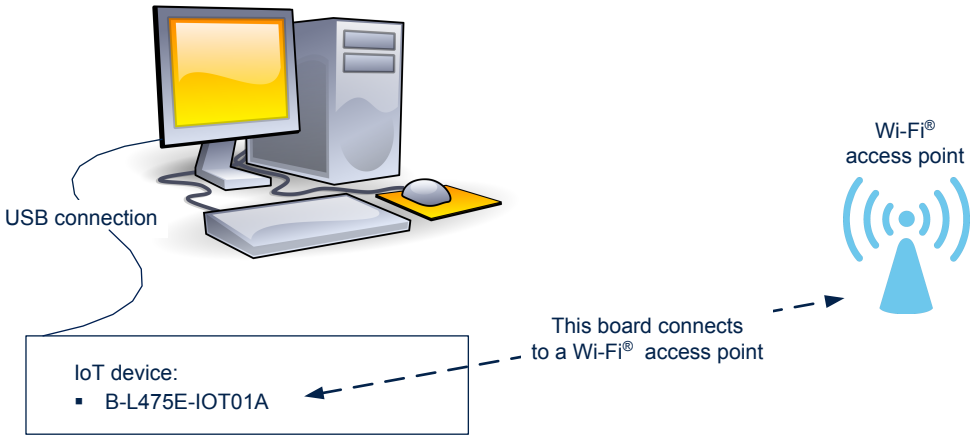
4 Hardware and software environment setup

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:

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

The B-L475E-IOT01A board must be connected to a Wi-Fi® access point as illustrated in Figure 9.

Figure 9. Hardware and software setup environment



The prerequisite for running the examples is:

- 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.
- 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` 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 the 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 of 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**

  The application first configures the board Wi-Fi® connectivity 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. The key format supported in this application is ES256.

- **Sending / getting Device State**

  Once securely connected, the application sends an updated Device State to GCP IoT core. This allows getting device information from the 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`.

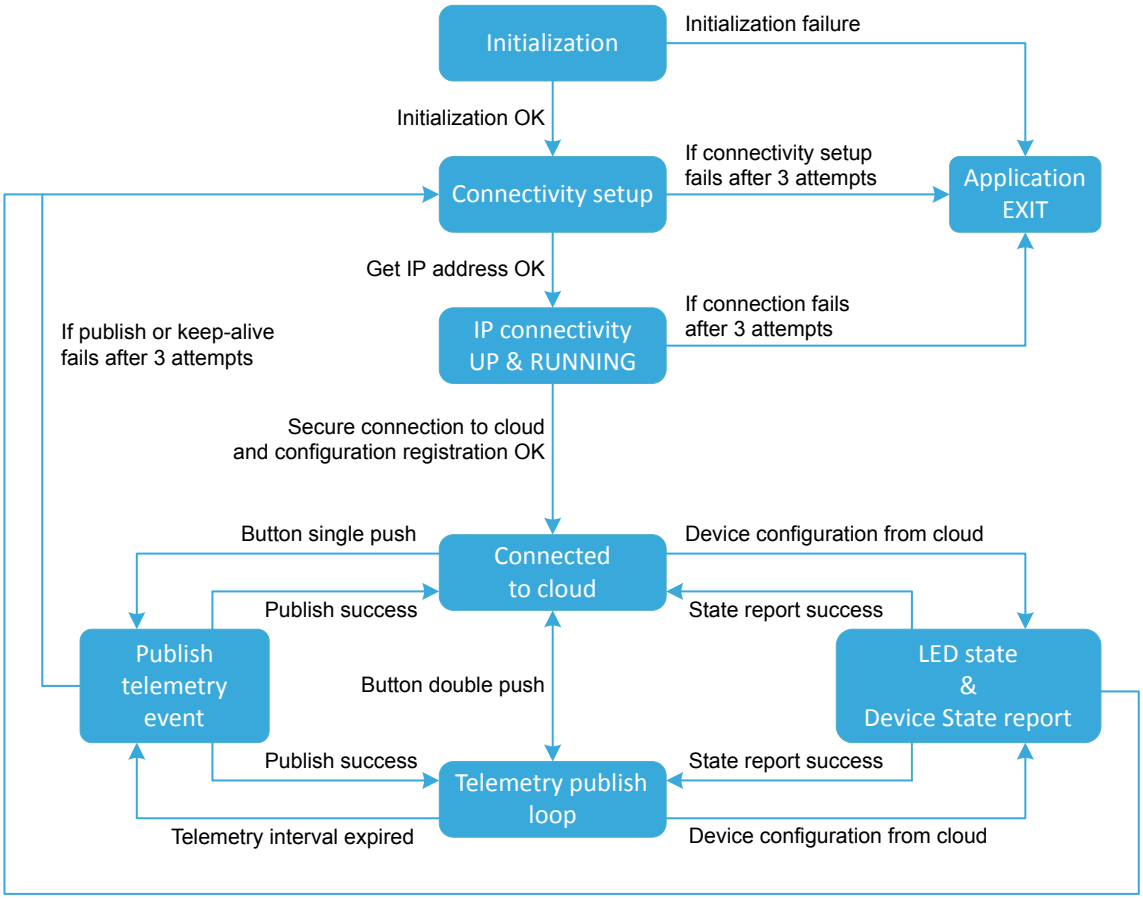
  As explained in GCP on-line documentation, a device can publish a telemetry event by issuing a PUBLISH message through the MQTT connection. The Quality of Service used in the application example is QOS1 (delivered at least once). Messages must be published to an MQTT topic in the following format: `/devices/{device-id}/events`

  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 10.

Figure 10. Application state machine



5.2 GCP and IoT Core account setup

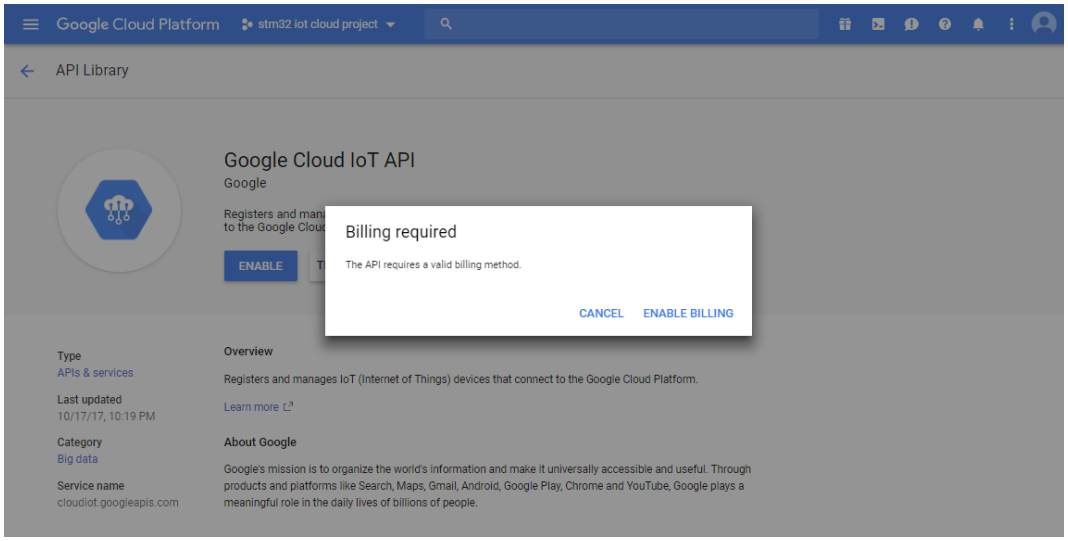
A Google Cloud Platform™ account is needed for using Google Cloud™ IoT Core services. The user must create an account before proceeding further and create or select a GCP project.

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

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

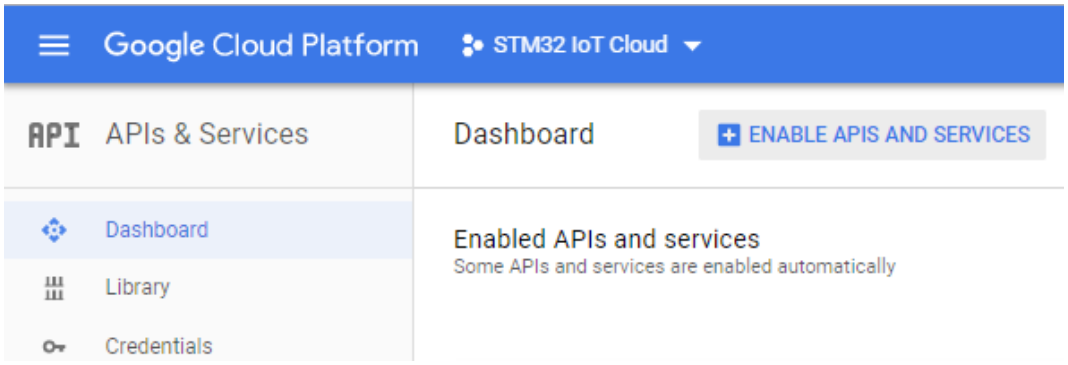
1. Enable billing for the project as shown in Figure 11. If billing is not enabled for his project, the user cannot proceed further with the account creation remaining steps.

Figure 11. Billing required



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 12.

Figure 12. Enable APIs menu



3. Use *ENABLE APIS AND SERVICES* and enable at least `Google Cloud IoT API` and `Cloud Pub/Sub API` as shown in Figure 13. Enable Google Cloud IoT API and Figure 14. Enable Cloud Pub/Sub API.

Figure 13. Enable Google Cloud IoT API

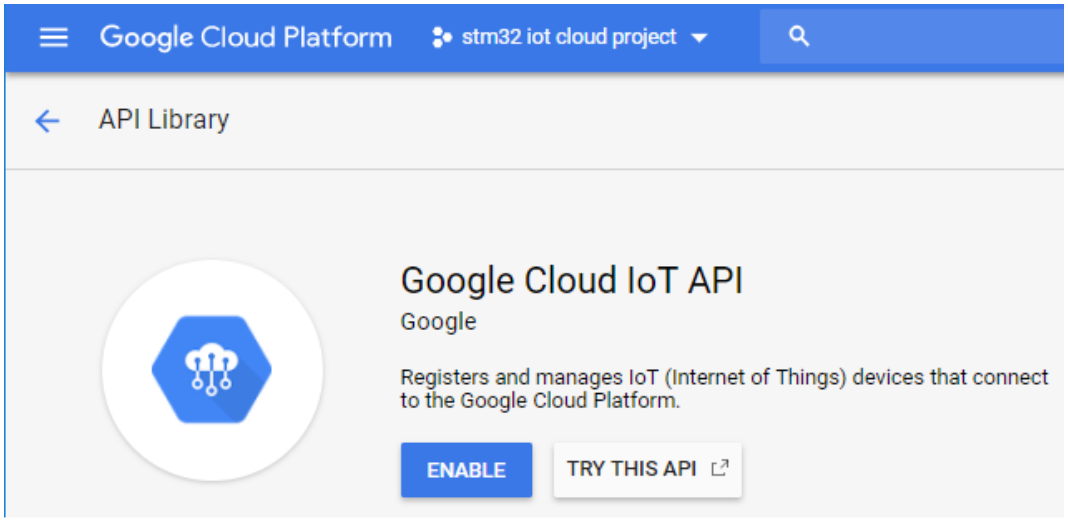
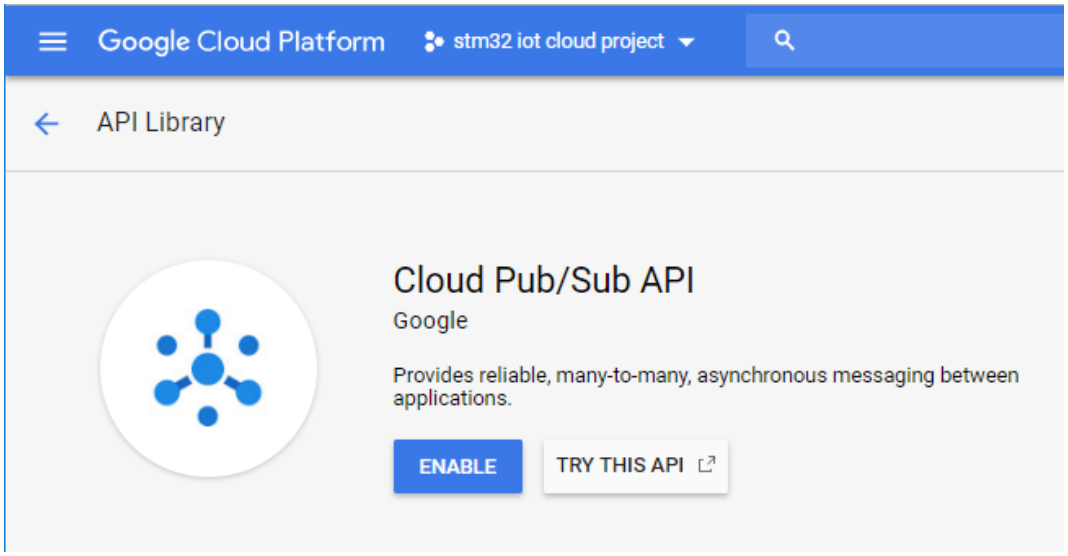


Figure 14. Enable Cloud Pub/Sub API



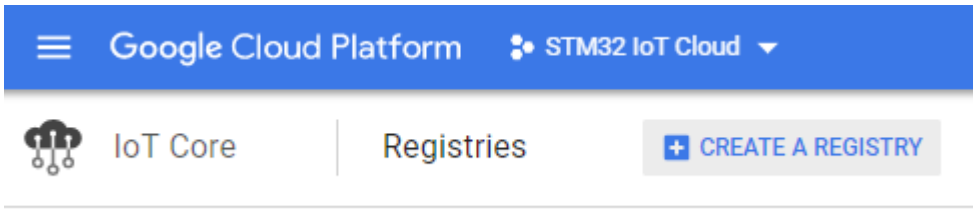
5.3 Device creation on GCP IoT core

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:

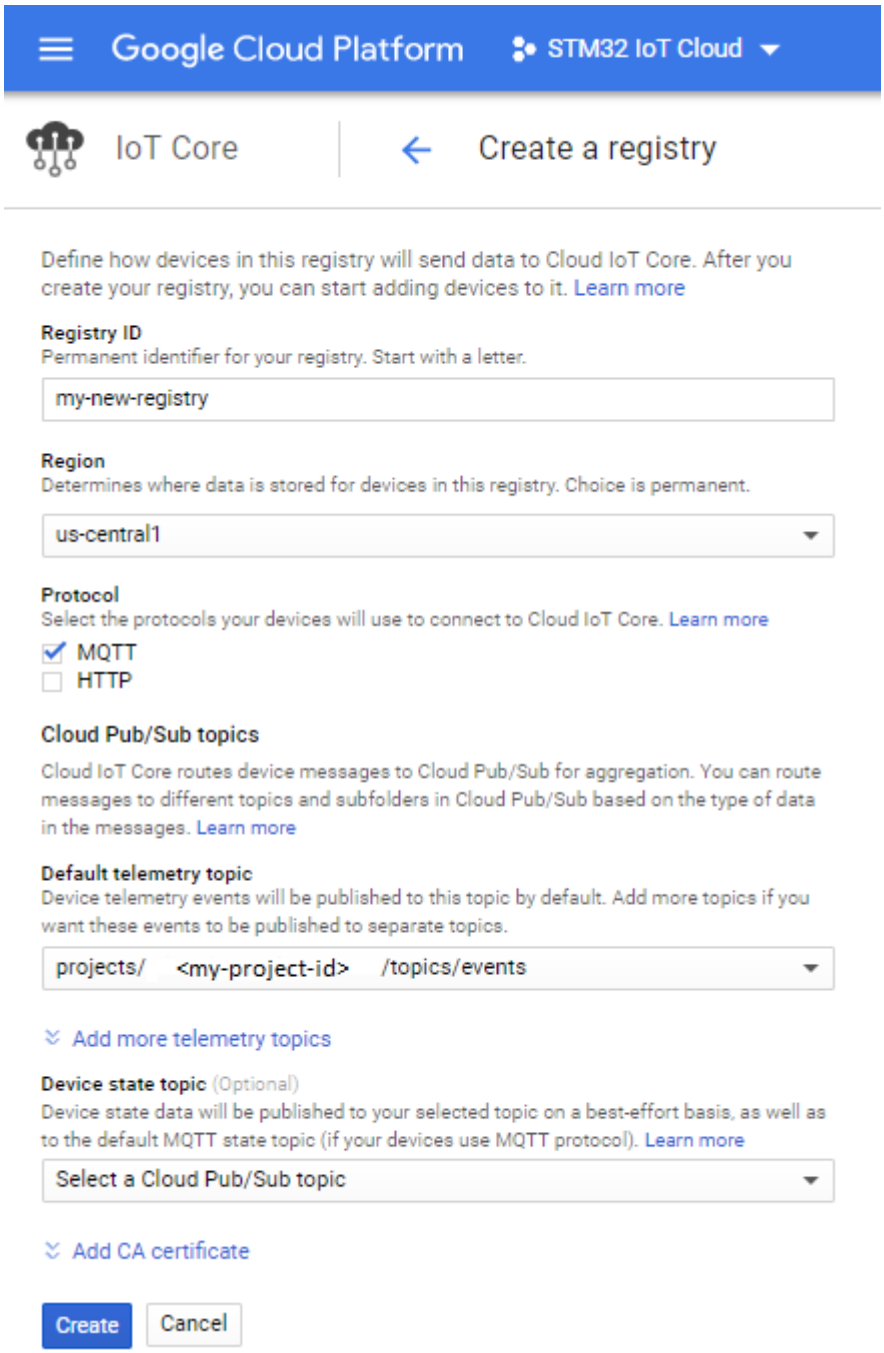
1. In Google Cloud™ IoT Core, create a new device registry, which is defined as `{registry-id}` in the connection string, as shown in Figure 15.

Figure 15. 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 16.

Figure 16. GCP 'Create a registry' settings



3. Add a new device in the newly created registry

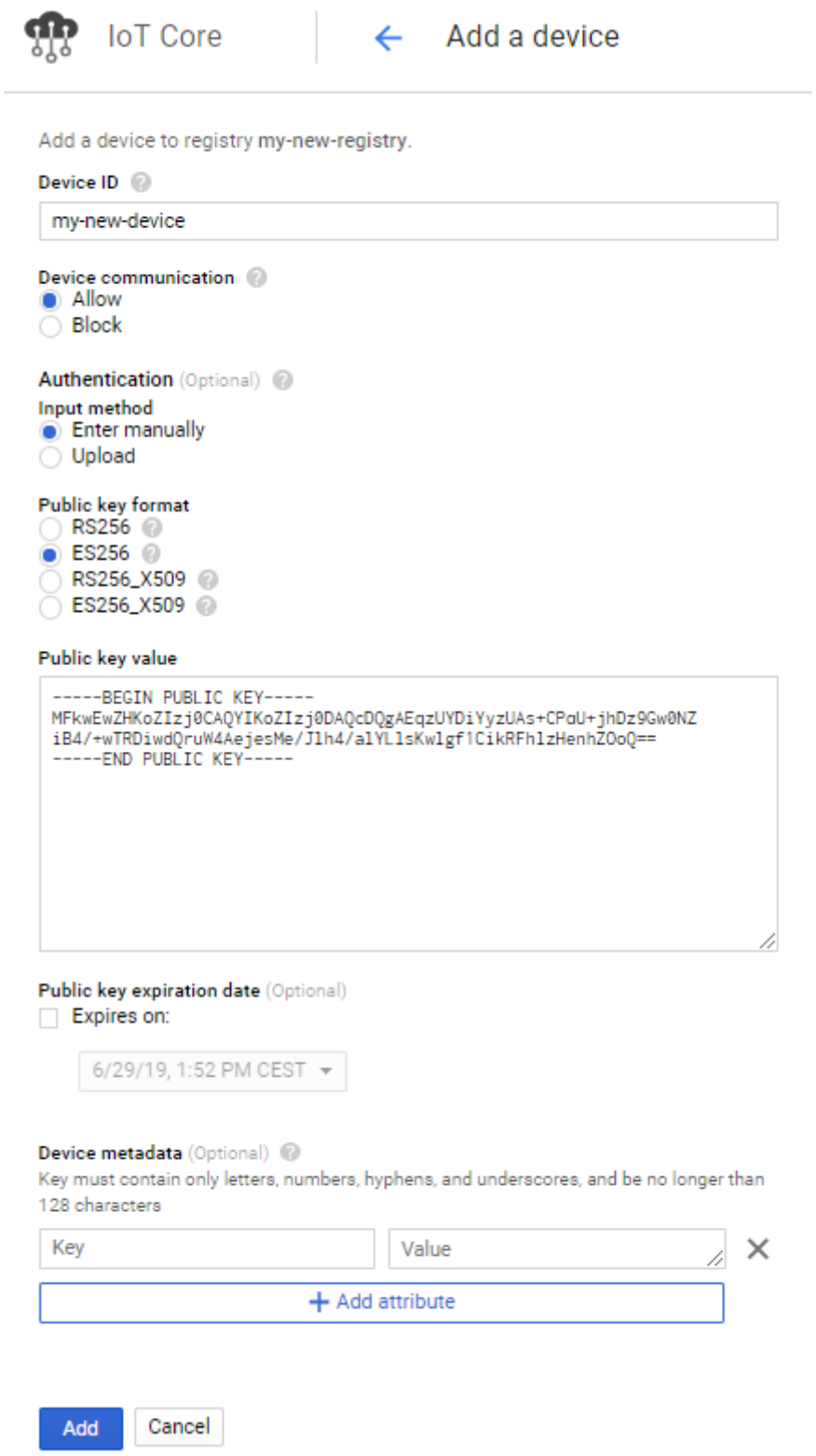
   Define a Device ID, which is `{device-id}` in the connection string.

   Use *"openssl"* for creating the public and private keys of type ES256 as described in the *"Creating Public/ Private Key Pairs"* documentation of GCP:

```
> 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 is passed to the device through console interface. The device creation is illustrated in Figure 17. GCP 'Create a device' settings.

Figure 17. **GCP 'Create a device' settings**



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

5.4 Application build and programming

Important:
Before opening the project with any toolchain, 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 toolchains.

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.
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 `Middlewares\Third_Party\GCP\samples\STM32Cube\globalsign_usertrust.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 *"Usertrust"* certificate).

   b. For GCP, in order to authenticate the Cloud server. Depending on the server, the `globalsign_usertrust.pem` may need to be updated based on Google Cloud™ list of supported CAs.
4. Set the device Private Key

   Copy-paste the content of the private key `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

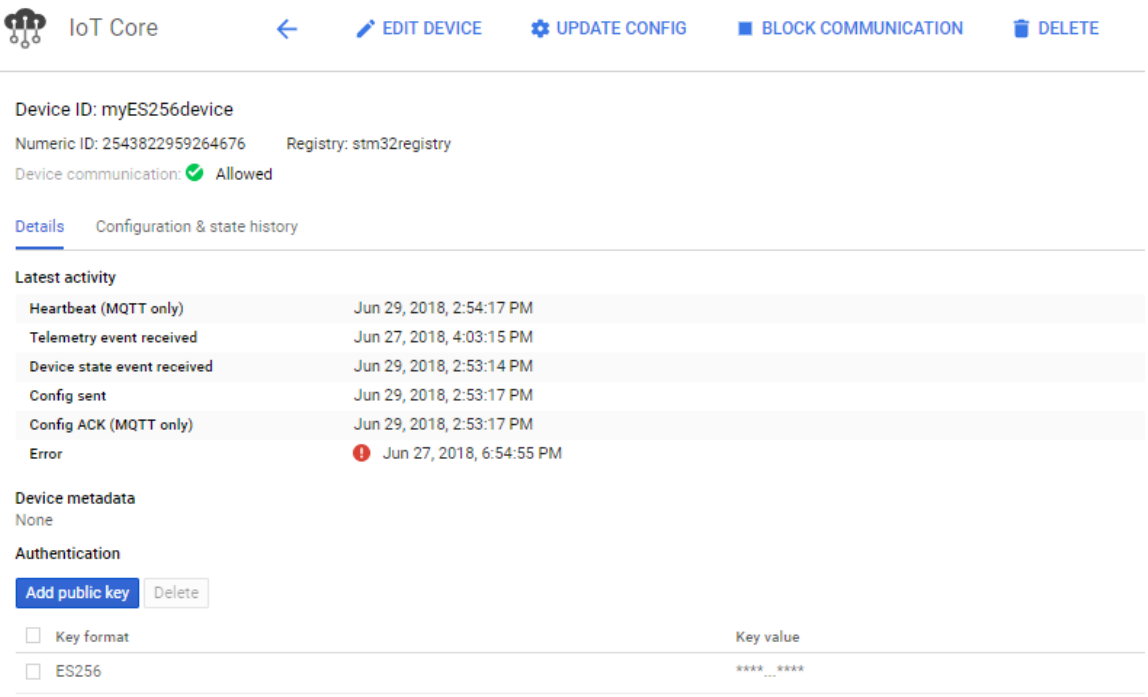
1. **RTC configuration**

   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 18. IoT core device console.

Figure 18. IoT core device console



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

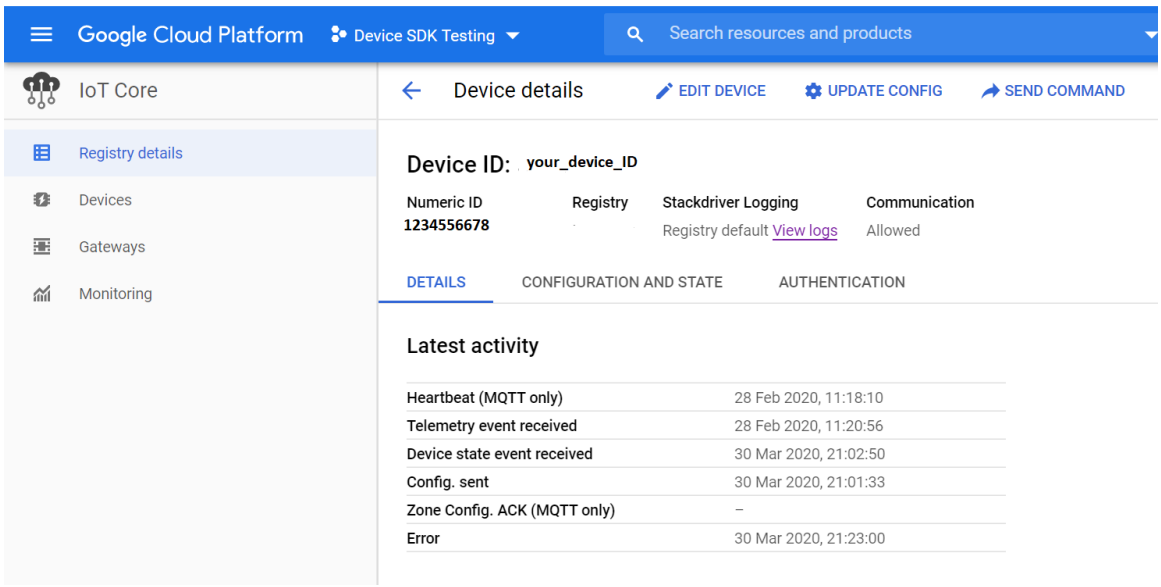
– At a periodic user-defined telemetry intervals if the user button is double pressed. The telemetry interval can be modified through the Device configuration.

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**

Cloud-to-device interaction is available from the GCP console using the *"UPDATE CONFIG"* menu options as shown in Figure 19.

Figure 19. IoT core device configuration



Both interactions enable sending commands from GCP IoT Core service to device using simple JSON string format. On firmware application side, the supported commands are:

a. LED control command:

```
{ "LedOn": true }
```

or

```
{ "LedOn": false }
```

b. Change of the telemetry publication interval:

```
{ "TelemetryInterval": <number of seconds> }
```

c. Reboot command:

```
{ "Reboot": true }
```

Any combination of above three commands can also be used, such as:

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

Note:

a. Both command and configuration support the same JSON string format and have identical effects, but the effect of command is immediate while configuration is applied each time the device is turned on. Therefore, the effect of command is lost if the device is switched off when the command is sent.

b. 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 as 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 the 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**].

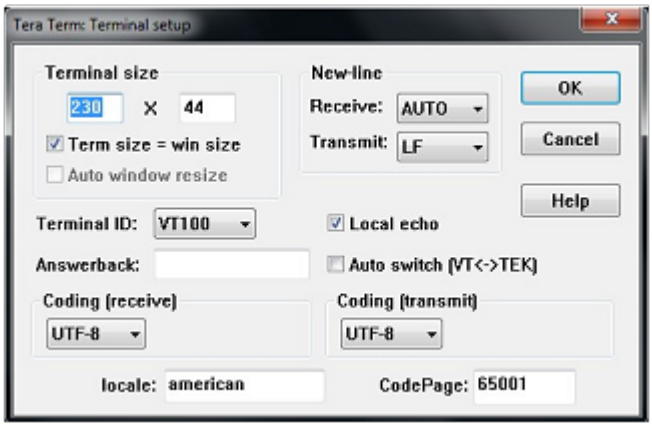
Note: The information provided below can be used to configure the UART terminal as an alternative to using the Tera Term initialization script.

Terminal setup

Terminal setup is illustrated in Figure 20. 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 20. Terminal setup



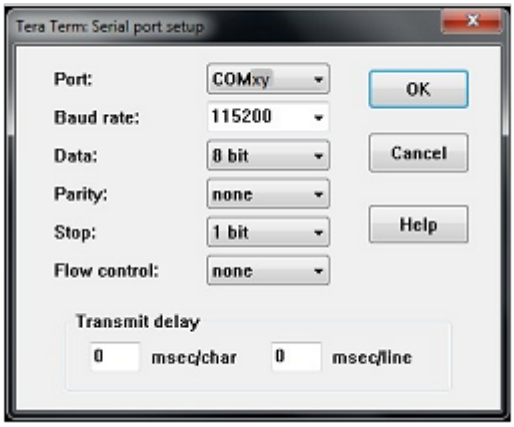
Serial port setup

The serial port must be configured with:

- COM port number
- 115200 baud rate
- 8-bit data
- No parity
- 1 stop bit
- No flow control

Serial port setup is illustrated in Figure 21.

Figure 21. **Serial port setup**

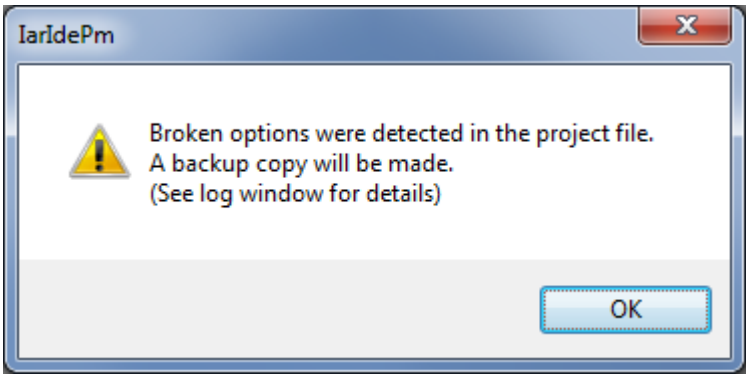


Once the UART terminal and the serial port are set up, press the board reset button (black). Follow the indications on the UART terminal to upload Wi-Fi® and cloud configuration data. These data remain in the Flash memory and are reused the next time the board boots.

7 Frequently asked questions

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

Figure 22. Pop-up when the IAR Embedded Workbench® version is not compatible with the one used for X-CUBE-GCP



A: It is very likely that the IAR Embedded Workbench® IDE version is older than the one used to develop the X-CUBE-GCP Expansion 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 Embedded Workbench® IDE version must 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 `publishTelemetryData()` or `GcpIoT_publishDeviceState()` respectively is needed in the in file `google_iot.c`.

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

A: Things that must be checked are:

1. Verify that the `project-id`, `registry-id` and `device-id` are correctly spelled and submitted in the 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.
24-Aug-2020	2	Focused the entire document on Wi-Fi® connectivity with the B-L475E-IOT01A Discovery kit. Introduced the use of the Secure Boot and Secure Firmware Update, and Network library. Updated: • Introduction and General description • Architecture and Folder structure • Hardware and software environment setup • *Cloud-to-device connection* in Application runtime Added: • Secure bootloader

Contents

List of tables

List of figures