UM2178
User manual

Getting started with X-CUBE-AWS STM32Cube Expansion Package for Amazon Web Services® IoT

Introduction

This user manual describes the content of the X-CUBE-AWS STM32Cube Expansion Package for AWS (Amazon Web Services®) IoT (Internet of Things).

The Amazon Web Services® Internet of Things service enables secure, bidirectional communication between IoT devices and the cloud over MQTT, HTTP and WebSockets.

The X-CUBE-AWS STM32Cube Expansion Package for AWS IoT provides application examples that connect and subscribe to the AWS IoT service via MQTT in order to receive information and publish data.

X-CUBE-AWS is available for the B-L475E-IOT01A, 32F413HDISCOVERY and 32F769IDISCOVERY Discovery kits, and for the P-L496G-CELL02 Discovery pack.

The X-CUBE-AWS features include:

- Ready-to-run firmware example using Wi-Fi®, cellular, and Ethernet connectivity to support quick evaluation and development of IoT cloud applications
- Interface to configure the board to connect to the AWS IoT
- AWS IoT connection, subscribe, publish, jobs
- Specific features on the B-L475E-IOT01A board:
  Measurement of humidity, temperature, 3-axis magnetic data, 3D acceleration, 3D gyroscope data, atmospheric pressure and proximity
- Secure Boot and Secure Firmware Update over HTTP
- Connection to the ST-AWS-Dashboard

Contents

List of tables

List of figures

1 General information

The X-CUBE-AWS Expansion Package runs on STM32 32-bit microcontrollers based on the Arm®[a] Cortex®-M processor.

1.1 Acronyms

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
AWS	Amazon Web Services®(1)
BSP	Board support package
CA	Certificate authority
HAL	Hardware abstraction layer
IAM	Identity and access management
IDE	Integrated development environment
IoT	Internet of Things
JSON	JavaScript object notation
LED	Light-emitting diode
NVM	Non-volatile memory
RAM	Random access memory
RFU	Remote firmware update
ROM	Read-only memory
RTC	Real-time clock
SB	Secure Boot
SFU	Secure Firmware Update

1. Amazon is a trademark of Amazon in the United States and/or other countries.

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

1.2 References

Table 2 presents STMicroelectronics documents providing complementary information about the topics presented in this user manual. These documents are available from STMicroelectronics web site at *www.st.com*.

Table 2. Reference documents

Idendifier	Title
UM2567	*Getting started with the X-CUBE-CELLULAR cellular connectivity Expansion Package for STM32Cube* user manual.
UM2426	*X-CUBE-CELLULAR cellular connectivity Expansion Package for STM32Cube* user manual.
UM2262	*Getting started with the X-CUBE-SBSFU STM32Cube Expansion Package* user manual.
AN5056	*Integration guide for the X-CUBE-SBSFU STM32Cube Expansion Package* application note.

2 Amazon Web Services® IoT

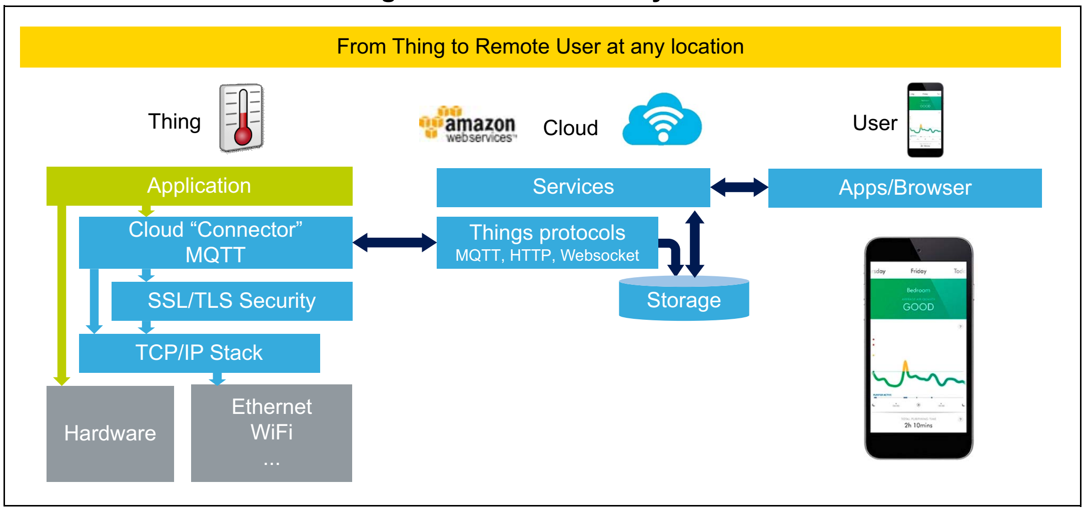
This section introduces AWS IoT.

AWS provide X.509 digital certificates to authenticate the IoT devices. Once a certificate is provisioned and activated, it can be installed on the device. The device uses this certificate to authenticate itself and send all the requests to AWS via the MQTT protocol.

AWS provide services such as database management, analytics, messaging and mobile services, among others. A user can connect to the cloud with a smartphone or a personal computer and access the information at any time and from any location.

This AWS IoT ecosystem is presented in *Figure 1*.

Figure 1. AWS IoT ecosystem



2.1 AWS IoT documentation

Amazon Web Services® (AWS) provide on-demand computing resources and services in the cloud. AWS IoT Core™ is the main service used in the scope of the application examples presented in this document.

General documentation, including a developer guide, is available at Amazon's AWS website. This documentation is particularly relevant to the use of the X-CUBE-AWS code.

Section 2.2 provides a rough overview of how to get AWS credentials. Nevertheless, the information presented in this document cannot substitute for the Amazon™ information, which is the reference.

2.2 AWS security credential creation

An AWS account is needed for the creation of AWS security credentials. Signing in at the AWS Amazon™ website is requested for getting these credentials.

2.2.1 Recommended way to create AWS IoT security credentials

AWS IAM (identity and access management) is a web service that helps to securely control users' access to AWS resources.

The IAM service is used for the control of:

- Authentication: to define who can use AWS resources
- Authorization: to define which resources can be used and in what ways

The recommended way to work with AWS is to:

1. Use the AWS identity and access management (IAM) service that allows the restriction of user rights
2. Create a group, for example *IoTDev*. The *AWSIoTFullAccess* existing policy can be attached to this group
3. Create users. In *Access type*, check *Programmatic access* and *AWS Management Console Access*
4. Assign the users to the group that has been created
5. Should a user be granted the *AmazonS3FullAccess* rights, he will be able to run the job examples described in this document
6. Then communicate the user name, password, access key ID, secret access key and console login link to the users. They are now able to create their things, certificates and policies. They must attach the thing and policy to the certificates
7. Each user must individually go through the following steps:
   - Log on to the AWS console
   - Select the closest server from his location with the top-right pull-down menu
   - Select the *IoT Core* service
   - Using the *Secure* section, create his policy to define his device rights with respect to AWS IoT Core™
     Refer to *Note: The default policy used for our application software* below for a policy example.
   - Using the *Manage* > *Things* section, create a single thing
   - Create his one-click certificate (recommended)
   - Download the device certificate, public and private key
   - Activate the certificate
   - Attach the policy previously created
   - Eventually register the thing

The artifacts that are needed for programming the board are:

- Certification authority certificate
  The *Amazon1_Comodo_Baltimore.crt* certificate is provided within the package under*./Middlewares/Third_Party/AWS/certs*
- Thing name
- Thing certificate
- Thing private key
- Endpoint. This is the hostname of the MQTT broker; it looks like: a27xxxx.iot.<region>.amazonaws.com. It is available in the *Settings* section.

Note: The default policy used for our application software

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:Publish",
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "iot:Subscribe",
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "iot:Receive",
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Connect"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

2.2.2 Quickest way to create AWS IoT security credentials (not the preferred method)

It is possible to use the AWS main account to create certificates, but, if this method is chosen, the restriction of the AWS services is not managed.

If the AWS IoT security credentials are created this way, follow the user steps described in *Section 2.2.1* and skip the IAM user creation, meaning starting from the step *7* listed in *Section 2.2.1*, but with the AWS main account.

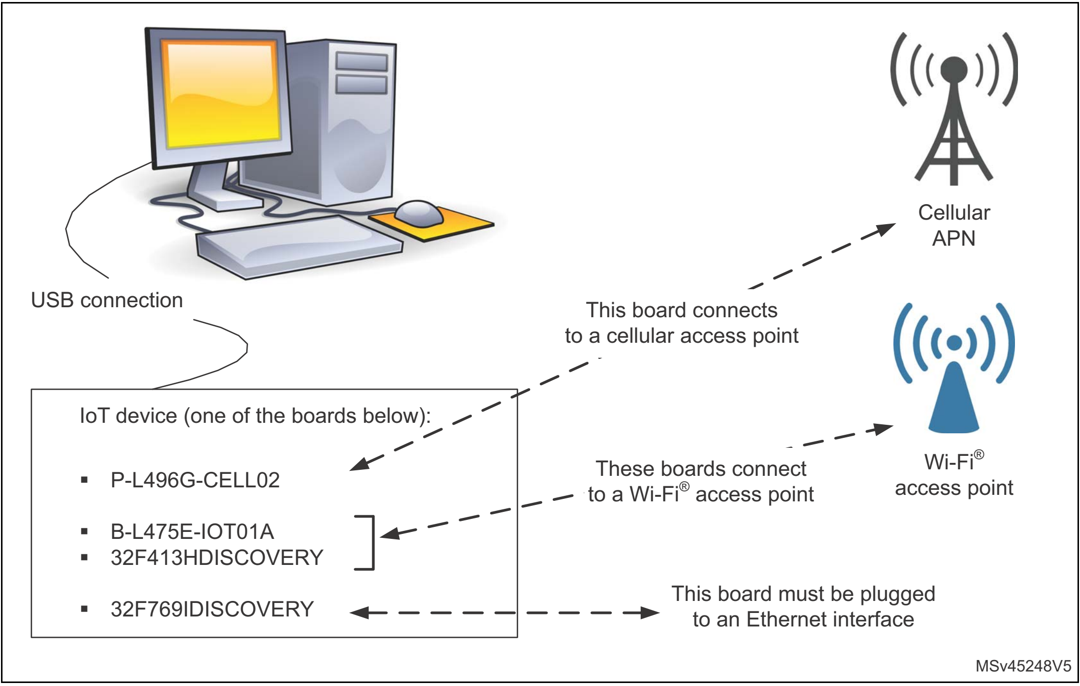
3 Hardware and software environment setup

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

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

The B-L475E-IOT01A or 32F413HDISCOVERY boards must be connected to a Wi-Fi® access point. The P-L496G-CELL02 kit must be connected to a cellular APN. The 32F769IDISCOVERY board must be connected to an Ethernet interface (see *Figure 2*).

Figure 2. Hardware and software setup environment


Cellular APN
USB connection
This board connects to a cellular access point
IoT device (one of the boards below):
P-L496G-CELL02
B-L475E-IOT01A
32F413HDISCOVERY
32F769IDISCOVERY
These boards connect to a Wi-Fi® access point
Wi-Fi® access point
This board must be plugged to an Ethernet interface
MSv45248V5

4 Package description

This section details the X-CUBE-AWS package content and how to use it.

4.1 Description

The X-CUBE-AWS package provides an AWS stack middleware for the STM32 microcontrollers. The package is split into the following components:

- AWS SDK for connecting to AWS IoT from a device using embedded C
- mbedTLS
- LwIP
- FreeRTOS™
- Wi-Fi® drivers
- Ethernet driver for the 32F769IDISCOVERY board
- Sensor drivers for the B-L475E-IOT01A board
- STM32L4 Series, STM32F4 Series and STM32F7 Series HAL
- AWS application examples
- Secure Boot loader derived from the X-CUBE-SBSFU Expansion Package
- Cellular Framework derived from X-CUBE-CELLULAR Expansion Package
- Connectivity framework

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

The following Integrated development environments are supported:

- IAR™ Embedded Workbench for ARM® (EWARM).
  Version 8.30.1 or higher must be used
- Keil® Microcontroller Development Kit (MDK-ARM)
  Version 5.26 or higher must be used
- System Workbench for STM32.
  Version 2.8.1or higher must be used

4.2 Architecture

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

The X-CUBE-AWS is an expansion for the STM32Cube. Its main features and characteristics are:

- It complies with the STM32Cube architecture
- It expands the STM32Cube in order to enable the development of applications accessing and using the AWS IoT
- It is based on the STM32Cube HAL, which is the hardware abstraction layer for the STM32 microcontrollers

The software layers used by the application software to access and use the AWS IoT are the following:

- **STM32Cube HAL layer**: 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 microcontroller unit (MCU).
  This structure improves the library code reusability and guarantees an easy portability onto other devices.
- **Board support package (BSP) layer**: 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 user button.
- **AWS middleware**: MQTT client.
- **mbedTLS**: AWS middleware relies on a TLS connection, which is provided by the mbedTLS library.
- The TCP/IP connection is handled either by the Wi-Fi® module, by the cellular module, or by the LwIP middleware (when an Ethernet connection is being used). In the X-CUBE-AWS package, only the 32F769IDISCOVERY board connects via the Ethernet.
- **FreeRTOS™** is a real-time operating system. It is a dependency of the LwIP sockets.
- **Secure Boot and Secure Firmware Update**:

  Secure Boot Loader: it manages application verification and authentication at boot time.

  Secure Firmware Upgrade: it manages the firmware update in a trusted secure way.
  The Secure Boot Loader 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-AWS Expansion Package.
- **jsmn**: jsmn is a minimalistic JSON parser in C.
- **Cellular framework**: the Cellular framework is the software stack allowing the drive ro cellular modems from an STM32 MCU.
- **STM32 Connect Library**:

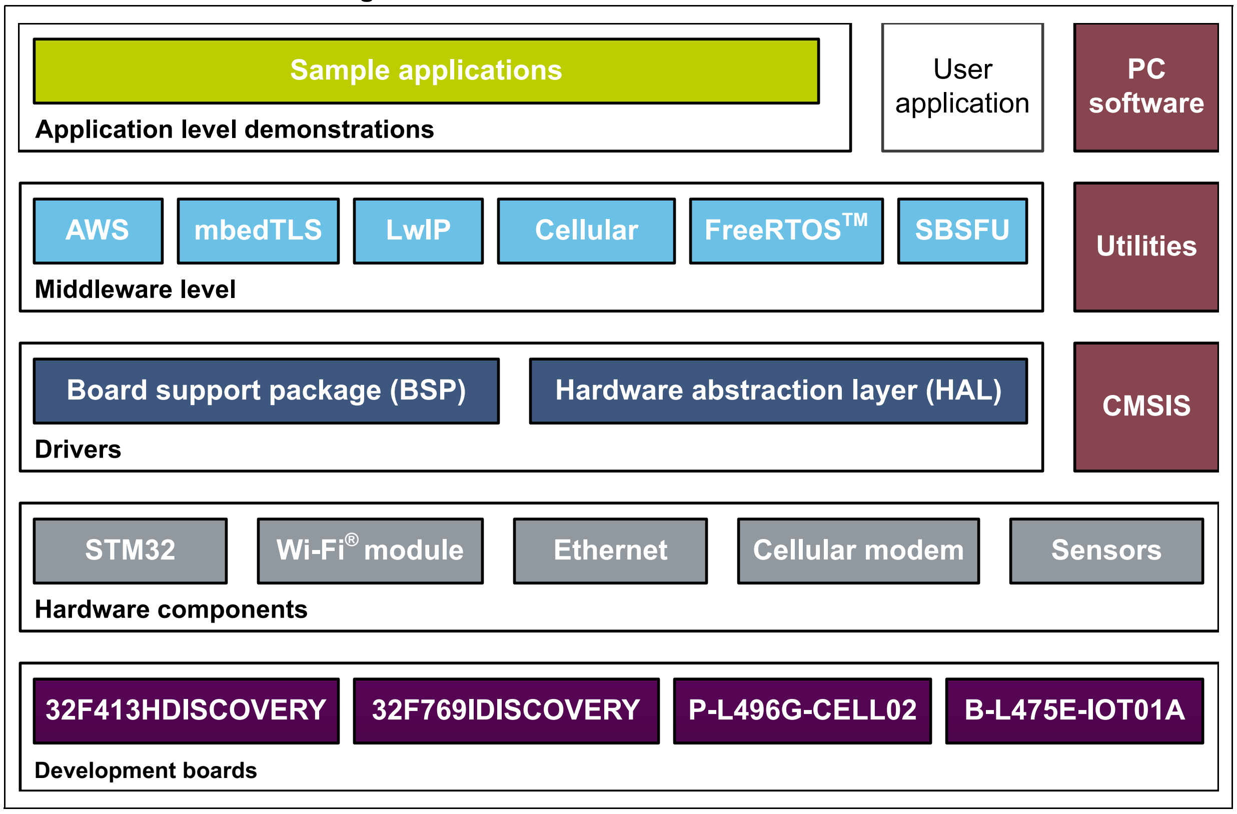
  The STM32 Connect Library provides an API to access network services on STM32 devices.

  It supports several network adapters and several protocols.

  This API is intended to be used in any STM32Cube application requiring network services.

Figure 3 outlines the X-CUBE-AWS software architecture.

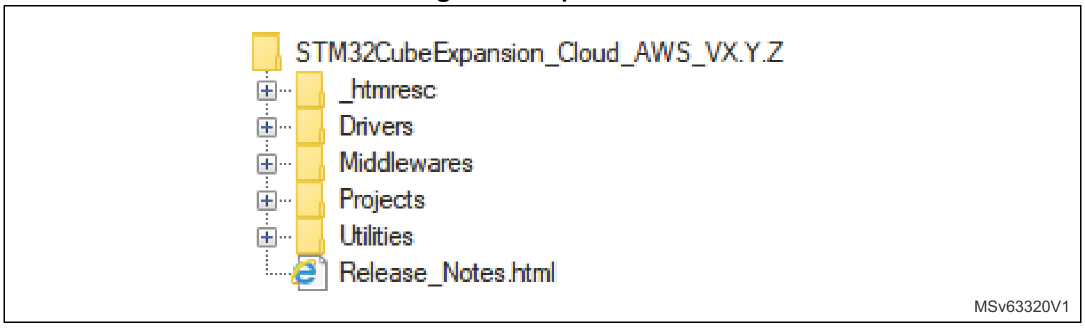
Figure 3. X-CUBE-AWS software architecture



4.3 Folder structure

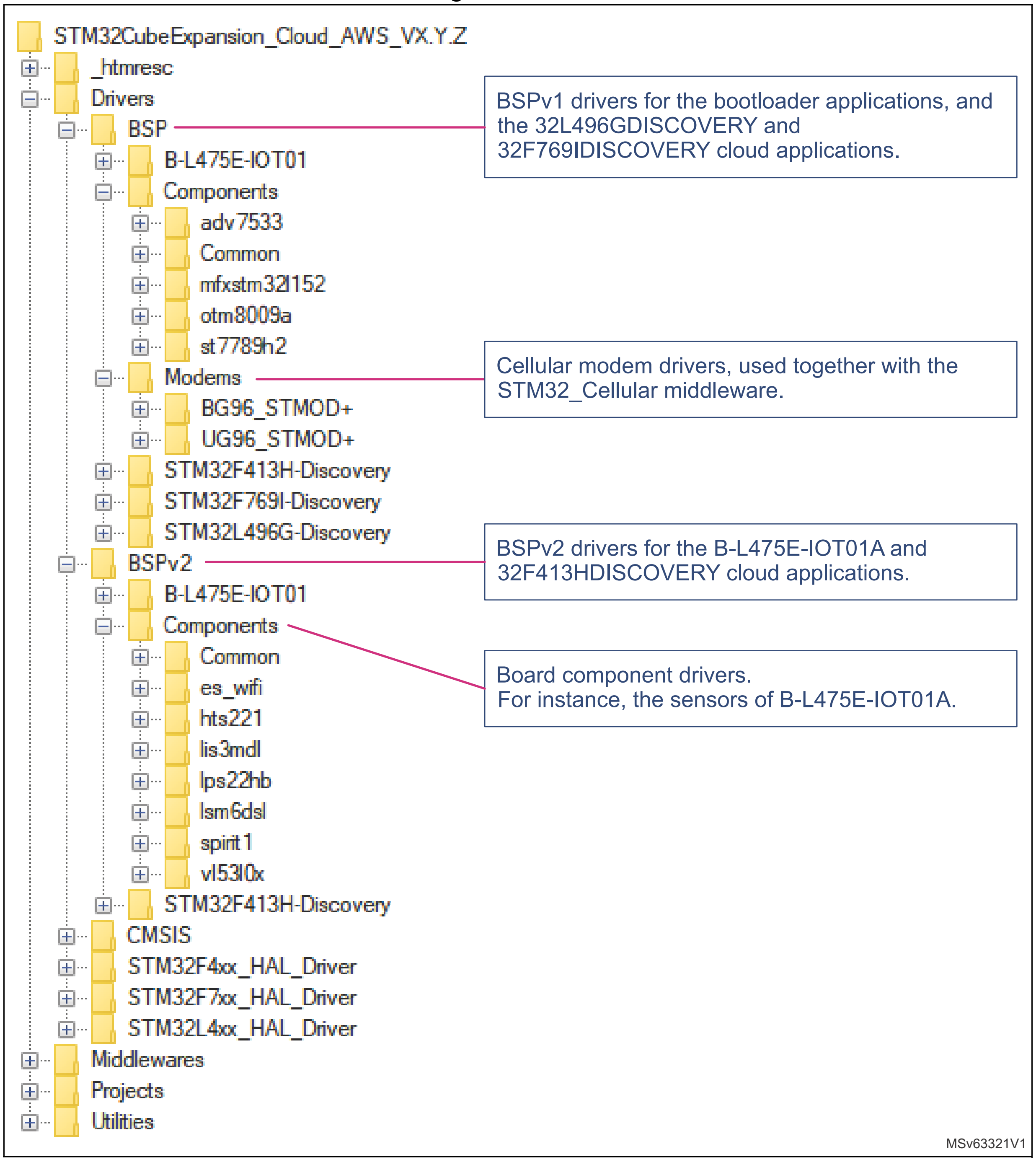
Figure 4 presents the top folder structure of the X-CUBE AWS package. *Figure 5: Drivers*, *Figure 6: Middleware*, and *Figure 7: Projects and utilities* further detail the top folder contents.

Figure 4. Top folders



```
STM32CubeExpansion_Cloud_AWS_VX.Y.Z
  _htmresc
  Drivers
  Middlewares
  Projects
  Utilities
  Release_Notes.html
```

Figure 5. Drivers



```
STM32CubeExpansion_Cloud_AWS_VX.Y.Z
├─ _htmresc
├─ Drivers
│  ├─ BSP
│  │  ├─ B-L475E-IOT01
│  │  ├─ Components
│  │  │  ├─ adv7533
│  │  │  ├─ Common
│  │  │  ├─ mfxstm32l152
│  │  │  ├─ otm8009a
│  │  │  └─ st7789h2
│  │  ├─ Modems
│  │  │  ├─ BG96_STMOD+
│  │  │  └─ UG96_STMOD+
│  │  ├─ STM32F413H-Discovery
│  │  ├─ STM32F769I-Discovery
│  │  └─ STM32L496G-Discovery
│  ├─ BSPv2
│  │  ├─ B-L475E-IOT01
│  │  ├─ Components
│  │  │  ├─ Common
│  │  │  ├─ es_wifi
│  │  │  ├─ hts221
│  │  │  ├─ lis3mdl
│  │  │  ├─ lps22hb
│  │  │  ├─ lsm6dsl
│  │  │  ├─ spirit1
│  │  │  └─ vl53l0x
│  │  └─ STM32F413H-Discovery
│  ├─ CMSIS
│  ├─ STM32F4xx_HAL_Driver
│  ├─ STM32F7xx_HAL_Driver
│  └─ STM32L4xx_HAL_Driver
├─ Middlewares
├─ Projects
└─ Utilities
```

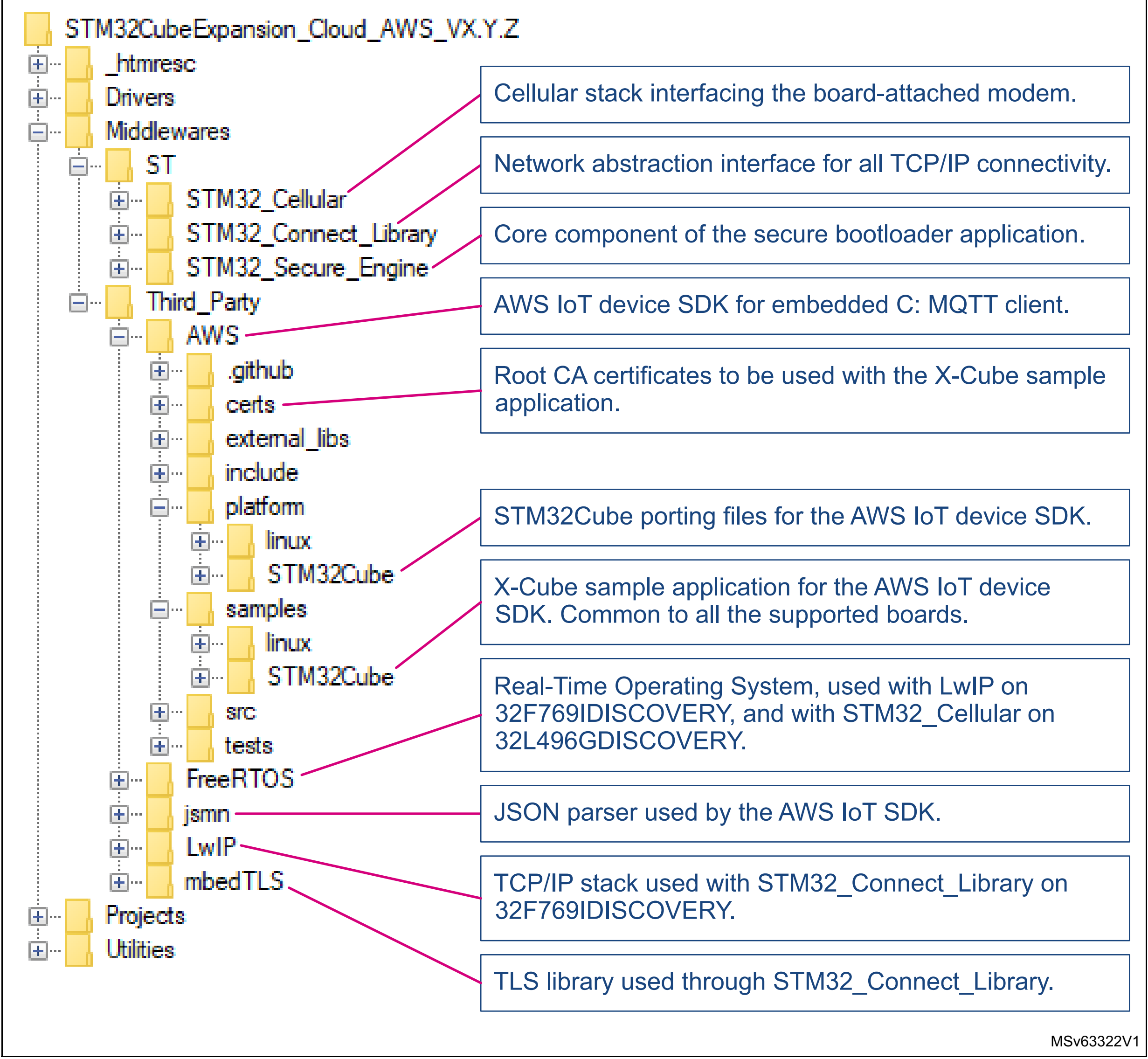
BSP: BSPv1 drivers for the bootloader applications, and the 32L496GDISCOVERY and 32F769IDISCOVERY cloud applications.

Modems: Cellular modem drivers, used together with the STM32_Cellular middleware.

BSPv2: BSPv2 drivers for the B-L475E-IOT01A and 32F413HDISCOVERY cloud applications.

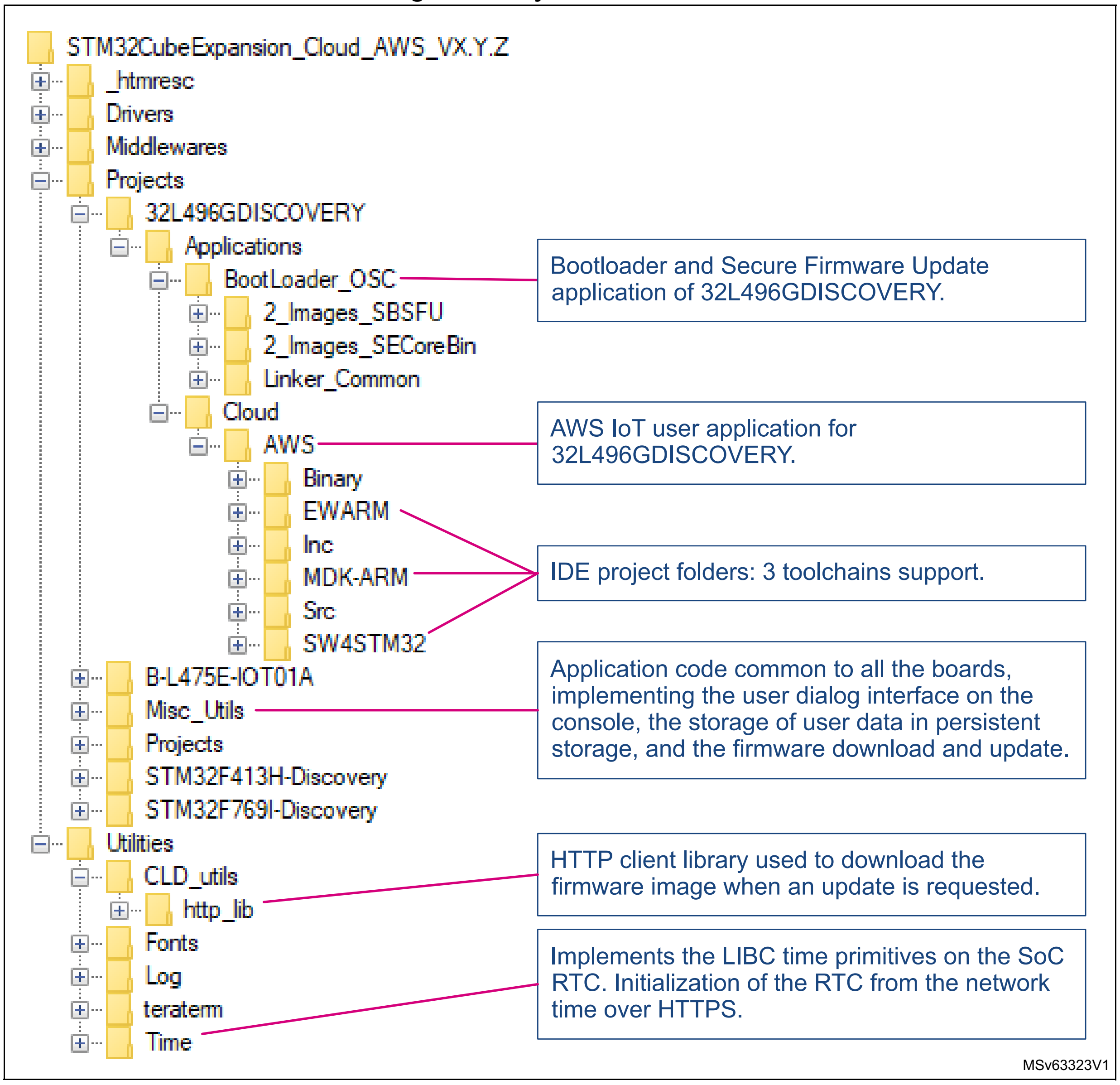
Components: Board component drivers. For instance, the sensors of B-L475E-IOT01A.

Figure 6. Middleware



STM32CubeExpansion_Cloud_AWS_VX.Y.Z
_htmresc
Drivers
Middlewares
ST
STM32_Cellular
STM32_Connect_Library
STM32_Secure_Engine
Third_Party
AWS
.github
certs
external_libs
include
platform
linux
STM32Cube
samples
linux
STM32Cube
src
tests
FreeRTOS
jsmn
LwIP
mbedTLS
Projects
Utilities
Cellular stack interfacing the board-attached modem.
Network abstraction interface for all TCP/IP connectivity.
Core component of the secure bootloader application.
AWS IoT device SDK for embedded C: MQTT client.
Root CA certificates to be used with the X-Cube sample application.
STM32Cube porting files for the AWS IoT device SDK.
X-Cube sample application for the AWS IoT device SDK. Common to all the supported boards.
Real-Time Operating System, used with LwIP on 32F769IDISCOVERY, and with STM32_Cellular on 32L496GDISCOVERY.
JSON parser used by the AWS IoT SDK.
TCP/IP stack used with STM32_Connect_Library on 32F769IDISCOVERY.
TLS library used through STM32_Connect_Library.
MSv63322V1

Figure 7. Projects and utilities


STM32CubeExpansion_Cloud_AWS_VX.Y.Z
_htmresc
Drivers
Middlewares
Projects
32L496GDISCOVERY
Applications
BootLoader_OSC
2_Images_SBSFU
2_Images_SECoreBin
Linker_Common
Cloud
AWS
Binary
EWARM
Inc
MDK-ARM
Src
SW4STM32
B-L475E-IOT01A
Misc_Utils
Projects
STM32F413H-Discovery
STM32F769I-Discovery
Utilities
CLD_utils
http_lib
Fonts
Log
teraterm
Time
Bootloader and Secure Firmware Update application of 32L496GDISCOVERY.
AWS IoT user application for 32L496GDISCOVERY.
IDE project folders: 3 toolchains support.
Application code common to all the boards, implementing the user dialog interface on the console, the storage of user data in persistent storage, and the firmware download and update.
HTTP client library used to download the firmware image when an update is requested.
Implements the LIBC time primitives on the SoC RTC. Initialization of the RTC from the network time over HTTPS.
MSv63323V1

4.4 Network connectivity

The applications of the present package use TCP and TLS client sockets, which are implemented by the `STM32_Connect_Library` middleware component.

With Wi-Fi® or Ethernet interfaces, *Projects/<board>/Applications/Cloud/AWS/Src/net_conf.c* contains the board adaptation code, based on the template file that is provided by the middleware component in *Middlewares/ST/ST_Connect_Library/templates*.

It depends on the es_wifi driver in the former case, and on the LwIP middleware in the latter.

With a cellular interface, the X-CUBE-CELLULAR stack is used, with its specific porting layer.

4.5 Reset push-button

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

4.6 User push-button

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

- Activate the Wi-Fi® and AWS security credentials configuration dialogs on the UART console, or to manually initiate a firmware update. This can be done at startup, after the bootloader has started the user application, and before the connection to AWS IoT.
- Publish messages for the desired LED status when the board is connected to AWS IoT.
- Double-push to exit the application when the board is connected to AWS IoT.

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

Depending on the board, the BSP functions are located in:

- *Drivers\BSP* with P-L496G-CELL02 and 32F769IDISCOVERY
- *Drivers\BSPv2* with B-L475E-IOT01A and 32F413HDISCOVERY

When using the BSP button functions with the value = `BUTTON_USER`, the application does not need to mind about how this button is connected from a hardware standpoint for a given platform. This mapping is handled by the BSP.

4.7 User LED

The configuration of the user LED is done via the board support package (BSP) functions located in the *Drivers\BSP\<board name>* directory.

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

Using the BSP button functions with the value = `LED_GREEN`, the application does not need to mind about how this LED is mapped for a given platform. This mapping is handled by the BSP.

4.8 Real time clock

The STM32 RTC is updated at startup from the www.gandi.net web server.

The user can use the `HAL_RTC_GetTime()` or `libc time()` function to get the current time.

These functions can be used to time stamp some messages for example.

4.9 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` #define.

The X-CUBE-AWS package uses *Projects\<board name>\Applications\Cloud\AWS\Inc\mbedtls_config.h* for project configuration.

This is implemented by having this below # 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
```

Some macros are redefined to minimize the RAM footprint, sometimes at cost of ROM memory. See below:

```
#define MBEDTLS_AES_ROM_TABLES

#define MBEDTLS_MPI_WINDOW_SIZE          1 /**< Maximum windows size
used. */

#define MBEDTLS_MPI_MAX_SIZE           512 /**< Maximum number of
bytes for usable MPIs. */

#define MBEDTLS_ECP_WINDOW_SIZE          2 /**< Maximum window size
used */

#define MBEDTLS_ECP_FIXED_POINT_OPTIM    0 /**< Disable fixed-point
speedup */

#define MBEDTLS_SSL_MAX_CONTENT_LEN   6000 /**< Maximum fragment
length in bytes, determines the size of each of the two internal I/O
buffers */
```

The configuration file specifies which ciphers to integrate.

4.10 Application examples

This section describes the way to configure the IoT device. This is independent from the platform.

Section 6, *Section 7* and *Section 8* detail the specificities for the B-L475E-IOT01A, 32F413HDISCOVERY, 32F769IDISCOVERY and P-L496G-CELL02 boards respectively.

4.10.1 General description

The X-CUBE-AWS package runs on four platforms:

- B-L475E-IOT01A supports Wi-Fi® connectivity with an on-board Inventek module. This board is equipped with a set of sensors able to report humidity, temperature, 3-axis

magnetic data, 3D accelerations, 3D gyroscope data, atmospheric pressure, proximity and gesture detection (X-CUBE-AWS does not use the gesture detection capability).
- P-L496G-CELL02 supports cellular connectivity through the attached BG96 Quectel external module.
- 32F413HDISCOVERY supports Wi-Fi® connectivity with an on-board Inventek module.
- 32F769IDISCOVERY provides a native Ethernet interface.

For those four platforms, a sample application supports below features:
- Configures the board with the Wi-Fi® or cellular parameters (if applicable) and AWS security credentials
- Connects to the Internet
- Allows for a locally-initiated firmware update
- Connects to AWS
- Publishes MQTT messages and subscribes to topics, depending on the board capabilities as described in *Table 3* below.

Table 3. Published messages and subscribe actions by the applications

-	B-L475E-IOT01A	32F413HDISCOVERY, 32F769IDISCOVERY, P-L496G-CELL02
Published information	Message to toggle the LED: "`desired state = ON`" when the LED is OFF and vice versa. A message is published each time the user button is pressed.	
	The sensor data is published around every 10 seconds.	-

Subscribe actions	The MQTT subscribe callback prints the received message.
	In addition, LED state control commands are obeyed, and custom AWS IoT jobs are implemented, which allows
	– simulation of actions on the board user button:
	`{ "myOperation": "simulate_single_push" }`
	or
	`{ "myOperation": "simulate_multiple_push" }`
	– trigger of a firmware download and update:
	`{ firmwareUpdate: <http url> }`
	
	Refer to *Chapter 6: Specificities with the B-L475E-IOT01A board* for details.	

4.10.2 LED behavior of the four platforms

The four platforms behave the same way as far as user LED and user button are concerned.

Once the initialization sequence is completed and the device has connected to AWS IoT, pushing the user button (blue) triggers the publication of a message on the *$aws/things/<YourThingNameDeclaredIn AWSAndStoredInFlash>/shadow/update* topic.

This message depends on the user LED status:

- If the LED is OFF, the following message is sent:
  `{"state":{"desired":{"value":"On"}}}`
- If the LED is ON, the following message is sent:
  `{"state":{"desired":{"value":"Off"}}}`

When a message is received on the *$aws/things/<YourThingNameDeclaredIn AWSAndStoredInFlash>/shadow/update/accepted* topic, a MQTT callback is executed. If a message dealing with the LED is received, the LED status is set according to the desired value received:

- The callback makes the LED switch off if the `"desired" : {"value":"Off"}` is received. Then, the callback publishes a `{"state":{"reported":{"LED_value":"Off"}}}` message.
- The callback makes the LED switch on if the `"desired" : {"value":"On"}` is received. Then, the callback publishes a `{"state":{"reported":{"LED_value":"On"}}}` message.

Hence, the LED toggles via the AWS cloud each time the user button (blue) is pressed.

4.10.3 Interacting with the boards

A serial terminal is required to:

- Configure the board
- Display locally the published and received AWS IoT 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 AWS 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 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 (see *Figure 5*); this script sets the correct parameters. To use it, open Tera Term, select "Setup" then "Restore setup…".

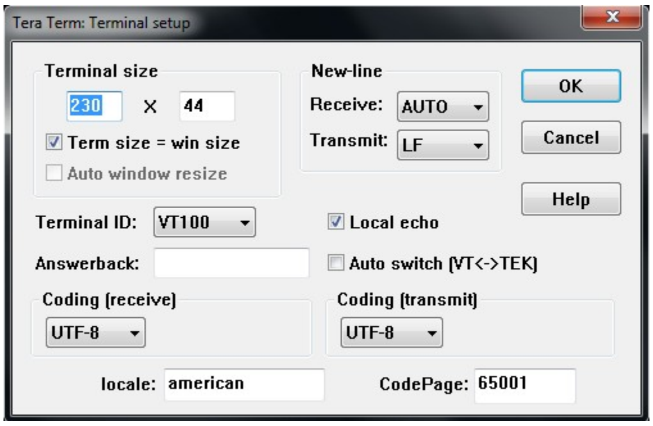
Note: The information provided below in this chapter can be used to configure the UART terminal as an alternative to using the Tera Term initialization script.

Terminal setup is illustrated in *Figure 8*, 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 the copy-paste from the UNIX®[a] type text files. The "Local echo" option makes copy-paste visible on the console.

a. UNIX is a registered trademark of The Open Group.

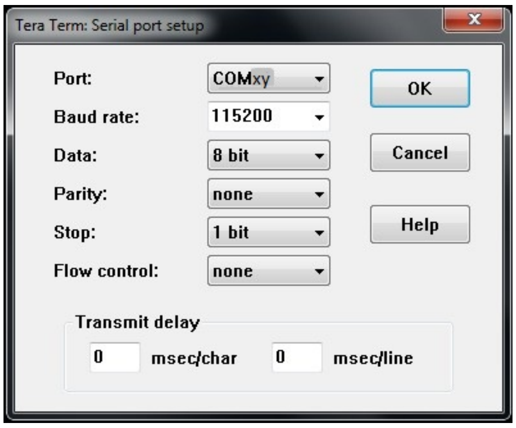
Figure 8. UART terminal setup


Tera Term: Terminal setup

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

The serial port is to be configured with: COM port number, 115200 baud rate, 8-bit data, parity none, 1 stop bit and no flow control, as highlighted in *Figure 9*

Figure 9. 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 AWS data. Those data remain in Flash and are reused the next time the board boots.

4.10.4 Wi-Fi®, cellular and AWS security credential update

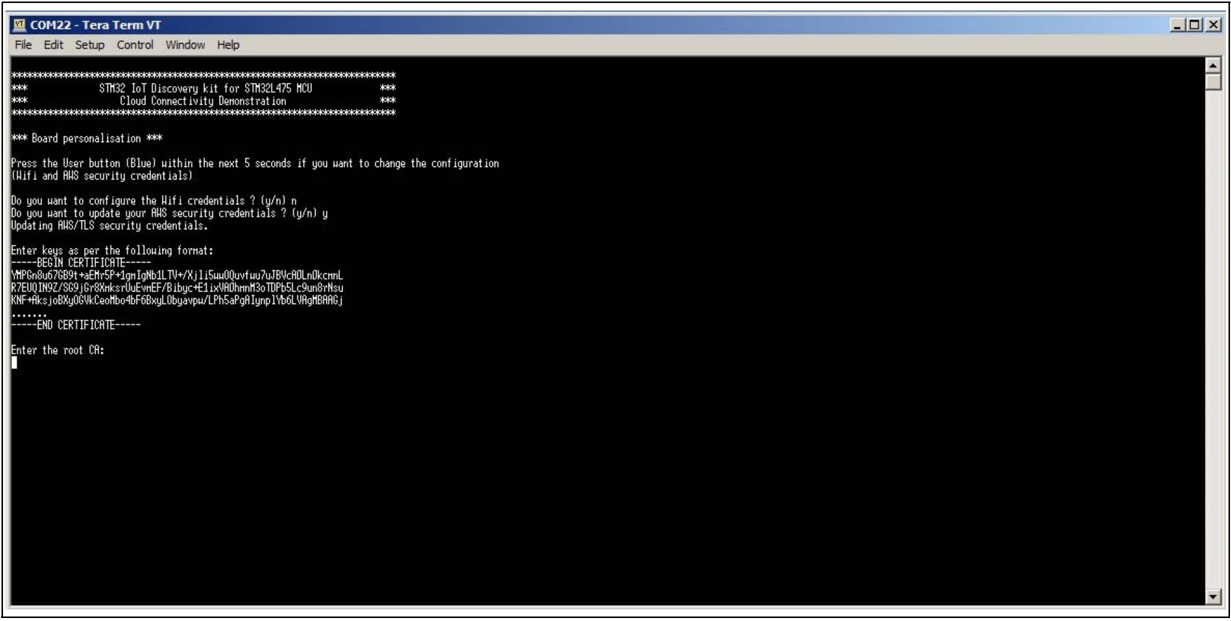
The credentials must be entered or updated as follows:

- Either by pressing the user button (blue) at board startup when prompted on the console
- Or, if the credentials have not been stored into the Flash memory yet, or if the Flash has been erased in the meantime, the console unconditionally asks for them

Figure 10, *Figure 11* and *Figure 12* show an example of an AWS root certificate update.

In *Figure 10*, the user has pressed the user button (blue) during the five seconds window after the start-up and has asked for the AWS credentials update.

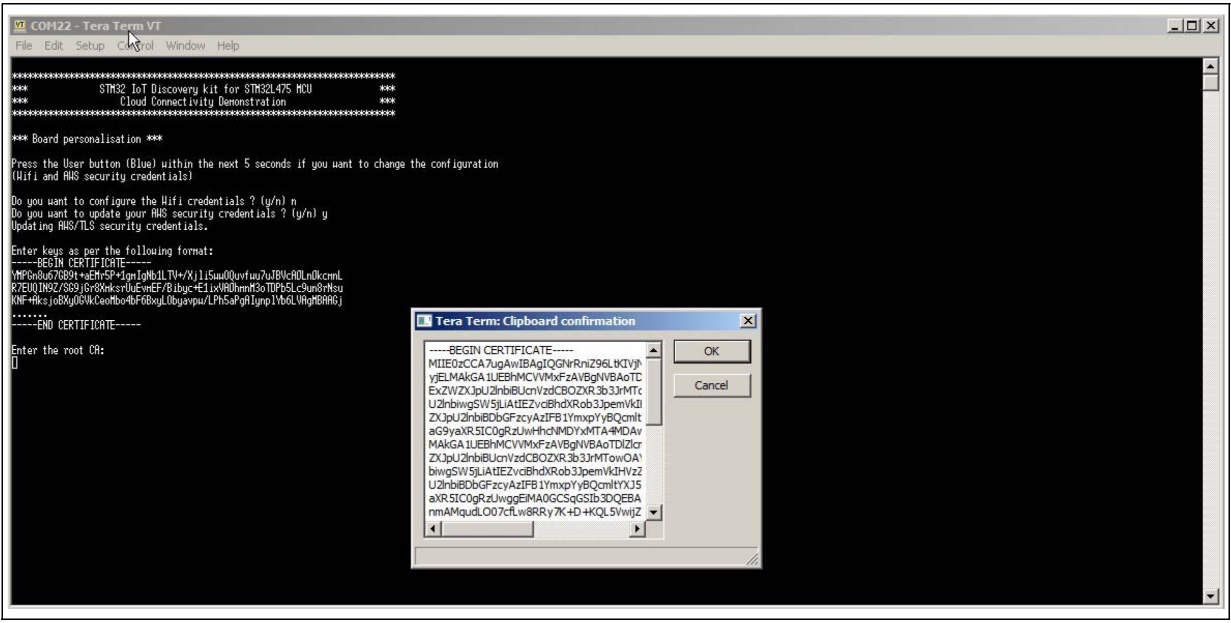
Figure 10. AWS root certificate update - prompt for entering root CA



```
****************************************************************************
***             STM32 IoT Discovery kit for STM32L475 MCU              ***
***                Cloud Connectivity Demonstration                    ***
****************************************************************************

*** Board personalisation ***

Press the User button (Blue) within the next 5 seconds if you want to change the configuration
(Wifi and AWS security credentials)

Do you want to configure the Wifi credentials ? (y/n) n
Do you want to update your AWS security credentials ? (y/n) y
Updating AWS/TLS security credentials.

Enter keys as per the following format:
-----BEGIN CERTIFICATE-----
YMPGn8u67GB9t+aEMr5P+1gmIgNb1LTV+/Xjli5ww0QuvfwU7uJBVcA0LnOkcmnL
R7EUQIN9Z/SG9jGr8XmksrUuEvmEF/Bibyc+E1ixVA0hmnM3oTDPb5Lc9un8rNsu
KNF+AksjoBXyOGVkCeoMbo4bF6BxyL0byavpw/LPh5aPgAIynplYb6LVAgMBAAGj
.......
-----END CERTIFICATE-----

Enter the root CA:
```

In *Figure 11*, the user copies his root CA that he has previously saved and pastes it into the console.

Figure 11. AWS root certificate update - copy paste root CA

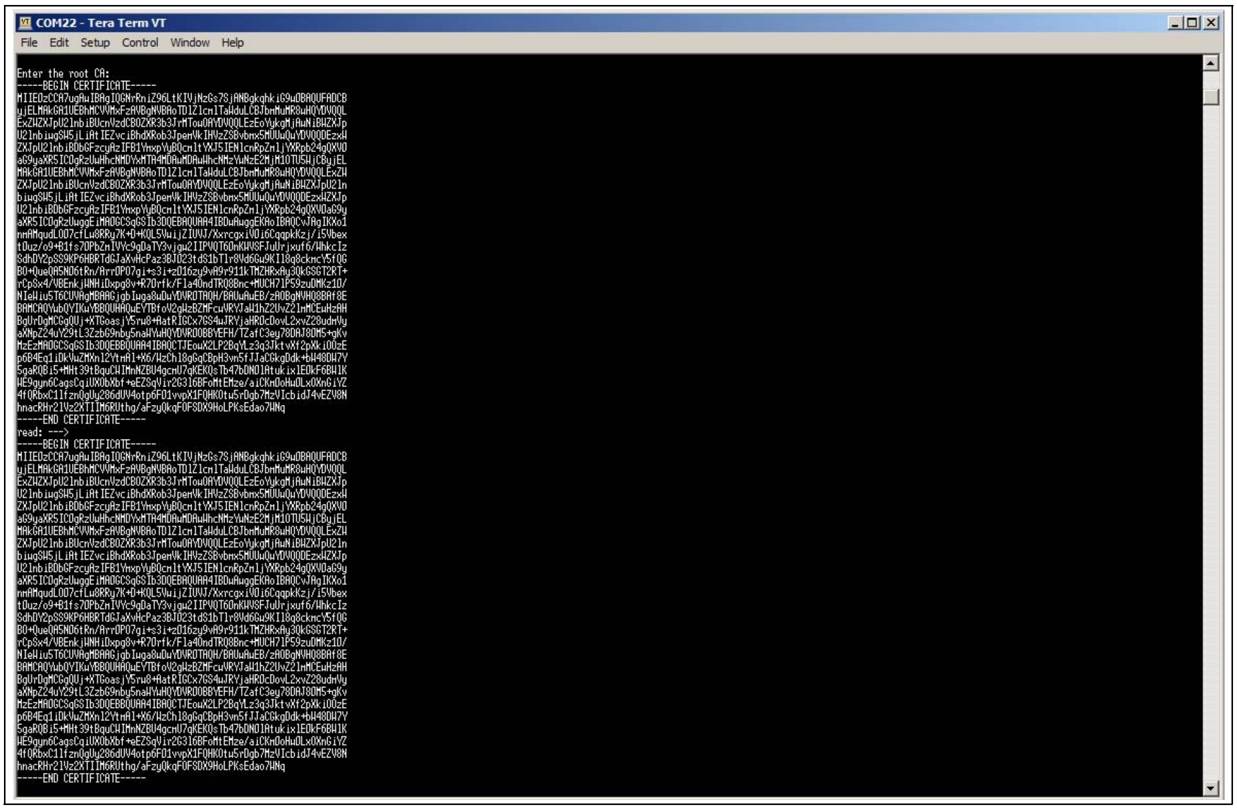

COM22 - Tera Term VT
File Edit Setup Control Window Help
Tera Term: Clipboard confirmation
OK
Cancel

```
****************************************************************************
***             STM32 IoT Discovery kit for STM32L475 MCU              ***
***                Cloud Connectivity Demonstration                    ***
****************************************************************************

*** Board personalisation ***

Press the User button (Blue) within the next 5 seconds if you want to change the configuration
(Wifi and AWS security credentials)

Do you want to configure the Wifi credentials ? (y/n) n
Do you want to update your AWS security credentials ? (y/n) y
Updating AWS/TLS security credentials.

Enter keys as per the following format:
-----BEGIN CERTIFICATE-----
YMPGn8u67GB9t+aEMr5P+1gmIgNb1LTV+/Xjli5ww0QuvfwU7uJBVcA0LnOkcmnL
R7EUQIN9Z/SG9jGr8XmksrUuEvmEF/Bibyc+E1ixVA0hmnM3oTDPb5Lc9un8rNsu
KNF+AksjoBXyOGVkCeoMbo4bF6BxyL0byavpw/LPh5aPgAIynplYb6LVAgMBAAGj
.......
-----END CERTIFICATE-----

Enter the root CA:
```

```
-----BEGIN CERTIFICATE-----
MIIE0zCCA7ugAwIBAgIQGNrRniZ96LtKIVjN
yjELMAkGA1UEBhMCVVMxFzAVBgNVBAoTD
ExZWZXJpU2lnbiBUcnVzdCBOZXR3b3JrMTc
U2lnbiwgSW5jLiAtIEZvciBhdXRob3JpemVkIE
ZXJpU2lnbiBDbGFzcyAzIFB1YmxpYyBQcmlt
aG9yaXR5IC0gRzUwHhcNMDYxMTA4MDAw
MAkGA1UEBhMCVVMxFzAVBgNVBAoTDlZlcr
ZXJpU2lnbiBUcnVzdCBOZXR3b3JrMTowOAY
biwgSW5jLiAtIEZvciBhdXRob3JpemVkIHVzZ
U2lnbiBDbGFzcyAzIFB1YmxpYyBQcmltYXJ5
aXR5IC0gRzUwggEiMA0GCSqGSIb3DQEBA
nmAMqudLO07cfLw8RRy7K+D+KQL5VwijZ
```

Figure 12 shows the acknowledgment from the device after “read: --->”. This corresponds to what is saved in the Flash memory.

Figure 12. AWS root certificate update - console feedback



```
Enter the root CA:
-----BEGIN CERTIFICATE-----
MIIE0zCCA7ugAwIBAgIQGNrRniZ96LtKIVjNzGs7SjANBgkqhkiG9w0BAQUFADCB
yjELMAkGA1UEBhMCVVMxFzAVBgNVBAoTDlZlcmlTaWduLCBJbmMuMR8wHQYDVQQL
ExZWZXJpU2lnbiBUcnVzdCBOZXR3b3JrMTowOAYDVQQLEzEoYykgMjAwNiBWZXJp
U2lnbiwgSW5jLiAtIEZvciBhdXRob3JpemVkIHVzZSBvbmx5MUUwQwYDVQQDEzxW
ZXJpU2lnbiBDbGFzcyAzIFB1YmxpYyBQcmltYXJ5IENlcnRpZmljYXRpb24gQXV0
aG9yaXR5IC0gRzUwHhcNMDYxMTA4MDAwMDAwWhcNMzYwNzE2MjM1OTU5WjCByjEL
MAkGA1UEBhMCVVMxFzAVBgNVBAoTDlZlcmlTaWduLCBJbmMuMR8wHQYDVQQLExZW
ZXJpU2lnbiBUcnVzdCBOZXR3b3JrMTowOAYDVQQLEzEoYykgMjAwNiBWZXJpU2ln
biwgSW5jLiAtIEZvciBhdXRob3JpemVkIHVzZSBvbmx5MUUwQwYDVQQDEzxWZXJp
U2lnbiBDbGFzcyAzIFB1YmxpYyBQcmltYXJ5IENlcnRpZmljYXRpb24gQXV0aG9y
aXR5IC0gRzUwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCvJAgIKXo1
nmAMqudLO07cfLw8RRy7K+D+KQL5VwijZIUVJ/XxrcgxiV0i6CqqpkKzj/i5Vbex
t0uz/o9+B1fs70PbZmIVYc9gDaTY3vjgw2IIPVQT60nKWVSFJuUrjxuf6/WhkcIz
SdhDY2pSS9KP6HBRTdGJaXvHcPaz3BJ023tdS1bTlr8Vd6Gw9KIl8q8ckmcY5fQG
BO+QueQA5N06tRn/Arr0PO7gi+s3i+z016zy9vA9r911kTMZHRxAy3QkGSGT2RT+
rCpSx4/VBEnkjWNHiDxpg8v+R70rfk/Fla4OndTRQ8Bnc+MUCH7lP59zuDMKz10/
NIeWiu5T6CUVAgMBAAGjgbIwga8wDwYDVR0TAQH/BAUwAwEB/zAOBgNVHQ8BAf8E
BAMCAQYwbQYIKwYBBQUHAQwEYTBfoV2gWzBZMFcwVRYJaW1hZ2UvZ2lmMCEwHzAH
BgUrDgMCGgQUj+XTGoasjY5rw8+AatRIGCx7GS4wJRYjaHR0cDovL2xvZ28udmVy
aXNpZ24uY29tL3ZzbG9nby5naWYwHQYDVR0OBBYEFH/TZafC3ey78DAJ80M5+gKv
MzEzMA0GCSqGSIb3DQEBBQUAA4IBAQCTJEowX2LP2BqYLz3q3JktvXf2pXkiOOzE
p6B4Eq1iDkVwZMXnl2YtmAl+X6/WzChl8gGqCBpH3vn5fJJaCGkgDdk+bW48DW7Y
5gaRQBi5+MHt39tBquCWIMnNZBU4gcmU7qKEKQsTb47bDN0lAtukixlE0kF6BWlK
WE9gyn6CagsCqiUXObXbf+eEZSqVir2G3l6BFoMtEMze/aiCKm0oHw0LxOXnGiYZ
4fQRbxC1lfznQgUy286dUV4otp6F01vvpX1FQHKOtw5rDgb7MzVIcbidJ4vEZV8N
hnacRHr2lVz2XTIIM6RUthg/aFzyQkqFOFSDX9HoLPKsEdao7WNq
-----END CERTIFICATE-----
read: --->
-----BEGIN CERTIFICATE-----
MIIE0zCCA7ugAwIBAgIQGNrRniZ96LtKIVjNzGs7SjANBgkqhkiG9w0BAQUFADCB
yjELMAkGA1UEBhMCVVMxFzAVBgNVBAoTDlZlcmlTaWduLCBJbmMuMR8wHQYDVQQL
ExZWZXJpU2lnbiBUcnVzdCBOZXR3b3JrMTowOAYDVQQLEzEoYykgMjAwNiBWZXJp
U2lnbiwgSW5jLiAtIEZvciBhdXRob3JpemVkIHVzZSBvbmx5MUUwQwYDVQQDEzxW
ZXJpU2lnbiBDbGFzcyAzIFB1YmxpYyBQcmltYXJ5IENlcnRpZmljYXRpb24gQXV0
aG9yaXR5IC0gRzUwHhcNMDYxMTA4MDAwMDAwWhcNMzYwNzE2MjM1OTU5WjCByjEL
MAkGA1UEBhMCVVMxFzAVBgNVBAoTDlZlcmlTaWduLCBJbmMuMR8wHQYDVQQLExZW
ZXJpU2lnbiBUcnVzdCBOZXR3b3JrMTowOAYDVQQLEzEoYykgMjAwNiBWZXJpU2ln
biwgSW5jLiAtIEZvciBhdXRob3JpemVkIHVzZSBvbmx5MUUwQwYDVQQDEzxWZXJp
U2lnbiBDbGFzcyAzIFB1YmxpYyBQcmltYXJ5IENlcnRpZmljYXRpb24gQXV0aG9y
aXR5IC0gRzUwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCvJAgIKXo1
nmAMqudLO07cfLw8RRy7K+D+KQL5VwijZIUVJ/XxrcgxiV0i6CqqpkKzj/i5Vbex
t0uz/o9+B1fs70PbZmIVYc9gDaTY3vjgw2IIPVQT60nKWVSFJuUrjxuf6/WhkcIz
SdhDY2pSS9KP6HBRTdGJaXvHcPaz3BJ023tdS1bTlr8Vd6Gw9KIl8q8ckmcY5fQG
BO+QueQA5N06tRn/Arr0PO7gi+s3i+z016zy9vA9r911kTMZHRxAy3QkGSGT2RT+
rCpSx4/VBEnkjWNHiDxpg8v+R70rfk/Fla4OndTRQ8Bnc+MUCH7lP59zuDMKz10/
NIeWiu5T6CUVAgMBAAGjgbIwga8wDwYDVR0TAQH/BAUwAwEB/zAOBgNVHQ8BAf8E
BAMCAQYwbQYIKwYBBQUHAQwEYTBfoV2gWzBZMFcwVRYJaW1hZ2UvZ2lmMCEwHzAH
BgUrDgMCGgQUj+XTGoasjY5rw8+AatRIGCx7GS4wJRYjaHR0cDovL2xvZ28udmVy
aXNpZ24uY29tL3ZzbG9nby5naWYwHQYDVR0OBBYEFH/TZafC3ey78DAJ80M5+gKv
MzEzMA0GCSqGSIb3DQEBBQUAA4IBAQCTJEowX2LP2BqYLz3q3JktvXf2pXkiOOzE
p6B4Eq1iDkVwZMXnl2YtmAl+X6/WzChl8gGqCBpH3vn5fJJaCGkgDdk+bW48DW7Y
5gaRQBi5+MHt39tBquCWIMnNZBU4gcmU7qKEKQsTb47bDN0lAtukixlE0kF6BWlK
WE9gyn6CagsCqiUXObXbf+eEZSqVir2G3l6BFoMtEMze/aiCKm0oHw0LxOXnGiYZ
4fQRbxC1lfznQgUy286dUV4otp6F01vvpX1FQHKOtw5rDgb7MzVIcbidJ4vEZV8N
hnacRHr2lVz2XTIIM6RUthg/aFzyQkqFOFSDX9HoLPKsEdao7WNq
-----END CERTIFICATE-----
```

4.11 Secure Boot loader

4.11.1 Overview

The pre-integrated bootloader allows the update of the user application (initially, the AWS IoT sample application), adding new features, and correcting potential issues. This is performed in a secure way, which prevents any unauthorized update or any 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 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.

The delivered pre-compiled bootloader does not enforce security, so that the user can still plug a debugger and debug the application.

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 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/flash/debug), specific pre- and post-build phases have been 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 flashed and run as usual.
- In order to prevent post-build dependencies issues with the Keil® and SW4STM32 IDEs, the pre-build phase deletes the *elf* file, so as to force the link and post-build even if the user project sources have not changed.

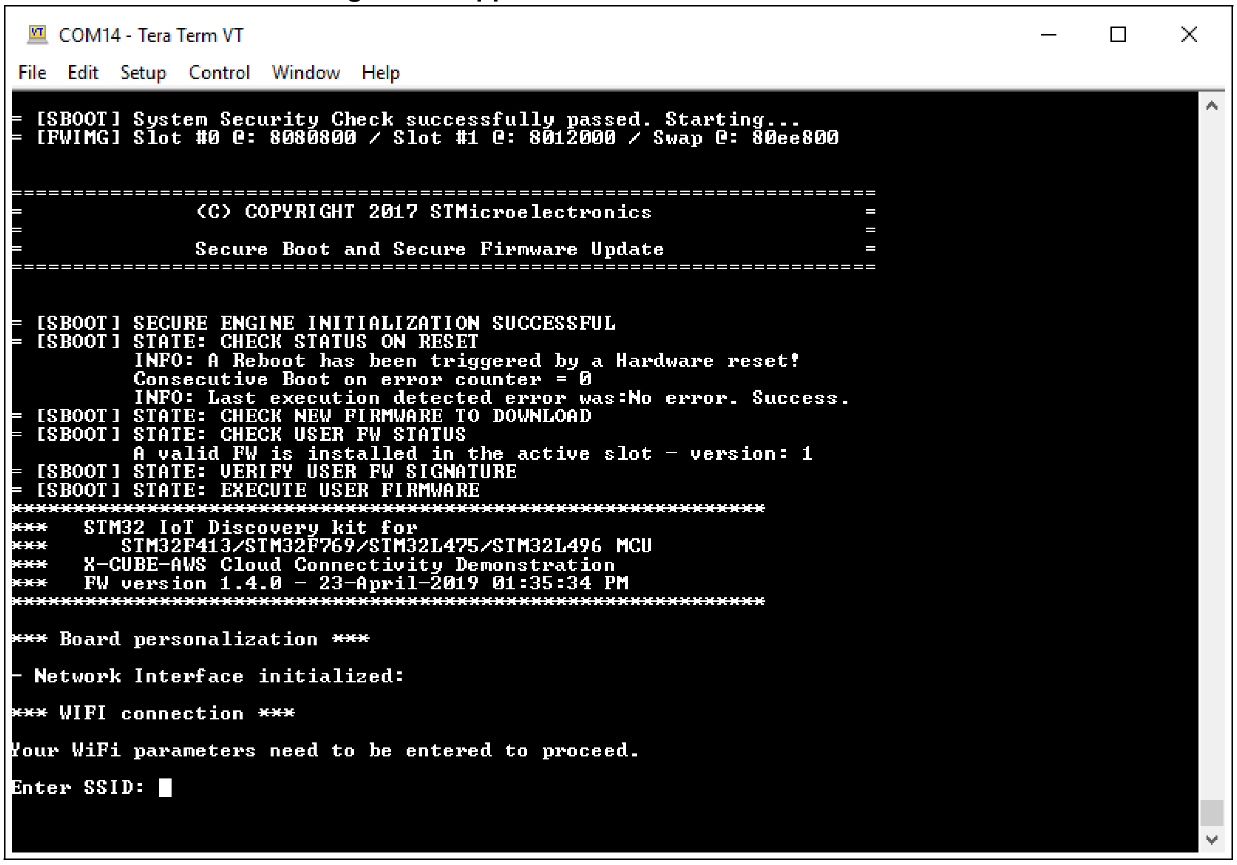
4.11.2 Application boot

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

- If no, it starts the already installed application
- If yes, it decrypts and checks the new firmware before installing and running it

Bootloader messages are prefixed by [SBOOT]. In the screenshot below, the bootloader does not find an application to install from Slot#1. Therefore, it checks the firmware image in Slot#0, and runs it. This sequence is highlighted in *Figure 13*.

Figure 13. Application boot - console feedback

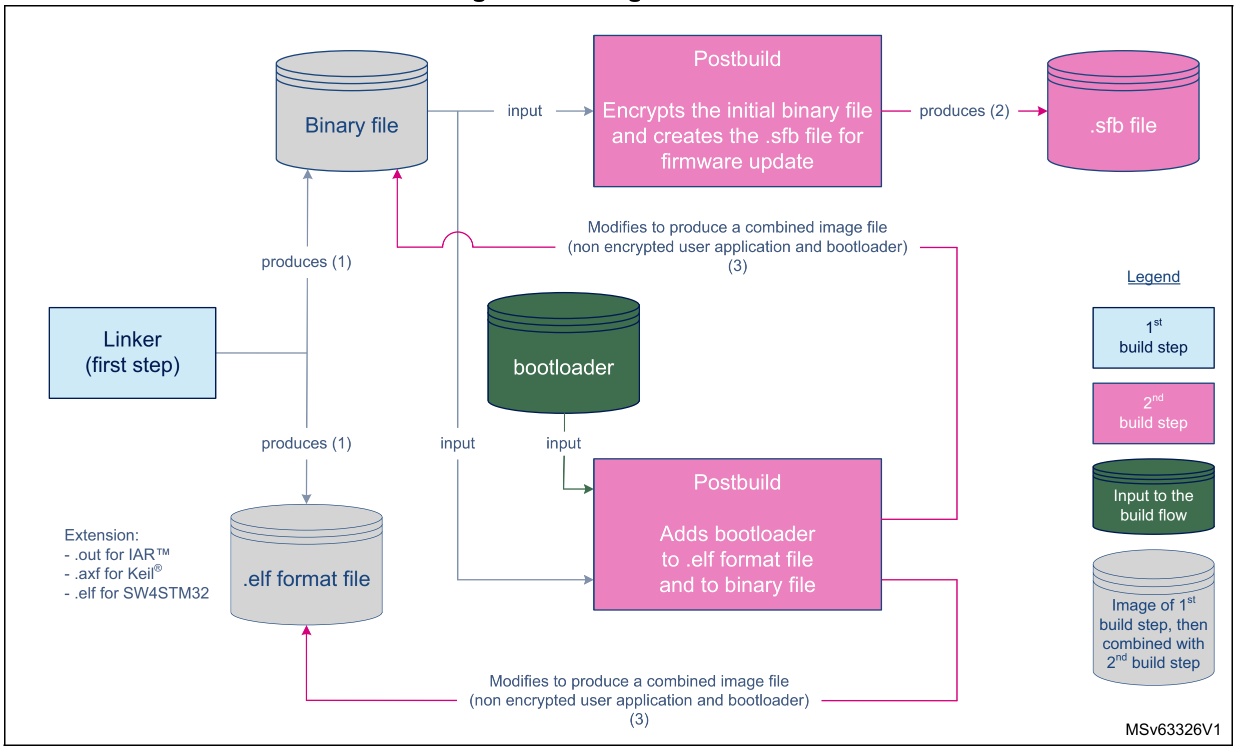


```
= [SBOOT] System Security Check successfully passed. Starting...
= [FWIMG] Slot #0 @: 8080800 / Slot #1 @: 8012000 / Swap @: 80ee800


======================================================================
=              (C) COPYRIGHT 2017 STMicroelectronics                 =
=                                                                    =
=              Secure Boot and Secure Firmware Update                =
======================================================================


= [SBOOT] SECURE ENGINE INITIALIZATION SUCCESSFUL
= [SBOOT] STATE: CHECK STATUS ON RESET
          INFO: A Reboot has been triggered by a Hardware reset!
          Consecutive Boot on error counter = 0
          INFO: Last execution detected error was:No error. Success.
= [SBOOT] STATE: CHECK NEW FIRMWARE TO DOWNLOAD
= [SBOOT] STATE: CHECK USER FW STATUS
          A valid FW is installed in the active slot - version: 1
= [SBOOT] STATE: VERIFY USER FW SIGNATURE
= [SBOOT] STATE: EXECUTE USER FIRMWARE
*************************************************************
***   STM32 IoT Discovery kit for
***      STM32F413/STM32F769/STM32L475/STM32L496 MCU
***   X-CUBE-AWS Cloud Connectivity Demonstration
***   FW version 1.4.0 - 23-April-2019 01:35:34 PM
*************************************************************

*** Board personalization ***

- Network Interface initialized:

*** WIFI connection ***

Your WiFi parameters need to be entered to proceed.

Enter SSID: █
```

4.11.3 Building the whole firmware image

Figure 14 describes the flow for building binary images.

Figure 14. Image build flow


Binary file
input
Postbuild
Encrypts the initial binary file and creates the .sfb file for firmware update
produces (2)
.sfb file
produces (1)
Modifies to produce a combined image file (non encrypted user application and bootloader) (3)
Legend
1st build step
2nd build step
Input to the build flow
Image of 1st build step, then combined with 2nd build step
Linker (first step)
bootloader
produces (1)
input
input
Postbuild
Adds bootloader to .elf format file and to binary file
Extension:
- .out for IAR™
- .axf for Keil®
- .elf for SW4STM32
.elf format file
Modifies to produce a combined image file (non encrypted user application and bootloader) (3)
MSv63326V1

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 file. It combines the bootloader and the user application.

Detail of output files built

The combined image file (binary and *.elf* format) is the bootloader combined with the initial user application. The application is not encrypted. It is located in the executable slot. The files are listed for the various IDEs:

- IAR™
  Project/<boardname>/Applications/Cloud/AWS/EWARM/<boardname>/Exe/<boardname>.out
  Project/<boardname>/Applications/Cloud/AWS/EWARM/<boardname>/<boardname>_AWS.bin
- Keil®
  Project/<boardname>/Applications/Cloud/AWS/MDK-ARM/<boardname>/Exe/<boardname>_AWS.axf

Project/<boardname>/Applications/Cloud/AWS/MDK-ARM/<boardname>/Exe/<boardname>_AWS.bin

- SW4STM32
  Project/<boardname>/Applications/Cloud/AWS/SW4STM32/<boardname>/Debug/<boardname>_AWS.elf
  Project/<boardname>/Applications/Cloud/AWS/SW4STM32/<boardname>/Debug/<boardname>_AWS.bin

The binary file with the *.sfb* format packs the firmware header and the firmware image of the user application. This file is the one used when performing firmware update. The files are listed for the various IDEs:

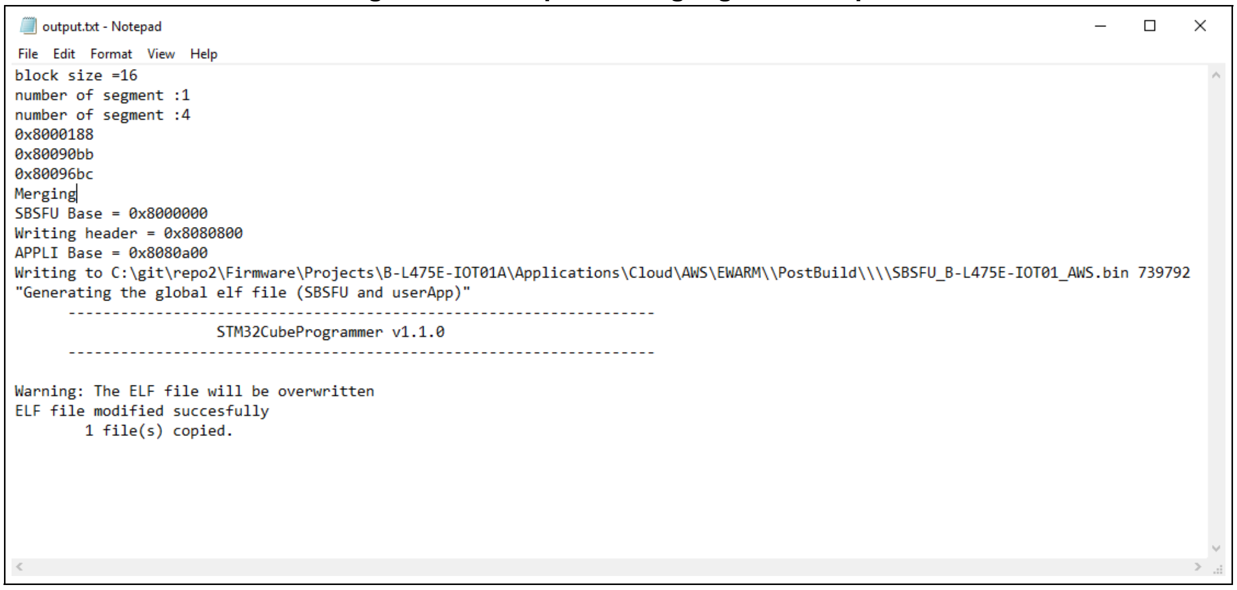
- IAR™
  Project/<boardname>/Applications/Cloud/AWS/EWARM/PostBuild /<boardname>_AWS.sfb
- Keil®
  Project/<boardname>/Applications/Cloud/AWS/MDK-ARM/PostBuild/<boardname>_AWS.sfb
- SW4STM32
  Project/<boardname>/Applications/Cloud/AWS/SW4STM32/PostBuild/<boardname>_AWS.sfb

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 files are listed for the various IDEs:

- IAR™
  Project/<boardname>/Applications/Cloud/AWS/EWARM/output.txt
- Keil®
  Project/<boardname>/Applications/Cloud/AWS/MDK-ARM/output.txt
- SW4STM32
  Project/<boardname>/Applications/Cloud/AWS/SW4STM32/output.txt

Figure 15 shows the example of a post-process log file.

Figure 15. Post-processing log file example



```
block size =16
number of segment :1
number of segment :4
0x8000188
0x80090bb
0x80096bc
Merging
SBSFU Base = 0x8000000
Writing header = 0x8080800
APPLI Base = 0x8080a00
Writing to C:\git\repo2\Firmware\Projects\B-L475E-IOT01A\Applications\Cloud\AWS\EWARM\\PostBuild\\\SBSFU_B-L475E-IOT01_AWS.bin 739792
"Generating the global elf file (SBSFU and userApp)"
      -------------------------------------------------------------------
                   STM32CubeProgrammer v1.1.0
      -------------------------------------------------------------------

Warning: The ELF file will be overwritten
ELF file modified succesfully
        1 file(s) copied.
```

4.11.4 Rebuilding the bootloader

The bootloader is based on the technology of the X-CUBE-SBSFU Expansion Package.

X-CUBE-AWS 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.

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 AWS sample application and to better integrate with the IDEs. The linker files (in *Linker_Common*) have been added a persistent Flash area holding the user data (credentials and other persistent data), which allows firmware update without wiping out the user configuration. Post-script templates are adapted to help with IDE integration.

Rebuilding the bootloader implies to:

1. First rebuild the Secure Engine library.
   The project is located in the *2_Images_SE_CoreBin* folder.
2. Then rebuild the Secure Boot / Secure Firmware Update executable.

The result is called “bootloader” in this document. Depending on the IDE, it is located in:

- IAR
  Project/<board name>/Applications/Bootloader_OSC2_Images_SBSFU/EWARM/<boardname>Exe/Project.out
- KEIL
  Project/<board name>/Applications/Bootloader_OSC2_Images_SBSFU/MDK-ARM/<boardname>_2_Images_SBSFU/Exe/SBSFU.axf
- SW4STM32
  Project/<board name>/Applications/Bootloader_OSC2_Images_SBSFU/SW4STM32/<boardname>_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. Please read the documentation of the X-CUBE-SBSFU package for more information.

Flashing 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 ST-LINK Utility tools.

Debugging the application

The debugger retrieves the debug information from the user application *.elf* file (the AWS sample, in the present case). It does not have access to the bootloader symbols. The debugger places a breakpoint on the `main()` function of the user application.

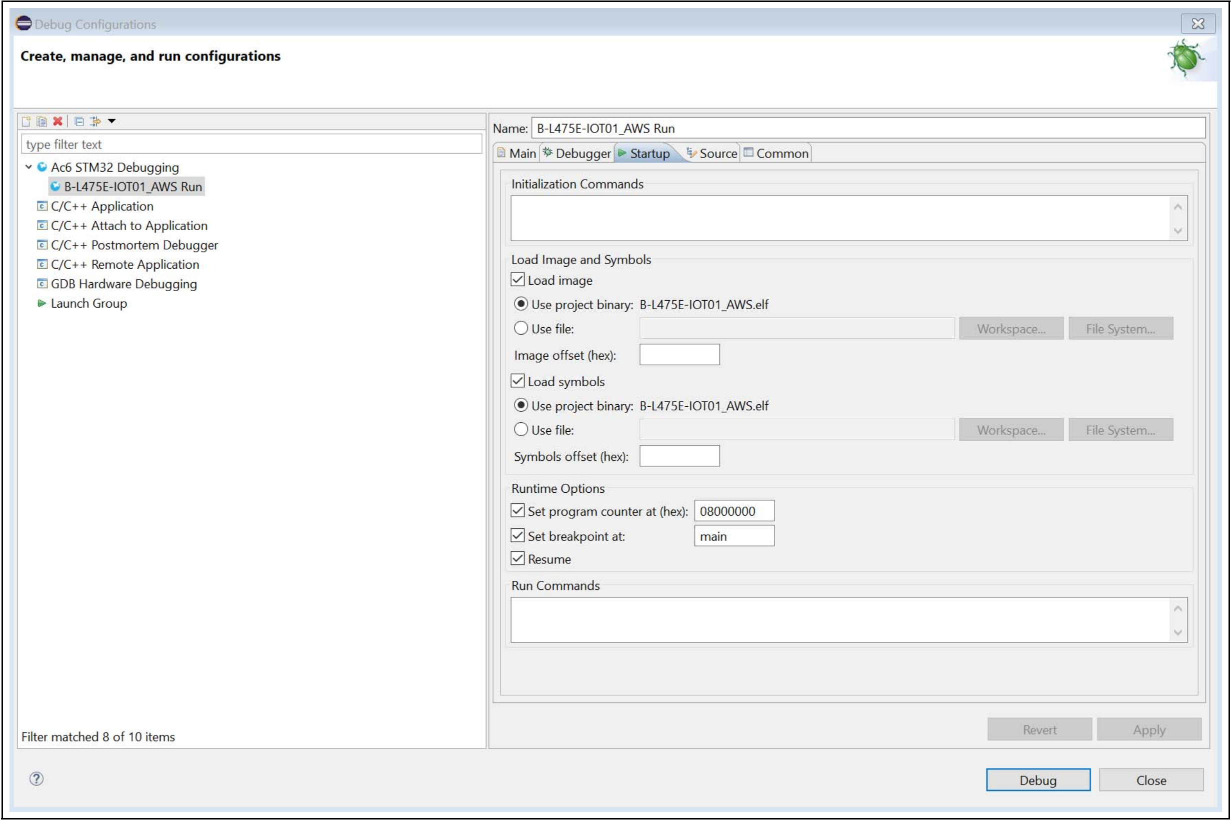
Upon reset:

- On IAR™ and Keil®, the debugger starts the application at address 0x0800 0000, so that the bootloader is executed.

  For this purpose, IAR™ is given an extra option: Debugger/extra options/command line option `--drv_vector_table_base=0x0800 0000`.
- The SW4STM32 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 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 16*

Figure 16. Program counter setting with the SW4STM32 IDE


Debug Configurations

Create, manage, and run configurations

type filter text
- Ac6 STM32 Debugging
  - B-L475E-IOT01_AWS Run
- C/C++ Application
- C/C++ Attach to Application
- C/C++ Postmortem Debugger
- C/C++ Remote Application
- GDB Hardware Debugging
- Launch Group

Filter matched 8 of 10 items

Name: B-L475E-IOT01_AWS Run

Main | Debugger | Startup | Source | Common

Initialization Commands

Load Image and Symbols
- [✓] Load image
- (●) Use project binary: B-L475E-IOT01_AWS.elf
- ( ) Use file:
- Image offset (hex):
- [✓] Load symbols
- (●) Use project binary: B-L475E-IOT01_AWS.elf
- ( ) Use file:
- Symbols offset (hex):

Runtime Options
- [✓] Set program counter at (hex): 08000000
- [✓] Set breakpoint at: main
- [✓] Resume

Run Commands

4.11.5 Firmware update

The network connectivity allows the board to download a new version of the user application, and install it by means of the SBSFU, without any connection to a development tool.

The sample application of the present package offers two means to trigger firmware download and update:

1. At init time, by typing on the UART console the HTTP or HTTPS URL of the .sbf file to download and install.
2. Or thanks to a custom AWS IoT job specifying the HTTP download URL, carried from the cloud to the device over MQTT.

The syntax for the AWS IoT job document is:

```
{
  "firmwareUpdate": "http[s]://path.to/firmware.sfb"
}
```

Here is an example using AWS S3 service for firmware storage:

- Store the *.sfb* firmware file on AWS S3 service (in an S3 bucket; for example, store *Project.sfb* in bucket "bucket").

  Make sure that this S3 bucket does not block public access and that the *.sfb* file is readable by everyone (Check the menu file > properties / permissions).
- Create a text file named *firmwareupdate.txt* with the following contents and store it on S3:

  ```
  {
  "firmwareUpdate":
  "https://s3.eu-central-1.amazonaws.com/bucket/Project.sfb"
  }
  ```
- On the AWS web portal, select the *IoT Core* service, then go to *Manage / Jobs*:
  - Click the "Create" button, then "create custom job"
  - Enter a job ID (unique alphanumeric sting)
  - Select the device you wish to update
  - Select the job file you just uploaded to S3: *firmwareupdate.txt*
  - Complete the job creation: *Next / Create*
- Once the job is created and the device is connected to AWS IoT through MQTT, the device receives the job, the application exits the MQTT loop and starts downloading the new firmware file over HTTP(S).

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

  Once the new application runs and is connected to AWS IoT again, it verifies that its version has changed since the update job was received, and notifies AWS IoT of the job status: completed, or failed.

Note about HTTP download

In order 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.

5 AWS IoT sample application behavior

Prior to running the application, the user must get his AWS IoT security credentials on the AWS web site. Refer to *Section 2.2: AWS security credential creation on page 8*.

After a startup and configuration phase, the application enters a loop, which:

- keeps the device connected with the AWS MQTT broker
- is responsive to Cloud2Device messages and AWS IoT jobs
- is responsive to user button actions

For emulating the user button actions (single push and multiple push) by a custom AWS IoT job, proceed as follows:

- Post the job file to an AWS S3 bucket
- Click on "Create" > "Create custom job" on the Manage/Jobs section of the AWS IoT console, and fill-in the form.

The contents of the job file can be either:

```
{ "myOperation": "simulate_single_push" }
```

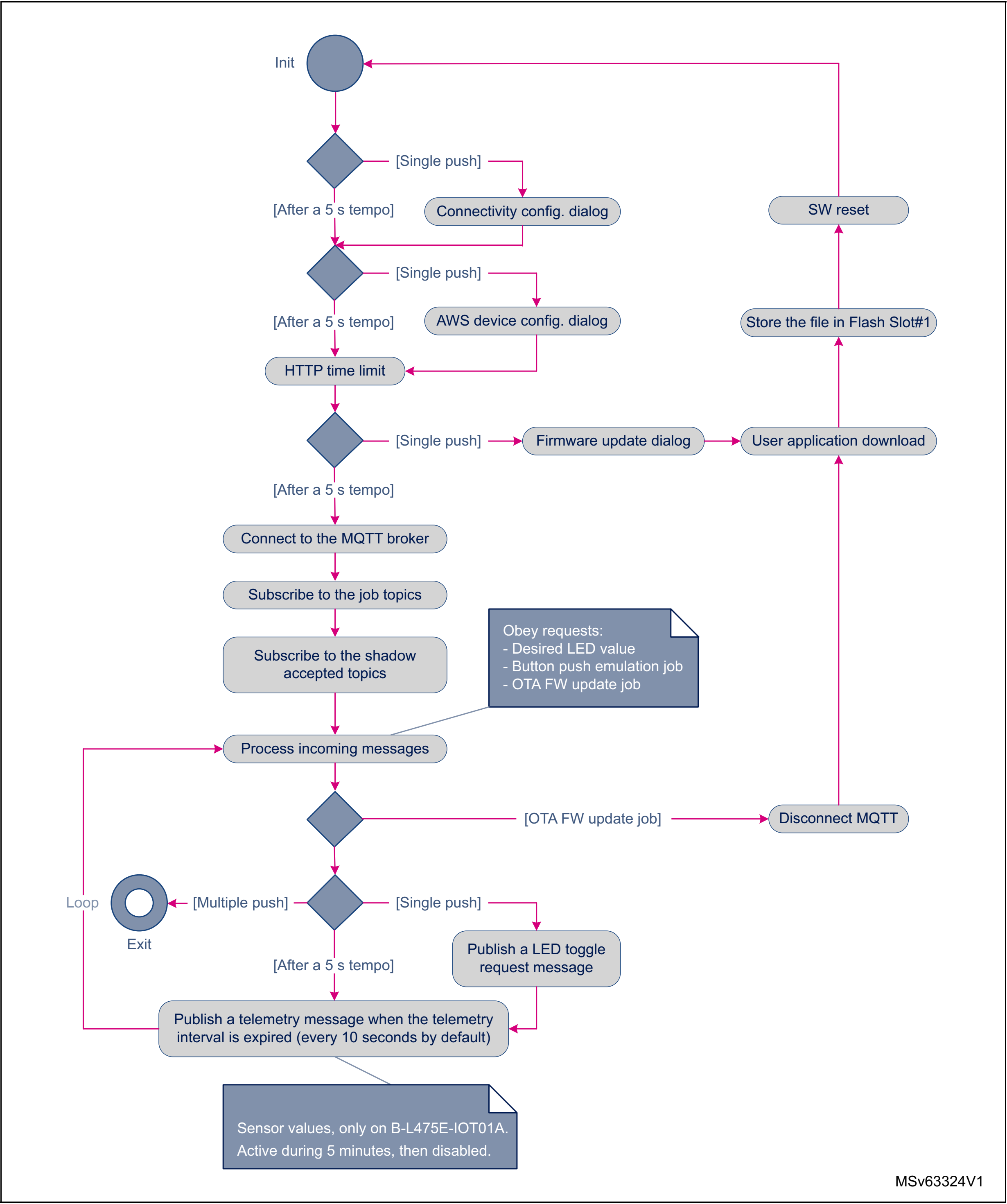
or:

```
{ "myOperation": "simulate_multiple_push" }
```

A possible application is to enroll several devices in a device group, and deploy a single-push job to the whole group. It toggles the LED of the devices almost synchronously.

The overall activity flow of the application is depicted in *Figure 17*

Figure 17. Sample application activity diagram


Init
[Single push]
[After a 5 s tempo]
Connectivity config. dialog
[Single push]
[After a 5 s tempo]
AWS device config. dialog
HTTP time limit
[Single push]
Firmware update dialog
User application download
Store the file in Flash Slot#1
SW reset
[After a 5 s tempo]
Connect to the MQTT broker
Subscribe to the job topics
Subscribe to the shadow accepted topics
Obey requests:
- Desired LED value
- Button push emulation job
- OTA FW update job
Process incoming messages
[OTA FW update job]
Disconnect MQTT
Loop
[Multiple push]
Exit
[Single push]
Publish a LED toggle request message
[After a 5 s tempo]
Publish a telemetry message when the telemetry interval is expired (every 10 seconds by default)
Sensor values, only on B-L475E-IOT01A.
Active during 5 minutes, then disabled.
MSv63324V1

6 Specificities with the B-L475E-IOT01A board

This section describes how to use the application example provided at *Projects\B-L475E-IOT01\Applications\Cloud\AWS\*.

6.1 Board capabilities

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)

6.2 Inventek module hardware interface

The Inventek module is connected to the MCU as described in *Table 4*.

Table 4. 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, push pull 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-triggered input.
ISM43362_SPI3_CSN	PE0	Output GPIO, push pull mode	The STM32 host shall set this output at 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 high when ready to communicate.
INTERNAL_SPI3_SCK	PC10	Mode SPI3 Alternate Function, push pull	SPI interface to read and write data to the Inventek Wi-Fi® module.
INTERNAL_SPI3_MISO	PC11		
INTERNAL_SPI3_MOSI	PC12		

6.3 Application behavior

In addition to cloud and interaction features, which are available on all the boards, telemetry data is published every 10 seconds during 5 minutes.

The sensor data are reported in the JSON format in such a way that the thing shadow service can be used.

The thing shadow service acts as an intermediary, allowing devices and applications to retrieve and update thing shadows. This is documented on the AWS IoT website.

1. The AWS IoT demo application subscribes to the topic *$aws/things/<the thing name which is declared in AWS and stored in Flash memory>/shadow/update/accepted*.
2. When subscribing to a topic, a MQTT callback mechanism allows the device to define the action to do when a message is received. In this example, the print to the console of the message received is implemented.
3. The board publishes a message on *$aws/things/<the thing name which is declared in AWS and stored in Flash memory>/shadow/update*.

The sensor values are published approximately every 10 seconds.

Example of a published sensor message:

```
{
 "state": {
  "reported": {
   "temperature": 27.64,
   "humidity": 38.42,
   "pressure": 995.94,
   "proximity": 8191,
   "acc_x": -15, "acc_y": 2, "acc_z": 1020,
   "gyr_x": -980, "gyr_y": -4060, "gyr_z": 1750,
   "mag_x": -140, "mag_y": 90, "mag_z": 319
  }
 }
}
```

Table 5 presents the units for the values reported by the sensors of the B-L475E-IOT01A board.

Table 5. Units for the values reported by the sensors with the B-L475E-IOT01A board

Data	Unit
Temperature	degree Celsius (°C)
Humidity	relative humidity in %
Pressure	hectopascal (hPa)
Proximity	millimeter (mm)

Table 5. Units for the values reported by the sensors with the B-L475E-IOT01A board (continued)

Data	Unit
Acceleration	milli g-force
Gyrometer	milli degree per second (mdps)
Magnetometer	milligauss (mG)

On top of sensor value publications, the LED and user button work as stated in *Section 4.10.2 on page 20*.

6.4 Running the application

The steps to run the application are listed below:

1. Get the AWS IoT security credentials on the AWS web site.
   Refer to *Section 2.2: AWS security credential creation on page 8*.
2. Make sure that the liner (which is a very thin film placed on the proximity sensor) has been removed, otherwise the 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. LED 6 (ST-LINK COM - bi color) must be lit (red) and LED 5 (5 V power) must also be lit (LED 5 is green).
6. Under the directory *Projects\ B-L475E-IOT01\Applications\Cloud\AWS\<tool chain>*, select the project, build and program the binary with the IDE (or directly drag and drop the *B-L475E-IOT01_Cloud_AWS_VX.Y.Z.bin* file, available in the application binary directory, to the board drive in Explorer).

6.5 AWS-based web dashboard

A companion AWS-based web dashboard is available to quickly evaluate the firmware package functions for an easy sensor data visualization and device control.

On the web application side:

- Open it at: *http://st-dashboard-iot-v2.s3-website-us-east-1.amazonaws.com*
- Sign-up the first time only
- Sign-in
- Create a device and save the credentials file. This information is needed for the device to connect to the web dashboard.

On the device side:

- Open the IDE preprocessor defined symbols tab and add `AWS_IOT_DASHBOARD`
- Build
- Start the board as indicated in the previous sections
- With the board console, enter the AWS credentials as follows:
  - For the server address, enter a1dyrwwtjpwo0n-ats.iot.us-east-1.amazonaws.com
  - Enter the device name with the MAC address used for the creation of the device with the web application
  - Enter the root CA (refer to *Figure 6: Middleware on page 16*).
  - For the device certificate, enter the first part of the credential file received at the creation of the device with the web application
  - For the device key, enter the second part of the credential file received at the creation of the device with the web application
- If everything went well, the following message is displayed to the board console: "Connected to a1dyrwwtjpwo0n-ats.iot.us-east-1.amazonaws.com:8883"

When the above is completed, it is possible to use the dashboard.

More information is available on line on the web application side.

7 Specificities with the 32F413HDISCOVERY board

This section describes how to use the application example provided in *Projects\STM32F413H-DISCO\Applications\Cloud\AWS\*.

7.1 Application behavior

The LED and user button work as stated in *Section 4.10.2 on page 20*.

7.2 Running the application

The steps to run the application are listed below:

1. Connect a Type A to Micro-B USB cable from the connector USB ST_LINK to a PC
2. Under the directory *Projects\ STM32F413H-Discovery\Applications\Cloud\AWS\<tool chain>*, select the project, build and program the binary with the IDE (or directly drag and drop the *STM32F413H_Discovery_Cloud_AWS_V1.0.0.bin* file, available in the application binary directory, to the board drive in Explorer)

8 Specificities with the 32F769IDISCOVERY board

This section describes how to use the application example provided in *Projects\STM32F469I-DISCO\Applications\Cloud\AWS\*.

8.1 Application behavior

The LED and user button work as stated in *Section 4.10.2 on page 20*.

8.2 Running the application

The steps to run the application are listed below:

1. Ensure that stlk is connected via a jumper
2. Connect a Type A to Micro-B USB cable from the (connector USB ST-LINK) to a PC
3. Connect an Ethernet cable to a server
4. Under the directory *Projects\STM32F769I-Discovery\Applications\Cloud\AWS\<tool chain>*, select the project, build and program the binary with the IDE (or directly drag and drop the *STM32F769I_Discovery_Cloud_AWS_V1.0.0.bin* file, available in the application binary directory, to the board drive in Explorer)

9 Specificities with the P-L496G-CELL02 pack

9.1 Board capabilities

The P-L496G-CELL02 Discovery pack allows users to develop their cellular applications with the STM32L4 Series microcontrollers based on Arm® Cortex®-M4 core.

It embeds the low-power STM32L496AGI6 microcontroller in a display-less variant of the STM32L496GDISCOVERY board, featuring 1 Mbyte of Flash memory and 320 Kbytes of SRAM, in UFBGA169 package. It also comes with an add-on board connected through the STMod+ connector.

The add-on board has:

- an LTE-IoT Cat M1/NB1/2G modem (Quectel BG96 module)
- an embedded ST InCard SIM with a MVNO profile, ready to use.
- a 32-Kbyte E2PROM, where instructions have been preloaded in factory. Users can use the factory FW and find in particular the way to activate the embedded SIM by using a console application.
- a USB port, which can be used to access the Quectel module (FW upgrade, AT commands). This USB port is also a possible additional power source.

9.2 Module hardware interface

The daughterboard is connected to the MCU through an STMod+ connector.

The control of the Quectel modem is described in *Table 6*.

Table 6. Quectel module hardware interface - Modem control

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

The SIM is selected by means of two pins as described in*Table 7*.

Table 7. Quectel module hardware interface - SIM selection

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

9.3 Running the application

The steps to run the application are listed below:

1. Retrieve the APN, username and password from the SIM provider.

   If an MVNO profile such as EMnify is in the embedded SIM: set APN to 'EM';.
   It is not needed to set a username and password.
2. Ensure that JP7 is closed (ST_LINK as a power source), SW1 (blue switch) is ON.
3. Connect a Type A to Micro-B USB cable from the STM32L496AGI6-like board (connector USB ST-LINK CN5) to a PC.
4. MCU side: LED 5 (ST-LINK COM - bi color) must be lit (red) and LED 8 (5 V power) must also be lit (LED 8 is green).
5. MCU side: LED 2 (green) then LED1 (red) must be consecutively lit
6. Cellular daughter board side: LED1 (red) must be lit (module is powered on) then LED2 (green) must blink (radio is running)
7. Under the directory *Projects\STM32L496G-Discovery\Applications\Cloud\<Cloud provider>\<tool chain>*, select the project, build and program the binary with the IDE.

10 Using an own MQTT server: Mosquitto

Alternatively to the AWS IoT cloud, the application can run with an own Mosquitto server. The AWS MQTT client and mbedTLS are compatible with Mosquitto and openssl.

To use an MQTT server, the following is needed:

- To generate keys and certificates
- A Linux host, which can be reached by the device over TCP/IP (or another OS which can run Mosquitto and openssl).

In the example application, the mbedTLS configuration is tailored to the AWS server TLS configuration. This sets some constraints on the TLS credentials that Mosquitto and the device are able to use.

The openssl utility (available on multiple platforms, including Linux and Cygwin - the below commands work with openssl 1.0.2) allows the user to create the proper keys and certificates:

1. Create a root certification authority and use it to generate the certificates for the Mosquitto server and for the device:

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

   This produces the files: ca.key, ca.crt

2. Create private key for Mosquitto server, certificate request from server key and certificate for the server

   a) Create a private key for the Mosquitto server:

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

   This produces the file: server.key

   b) Create a certificate request from the server key:

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

   This produces the file: server.csr

*Note: It is important that the Common Name (CN) that is set in the certificate request matches the hostname of the host where Mosquitto runs. Otherwise, the server certificate verification fails and the AWS MQTT client does not connect. For instance, if Mosquitto runs at myiothost.st.com, the Common Name 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 mqttInitParams.isSSLHostnameVerify = false before initializing the MQTT client by aws_iot_mqtt_init().*

   c) Create the certificate for the server, 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 the files: server.crt, ca.srl

3. Create private key for the device, certificate request from device key and certificate for the device
   a) Create a private key for the device:

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

   This produces the file: client.key

   b) Create a certificate request from the device key:

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

   This produces the file: client.csr

   c) Create the 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 the files: client.crt, ca.srl

Then, on server side, the server credentials must be passed to Mosquitto through the 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
require_certificate true
```

The server is launched on the internet host (for example a Linux virtual host):

```
mosquitto -v -c mosquitto.conf
```

Finally, on device side:

- Set the security credentials through the terminal as in AWS case, but use the files ca.crt, client.crt and client.key.
- Set the server address to the address of the host where Mosquitto runs. The device name does not matter.
- Run the application.

11 Memory footprint

SBSFU imposes some specific constraints to the memory placement of the application:

- ROM code and data must be placed in the `SLOT0` region, starting at a specific offset (0x200 for STM32F4 and STM32L4 devices, 0x400 for STM32F7 devices).
- ROM section size must be multiple of 16 bytes (specific rules in linker files)
- RAM data must be placed in `SB_SRAM` region
- "vector_start" section must be placed at beginning of `SLOT0` region + offset
- A specific section is used to store user configuration that must persist to a firmware update operation. This is mapped to an area unknown to SBFU.

The definitions of `SLOT0` and `SB_SRAM` are specified in the SBSFU linker common files (*Project/<board name>/Applications/BootLoader_OSC/Linker_Common*).

These section definitions are mostly imposed by the internal Flash sector structure and SBSFU internal constraints. Refer to the AN5056 application note in *Section 1.2: References on page 7*.

Upon firmware update, the new firmware image is downloaded to the SLOT#1 region over the network. The Secure Firmware Update relies on a Flash swap area to perform installation in the SLOT#0 region. The old firmware is backed up in SLOT#1 to enable rollback. *Table 8* shows the sector and slot sizes depending on the target device.

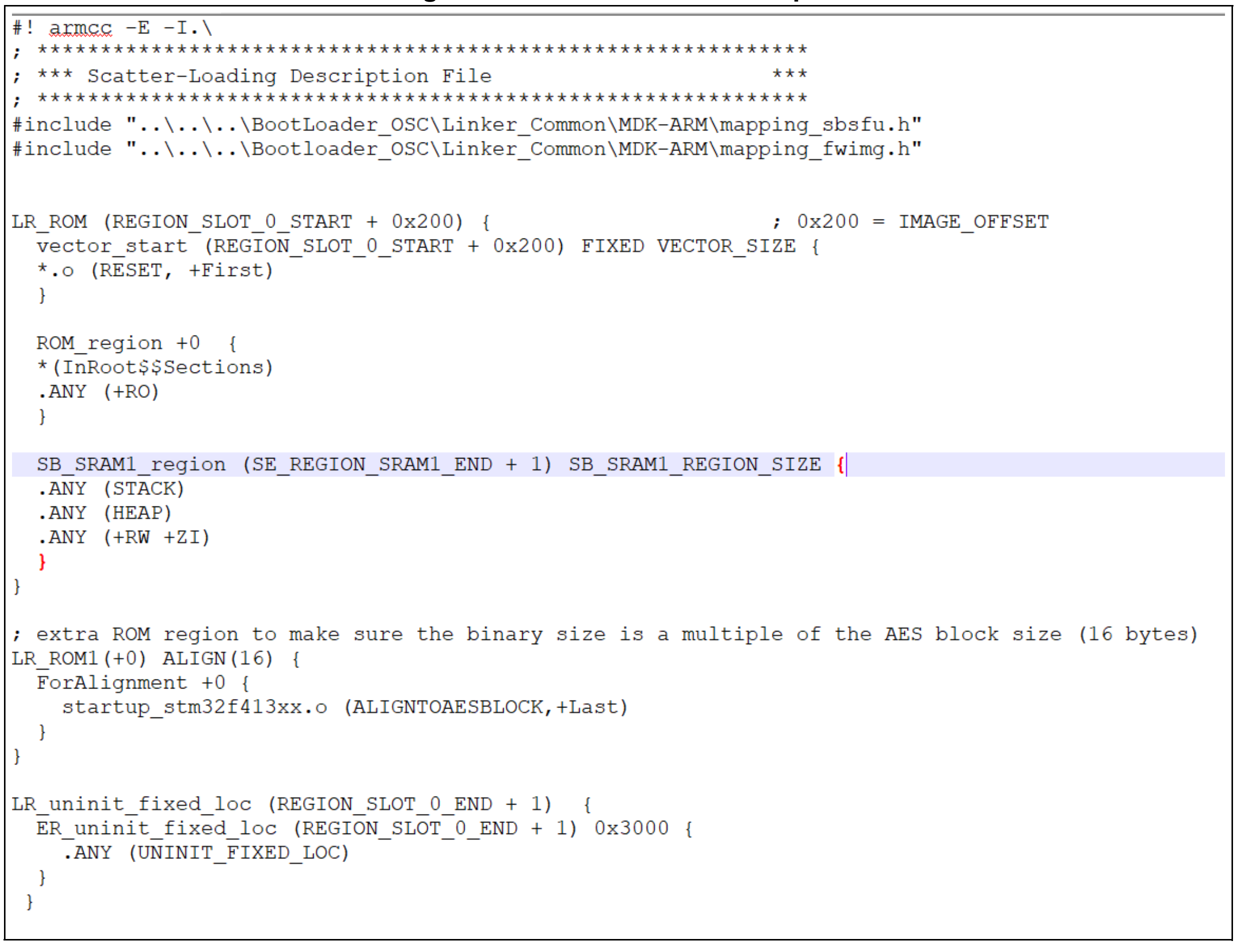
The application code size is limited by the slot size.

Table 8. Memory footprint values

Board	Sector size (Kbyte)	Sectors per slot	Slot size (Kbyte)
P-L496G-CELL02	16	27	432
B-L475E-IOT01A	8	55	440
32F413HDISCOVERY	128	4	512
32F769IDISCOVERY	256	3	768

An example of STM32F413 Keil® AWS linker file is shown in *Figure 18*.

Figure 18. AWS linker file example.



```
#! armcc -E -I.\
; *************************************************************
; *** Scatter-Loading Description File                     ***
; *************************************************************
#include "..\..\..\BootLoader_OSC\Linker_Common\MDK-ARM\mapping_sbsfu.h"
#include "..\..\..\Bootloader_OSC\Linker_Common\MDK-ARM\mapping_fwimg.h"


LR_ROM (REGION_SLOT_0_START + 0x200) {                      ; 0x200 = IMAGE_OFFSET
  vector_start (REGION_SLOT_0_START + 0x200) FIXED VECTOR_SIZE {
  *.o (RESET, +First)
  }

  ROM_region +0  {
  *(InRoot$$Sections)
  .ANY (+RO)
  }

  SB_SRAM1_region (SE_REGION_SRAM1_END + 1) SB_SRAM1_REGION_SIZE {
  .ANY (STACK)
  .ANY (HEAP)
  .ANY (+RW +ZI)
  }
}

; extra ROM region to make sure the binary size is a multiple of the AES block size (16 bytes)
LR_ROM1(+0) ALIGN(16) {
  ForAlignment +0 {
    startup_stm32f413xx.o (ALIGNTOAESBLOCK,+Last)
  }
}

LR_uninit_fixed_loc (REGION_SLOT_0_END + 1)  {
  ER_uninit_fixed_loc (REGION_SLOT_0_END + 1) 0x3000 {
    .ANY (UNINIT_FIXED_LOC)
  }
 }
```

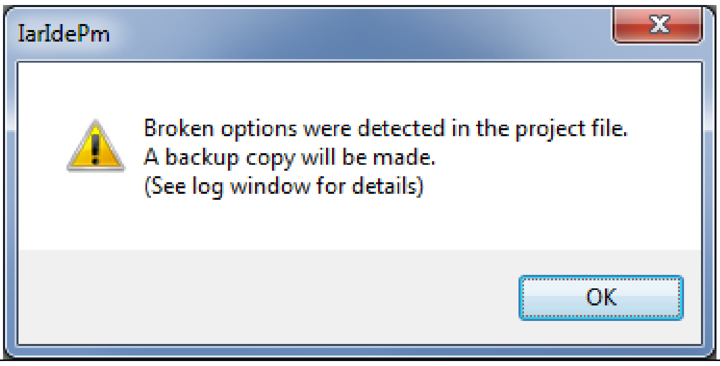
12 Frequently asked questions

Q: Do I need an account to use AWS IoT?

A: Yes

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

Figure 19. Pop-up when the IAR IDE version is not compatible with the one used for X-CUBE-AWS



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 (see *Section 4.1: Description*), 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: The file that implements code for publishing and subscribing is common to the three platforms and is called *subscribe_publish_sensor_values.c.* This file can be customized to change the way to subscribe or publish.

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: My device does not connect AWS IoT. How shall I proceed?

A: The reason to this can be that credentials have been wrongly entered. Enter them again by pressing the user button (blue) up to five seconds after board reset.

On the AWS IoT side, the reason can be that:

- A policy may not be defined or the policy is not attached to the certificate
- A thing may not be defined or the thing is not attached to the certificate

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.

13 Revision history

Table 9. Document revision history

Date	Revision	Changes
29-Mar-2017	1	Initial release.
2-Jul-2019	2	Document entirely updated for: – Support of the Secure Firmware Update using X-CUBE-SBSFU – Integration of X-CUBE-CELLULAR and Connectivity middleware with the support of LTE Cat M1/NB modem with 2G fallback – Connection to STMicroelectronics AWS web dashboard