UM2178
User manual

Getting started with X-CUBE-AWS
Amazon web services IoT software expansion for STM32Cube

Introduction

This user manual describes the content of the STM32 AWS (Amazon web services) IoT (Internet of Things) software expansion package for STM32Cube.

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

The STM32 AWS IoT software expansion package (X-CUBE-AWS) for STM32Cube provides application examples that connect and subscribe to the AWS IoT service via MQTT in order to receive information and to publish data.

X-CUBE-AWS is available for B-L475E-IOT01, 32F413HDISCOVERY and 32F769IDISCOVERY boards.

The X-CUBE-AWS features are as follows:

- Ready to run firmware example using WiFi® and Ethernet connectivity to support quick evaluation and development of IoT cloud applications
- Interface to configure the board to connect to the AWS IoT
- WiFi® connection
- AWS IoT connection, subscribe and publish tasks
- Specific features on the B-L475E-IOT01 board:
  - Measurement of humidity, temperature, 3-axis magnetic data, 3D acceleration, 3D gyroscope data, atmospheric pressure and proximity
  - AWS private key protection with firewall
  - Remote firmware update

Contents

List of tables

List of figures

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
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

2 Amazon web service IoT

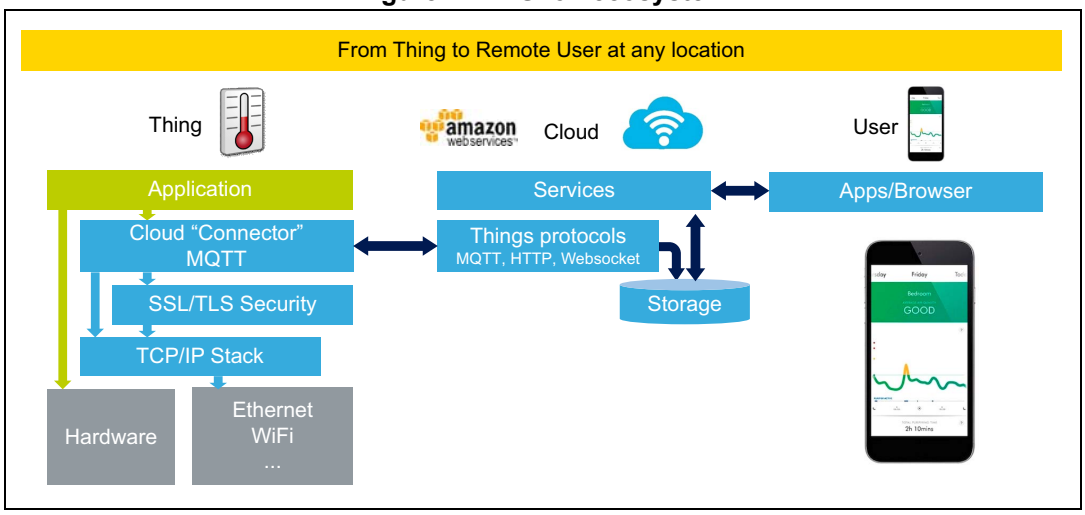
This section introduces AWS IoT.

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) provides on-demand computing resources and services in the cloud. AWS IoT is the 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, use "access type = programmatic access" and assign the users to the group that has been created
4. 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 have to attach the thing and policy to the certificates
5. Each user must individually go through the following steps:

- Log on to the AWS IoT service
- Create his thing in "Registry>Things"
- Create his certificate in "Security>Certificates"
- Create his policy in "Security>Policies" (see the *Note: The default policy used for our application software:* at the end of this list)
- Return to the certificate and attach his thing and policy
- Save on his PC the artifacts that are needed for flashing the board. Those artifacts are:
  - Certification authority certificate
  - Thing name
  - Thing certificate
  - Thing private key
  - Endpoint. This is the hostname of the MQTT broker; it looks like: a27xxxx.iot.<region>.amazonaws.com.

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)

The AWS main account can be used 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 *5.* listed in *Section 2.2.1*, but with the AWS main account.

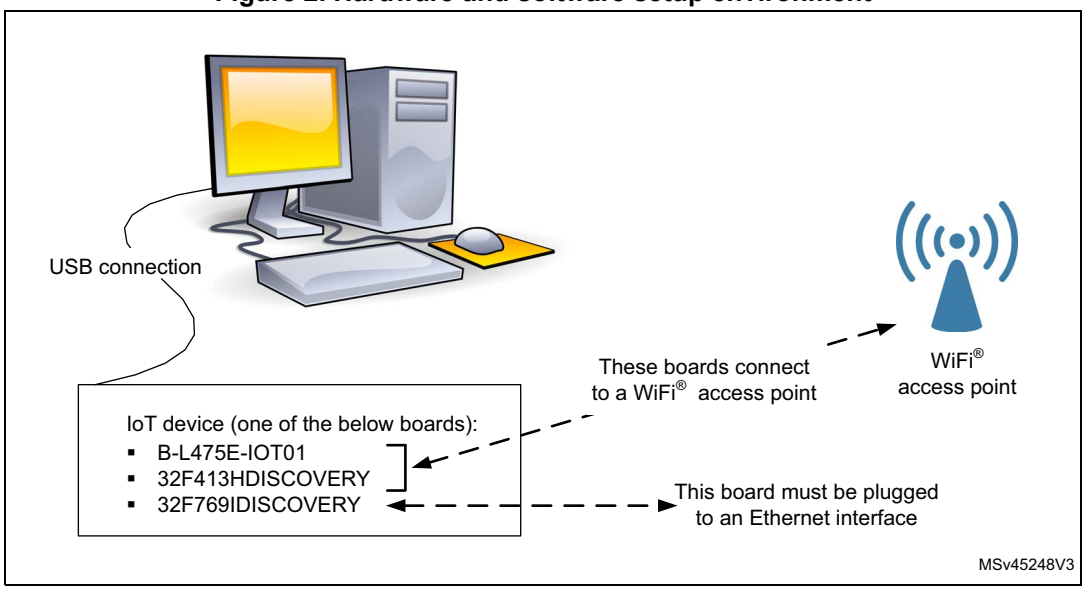
3 Hardware and software environment setup

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

- Flash the board
- Store the WiFi® and the AWS security credentials
- Interact with the board via a UART console
- Debug

The B-L475E-IOT01 or 32F413HDISCOVERY boards must be connected to a WiFi® access point. The 32F769IDISCOVERY board must be connected to an Ethernet interface (see *Figure 2*).

Figure 2. Hardware and software setup environment



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
- WiFi drivers
- Ethernet driver for the 32F769IDISCOVERY board
- Sensor drivers for the B-L475E-IOT01 board
- STM32L4 Series, STM32F4 Series and STM32F7 Series HAL
- AWS application examples

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

The following Integrated development environments are supported:

- IAR Embedded Workbench for ARM® (EWARM).
  IAR version 7.80.4 or higher must be used
- Keil Microcontroller Development Kit (MDK-ARM)
- System Workbench for STM32. Version 1.14.0 or higher must be used

Note that the Keil and the System Workbench projects present the following limitations:

- They cannot directly create an image file (which may be installed through RFU). They would require a utility which would translate a hex image file into an IAR simple-code image file. Although not excessively complex, it is not included in the present version of the package.
- The firewall build is not ported to those compilation suites yet.

4.2 Architecture

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

The X-CUBE-AWS software is an expansion for the STM32Cube, its main features and characteristics are:

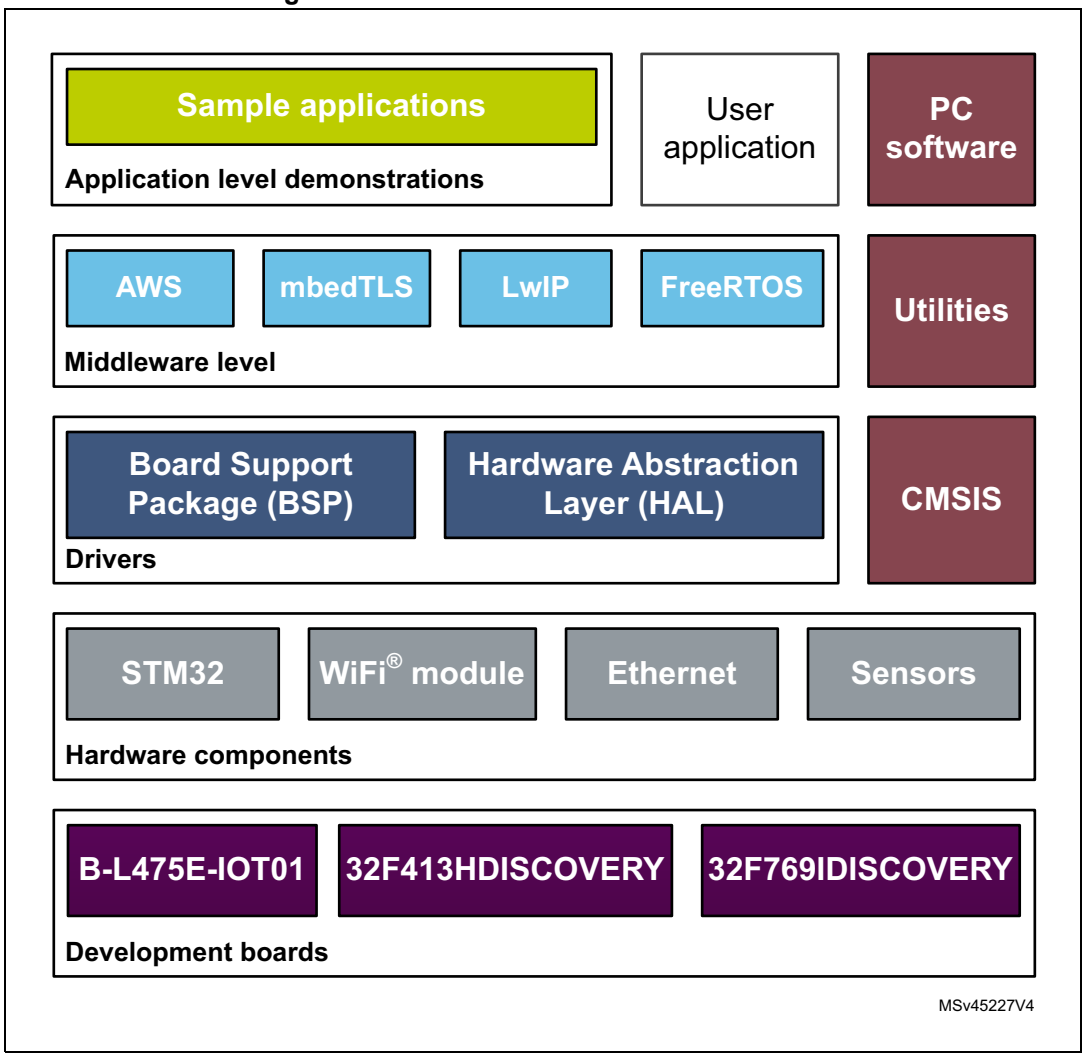
- It is fully compliant with the STM32Cube architecture
- It expands the STM32Cube in order to enable the development of applications accessing and using the AWS IoT
- It is based on the STM32CubeHAL, which is the hardware abstraction layer for the STM32 microcontrollers

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 the user button.
- **AWS middleware**: MQTT client.
- **mbedTLS**: the AWS middleware uses a TLS connection which is ensured by the mbedTLS library.
- The TCP/IP connection can be handled either by the WiFi module (when a WiFi connection is being used) or by the LwIP middleware (when an Ethernet connection is being used). In the X-CUBE-AWS package, only the 32F769IDISCOVERY board can connect via Ethernet.
- FreeRTOS is a real-time operating system used by LwIP.

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

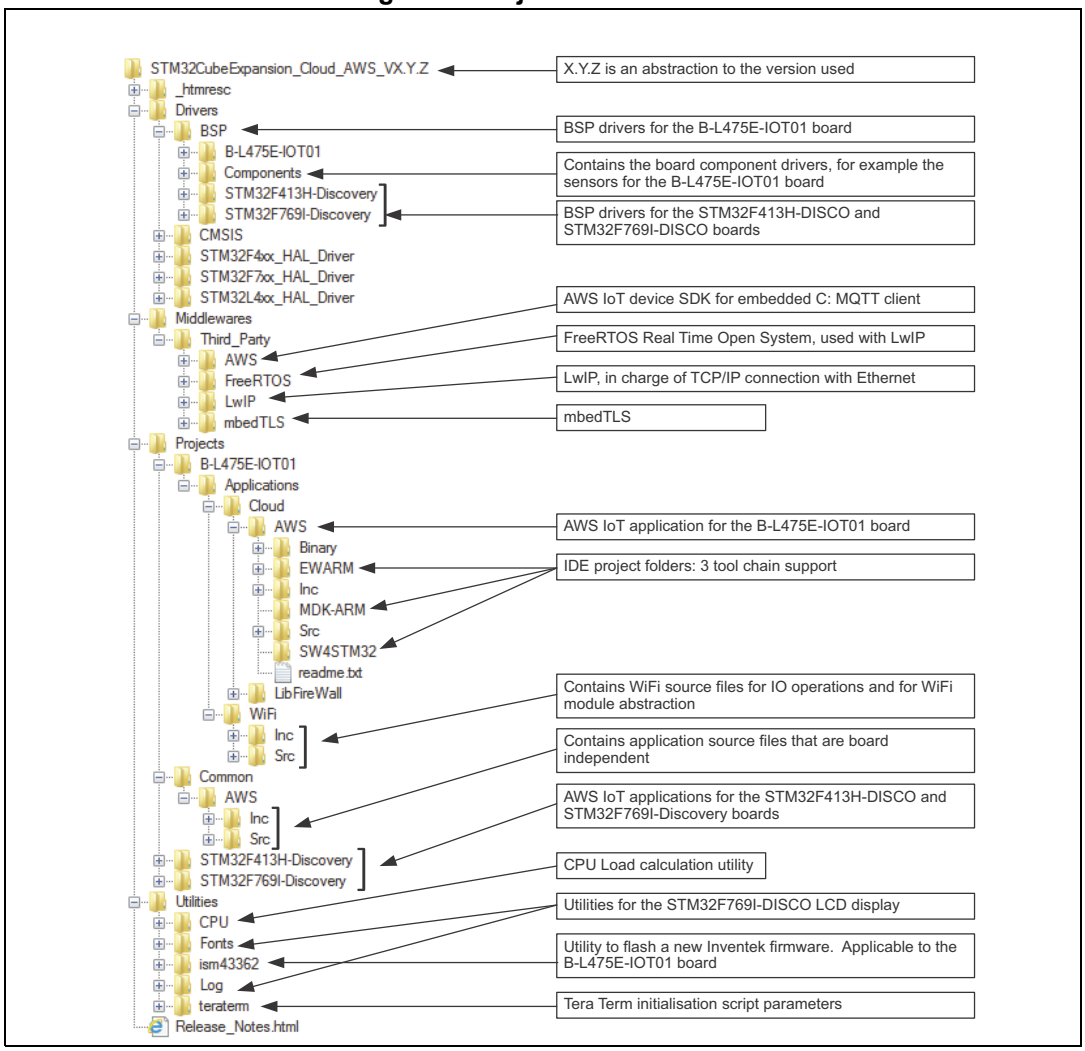
Figure 3. X-CUBE-AWS software architecture



4.3 Folder structure

Figure 4 presents the folder structure of the X-CUBE AWS package.

Figure 4. Project file structure



4.4 WiFi® components

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

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:

- To configure the WiFi and AWS security credentials. This can be done from the time that the board starts up and up to 5 seconds after that.
- To publish messages for the desired LED status when the board is connected to AWS IoT.
- To enter the firmware management dialog: RFU and bank boot switch. There is a period window during the board initialization phase where this can be done. This period is displayed on the UART console.

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

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

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 that is used by the applications is done via the board support package (BSP) functions.

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

Using the BSP button functions with the 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.amazon.com web server.

The user can use the HAL_RTC_GetTime() function to get the time.

This function can be used to time stamp some messages for example.

4.9 mbedTLS configuration

The mbedTLS middleware support is fully configurable thanks to a #include configuration file.

The name of the configuration file can be overridden thanks to the MBEDTLS_CONFIG_FILE #define.

The X-CUBE-AWS package uses “aws_mbedtls_config.h” for project configuration.

This is implemented by having this below # directives at the beginning of the mbedTLS.c and 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             256 /**< Maximum number of bytes
for usable MPIs. *///
#define MBEDTLS_ECP_MAX_BITS             521 /**< 521 Maximum bit size of
groups */
#define MBEDTLS_ECP_WINDOW_SIZE            2 /**< Maximum window size used
*/
#define MBEDTLS_ECP_FIXED_POINT_OPTIM      0 /**< Disable fixed-point speed-
up */
#define MBEDTLS_SSL_MAX_CONTENT_LEN    4096 /**< 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 our IoT device. This way to configure is independent of the platforms

Section 5, *Section 6* and *Section 7* detail the specificities for the B-L475E-IOT01, 32F413HDISCOVERY and 32F769IDISCOVERY boards respectively.

4.10.1 General description

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

- B-L475E-IOT01 supports WiFi 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).
- 32F413HDISCOVERY supports WiFi connectivity with an on board Inventek module.
- 32F769IDISCOVERY natively provides an Ethernet interface.

For those three platforms, a sample application supports below features:

- Configures the board with the WiFi parameters (if WiFi is used) and AWS security credentials
- Connects to the Internet
- Connects to AWS
- Publishes MQTT messages and subscribes to topics, depending on the board capabilities as described in *Table 2* below.

Table 2. Published messages and subscribe actions by the applications

-	B-L475E-IOT01	32F413HDISCOVERY and 32F769IDISCOVERY
Published information	Message to toggle the LED: “desired state = ON” when the LED is OFF and vice versa. A message is published each time that the user button is pressed. The sensor data is published around every 10 seconds.	Message to toggle the LED: “desired state = ON” when the LED is OFF and vice versa. A message is published each time that the user button is pressed.
Subscribe actions	The MQTT subscribe callback prints the received message. On top, if the message received deals with the LED, the MQTT subscribe callback sets the LED in the desired state.	The MQTT subscribe callback prints the received message and sets the LED in the desired state.

All the boards subscribe to the $aws/things/<YourThingNameDeclaredInAWSAndStoredInFlash>/shadow/update/accepted topic and publish to the $aws/things/<YourThingNameDeclaredIn AWSAndStoredInFlash>/shadow/update topic, regardless of the type of information (LED or sensor data).

4.10.2 LED behavior of the three platforms

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

When the initialization sequence is completed, pressing 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.

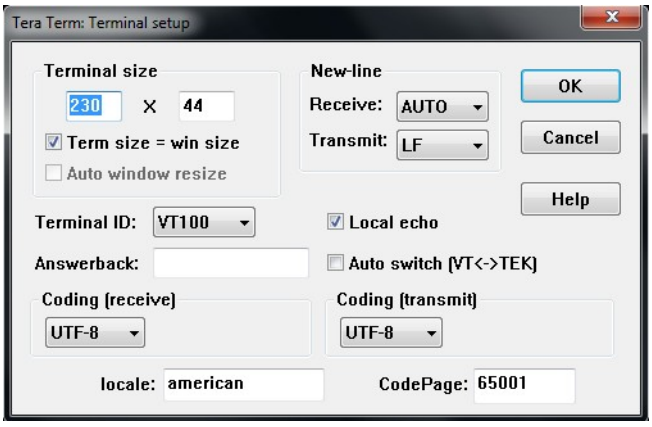
A Tera Term initialization script is provided in the package utility directory (see *Figure 4*); this script sets the correct parameters. To use it, open Tera Term, select “Setup” then “Restore setup…”.

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 5*, which shows the terminal setup and the “New-line” recommended parameters.

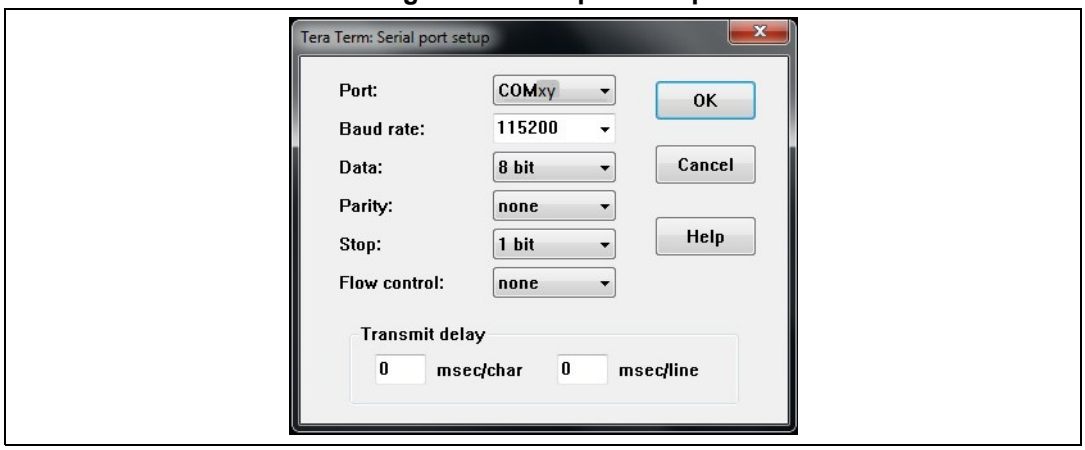
The virtual terminal new-line transmit configuration must be set to LineFeed (\n or LF) in order to allow the copy-paste from the UNIX type text files. The “Local echo” option makes copy-paste visible on the console.

Figure 5. UART terminal setup



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 6*

Figure 6. Serial port setup



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

Follow the indications on the UART terminal to upload WiFi and AWS data. Those data remain in Flash and are reused the next time the board boots.

4.10.4 WiFi and AWS security credential update

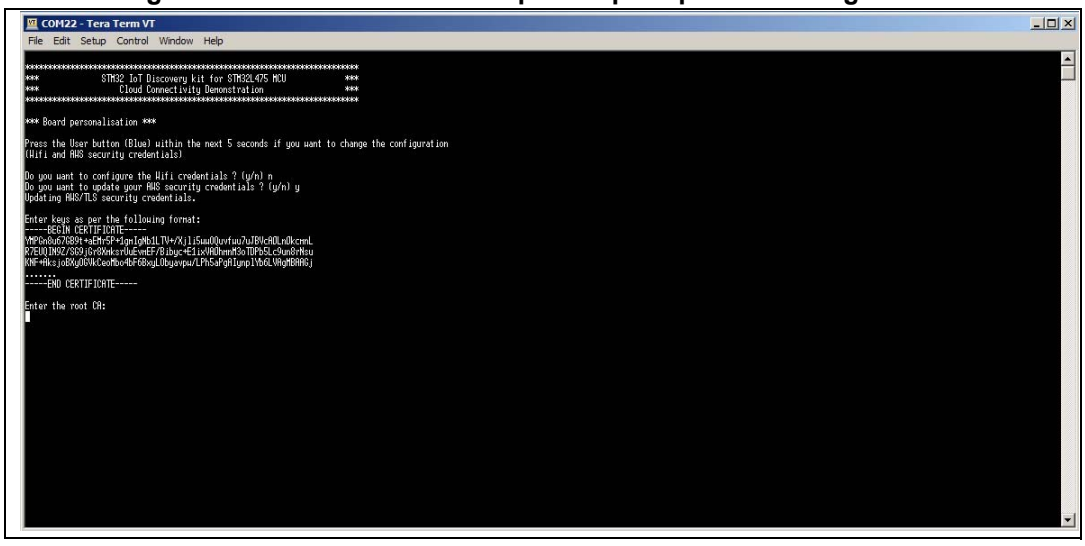
WiFi and AWS security credentials can be entered or updated in two ways:

- By pressing the user button (blue) within 5 seconds after the board start
- If the credentials are not entered in Flash or if the Flash is erased, the console prompts for the WiFi and AWS security credentials

Figure 7, *Figure 8* and *Figure 9* show an example of an AWS root certificate update.

In *Figure 7*, 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 7. AWS root certificate update - prompt for entering root CA



In *Figure 8*, the user copies his root CA that he has previously saved and pastes it into the console.

Figure 8. AWS root certificate update - copy paste root CA

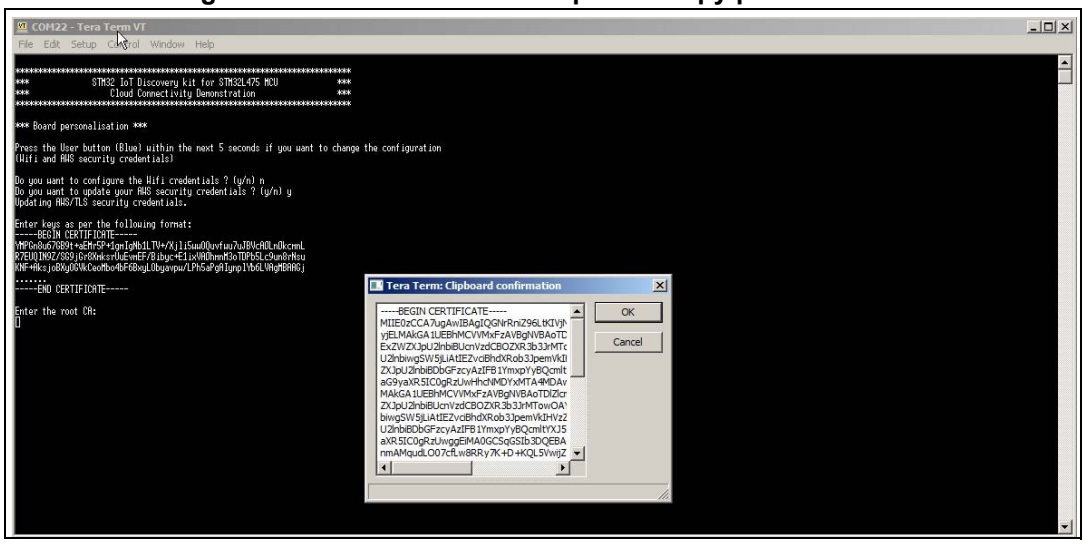


Figure 9 shows the acknowledgment from the device after “read: --->”. This corresponds to what is saved in the Flash memory.

Figure 9. AWS root certificate update - console feedback

5 AWS cloud demo with the B-L475E-IOT01 board

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

5.1 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)

5.2 Inventek module hardware interface

The Inventek module is connected to the MCU as described in *Table 3*.

Table 3. 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 trigged 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 WiFi module.
ISM43362_DRDY_EXTI1	PE1	Input GPIO, interrupt mode when rising	The Inventek module sets this pin high when ready to communicate.

Table 3. Inventek module hardware interface (continued)

Name	Pin	Type	Comment
INTERNAL_SPI3_SCK	PC10	Mode SPI3 Alternate Function, push pull	SPI interface to read and write data to the Inventek WIFI module.
INTERNAL_SPI3_MISO	PC11		
INTERNAL_SPI3_MOSI	PC12		

5.3 Application behavior

The subscribing and publishing steps for the sensor code are listed below.

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 4 presents the units for the values reported by the sensors of the B-L475E-IOT01 board.

Table 4. Units for the values reported by the sensors with the B-L475E-IOT01 board

Data	Unit
Temperature	degree Celsius (°C)
Humidity	relative humidity in %
Pressure	hectopascal (hPa)
Proximity	millimeter (mm)
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 17*.

5.4 Running the application

The steps to run the application are listed below:

1. Get the AWS IoT security credentials on the AWS web site.
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) should be lit (red) and LED 5 (5 V power) should 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 flash 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).

5.5 Key protection with the firewall

Optionally, a key protection via the firewall can be activated on the B-L475E-IOT01 board. By default, this feature is not used in the project. This section describes how the firewall works and how to use it.

The firewall hardware unit is an additional protection system offered by microcontrollers of the STM32L0 Series and STM32L4 Series. The example project takes advantage of this feature to protect the AWS device private key. This key is used by the mbedTLS middleware when connecting and identifying the AWS server.

The firewall feature is used to protect some parts of code or data (inside the Flash or the SRAM memories) from any attack intended to dump the code or to get visibility of the associated sensitive data.

Each illegal access to these protected areas is detected by the firewall hardware unit which generates a reset, kicking off any intrusion. The firewall hardware unit can protect three distinct configurable areas (more commonly called segments):

- Code segment located in the Flash or in the SRAM memories
- Non-volatile data segment located in the Flash memory
- Volatile data segment located in the SRAM memory

Each of these segments may be accessed by the CPU when the firewall is open (allowed access types depend on the segment in which the access is performed). While the firewall is closed, there is no possible access to these protected segments. Each access triggers a reset. A specific sequence called “call gate” needs to be executed to open the firewall.

The “call gate” sequence is the single entry point to open the firewall and unlock the accesses to the protected code and data areas (protected code execution as well). Once the critical code has been executed, coming back from the call gate closes the firewall.

5.5.1 Firewall project

The firewall mechanism compares instruction and data addresses with some preprogrammed sections. Some functions have to be duplicated because they are used in normal mode and in firewall mode. It is the case of a low level library (bignum.c) used in the mbedTLS middleware. These low levels functions perform bit operations required by cypher and hash operations.

The simplest way to manage the firewall is to create two different projects:

- Main project named “*Project*”
- Firewall project named “*FireWallLib*”

The FirewallLib project contains the sensitive functions and data. The entry point is a dedicated function, named *FireWallCallGate*. To match the firewall hardware constraints, the function is placed at the beginning of the code segment + 4-byte offset. The output of the project is not a library but an executable binary file.

The main project holds the non-sensitive code and the different middleware. The main project is linked with the firewall binary file to create a single output binary file with the complete application code.

5.5.2 Firewall Services

There is a single entry point to the firewall. A switch / case statement gives access to the different functionalities. Arguments are passed and retrieved thanks to the *va_args* mechanism.

- FIREWALL_INIT_FUNC: performs the initialization for the FireWallLib. It consists in copying some data from ROM to RAM (lib startup) and in programming the firewall

address range register. The main program has then to write a single register to really activate the firewall protection.

- FIREWALL_PK_PARSE_KEY_FUNC: parses and checks a private key file in PEM string format and creates the associated context. It calls the mbedTLS library function *mbedtls_pk_parse_key(pk,key,keylen,pwd,pwdlen*).
- FIREWALL_SIGN_FUNC: computes a signature using the private key context.
- FIREWALL_CTX_FREE_FUNC: frees the dedicated memory allocation to the private key context.
- FIREWALL_CANDO_FUNC: queries the key context to check if a specific cypher is supported or not.
- FIREWALL_HEAP_STAT_FUNC: gets the maximum heap and stack level used by the firewall library, useful for sizing memory needs. The memory requirements strongly depend on mbedTLS configuration.
- FIREWALL_FLASH_KEY_FUNC: the private key is provided as a PEM string. This string is flashed to a firewall protected NVM area. The flashing operation has to write and read this protected NVM area.

The firewall-protected software uses a dedicated runtime stack and a dedicated heap. Both of these data segments are protected by the firewall. When entering / leaving the firewall-protected software, the stack pointer is switched to / from the protected stack (see assembly file *stackswitch.c*, Thumb code is assumed).

Heap stack and heap monitoring can be disabled / enabled thanks to the “HEAP_DEBUG” preprocessor directive defined in the project options. Having defined HEAP_DEBUG increases a bit the HEAP size because it also implements leak detection (see file *heap.c*).

Flashing the NVM memory depends on the *HAL_GetTick* function for timeout detection. Under firewall, interruptions are masked, it is not possible to have a valid tick. Therefore, the *GetTick* function is locally stubbed, and the possible timeouts cannot be detected.

```
uint32_t HAL_GetTick(void)
{
  return 0;
}
```

5.5.3 Firewall wrapper

A wrapper is used to reach the firewall functionalities. The wrapper manages interruptions. The function pointer *FireWallCallGatePtr* points to the firewall entry point.

Extract from the firewall.h file showing the fire wall call gate prototype:

```
extern int (*FireWallCallGatePtr)(funcid_t funcid,...)
```

Extract from the firewall_wrapper.c file showing how the function parsing the key has to be implemented to pass the firewall call gate:

```
int mbedtls_firewall_pk_parse_key(  mbedtls_pk_context *pk,
                  const unsigned char *key, size_t keylen,
                  const unsigned char *pwd, size_t pwdlen )
{
    int ret;
   __disable_irq();
```

```
    ret =
(*FireWallCallGatePtr)(FIREWALL_PK_PARSE_KEY_FUNC,pk,key,keylen,pwd,pwdlen
);
    __enable_irq();
    return ret;
}
```

The development strategy is to minimize the changes in the third part middleware such as mbedTLS. The mbedTLS middleware relies on a specific structure named mbedtls_pk_context.

This structure is an interface to the public key cryptosystem and provides primary functionalities:

- Encrypt
- Decrypt
- Sign
- Others

The structure is split in two parts:

- pk_info contains a list of function pointers to the primary functionalities. This structure depends on the type of the selected public Key algorithm.
- pk_ctx contains the real key information, which must be protected.

Extract from the pk.h file, which is the public key abstraction layer interface:

```
typedef struct
{
    const mbedtls_pk_info_t *   pk_info; /**< key informations */
    void *                      pk_ctx;  /**< key context */
} mbedtls_pk_context;
```

The modification for firewall consists in having the *pk_ctx* structure filled and allocated by code running under the firewall. For this purpose, the function *mbedtl_pk_parse_key* has to be run under firewall while parsing the private key (see file *network_st_wrapper.c*).

This function returns the *pk_info* and the *pk_ctx* data, allocated in the firewall-protected memory area.

The *pk_info* structure must point to the firewall wrapper entry points. In case of TLS protocol, only a few functions are needed. The *pk_info* structure is replaced by a locally managed structure initialized as below; the *xxx_func* are not used. The functions *sign_wrap()* and *can_do()* are redirected to their counterpart to reach the firewall (see file *mbedtls_patch.c* for more details).

In network_st_wrapper.c the pk_info structure is overwritten by the mbedtls_firewall_info.

Extract from the mbedtls_patch.c file:

```
mbedtls_pk_info_t mbedtls_firewall_info = {
    MBEDTLS_PK_NONE,
    "FIREWALL",
    xxx_get_bitlen,
    firewall_can_do,
    xxx_verify_wrap,
    firewall_sign_wrap,
```

```
    xxx_decrypt_wrap,
    xxx_encrypt_wrap,
    xxx_check_pair_wrap,
    xxx_alloc_wrap,
    firewall_free_wrap,
    xxx_debug,
};
```

Firewall linker configuration file specificities

The firewall project is built as a standalone executable. It uses a different address range than the main project. By design, it does not depend on the main project. The firewall function entry point must be placed at a 4-byte offset from the code segment origin. Linker control statements are added to the default *.icf* file.

In the FireWallLib.icf file, we find the below settings:

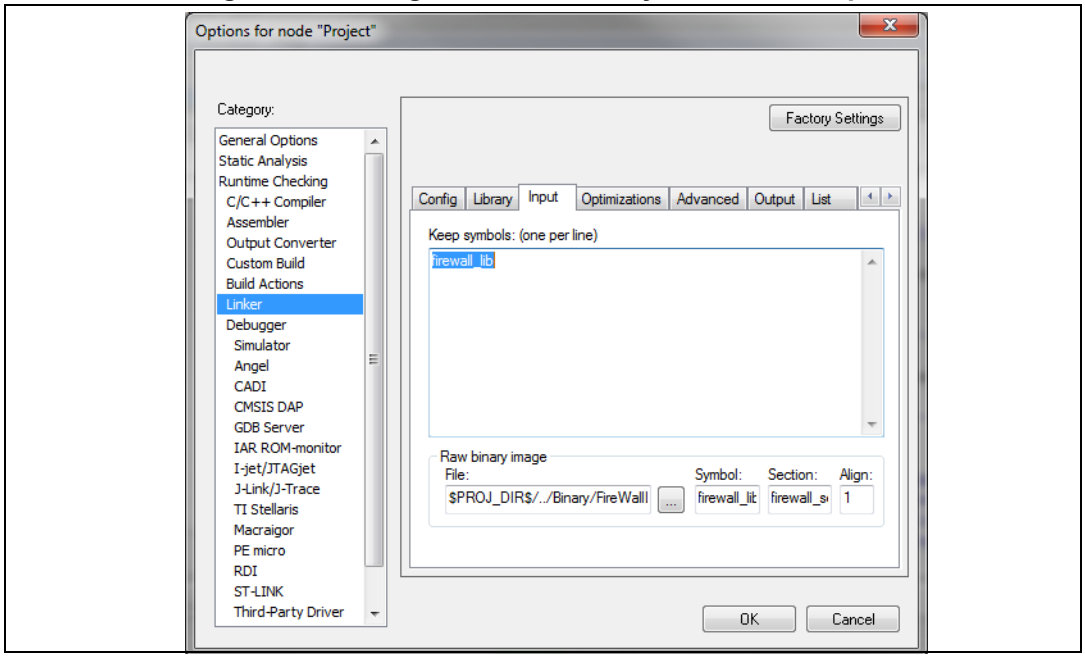
```
define exported symbol __ICFEDIT_region_ROM_start__    = 0x08066D00;
define symbol __firewall_start__ = __ICFEDIT_region_ROM_start__ + 4 ;
place at address mem:  __firewall_start__ { ro code object firewall.o  };
```

The FirewallGate function must be the first one in the source file "*firewall.c*".

The binary output file is used by the main project and must be loaded at a predefined address. This is achieved thanks to the linker input menu and the linker file.

Figure 10 shows how to add the firewall binary to the project with the IAR IDE.

Figure 10. Adding the firewall binary to the linker input



The linker file (*.icf*) forces the placement of the section *firewall_section* at the specific address corresponding to the one used in FireWallLib project.

In the stm32l475xx_flash.icf file, we find the below settings:

```
Define exported symbol __firewall_ROM_start = 0x08066D00;
```

```
place at address mem:__firewall_ROM_start { readonly section
firewall_section };
```

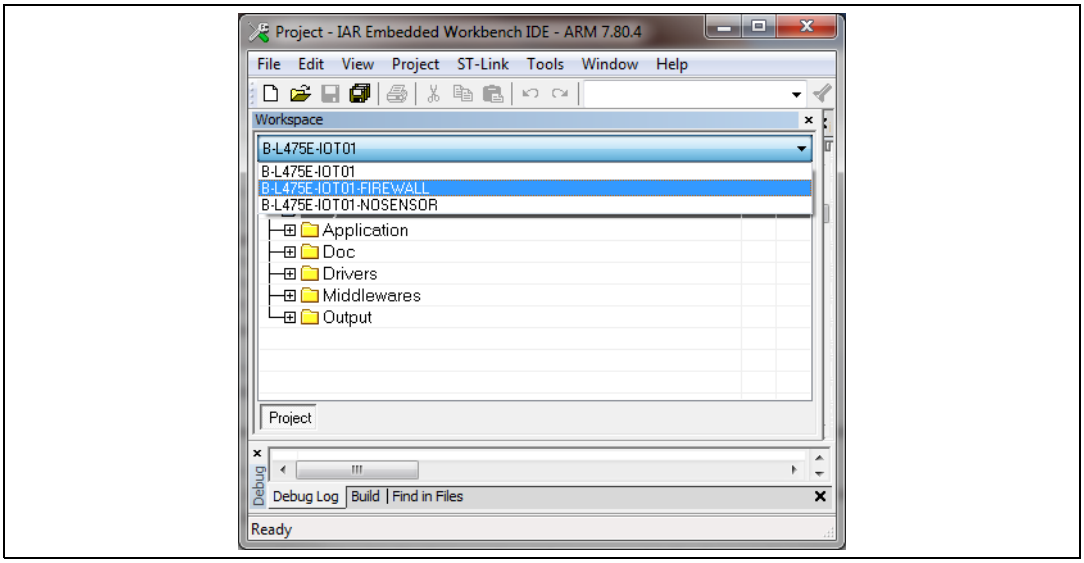
5.6 Running the project with the firewall activated

A configuration is provided in the IAR project for running the project with the firewall activated.

First, the project FireWallLib must be built with the help of the IAR project under the directory Projects\B-L475E-IOT01\Applications\Cloud\LibFireWall\EWARM.

Then, the main project must be built using the configuration named “Project - B-L475-IOT01-FIREWALL”. *Figure 11* shows how to select the right configuration with the IAR IDE.

Figure 11. Selection of the configuration named “B-L475-IOT01-FIREWALL” with IAR



This configuration sets the FIREWALL_MBEDLIB preprocessor define and the firewall binary is added to the main project (as highlighted by *Figure 10*). Using this configuration, the user does not need to modify the code.

5.7 Remote firmware update (RFU)

The network connectivity, for instance to an HTTP server, allows the board to download and run a new version of the firmware without connection to any development tool.

This section describes how the RFU (remote firmware update) feature has been implemented.

5.7.1 RFU overview

The provided example is implemented for the STM32L4 Series, and relies on specific capabilities of this core:

- Dual-boot from the dual-bank internal Flash memory, as detailed in the application note *“STM32 microcontroller system memory boot mode”* (AN2606) available at

www.st.com and illustrated in this STM32Cube example: STM32Cube_FW_L4_V1.6.0\Projects\STM32L476G-Discovery\Examples\FLASH\FLASH_DualBoot

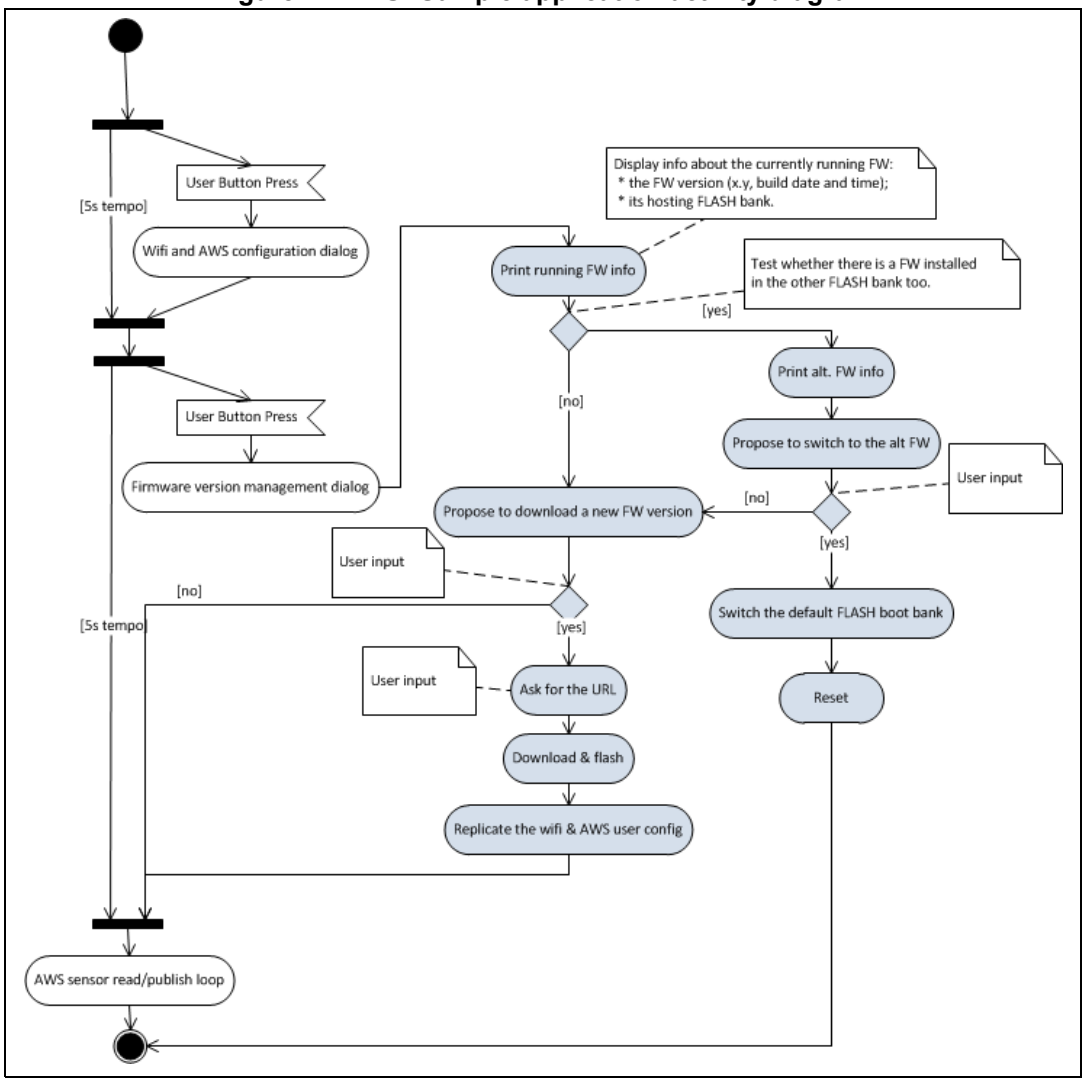
- 2k-paged internal Flash memory

The example consists in:

- Downloading the new firmware binary through HTTP
- Programming on-the-fly the downloaded payload to the alternate flash bank
- Let the application or the user decide to switch to the new firmware version by booting from the alternate flash bank

The sample application activity is depicted *Figure 12*.

Figure 12. RFU: Sample application activity diagram



1. The blue boxes represent the action states which are part of the firmware version management dialog.

5.7.2 Design

This sections describes the code that has been developed for the RFU feature.

Note: The IAR implementation is explained in this section as reference. The firmware information and user settings sections are implemented for Keil and System Workbench as well.

However those 2 latter tools cannot export the IAR simple code file format that is supported for the RFU feature.

For Keil and System Workbench, an additional PC utility development is needed to produce the IAR simple-code file format that is used.

Firmware version and build information

A firmware is identified by the following versioning structure, which is set at build time in *rfu.c* and placed at a fixed ROM position.

```
typedef struct {
  uint8_t major;
  uint8_t minor;
  uint8_t patch;
  char build_date[FWVERSION_DATE_SIZE];    /**< __DATE__ */
  char build_time[FWVERSION_TIME_SIZE];    /**< __TIME__ */
} firmware_version_t;


/** Important: lFwVersion must be stored at an address which persists
through the FW builds. */
const firmware_version_t lFwVersion @ "INITED_FIXED_LOC" = { <x>, <y>,
__DATE__, __TIME__ };
```

The location is defined in the project.*icf* file.

```
define symbol __ICFEDIT_region_FIXED_LOC_start__ = 0x08064000;
define region inited_fixed_loc = mem:[from
__ICFEDIT_region_FIXED_LOC_start__ + 8K size 1K];
(…)
place in inited_fixed_loc  { readonly section INITED_FIXED_LOC };
```

This version's information is displayed by the application to the user in the firmware management dialog. It allows the user to verify which version is running, and which version is available in the alternate Flash bank, if any.

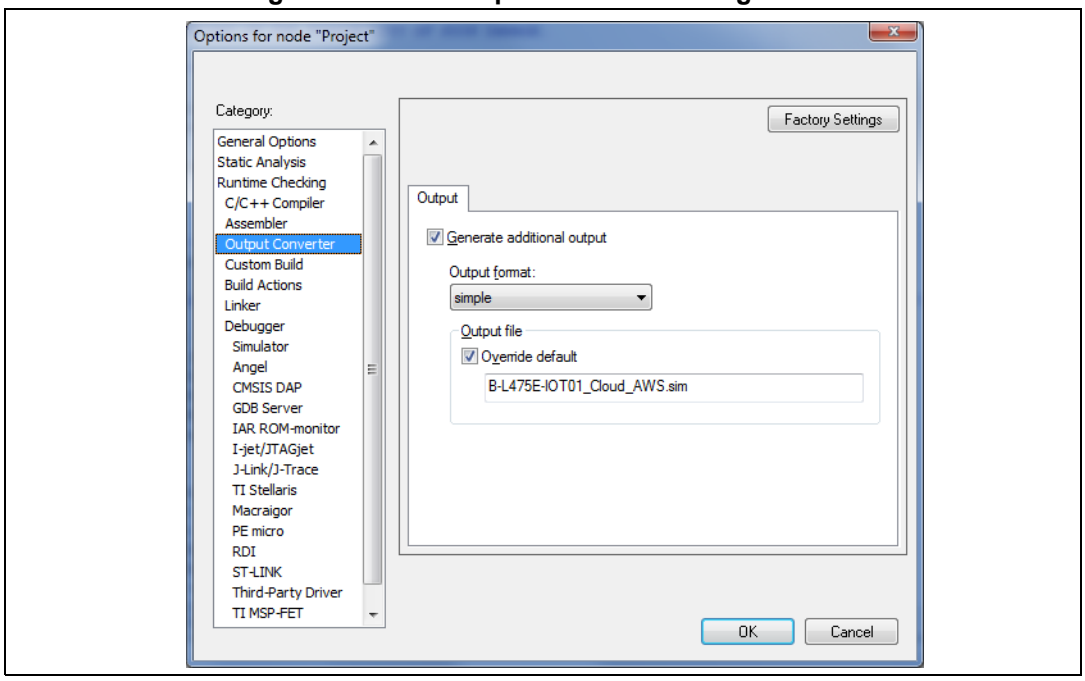
Firmware binary file format

The sample application decodes the IAR simple-code format in order to place the ROM segments at the right address.

The file can be formatted by the IAR output converter as depicted *Figure 13*. Note that the format is simple enough to allow the user to add sections at their convenience. See *simpleformat.h*.

Note: The full executable and user data should preferably fit in a single Flash bank. If data are spread across both banks, the user must take a special care to the ROM layout to prevent any collision when the CPU boots to the alternate bank.

Figure 13. RFU: Output converter configuration



Progressive HTTP download

In order to minimize the RAM footprint without relying on a local storage, the firmware file is downloaded through ranged HTTP requests. Record after record, the program data are retrieved by chunks of 2 kilobytes, and immediately written to their destination Flash page.

A checksum of all the received data is kept updated during this process.

Once the whole firmware has been received and programmed (that is when the end record is received), the application compares the computed checksum with the checksum read from the end record of the file.

Note: Whenever the download session is interrupted, the application could resume it without having to download again the beginning of the file.

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

The software interface to the download session is similar to a web socket, which simplifies the file-format decoding.

See *http_sock_open(), http_sock_close() and http_sock_rcv().*

Limitations:

- The sample application is not a compliant HTTP client.
  It can only issue bare GET requests, and does not support more advanced protocol features such as URL redirections or proxy usage.

Flash update

Each data segment decoded from the firmware file is translated to the alternate Flash bank. For instance, the program entry that should normally be programmed at 0x08000000 is

actually written at 0x08080000. After the bank switch and CPU reset, it is seen at 0x08000000 by the CPU.

FLASH_update() programs arbitrarily-aligned buffers into the Flash memory: the impacted pages are copied to the RAM before the page is erased, and rewritten.

When the source buffer is already page-aligned and fully overwrites the target pages, *FLASH_unlock_erase*() and *FLASH_write_at*() are rather directly used in order to prevent two useless memory copies.

Note: The written data are compared to the source buffer to detect any defective Flash cell.

Replication of the user configuration

Once the new firmware is programmed, the user configuration is replicated from the active Flash bank to the alternate Flash bank.

Similarly to the firmware version information, the user configuration structure is placed at a fixed location. The difference is that it is initialized from the console or by replication and not at build time.

```
/** Static user configuration data which must survive reboot and firmware
update. */
typedef struct {
  char tls_root_ca_cert[USER_CONF_TLS_OBJECT_MAX_SIZE];
  char tls_device_cert[USER_CONF_TLS_OBJECT_MAX_SIZE];
  char tls_device_key[USER_CONF_TLS_OBJECT_MAX_SIZE];
  wifi_config_t wifi_config;
  iot_config_t iot_config;
  uint64_t tls_magic;                        /**< The USER_CONF_MAGIC magic
word signals that the TLS strings above was once written to FLASH. */
} user_config_t;


/** Do not zero-initialize the static user configuration.
 *  Otherwise, it must be entered manually each time the device FW is
updated by STLink.
 */
__no_init const user_config_t lUserConfig @ "UNINIT_FIXED_LOC";
```

The location is defined in the *project.icf* file.

```
define symbol __ICFEDIT_region_FIXED_LOC_start__ = 0x08064000;
define region uninit_fixed_loc = mem:[from
__ICFEDIT_region_FIXED_LOC_start__ size 8K];
(…)
place in uninit_fixed_loc  { readonly section UNINIT_FIXED_LOC };
```

Boot bank switch

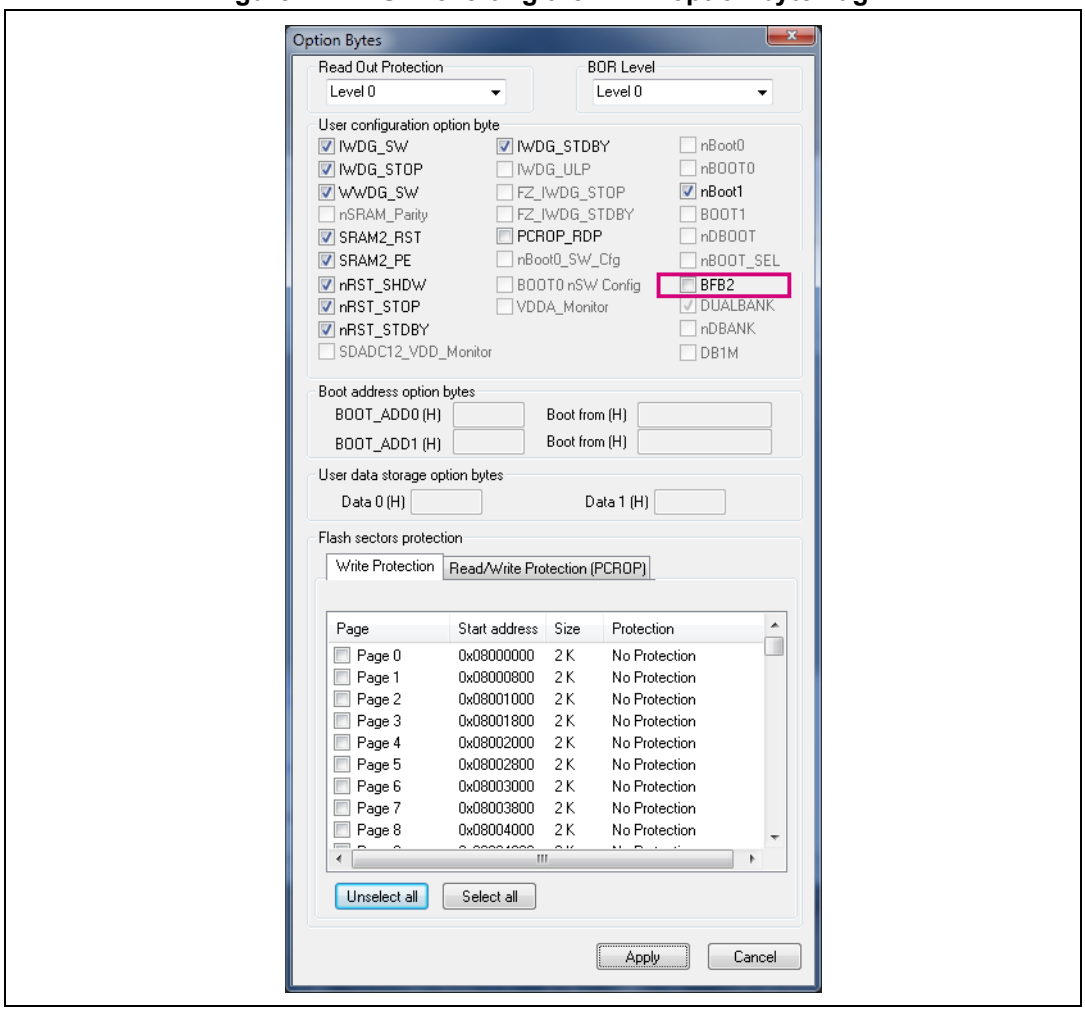
Finally, once the new firmware and the user configuration structure are in place in the alternate Flash bank, the user is allowed to switch the bootloader to the alternate bank thanks to *FLASH_set_boot_bank*().

This sets or resets the BFB2 option byte bit which allows booting from the second Flash bank.

Note: While programming through ST-LINK or the IDE, please remind that the first bank is being flashed. If another firmware is already programmed in the second bank, the bootloader boots there as soon as the BFB2 option byte flag is set and a plausible stack pointer initializer is found at 0x08080000.
See application note "STM32 microcontroller system memory boot mode" (AN2606) available at www.st.com for more details.
Booting to the first bank can be obtained by resetting the BFB2 option byte flag, or by erasing the 0x08080000 page in the STM32 ST-LINK utility.

Figure 14 highlights where to revert the BFB2 option flag.

Figure 14. RFU: Reverting the BFB2 option byte flag.



6 Running the AWS cloud demo with the 32F413HDISCOVERY board

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

6.1 Application behavior

The LED and user button work as stated in *Section 4.10.2 on page 17*.

6.2 Running the application

The steps to run the application are listed below:

1. Get the AWS IoT security credentials on the AWS web site
2. Connect a Type A to Micro-B USB cable from the connector USB ST_LINK to a PC
3. Under the directory Projects\ STM32F413H-Discovery\Applications\Cloud\AWS\<tool chain>, select the project, build and flash 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).

7 Running the AWS cloud demo with the 32F769IDISCOVERY board

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

7.1 Application behavior

The LED and user button work as stated in *Section 4.10.2 on page 17*.

7.2 Running the application

The steps to make run the application are listed below:

1. Get the AWS IoT security credentials on the AWS web site
2. Ensure that stlk is connected via a jumper
3. Connect a Type A to Micro-B USB cable from the (connector USB ST-LINK) to a PC
4. Connect an Ethernet cable to a server
5. Under the directory Projects\STM32F769I-Discovery\Applications\Cloud\AWS\<tool chain>, select the project, build and flash 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).

8 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.

9 Memory footprint

The values in *Figure 5* have been measured for the following configuration with the EWARM IDE 7.80.4:

- Optimization: optimized for size level 3
- Debug option: on
- Board: B-L475E-IOT01
- Firewall not used

Table 5. Memory footprint values

AWS memory status	Available (Kbyte)	Used (Kbyte)	Remains (Kbyte)
Flash	1024	200	824
SRAM(1)	96	41	55
SRAM2	32	0	32

1. Heap size = 28 Kbyte and stack size = 8 Kbyte.

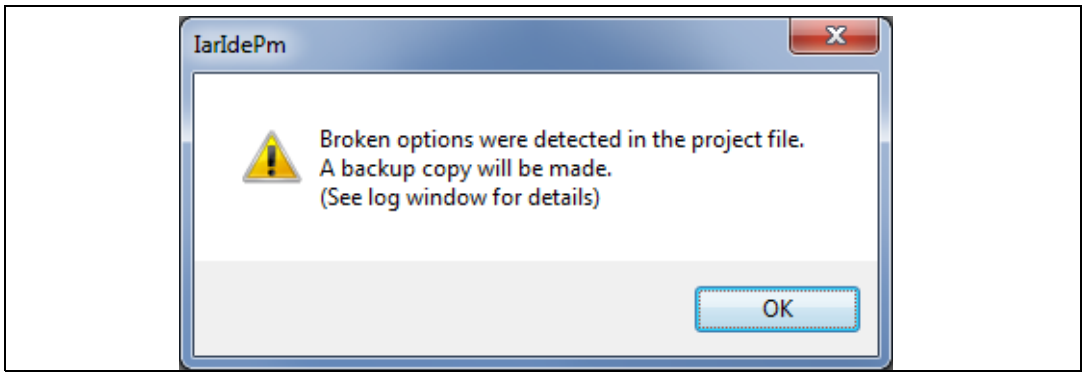
10 Frequently asked questions

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



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 WiFi access point. How shall I proceed?

A: Make sure that another device can connect to the WiFi access point. If it can, enter the WiFi 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.

11 Revision history

Table 6. Document revision history

Date	Revision	Changes
29-Mar-2017	1	Initial release.