life.augmented

UM2242
User manual

Getting started with Microsoft® Azure IoT cloud software expansion for STM32Cube

Introduction

This user manual describes the content of the STM32 Microsoft® Azure IoT (Internet of Things) cloud software expansion package for STM32Cube.

Microsoft® Azure is a cloud computing service created by Microsoft® for building, testing, deploying, and managing applications and services through a global network of Microsoft®-managed data centers. It provides software as a service (SaaS), platform as a service (PaaS), and infrastructure as a service (IaaS) and supports many different programming languages, tools, and frameworks, including both Microsoft®-specific and third-party software and systems.

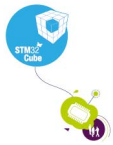
The Microsoft® Azure IoT software expansion package for STM32Cube (X-CUBE-AZURE) provides application examples that connect STMicroelectronics boards to the Azure IoT Hub.

X-CUBE-AZURE runs on the B-L475E-IOT01, 32F413HDISCOVERY and 32F769IDISCOVERY boards.

Implementation examples are included for device-to-cloud telemetry reporting and cloud-to-device messages for sending commands and notifications to the connected devices. The tracking of message deliveries with acknowledgment receipts is also implemented.

The X-CUBE-AZURE features are as follows:

- Ready to run firmware example using WiFi® and Ethernet connectivity to support quick evaluation and development of Azure device applications
- Interface to configure the board for connection to the Azure IoT Hub
- Connection to the Azure IoT Hub and various call-back registrations
- Azure IoT Hub, and bidirectional communication examples implemented
- The B-L475E-IOT01 board measures and reports the following values:
  - Humidity
  - Temperature
  - 3D magnetic data
  - 3D acceleration
  - 3D gyroscope data
  - Atmospheric pressure
  - Proximity



Contents

1 **Acronyms** 5

2 **Azure IoT Hub** 6

3 **Package description** 8

3.1 General description 8

3.2 Architecture 9

3.3 Folder structure 11

3.4 B-L475E-IOT01 board sensors 12

3.5 WiFi® components 12

3.6 Reset push-button 13

3.7 User push-button 13

3.8 User LED 13

3.9 Real-time clock 13

3.10 mbedTLS configuration 14

4 **Hardware and software environment setup** 15

5 **Interacting with the boards** 17

6 **Application examples** 19

6.1 Application description 19

6.2 Application setup 19

6.2.1 Azure device creation 19

6.2.2 Application build and flash 19

6.2.3 Firmware programming on the STM32 board 19

6.2.4 Application first launch 20

6.3 Application runtime 20

7 **Frequently asked questions** 26

8 **Revision history** 27

List of tables

Table 1. List of acronyms 5
Table 2. Units for the values reported by the sensors of the B-L475E-IOT01 board 12
Table 3. iothub-explorer command lines. 23
Table 4. Document revision history 27

List of figures

Figure 1. Azure IoT ecosystem 6
Figure 2. X-CUBE-AZURE software architecture. 10
Figure 3. Project file structure 11
Figure 4. Hardware and software setup environment 15
Figure 5. Terminal setup 17
Figure 6. Serial port setup 18
Figure 7. Runtime state flow 22
Figure 8. Pop-up when the IAR™ IDE version is not compatible with the one used for X-CUBE-AZURE 26

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
BSP	Board support package
C2D	Cloud to device
CA	Certification authority
D2C	Device to cloud
DHCP	Dynamic host configuration protocol
DNS	Domain name server
HAL	Hardware abstraction layer
IDE	Integrated development environment
IoT	Internet of things
IP	Internet protocol
JSON	JavaScript object notation
LED	Light-emitting diode
RTC	Real-time clock
UART	Universal asynchronous receiver/transmitter

2 Azure IoT Hub

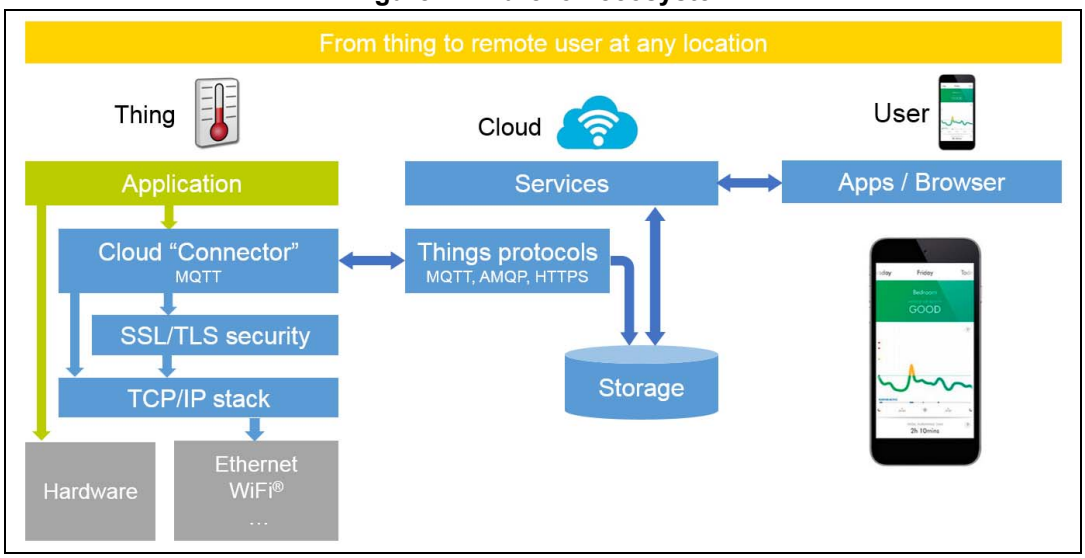
This section introduces the Azure IoT Hub service.

The X-CUBE-AZURE package implements Azure IoT device SDKs in C language which allows the board to securely connect to the Azure IoT Hub service.

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

Figure 1 presents the Azure IoT ecosystem targeted by the X-CUBE-AZURE package. The X-CUBE-AZURE package implements the connection of the thing to the Azure IoT Hub service with the MQTT protocol. The Apps / Browser and other things protocols are only shown in *Figure 1* for information as existing Azure features.

Figure 1. Azure IoT ecosystem



In addition to a rich set of device-to-cloud (D2C) and cloud-to-device (C2D) communication options, including messaging, file transfers, and request-reply methods, Azure IoT Hub addresses device-connectivity in the following ways:

- Device twins. Using device twins, users can store, synchronize, and query device meta-data and state information. Device twins are JSON documents that store device state information (meta-data, configurations, and conditions). IoT Hub persists a device twin for each device connected to IoT Hub.
- Per-device authentication and secure connectivity. Users can provision each device with its own security key to enable it to connect to IoT Hub. The IoT Hub identity registry stores device identities and keys in a solution. A solution back end can add individual devices to allow or deny lists to enable complete control over device access.
- Route device-to-cloud messages to Azure services based on declarative rules. IoT Hub enables users to define message routes based on routing rules to control where the hub sends device-to-cloud messages. Routing rules do not require users to write any code, and can take the place of custom post-ingestion message dispatchers.
- Monitoring of device connectivity operations. Users can receive detailed operation logs about device identity management operations and device connectivity events. This monitoring capability enables their IoT solution to identify connectivity issues, such as

devices that try to connect with wrong credentials, send messages too frequently, or reject all cloud-to-device messages.

- Extensive set of device libraries. Azure IoT deviceSDKs are available and supported for various languages and platforms: C for many Linux® distributions, Windows®, and real-time operating systems. Azure IoT device SDKs also support managed languages, such as C#, Java, and JavaScript.
- IoT protocols and extensibility. If the solution cannot use the device libraries, IoT Hub exposes a public protocol that enables devices to natively use the MQTTv3.1.1, HTTP 1.1, or AMQP 1.0 protocols. The user can also extend IoT Hub to support for custom protocols by:
  - Creating a field gateway with Azure IoT Edge that converts a custom protocol to one of the three protocols understood by IoT Hub.
  - Customizing the Azure IoT protocol gateway, an open source component that runs in the cloud.
- Scale. Azure IoT Hub scales to millions of simultaneously connected devices and millions of events per second.

For a complete description of Microsoft® Azure and Azure IoT Hub, refer to the information available at the overview of the Azure IoT Hub service webpage.

3 Package description

This section details the X-CUBE-AZURE package content and the way to use it.

3.1 General description

The X-CUBE-AZURE package provides an Azure stack middleware for STM32 microcontrollers.

It is ported to the B-L475E-IOT01, 32F413HDISCOVERY and 32F769IDISCOVERY boards and connects to the Internet through the on-board network interface:

- 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, 3D-axis magnetic data, 3D accelerations, 3D gyroscope data, atmospheric pressure, proximity and gesture detection (X-CUBE-AZURE does not use the gesture detection capability).
- 32F413HDISCOVERY supports WiFi® connectivity with an on board Inventek module.
- 32F769IDISCOVERY natively provides an Ethernet interface.

The package is split into the following components:

- C99 SDK for connecting devices to Microsoft® Azure IoT services
- 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
- Azure 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 starting from 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

3.2 Architecture

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

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

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

The software components used by the application software to access and use the Azure IoT Hub are the following:

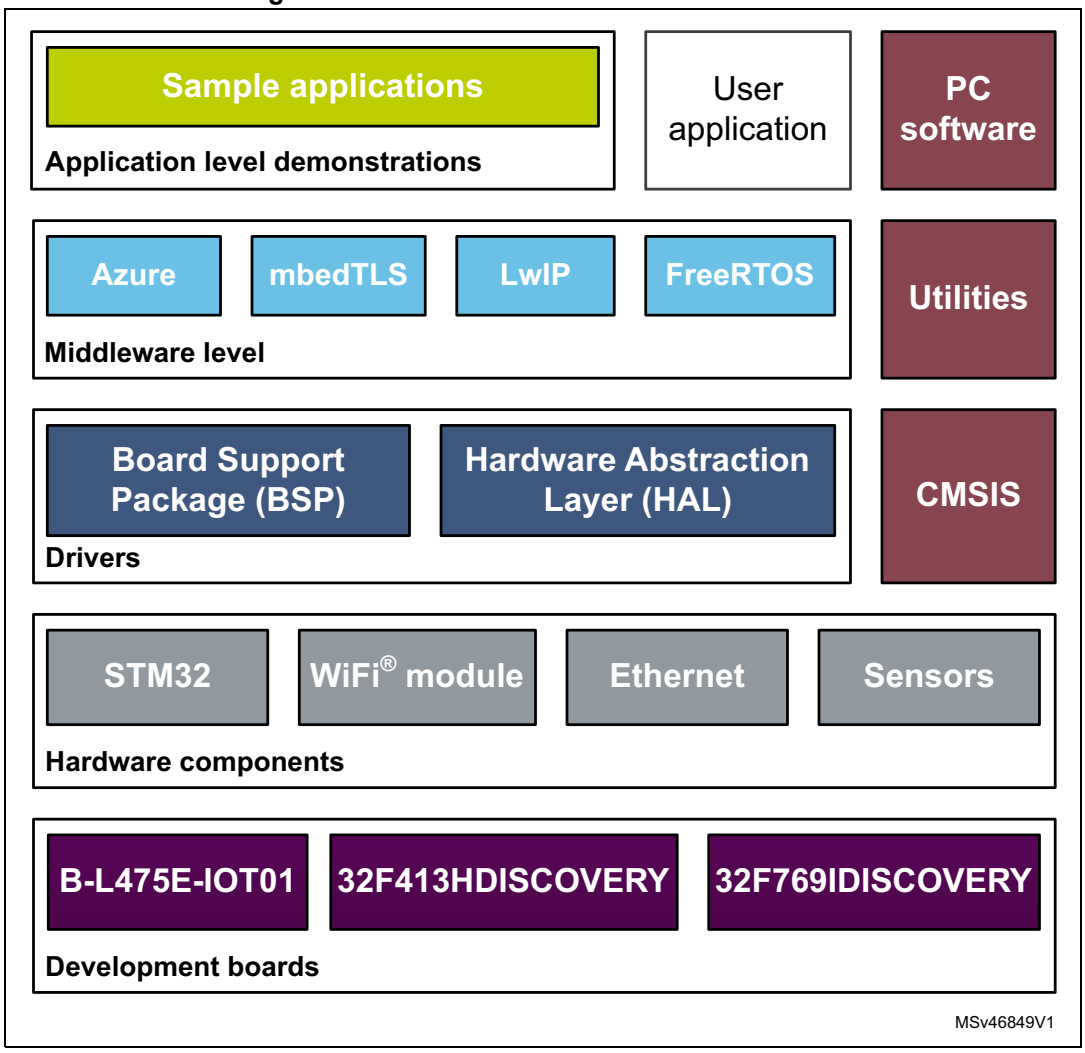
- STM32Cube HAL
  The HAL driver layer provides a generic multi instance simple set of APIs (application programming interfaces) to interact with the upper layers (application, libraries and stacks).

  It is composed of generic and extension APIs. It is directly built around a generic architecture and allows the layers that are built upon, such as the middleware layer, to implement their functionalities without dependencies on the specific hardware configuration for a given microcontroller unit (MCU).

  This structure improves the library code reusability and guarantees an easy portability onto other devices.
- Board support package (BSP)
  The software package needs to support the peripherals on the STM32 boards apart from the MCU. This software is included in the board support package (BSP). This is a limited set of APIs which provides a programming interface for certain board specific peripherals such as the LED and the user button.
- Azure middleware
  It is composed of the Azure IoT Hub client library, a JSON parser, a JSON serializer, an MQTT client (used as a transport layer by the IoT Hub client library), and various C utilities used by the client library.
- mbedTLS
  The Azure middleware uses a TLS connection which is managed by the mbedTLS library.
- TCP/IP
  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-AZURE package, only the 32F769IDISCOVERY board can connect via Ethernet.
- FreeRTOS™
  It is a real-time operating system required by LwIP for providing a socket-based interface to the user.

Figure 2 outlines X-CUBE-AZURE software architecture.

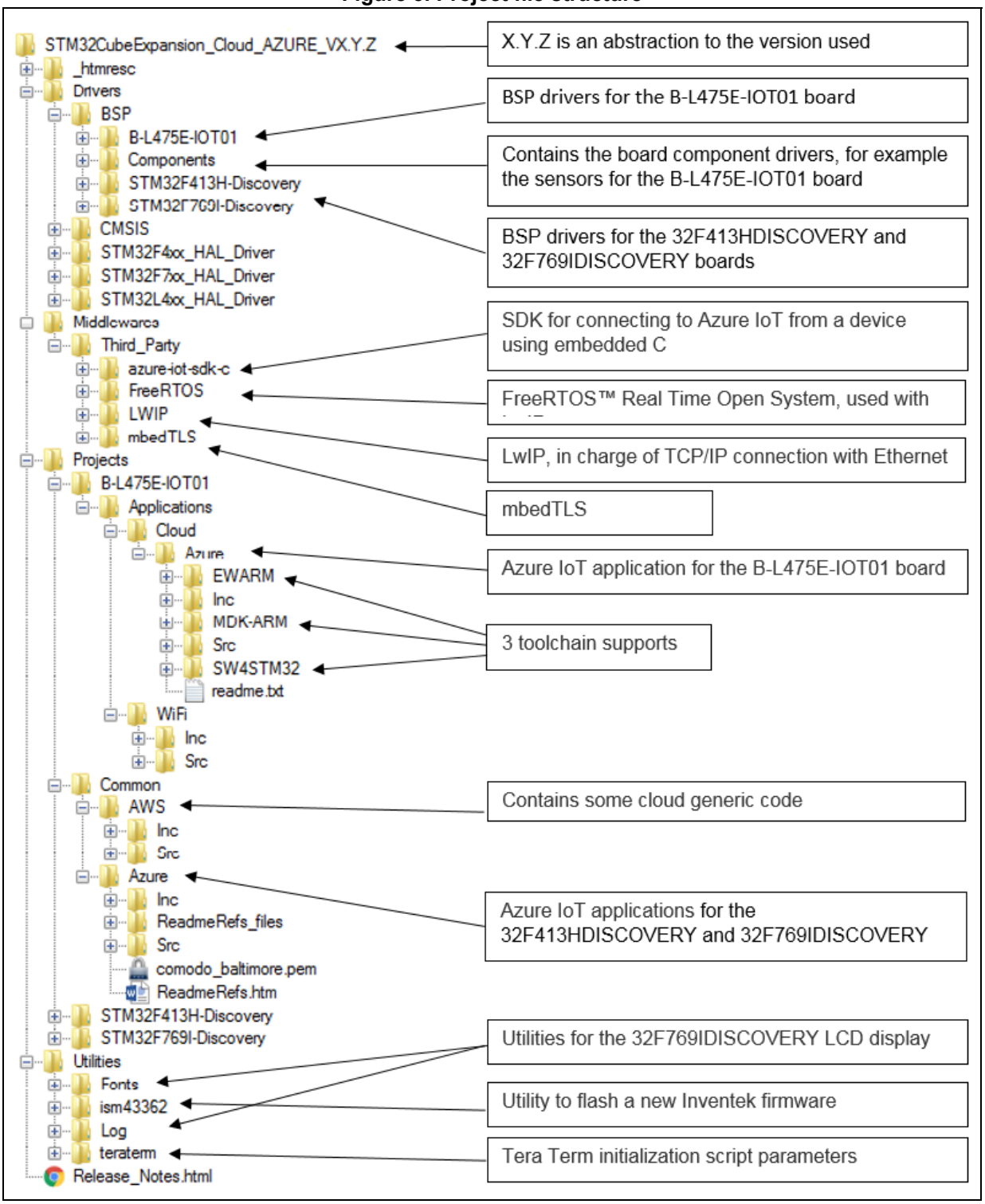
Figure 2. X-CUBE-AZURE software architecture



3.3 Folder structure

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

Figure 3. Project file structure



3.4 B-L475E-IOT01 board sensors

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

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

Example of a published sensor message:

```
{
  "mac": "<mac address of the device>",
  "temperature": 31.39856,
  "humidity": 29.069721,
  "pressure": 997.830017,
  "proximity": 8190,
  "accX": -13,
  "accY": -14,
  "accZ": 1024,
  "gyrX": 1750,
  "gyrY": -4970,
  "gyrZ": 1470,
  "magX": 170,
  "magY": -180,
  "magZ": 605,
  "ts": "2017-06-07T15:14:22Z"
}
```

Table 2 presents the units for the values reported by the sensors of the B-L475E-IOT01 board.

Table 2. Units for the values reported by the sensors of the B-L475E-IOT01 board

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

3.5 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.

3.6 Reset push-button

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

3.7 User push-button

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

- To configure the WiFi® and Azure security credentials. This can be done from the time that the board starts up and up to five seconds after that.
- When the board has been initialized to control the way data are published to the Azure IoT Hub refer to *Figure 7*

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

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

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

3.8 User LED

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

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

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

3.9 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()` function to get the time value.

This function can for instance be used to time stamp messages.

3.10 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 mans of the MBEDTLS_CONFIG_FILE #define.

The X-CUBE-AZURE package uses file az_mbedtls_config.h for project configuration.

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

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

The configuration file specifies the ciphers to integrate.

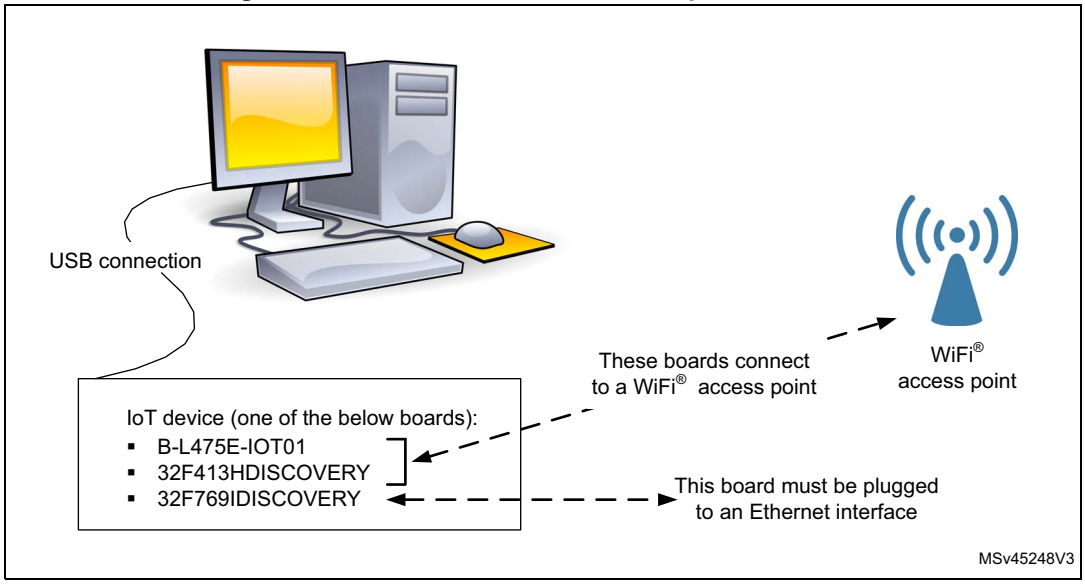
4 Hardware and software environment setup

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 Azure 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 while the 32F769IDISCOVERY board must be connected to an Ethernet interface as illustrated in *Figure 4*.

Figure 4. Hardware and software setup environment



The prerequisites for running the examples are:

- A WiFi® access point, with a transparent Internet connectivity meaning that neither a proxy, nor a firewall are blocking the outgoing traffic. It has to run a DHCP server delivering the IP and DNS configuration to the board.
- A computer for running a device management application, with a transparent Internet connectivity meaning that neither a proxy, nor a firewall are blocking the outgoing

traffic. This can for instance be the development PC, a virtual private server or a single-board computer. It can be connected to the same router as the MCU board.

- An Azure IoT account to create an IoT Hub. Refer to section *Before you run the samples* at iot-hub-device-sdk-c-intro - GitHub web page
- An Azure device management application which is required to communicate with the device.

  There are two options which can be downloaded from the above web page.

  In this document, we have selected the iothub-explorer tool as the reference for the present Cube expansion package. It runs on any operating system. It is a node.js application.

  Alternatively, Device Explorer which runs on Windows® (.NET) can be used
- A development PC for building the application, programming through ST-Link, and running the virtual console.

5 Interacting with the boards

A serial terminal is required to:

- Configure the board
- Display locally the received Azure IoT C2D 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 Azure IoT identification 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.

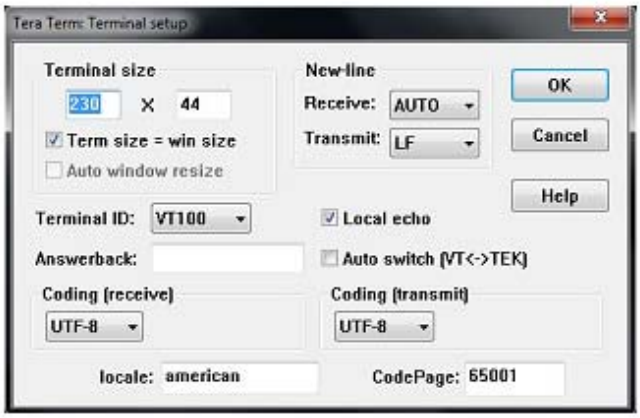
A Tera Term initialization script is provided in the package utility directory (refer to *Figure 3*); this script sets the correct parameters. To use it, open Tera Term, select Setup and then Restore setup.

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 copy-paste from UNIX type text files. The Local echo option makes copy-paste visible on the console.

Figure 5. Terminal setup

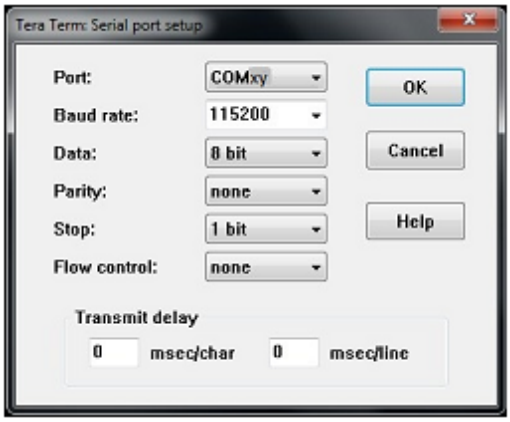


The serial port must be configured with:

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

Serial port setup is illustrated in *Figure 6*.

Figure 6. Serial port setup



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

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

6 Application examples

6.1 Application description

The AzureXcubeSample application illustrates the various ways for an Azure device to interact with an Azure IoT Hub.

The application connects to an Azure IoT Hub on basis of the credentials provided by the user on the console.

6.2 Application setup

The setup of the application requires that the steps described from *Section 6.2.1* to *Section 6.2.4* are executed in sequence.

6.2.1 Azure device creation

The two next commands create an Azure device and get its connection string:

- $ iothub-explorer login <your Azure IoT Hub Connection String>
- $ iothub-explorer create <devId> --connection-string

The board MAC address can for instance be used as a device ID. A smart nickname can also be chosen for convenience.

It is advised to keep a copy of the device connection string at hand since the AzureXcubeSample application requests it on the console when it is launched for the first time.

The next command allows to verify that the device twin status can be retrieved:

- $ iothub-explorer get-twin <devId>

6.2.2 Application build and flash

Open the selected toolchain at STM32CubeExpansion_Cloud_AZURE_Vx.y.z\Projects\<board name>\Applications\Cloud\Azure\<IDE> and build the project.

Refer to *Section 3.1: General description on page 8* for detailed information about the IDE version requirements.

6.2.3 Firmware programming on the STM32 board

The binary file generated in STM32CubeExpansion_Cloud_AZURE_Vx.y.z\Projects \<board name>\Applications\Cloud\Azure\<IDE>\Exe can be copied or dragged and dropped to the USB mass storage location created when the STM32 board is plugged to the PC.

If the host is a Linux® PC, the STM32 device can be found in the /media folder with name DIS_L4IOT. For example, if the created mass storage location is /media/DIS_L4IOT, then the command to program the board with a binary file named my_firmware.bin is simply:

cp my_firmware.bin /media/DIS_L4IOT.

Alternatively, the STM32 board can directly be programmed through one of the supported development toolchains.

6.2.4 Application first launch

The board must be connected to a PC through USB (ST-LINK USB port).

Open the console through a serial terminal emulator such as Tera Term (refer to *Section 3.2: Architecture on page 9*).

On the console:

- For WiFi®-enabled boards, enter the Wifi® SSID, encryption mode and password
- Set the device connection string (refer to *Section 6.2.1*), excluding enclosing quotes (")
- Set the TLS root CA certificates by copy-pasting the contents of STM32CubeExpansion_Cloud_AZURE_Vx.y.z\Projects\Common\Azure\comodo_baltimore.pem. The device uses them to authenticate the remote hosts through TLS.

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

1. For the HTTPS server which is used to retrieve the current time and date at boot time.(the Comodo certificate for the www.gandi.net server)

2. For the IoT Hub server (the Baltimore certificate)

The concatenated string must end with an empty line. This is comodo_baltimore.pem.

After the parameters are configured, it is possible to change them by restarting the board and pressing the user button (blue button) just after boot.

6.3 Application runtime

This section describes the life-cycle steps of the application that:

- Make a single HTTPS request to retrieve the current time and date, and configure the RTC
- Connect to the Azure IoT Hub
- Get the status of the device twin
- Update its local properties (DesiredTelemetryInterval) from the desired properties of the device twin
- Report the reported properties to the device twin (TelemetryInterval and LedStatus)

  Note: From this point, the user can get the twin status updates through the

  $ iothub-explorer get-twin <devId> command
- Stay idle, pending on local user, or hub-initiated events

From this point the possible local user actions are:

- Single push on the user button: this action triggers a message publication to the IoT Hub through a DeviceToCloud (D2C) message.
- Double push on the user button: this action starts or stops message publication loop. When the loop is running, the messages are published every TelemetryInterval seconds.

  Note: Each message publication is signaled by the user LED blinking quickly for half a second.

  The message contents depends on the board type used:

  – *B-L475E-IOT01 reports the sensor values and a timestamp*

  – *32F413HDISCOVERY and 32F769IDISCOVERY only report a timestamp*

The implemented hub-initiated events are:

- CloudToDevice (C2D) message
  The message is displayed on the board console
- C2D twin update
  Used to change the telemetry publication period (that is the DesiredTelemetryInterval parameter)
- C2D method
  Used to call one of the following device methods:
  – Reboot: reboots the board
  – Hello: displays the message passed as parameter on the board console
- C2D action call
  LedToggle can be called to make the user LED state toggle

Figure 7 presents the runtime state flow.

Figure 7. Runtime state flow

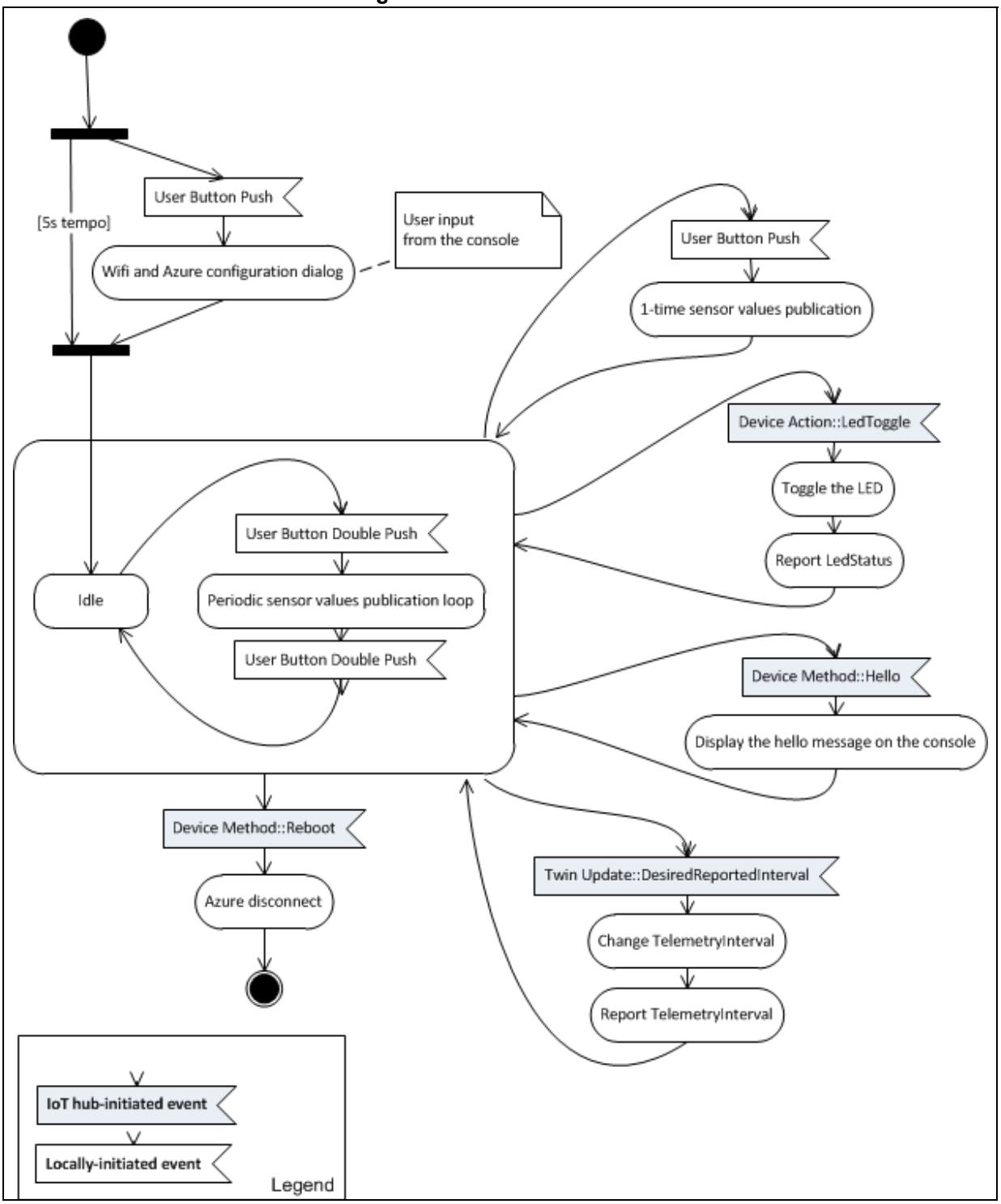


Table 3 lists the iothub-explorer command lines for the user to trig hub-initiated events, and see the results. The seven communication interfaces between the device and the cloud are listed. The commands in the way to call column uses the iothub-explorer node package.

Table 3. iothub-explorer command lines

Communication interface	Purpose in application	Way to call, or format	Comment

D2C message	Publishes telemetry data	monitor-events -l <IoTHubConnectionString> < devId>
		>
		{
		"mac": "<mac address of the device>",
		"temperature": 31.39856,
		"humidity": 29.069721,
		"pressure": 997.830017,
		"proximity": 8190,
		"accX": -13,
		"accY": -14,
		"accZ": 1024,
		"gyrX": 1750,
		"gyrY": -4970,
		"gyrZ": 1470,
		"magX": 170,
		"magY": -180,
		"magZ": 605,
		"ts": "2017-06-07T15:14:22Z"
		}
			The telemetry messages can be monitored if the user activates the publication by means of the user button on the board. Refer to *Figure 7*.

C2D message	Sends Hello world	send <devId> 'Hello world'	The Azure SDK prints an error log because a C2D JSON command syntax is expected by the message callback implementation on the device, while a simple text is sent.

Table 3. iothub-explorer command lines (continued)

Communication interface	Purpose in application	Way to call, or format	Comment
C2D twin update	Changes the telemetry publication period	update-twin <devId> '{ "properties": { "desired": { "DesiredTelemetryInterval": 6 } } }'	The client SDK, iothub-explorer and the hub assume different payload formats for the twin-update: at connection time, the full string is received from the hub: { "desired": { "DesiredTelemetryInterval": x } } at runtime, when using update-twin, it contains only { "DesiredTelemetryInterval": x } The callback implementation on the device is compatible with both formats. Still, it calls the JSON parser to identify the format. If the Azure SDK logtrace option was set, an error log gets printed when the desired key is not found. On twin update, the new TelemetryInterval is automatically reported through a D2C twin update.

D2C twin update	Reports the telemetry activation parameter, and the LED status	The twin-update is automatically performed by the application running on the device.
		The changes can be retrieved as follows:
		`get-twin [-r] <devId>`
		`>`
		`{`
		`"deviceId":"C47F510111A7",`
		`"properties":{`
		`"desired":{`
		`"DesiredTelemetryInterval":6,`
		`},`
		`"reported":{`
		`"TelemetryInterval":6,`
		`"LedStatusOn":false`
		`}`
		`}`
		`}`
			Updates happen at connection time, upon C2D twin update, and upon C2D LedToggle call.

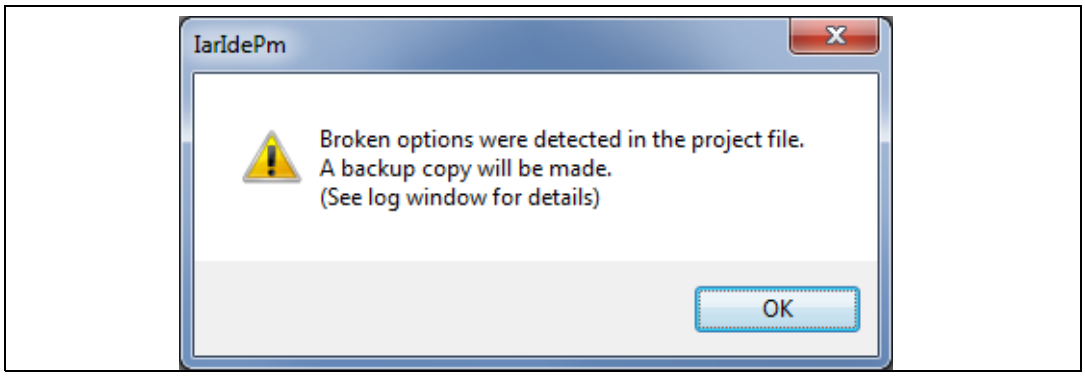
Table 3. iothub-explorer command lines (continued)

Communication interface	Purpose in application	Way to call, or format	Comment
C2D method	Reboots, sends a message	device-method <devId> Reboot '{ "when": "now"}' an device-method <devId> Hello '{ "msg": "World!" }'	For interoperability reasons, a valid JSON string must be passed as parameter to each device method call, even if the target function has no parameter. The Reboot implementation does not use any of the passed parameters.
C2D call	Sends a LedToggle message for interpretation by the device	send <devId> '{ "Name": "LedToggle", "Parameters": "" }'	For interoperability reasons, the Parameters key must be given a value (any value).
D2C upload to blob	Unused	-	Not implemented.

7 Frequently asked questions

Q: Why do I get this pop up (refer to *Figure 8*) when I open the project with IAR™?

Figure 8. Pop-up when the IAR™ IDE version is not compatible with the one used for X-CUBE-AZURE



A: It is very likely that the IAR™ IDE version is older than the one used to develop the package (refer to *Section 3.1: General description on page 8*), hence the compatibility is not ensured. In this case, the IAR™ IDE version needs to be updated.

Q: My device does not connect to the 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: 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.

8 Revision history

Table 4. Document revision history

Date	Revision	Changes
20-Jul-2017	1	Initial release.