AN12186

A71CH for secure connection to IBM Watson IoT

Rev. 1.0 — 26 September 2018 Application note
464110 COMPANY PUBLIC

Document information

Info	Content
Keywords	IoT, Security IC, A71CH, IBM Watson IoT, Trusted cloud connection
Abstract	This application note describes how to set up a trusted connection to IBM Watson IoT Platform using the A71CH Customer Programable type and the A71CH Provisioned and Programmable.

Revision history

Rev	Date	Description
1.0	20180926	First release

Contact information

For more information, please visit: http://www.nxp.com

© NXP Semiconductors N.V. 20184. All rights reserved.

1. Introduction

One critical component in IoT is device identity. The cloud platform needs to be able to verify the identity of the device to trust its data and grant access to the cloud platform. As such, the IoT device identity should be unique, verifiable, and trustworthy to establish a root of trust. The IoT device identity can be verified using public key cryptography. Public key cryptography algorithms are based on key pairs: one private key and one public key.

The private key must be protected for the entire lifetime of the product to prevent malicious attackers from being able to falsify the identity of your devices. The private key must be isolated from users, software, and flash memory of microcontrollers to achieve confidentiality.

The A71CH can prevent key leakage by providing a tamper-resistant platform, capable of securely storing and provisioning credentials, securely connecting IoT devices to cloud services and performing cryptographic node authentication. The A71CH solution offers an outstanding level of security measures, which protects it against physical and logical attacks. In addition, it can be used with various host platforms and host operating systems to secure a broad range of applications.

2. How to use this document

This application note considers two A71CH product variants:

- The ***A71CH Customer Programmable type:*** This product variant is delivered empty and without any credential provisioned. This type is intended for use during the evaluation, testing and prototyping design phases. It can also be used in case your organization owns PKI infrastructure or subcontracts a third-party PKI infrastructure to provision the A71CH.
- The ***A71CH Provisioned & Programmable type; Ready for IBM Watson IoT***: This product variant is delivered already provisioned by NXP. It includes one ECC key pair, one X.509 certificate for use in an IoT device and one X.509 certificate for use in an IoT gateway. These credentials injected during the NXP Trust Provisioning support the establishment of a trusted TLS connection to IBM Watson IoT Platform. This type is suggested for field deployment and operational stages.

For each step in the implementation process different chapters of this application note are relevant. This application note is organized in the following chapters:

- Section 3 describes how to set up your IBM Watson IoT Platform account, as well as how to register IoT devices and certificates in the IBM Platform. Reading this chapter is suggested if you need to prepare your IBM Watson IoT Platform environment.
- Section 4 details how to use an A71CH Arduino compatible development kit, an i.MX6UltraLite board and a demo application to illustrate how to prepare an ***A71CH Customer Programmable type*** for connection with IBM Watson IoT Platform. Reading this chapter is suggested if you want to understand the credentials involved, the IoT device authentication process to IBM Watson IoT Platform as well as to evaluate the A71CH integration with the i.MX6UltraLite Host MCU.
- Section 5 details how to use an A71CH Arduino compatible development kit, a FRDM-K64F board and a demo application to illustrate how to prepare an ***A71CH***

Customer Programmable type for connection with IBM Watson IoT Platform. Reading this chapter is suggested if you want to understand the credentials involved, the IoT device authentication process to IBM Watson IoT Platform as well as to evaluate the A71CH integration with the K64F Host MCU.

- Section 6 explains how to integrate and prepare an ***A71CH Provisioned & Programmable type; Ready for IBM Watson IoT*** for field deployment or operational use with your IBM Watson IoT Platform. Reading this chapter is suggested to understand how to integrate and onboard the A71CH Provisioned & Programmable into your mass production environment or manufacturing flow of your IoT application.

As a result, the best way to use this document is as a reference guide. We recommend reading the chapters that are relevant to you based on the implementation stage and design requirements.

3. IBM Watson IoT platform setup

The IBM Watson IoT Platform is a fully managed, cloud-hosted service with capabilities for device registration, connectivity, control, rapid visualization and data storage. The IBM Watson IoT Platform allows us to collect and secure data and run analytics of our IoT device. To get up and running with the IBM Watson IoT Platform, you must connect your IoT devices to the platform. IoT devices are connected by completing the following steps:

0. Prerequisite: An A71CH that has been provisioned with the required credentials.
1. Set up the Watson IoT Platform environment.
2. Register the CA certificates in Watson IoT Platform.
3. Configure a Connection Security Policy in Watson IoT Platform.
4. Define a Device Type and register the IoT device in Watson IoT Platform.
5. Connect the IoT device to Watson IoT Platform.

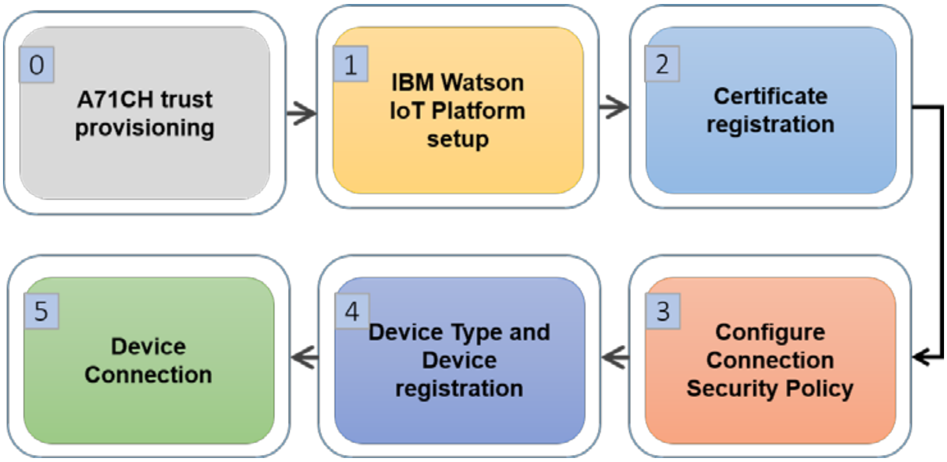

0
A71CH trust provisioning
1
IBM Watson IoT Platform setup
2
Certificate registration
3
Configure Connection Security Policy
4
Device Type and Device registration
5
Device Connection

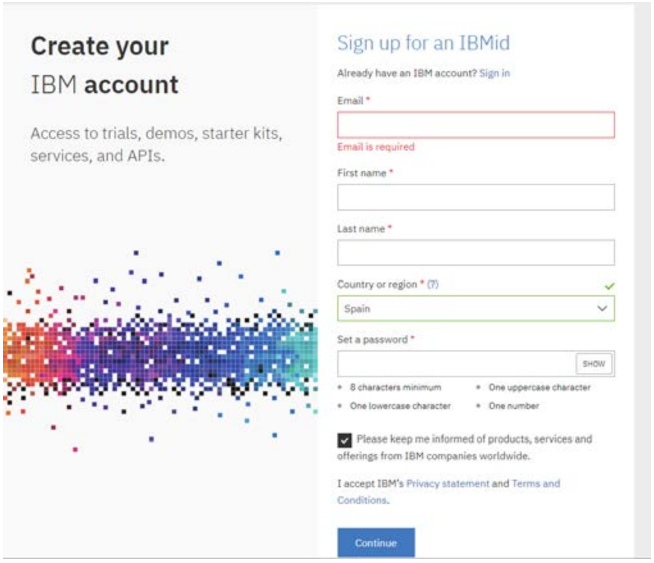
Fig 1. Flow diagram of the connection of the A71CH Security IC to IBM Watson.

3.1 Watson IoT Platform setup

Setting up the Watson IoT Platform environment is a two-step process. First, it is necessary to create an IBM account, known as an IBMid, and, second, a personal space must inside Watson IoT Platform must be created.

3.1.1 IBM Cloud account creation

An IBM Cloud account is needed to configure and prepare the Watson IoT Platform. Enter the IBM website [IBM_CLOUD] and click on ‘*Create an IBMid’*. Fig 2 illustrates the fields that need to be filled in for registering a new account. When all the fields are filled in and the terms accepted, click ‘*Continue’*.



Create your IBM account

Access to trials, demos, starter kits, services, and APIs.

Sign up for an IBMid

Already have an IBM account? Sign in

Email *

Email is required

First name *

Last name *

Country or region * (?) ✓

Spain

Set a password * SHOW

- 8 characters minimum
- One uppercase character
- One lowercase character
- One number

☑ Please keep me informed of products, services and offerings from IBM companies worldwide.

I accept IBM's Privacy statement and Terms and Conditions.

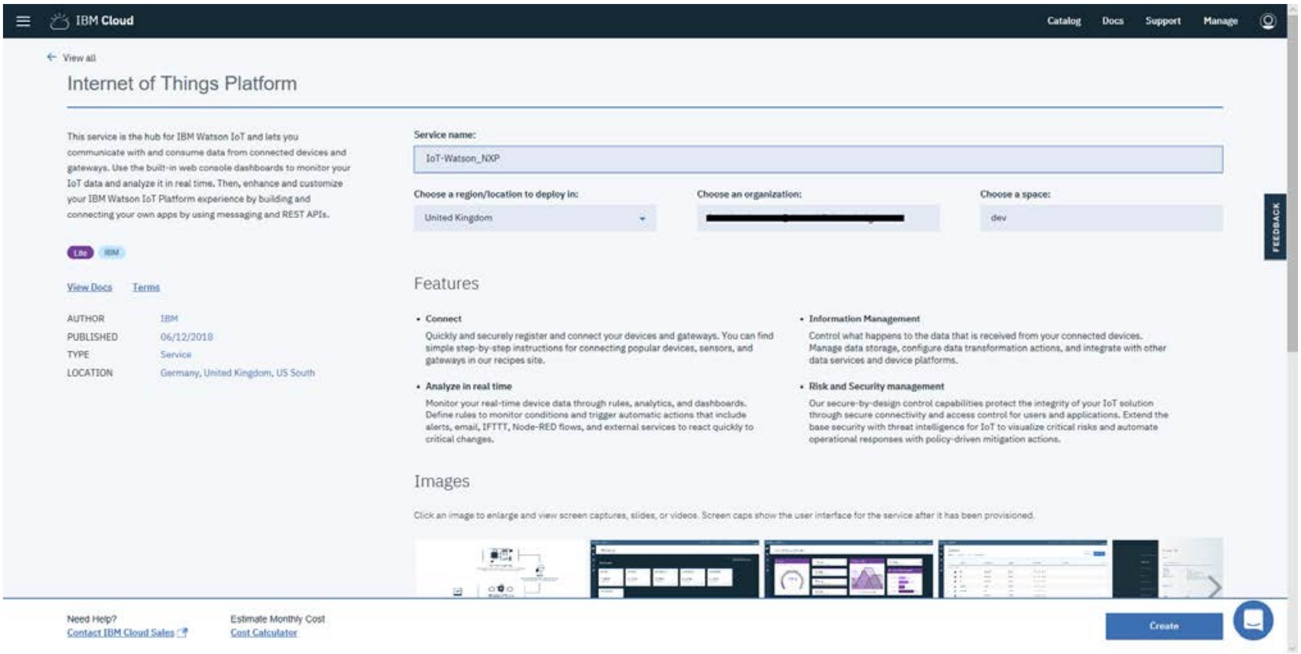
Continue

Fig 2. IBM Cloud account creation

3.1.2 Watson IoT Platform instance creation

The next step is to create an instance of IBM Watson IoT Platform. An instance is a ‘*personal space’* in the IBM Watson IoT Platform.

Enter the IBM website and click *‘Log in’* to enter with the account created in Section 3.1. Fig 3 illustrates the screen that appears after logging in. Select a name for the instance, choose a region and select the Lite pricing plan. The region indicates where the IBM Watson IoT Platform server is hosted, while the Lite pricing plan can be used for development and evaluation purposes. Finally, click on ‘*Create’* to finish the process and create the instance of IBM Watson IoT Platform.



← View all

Internet of Things Platform

This service is the hub for IBM Watson IoT and lets you communicate with and consume data from connected devices and gateways. Use the built-in web console dashboards to monitor your IoT data and analyze it in real time. Then, enhance and customize your IBM Watson IoT Platform experience by building and connecting your own apps by using messaging and REST APIs.

AUTHOR	IBM
PUBLISHED	06/12/2018
TYPE	Service
LOCATION	Germany, United Kingdom, US South

Service name: IoT-Watson_NXP

Choose a region/location to deploy in: United Kingdom

Choose an organization: [redacted]

Choose a space: dev

Features

- **Connect**
  Quickly and securely register and connect your devices and gateways. You can find simple step-by-step instructions for connecting popular devices, sensors, and gateways in our recipes site.
- **Analyze in real time**
  Monitor your real-time device data through rules, analytics, and dashboards. Define rules to monitor conditions and trigger automatic actions that include alerts, email, IFTTT, Node-RED flows, and external services to react quickly to critical changes.
- **Information Management**
  Control what happens to the data that is received from your connected devices. Manage data storage, configure data transformation actions, and integrate with other data services and device platforms.
- **Risk and Security management**
  Our secure-by-design control capabilities protect the integrity of your IoT solution through secure connectivity and access control for users and applications. Extend the base security with threat intelligence for IoT to visualize critical risks and automate operational responses with policy-driven mitigation actions.

Images

Click an image to enlarge and view screen captures, slides, or videos. Screen caps show the user interface for the service after it has been provisioned.

Need Help?
Contact IBM Cloud Sales

Estimate Monthly Cost
Cost Calculator

Fig 3. Watson IoT Platform instance creation

3.2 Certificate registration

Watson IoT Platform makes use of the following concept: Organization, which is an administrative grouping of resources and services. For fast access and use, a generic organization already exists in Watson IoT Platform, which can be accessed through [QUICKSTART_IBM]. This is an open public pool where devices can be quickly registered and tested. In all other cases, you must always use your own organization.

To connect to your own organization, IBM Watson IoT Platform requires the registration of the CA certificate. This CA certificate enables the organization to recognize the client certificates on IoT devices as trusted so that devices can connect to the server. Any devices that do not have valid signed certificate are denied access and cannot communicate with the server. Optionally, an intermediate CA certificate can also be uploaded.

To register the CA certificate into your account, log in the IBM Cloud console with your account credentials. Click the *Launch* button and you will access the platform dashboard.

Note: Section 4.4 explains how to obtain the CA certificates when working with an A71CH Customer Programmable and an i.MX6UltraLite. Section 5.2 explains it in the case of working with an A71CH Customer Programmable and an FRDM-K64F board. Section 6.1 explains where to obtain them from when an A71CH Provisioned & Programmable is used.

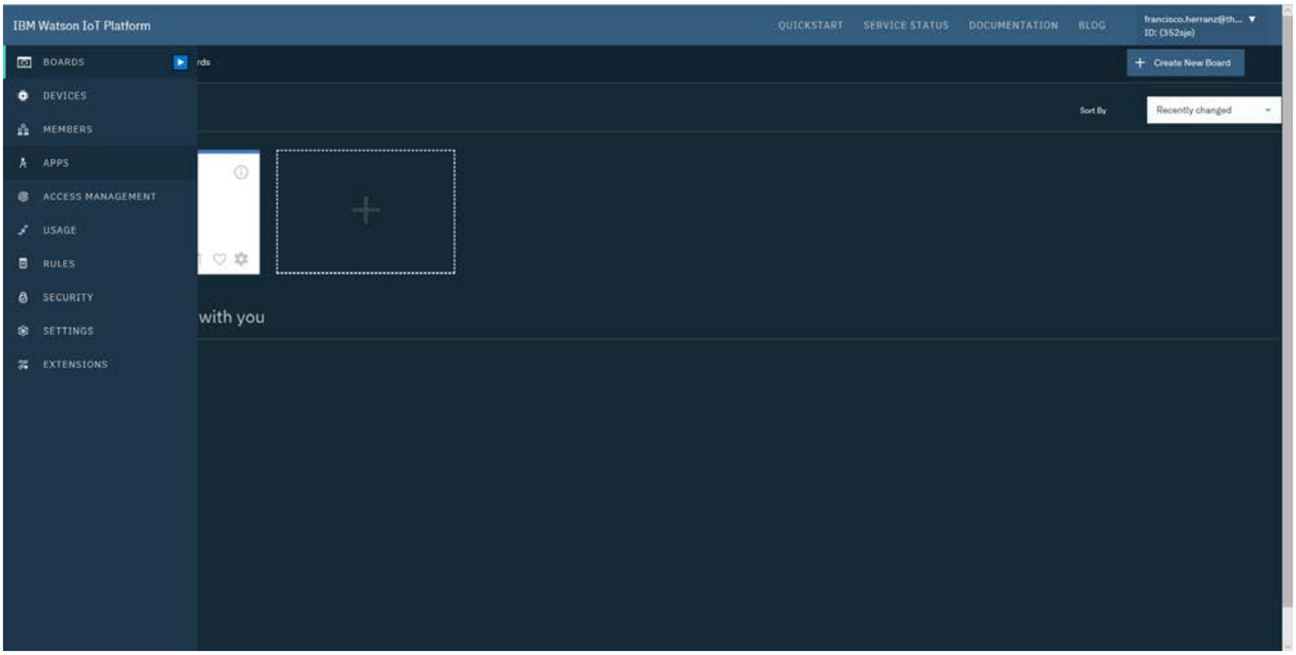


Fig 4. Dashboard of the service.

In the dashboard, follow these steps to upload the CA certificate:

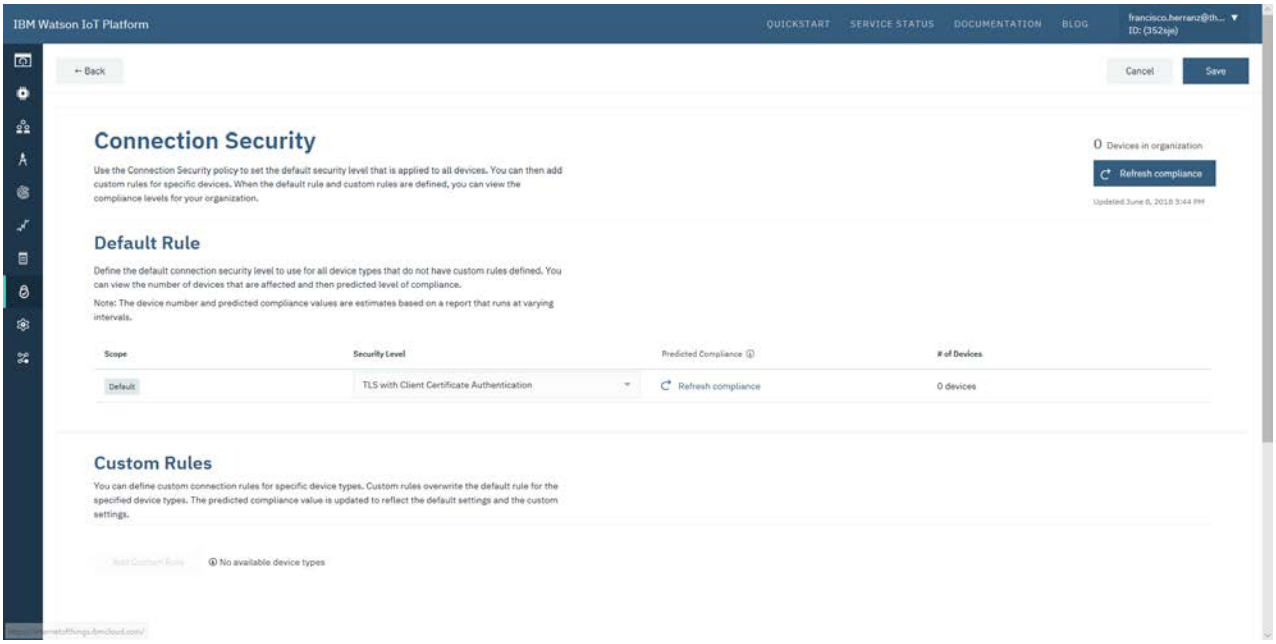
1. Click on ‘**Settings**’ tab.
2. Click on ‘**CA Certificates**’.
3. Click on ‘**Add Certificate**’.
4. A new pop-up appears. Click on ‘**Select a file**’ option and choose the certificate or certificates to be uploaded.
5. Finally, click on ‘**Save**’.

3.3 Configure Connection Security Policy

The next step is to set the ‘*Default Connection Security’* to ‘TLS with Client Certificate Authenticate’.

1. Click on ‘**Security**’ tab.
2. Click on the pencil of ‘**Connection Security**’ to edit the preferences.
3. A new window is loaded, ‘**Connection Security**’. In the Security Level field, select ‘**TLS with Client Certificate Authentication**’.
4. Click on ‘**Refresh compliance**’ to update the default rule.
5. Finally, click on ‘**Save**’.

At this stage, the necessary certificates are uploaded to the platform and the security policies have been changed to accept client certificate authentication.



Connection Security

Use the Connection Security policy to set the default security level that is applied to all devices. You can then add custom rules for specific devices. When the default rule and custom rules are defined, you can view the compliance levels for your organization.

Default Rule

Define the default connection security level to use for all device types that do not have custom rules defined. You can view the number of devices that are affected and then predicted level of compliance.

Note: The device number and predicted compliance values are estimates based on a report that runs at varying intervals.

Scope	Security Level	Predicted Compliance ⓘ	# of Devices
Default	TLS with Client Certificate Authentication	Refresh compliance	0 devices

Custom Rules

You can define custom connection rules for specific device types. Custom rules overwrite the default rule for the specified device types. The predicted compliance value is updated to reflect the default settings and the custom settings.

ⓘ No available device types

Fig 5. Security Policy page.

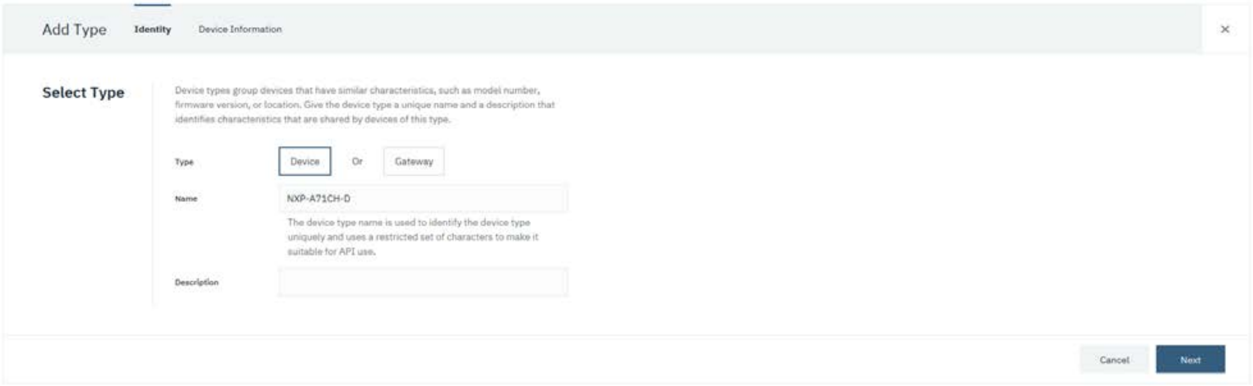
3.4 Device type and device registration.

IoT devices can be configured as *IBM Watson IoT Platform Devices* or *Gateways*. *Gateways* are a specialized class of devices in IBM Watson IoT Platform, and they serve as access points to the IBM Watson IoT Platform for other devices.

A device type and a device need to be registered before your IoT device can connect to the IBM Watson IoT Platform. First, to register a new device type:

1. Click on ‘**Devices**’ tab and ‘**Device Types**’.
2. Click on ‘**Add device type**’.
3. Select either the ‘**Device**’ or ‘**Gateway**’ type and write a name (e.g., NXP-A71CH-D for devices and NXP-A71CH-G). Optionally, add a description.
4. Click ‘**Next**’.
5. Optionally, add information to the rest of the fields. In this guide all the fields have been intentionally left empty. Finally, click ‘**Done**’.

© NXP Semiconductors N.V. 2018. All rights reserved.



Select Type

Device types group devices that have similar characteristics, such as model number, firmware version, or location. Give the device type a unique name and a description that identifies characteristics that are shared by devices of this type.

Type: Device

Name: NXP-A71CH-D

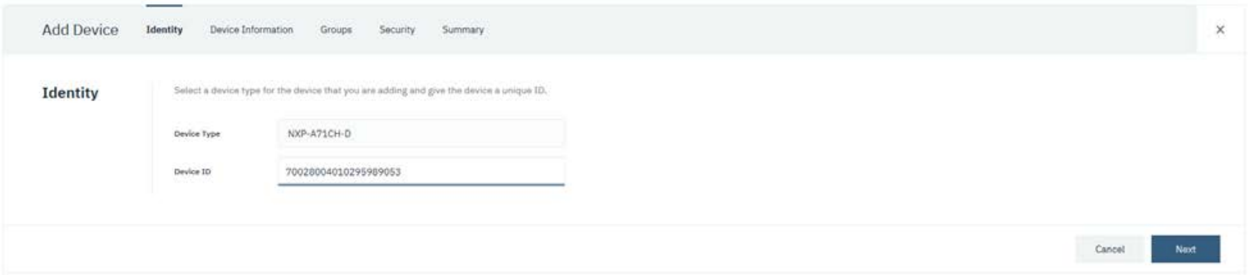
The device type name is used to identify the device type uniquely and uses a restricted set of characters to make it suitable for API use.

Description:

Fig 6. Device type registration.

After the device type is registered, add the device:

1. Click on ‘**Devices**’ tab.
2. Click on ‘**Add Device**’.
3. In the ‘**Identity**’ tab, select as ‘**Device Type**’ the one previously created and add an ID for the Device in the ‘**Device ID**’ tab. Sections 4.2, 5.3, 6.2.3 describe how to obtain this Device ID for each case (using A71CH Customer Programmable type with i.MX6UltraLite, using A71CH Customer Programmable type with FRDM-K64F or using A71CH Provisioned & Programmable, respectively).
4. Click on ‘**Next**’ button. Optionally, more details can be added regarding the ‘**Device Information**’, ‘**Groups**’ or ‘**Security**’ but for this guide all the fields and options have been intentionally left empty.
5. Click on ‘**Done**’ button to create the device.



Identity

Select a device type for the device that you are adding and give the device a unique ID.

Device Type: NXP-A71CH-D

Device ID: 70028004010295989053

Fig 7. Device registration.

Note: After leaving the **‘Security’** tab empty, an Authentication Token will be automatically generated. Once the Device creation is finished, the panel will show how to add this token to your IoT device (or gateway). However, it is important to ignore this, since authentication is carried out using the A71CH security IC.

© NXP Semiconductors N.V. 2018. All rights reserved.

Watson IoT Platform provides REST API for “Bulk Operations”, i.e., get, add or remove multiple devices simultaneously. For details on these “Bulk Operations”, please refer to the documentation available in [BULK_OPS].

3.5 IoT device connection to Watson IoT Platform.

After you register a device with IBM Watson IoT Platform, you can use the registration information to connect the device and start receiving device data.

For illustrative purposes, NXP provides a demo application to connect to IBM Watson IoT Platform running in i.MX6UltraLite board and FRDM-K64F. Please, continue reading Section 4 and Section 5 respectively for a detailed step-by-step guide on how to execute these demos.

4. Trusted connection to Watson IoT Platform using A71CH Customer Programmable type and i.MX6UltraLite demo application

This chapter shows how to use an A71CH Arduino compatible development kit, an i.MX6UltraLite board and a demo application to illustrate how to prepare an A71CH Customer Programmable type for connection to IBM Watson IoT Platform. The source code that can be built and installed in i.MX6UltraLite can be downloaded from this GitHub repository [IMX6UL_REPO].

4.1 Hardware setup

The hardware setup for running this demo consists of:

- MCIMX6UL-EVK: i.MX6UltraLite MCU evaluation board with Internet connectivity via Ethernet. This board will act as the IoT device and will connect to the A71CH through OM3710/A71CHARD Arduino shield.
- OM3710/A71CHARD: Arduino development kit containing a mini PCB board with the A71CH security IC and an Arduino shield compatible with the MCIMX6UL-EVK.
- Development PC: a Windows platform will be used to configure and prepare the Watson IoT Platform account and register a demo CA certificate. Additionally, the i.MX6UltraLite will be controlled from the development PC using Tera Term.

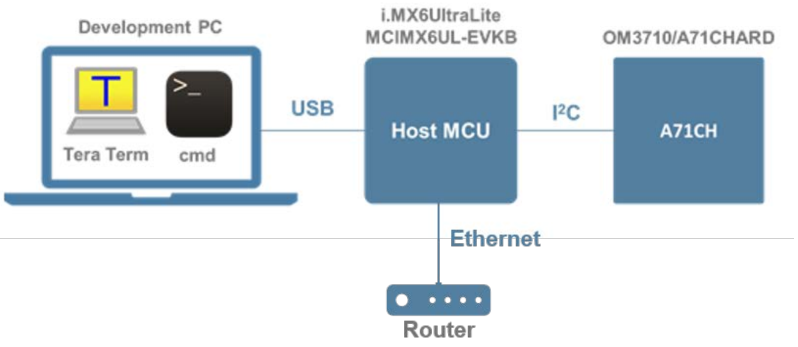


Fig 8. Schematic demo system setup.

Fig 8 depicts the setup that will be used for this demonstration. The A71CH security IC will be connected to the i.MX6UltraLite through the OM3710/A71CHARD Arduino Shield. Finally, an Ethernet cable is needed to connect the i.MX6UltraLite to Internet, as can be seen in Fig 9.

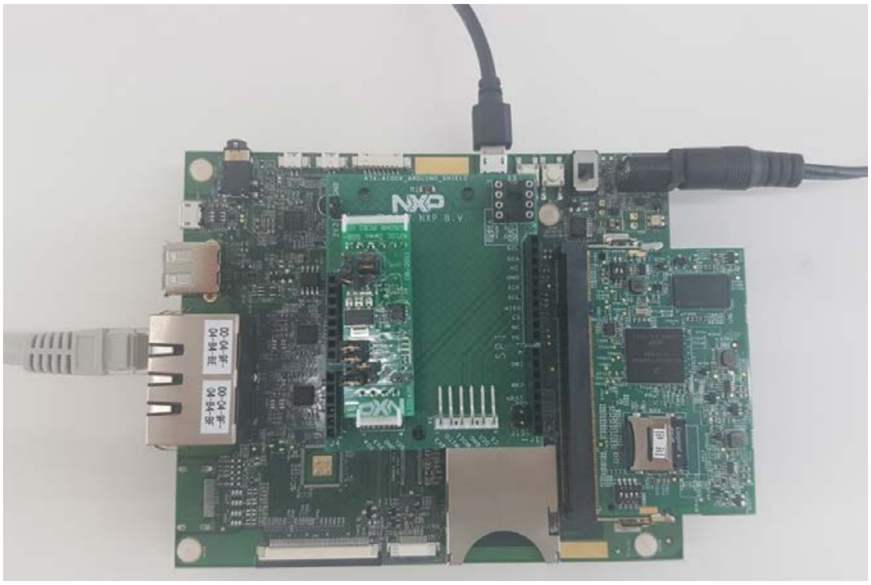
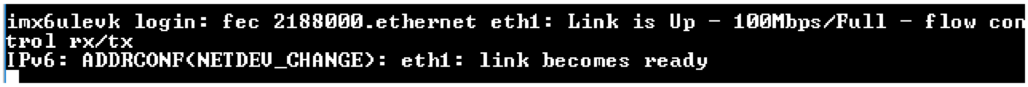


Fig 9. Physical demo system setup.

Note: Before moving to Section 4.2, prepare your hardware and software development environment. Please refer to the ***AN12119 – A71CH Quick start guide for OM3710A71CHARD and i.MX6UltraLite*** in [QUICK_START_IMX6] to perform this task.

4.2 Download source code and how to build it.

The i.MX6UltraLite will be controlled from the development PC using the Tera Term terminal, as explained in [QUICK_START_IMX6]. After initializing the board, Fig 10 shows the message that will appear to confirm that internet connectivity is available through the Ethernet cable. If there is no Ethernet cable during the start-up/booting of the system, it is possible to plug the Ethernet cable later. The message will appear after plugging the cable. To continue with the configuration, simply press Enter.



```
imx6ulevk login: fec 2188000.ethernet eth1: Link is Up - 100Mbps/Full - flow con
trol rx/tx
IPv6: ADDRCONF(NETDEV_CHANGE): eth1: link becomes ready
```

Fig 10. Ethernet connection is available

Use the following command to download the C Client Library source:

```
git clone https://github.com/ibm-watson-iot/iot-nxpimxa71ch-c
```

© NXP Semiconductors N.V. 2018. All rights reserved.

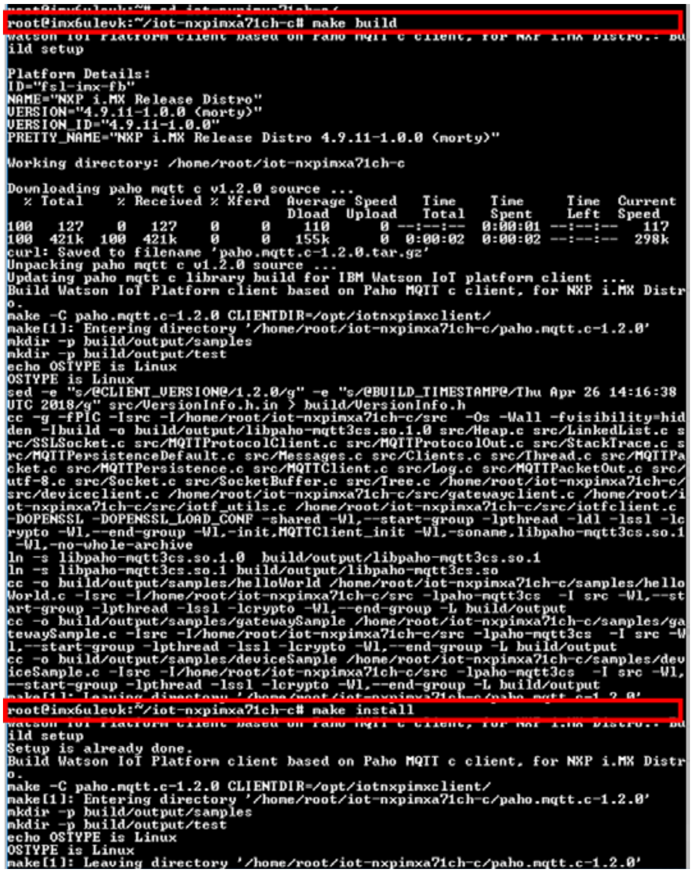
Note: It is possible that the ‘git’ command is not available in the system. There is a workaround using the ‘curl’ command:

```
curl -LJO https://github.com/ibm-watson-iot/iot-nxpimxa71ch-
c/archive/master.zip
unzip iot-nxpimxa71ch-c.master.zip
mv iot-nxpimxa71ch-c.master iot-nxpimxa71ch-c
```

After executing one of these commands, a new folder will appear, named ‘*iot-nxpimxa71ch-c*’. Execute the following commands:

```
cd iot-nxpimxa71ch-c
make build
make install
```

The first command is used to enter the folder created above. The ‘`make build`’ command usually takes a minute to finish (this command compiles the downloaded C Client Library code). Finally, ‘`make install`’ installs the previously compiled code into the system. Fig 11 illustrates the process.



```
root@imx6ulevk:~# cd iot-nxpimxa71ch-c/
root@imx6ulevk:~/iot-nxpimxa71ch-c# make build
Watson IoT Platform client based on Paho MQTT c client, for NXP i.MX Distro.: Bu
ild setup

Platform Details:
ID="fsl-imx-fb"
NAME="NXP i.MX Release Distro"
VERSION="4.9.11-1.0.0 (morty)"
VERSION_ID="4.9.11-1.0.0"
PRETTY_NAME="NXP i.MX Release Distro 4.9.11-1.0.0 (morty)"

Working directory: /home/root/iot-nxpimxa71ch-c

Downloading paho mqtt c v1.2.0 source ...
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   127    0   127    0     0    110      0 --:--:--  0:00:01 --:--:--   117
100  421k  100  421k    0     0   155k      0  0:00:02  0:00:02 --:--:--  298k
curl: Saved to filename 'paho.mqtt.c-1.2.0.tar.gz'
Unpacking paho mqtt c v1.2.0 source ...
Updating paho mqtt c library build for IBM Watson IoT platform client ...
Build Watson IoT Platform client based on Paho MQTT c client, for NXP i.MX Distr
o.
make -C paho.mqtt.c-1.2.0 CLIENTDIR=/opt/iotnxpimxclient/
make[1]: Entering directory '/home/root/iot-nxpimxa71ch-c/paho.mqtt.c-1.2.0'
mkdir -p build/output/samples
mkdir -p build/output/test
echo OSTYPE is Linux
OSTYPE is Linux
sed -e "s/@CLIENT_VERSION@/1.2.0/g" -e "s/@BUILD_TIMESTAMP@/Thu Apr 26 14:16:38
UTC 2018/g" src/VersionInfo.h.in > build/VersionInfo.h
cc -g -fPIC -Isrc -I/home/root/iot-nxpimxa71ch-c/src  -Os -Wall -fvisibility=hid
den -Ibuild -o build/output/libpaho-mqtt3cs.so.1.0 src/Heap.c src/LinkedList.c s
rc/SSLSocket.c src/MQTTProtocolClient.c src/MQTTProtocolOut.c src/StackTrace.c s
rc/MQTTPersistenceDefault.c src/Messages.c src/Clients.c src/Thread.c src/MQTTPa
cket.c src/MQTTPersistence.c src/MQTTClient.c src/Log.c src/MQTTPacketOut.c src/
utf-8.c src/Socket.c src/SocketBuffer.c src/Tree.c /home/root/iot-nxpimxa71ch-c/
src/deviceclient.c /home/root/iot-nxpimxa71ch-c/src/gatewayclient.c /home/root/i
ot-nxpimxa71ch-c/src/iotf_utils.c /home/root/iot-nxpimxa71ch-c/src/iotfclient.c
-DOPENSSL -DOPENSSL_LOAD_CONF -shared -Wl,--start-group -lpthread -ldl -lssl -lc
rypto -Wl,--end-group -Wl,-init,MQTTClient_init -Wl,-soname,libpaho-mqtt3cs.so.1
 -Wl,-no-whole-archive
ln -s libpaho-mqtt3cs.so.1.0  build/output/libpaho-mqtt3cs.so.1
ln -s libpaho-mqtt3cs.so.1 build/output/libpaho-mqtt3cs.so
cc -o build/output/samples/helloWorld /home/root/iot-nxpimxa71ch-c/samples/hello
World.c -Isrc -I/home/root/iot-nxpimxa71ch-c/src -lpaho-mqtt3cs  -I src -Wl,--st
art-group -lpthread -lssl -lcrypto -Wl,--end-group -L build/output
cc -o build/output/samples/gatewaySample /home/root/iot-nxpimxa71ch-c/samples/ga
tewaySample.c -Isrc -I/home/root/iot-nxpimxa71ch-c/src -lpaho-mqtt3cs  -I src -W
l,--start-group -lpthread -lssl -lcrypto -Wl,--end-group -L build/output
cc -o build/output/samples/deviceSample /home/root/iot-nxpimxa71ch-c/samples/dev
iceSample.c -Isrc -I/home/root/iot-nxpimxa71ch-c/src -lpaho-mqtt3cs  -I src -Wl,
--start-group -lpthread -lssl -lcrypto -Wl,--end-group -L build/output
make[1]: Leaving directory '/home/root/iot-nxpimxa71ch-c/paho.mqtt.c-1.2.0'
root@imx6ulevk:~/iot-nxpimxa71ch-c# make install
Watson IoT Platform client based on Paho MQTT c client, for NXP i.MX Distro.: Bu
ild setup
Setup is already done.
Build Watson IoT Platform client based on Paho MQTT c client, for NXP i.MX Distr
o.
make -C paho.mqtt.c-1.2.0 CLIENTDIR=/opt/iotnxpimxclient/
make[1]: Entering directory '/home/root/iot-nxpimxa71ch-c/paho.mqtt.c-1.2.0'
mkdir -p build/output/samples
mkdir -p build/output/test
echo OSTYPE is Linux
OSTYPE is Linux
make[1]: Leaving directory '/home/root/iot-nxpimxa71ch-c/paho.mqtt.c-1.2.0'
```

Fig 11. Building and installing the C Client Library

Once the installation is finished, the client library, header files, sample client binaries, configuration files, and certificates can be found in the following directories:

Table 1. Installed files

Directory Location	Content
/usr/local/lib	Client libraries
/usr/local/include	Header files for device client build
/opt/iotnxpimxclient/bin	Device client sample binaries
/opt/iotnxpimxclient/config	Configuration files for device client samples
/opt/iotnxpimxclient/certs	Certificates used by device client samples

4.3 Credentials provisioning

The next step is to create the necessary credentials. The process of creating the credentials and injecting them into the A71CH Security IC is performed by the bash shell script ‘`provisionA71CH_WatsonIoT.sh`’. Copy this file from the path ‘`home/root/iot-nxpimxa71ch-c/samples`’ to the path ‘`home/root/tools`’.

```
cp /home/root/iot-nxpimxa17ch-c/samples/provisionA71CH_WatsonIoT.sh
/home/root/tools/
```

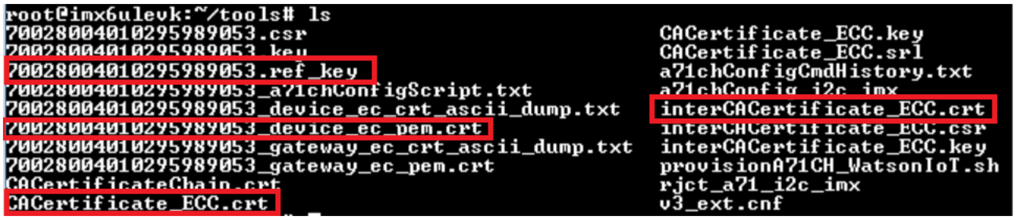
Once the file has been copied, execute it using the following commands.

```
cd /home/root/tools

chmod +x provisionA71CH_WatsonIoT.sh

./provisionA71CH_WatsonIoT.sh
```

The `./provisionA71CH_WatsonIoT.sh` scripts creates:

- A Root CA key pair, an Intermediate CA key pair and the associated certificates are created.
- A device key pair on the NIST-P256 ECC Curve is created.
- A device-specific certificate and a gateway-specific certificate, signed by the Intermediate CA, including the UID of the attached A71CH and the public key of the device key pair are issued.

The created credentials are injected into the A71CH. Fig 12 shows the different keys, certificate signing requests and certificates that have been generated by the script. Only the highlighted elements will be used for the connection to Watson IoT Platform. The rest are still available for other applications. Note down your device UID (the number in the name of the files) to register the device into the Watson IoT Platform (use it as the device ID mentioned in Section 3.4).

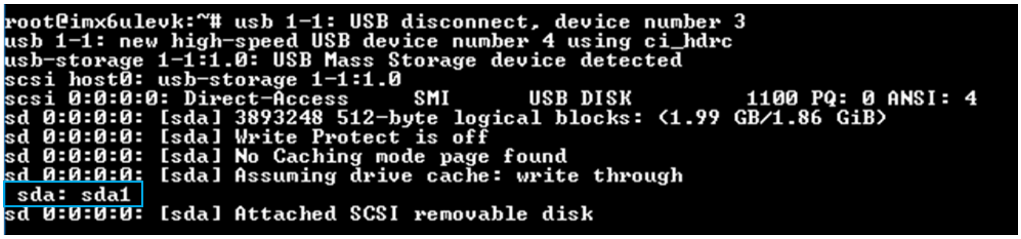


```
root@imx6ulevk:~/tools# ls
70028004010295989053.csr                                  CACertificate_ECC.key
70028004010295989053.key                                  CACertificate_ECC.srl
70028004010295989053.ref_key                              a71chConfigCmdHistory.txt
70028004010295989053_a71chConfigScript.txt                a71chConfig_i2c_imx
70028004010295989053_device_ec_crt_ascii_dump.txt         interCACertificate_ECC.crt
70028004010295989053_device_ec_pem.crt                    interCACertificate_ECC.csr
70028004010295989053_gateway_ec_crt_ascii_dump.txt        interCACertificate_ECC.key
70028004010295989053_gateway_ec_pem.crt                   provisionA71CH_WatsonIoT.sh
CACertificateChain.crt                                    rjct_a71_i2c_imx
CACertificate_ECC.crt                                     v3_ext.cnf
```

Fig 12. List of keys and certificates created.

4.4 Extraction of CA certificates.

The next step is to upload the CA Certificate and Intermediate CA certificate to Watson IoT Platform. First connect a USB drive into the i.MX6UltraLite board. A new message will appear indicating that a USB Mass Storage device has been detected. Fig 13 illustrates the content of this message. The most important part is the name of the device that Linux assigns to the detected USB drive. In this case, the name is '**sda1**' (highlighted in blue).



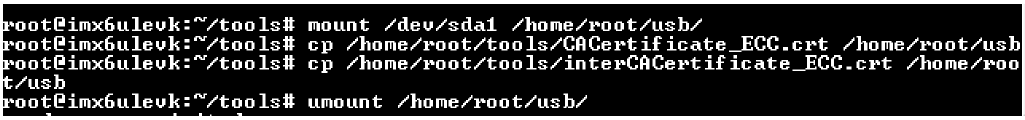
```
root@imx6ulevk:~# usb 1-1: USB disconnect, device number 3
usb 1-1: new high-speed USB device number 4 using ci_hdrc
usb-storage 1-1:1.0: USB Mass Storage device detected
scsi host0: usb-storage 1-1:1.0
scsi 0:0:0:0: Direct-Access     SMI      USB DISK         1100 PQ: 0 ANSI: 4
sd 0:0:0:0: [sda] 3893248 512-byte logical blocks: (1.99 GB/1.86 GiB)
sd 0:0:0:0: [sda] Write Protect is off
sd 0:0:0:0: [sda] No Caching mode page found
sd 0:0:0:0: [sda] Assuming drive cache: write through
 sda: sda1
sd 0:0:0:0: [sda] Attached SCSI removable disk
```

Fig 13. 'Sda1' is the name of the USB drive

Once the USB drive has been detected, mount the device, copy the certificates to it and finally, unmount the USB drive:

- '**mkdir**' command is used to create a mount point; i.e., a folder where the content of the USB will be shown.
- '**mount**' command indicating the desired USB (in this case, it is 'sda1') and the mounting point (the USB folder previously created).
- '**cp**' commands to copy the root CA certificate, named CACertificate_ECC.crt, and the intermediate certificate, named interCACertificate_ECC.crt, from their original folder ('/home/root/tools') to the USB drive.
- Finally, '**umount**' command unmounts the USB drive. After executing this command, it can be unplugged from the i.MX6UltraLite.

Fig 14 depicts the commands used to perform these operations:



```
root@imx6ulevk:~/tools# mount /dev/sda1 /home/root/usb/
root@imx6ulevk:~/tools# cp /home/root/tools/CACertificate_ECC.crt /home/root/usb
root@imx6ulevk:~/tools# cp /home/root/tools/interCACertificate_ECC.crt /home/roo
t/usb
root@imx6ulevk:~/tools# umount /home/root/usb/
```

Fig 14. Mounting the USB drive, copying the certificates and unmounting the USB device.

The root CA certificate and the intermediate certificate are now stored in the USB drive. These certificates must be uploaded to the dashboard (personal space) of the Watson IoT Platform, as explained in Section 3.2.

4.5 Configure and connect the IoT device

Register the device type and device to Watson IoT Platform as indicated in Section 3.4. After that, update the i.MX6UltraLite configuration file according to the registered device in Watson IoT platform. This file contains details about the organization ID, device ID and the path to the different certificates and credentials created in Section 4.3. The organization id can be found in the URL of the dashboard:

```
https://org_id.internetofthings.ibmcloud.com/dashboard/#/overview
```

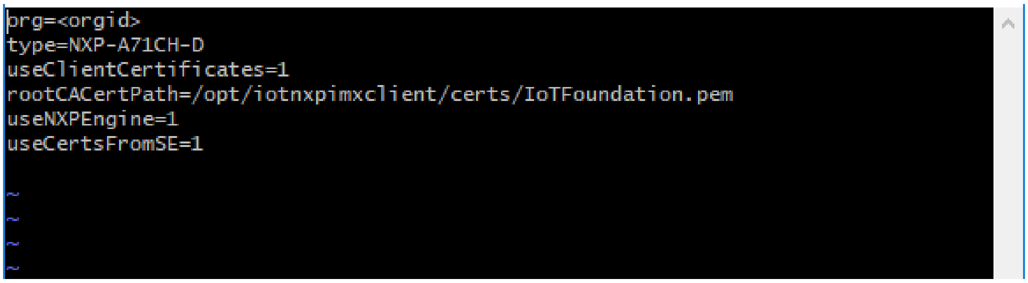
This configuration file can be found in the following folder:

```
cd /opt/iotnxpimxclient/config/
```

If the A71CH is to be used in an IoT device, the file to be used is ‘**device_a71ch.cfg**’. On the other hand, if it is being used in an IoT gateway, the file is ‘**gateway_a71ch.cfg**’. However, the editing process is the same, so it is only explained for the case of the device. To edit this file, ‘**vi**’ is used. Vi is present in *almost* all the Unix devices, and the environment requires a small amount of resources. To edit the configuration file:

```
vi device_a71ch.cfg
```

Fig 15 shows the content of the file. To start adding text, hit the key ‘**i**’ and modify the org field according to your organization ID. Press ‘**ESC**’ to exit the adding text mode and type ‘**:wq**’ to save the document and exit Vi.

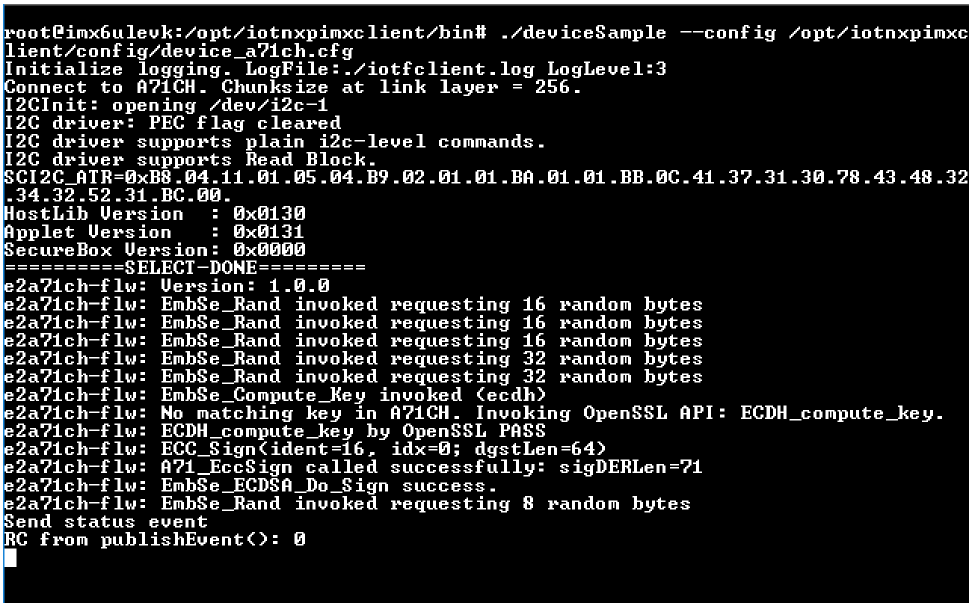


```
org=<orgid>
type=NXP-A71CH-D
useClientCertificates=1
rootCACertPath=/opt/iotnxpimxclient/certs/IoTFoundation.pem
useNXPEngine=1
useCertsFromSE=1

~
~
~
~
```

Fig 15. Content of ‘device_a71ch.cfg’ file

The last step is to connect the device to the Watson IoT Platform. Move to the `/opt/iotnxpimxclient/bin/` directory and run either the device sample or the gateway sample, depending on the device type that has been created in Section 3.4.

```
cd /opt/iotnxpimxclient/bin/
./deviceSample --config /opt/iotnxpimxclient/config/device_a71ch.cfg
./gatewaySample --config /opt/iotnxpimxclient/config/gateway_a71ch.cfg
```

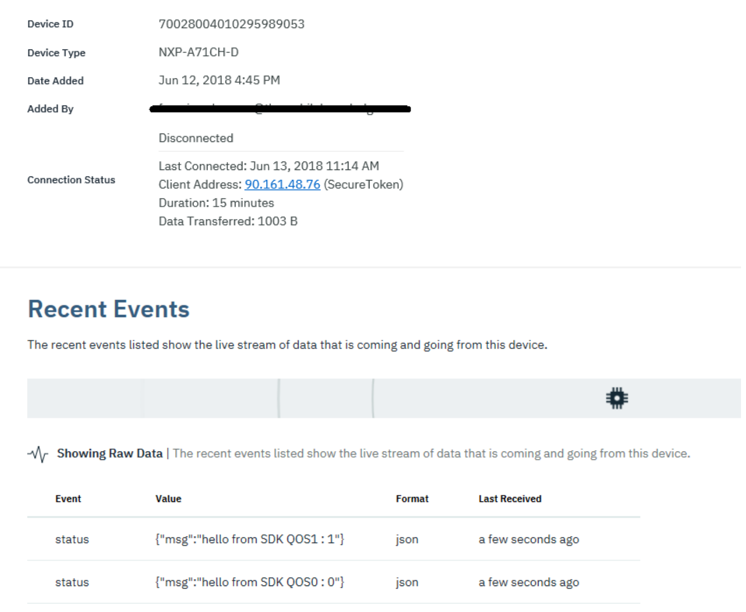


```
root@imx6ulevk:/opt/iotnxpimxclient/bin# ./deviceSample --config /opt/iotnxpimxc
lient/config/device_a71ch.cfg
Initialize logging. LogFile:./iotfclient.log LogLevel:3
Connect to A71CH. Chunksize at link layer = 256.
I2CInit: opening /dev/i2c-1
I2C driver: PEC flag cleared
I2C driver supports plain i2c-level commands.
I2C driver supports Read Block.
SCI2C_ATR=0xB8.04.11.01.05.04.B9.02.01.01.BA.01.01.BB.0C.41.37.31.30.78.43.48.32
.34.32.52.31.BC.00.
HostLib Version  : 0x0130
Applet Version   : 0x0131
SecureBox Version: 0x0000
==========SELECT-DONE=========
e2a71ch-flw: Version: 1.0.0
e2a71ch-flw: EmbSe_Rand invoked requesting 16 random bytes
e2a71ch-flw: EmbSe_Rand invoked requesting 16 random bytes
e2a71ch-flw: EmbSe_Rand invoked requesting 16 random bytes
e2a71ch-flw: EmbSe_Rand invoked requesting 32 random bytes
e2a71ch-flw: EmbSe_Rand invoked requesting 32 random bytes
e2a71ch-flw: EmbSe_Compute_Key invoked (ecdh)
e2a71ch-flw: No matching key in A71CH. Invoking OpenSSL API: ECDH_compute_key.
e2a71ch-flw: ECDH_compute_key by OpenSSL PASS
e2a71ch-flw: ECC_Sign(ident=16, idx=0; dgstLen=64)
e2a71ch-flw: A71_EccSign called successfully: sigDERLen=71
e2a71ch-flw: EmbSe_ECDSA_Do_Sign success.
e2a71ch-flw: EmbSe_Rand invoked requesting 8 random bytes
Send status event
RC from publishEvent(): 0
```

Fig 16. i.MX6UltraLite sending data to Watson IoT Platform.

To see the data in the Watson IoT Platform dashboard, follow these steps:

1. Click on ‘**Devices**’ tab.
2. Select the created device and click on ‘**More info**’ button.
3. ‘**Recent Events**’ will be refreshed every time that the i.MX6UltraLite sends data to Watson IoT Platform.



Device ID	70028004010295989053
Device Type	NXP-A71CH-D
Date Added	Jun 12, 2018 4:45 PM
Added By	[redacted]
Connection Status	Disconnected Last Connected: Jun 13, 2018 11:14 AM Client Address: 90.161.48.76 (SecureToken) Duration: 15 minutes Data Transferred: 1003 B

Recent Events

The recent events listed show the live stream of data that is coming and going from this device.

Showing Raw Data | The recent events listed show the live stream of data that is coming and going from this device.

Event	Value	Format	Last Received
status	{"msg":"hello from SDK QOS1 : 1"}	json	a few seconds ago
status	{"msg":"hello from SDK QOS0 : 0"}	json	a few seconds ago

Fig 17. i.MX6UltraLite sending data to Watson IoT Platform.

5. Trusted connection to Watson IoT Platform using A71CH Customer Programmable type and FRDM-K64F demo application

This section details how to use an A71CH Arduino compatible development kit, a FRDM-K64F board and a demo application to illustrate how to prepare an ***A71CH Customer Programmable type*** for connection to IBM Watson IoT Platform. The software necessary to run this demo is available in the Release 1.5.0 (or later) of the A71CH Host Software Package.

5.1 Hardware setup

The hardware setup for running this demo consists of:

- FRDM-K64F: Freedom Development Platform for Kinetis K64 with Internet connectivity via Ethernet. This board will act as the IoT device and will connect to the A71CH through the OM3710/A71CHARD Arduino shield.
- OM3710/A71CHARD: Arduino development kit containing a mini PCB board with the A71CH security IC and an Arduino shield compatible with the MCIMX6UL-EVK.
- Development PC: a Windows platform will be used to configure and prepare the Watson IoT Platform account and register a demo CA certificate. Additionally, the i.MX6UltraLite will be controlled from the development PC using Tera Term.

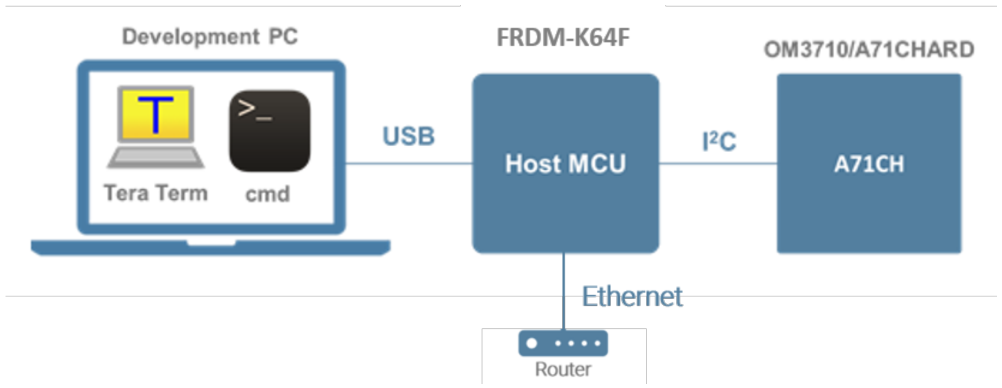


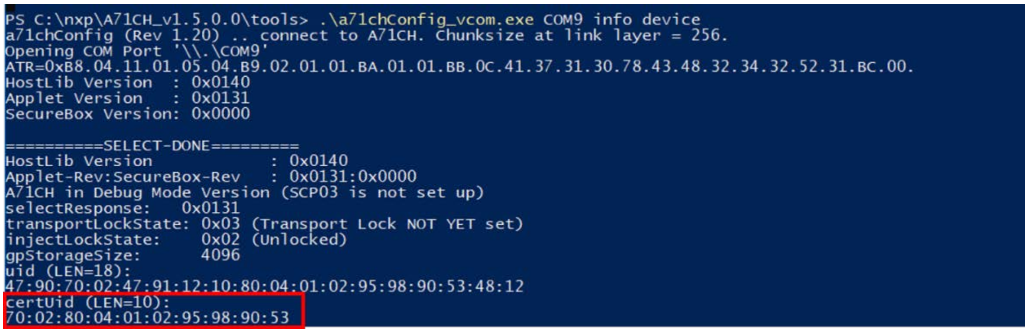
Fig 18. Schematic demo system setup.

Note: Before moving to Section 5.2, prepare your hardware and software development environment. Please refer to the ***AN12135 – A71CH Quick start guide for OM3710A71CHARD and Kinetis*** in [QUICKSTART_KIN] to perform this task.

5.2 Create device credentials

The public/private key pair will be generated using the Development PC. Later, they will be securely stored in the A71CH security IC. To create the credentials of the device, you can use the scripts located in the folder `/demos/ibm_watson_demo/scripts`.

The file *RunOnce_CreateCaCertificate.bat* will create a demo CA certificate and the *CreateDeviceCertificate.bat* file will create the credentials of the device. Using the A71CH Configure Tool, the certUID of the attached A71CH must be retrieved, as shown in Fig 19.

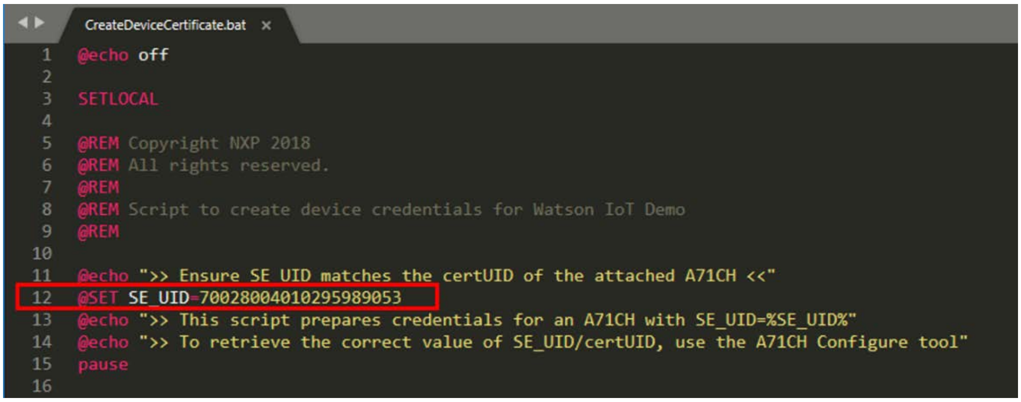


```
PS C:\nxp\A71CH_v1.5.0.0\tools> .\a71chConfig_vcom.exe COM9 info device
a71chConfig (Rev 1.20) .. connect to A71CH. Chunksize at link layer = 256.
Opening COM Port '\\.\COM9'
ATR=0xB8.04.11.01.05.04.B9.02.01.01.BA.01.01.BB.0C.41.37.31.30.78.43.48.32.34.32.52.31.BC.00.
HostLib Version  : 0x0140
Applet Version   : 0x0131
SecureBox Version: 0x0000

==========SELECT-DONE=========
HostLib Version            : 0x0140
Applet-Rev:SecureBox-Rev   : 0x0131:0x0000
A71CH in Debug Mode Version (SCP03 is not set up)
selectResponse:   0x0131
transportLockState: 0x03 (Transport Lock NOT YET set)
injectLockState:    0x02 (Unlocked)
gpStorageSize:      4096
uid (LEN=18):
47:90:70:02:47:91:12:10:80:04:01:02:95:98:90:53:48:12
certUid (LEN=10):
70:02:80:04:01:02:95:98:90:53
```

Fig 19. A71CH Configure Tool command to retrieve certUID.

Note: Use the ***AN12135 – A71CH Quick start guide for OM3710A71CHARD and Kinetis*** document (Section 7.2.3) in [QUICKSTART_KIN] to see how to obtain the certUID using the A71CH Configure Tool.

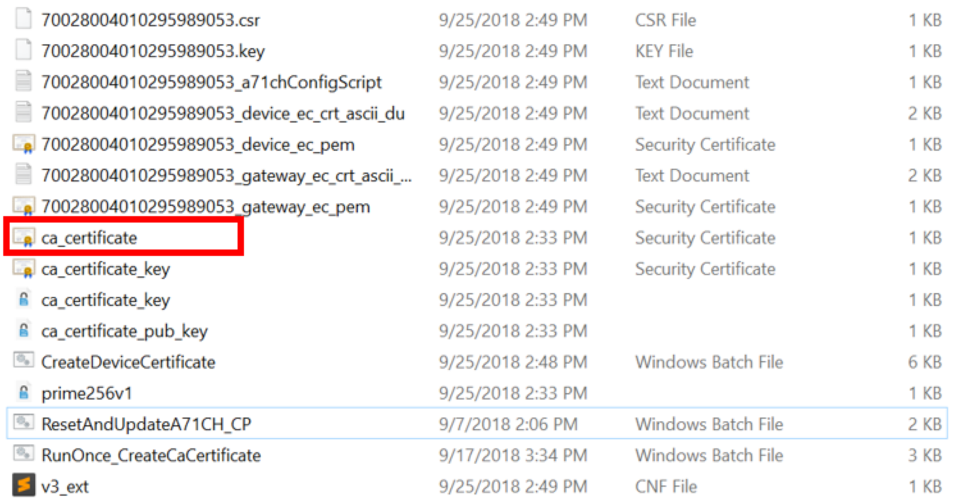
Make sure that the SE_UID in *CreateDeviceCertificate.bat* matches the certUID of the attached A71CH, as in Fig 20, and run the batch file to create the device credentials.



```
@echo off

SETLOCAL

@REM Copyright NXP 2018
@REM All rights reserved.
@REM
@REM Script to create device credentials for Watson IoT Demo
@REM

@echo ">> Ensure SE UID matches the certUID of the attached A71CH <<"
@SET SE_UID=70028004010295989053
@echo ">> This script prepares credentials for an A71CH with SE_UID=%SE_UID%"
@echo ">> To retrieve the correct value of SE_UID/certUID, use the A71CH Configure tool"
pause

```

Fig 20. Line of the CreateDeviceCertificate.bat file where the UID must be modified.

Fig 21 shows the contents of the *scripts* folder, once all the device credentials have been created. The certificate on a red box must be uploaded to Watson IoT Platform, according to the procedure of Section 3.2.



70028004010295989053.csr	9/25/2018 2:49 PM	CSR File	1 KB
70028004010295989053.key	9/25/2018 2:49 PM	KEY File	1 KB
70028004010295989053_a71chConfigScript	9/25/2018 2:49 PM	Text Document	1 KB
70028004010295989053_device_ec_crt_ascii_du	9/25/2018 2:49 PM	Text Document	2 KB
70028004010295989053_device_ec_pem	9/25/2018 2:49 PM	Security Certificate	1 KB
70028004010295989053_gateway_ec_crt_ascii_...	9/25/2018 2:49 PM	Text Document	2 KB
70028004010295989053_gateway_ec_pem	9/25/2018 2:49 PM	Security Certificate	1 KB
ca_certificate	9/25/2018 2:33 PM	Security Certificate	1 KB
ca_certificate_key	9/25/2018 2:33 PM	Security Certificate	1 KB
ca_certificate_key	9/25/2018 2:33 PM		1 KB
ca_certificate_pub_key	9/25/2018 2:33 PM		1 KB
CreateDeviceCertificate	9/25/2018 2:48 PM	Windows Batch File	6 KB
prime256v1	9/25/2018 2:33 PM		1 KB
ResetAndUpdateA71CH_CP	9/7/2018 2:06 PM	Windows Batch File	2 KB
RunOnce_CreateCaCertificate	9/17/2018 3:34 PM	Windows Batch File	3 KB
v3_ext	9/25/2018 2:49 PM	CNF File	1 KB

Fig 21. Contents of the *scripts* folder, with the created credentials.

Finally, follow the steps in Section 5.3 to use *ResetAndUpdateA71CH_CP.bat* to provision the attached A71CH CP type.

5.3 Credentials provisioning into the A71CH

To inject the keys in the A71CH, connect the FRDM-K64F board to the Development PC with a mini USB cable through the OpenSDA port. Using MCUXpresso, you need to import and build the ‘`stub_frdmk64f_bm_vcomA71CH`’ project (located in `projects_frdmk64f` folder).

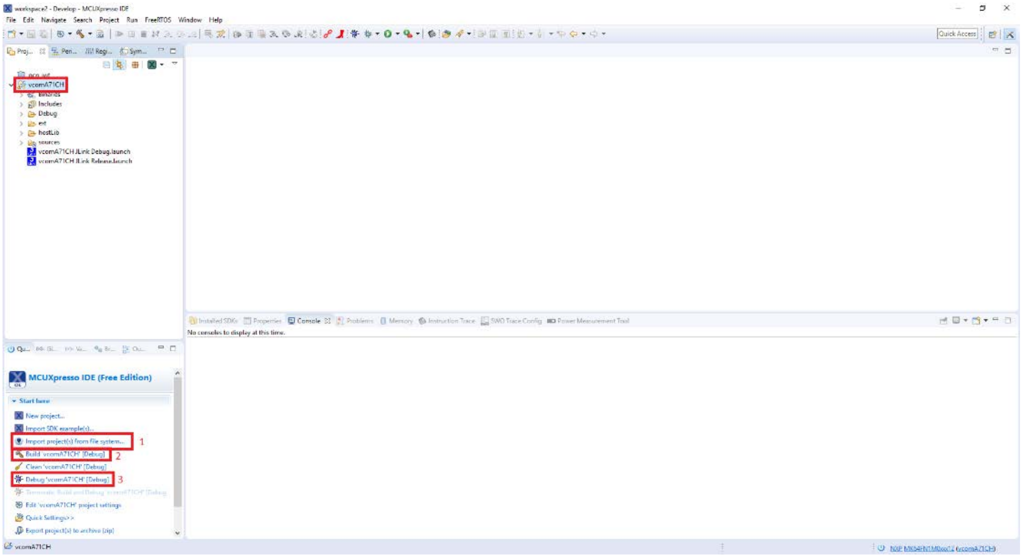

workspace2 - Develop - MCUXpresso IDE
File Edit Navigate Search Project Run FreeRTOS Window Help
Quick Access
Proj... Peri... Regi... Sym...
aca_ws
vcomA71CH
Binaries
Includes
Debug
ext
hostLib
sources
vcomA71CH JLink Debug.launch
vcomA71CH JLink Release.launch
Installed SDKs Properties Console Problems Memory Instruction Trace SWO Trace Config Power Measurement Tool
No consoles to display at this time.
Qu...
MCUXpresso IDE (Free Edition)
Start here
New project...
Import SDK example(s)...
Import project(s) from file system...
1
Build 'vcomA71CH' [Debug]
2
Clean 'vcomA71CH' [Debug]
Debug 'vcomA71CH' [Debug]
3
Terminate, Build and Debug 'vcomA71CH' [Debug]
Edit 'vcomA71CH' project settings
Quick Settings>>
Export project(s) to archive (zip)
vcomA71CH

Fig 22. MCUXpresso IDE.

Then, connect the K64F mini USB port to the Development PC and check the port name assigned to the board (Virtual Com Port).

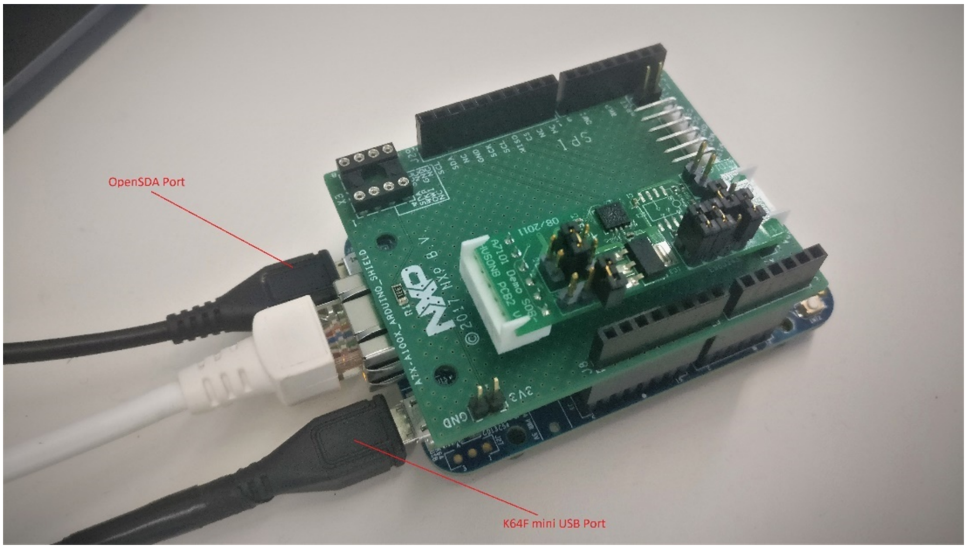


Fig 23. Setup of the example with both mini-USB cables and the Ethernet connection.

Fig 23 shows the necessary setup to download the keys on the A71CH.

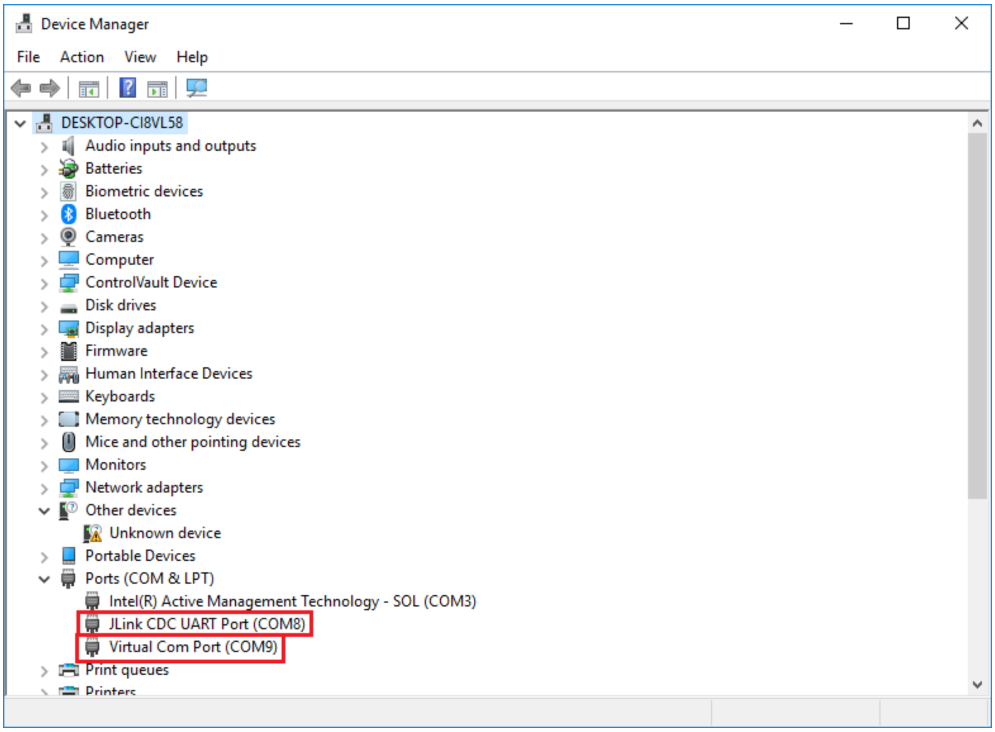
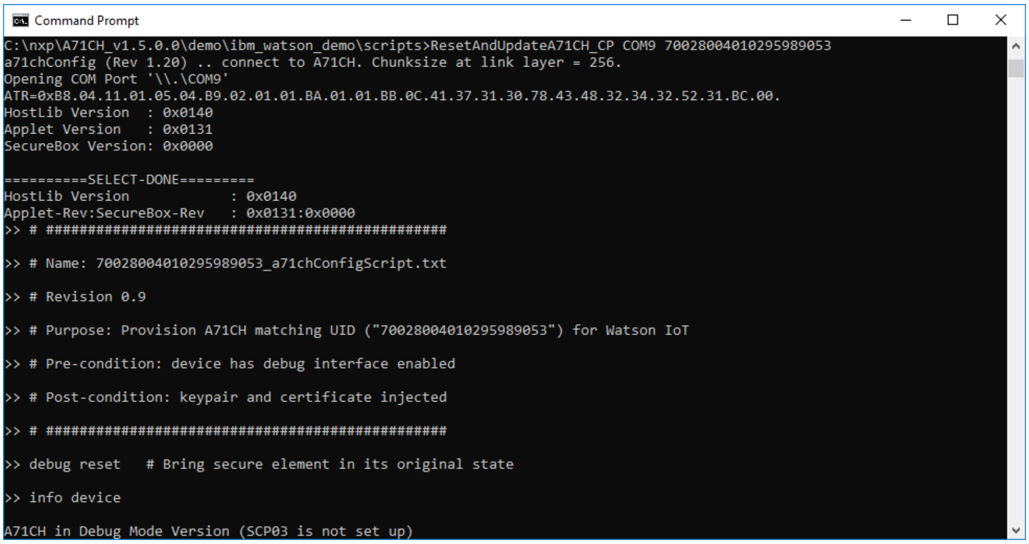


Fig 24. The Virtual Com Port name is COM9.

Using Windows Command Prompt, run the batch file *ResetAndUpdateA71CH_CP.bat* with the Windows command line, specifying the COM port name and the Cert UID. This file is in the folder `/demos/ibm_watson_demo/scripts`.



```
C:\nxp\A71CH_v1.5.0.0\demo\ibm_watson_demo\scripts>ResetAndUpdateA71CH_CP COM9 70028004010295989053
a71chConfig (Rev 1.20) .. connect to A71CH. Chunksize at link layer = 256.
Opening COM Port '\\.\COM9'
ATR=0xB8.04.11.01.05.04.B9.02.01.01.BA.01.01.BB.0C.41.37.31.30.78.43.48.32.34.32.52.31.BC.00.
HostLib Version  : 0x0140
Applet Version   : 0x0131
SecureBox Version: 0x0000

==========SELECT-DONE=========
HostLib Version            : 0x0140
Applet-Rev:SecureBox-Rev   : 0x0131:0x0000
>> # ################################################

>> # Name: 70028004010295989053_a71chConfigScript.txt

>> # Revision 0.9

>> # Purpose: Provision A71CH matching UID ("70028004010295989053") for Watson IoT

>> # Pre-condition: device has debug interface enabled

>> # Post-condition: keypair and certificate injected

>> # ################################################

>> debug reset   # Bring secure element in its original state

>> info device

A71CH in Debug Mode Version (SCP03 is not set up)
```

Fig 25. Execution of the *ResetAndUpdateA71CH_CP.bat* batch file.

© NXP Semiconductors N.V. 2018. All rights reserved.

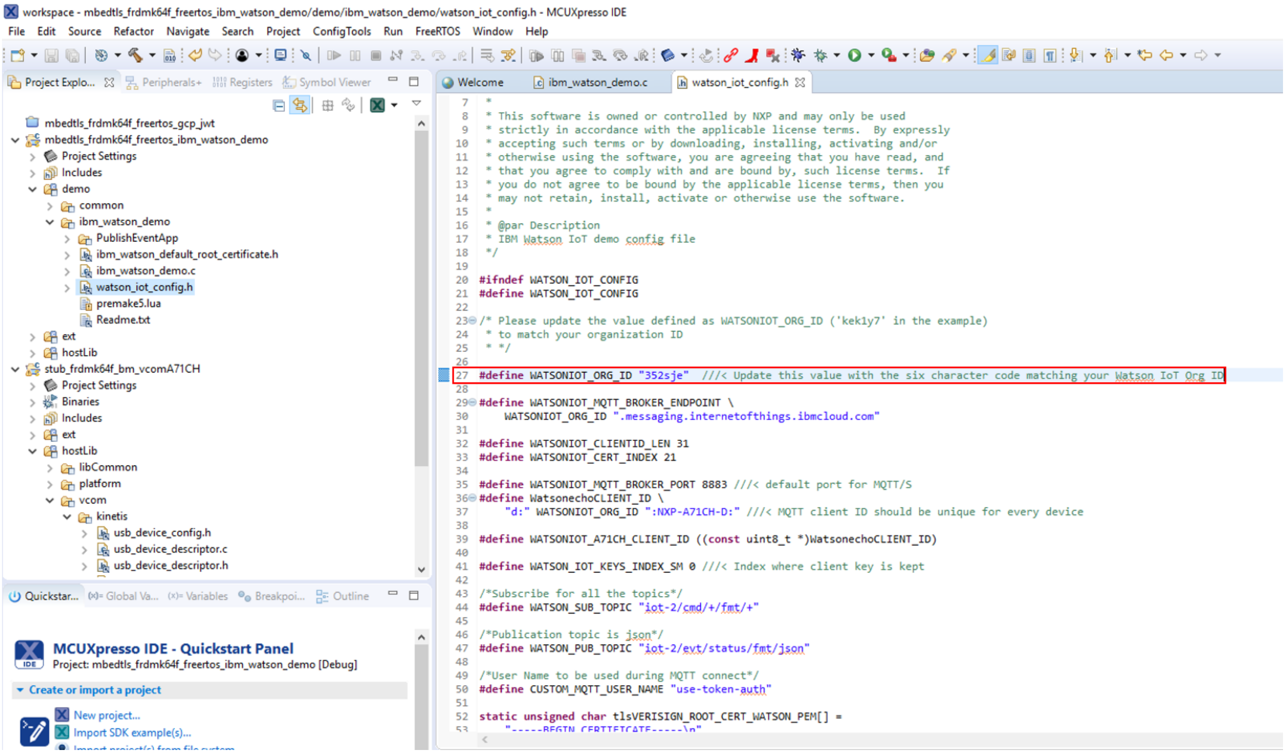
This file will reset the device and insert the keys and the certificates to the A71CH security IC. The Cmd tool will show the different keys being injected.

5.4 Connect your device to IBM Watson IoT Platform

The `projects_frdmk64f/mbedTLS_frdmk64f_freertos_ibm_watson_demo` needs to be executed to connect your device to IBM Watson IoT Platform. To load the application, click on '*Import project(s) from file system'* from MCUXpresso Quick Start. In the '*Project directory'* option, click on Browse and move to the directory where the A71CH Host Library has been installed. Select the project in `projects_frdmk64f/mbedTLS_frdmk64f_freertos_ibm_watson_demo` and, finally, click on '*Finish'*.

It is necessary to change the Organization ID in the MCUXpresso project. As shown in Fig 26, go to the `watson_iot_config.h` file and write the ID of your Organization in the WATSONIOT_ORG_ID macro. The Organization ID can be found in the URL of the IBM Watson IoT Platform dashboard:

```
https://org_id.internetofthings.ibmcloud.com/dashboard/#/overview
```


workspace - mbedtls_frdmk64f_freertos_ibm_watson_demo/demo/ibm_watson_demo/watson_iot_config.h - MCUXpresso IDE
File Edit Source Refactor Navigate Search Project ConfigTools Run FreeRTOS Window Help
Project Explo... Peripherals+ Registers Symbol Viewer
mbedtls_frdmk64f_freertos_gcp_jwt
mbedtls_frdmk64f_freertos_ibm_watson_demo
Project Settings
Includes
demo
common
ibm_watson_demo
PublishEventApp
ibm_watson_default_root_certificate.h
ibm_watson_demo.c
watson_iot_config.h
premake5.lua
Readme.txt
ext
hostLib
stub_frdmk64f_bm_vcomA71CH
Project Settings
Binaries
Includes
ext
hostLib
libCommon
platform
vcom
kinetis
usb_device_config.h
usb_device_descriptor.c
usb_device_descriptor.h
Quickstar... Global Va... Variables Breakpoi... Outline
MCUXpresso IDE - Quickstart Panel
Project: mbedtls_frdmk64f_freertos_ibm_watson_demo [Debug]
Create or import a project
New project...
Import SDK example(s)...
Welcome
ibm_watson_demo.c
watson_iot_config.h

```
 *
 * This software is owned or controlled by NXP and may only be used
 * strictly in accordance with the applicable license terms.  By expressly
 * accepting such terms or by downloading, installing, activating and/or
 * otherwise using the software, you are agreeing that you have read, and
 * that you agree to comply with and are bound by, such license terms.  If
 * you do not agree to be bound by the applicable license terms, then you
 * may not retain, install, activate or otherwise use the software.
 *
 * @par Description
 * IBM Watson IoT demo config file
 */

#ifndef WATSON_IOT_CONFIG
#define WATSON_IOT_CONFIG

/* Please update the value defined as WATSONIOT_ORG_ID ('kek1y7' in the example)
 * to match your organization ID
 * */

#define WATSONIOT_ORG_ID "352sje"  ///< Update this value with the six character code matching your Watson IoT Org ID

#define WATSONIOT_MQTT_BROKER_ENDPOINT \
    WATSONIOT_ORG_ID ".messaging.internetofthings.ibmcloud.com"

#define WATSONIOT_CLIENTID_LEN 31
#define WATSONIOT_CERT_INDEX 21

#define WATSONIOT_MQTT_BROKER_PORT 8883 ///< default port for MQTT/S
#define WatsonechoCLIENT_ID \
    "d:" WATSONIOT_ORG_ID ":NXP-A71CH-D:" ///< MQTT client ID should be unique for every device

#define WATSONIOT_A71CH_CLIENT_ID ((const uint8_t *)WatsonechoCLIENT_ID)

#define WATSON_IOT_KEYS_INDEX_SM 0 ///< Index where client key is kept

/*Subscribe for all the topics*/
#define WATSON_SUB_TOPIC "iot-2/cmd/+/fmt/+"

/*Publication topic is json*/
#define WATSON_PUB_TOPIC "iot-2/evt/status/fmt/json"

/*User Name to be used during MQTT connect*/
#define CUSTOM_MQTT_USER_NAME "use-token-auth"

static unsigned char tlsVERISIGN_ROOT_CERT_WATSON_PEM[] =
    "-----BEGIN CERTIFICATE-----\n"
```

Fig 26. Update the WATSONIOT_ORG_ID macro with your organization ID.

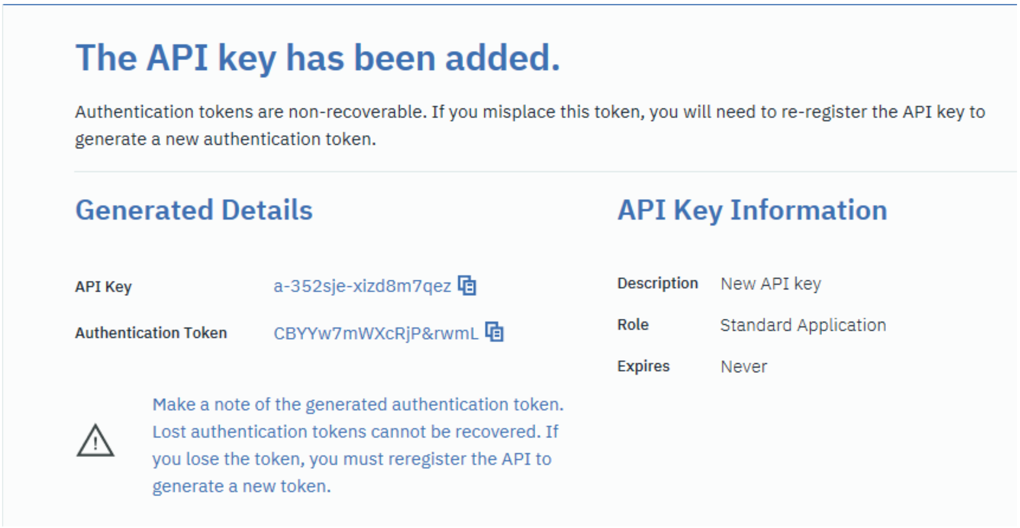
5.5 Run and configure the publish application

The publish application is a Python file that subscribes to IBM Watson IoT Platform. The messages received are used to change the LED color status on the FRDM-K64F board.

© NXP Semiconductors N.V. 2018. All rights reserved.

It is necessary to create an API key and an authentication token in IBM Watson IoT Platform. Go to the Dashboard and click on '*Apps'* menu and select '*Generate API Key'*. Click on '*Next'* and '*Generate Key*' and the API key will be automatically generated (you may add a description for it). In addition, go to the '*Role'* label and select '*Standard*'.

Once the key is generated, write down the API key and the token. Since tokens are non-recoverable, if you lose or forget it, you will need to re-register the API key to generate a new one.



The API key has been added.

Authentication tokens are non-recoverable. If you misplace this token, you will need to re-register the API key to generate a new authentication token.

Generated Details

API Key: a-352sje-xizd8m7qez

Authentication Token: CBYYw7mWXcRjP&rwmL

Make a note of the generated authentication token. Lost authentication tokens cannot be recovered. If you lose the token, you must reregister the API to generate a new token.

API Key Information

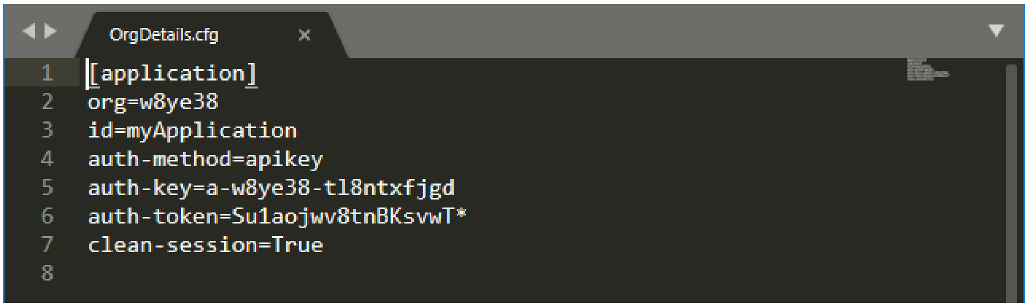
Description: New API key

Role: Standard Application

Expires: Never

Fig 27. API key creation in Watson IoT Platform.

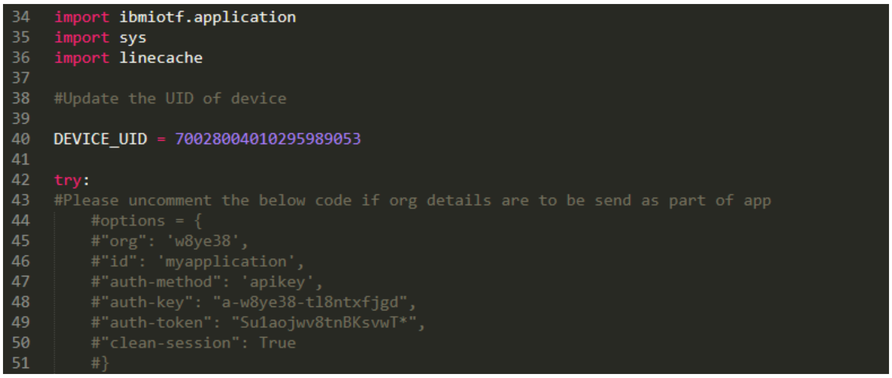
Now, configure the '*OrgDetails.cfg'* file in demo/ibm_watson_demo/PublishEventApp according to the ID of your organization and the API key you have just created.



```
[application]
org=w8ye38
id=myApplication
auth-method=apikey
auth-key=a-w8ye38-tl8ntxfjgd
auth-token=Su1aojwv8tnBKsvwT*
clean-session=True

```

Fig 28. Configuration file structure.

The '*test.py'* file must contain the certificate UID of the device. This file can also be found in the demo/ibm_watson_demo/PublishEventApp folder.

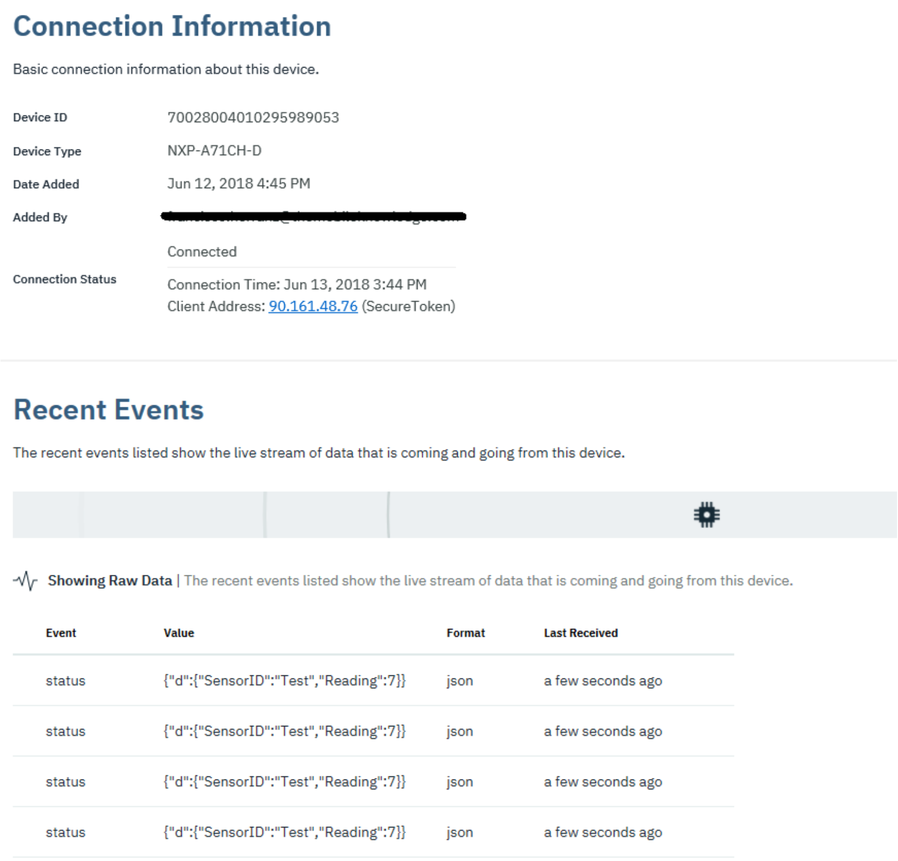


```
import ibmiotf.application
import sys
import linecache

#Update the UID of device

DEVICE_UID = 70028004010295989053

try:
#Please uncomment the below code if org details are to be send as part of app
    #options = {
    #"org": 'w8ye38',
    #"id": 'myapplication',
    #"auth-method": 'apikey',
    #"auth-key": "a-w8ye38-tl8ntxfjgd",
    #"auth-token": "Su1aojwv8tnBKsvwT*",
    #"clean-session": True
    #}
```

Fig 29. Publish application test.py file.

Run the python Publish Application using the following command in Windows Command Prompt. It will ensure that the Green LED is ON, though Red and Blue LEDs can also be turned ON or OFF.

```
python test.py OrgDetails.cfg GREEN ON
```



Connection Information

Basic connection information about this device.

Device ID 70028004010295989053

Device Type NXP-A71CH-D

Date Added Jun 12, 2018 4:45 PM

Added By [redacted]

Connection Status Connected

Connection Time: Jun 13, 2018 3:44 PM

Client Address: 90.161.48.76 (SecureToken)

Recent Events

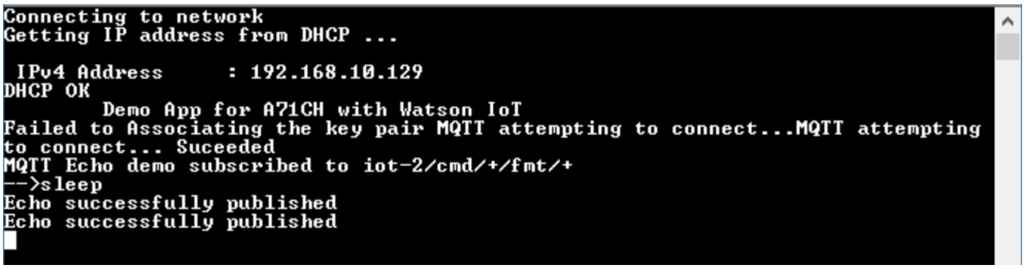
The recent events listed show the live stream of data that is coming and going from this device.

Showing Raw Data | The recent events listed show the live stream of data that is coming and going from this device.

Event	Value	Format	Last Received
status	{"d":{"SensorID":"Test","Reading":7}}	json	a few seconds ago
status	{"d":{"SensorID":"Test","Reading":7}}	json	a few seconds ago
status	{"d":{"SensorID":"Test","Reading":7}}	json	a few seconds ago
status	{"d":{"SensorID":"Test","Reading":7}}	json	a few seconds ago

Fig 30. Events of the IoT device in Watson IoT Platform.

In addition, to visualize the messages exchanged between the FRDM-K64F board and the Watson IoT Platform in your development PC, you can configure the Tera Term application.



```
Connecting to network
Getting IP address from DHCP ...

 IPv4 Address      : 192.168.10.129
DHCP OK
        Demo App for A71CH with Watson IoT
Failed to Associating the key pair MQTT attempting to connect...MQTT attempting
to connect... Suceeded
MQTT Echo demo subscribed to iot-2/cmd/+/fmt/+
-->sleep
Echo successfully published
Echo successfully published
```

Fig 31. Tera Term shows the messages of the connection.

6. A71CH Provisioned & Programmable type; Ready for IBM Watson IoT

This chapter explains how to integrate and onboard the A71CH Provisioned & Programmable type into your mass production environment or manufacturing flow of your IoT application.

6.1 A71CH Provisioned & Programmable overview

The A71CH Provisioned & Programmable type is a specialized A71CH product variant offering zero-touch connectivity to IBM Watson IoT Platform. It is intended for field deployment or operational use, enabling a fast and easy way to deploy secure connections to the IBM Watson IoT Platform. The A71CH Provisioned & Programmable type can be ordered via NXP or NXP’s distribution channel.

Table 2. A71CH Provisioned & Programmable ordering details

Type	Part number	12NC	Description	Package
A71CH Provisioned & Programmable	A7101CHTK2/T0BC2BJ	9353 737 63118	Security IC with standard temp range (-25 to +85 °C) Ready for IBM Watson IoT	HVSON8, Reel
A71CH Provisioned & Programmable	A7102CHTK2/T0BC2CJ	9353 741 46118	Security IC with extended temp range (-40 to +90 °C) Ready for IBM Watson IoT	HVSON8, Reel

With the A71CH Provisioned & Programmable type, your organization does not need to take care of the key injection, owning PKI infrastructure or subcontracting a third-party PKI infrastructure for the IC trust provisioning. The A71CH Provisioned & Programmable type is delivered already provisioned by NXP Trust Provisioning with the following credentials:

- One ECC key pair
- One X.509 certificate for an IoT device.
- One X.509 certificate for an IoT gateway.

The Root CA and Intermediate CA also need to be registered in your Watson IoT Platform organization. The CA certificates used for the NXP Trust provisioning can be obtained from:

- NXP Root CA certificate: https://www.gp-ca.nxp.com/CA/getCA?caid=63709315030001
- NXP Intermediate CA certificate: https://www.gp-ca.nxp.com/CA/getCA?caid=63709315040001

Note: These CA certificates are downloaded in DER format. It is necessary to convert them to PEM format before uploading them to Watson IoT Platform.

The credentials injected in the A71CH Provisioned & Programmable during the NXP Trust provisioning together with the CA certificates support the establishment of a trusted TLS connection to IBM Watson IoT Platform.

Note: In case your IoT product needs any additional data or credentials on top of what is provisioned by default in the A71CH Provisioned & Programmable, you can refer to the NXP Trust provisioning offering. You may also refer to the alternative Trust Provisioning options via NXP distributors and third-party partners in programming centers. For more information, please check the ***AN12227-A71CH Trust Provisioning*** document.

Section 3.2 details the process of uploading these CA certificates to Watson IoT Platform. It is important to notice that the OEM only needs to upload these certificates once, when setting up its organization in Watson IoT Platform.

6.2 A71CH Provisioned & Programmable integration and onboarding

The A71CH Provisioned & Programmable integration into your IoT application manufacturing flow requires these three steps:

4. **A71CH Provisioned & Programmable transport seal verification:** The transport seal verification is used as a proof that the ICs were not manipulated or tampered during transport and delivery.
5. **A71CH Provisioned & Programmable closing script**: The closing script is used to make sure that it is no longer possible to add or remove credentials or data into the ICs. Running this closing script is required to prevent accidental or malicious modifications.
6. **Read device or gateway certificate UID**: The Watson IoT Platform defines a *Device ID* field as part of the device or gateway registration process in the platform. The device or gateway certificate UID is used as this *Device ID* field. This certificate UID is die-individual; it is a 10-byte subset generated from the A71CH 18-byte UID.

To execute these three steps, it is required that your organization has IC programming capabilities in the manufacturing process.

6.2.1 A71CH Provisioned & Programmable transport seal verification

The A71CH Provisioned & Programmable leaves the NXP facilities with the transport seal activated. The transport seal feature is implemented using the A71CH transport lock mechanism based on a 16-byte symmetric secret.

For illustrative purposes, the command below demonstrates how this verification can be performed using the A71CH Arduino compatible development kit, an i.MX6UltraLite board and the A71CH Configure tool:

```
./a71chConfig_i2c_imx transport unlock -h
6cfb2fc7edc5133a2bfc90cbd5f9a0d5
```

Note: As the transport lock is a seal, the key value is not considered confidential.

6.2.2 A71CH Provisioned & Programmable closing script

As part of the NXP trust provisioning, the A71CH Provisioned & Programmable key pair index #0 is locked, as well as the general-purpose index #0 and #1 where the device and gateway X.509 certificates are provisioned.

The closing script is intended to prevent accidental or malicious modifications of those credentials and to prevent adding or removing other data or certificates. This closing script needs to address the following actions:

- Inject a random value into both the Key Pair and Public Key config key
- Inject a set of random values as SCP03 keys (as a side effect SCP03 can no longer be enabled, unless the attacker knows the random values)
- Lock key pair slot 1 to 3
- Lock public key slots 0 to 2
- Inject random values into all symmetric key slots
- Lock lookup table of GP storage

For illustrative purposes, the script below is a bash shell script example used to close the A71CH Provisioned & Programmable.

```
# ########################################################
# Pre Deployment bash script to lock down A71CH-WATSON-IOT type
# Revision 0.8
# Purpose: Prepare sample for Operational Phase
# Pre-condition:  device comes from NXP production line
# Post-condition: locked down device ready for deployment
# #########################################################

# Customization Variables
# -----------------------
sealKey="6cfb2fc7edc5133a2bfc90cbd5f9a0d5"
scp03KeyFile="scp03Confidential.txt"

seNickname="A71CH-WATSON-IOT"

# Tools
# -----
A71CH_CONFIG_TOOL="./a71chConfig_i2c_imx"

# Set global variables to default values
gExecMsg=""
```

```
gExecFailMsg=""
gRetValue=0

# Utility functions
# -----------------
# xCmd will stop script execution when the program executed does return gRetValue (0 by
default) to the shell
xCmd () {
    local command="$*"
    if [ "${gExecMsg}" != "" ]; then
        echo ">> ${gExecMsg}"
    fi
    echo ">> ${command}"
    ${command}
    local nRetProc="$?"
    if [ ${nRetProc} -ne ${gRetValue} ]; then
        echo "\"${command}\" failed to run successfully, returned ${nRetProc}"
        if [ "${gExecFailMsg}" != "" ]; then
            echo ">> ${gExecFailMsg}"
        fi
        echo "** Script execution failed **"
        exit 2
    fi
    echo ""
    # Set global variables to default values
    gExecMsg=""
    gExecFailMsg=""
    gRetValue=0
}

random_16() {
    local rnd16=$(openssl rand -hex 16 | tr -d '\r')
    echo -n ${rnd16}
}
# First argument to script can overrule default value of Transport Unlock key
#   In case first argument equals NO, unlock is skipped
#   Otherwise argument is taken as unlock key

echo "Script to prepare ${seNickname} for deployment in the field"
doUnlock="yes"
if [ "$#" -eq 1 ]; then
    if [ $1 = "NO" ]; then
        echo "Skip unlock step"
        doUnlock="no"
    else
        # Check that argument (Unlock Key) is 32 ASCII characters long
        sealKey="$1"
        if [ "${#sealKey}" -ne 32 ]; then
            echo "  The transport key (provided as argument) must be exactly 32
characters long"
            echo "The current argument is ${#sealKey} characters long"
            echo "Exiting..."
            exit 4
        fi
    fi
elif [ "$#" -lt 1 ]; then
    echo "Taking ${sealKey} as unlock key"
fi
#
# Validation of incoming device
#
if [ "${doUnlock}" = "yes" ]; then
    gExecFailMsg="Cannot unseal ${seNickname}"
    xCmd "${A71CH_CONFIG_TOOL} transport unlock -h ${sealKey}"
fi
# Check on debug reset
# Not applicable during development

#
# Provisioning device
#
echo "Assumption: No additional provisioning is required."
echo "No need to inject additional key pairs, public keys, symmetric keys or plain data
in GP storage"

echo "Fill Key Pair & Public Key config keys with random data"

rndValCfgPrivate=$(random_16)
gExecFailMsg="Cannot inject random value for Key Pair Config key"
xCmd "${A71CH_CONFIG_TOOL} set cfg -x 1 -h ${rndValCfgPrivate}"

rndValCfgPublic=$(random_16)
gExecFailMsg="Cannot inject random value for Public Key Config key"
xCmd "${A71CH_CONFIG_TOOL} set cfg -x 2 -h ${rndValCfgPublic}"

echo "Disable usage of SCP03 through the injection of Random Base Keys"
scp03Enc=$(random_16)
scp03Mac=$(random_16)
scp03Dek=$(random_16)
echo "# If an attacker gets hold of this file, he can mount a denial of service attack"
> ${scp03KeyFile}
echo "ENC ${scp03Enc}" >> ${scp03KeyFile}
echo "MAC ${scp03Mac}" >> ${scp03KeyFile}
echo "DEK ${scp03Dek}" >> ${scp03KeyFile}
gExecFailMsg="Cannot inject random value's as SCP03 key set"
xCmd "${A71CH_CONFIG_TOOL} scp put -h 01 -k ${scp03KeyFile}"

rm ${scp03KeyFile}

#
# Closing device
#

echo "Lock key pair slots"
idx=1 # First keypair is already locked
while [ $idx -lt 4 ]; do
    gExecFailMsg="Cannot lock key pair slot ${idx}"
    xCmd "${A71CH_CONFIG_TOOL} lock pair -x ${idx}"
    let idx=idx+1
done

echo "Lock public key slots"
idx=0
while [ $idx -lt 3 ]; do
    gExecFailMsg="Cannot lock public key slot ${idx}"
    xCmd "${A71CH_CONFIG_TOOL} lock pub -x ${idx}"
    let idx=idx+1
done

echo "Write random values into symmetric key store"
idx=0
while [ $idx -lt 8 ]; do
    symRndVal=$(random_16)
    gExecFailMsg="Cannot write random symmetric data into slot ${idx}"
    xCmd "${A71CH_CONFIG_TOOL} set sym -x ${idx} -h ${symRndVal}"
    let idx=idx+1
```

```
done

echo "Lock lookup table"
# NOTE: The size of the lookup table depends on the number of
# objects stored.
gExecFailMsg="Cannot lock lookup table"
xCmd "${A71CH_CONFIG_TOOL} lock gp -h 0FE0 -n 1"


echo "Successfully completed predeployment
```

Fig 32. Bash shell script to close A71CH

A similar script needs to be replicated during your IoT application manufacturing process.

Note: Please, refer to the ***A71CH Security Recommendations*** document [SECURITY_GUIDELINES] to comply with the security guidelines for this closing script.

6.2.3 Read device or gateway certificate UID:

The device or gateway certificate UID (according to the type of device to be registered in the IBM Watson IoT Platform) for each A71CH Provisioned & Programmable sample needs to be read. This certificate UID is used and required for the device or gateway registration in your IBM Watson IoT Platform account.

The A71CH Host software package (starting from Rev 1.5.0) provides the dedicated `A71_GetCertUid` API Call to retrieve the certificate UID.

```
U16 A71_GetCertUid (U8 *certUid, U16 *certUidLen)
```

The same command can be used in the A71CH Configure tool to retrieve the certificate UID.

As established in Section 3.4, using Bulk Configuration APIs, several devices can be registered at once in Watson IoT Platform. Similarly, a customer can create a script that automatically reads out the UIDs of its IoT devices or gateways. This process depends of the customer and can be optimized according to its programming capabilities.

7. FAQs

Is there a development board available for A71CH Provisioned & Programable; Ready for IBM Watson IoT type?

No. Development boards are only available for A71CH Customer Programable type.

Can the A71CH Provisioned & Programable; Ready for IBM Watson IoT type be used for prototyping?

Yes, but customer shall solder it into a board. Additionally, it is recommended to use the A71CH Customer Programable because of the possibility to use Debug Reset.

These and more questions can be found in the online Community for Secure Authentication [COMMUNITY].

8. References

Table 3. Referenced Documents

[QUICK_START_IMX6]	**AN12119 Quick start guide for OM3710A71CHARD i.MX6** – Application note, document number 4582**[1]
[IBM_CLOUD]	**IBM Cloud Website** - https://console.bluemix.net/
[QUICKSTART_IBM]	**Quickstart IBM -** https://quickstart.internetofthings.ibmcloud.com
[QUICKSTART_KIN]	**AN12135 – A71CH Quick start guide for OM3710A71CHARD and Kinetis**– Application note, document number 4582**[1]
[IMX6UL_REPO]	**GitHub repository -** https://github.com/ibm-watson-iot/iot-nxpimxa71ch-c
[SECURITY_GUIDELINES]	**AN12211 A71CH Security Guidelines –** Application note, document number 4781**[1]
[BULK_OPS]	**IBM Watson IoT Platform Organization Administration REST APIs -** https://docs.internetofthings.ibmcloud.com/apis/swagger/v0002/org-admin.html?cm_mc_uid=71367544061615028292336&cm_mc_sid_50200000=96466801537194639901
[COMMUNITY]	**Secure authentication online Community -** https://community.nxp.com/community/identification-security/secure-authentication
NXP Root CA certificate	https://www.gp-ca.nxp.com/CA/getCA?caid=63709315030001
NXP Intermediate CA certificate	https://www.gp-ca.nxp.com/CA/getCA?caid=63709315040001

[1] **… document version number

9. Legal information

9.1 Definitions

Draft — The document is a draft version only. The content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

9.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the *Terms and conditions of commercial sale* of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Translations — A non-English (translated) version of a document is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Evaluation products — This product is provided on an "as is" and "with all faults" basis for evaluation purposes only. NXP Semiconductors, its affiliates and their suppliers expressly disclaim all warranties, whether express, implied or statutory, including but not limited to the implied warranties of non-infringement, merchantability and fitness for a particular purpose. The entire risk as to the quality, or arising out of the use or performance, of this product remains with customer.

In no event shall NXP Semiconductors, its affiliates or their suppliers be liable to customer for any special, indirect, consequential, punitive or incidental damages (including without limitation damages for loss of business, business interruption, loss of use, loss of data or information, and the like) arising out the use of or inability to use the product, whether or not based on tort (including negligence), strict liability, breach of contract, breach of warranty or any other theory, even if advised of the possibility of such damages.

Notwithstanding any damages that customer might incur for any reason whatsoever (including without limitation, all damages referenced above and all direct or general damages), the entire liability of NXP Semiconductors, its affiliates and their suppliers and customer's exclusive remedy for all of the foregoing shall be limited to actual damages incurred by customer based on reasonable reliance up to the greater of the amount actually paid by customer for the product or five dollars (US$5.00). The foregoing limitations, exclusions and disclaimers shall apply to the maximum extent permitted by applicable law, even if any remedy fails of its essential purpose.

9.1 Licenses

ICs with DPA Countermeasures functionality

LICENSED DPA COUNTERMEASURES™

NXP ICs containing functionality implementing countermeasures to Differential Power Analysis and Simple Power Analysis are produced and sold under applicable license from Cryptography Research, Inc.

9.2 Trademarks

Notice: All referenced brands, product names, service names and trademarks are property of their respective owners.

FabKey — is a trademark of NXP B.V.

I²C-bus — logo is a trademark of NXP B.V.

10. List of figures

11. List of tables

12. Contents

Please be aware that important notices concerning this document and the product(s) described herein, have been included in the section 'Legal information'.

© NXP Semiconductors N.V. 2018. All rights reserved.

For more information, visit: http://www.nxp.com

Date of release: 26 September 2018
464110

Document identifier: 464110