**FT900 IoT Library**

**Design Document**

**FT900 IoT Architecture:**

**1. IoT connectivity**

A. MQTT library of Amazon FreeRTOS – OK

<https://github.com/richmondu/FT900/tree/master/IoT/aws_demos_ft_greengrass_lwip_mbedtls>

B. MQTT library of LWIP – OK (To be used for all IoT examples)

<https://github.com/richmondu/FT900/tree/master/IoT/ft90x_iot_aws_gcp_azure>

**2. IoT security**

A. mbedTLS integration - OK

B. ciphersuite-memory tradeoff options - OK

C. TLS/SSL X509 certificate authentication - OK

D. JWT/SAS security token authentication – OK

E. Certificate compilation – OK (contributed by Gordon McNab)

**3. IoT cloud connectivity**

A. amazon aws iot – OK (X509 Certificate authentication)

B. google gcp iot - OK (JWT Security Token authentication)

C. microsoft azure iot - OK (X509 Certificate authentication and SAS Security Token authentication)

D. amazon greengrass (iot edge) – OK

E. google gcp iot edge – TODO

F. microsoft azure iot edge - TODO

**4. IoT cloud examples**

A. amazon aws – OK (backend: OK, frontend: OK)

<https://github.com/richmondu/FT900/tree/master/IoT/aws_demos_ft_greengrass_lwip_mbedtls>

<https://github.com/richmondu/FT900/tree/master/IoT/ft90x_iot_aws_gcp_azure/cloud/aws_iot>

B. google gcp - ONGOING (backend: OK, frontend: webpage integration by Unna)

<https://github.com/richmondu/FT900/tree/master/IoT/ft90x_iot_aws_gcp_azure/cloud/google_iot>

C. microsoft azure - ONGOING (backend: OK, frontend: webpage integration by Unna)

<https://github.com/richmondu/FT900/tree/master/IoT/ft90x_iot_aws_gcp_azure/cloud/azure_iot>

**5. IoT library**

<https://github.com/richmondu/FT900/tree/master/IoT/ft90x_iot_aws_gcp_azure/lib/iot>

A. IoT api – OK

B. IoT utilities – OK

C. IoT configuration - OK

**6. IoT toolchain**

A. eclipse IoT project templates (AWS IoT project, Google IoT project, Azure IoT project) – OK

B. eclipse IoT Configuration view (Similar to FreeRTOS Configuration view) - OK

**7. Others:**

A. memory footprint (program and data) – TODO

B. certificates in DER binary format – TODO

C. IoT sensor examples – TODO

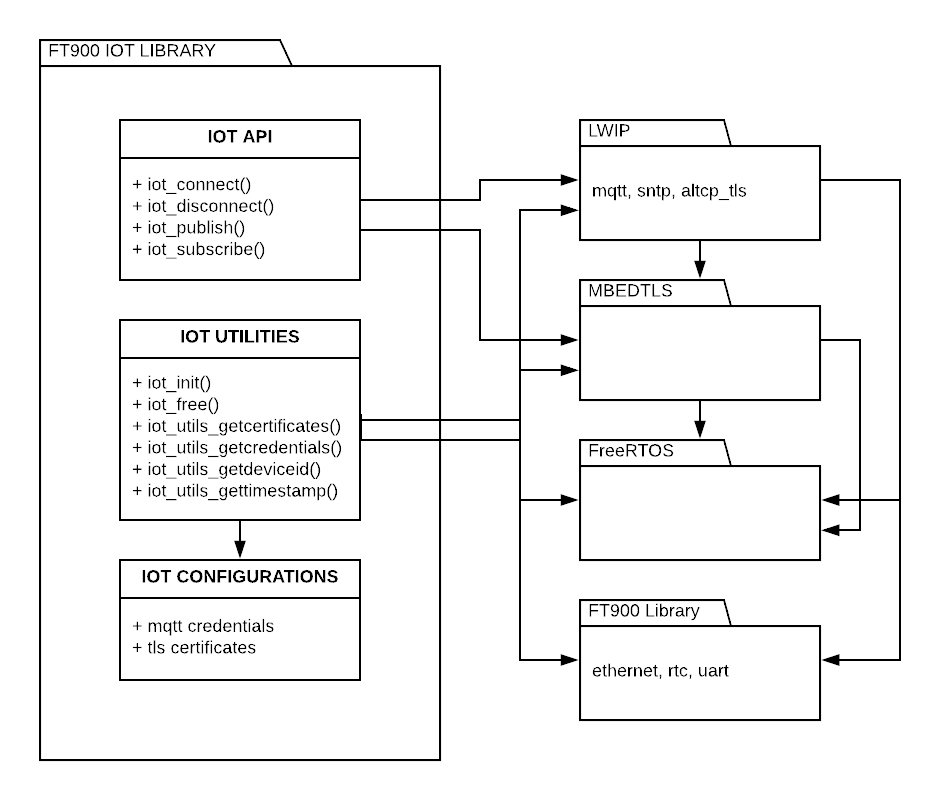
D. FT900 RevC debugging/support – TODO (Tested working with FT900 RevA and RevB only.)

E. DHCP related issues - TODO

**FT900 IoT Library**

The IoT Library simplifies IoT development by abstracting MQTT protocol, together with secure authentication (TLS certificates or JWT/SAS security tokens), time management (SNTP and RTC) and IoT connectivity with the leading cloud platforms Amazon AWS, Google Cloud and Microsoft Azure.

**UML Class Diagram**

****

The FT900 IoT library depends on:

Third-party libraries (mbedTLS, LWIP, FreeRTOS and tinyprintf) and

FT900 libraries (Ethernet, RTC and UART).

**FT900 IoT Library Components**

The IoT Library is composed of API interface, utilities and configuration files.

**IoT API**

The IoT API is a simple lightweight abstraction of LWIP’s MQTT protocol and TLS certificates authentication.

It is designed so that users have complete access of the MQTT credentials and TLS certificates to be used for the connection.

It is designed so that it is possible to modify LWIP’s MQTT protocol with a different MQTT library.

It is designed with considerations of performance and memory footprint.

**IoT Utilities**

The IoT Utilities simplifies connectivity with Amazon AWS, Google Cloud and Microsoft Azure. These cloud providers have different requirements for authentication and different formats for MQTT credentials. The utilities derives/generates the necessary formatted values using raw information form IoT configuration files.

It is designed to be independent of IoT API. So users may or may not use it.

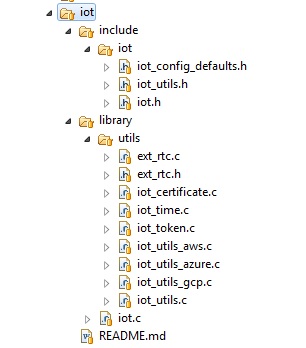
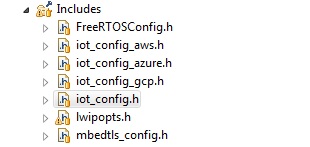
It is designed modular (different files for Amazon AWS, Google Cloud and Microsoft Azure).

**IoT Configuration**

The IoT Utilities retrieves the raw information from the IoT Configuraiton files.

Sample configuration files are provided for Amazon AWS, Google Cloud and Microsoft Azure.

Below is the folder structure of the components:

****

**FT900 IoT API**

The IoT API simplifies IoT development by abstracting MQTT protocol, together with secure authentication (TLS certificates or JWT/SAS security tokens).

**iot\_connect( iot\_certificates\_cb, iot\_credentials\_cb )**

Accepts callback functions for setting the TLS certificates and MQTT credentials.

**iot\_certificates\_cb** (iot\_certificates\* tls\_certificates)

**iot\_credentials\_cb** (iot\_credentials\* mqtt\_credentials)

The callback functions will be called by this function when it is needed. This strategy is more efficient in terms of memory than sending the structure of TLS certificates and MQTT credentials. The function will release the memory for TLS certificates immediately once it has configured mbedTLS.

Sample callback functions are provided for users. This functions generates/derives the necessary information from iot\_config.h and the certificates stored in the Certificates folder.

**iot\_utils\_getcertificates** (iot\_certificates\* tls\_certificates)

**iot\_utils\_getcredentials** (iot\_credentials\* mqtt\_credentials)

Users can opt to use these utility functions to their applications.

Returns a handle to be used for the other functions below.

**iot\_disconnect( iot\_handle )**

Disconnect from server and cleanup dynamic memory and resources used.

Accepts a handle retrieved from iot\_connect.

**iot\_publish( iot\_handle, topic, payload, payload\_len )**

Send/publish (sensor) data to a specified publish topic.

Accepts a handle retrieved from iot\_connect together with the topic and payload to publish.

**iot\_subscribe( iot\_handle, topic, subscribe\_cb )**

Registers a callback function to receive packets from a specified subscription topic

Accepts a handle retrieved from iot\_connect together with the topic and callback functions that will be called when messages are received for the specified topic.

**iot\_certificates**

Structure of TLS certificates (ca, cert, pkey, ca\_len, cert\_len, pkey\_len)

**iot\_credentials**

Structure of MQTT credentials (host, port, client\_id, client\_user, client\_pass)

**FT900 IoT utilities**

The IoT utilities simplifies IoT development by abstracting the derivation of MQTT credentials as well as the retrieval of TLS certificates necessary for secure IoT connectivity with popular cloud platforms: Amazon AWS, Google Cloud and Microsoft Azure.

**iot\_utils\_init()**

To be called before calling the below utility functions

**iot\_utils\_free()**

Release memory called by iot\_utils\_init()

**iot\_utils\_getcertificates()**

Retrieves the TLS certificates (ca, client certificate, client private key) as needed by Amazon AWS, Google Cloud and Microsoft Azure. These 3 cloud platforms differ in the TLS certificate authentication requirements. This function abstracts this from users.

Note that this requires update of **iot\_config.h** and update of certificates inside the **Certificates folder**.

**iot\_utils\_getcredentials()**

Retrieves the MQTT credentials (broker, port, client\_id, client\_user, client\_pass) as needed by Amazon AWS, Google Cloud and Microsoft Azure. These 3 cloud platforms differ in the format and details of MQTT credentials authentication requirements. This function abstracts this from users.

Note that this requires update of **iot\_config.h**.

**iot\_utils\_getdeviceid()**

Retrieves the deviceid from iot\_config.h. This can be used for publish/subscribe.

**iot\_utils\_gettimestampepoch()**

Retrieves the current time from epoch in decimal format. This can be used for adding timestamps to the published packets.

**iot\_utils\_gettimestampiso()**

Retrieves the current time from epoch in string format. This can be used for adding timestamps to the published packets.

**FT900 IoT API usage**

Pseudocode:

iot\_task()

{

net\_init() **// Initialize network**

iot\_utils\_init() //

while (1) {

While (!net\_is\_ready()) **// Wait until network is ready**

net\_get\_ip(), net\_get\_gateway(), net\_get\_netmask **// Display network information**

**// Connect to broker, callback functions will be called for user to set the necessary info**

iot\_handle = iot\_connect ( iot\_utils\_getcertificates\_cb, iot\_utils\_getcredentials\_cb )

**// Register subscription topic and callback function**

iot\_subscribe ( iot\_handle, sub\_topic, subscription\_cb )

**// Send/publish (sensor) data to a specified topic**

while (1) {

pub\_topic = generate\_topic()

pub\_payload = generate\_payload( &publish\_payload\_len )

ret = iot\_publish ( iot\_handle, pub\_topic, pub\_payload, pub\_payload\_len )

}

iot\_disconnect ( iot\_handle )

}

iot\_utils\_free()

}

**IoT TLS Certificates**

**Certificates** folder

Certificates.mk // makefile to compile the certificates and generate .S files

Sample for **Amazon AWS IoT**

Rootca.pem

Ft900device1\_cert.pem

Ft900device1\_pkey.pem

Sample for **Amazon AWS Greengrass**

Rootca\_gg.pem // dynamically retrieved; can also be retrieved using AWS Python SDK

Ft900device1\_cert.pem

Ft900device1\_pkey.pem

Sample for **Google Cloud IoT**

Ft900device1\_cert.pem // not used by device, registered in cloud only

Ft900device1\_pkey.pem // used to generate the JWT security token

Sample for **Microsoft Azure IoT** (using SAS security token authentication)

Rootca\_azure.pem // converted .der file from <https://github.com/Azure/azure-iot-sdk-c/blob/master/certs/ms.der> using openssl

Ft900device1\_sas\_azure.pem // used to generate SAS security token

Sample for **Microsoft Azure IoT** (using TLS certificate authentication)

Rootca\_azure.pem // converted .der file from <https://github.com/Azure/azure-iot-sdk-c/blob/master/certs/ms.der> using openssl

Ft900device1\_cert.pem

Ft900device1\_pkey.pem

**IoT MQTT Credentials**

The function **iot\_utils\_getcredentials()** in **iot\_utils.c** retrieves/derives this information from **iot\_config.h**.

Default:

MQTT\_BROKER

MQTT\_BROKER\_PORT = 8883

MQTT\_CLIENT\_NAME

MQTT\_CLIENT\_USER

MQTT\_CLIENT\_PASS

Amazon AWS IoT

MQTT\_BROKER = “IDENTIFIER.iot.REGION.amazonaws.com”

MQTT\_BROKER\_PORT = 8883

MQTT\_CLIENT\_NAME = DEVICE\_ID or THING\_NAME

MQTT\_CLIENT\_USER = NULL // not needed

MQTT\_CLIENT\_PASS = NULL // not needed

Amazon AWS Greengrass

MQTT\_BROKER = IP address or host name of local Greengrass device

MQTT\_BROKER\_PORT = 8883

MQTT\_CLIENT\_NAME = DEVICE\_ID or THING\_NAME

MQTT\_CLIENT\_USER = NULL // not needed

MQTT\_CLIENT\_PASS = NULL // not needed

Google Cloud IoT

MQTT\_BROKER = “mqtt.googleapis.com”

MQTT\_BROKER\_PORT = 8883

MQTT\_CLIENT\_NAME = “projects/PROJECT\_ID/locations/LOCATION\_ID/registries/REGISTRY\_ID/devices/DEVICE\_ID”

MQTT\_CLIENT\_USER = “ “ // any

MQTT\_CLIENT\_PASS = JWT security token (generated with private key)

Microsoft Azure IoT (SAS security token authentication)

MQTT\_BROKER = “HUB\_NAME.azure-devices.net”

MQTT\_BROKER\_PORT = 8883

MQTT\_CLIENT\_NAME = DEVICE\_ID

MQTT\_CLIENT\_USER =

“HUB\_NAME.azure-devices.net/DEVICE\_ID/api-version=2016-11-14”

MQTT\_CLIENT\_PASS = SAS security token (generated with shared access key)

Microsoft Azure IoT (TLS certificate authentication)

MQTT\_BROKER = “HUB\_NAME.azure-devices.net”

MQTT\_BROKER\_PORT = 8883

MQTT\_CLIENT\_NAME = DEVICE\_ID

MQTT\_CLIENT\_USER =

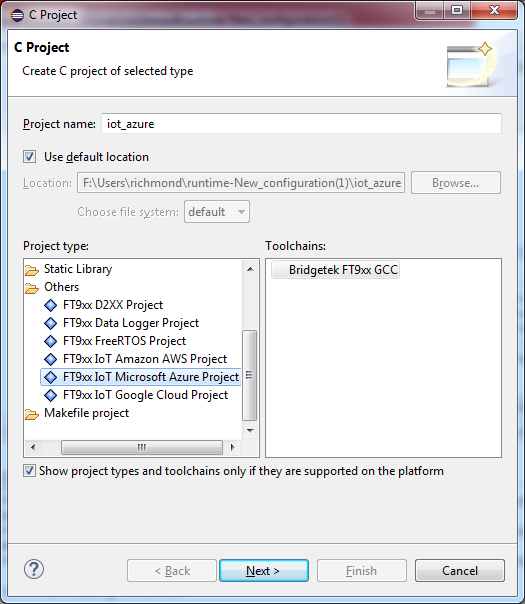
“HUB\_NAME.azure-devices.net/DEVICE\_ID/api-version=2016-11-14”

MQTT\_CLIENT\_PASS = NULL // not needed

**FT900 IoT Project Templates for FT900 Eclipse Toolchain 2.6.0 (IOT Edition)**

**IoT Made Easy**

Users can now create IoT applications via the IoT Project Template feature in FT900 Eclipse IDE Toolchain.



**Procedures:**

1. Add project template:
   1. FILE > NEW > C PROJECT
   2. Type project name and select IoT AWS, IoT Azure or IoT GCP.
2. (Optional) Update MQTT credentials and TLS certificates.
   1. Update MQTT credentials in iot\_config.h
   2. Update TLS certificates in Certificates folder.
3. Compile and run.

**Notes:**

Currently, for Alpha Release 1, I am using my certificates and cloud account. This is temporary for validation purposes. In public toolchain release, the following files must be modified:

Files to be modified

FT9xx Toolchain\Toolchain\eclipse\templates\templates\_iot

\Certificates\ft900device1\_cert.pem

\Certificates\ft900device1\_pkey.pem

\Certificates\ft900device1\_sas\_azure.pem

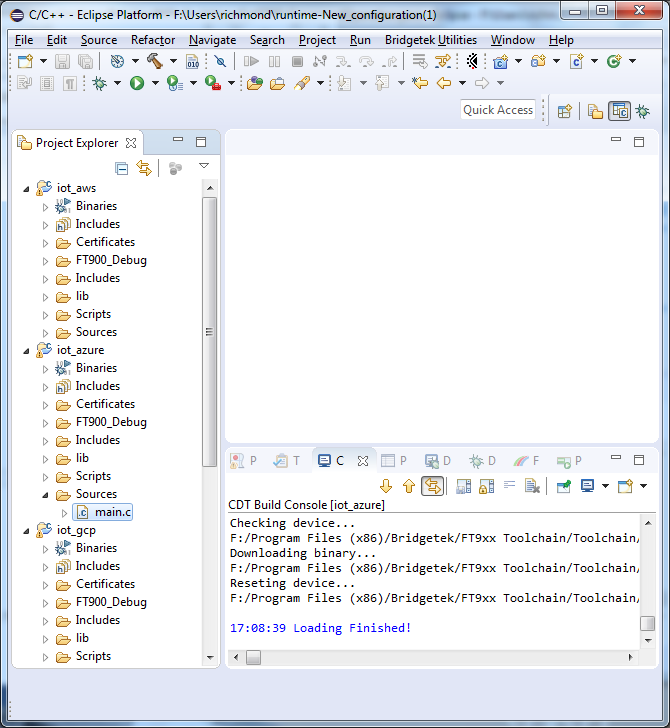
\Certificates\rootca.pem

\Certificates\rootca\_gg.pem

\Includes\iot\_config\_aws.h

\Includes\iot\_config\_azure.h

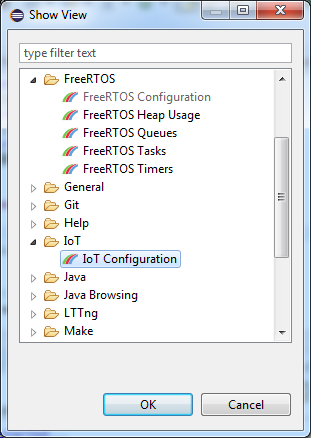
\Includes\iot\_config\_gcp.h



**FT900 IoT Configuration View for FT900 Eclipse Toolchain 2.6.0 (IOT Edition)**

**IoT Configuration view:**

The IoT Configuration view provides a UI to modify the IoT configurations.



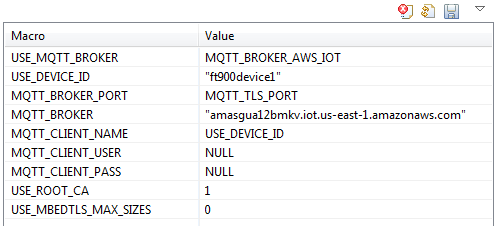
Below are snapshot of the IoT Configuration views for AWS, GCP and Azure.

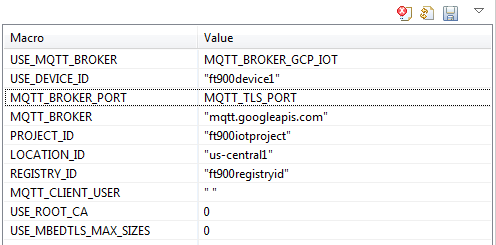
The IoT Configuration view behaves similar to FreeRTOS Configuration view. It provides the following buttons:

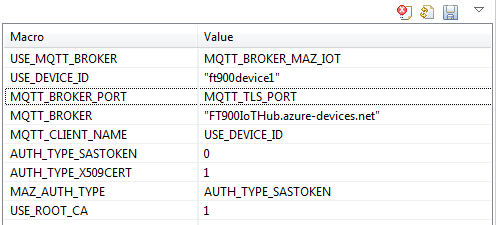
Clear button – clear the contents of the view.

Refresh button – refresh the contents of the view.

Save button – overwrite iot\_config.h with the new values. Previous file will be renamed with current date and time.

****

****

****

**FT900 IoT Development Environment:**

1. Windows 7 X64 laptop, Windows 7 X64 desktop
2. Python 2 and Python 3
3. FT900 RevA, FT900 RevB
4. Router X network (fast), Router Y network (slow)
5. Amazon AWS Console <https://console.aws.amazon.com/console>
6. Google Cloud Console <https://console.cloud.google.com>
7. Microsoft Azure Portal <https://portal.azure.com>

**FT900 IoT Cloud Setup Guides:**

1. <https://github.com/richmondu/FT900/tree/master/IoT/aws_demos_ft_greengrass_lwip_mbedtls>
2. <https://github.com/richmondu/FT900/blob/master/IoT/ft90x_iot_aws_gcp_azure/cloud/google_iot/FT900IoTDemo_SetupGuide_GoogleCloud.docx>
3. <https://github.com/richmondu/FT900/blob/master/IoT/ft90x_iot_aws_gcp_azure/cloud/azure_iot/FT900IoTDemo_SetupGuide_MicrosoftAzure.docx>

**FT900 IoT Known Issues and Limitations:**

1. Fix certificate compilation issue.

First compile fails because compiled certificates cannot be found. (Gordon is aware of this issue in certificates.mk) Recompiling again will succeed.

[FIXED in Alpha Release 2]

1. Support RevC.

FT900 RevC does not work according to Unna. (I dont have a RevC).

1. Investigate stress test issue.

MQTT Publish can fail with -1 or -11. When it fails -1, we retry 5 times. If -11, we don’t. I found that when it -11 is encountered, net\_is\_ready returns 0 because the link layer returns 0. So for some reason, the network link layer fails after some time so it cannot recover. As a workaround, I added chip\_reboot() in main.c so that overnight stress test passes.

1. Fix DHCP-related issues
2. Improve memory footprint.

a. Use memory footprint optimizations from previous application

b. Support certificates in .der binary format.