Chapter 7D: Cloud Connectivity using MQTT

At this end of Chapter 7D you will understand in **DETAIL** how to write WICED firmware to interact with the AWS IoT Cloud using MQTT.

7D.1 Using MQTT with AWS in WICED

The WICED SDK contains a library of functions that make it easier to create MQTT firmware. These functions are contained in libraries/protocols/MQTT. To include the library in a project, two things are needed:

In the Make File: *$(NAME)_COMPONENTS := protocols/MQTT*

In the C Source: *#include "mqtt_api.h"*

In addition to the library, there are several demo applications that can be used as a starting point for using MQTT with AWS. These applications are: apps/demo/aws_iot:

- *apps/demo/aws_iot/pub_sub/publisher*
  - Publish a message when a button is pressed on the kit.
- *apps/demo/aws_iot/pub_sub/subscriber*
  - Subscribe to messages that control an LED on the kit.
- *apps/demo/aws_iot/shadow*
  - Interact with a *thing* via the shadow. This project also demonstrates using a configuration AP and a web server to set up the Wi-Fi configuration and security certificates.
- *apps/demo/aws_iot/temperature_controlled_device/device*
  - Subscribe to messages from the remote sensor and turn an LED ON/OFF depending on the messages received. This project also uses a configuration AP and web server to set up the Wi-Fi configuration and security certificates.
- *apps/demo/aws_iot/temperature_controlled_device/remote_sensor*
  - Measure room temperature from a thermistor and get the outside temperature from openweathermap.org. Depending on the temperature difference, publish messages to the temperature controlled device. This project also uses a configuration AP and web server to set up the Wi-Fi configuration and security certificates.

The security certificates and keys for each of these projects are read from resources/apps/aws_iot. The keys are included in the project using the following:

In the Make File:
```
$(NAME)_RESOURCES  := apps/aws_iot/rootca.cer \
apps/aws_iot/client.cer \
apps/aws_iot/privkey.cer
```

In the C Source: *#include "resources.h"*

In the exercises below, we will use the publisher, subscriber, and shadow projects as starting points to learn the details of using WICED to interact with AWS using MQTT.

7D.2 Exercise(s)

Exercise - 7D.1 Test the demo.aws_iot.pub_sub.publisher App

1. Copy the WICED apps/demo/aws_iot/pub_sub/publisher project to your own directory (i.e. ww101/07b/02_publisher) and update the files.
    a. Hint: Make sure you add your platform to the valid platforms in the makefile.
2. Modify the DCT for your network.
3. If you are not using the region US East time zone 1, make sure you search for us-east-1 in the code and modify as appropriate.
4. Copy the certificates that you generated in the first exercise into the resources/apps/aws_iot directory. Replace two of the existing files in that directory as follows:

Name of Downloaded File	New Name	Description
<name>-certificate.pem.crt	client.cer	The certificate for your thing. This is how AWS knows that it is a valid *thing* that is trying to talk to it.
<name>-private.pem.key	privkey.cer	The private key that your application will use to decrypt data that it gets back from AWS. Since Amazon created the key, it already has the public key.

The rootca.cer file in that folder is the certificate for Amazon. This allows your thing to know that it is really talking to the AWS Cloud. This is a known-good key for AWS that is built into the SDK. It does not need to be modified since it never changes (at least not until it expires in on July 16, 2036).

The other file that you downloaded called "<name>-public.pem.key" is a public key for your thing. In this case, Amazon already has the public key so you don't need to provide it.

5. Run a "clean" before rebuilding or else your project may not see the new keys. You will find clean at the top of the list of Make Targets. Just double-click on it to run it.
6. Create a Make Target for your project.
7. Modify the #define for MQTT_BROKER_ADDRESS. The address can be found by clicking on "Settings" at the lower left corner of the main Amazon AWS console window. The broker address is listed as the "Endpoint".
8. Modify the #define for WICED_TOPIC. Use the topic from exercise 2 with your initials in the name.
9. Modify the #define for CLIENT_ID to include your initials. This is necessary to prevent conflicts since everyone is using the same class broker.
10. Build and program your project.
11. Open the serial port and watch your terminal session.
12. Subscribe to your topic using the AWS MQTT Test Client. When you press the button, you should see updates to the topic in the test window.

Exercise - 7D.2 WICED MQTT Firmware Flow

Explain in detail the firmware flow for the publisher app by answering the following questions:

1. How do the MQTT library functions (e.g. *wiced_mqtt_publish()*) get into your project?

2. What function is called when the button is pressed?

3. How does the button callback unlock the main thread?

4. What WICED SDK RTOS mechanism does the “wait_for_response” function use to “wait”?

5. Why did the firmware author create a function called “wait_for_response”?

6. Are all messages sent to the AWS IOT MQTT Message Broker required to be in JSON format?

7. What are the 7 WICED MQTT events? What file are they defined in?

8. Do you have to name the client certificate client.cer? How would you change the name?

9. What is the naming convention used to differentiate WICED MQTT library functions versus wrappers around those functions in the publisher app?

10. What steps are required to get an MQTT connection established?

11. What prevents a hung connection from deadlocking the publisher app?

12. What is the name of the flag that prevents the firmware from sending multiple button presses before the publish is finished?

Exercise - 7D.3 Build and Test the demo.aws_iot.pub_sub.subscriber App

1. Copy the WICED application from apps/demo/aws_iot/pub_sub/subscriber to your directory (i.e. wa101/07b/05_subscriber) and modify the DCT and makefile.
2. Update the topic and broker #defines to the same ones you used for exercise 4.
3. Update the #define for CLIENT_ID to contain your initials.
4. Search for and update the region if necessary.
5. We will use the same *thing*, certificate, and keys that we did for exercise 2.
6. Publish messages using the AWS Test MQTT Client.
    a. Determine what string needs to be sent to turn the light on or off.
        i. Hint: Look in the source code to find the string that is being looked for when a message is received.
        ii. Hint: If you are successful, the LED on the shield should turn on/off.

Exercise - 7D.4 (Advanced) Implement the subscriber and publisher in two different kits and test

1. In a real-world application, you would typically have one or more devices publishing data to a broker and one or more devices reading data from that same broker. So, let's try that out with two different kits. You should team up with another student for this lab.
    a. Make a new *thing* in the AWS console for the subscriber and create a new certificate for it. You can attach the same policy that you created previously.
        i. Note: You could actually use the same *thing*, certificate, and policy for both the subscriber and publisher if you wanted, but in many cases, you will want each type of *thing* to have different permissions or settings. For example, you might want the subscriber to be able to read values but not modify them. In that case, the certificate and the attached policy for the subscriber *thing* would be different.
        ii. Hint: Make sure the CLIENT_ID is different between the two projects. Otherwise they will interfere with one another.
    b. Save the new subscriber certificate files but use different names for *client.cer* and *privkey.cer* so that the subscriber and publisher can use different files.
    c. Update the makefile so that the subscriber points to the new certificates.
        i. Hint: The rootca.cer will not need to change since it is the Amazon AWS public key which is always the same.
    d. Update the lines in "subscriber.c" that point to the new credential and key files.
        i. Hint: the credentials are listed as *resources_apps_DIR_aws_iot_DIR_client_cer* and *resources_apps_DIR_aws_iot_DIR_privkey_cer*. These names are the path in the resources folder where folder names are separated by the keyword "_DIR_" and the period before cer is replaced with "_". You could move the credentials to another location in the resources folder by following the naming convention or just change the names of the files and put them in the same folder.
    e. Program the updated subscriber firmware.
    f. Power up both kits.

g. Subscribe to the topic that you chose using the AWS test MQTT Client.
h. Press the button on the publisher and watch it change the state of the LED on the subscriber. Also watch the messages in the AWS test MQTT Client window.

Exercise - 7D.5 (Advanced) Build and test the Shadow App

1. This example uses a configuration Access Point serving a web browser so that the *thing* name, credentials, keys, and settings for the network to connect to can be configured from a web browser on a device attached to the configuration AP.
2. Copy the WICED application from *apps/demo/aws_iot/shadow* to your directory and update the makefile.
   Update the DCT to have a Config AP for configuration with an SSID name that is unique (so as not to collide with others in your class).
   Search for and update the region if necessary.
3. Update the message broker address to match what you created in the previous exercises.
   a. Hint: The message broker address goes in "*aws_common.h"* in the #define for AWS_IOT_HOST_NAME.
4. Program the kit.
5. Attach to the Config AP on your board from your computer's Wi-Fi.
   a. Connect to the SSID that you programmed into the board.
   b. Go to the webserver (The IP address is printed on the terminal when the device boots and starts the AP).
      i. Hint: Don't use Firefox for this step – it sometimes gives strange results.
   c. Update the *thing* name to match what you created in previous exercises.
      i. Hint: The default name that shows up is in "*aws_config.h"* so you could also change it there before programming the board.
   d. Follow the instructions to upload the client certificate and private key.
   e. Click on "Wi-Fi Setup >", click on the class Wi-Fi network, enter the password, and click connect.
   f. The board will reboot. Once it has done that, it will connect as a station to the Wi-Fi network that you configured in the previous step.
6. Attach to a Wi-Fi access point from your computer.
7. Go to console.aws.amazon.com and go to the test MQTT Client.
8. Subscribe to the device's shadow topics.
   a. Hint: *$aws/things/<YourThingName>/shadow/#* will subscribe to all shadow topics for your *thing*. The # is a wildcard.
9. Press the button on the board and see the messages.
10. Answer the question: What is the sequence of events that changes the LED from On to Off?

7D.3 Related Example “Apps”

App Name	Function
demo.aws_iot_pub_sub/publisher	Demonstrates publishing information to the AWS cloud.
demo.aws_iot_pub_sub/subscriber	Demonstrates subscribing to a topic in the AWS cloud.
demo.aws_iot_shaddow	Demonstrates using a shadow device with AWS.