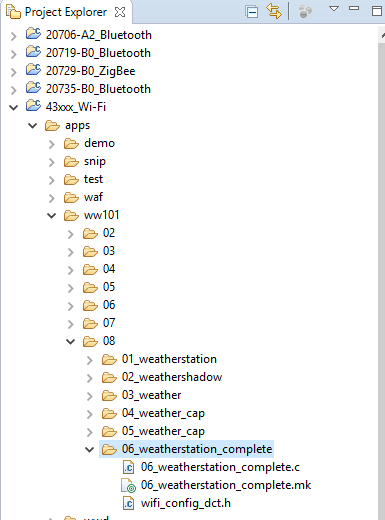
**HOW TO TEST AND USE THE AWS WEATHERSTATION PROJECT WITH AWS PUBLISHER AND SUBSCRIPTION FUNCTIONALITY**

Adding project to WICED and creating MAKE file

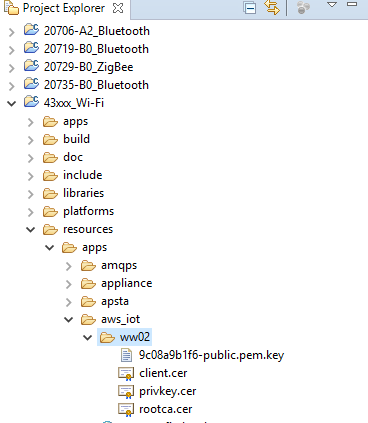
1. Unzip the package. There are 2 sets of files. The first is the application files, they need copying to a directory that is named

43xxx\_WiFi- -> apps -> ww101 -> 08



1. If you use a different directory you will need to update the make file appropriately.
2. The second set of files are the certificates and keys used for communicating to the AWS cloud. If you generate your own account and “Thing” you will need to replace these with your own keys.
3. The files need copying to the following directory

43xxx\_WiFi- -> resources -> apps -> aws\_iot -> ww02



1. Create a make target with the following name

ww101.08.06\_weatherstation\_complete-BCM943907AEVAL1F\_WW101 download run

1. Change the name at the start to match your name if different
2. If you know the name of the SSID and password to connect to you can change this in the wifi\_config\_dct.h file
3. Save all changes, connect the board, and then run the make target.
4. Open a HyperTerminal/Tera-term window, connect to the correct COM port, and set the serial settings to be 115200 baud.
5. Hit reset on the board.
6. It should then either connect to the SSID specified, or prompt you to enter a SSID. If prompted enter the ID followed by Enter/CR. Then do the same for the password. If you enter incorrectly press escape to restart entry.
7. You will then be prompted to enter a name. this is used as an ID as well as the Client ID for AWS. This needs to be unique for every node so be sure to use a different one for each node.
8. Once entered it should then open a publish topic and a subscribe topic.

Operation

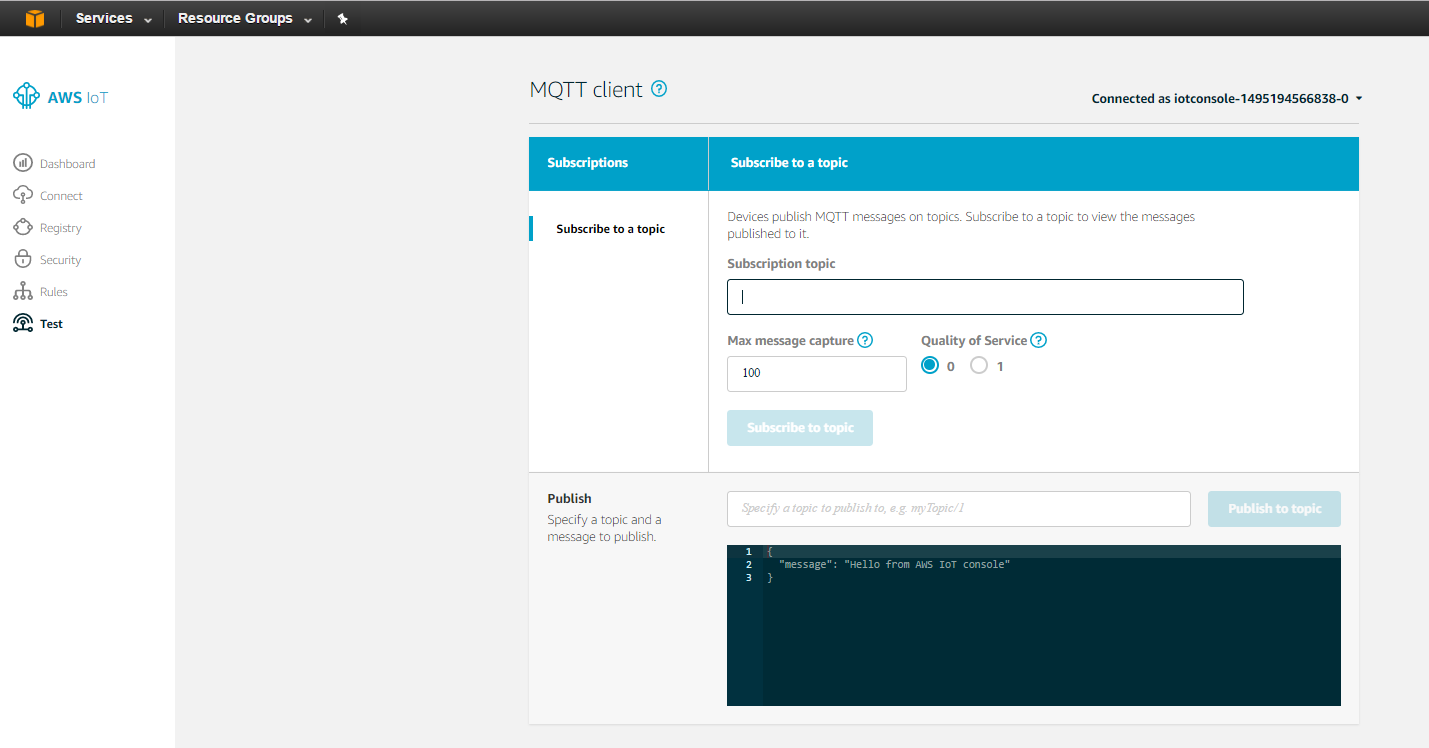
1. By default the project is designed to publish and subscribe to the same topic. This is the shadow topic created in the class. If you have multiple nodes, data from one should be replicated on the other.
2. When the use presses Cap button 0 or 1, it should publish to the topic currently defined. This will update the display with 2 different displays depending on which button is pressed. If the subscribe topic is the same as the publish topic, this will get overwritten with the data received from the cloud.
3. Pressing MB0 will cause the publish topic to change. There are currently 8 topics defined, shadow, topic 1 through 7. Each press of the button increments the topic id by 1. When logged in to the AWS console, and subscribe to topic # you should see the ID’s at the top of each message.
4. Pressing MB1 will cause the subscribe topic to change. This requires the device to unsubscribe first and the subscribe. If there is an error in this process it may be required to press again until you get all the way through successfully.
5. If the local node is publishing to a different topic to that which it is subscribing to, on Cap button 0 or 1 presses, it will only display local information.
6. The terminal is used throughout to output logs as to what is going on so if the operation fails or hangs at any point this can be used to debug what has happened.

Code

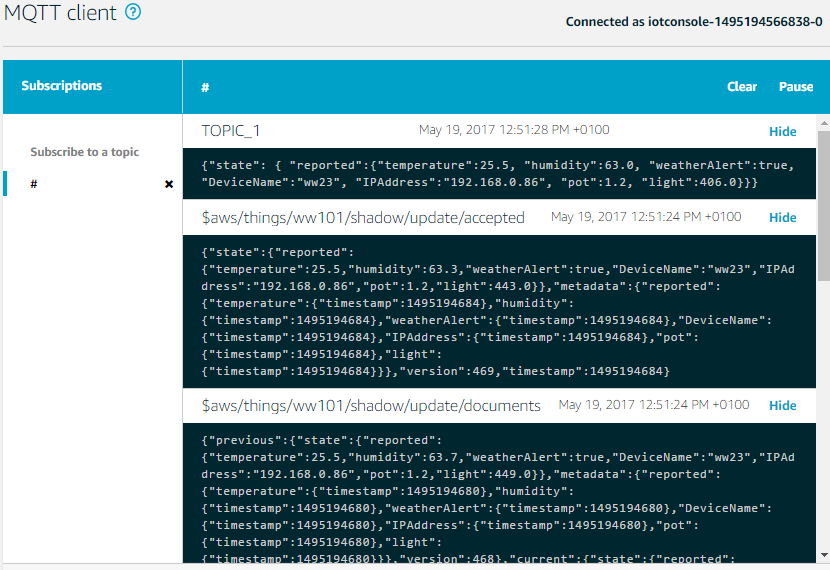
1. The project itself reads the i2c buttons from the co processor every 100ms. If it detects a press it does something. If C0 is pressed it publishes to the current topic and updates the display with 1 set of information by setting the semaphore for a thread
2. If C1 is pressed it again publishes but sets a different semaphore for updating the display with extra information
3. On both presses a semaphore is set to publish data. This thread sends the message to the cloud and then waits to make sure it has been sent successfully. If it gets a failed call back, we try to close then reopen the connection.
4. In the call back, if we get a subscribe message event we strip the data out that is received, pass it through the json parser, and display the received data on the display.
5. The JSON Parser function looks for a string with a name (e.g. temperature). If it finds it, it then pulls the value out that follows and stores it in an array we have declared.
6. There is another thread that is running to read the sensor data from the co-processor, that runs whenever it’s semaphore is set.
7. For all the i2c accesses, there is a mutex around them to prevent multiple attempts to use the same resource.
8. When starting up the application users can enter the SSID and password they want to connect to over UART. To force this, leave the DCT blank before programming.

Testing

1. To test, log in to your AWS account
2. Go to AWS IOT then to Test



1. In the subscribe section type “#” minus the quotation marks.
2. This will subscribe you to all topics. Every time you press C0 or C1 you should see new data here, with the topic. This image shows shadow and topic examples:



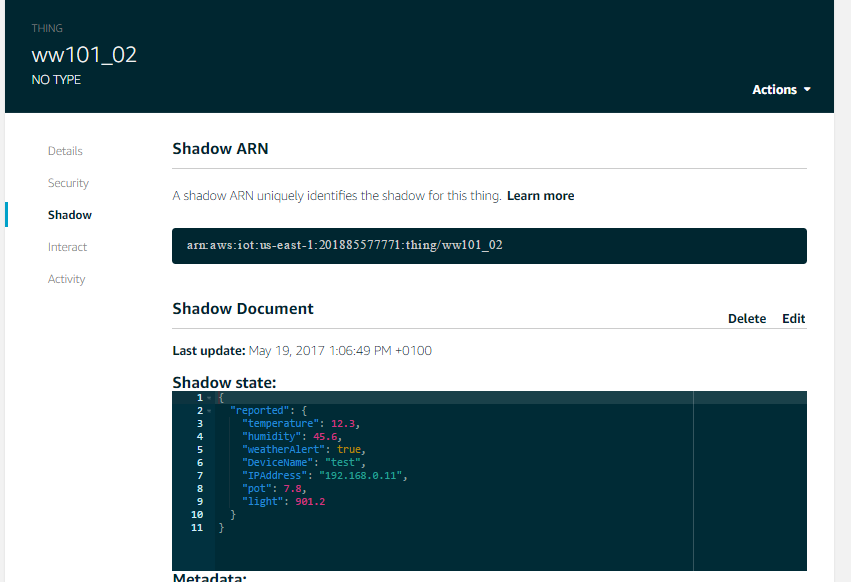
1. To test publishing, change the topic to TOPIC\_1 by pressing MB1, type TOPIC\_1 in the Publish space, and enter the following code in the message window

{"state": { "reported":{"temperature":12.3, "humidity":45.6, "weatherAlert":true, "DeviceName":"test", "IPAddress":"192.168.0.11", "pot":7.8, "light":901.2}}}

1. You should see the data updated on the screen on the node. You can update individual elements are the entire array. To change just the name send this:

{"DeviceName":"grsm"}

1. This should result in just the name changing.
2. If you publish to the shadow topic, and you have the topic defined correctly it should store the data to the shadow. To check, go to Registry, Things, then Shadow. It should show the same information as you published



1. If you want to modify the project to use your own shadow and your own “Thing” you will need to change lines 135, 136 and 144 in weatherstation\_complete.c. for example if your “thing” was called TEST\_123, the line should look like this - "$aws/things/TEST\_123/shadow/update"